SIAM J. COMPUT.
Vol. 14, No. 4, November 1985

ON APPROXIMATION ALGORITHMS FOR #P*

LARRY STOCKMEYER†

Abstract. The theme of this paper is to investigate to what extent approximation, possibly together with randomization, can reduce the complexity of problems in Valiant's class #P. In general, any function in #P can be approximated to within any constant factor by a function in the class Δ_3^p of the polynomial-time hierarchy. Relative to a particular oracle, Δ_3^p cannot be replaced by Δ_2^p in this result. Another part of the paper introduces a model of random sampling where the size of a set X is estimated by checking, for various "sample sets" S, whether or not S intersects X. For various classes of sample sets, upper and lower bounds on the number of samples required to estimate the size of X are discussed. This type of sampling is motivated by particular problems in #P such as computing the size of a backtrack search tree. In the case of backtrack search trees, a sample amounts to checking whether a certain path exists in the tree. One of the lower bounds suggests that such tests alone are not sufficient to give a polynomial-time approximation algorithm for this problem, even if the algorithm can randomize.

Key words. #P, approximation algorithms, probabilistic algorithms, computational complexity, relativization

1. Introduction. There are several computational problems which can be formulated as problems of counting the number of objects having a certain property. Valiant [17] has introduced the class #P which includes a variety of counting problems such as counting the number of perfect matchings in a graph, computing the permanent of a matrix [17], finding the size of a backtrack search tree [8], and computing the probability that a network remains connected when links can fail with a certain probability [18]. For many problems in #P, including those just mentioned, no polynomial-time algorithms are known. Indeed, it is not known whether these problems can be solved at any fixed level of the polynomial-time hierarchy [1], [15]. The obvious algorithm of explicitly counting the number of objects is not efficient since the number of objects grows exponentially in the size of the input. For example, in computing the permanent of an $n \times n$ matrix with 0–1 entries a_{ij}, the objects are the $n!$ permutations on $\{1, \cdots, n\}$, and the permanent is equal to the number of permutations π with $\prod_{i=1}^{n} a_{i,\pi(i)} = 1$. The best known general upper bound is that any #P problem can be solved within polynomial space.

Two known approaches for reducing the computational complexity of problems are approximation and randomization. Several NP-complete optimization problems, which are apparently intractable to solve exactly, can be solved in polynomial time if one is willing to settle for a solution within a constant factor of the optimum [3, Chapter 6]. Randomization, that is, allowing algorithms to make decisions based on the outcomes of random coin tosses but requiring that the probability of error be small, has been useful in solving certain number theoretic problems faster than the best known deterministic algorithms [12], [14]. The purpose of this paper is to investigate to what extent approximation, or approximation together with randomization, can reduce the complexity of counting problems.

Before proceeding to outline the remainder of the paper, we should explain informally our definitions of "approximation" and "randomization." If $f(x)$ is an

* Received by the editors October 25, 1983, and in revised form July 5, 1984. Portions of this paper have been reprinted, with permission, from *The complexity of approximate counting* by L. Stockmeyer, appearing in the Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 118–126, © 1983, Association for Computing Machinery, Inc.

† Computer Science Department, IBM Research Laboratory K51-281, 5600 Cottle Road, San Jose, California 95193.

integer valued function and r is a constant with $r > 1$, then a probabilistic algorithm approximates $f(x)$ to within the factor r if

(1) The value produced by the algorithm is between $f(x)/r$ and $r \cdot f(x)$ with probability $\geqq \frac{1}{2} + \varepsilon$ for some fixed $\varepsilon > 0$.

(In the case of #P problems, each input x determines a set X_x such that membership in X_x can be checked in polynomial time, and $f(x) = |X_x|$.)

Regarding previous work, Knuth [8] gives a polynomial-time probabilistic algorithm for estimating the size of a backtrack search tree where the expected value of the output of the algorithm is exactly the size of the tree. Lovasz does the same for computing the permanent of a 0-1 matrix. However, for both of these algorithms the variance is very large and the algorithms do not satisfy (1). By analyzing the variance of Lovasz's algorithm and related algorithms, Karmarkar, Karp, Lipton and Luby [10] give probabilistic algorithms for the 0-1 permanent which satisfy (1) and whose running times, although not polynomial in n, are better than the best known deterministic algorithm for computing the permanent exactly.

One well-known approach for estimating the size of a set X is random sampling. Assuming that X is a subset of a finite universe U of known size, one would randomly choose elements of U, compute the fraction γ of chosen elements which belong to X, and give $\gamma|U|$ as an estimate for the size of X. Goldschlager [5] suggests this as an approximation method for computing the permanent. Since $|U|$ is typically exponential in n, this "singleton sampling" runs in time polynomial in n and satisfies (1) only if $|X| \geqq |U|/n^k$ for some constant k. For example, if $|X|$ is small, say $|X| \approx |U|^{1/2}$, then singleton sampling would require more than $|U|^{1/2}$ samples to reach a good estimate.

In § 2 we define and study a class of restricted, but very natural, probabilistic sampling methods motivated by the particular counting problems mentioned above. Instead of "singleton sampling" the algorithm is allowed to sample a large set $S \subseteq U$ in one step; the information returned from the sample is whether $S \cap X = \varnothing$. Depending on the application, there might be restrictions on the types of subsets S which can be sampled (some such restrictions are detailed in § 2), so we are interested in how the complexity, measured as the number of samples, depends on the types of samples which are allowed. Two motivating examples are given in § 2. The main technical results of § 2 establish, for two particular classes of sample sets, lower bounds on the number of samples required to approximate $|X|$. One of these lower bounds suggests that this type of sampling will not give a polynomial-time algorithm for approximating the size of a backtrack search tree.

In § 3 we attempt to classify the complexity of approximately computing functions in #P. The classification is done in terms of the polynomial-time hierarchy (for short, P-hierarchy) [15]. For any function in #P, it is shown that a machine at the Δ_3^p level of the P-hierarchy can compute approximate solutions that are accurate to within the factor $(1 + \varepsilon n^{-d})$ for any fixed constants d and $\varepsilon > 0$. (As noted above, computing #P functions exactly is not known to be possible at any finite level of the P-hierarchy.) The proof is based on the technique, introduced recently by Sipser [13], of estimating the size of a set by hashing the set into a second set of known size. In the relativized world, cf. [2], we also can give a corresponding lower bound to the general problem of approximately computing #P functions: there is an oracle A and a function in $\#\mathrm{P}^A$ ($\#\mathrm{P}$ relativized to A) such that the function cannot be approximately computed to within any constant factor by a machine at the $\Delta_2^{p,A}$ level of the A-relativized P-hierarchy. The Δ_3^p upper bound relativizes to an arbitrary oracle. (See [2], [15] for definitions of oracle machines and the relativized polynomial-time hierarchy.)

In § 4 we give a relativization result that complements a recent result of Sipser and Gács [13] that BPP is contained in the second level of the P-hierarchy. Recall that BPP is the class, as defined by Gill [4], of languages accepted by probabilistic polynomial-time Turing machines with error probability $\leqq \frac{1}{2} - \varepsilon$ for some fixed $\varepsilon > 0$. Sipser and Gács show that

$$\mathrm{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$$

and this result relativizes to an arbitrary oracle. Given this inclusion, it is natural to ask whether BPP can be placed even lower in the P-hierarchy. The next lowest class below $\Sigma_2^p \cap \Pi_2^p$ is Δ_2^p, the class of languages accepted by deterministic polynomial-time Turing machines using oracles in NP. We show that there is an oracle A such that

$$\mathrm{BPP}^A \not\subset \Delta_2^{p,A}.$$

Thus, any attempt to prove $\mathrm{BPP} \subseteq \Delta_2^p$ cannot proceed by a proof that relativizes to an arbitrary oracle. The connection between this result and the rest of the paper is that there is an oracle such that a BPP machine can approximately count the number of strings of a given length that are in the oracle whereas Δ_2^p machines cannot.

2. Intersection-samples.

2.1. Definitions and motivation. If U is a set, 2^U denotes the set of subsets of U, and $|U|$ denotes the cardinality of U. For integer $n \geqq 0$, $\{0,1\}^n$ denotes the set of binary words of length n and $\{0,1\}^{\leqq n}$ denotes the set of binary words of length n or less. If x is a word, $|x|$ denotes the length of x.

It will be convenient first to define a framework which captures several counting problems. In this framework, a problem is specified by a finite set (*universe*) U and a collection of *inputs* $\mathscr{X} \subseteq 2^U$. Given an arbitrary input $X \in \mathscr{X}$, we want to approximate $|X|$ to within a constant factor. Let $N = |U|$. For the situations which motivate this question, N is so large that the count cannot be done explicitly. For example, $N = 2^n$ where n is the natural measure of the "size" of the problem. We are also given a class of sets, the *samples*,

$$\mathscr{C} \subseteq 2^U.$$

For an arbitrary $S \in \mathscr{C}$, a unit cost operation is computing the predicate

$$\mathrm{INT}(X, S) = \begin{cases} 1 & \text{if } S \cap X \neq \varnothing, \\ 0 & \text{otherwise.} \end{cases}$$

For what types of sample classes $\mathscr{C}$ can a probabilistic algorithm with running time polynomial in n ($=\log N$) compute an estimate to $|X|$ accurate to within a constant factor? Before giving some answers, we should first give some motivation. In particular, we must justify why $\mathrm{INT}(X, S)$ can be viewed as a unit cost operation even when S is a large set. (Since we are only interested in distinguishing polynomial from nonpolynomial running times, any operation which can be done within time polynomial in n is viewed as a "unit cost" operation.)

1. *Estimating the size of a backtrack search tree.* Knuth [8] has considered the problem of estimating the size of a tree X where X is described implicitly by a polynomial-time backtrack search procedure. That is, given the name u of a node of X, the search procedure tells us which sons of u, if any, are in X. Let n be a known upper bound on the height of X, and to simplify the notation assume that X is binary. Imagine that X is embedded in F, the full binary tree of height n. The nodes of F are labeled by strings in $\{0,1\}^{\leqq n}$: the root is labeled λ, and the sons of u are labeled $u0$

and $u1$. Thus, the universe is $U_{\text{tree}} = \{0, 1\}^{\leq n}$. In this example, since the input $X \subseteq U$ is a tree with the same root as F, X must satisfy the following *tree property*:

$$\text{if } u \in X \text{ then } w \in X \text{ for all prefixes } w \text{ of } u.$$

Let $\mathscr{X}_{\text{tree}}$ be the class of sets X satisfying the tree property. The *tree size problem* is

$$\mathscr{P}_{\text{tree}} = (U_{\text{tree}}, \mathscr{X}_{\text{tree}}).$$

(We have actually defined a family of problems for $n \geqq 1$. For simplicity in this example and the next, the index n has been suppressed.)

For w an arbitrary node of F, we can determine in time polynomial in n whether the full subtree of F rooted at w contains a node of X simply by attempting to follow the search procedure from the root of F to w. Thus the relevant class of samples is the class of *subtree samples*:

$$\mathscr{C}_{\text{subtree}} = \{S_w | w \in \{0, 1\}^{\leq n}\}$$

where

$$S_w = \{u \in \{0, 1\}^{\leq n} | w \text{ is a prefix of } u\}$$

is the full subtree of F rooted at w. If we view the backtrack search procedure as a "black box," i.e., if the only way we can access the procedure is by giving it inputs and looking at its outputs, then subtree samples are, intuitively, the most general way of accessing the tree X.

2. *Counting perfect matchings.* For some #P-complete problems such as counting the number of perfect matchings in a graph (equivalently, for bipartite graphs, computing the permanent of a 0-1 matrix), the corresponding existential question, does there exist a perfect matching, can be solved in polynomial time [9]. Can this fact be used to estimate the number of perfect matchings in polynomial time? We do not have a definitive answer to this question, but we have some preliminary evidence on the pessimistic side. The problem fits the general framework as follows. Suppose we want to solve this problem for graphs with n vertices. Let $m = n(n-1)/2$ and let $E = \{e_1, \cdots, e_m\}$ be the edges of K_n, the complete graph on n vertices. In this case the universe is $U_{\text{word}} = \{0, 1\}^m$. A particular graph G (viewed as a subgraph of K_n) defines a subset X_G of U_{word} as follows:

$$\text{if } u = u_1 u_2 \cdots u_m \in \{0, 1\}^m, \text{ then}$$

$$u \in X_G \text{ iff } \{e_j | u_j = 1\} \text{ is a perfect matching in } G.$$

The ability to check the existence of perfect matchings in subgraphs of G allows us to compute a variety of intersection samples as follows. Given $E_{\text{in}}, E_{\text{out}} \subseteq E$ with $E_{\text{in}} \cap E_{\text{out}} = \varnothing$, in polynomial time we can compute the predicate $P(G, E_{\text{in}}, E_{\text{out}})$ defined to be *true* iff G has a perfect matching $M \subseteq E$ with $E_{\text{in}} \subseteq M$ and $E_{\text{out}} \cap M = \varnothing$. Specifically, if E_{in} contains an edge not in G or if E_{in} contains two edges with a common endpoint then $P(G, E_{\text{in}}, E_{\text{out}})$ is false. Otherwise, remove from G all edges in E_{in} and E_{out} and all edges incident on an endpoint of an edge in E_{in}. Then $P(G, E_{\text{in}}, E_{\text{out}})$ iff the resulting subgraph has a perfect matching. Note that $P(G, E_{\text{in}}, E_{\text{out}})$ is really an intersection sample in disguise. Specifically, if we define $w = w_1 \cdots w_m$ by setting $w_j = 1$ for all $e_j \in E_{\text{in}}$, $w_j = 0$ for all $e_j \in E_{\text{out}}$, and $w_j = *$ otherwise, then

$$P(G, E_{\text{in}}, E_{\text{out}}) = \text{INT}(X_G, Q_w)$$

where

$$Q_w = \{u \in \{0,1\}^m | u \text{ matches } w\}.$$

Here, * is a "don't care" symbol that matches 0 and 1. Thus the relevant class of samples, the *partial match samples*, is

$$\mathscr{C}_{pm} = \{Q_w | w \in \{0,1,*\}^m\}.$$

Let $\mathscr{X}_{\text{all}} = \{X | X \subseteq U_{\text{word}}\}$. The *word counting problem* is

$$\mathscr{P}_{\text{word}} = (U_{\text{word}}, \mathscr{X}_{\text{all}}).$$

An obvious weakness in this definition is the choice of the class $\mathscr{X}$ of inputs as all subsets of U_{word} rather than all X of the form X_G for some graph G. The reason for this choice is discussed below.

Another #P-complete problem with a polynomial-time existential question is: given a graph G and vertices s and t, count the number of subgraphs H of G for which there is a path from s to t in H [18] (this problem arises in network reliability). In this case, a partial match sample asks whether there is a path from s to t in a certain subgraph of G.

Our formal model of computation in this section is a variation of the probabilistic decision tree studied previously by Manber and Tompa [11]. Fix some counting problem $\mathscr{P}$ consisting of a universe U and a class of inputs $\mathscr{X}$, and fix a class of samples $\mathscr{C}$. Each internal node of the decision tree is either a *sample node* or a *randomizing node*. A sample node is labeled by a set $S \in \mathscr{C}$ and has two branches that are taken depending on the outcome of $\text{INT}(X, S)$ where X is the particular input. A randomizing node has any number d of branches; each branch is taken with probability $1/d$. Each leaf is labeled by an integer between 0 and N, where $N = |U|$. A decision tree *solves $\mathscr{P}$ to within the factor r with error probability δ* if, for any input $X \in \mathscr{X}$, the tree reaches a leaf with label between $|X|/r$ and $r|X|$ with probability $\geq 1 - \delta$. The *sample height* of the tree is the maximum number of sample nodes along any path from the root to a leaf. As Manber and Tompa point out, a lower bound on sample height implies, to within a constant factor, a lower bound on the expected number of samples (expectation with respect to the choices made at randomizing nodes), since branches which are much longer than the expected value can be pruned while increasing the error probability only slightly.

DEFINITION. $H(N, \mathscr{P}, \mathscr{C}, r, \delta)$ is the minimum sample height of a probabilistic decision tree, using only samples from $\mathscr{C}$, which solves $\mathscr{P}$ to within the factor r with error probability δ, where N is the size of the universe of $\mathscr{P}$.

Throughout the rest of § 2, r and δ are fixed constants with $r > 1$ and $0 < \delta < \frac{1}{2}$.

2.2. Statements of results and discussion. In the case of subtree samples, the following lower and upper bounds are proved:

$$H(N, \mathscr{P}_{\text{tree}}, \mathscr{C}_{\text{subtree}}, r, \delta) = \Omega(N^{1/2}),$$

$$H(N, \mathscr{P}_{\text{tree}}, \mathscr{C}_{\text{subtree}}, r, \delta) = O((N \log N)^{1/2}).$$

(Here and subsequently, the constants implicit in the O- and Ω-notations depend on r and δ.) Knuth [8] has reported success in practice with a simple $O(\log N)$ time probabilistic procedure for estimating the size of a backtrack search tree. The $\Omega(N^{1/2})$ lower bound is not meant to contradict this, but it does suggest that the success must be linked to the properties of trees generated by particular backtrack search procedures. It should also be noted that the lower bound $\Omega(N^{1/2})$ is proved when the input X is

further restricted to a subset $\mathscr{X}'$ of $\mathscr{X}_{\text{tree}}$. For $1 \leqq d \leqq n$, let $X(d, 0)$ be the tree whose nodes are $\{0, 1\}^{\leqq d}$. For $1 \leqq k \leqq 2^d$, let $X(d, k)$ be the tree $X(d, 0)$ together with the full subtree of height $n-d$ attached to the kth leaf of $X(d, 0)$. Then $\mathscr{X}'$ is the set of trees $X(d, k)$ for $1 \leqq d \leqq n$ and $0 \leqq k \leqq 2^d$. It is relevant that the input can be restricted to $\mathscr{X}'$ because the trees in $\mathscr{X}'$ can actually be generated by backtrack search procedures of size $O(n)$, whereas just naming trees in $\mathscr{X}_{\text{tree}}$ requires names of length 2^n.

In the case of partial match samples we show that

$$H(N, \mathscr{P}_{\text{word}}, \mathscr{C}_{\text{pm}}, r, \delta) = \Omega(N^{1/5}).$$

Since N is exponential in the size of the graph, the $\Omega(N^{1/5})$ lower bound suggests that this type of sampling does not give an efficient way of estimating the number of perfect matchings. However, there are two loopholes in this interpretation. First, as noted above, we do not restrict the input X to specify the set of perfect matchings of some graph. Unfortunately, with this restriction an interesting lower bound cannot be proved for the (nonuniform) decision tree model. To see this, let $w_i \in \{0, 1, *\}^m$ have 1 in the ith position and *'s elsewhere. If $\text{INT}(X_G, Q_{w_i}) = 1$ then e_i is an edge of the graph G. If $\text{INT}(X_G, Q_{w_i}) = 0$ then we can assume that e_i is not an edge of the graph since it is not used in any perfect matching. Thus, by making m samples, the decision tree can determine the graph. Since the graph determines the number of its perfect matchings, a nonuniform decision tree, using the samples w_i, can find the number of perfect matchings within time polynomial in the size of the graph. The second loophole is that it is conceivable that a different class of samples could lead to an efficient estimation procedure. There is room for more work on this problem.

Remark. It is natural to ask whether the structure of the subtree and partial match samples is being used to prove these lower bounds. Do lower bounds of the form N^ε hold for arbitrary classes of samples? It is not difficult to see that a much smaller sample height suffices if any subset can be a sample. Letting

$$\mathscr{P}_{\text{all}} = (U, \mathscr{X}_{\text{all}}) \quad \text{and} \quad \mathscr{X}_{\text{all}} = \mathscr{C}_{\text{all}} = 2^U,$$

it can be shown that

$$H(N, \mathscr{P}_{\text{all}}, \mathscr{C}_{\text{all}}, r, \delta) = O(\text{loglog } N \, \text{logloglog } N),$$
$$H(N, \mathscr{P}_{\text{all}}, \mathscr{C}_{\text{all}}, r, \delta) = \Omega(\text{loglog } N).$$

The simple proofs are sketched in [16]. Of course, this upper bound is useless in cases where N is exponentially large since time N is needed just to *name* a member of $\mathscr{C}_{\text{all}}$. It is also noted in [16] that without approximation (i.e. $r = 1$) or without randomization (i.e. $\delta = 0$), the obvious upper bound of N samples cannot be significantly improved, even when any subset of U can be a sample:

$$H(N, \mathscr{P}_{\text{all}}, \mathscr{C}_{\text{all}}, 1, \delta) = \Omega(N),$$
$$H(N, \mathscr{P}_{\text{all}}, \mathscr{C}_{\text{all}}, r, 0) = \Omega(N).$$

2.3. Proofs. For the lower bound proofs it is useful to assume that any probabilistic decision tree has the property that there is a constant p such that for any input X and any leaf l, the probability that the leaf l is reached is either p or 0. Any decision tree T not having this property can be modified to have the property without changing its sample height. First, by adding new randomizing nodes with outdegree 1, modify T so that the same number, say c, of randomizing nodes appear on every root-to-leaf path. Second, modify T so that all randomizing nodes have the same outdegree d

(where d can be taken as the least common multiple of the outdegrees of all randomizing nodes in the original T). Then $p = d^{-c}$. Details are left to the reader.

The proofs of the lower bounds for subtree and partial match samples are similar. In both cases we assume that the problem can be solved by a probabilistic decision tree T with sample height smaller than the desired lower bound. We find inputs D and $D \cup V$, with $r|D| < |D \cup V|/r$, such that a substantial fraction of the leaves with label between $|D|/r$ and $r|D|$ which are reached by T on input D are also reached by T on input $D \cup V$. From this we infer a contradiction, since an answer $\leqq r|D|$ is incorrect for input $D \cup V$. There are two main steps.

I. Find a set $D \subseteq U$ with $|D| \approx N^{1/2}$ such that if $S \in \mathscr{C}$ and S is "large" (roughly $|S| \geqq N^{1/2}$) then $D \cap S \neq \varnothing$. (Informally, if we restrict attention to inputs X with $D \subseteq X$, then all "large" samples supply no information about $X - D$ since the answer to any such sample is always that $X \cap S \neq \varnothing$.)

II. Find a collection of sets $V_1, \cdots, V_t \subseteq U$ such that

(A) $|V_j| \geqq (r^2+1)|D|$ for all j, and

(B) if $S_1, S_2, \cdots, S_g \in \mathscr{C}$, each S_k is "small" (i.e., $S_k \cap D = \varnothing$ for $1 \leqq k \leqq g$) and $g \leqq h$ where h is the lower bound on sample height to be proved, then the union of S_k for $1 \leqq k \leqq g$ intersects at most the fraction $(\frac{1}{2} - \delta)$ of the V_j's.

Given I and II, the lower bound h on sample height is proved as follows. Let T be a decision tree with sample height h that approximates $|X|$ to within the factor r with error probability δ. If τ is a root-to-leaf path and $J \subseteq U$, say that τ is *valid* for J if the answers given at sample nodes along τ are consistent with the input being J. Let $\tau_1, \cdots, \tau_s$ be the root-to-leaf paths in T that are valid for D and such that the leaf label is between $|D|/r$ and $r|D|$. By letting the input be D, we must have $sp \geqq 1 - \delta$, where p is the probability defined in the first paragraph of § 2.3. Consider the $s \times t$ 0–1 array $C = \{c_{ij}\}$ defined as follows. For $1 \leqq i \leqq s$ and $1 \leqq j \leqq t$, if $S_1, S_2, \cdots, S_g$ are the sample nodes on path τ_i that do not intersect D and if the union of the S_k for $1 \leqq k \leqq g$ intersects V_j then $c_{ij} = 1$; otherwise, $c_{ij} = 0$. By II(B), the density of 1's along any row of C is at most $(\frac{1}{2} - \delta)$. Therefore, there must be a column z such that the density of 1's in column z is at most $(\frac{1}{2} - \delta)$, so the density of 0's is at least $(\frac{1}{2} + \delta)$. But $c_{iz} = 0$ means that the path τ_i is valid for $D \cup V_z$. This is true because, by (I), any large sample on the path τ_i gives the same answer, "$\neq \varnothing$," for any input containing D, and by definition of c_{iz} no small sample on τ_i intersects V_z. By II(A), the answer at the leaf of any path τ_i is $\leqq r|D| < |D \cup V_z|/r$. Therefore, when given the input $D \cup V_z$, the tree gives a too small answer with probability at least

$$(\tfrac{1}{2} + \delta)sp \geqq (\tfrac{1}{2} + \delta)(1 - \delta) \geqq \tfrac{1}{2} > \delta.$$

THEOREM 2.1. $H(N, \mathscr{P}_{\text{tree}}, \mathscr{C}_{\text{subtree}}, r, \delta) = \Omega(N^{1/2})$.

Proof. Recall from § 2.1 that F is the full binary tree of height n. The *depth* of a node of F is its distance to the root. $N = 2^{n+1} - 1$ is the number of nodes of F. Let

$$d = \lfloor (n - 1 - \log(r^2+1))/2 \rfloor,$$

and let D be the set of (labels of) nodes of F with depth $\leqq d$. Let $u_1, \cdots, u_t$ be the nodes of depth d. For $1 \leqq j \leqq t$, let V_j be the set of (labels of) nodes in the full subtree of F rooted at u_j. A simple calculation shows that II(A) holds by choice of d, and that $t = \Omega(N^{1/2})$. From the tree structure it can be seen that if $S \in \mathscr{C}_{\text{subtree}}$ and S is "small," i.e., if $S \cap D = \varnothing$, then there is exactly one j such that $S \cap V_j \neq \varnothing$. (Specifically, if w is the root of the subtree S, then $S \cap D = \varnothing$ implies that the depth of w is greater than d, so exactly one of $u_1, \cdots, u_t$ is an ancestor of w.) Thus, a union of h subtree samples can intersect at most the fraction h/t of the V_j's. Taking $h = (\frac{1}{2} - \delta)t$ gives the result. □

THEOREM 2.2. $H(N, \mathscr{P}_{\text{word}}, \mathscr{C}_{\text{pm}}, r, \delta) = \Omega(N^{1/5})$.

Proof. Recall that $N = 2^m$. Say for simplicity that m is even. Using only the fact that $\mathscr{C}_{\text{pm}}$ contains 3^m subsets, a simple probabilistic argument shows that there is a set D with $|D| = O(mN^{1/2})$ such that if $Q \in \mathscr{C}_{\text{pm}}$ and $|Q| \geqq N^{1/2}$ then $Q \cap D \neq \varnothing$. Specifically, for each $u \in U$, put u in D with probability $q = \alpha m N^{-1/2}$ where α is a constant to be chosen later. By Chebyshev's inequality,

$$\Pr\{|D| \geqq 2\alpha m N^{1/2}\} = O(N^{-1/2}).$$

If $Q \subseteq U$ and $|Q| \geqq N^{1/2}$,

$$\Pr\{D \cap Q = \varnothing\} \leqq (1-q)^{|Q|} \leqq e^{-\alpha m}.$$

Choose the constant α so that $3^m e^{-\alpha m} \leqq \frac{1}{2}$. Then for all sufficiently large N, with nonzero probability $|D| = O(mN^{1/2})$ and D intersects all $Q \in \mathscr{C}_{\text{pm}}$ having $|Q| \geqq N^{1/2}$.

If $w \in \{0, 1, *\}^m$ is the name of a partial match sample Q_w, let $*w$ be the number of $*$'s in w. Note that $|Q_w| = 2^{*w}$. Choose integer d such that

$$2^{m/2+d} \geqq (r^2+1)|D|.$$

Note that $d = O(\log m)$. Let

$$\{V_1, \cdots, V_t\} = \{Q_w | w \in \{0, 1, *\}^m \text{ and } *w = m/2 + d\}.$$

II(A) holds by choice of d. If Q_u is "small," i.e., if $*u \leqq m/2$, we must find an upper bound for $f(u)$ defined to be the fraction of the V_j's that Q_u intersects. Note that $Q_u \cap Q_w \neq \varnothing$ iff u matches w. Since $f(u)$ is an upper bound, we can assume that $*u = m/2$, and by symmetry assume that $u = u_1 u_2$ where u_1 is a string of $m/2$ 0's and 1's and u_2 is a string of $m/2$ $*$'s. Consider the w's with $*w = m/2 + d$ that have k $*$'s in the first $m/2$ positions and therefore $m/2 + d - k$ $*$'s in the last $m/2$ positions. Letting $N(u, k)$ be the number of such w's that match u,

$$N(u, k) = \binom{m/2}{k}\binom{m/2}{m/2+d-k} 2^{k-d},$$

$$t = \binom{m}{m/2+d} 2^{m/2-d},$$

$$f(u) = \left(\sum_{k=d}^{m/2} N(u, k)\right) \Big/ t.$$

Since $d = O(\log m)$, there is a constant c such that

$$\binom{m}{m/2+d} \geqq 2^m / m^c,$$

so $t \geqq 2^{3m/2-d}/m^c$. Therefore,

$$f(u) \leqq \left[\sum_{k=d}^{m/2} \binom{m/2}{k} 2^k\right]\left[\sum_{k=d}^{m/2} \binom{m/2}{k-d}\right] 2^{-3m/2} m^c \leqq 3^{m/2} 2^{m/2} 2^{-3m/2} m^c.$$

Therefore, a union of $N^{1/5} (= 2^{m/5})$ partial match samples intersects at most the fraction

$$2^{m/5} \cdot f(u) \leqq 2^{m/5} 3^{m/2} 2^{-m} m^c = o(1)$$

of the V_j's, and the lower bound follows as outlined above. □

THEOREM 2.3. $H(N, \mathscr{P}_{\text{tree}}, \mathscr{C}_{\text{subtree}}, r, \delta) = O((N \log N)^{1/2})$.

Proof. Number the nodes of F in inorder (to number a tree, number the left subtree, then the root, then the right subtree). For $u \in \{0,1\}^{\leq n}$ let $d(u)$ be the inorder label of the node u. If X is a subtree of F, define

$$d(X) = \{d(u) | u \in X\}.$$

For $1 \leq a \leq b \leq N$, let

$$[a, b] = \{k | a \leq k \leq b\},$$

and define $\text{INT}(X, [a, b])$ to be 1 iff $d(X) \cap [a, b] \neq \emptyset$. Because the integer labeling is inorder, for each interval $[a, b]$ and any two nodes u and v with $d(u), d(v) \in [a, b]$, if w is the least common ancestor of u and v, then $d(w) \in [a, b]$. Therefore, for each $[a, b]$ there is a unique w $(= w(a, b))$ such that $d(w) \in [a, b]$ and, for all nodes u, $d(u) \in [a, b]$ implies that w is an ancestor of u. Since X satisfies the tree property

$$\text{INT}(X, [a, b]) = \text{INT}(X, S_{w(a,b)}).$$

Therefore, it suffices to estimate $|X|$ using "interval samples" of the form $[a, b]$. For some parameter M, the algorithm has two cases, $|X| \leq M$ and $|X| \geq M$. The first case has M phases and does not use randomization. In the first phase, by asking questions of the form $\text{INT}(X, [1, k])$ and doing binary search on k, find the smallest k, say k_1, such that $k \in d(X)$; this takes $O(\log N)$ samples. During the second phase, by asking questions of the form $\text{INT}(X, [k_1 + 1, k])$ and again doing binary search on k, find the second smallest k with $k \in d(X)$, and so on. The first case uses $O(M \log N)$ samples. If $|X| < M$, then $|X|$ will be found exactly at phase number $|X| + 1$. If each phase in the first case finds a new element of $d(X)$, then we know that $|X| \geq M$, and the second case is invoked. In this case, since $|X| \geq M$, Chebyshev's inequality implies that there is a constant α (depending only on r and δ) such that the algorithm approximates $|X|$ to within the factor r with error probability δ by making $\alpha N/M$ independent "singleton" samples of the form $[k, k]$, calculating the fraction γ of samples with $\text{INT}(X, [k, k]) = 1$, and answering $\lfloor \gamma N \rfloor$. The total number of samples for both cases is $O(M \log N + \alpha N/M)$ so choosing $M = (N/\log N)^{1/2}$ gives the result. □

3. A general upper bound. An NP-*machine* is a nondeterministic polynomial-time Turing machine [3], [6]. Assume that NP-machines have at most two nondeterministic choices at each step so that accepting computations on inputs of length n can be represented as binary strings of length $p(n)$ where $p(n)$ is the machine's polynomial time bound. If M is an NP-machine and $x \in \{0,1\}^*$ is an input, let $\text{Acc}_M(x) \subseteq \{0,1\}^{p(|x|)}$ be the set of accepting computations of M on input x. Define the function $C_M : \{0,1\}^* \to \mathbf{N}$ by

$$C_M(x) = |\text{Acc}_M(x)|.$$

Valiant's [17] class #P is

$$\#\text{P} = \{C_M | M \text{ is an NP-machine}\}.$$

For $A \subseteq \{0,1\}^*$, $\#\text{P}^A$ is defined similarly except that M is a nondeterministic polynomial-time oracle machine [2], [15] that can construct binary strings and make decisions based on whether or not they belong to A. We also need a few classes of the relativized P-hierarchy. Let $\text{P}^A(\text{NP}^A)$ be the class of languages accepted by deterministic (nondeterministic) polynomial-time oracle machines with oracle A. If $\mathscr{L}$ is a class of languages, let $\text{P}(\mathscr{L})(\text{NP}(\mathscr{L}))$ be the union of $\text{P}^A(\text{NP}^A)$ over all $A \in \mathscr{L}$. Let co-$\mathscr{L}$

be the class of complements of languages in $\mathscr{L}$. Define

$$\Sigma_2^{p,A} = \mathrm{NP}(\mathrm{NP}^A),\ \Pi_2^{p,A} = \text{co-}\Sigma_2^{p,A},$$
$$\Delta_2^{p,A} = \mathrm{P}(\mathrm{NP}^A),\ \Delta_3^{p,A} = \mathrm{P}(\Sigma_2^{p,A}).$$

When A is not present, the empty oracle is assumed. We extend the classes $\Delta_2^{p,A}$ and $\Delta_3^{p,A}$ to include functions from $\{0,1\}^*$ to $\mathbf{N}$; since the "toplevel" machine is deterministic, the definition of this extension should be clear.

DEFINITION. If $f, g: \{0,1\}^* \to \mathbf{N}$ and $r: \mathbf{N} \to \mathbb{R}$, we say that g *r-approximates* f if, for all n and all x of length n,

$$\frac{f(x)}{r(n)} \leqq g(x) \leqq r(n) \cdot f(x).$$

THEOREM 3.1. *Let $f \in \#\mathrm{P}$ and let $\varepsilon, d > 0$. There is a $g \in \Delta_3^p$ such that g $(1+\varepsilon n^{-d})$-approximates f. Moreover, this relativizes to an arbitrary oracle A.*

Proof. Let $f = C_M$ for some NP-machine M and let $p(n)$ be M's polynomial time bound. Fix an input length n, and let $t = p(n)$. Consider the following predicate.

Hash(x, m):
There exist $m \times t$ 0-1 matrices $H_1, \cdots, H_m$ such that

(2) for each $z \in \mathrm{Acc}_M(x)$, there exists an i such that $H_i z \neq H_i z'$ for all $z' \in \mathrm{Acc}_M(x) - \{z\}$.

Here, $H_i z$ means multiplication of the $m \times t$ matrix H_i by the t-vector z, yielding some m-vector in $\{0,1\}^m$, where arithmetic is done modulo 2. The key fact, proved by Sipser [13], is that there is a fixed constant c such that

$$|\mathrm{Acc}_M(x)| \leqq 2^{m-c} \Rightarrow \mathrm{Hash}(x, m). \tag{3}$$

(Actually, this holds for any subset of $\{0,1\}^t$, not just those of the form $\mathrm{Acc}_M(x)$.) On the other hand, if Hash(x, m) is true then, by (2), for each $z \in \mathrm{Acc}_M(x)$ there is a *unique* pair $(i, H_i z)$ where $1 \leqq i \leqq m$ and $H_i z \in \{0,1\}^m$, so

$$\mathrm{Hash}(x, m) \Rightarrow |\mathrm{Acc}_M(x)| \leqq m 2^m. \tag{4}$$

Gács has observed that the predicate Hash(x, m) belongs to Σ_2^p (see [13]). To see this, first note that the existential quantifier, "there exists an i" in (2) has range $m = O(p(n))$, so it can be replaced by a deterministic search. Once this is done, the definition of Hash(x, m) has an obvious $\exists\forall$ form. Therefore, a deterministic polynomial-time machine, making calls on an oracle for Hash, can find the minimum m such that Hash(x, m) is true. If m is this minimum then, by (3) and (4),

$$2^{m-c-1} \leqq |\mathrm{Acc}_M(x)| \leqq m 2^m,$$

so we have computed $C_M(x)$ to within the factor $m2^{c+1}$. To achieve the smaller factor $1+\varepsilon n^{-d}$, by running M k times in series it is easy to modify M to an NP-machine R with

$$C_R(x) = (C_M(x))^k \quad \text{for all } x.$$

Applying the above hashing procedure to R, we can compute $C_R(x)$ to within the factor $m2^{c+1}$, so we can compute $C_M(x)$ to within the factor $(m2^{c+1})^{1/k}$, where now $m = O(k \cdot p(n))$. By choosing k to be a sufficiently large polynomial ($k = O(n^{d+1})$ works), this latter factor is less than $1+\varepsilon n^{-d}$. The relativized case is identical. □

The next result shows that any attempt to replace Δ_3^p by Δ_2^p in Theorem 3.1 cannot proceed by a proof that relativizes to an arbitrary oracle.

THEOREM 3.2. *There are a recursive oracle A and a function $f \in \#\mathrm{P}^A$ such that, for any constant r, if g r-approximates f then $g \notin \Delta_2^{p,A}$.*

Proof. Let $h(n) = n^{\log n}$. For $A \subseteq \{0,1\}^*$, let

$$C_A(n) = |A \cap \{0,1\}^n|.$$

The oracle A will be constructed such that, for all n,

$$\text{either } C_A(n) \leqq h(n) \text{ or } C_A(n) \geqq 2^n - h(n). \tag{5}$$

Let $f(x) = C_A(|x|)$. Clearly $f \in \#\mathrm{P}^A$. Let

$$L = \{x \mid C_A(|x|) \leqq h(|x|)\}.$$

If $g \in \Delta_2^{p,A}$ and g r-approximates f for some constant r, then by using property (5) of A it is easy to see that $L \in \Delta_2^{p,A}$. We construct A so that $L \notin \Delta_2^{p,A}$.

Let $M_1, M_2, \cdots$ be an enumeration of all pairs (D_j, N_k) for $j, k \geqq 1$, where $D_1, D_2, \cdots$ is an enumeration of the deterministic polynomial-time oracle Turing machines and $N_1, N_2, \cdots$ is an enumeration of the nondeterministic polynomial-time oracle Turing machines. Let p_j (resp., q_k) be the polynomial running time of D_j (resp., N_k). If $L \in \Delta_2^{p,A}$ then there is some $M_i = (D_j, N_k)$ such that D_j accepts L when D_j calls N_k as an oracle and N_k calls A as an oracle.

The construction of A proceeds by stages. During the construction we have two disjoint finite sets of strings B and $\bar{B}$ that grow dynamically. At any point, B (resp., $\bar{B}$) is the set of strings that have been committed to be in A (resp., not in A). Strings are never removed from B or $\bar{B}$. Initially, $B = \bar{B} = \varnothing$. During the ith stage we extend the definitions of B and $\bar{B}$ so that M_i does not accept L. Let n_1 be so large that

$$h(n_1) < 2^{n_1} - h(n_1).$$

Place all strings of length $\leqq n_1$ in $\bar{B}$. The construction has the property that just before the start of stage i, a string is committed (in either B or $\bar{B}$) iff its length is $\leqq n_i$.

Stage i. Let $M_i = (D_j, N_k)$. Choose $n > n_i$ so large that

$$p_j(n) \cdot q_k(p_j(n)) < h(n).$$

Set $n_{i+1} = q_k(p_j(n))$. If $n_i < |z| \leqq n_{i+1}$ and $|z| \neq n$, place z in $\bar{B}$. Let $x = 0^n$. Start simulating D_j on input x until D_j makes its first oracle call, asking whether the string y is accepted by N_k with oracle A. Since $|y| \leqq p_j(n)$, any oracle query "$z \in A$?" made by N_k has $|z| \leqq n_{i+1}$. Thus, at this point, the answer to any such query with $|z| \neq n$ is determined by the commitments to B and $\bar{B}$ made so far. There are two possibilities.

1. For all ways of extending B and $\bar{B}$ by adding uncommitted strings of length n, N_k does not accept y. In this case, the answer to D_j's query is that N_k does not accept y. Continue simulating D_j until its next oracle call.

2. If the first case does not hold, then N_k has an accepting computation α on input y. Let S be the set of uncommitted strings of length n that are queried along α. Note that $|\alpha| \leqq n_{i+1}$, so $|S| \leqq n_{i+1}$. For each $z \in S$, commit z to be in either B or $\bar{B}$ depending on whether the answer to the query "$z \in A$?" in α is "yes" or "no," respectively. After these commitments, N_k will accept y no matter how B and $\bar{B}$ are extended further, since a nondeterministic machine accepts if it has any accepting computation, and further additions to B and $\bar{B}$ cannot invalidate the accepting computation α. Continue simulating D_j until its next oracle call.

Each subsequent oracle call of D_j is handled similarly by either case 1 or 2. At the end of D_j's computation, at most $p_j(n) \cdot n_{i+1} < h(n)$ strings of length n have been committed. If D_j accepts (resp., rejects) x, place all uncommitted strings of length n in B (resp., $\bar{B}$). In either case, D_j makes an error. □

4. BPP and the P-hierarchy. A *probabilistic oracle Turing machine* has both coin-tossing states as in the definition of probabilistic Turing machines [4] and oracle query states as in the definition of oracle Turing machines [2], [6]. Following Gill [4] define BPP^A to be the class of languages accepted by polynomial-time probabilistic oracle Turing machines which, for all inputs, have error probability $\leqq \frac{1}{2} - \varepsilon$ for some fixed $\varepsilon > 0$ when the oracle A is used.

Sipser and Gács [13] show that, for any oracle A,

$$\mathrm{BPP}^A \subseteq \Sigma_2^{p,A} \cap \Pi_2^{p,A}.$$

THEOREM 4.1. *There is a recursive oracle A such that*

$$\mathrm{BPP}^A \not\subset \Delta_2^{p,A}.$$

Proof. Let $h(n)$, A and L be as in the proof of Theorem 3.2. Since $L \notin \Delta_2^{p,A}$ we only have to observe that $L \in \mathrm{BPP}^A$. Given an input of length n, the probabilistic oracle Turing machine generates a random string z of length n and asks whether $z \in A$. If $z \in A$ then the machine rejects, or if $z \notin A$ then the machine accepts. The error probability is $\leqq n^{\log n}/2^n$ which is less than $\frac{1}{4}$ for all sufficiently large n. □

5. Conclusion. It has been shown that any function in #P can be approximated by a function in Δ_3^p. Concerning lower bounds, if the #P problem is based on an NP-complete problem, for example, counting the number of satisfying truth assignments of a given propositional formula, it is easy to see that computing r-approximate solutions is NP-hard for any constant r. However, for approximately computing the permanent of a 0-1 matrix, where the corresponding existential question is solvable in polynomial time, the issue of lower bounds is open.

Question. Classify the computational complexities of approximately computing the permanent of a 0-1 matrix and approximately counting the number of satisfying assignments of a propositional formula. In particular, is the former problem NP-hard? Is the latter problem $\mathcal{L}$-hard for some class $\mathcal{L}$ of the P-hierarchy above NP?

Recently, Karp and Luby [7] have discovered polynomial-time probabilistic approximation algorithms for certain problems in #P such as counting the number of satisfying assignments of a propositional formula given in disjunctive normal form.

Acknowledgments. I am grateful to Richard Lipton for suggesting the question of whether the size of a tree could be estimated using subtree samples and for many helpful discussions. One of the referees provided an extensive list of comments which helped improve the presentation of this material.

REFERENCES

[1] D. ANGLUIN, *On counting problems and the polynomial time hierarchy*, Theoret. Comput. Sci., 12 (1980), pp. 161-173.

[2] T. BAKER, J. GILL AND R. SOLOVAY, *Relativizations of the* P = ?NP *question*, this Journal, 4 (1975), pp. 431-442.

[3] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability, A Guide to the Theory of* NP-*Completeness*, W. H. Freeman, San Francisco, 1979.

[4] J. GILL, *Computational complexity of probabilistic Turing machines*, this Journal, 6 (1977), pp. 675-695.

[5] L. M. GOLDSCHLAGER, *An approximation algorithm for computing the permanent*, Lecture Notes in Mathematics, 829, Springer-Verlag, Berlin, pp. 141–147.
[6] J. E. HOPCROFT AND J. D. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.
[7] R. KARP AND M. LUBY, *Monte-Carlo algorithms for enumeration and reliability problems*, Proc. 24th IEEE Symposium on Foundations of Computer Science, 1983, pp. 56–64.
[8] D. E. KNUTH, *Estimating the efficiency of backtrack programs*, Math. of Comput., 29 (1975), pp. 121–136.
[9] E. LAWLER, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.
[10] M. LUBY, personal communication.
[11] U. MANBER AND M. TOMPA, *Probabilistic, nondeterministic, and alternating decision trees*, Proc. 14th ACM Symposium on Theory of Computing, 1982, pp. 234–244.
[12] M. O. RABIN, *Probabilistic algorithms in finite fields*, this Journal, 9 (1980), pp. 273–280.
[13] M. SIPSER, *A complexity theoretic approach to randomness*, Proc. 15th ACM Symposium on Theory of Computing, 1983, pp. 330–335.
[14] R. SOLOVAY AND V. STRASSEN, *A fast Monte-Carlo test for primality*, this Journal, 6 (1977), pp. 84–85.
[15] L. J. STOCKMEYER, *The polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1977), pp. 1–22.
[16] ———, *The complexity of approximate counting*, Proc. 15th ACM Symposium on Theory of Computing, 1983, pp. 118–126.
[17] L. G. VALIANT, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189–201.
[18] ———, *The complexity of enumeration and reliability problems*, this Journal, 8 (1979), pp. 410–421.