INFORMATION AND COMPUTATION **82**, 34–44 (1989)

Enumerative Counting Is Hard

JIN-YI CAI*

Department of Computer Science, Yale University, New Haven, Connecticut 06520

AND

LANE A. HEMACHANDRA[†]

Department of Computer Science, University of Rochester, Rochester, New York 14627

An n-variable Boolean formula may have anywhere from 0 to 2^n satisfying assignments. Can a polynomial-time machine, given such a formula, reduce this exponential number of possibilities to a small number of possibilities? We call such a machine an enumerator and prove that if there is a good polynomial-time enumerator for $\#\mathrm{P}$ (i.e., one where for every Boolean formula f, the small set has at most $O(|f|^{1-\varepsilon})$ numbers), then $\mathrm{P} = \mathrm{NP} = \mathrm{P}^{\#\mathrm{P}}$ and probabilistic polynomial time equals polynomial time. Furthermore, we show that $\#\mathrm{P}$ polynomial-time Turing reduces to enumerating $\#\mathrm{P}$. © 1989 Academic Press, Inc.

1. INTRODUCTION

The class $\#\mathrm{P}$ consists of functions that count the accepting paths of some NP machine (Valiant, 1979b; Angluin, 1980; Stockmeyer, 1985; Wagner, 1986). Valiant proved that these functions can count the number of cliques of a given size, compute the permanent of a matrix, and solve many other counting versions of NP and P problems (Valiant, 1979b, 1979a). $\mathrm{P}^{\#\mathrm{P}}$ is the class of languages computable by polynomial-time machines given an oracle from $\#\mathrm{P}$.

$\#\mathrm{P}$ questions seem computationally complex. Though they can be answered by brute force in PSPACE, it is not known if they are in the polynomial hierarchy. However, some structural relations and

* Supported by a Cornell University Sage Fellowship and NSF Grants DCR-8301766 and CCR-8709818.

[†] Supported by a Fannie and John Hertz Foundation Fellowship, a Hewlett-Packard Corporation equipment grant, and NSF Grants DCR-8301766, DCR-8520597, and CCR-8809174. Work done while at Columbia University and Cornell University.

relativizations of $P^{\#P}$ are known. Clearly, $P^{\#P} \supseteq \Delta_2^p$. Angluin (1980) showed that there exists a relativized world in which $P^{\#P^A} \nsubseteq \Sigma_2^{p,A} \cup \Pi_2^{p,A}$. This was improved significantly by Yao, who showed, in a certain relativized world A, that $P^{\#P^A}$ is not contained in the polynomial hierarchy relative to A (Yao, 1985). This result was improved by Cai, who showed that $P^{\#P^A}$ is not contained in the polynomial hierarchy relative to A, for almost all oracles A (Cai, 1986; see also Babai, 1987). Related work on circuit complexity can also be found in Hastad (1986) and Smolensky (1987).

The complexity of the class #P is usually studied by considering #SAT. Cook's reduction (Cook, 1971; Hopcroft and Ullman, 1979) from NP machines to formulas can be made parsimonious (Garey and Johnson, 1979, p. 169; Simon, 1977; Valiant, 1979b). Thus #SAT, the function mapping from Boolean formulas f to their numbers of solutions $\|f\|$, is a canonical hardest #P function. We speak interchangeably of computing #P and #SAT, as #SAT can be computed in polynomial time if and only if #P can.

Stockmeyer (1985) has analyzed the complexity of $r(\cdot)$-approximating #SAT in the sense of approximating the value within a multiplicative factor—finding a function g such that

$$\frac{\|f\|}{r(|f|)} \leqslant g(f) \leqslant \|f\| \, r(|f|),$$

where $|f|$ stands for the size of f. He shows that, for any ε, $d > 0$, we can $(1 + \varepsilon\, |f|^{-d})$-approximate #SAT in Δ_3^p. This is a good bound when the number of solutions $\|f\|$ is relatively small. However, for formulas with *many* solutions, the size of the set of possible values that the approximation admits may be exponentially large in terms of $|f|$.

In his paper, Stockmeyer (1985) also shows that there is a relativized world where for no constant k can #SAT be k-approximated even with a Δ_2^p function. Though this does not prove that #SAT is hard to approximate, the result can be taken as evidence that we lack the mathematical tools needed to determine the complexity of approximating #SAT.

In this paper, we consider a different approach to approximately solving #P problems. In *an enumeration scheme* one tries to reduce the number of possible values of $\|f\|$ by giving a small list of possibilities for $\|f\|$. A function A is said to $s(\cdot)$-*enumerate* #SAT if $A(f)$ is a list of at most $s(|f|)$ integers between 0 and $2^{|f|}$ in which $\|f\|$ appears. If A can be computed in polynomial time, we call it a *polynomial-time* $s(\cdot)$-*enumerator* for #SAT. We show that the existence of a good enumerator that reduces the number of values substantially would imply that $P = NP = P^{\#P}$, and that probabilistic polynomial time equals polynomial time. Thus we believe

"#SAT *is* hard to enumerate" with greater certainty than we believe "$P \neq NP$."

Though enumerators reduce the number of possible solutions to a small set, the values in this set may vary over a great range. Thus enumeration is neither strictly stronger nor strictly weaker than the type of approximation studied by Stockmeyer.

Section 2 introduces our proof techniques in a simple setting. If #SAT has a polynomial-time k-enumerator then $P = NP$. Section 3 extends this result to show that if *Enum* is a function n^α-enumerating #SAT, $\alpha < 1$, then $P^{\#P} \subseteq P^{Enum}$. Thus exact counting polynomial-time Turing reduces to approximate counting. Furthermore, if *Enum* is also computable in $P^{\#P}$ (as is true for any useful enumerator), then we may conclude that $P^{\#P} = P^{Enum}$. In particular, if #SAT has a polynomial-time n^α-enumerator ($\alpha < 1$), then $P = P^{\#P}$.

These results demonstrate that efficiently enumerating #P implies $P = NP$. Thus we have structural evidence that #P cannot be easily enumerated.

Our proof ues an arithmetic of formulas that extends the work of Papadimitriou and Zachos (1983). Section 4 presents an immediate consequence of this: $NP^{\#P} = NP^{\#P[1]}$, where [1] indicates that each computation path of the NP machine makes at most one oracle call. We invite comparison between this n^k-for-one result over NP machines, and the recently developed theory of polynomial terseness (Amir and Gasarch, 1988; Goldsmith, Joseph, and Young, 1987; Beigel *et al.*, 1987).

2. If #SAT Can Be Polynomial-Time k-Enumerated then $P = NP$

The proofs of this section and Section 3 have the same architecture. We develop a novel technique to repeatedly expand and prune a formula tree. In this section we keep the tree constantly thin. Section 3 allows trees that are polynomially bushy.

This section shows that if #SAT can be polynomial-time k-enumerated then $P = NP$. First, we state a lemma which says that we can easily combine many small formulas into a single larger formula. This lemma generalizes a technique developed by Papadimitriou and Zachos (1983) in their proof that $P^{NP[\log]} \subseteq P^{\#P[1]}$. The single large formula has the property that, given its number of solutions, we can quickly determine the number of solutions of each of the small formulas.

LEMMA 2.1. *There are polynomial-time functions combiner and decoder such that for any Boolean formulas f and g,* combiner(f, g) *is a Boolean formula and* decoder(f, g, $\|$combiner(f, g)$\|$) *prints* $\|f\|$, $\|g\|$.

Proof. Let $f=f(x_1, \ldots, x_n)$ and $g=g(y_1, \ldots, y_m)$, where $x_1, \ldots, x_n$, $y_1, \ldots, y_m$ are distinct. Let z and z' be two new Boolean variables. Then

$$h=(f \wedge z) \vee (\bar{z} \wedge x_1 \wedge \cdots \wedge x_n \wedge g \wedge z') \tag{1}$$

is the desired combination, since $\|h\| = \|f\|\, 2^{m+1} + \|g\|$ and $\|g\| \leqslant 2^m$.
Q.E.D.

We can easily extend this technique to combine three, four, or even polynomially many formulas. (Using Cook's connection between formulas and machines, one can equivalently view this result as about counting and combining NP machines and accepting paths.)

Now we show that if #SAT has a polynomial-time k-enumerator then P = NP.

THEOREM 2.2. *If* #SAT *can be polynomial-time k-enumerated then* P = NP.

Proof. Say we are given a formula $F(x_1, \ldots, x_n)$ and we would like to know if $F \in$ SAT. We substitute variables one at a time so that we always have a set S of at most k partial assignments satisfying the following invariant:

$$F \in \text{SAT} \Leftrightarrow \text{there is a satisfying assignment extending a partial assignment in } S. \tag{$*$}$$

Each stage assigns a new variable and has three steps. Initially, S consists of the empty assignment.

Stage i:

1. EXPAND TREE. For each partial assignment in S, assign the variable x_i both true and false (Fig. 1A). Applying these assignments to F, we have at most $2k$ formulas.

2. COMBINE FORMULAS and RUN ENUMERATOR. Combine the $2k$ formulas into a single superformula as described in Lemma 2.1. Run our polynomial-time k-enumerator on that superformula. The enumerator prints at most k guesses for the number of solutions of the superformula, which are translated immediately to at most k vectors of guesses of the number of solutions of the $2k$ formulas. (For example, in Fig. 1B (where $k=3$) the first guess says that the four little formulas in our superformula have, respectively, 7, 0, 3, and 3 solutions.)

3. PRUNE THE TREE. Note that if $F \in$ SAT, then at least one of the $2k$ formulas is satisfiable, by the inductive hypothesis ($*$). Thus, if these k guesses are all zero vectors then the formula is unsatisfiable. If they are not all zero vectors, we can choose a set T of at most k columns so that

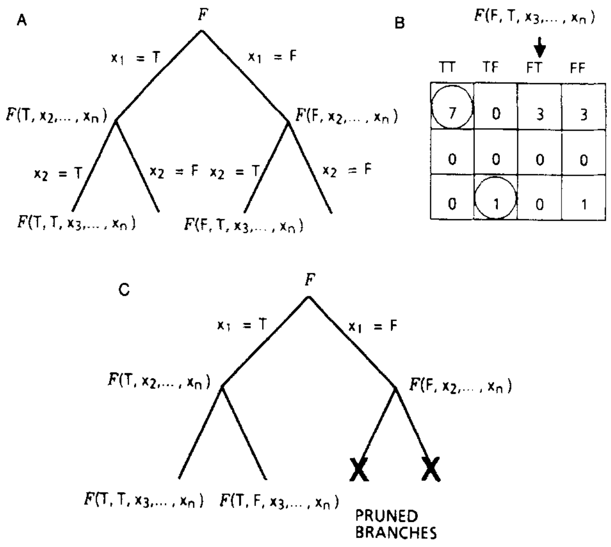

FIG. 1. A. The tree; B. the guess matrix; C. the pruned tree.

each nonzero row of our guess matrix (Fig. 1B) has a nonzero in a column in T. For example, we let $T = \{p \mid$ a row of the guess matrix has its first nonzero entry in column $p\}$.

Now we prune the tree (Fig. 1C) by setting S to the partial assignements corresponding to the columns in T (there are at most k). Suppose $F \in \mathrm{SAT}$. Then by the inductive hypothesis, the correct guess is a nonzero vector, thus by our choice of T the correct guess has a nonzero in some column of T. We have assigned another variable and maintained invariant $(*)$.
End of Stage i.

At the final stage all variables are assigned and we just have to look at our set of k complete assignments to F and see if any of them satisfies F. We know by $(*)$ that F is satisfiable if and only if one of these assignments satisfies F. Q.E.D.

3. Main Result

This section demonstrates that exact counting polynomial-time Turing reduces to enumerative counting. This strengthens the result of the previous section.

THEOREM 3.1. *If Enum is an n^{α}-enumerator for* $\#\mathrm{SAT}$, $\alpha < 1$, *then*

$$\mathrm{P}^{\#\mathrm{P}} \subseteq \mathrm{P}^{Enum}.$$

COROLLARY 3.2. *If Enum is a n^{α}-enumerator for* #SAT, $\alpha < 1$ *and is computable in* $\mathrm{P}^{\#\mathrm{P}}$, *then*

$$\mathrm{P}^{\#\mathrm{P}} = \mathrm{P}^{Enum}.$$

COROLLARY 3.3. *If* #SAT *can be polynomial-time n^{α}-enumerated,* $\alpha < 1$, *then* $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$.

Since $\mathrm{P}^{\#\mathrm{P}} = \mathrm{P}^{\mathrm{PP}}$ (Gill, 1977; Simon, 1975), where PP is probabilistic polynomial time, the existence of good enumerators for #SAT also implies that probabilistic and deterministic polynomial time are equivalent.

COROLLARY 3.4. *If* #SAT *can be polynomial-time n^{α}-enumerated,* $\alpha < 1$, *then* $\mathrm{P} = \mathrm{PP}$.

Theorem 3.1 differs from Theorem 2.2 in two important ways. One is that we are satisfied with an n^{α}-enumerator, $\alpha < 1$. The more interesting point is that we conclude $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$. We now discuss each of these improvements.

3.1. *How to Count Solutions*

The first major change is that we find out not only if a formula is satisfiable, but also how many satisfying assignments it has. We do this with a more rigorous analysis of the guess matrix and a refined pruning strategy.

LEMMA 3.5. *If* #SAT *can be polynomial-time k-enumerated then* $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$.

Proof. Consider the tree pruning procedure in Theorem 2.2. Here we want to keep a set S of p $(1 \leqslant p \leqslant k)$ leaves in the partially grown tree, such that $\langle \|f_1\|, \|f_2\|, ..., \|f_p\| \rangle$ uniquely determine $\|f\|$, where f_j is the formula obtained from f by the partial assignment associated with the jth leaf in S. (We will speak interchangeably of the jth leave in S and f_j.)

Again we substitute variables one at a time. Inductively, for the formulas $f_1, ..., f_p$ in S, we wish to maintain at most k vectors u_i $(i = 1, ..., q,\ q \leqslant k)$ of dimension p, and integers $s_1, ..., s_q$, such that the following conditions hold:

1° $(\forall i)[u_i \neq 0]$,

2° $(\forall i \neq j)[u_i \neq u_j]$, and

3° $f \in \mathrm{SAT} \Rightarrow (\exists i)[u_i = \langle \|f_1\|, ..., \|f_p\| \rangle \wedge s_i \|f\|]$.

Notice that when the tree has fully grown, for the formulas $f_1, ..., f_p$ in S it can be easily checked whether some $u_i = \langle f_1\|, ..., \|f_p\| \rangle$. If no such u_i exists, then f is unsatisfiable, by 3°. If such u_i exists, we know from 1° that f

is satisfiable, and the u_i must be unique, by condition 2°. And thus by condition 3° we can ouput $\|f\| = s_i$.

The proof is a double induction. Each stage has the same general structure as in the proof of Theorem 2.2.

Initially S consists of f (or, the empty assignement) and we apply our enumerator on f. If the enumerator guesses all zeros, then output $\|f\| = 0$. Otherwise, let $u_1, ..., u_q$ be all the distinct nonzero guesses $(1 \leqslant q \leqslant k)$, and $s_i = u_i$ trivially.

We inductively maintain 1°, 2°, and 3° as we go along, and at each stage we use a second induction for the tree pruning process.

Stage l:

1. EXPAND TREE. For each partial assignment in S, assign the variable x_l both true and false. We have r new leaves, where $2 \leqslant r = 2\,|S| \leqslant 2k$.

2. COMBINE FORMULAS and RUN ENUMERATOR. Combine f with the formulas $f_1, ..., f_r$ associated with these new leaves. Let G be the resulting "superformula." Run our polynomial-time k-enumerator for G. We obtain at most k guesses for the number of solutions of G. Using our decoder we get up to k distinct vectors, say $\widehat{v_1}, ..., \widehat{v_{q'}}$, $1 \leqslant q' \leqslant k$, where $\widehat{v_i} = \langle v_{i0}, v_{i1}, ..., v_{ir} \rangle$ is a guess for $\langle \|f\|, \|f_1\|, ..., \|f_r\| \rangle$.

3. PRUNE THE TREE. Let $v_i = \langle v_{i1}, ..., v_{ir} \rangle$. If some $v_i = 0$ we may discard $\widehat{v_i}$. In fact, if $f \in \mathrm{SAT}$ then at least one of the formulas in S is satisfiable, by inductive hypotheses 3° and 1°. Thus at least one of the new leaves is satisfiable. Since $v_i = 0$ cannot be a correct guess if $f \in \mathrm{SAT}$, deleting it causes no harm, in terms of maintaining 1°, 2°, and 3°.

Second, for any pair $\widehat{v_i}$, $\widehat{v_j}$, if $v_i = v_j$, we can effectively delete at least one of them. This is because $v_i \neq v_j$ and $v_i = v_j$ imply $\widehat{v_{i0}} \neq \widehat{v_{j0}}$. Now $\langle v_{i1} + v_{i2}, ..., v_{ir-1} + v_{ir} \rangle$ must equal one of the u's (call it u_t) associated with the formulas in S (otherwise $\widehat{v_i}$ as well as $\widehat{v_j}$ is clearly false by 3°). This u_t must be unique by 2°; furthermore, either $v_{i0} \neq s_t$ or $v_{j0} \neq s_t$.

If $v_{i0} \neq s_t$, clearly $\widehat{v_i}$ is false; we may delete $\widehat{v_i}$. The same argument applies to v_j. Without loss of generality, we are left with $\langle \widehat{v_1}, ..., \widehat{v_q} \rangle$, $q \leqslant k$. If $q = 0$ then output $\|f\| = 0$. In fact, if $f \in \mathrm{SAT}$ then some $\widehat{v_i}$ with $v_i \neq 0$ must be a correct guess and must have been kept.

Let $s_i = v_{i0}$, $i = 1, ..., q$, $1 \leqslant q \leqslant k$. Note that $v_1, ..., v_q$ and the s_i's satisfy conditions 1°, 2°, and 3°. Let the guess matrix consist of $v_1, ..., v_q$ as row vectors. We will inductively extract at most q columns of the matrix, so that the q row vectors of the submatrix (the projection of $v_1, ..., v_q$ onto the chosen dimensions) also satisfy 1°, 2°, and 3°.

Since 3° is automatically satisfied with any subset of columns, we need only to maintain 1° and 2°. To prune, initially let $j_1 = \min\{j \geqslant 1 \mid v_{1j} \neq 0\}$.

Let $w_1 = (v_{1j_1}) \in N^1$. Inductively, suppose $w_1, ..., w_h \in N^{h'}$ have been constructed, $h' \leqslant h < q$, satisfying 1° and 2°. Each w_i is the projection of v_i onto the h' chosen columns. Let $\widetilde{w_{h+1}}$ be the projection of v_{h+1} on these h' columns.

If $\widetilde{w_{h+1}} = w_i$ for some $1 \leqslant i \leqslant h$, then this i is unique, by 2°. Also, $\widetilde{w_{h+1}} \neq 0$ by 1°. Now all we need to do is to distinguish w_{h+1} from w_i. But since $v_{h+1} \neq v_i$, this is easily done by choosing one more column. (Every $w_1, ..., w_h$, $\widetilde{w_{h+1}}$ is extended one dimension to get the new $w_1, ..., w_h$ and w_{h+1}.) If $\widetilde{w_{h+1}} \neq w_i$, for all $1 \leqslant i \leqslant h$, then we only need to ensure $\widetilde{w_{h+1}} \neq 0$. Again this is easily done by extending at most one dimension, since $v_{h+1} \neq 0$.

Finally, $w_1, ..., w_q$ are constructed. Set u_i to w_i and p to the dimension of w_i; S consists of those new leaves corresponding to the p selected columns. 1°, 2°, and 3° are satisfied.

End of Stage l. Q.E.D.

3.2. *Dealing with Polynomial Enumerators*

In this section we show how to combine many formulas into a superformula efficiently and prune so that our tree does not blow up in width. This is an extension of the technique, from Section 2, of combining formulas.

We now discuss how to proceed with an n^α-enumerator, $\alpha < 1$. We maintain a polynomially wide band as we prune down the tree. Suppose we are given m Boolean formulas $f_1, f_2, ..., f_m$ on variables $x_1, x_2, ..., x_n$. We first make all the variables distinct among different f_i's. This blows up the size of each formula by a factor of at most $1 + \log m$. Second, we choose m new variables $z_1, z_2, ..., z_m$ each of size $O(\log m)$, and combine the formulas $f_1, f_2, ..., f_m$ via the straightforward generalization of Eq. (1). Let F be the resulting superformula.

We wish to bound the size of F in terms of the sizes of the f_i's. Let N be a bound on the sizes of each f_i, $|f_i| \leqslant N$. We conclude, upon examining our combination formula, that the size of F is bounded above by

$$B(m, N) = 2mN(1 + \log m) + O(m \log m). \tag{2}$$

Note that for large N, $(B(2N^t + 1, N))^\alpha < N^t$, if $\alpha < 1$ and $t > 2\alpha/(1-\alpha)$. Hence, for an n^α-enumerator, $\alpha < 1$, we can maintain a bushy tree of width N^t as we carry out the tree pruning.

4. An n^k-for-One Result: $\mathrm{NP}^{\#\mathrm{P}} = \mathrm{NP}^{\#\mathrm{P}[1]}$

The recently developed theory of terseness asks if many queries to an oracle are more powerful than one query (Amir and Gasarch, 1988;

Goldsmith *et al.*, 1987; Beigel *et al.*, 1987). Kadin has proven that if for some k, $P^{NP[k]} = P^{NP[k+1]}$, then the polynomial hierarchy collapses (Kadin, 1987, 1988). On the other hand, it is easy to see that $NP^{NP} = NP^{NP[1]}$. We show here that $NP^{\#P} = NP^{\#P[1]}$. Our proof exploits the combination of formulas developed in Lemma 2.1, query postponement (Furst *et al.*, 1984), and nondeterministic guessing. This result may be viewed as an n^k-for-one "non-NP-terseness" result.

THEOREM 4.1. $NP^{\#P} = NP^{\#P[1]}$.

Proof. Given an $NP^{\#P}$ machine, we make a new NP machine that simulates the original one, except each time an oracle call is made in the original machine, our new machine nondeterministically guesses an answer that the oracle might give. Since $\#P$ returns a numerical value, a path chooses one out of an exponential number of plausible answers. Each leaf of our new tree combines all the $\#P$ queries along its path into a single superformula (using a straightforward polynomial-to-one formula version of Lemma 2.1) and queries $\#P$ once about the superformula. The leaf then accepts only if its guesses of formula values were all correct and the path of the original NP machine that the leaf has followed accepts. Q.E.D.

Since the construction in the above proof ensures that on every input the number of accepting paths of the new NP machine is identical to the number of accepting paths of the original NP machine, the proof also applies to path-bounded versions of NP. In particular, the proof applies to Valiant's unique polynomial time, UP, and Allender's class FewNP—classes that are closely related to cryptography and one-way functions (Valiant, 1976; Allender, 1986; Grollmann and Selman, 1988).

UP is the class of languages accepted by nondeterministic polynomial-time Turing machines that never have more than one accepting computation path. FewNP is the class of languages accepted by nondeterministic polynomial-time Turing machines that never have more than a polynomial number of accepting computation paths. Relativizations are defined in the natural way, e.g., $UP^{\mathscr{C}} = \{L \mid$ there is an NP machine N_i and a $B \in \mathscr{C}$ such that $L = L(N_i^B)$ and for every x, $N_i^B(x)$ has at most one accepting path$\}$.

COROLLARY 4.2. 1. $UP^{\#P} = UP^{\#P[1]}$.

2. $FewNP^{\#P} = FewNP^{\#P[1]}$.

Another use of formula combination apears in Cai and Hemachandra (1987), which shows that parity polynomial time (Papadimitriou and Zachos, 1983) is closed under bounded truth-table reductions and contains FewNP (Allender, 1986).

5. Open Questions and Conclusions

We have seen that if #SAT can be polynomial-time n^{α}-enumerated, $\alpha < 1$, many complexity classes collapse. One wonders if polynomial-time n^{α}-enumerability, for arbitrary α, also has consequences. We present a related oracle result. The following theorem is easily proved with a modification of the relativization techniques of Stockmeyer (1985).

Theorem 5.1. *There is a relativized world in which no Δ_2^p function can, for any k, n^k-enumerate* #SAT.

We conjecture that even polynomial-time n^k-enumerability implies $P = P^{\#P}$. Though $NP^{\#P} = NP^{\#P[1]}$, it is an open question whether $P^{\#P} = P^{\#P[1]}$. Our evidence on the structural consequences of enumerating #P shows that the fabric of complexity theory is tightly woven. The conjecture that #P is hard to enumerate is closely tied to the $P = ?NP$ question.

Acknowledgments

We are indebted to Professor Juris Hartmanis for valuable advice and encouragement. We thank Larry Stockmeyer and an anonymous referee for many helpful comments.

References

Amir, A., and Gasarch, W. (1988), Polynomial terse sets, *Inform. and Comput.* **77**, 37–56.

Allender, E. (1986), The complexity of sparse sets in *P*, *in* "Proceedings, 1st Structure in Complexity Theory Conference," Lecture Notes in Computer Science Vol. 223, pp. 1–11, Springer-Verlag, New York/Berlin.

Angluin, D. (1980), On counting problems and the polynomial-time hierarchy, *Theoret. Comput. Sci.* **12**, 161–173.

Babai, L. (1987), A random oracle separates PSPACE from the polynomial hierarchy, *Inform. Process. Lett.* **26**, No. 1, 51–53.

Beigel, R., Gasarch, W., Gill, J., and Owings, J. (1987), "Terse, Superterse, and Verbose Sets," Technical Report TR-1806, University of Maryland, Department of Computer Science, College Park, MD.

Cai, J. (1986), With probability one, a random oracle separates PSPACE from the polynomial-time hierarchy, *in* "18th ACM Symposium on Theory of Computing, pp. 21–29.

Cai, J., and Hemachandra, L. (1987), "On the Power of Parity Polynomial Time," Technical Report CUCS 274–87, Columbia Computer Science Department, New York, December. (Appears in proceedings of STACS '89, Springer-Verlag, 1989.)

Cook, S. (1971), The complexity of theorem-proving procedures, *in "3rd ACM Symposium on Theory of Computing,"* pp. 151–158.

Furst, M., Saxe, J., and Sipser, M. (1984), Parity, circuits, and the polynomial-time hierarchy, *Math. Systems Theory* **17**, 13–27.

GILL, J. (1977), Computational complexity of probabilistic Turing machines, *SIAM J. Comput.* **6**, No. 4, 675–695.

GAREY, M., AND JOHNSON, D. (1979), "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman, San Francisco.

GOLDSMITH, J., JOSEPH, D., AND YOUNG, P. (1987), Self-reducible, *P*-selective, near-testable, and *P*-cheatable sets: The effect of internal structure on the complexity of a set, *in* "Proceedings, 2nd Structure in Complexity Theory Conference," pp. 50–59.

GROLLMANN, J., AND SELMAN, A. (1988), Complexity measures for public-key cryptosystems, *SIAM J. Comput.* **17**, 309–335.

HASTAD, J., (1986), Almost optimal lower bounds for small depths circuits, *in* "18th ACM Symposium on Theory of Computing," pp. 6–20.

HOPCROFT, J., AND ULLMAN, J. (1979), "Introduction to Automata Theory, Languages, and Computation," Addison–Wesley, Reading, MA.

KADIN, J. (1987), "Is One NP Question as Powerful as Two?" Technical Report TR 87–842, Cornell University, Department of Computer Science, June.

KADIN, J. (1988), The polynomial time hierarchy collapses if the boolean heirarchy collapses, *In* "Proceedings, 3rd Structure in Complexity Theory Conference," pp. 278–292, IEEE Computer Society Press, New York.

PAPADIMITRIOU, C., AND ZACHOS, S. (1983), Two remarks on the power of counting, *in* "Proceedings, 6th GI Conference on Theoretical Computer Science," Lecture Notes in Computer Science Vol. 145, pp. 269–276, Springer-Verlag, New York/Berlin.

SCHÖNING, U. (1986), "Complexity and Structure," Lecture Notes in Computer Science Vol. 211, Springer-Verlag, New York/Berlin.

SIMON, J. (1975), "On Some Central Problems in Computational Complexity," Ph.D. thesis, Cornell University.

SIMON, J. On the difference between one and many, *in* "Automata, Languages, and Programming (ICALP 1977), Lecture Notes in Computer Science Vol. 52, pp. 480–491, Springer-Verlag, New York/Berlin.

SMOLENSKY, R. (1987), Algebraic methods in the theory of lower bounds for boolean circuit complexity, *in* "19th ACM Symposium on Theory of Computing," pp. 77–82, ACM, New York.

STOCKMEYER, L. (1985), On approximation algorithms for #P, *SIAM J. Comput.* **14**, No. 4, 849–861.

VALIANT, L. (1976), The relative complexity of checking and evaluating, *Inform. Process. Lett.* **5**, 20–23.

VALIANT, L. (1979a), The complexity of computing the permanent, *Theoret. Comput. Scie.* **8**, 189–201.

VALIANT, L. (1979b), The complexity of enumeration and reliability problems, *SIAM J. Comput.* **8**, No. 3, 410–421.

WAGNER, K. "Some Observations on the Connection Between Counting and Recursion," Technical Report MIP-8611, Universität Passau, Fakultät für Mathematik und Informatik.

YAO, A. (1985), Separating the polynomial-time hierarchy by oracles, *in* "Proceedings, 26th IEEE Symposium on Foundations of Computer Science," pp. 1–10.