JOURNAL OF COMPUTER AND SYSTEM SCIENCES 39, 282–298 (1989)

$P^{NP[O(\log n)]}$ and Sparse Turing-Complete Sets for NP*

JIM KADIN†

Department of Computer Science, Cornell University, Ithaca, New York 14853

Received August 28, 1987; revised October 16, 1988

$P^{NP[O(\log n)]}$ is the class of languages recognizable by deterministic polynomial time machines that make $O(\log n)$ queries to an oracle for NP. Our main result is that if there exists a sparse set $S \in NP$ such that co-$NP \subseteq NP^S$, then the polynomial hierarchy (PH) is contained in $P^{NP[O(\log n)]}$. Thus if there exists a sparse $\leqslant_T^P$-complete set for NP, $PH \subseteq P^{NP[O(\log n)]}$. We show that this collapse is optimal by showing for any function $f(n)$ that is $o(\log n)$, there exists a relativized world where NP has a sparse $\leqslant_T^P$-complete set and yet $P^{NP[O(\log n)]} \not\subseteq P^{NP[f(n)]}$. We also discuss complete problems for $P^{NP[O(\log n)]}$ and show languages related to the optimum solution size of Clique and K-SAT are $\leqslant_m^P$-complete. In related research, we investigate when the class of languages $\leqslant_m^P$-reducible to a set C equals $P^{C[O(\log n)]}$. We obtain results that allow us to prove that if D^P is closed under complementation, then $P^{NP[O(\log n)]} = D^P$.

1. INTRODUCTION

The class of languages accepted by deterministic polynomial time machines that make $O(\log n)$ queries to an oracle for NP is a subclass of Δ_2^P (P^{NP}). We call this class $P^{NP[O(\log n)]}$. This class was first considered by Papadimitriou and Zachos in [22]. Many of the languages related to optimum solution sizes of NP optimization problems are members of $P^{NP[O(\log n)]}$. UOCLIQUE, the set of graphs with one clique containing more vertices than all the other cliques of the graph, is one example of such a language. A binary search using $O(\log n)$ queries can determine the maximum clique size; one more query can determine uniqueness of the optimum clique. $P^{NP[O(\log n)]}$ is the class that captures this technique of binary search over polynomially many NP questions.

This paper presents results concerning $P^{NP[O(\log n)]}$. Our main result relates $P^{NP[O(\log n)]}$ to the study of sparse oracles for NP. We prove that if co-$NP \subseteq NP^S$, where $S \in NP$ and sparse, then the polynomial time hierarchy (PH) collapses to $P^{NP[O(\log n)]}$. As a corollary, we can conclude that if NP has a sparse $\leqslant_T^P$-complete set, then $PH \subseteq P^{NP[O(\log n)]}$. This sharpens Mahaney's result that the existence of a sparse $\leqslant_T^P$-complete set for NP implies $PH \subseteq P^{NP}$ [19].

* Presented at the "Second Annual IEEE Computer Society Symposium on Structure in Complexity Theory held June 16–19, 1987, at Cornell University, Ithaca, New York.

† Supported in part by NSF Research Grant DCR-8520597. Author's current address: Dept. of Computer Science, Neville Hall, University of Maine, Orono, ME 04469.

0022-0000/89 $3.00

This result is somewhat surprising since we show that the languages in the PH can be accepted without enumerating the strings in the sparse oracle—which would almost certainly require polynomially many queries. Instead, we need only compute the census function of the sparse oracle, and this can be done by binary search with $O(\log n)$ queries.

$P^{NP[O(\log n)]}$ turns out to be a class with many different natural definitions. Hemachandra, using essentially the same census function technique as we use in this paper, has shown the surprising result that $P^{NP[O(\log n)]}$ equals the class of languages polynomial time truth-table reducible to SAT [9]. Equivalently, it is the class of languages accepted by deterministic polynomial time machines that make *parallel* queries to an oracle for SAT, that is, machines that write down their (polynomially many) queries before getting any answers back. A variety of other classes defined in different ways have also been shown to coincide with $P^{NP[O(\log n)]}$ [26, 4, see also 1]. The census function technique of this paper can also be used to prove many of these equivalences. These alternative characterizations include the class of languages deterministic log-space Turing reducible to NP and the classes of languages reducible to NP via various restricted truth-table reducibilities.

It is important to keep in mind that P^{NP} and $P^{NP[O(\log n)]}$ are classes of languages and not classes of functions. Krentel [16] has shown that for the corresponding function classes FP^{NP} and $FP^{NP[O(\log n)]}$,

$$FP^{NP} = FP^{NP[O(\log n)]} \Rightarrow P = NP.$$

While our result along with Krentel's seems temptingly close to comprising a proof that a sparse $\leqslant_T^P$-complete set for NP implies $P = NP$, the collapse of the language classes P^{NP} and $P^{NP[O(\log n)]}$ does not imply the collapse of the function classes. Krentel has shown that there are relativized worlds for which $P^{NP} = P^{NP[O(\log n)]}$ and $FP^{NP} \neq FP^{NP[O(\log n)]}$ [16].

We take this a step further and show that the collapse to $P^{NP[O(\log n)]}$ is optimal in the sense that there are relativized worlds where the collapse goes no lower. Specifically we show that for any function $f(n)$ that is $o(\log n)$, there exists a recursive oracle B such that NP^B has a sparse $\leqslant_T^P$-complete set (implying $PH^B \subseteq [P^B]^{NP^B[O(\log n)]}$), but $[P^B]^{NP^B[O(\log n)]} \not\subseteq [P^B]^{NP^B[f(n)]}$. Hence for results that relativize, we now know the limits of the collapse caused by the existence of sparse $\leqslant_T^P$-complete sets for NP.

Recall that for $\leqslant_m^P$-reducibility, Mahaney has shown that the existence of a sparse $\leqslant_m^P$-complete (or hard) set for NP implies that $P = NP$ [19]. Since both Mahaney's result and the collapse to $P^{NP[O(\log n)]}$ relativize, we have a clear distinction between the effects of a sparse $\leqslant_m^P$-complete set and a sparse $\leqslant_T^P$-complete set.

We also study $\leqslant_m^P$-complete sets for $P^{NP[O(\log n)]}$. We present two complete languages related to uniqueness of solutions to NP-complete optimization problems: UOCSAT and a variant of UOCLIQUE. UOCSAT is the language of CNF Boolean formulas with the property that all the assignments that satisfy the maximum number of clauses satisfy the same clauses.

In related research, we consider the question of which sets C have the property that $\mathrm{P}^{C[O(\log n)]}$ is equal to the class of languages that are polynomial time many–one reducible to C. We give a partial answer to this question by showing that sets that have "logical" combining functions have the property. Using this result, we show that if the class D^{P} is closed under complementation, then $\mathrm{P}^{\mathrm{NP}[O(\log n)]} = \mathrm{D}^{\mathrm{P}}$.

2. Preliminaries

We assume the reader is familiar with oracle machines, oracle computations, and the classes P, NP, and the PH [10, 24].

Definition. For any set C, NP^C is the class of languages accepted by nondeterministic polynomial time oracle machines using C as an oracle.

Definition. For any set C, P^C is the class of languages accepted by deterministic polynomial time oracle machines using C as an oracle.

$$L \leqslant_{\mathrm{T}}^{\mathrm{P}} C \Leftrightarrow L \in P^C.$$

Definition. $\mathrm{P}^{C[O(\log n)]}$ is the subset of P^C that is accepted by machines that make at most $O(\log n)$ queries for all inputs of size n (in this paper, all logs are taken base 2),

$$L \leqslant_{\mathrm{T}[O(\log n)]}^{\mathrm{P}} C \Leftrightarrow L \in \mathrm{P}^{C[O(\log n)]}.$$

Definition. For any function f, $\mathrm{P}^{C[f(n)]}$ is the subset of P^C that is accepted by machines that make at most $f(n)$ queries for all inputs of size n.

Definition. For any integer constant k, $\mathrm{P}^{C[k]}$ is the subset of P^C that is accepted by machines that make at most k queries for all inputs.

Definition. For any set C, $\mathrm{m}-1[C]$ is the class of languages that are polynomial time many–one reducible to C,

$$L \leqslant_{\mathrm{m}}^{\mathrm{P}} C \Leftrightarrow L \in \mathrm{m}-1[C].$$

Since queries to an oracle for any NP language can be translated into queries for a SAT oracle, $\mathrm{P}^{\mathrm{NP}} = \mathrm{P}^{\mathrm{SAT}}$. We use these terms as synonyms. We also use the terms $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ and $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$ as synonyms.

For any finite set C, we write $\| C \|$ for the cardinality of C.

For any string x, we write $|x|$ for the length of x.

For any set of strings C, $C^{=n}$ and $C^{\leqslant n}$ are the sets of strings in C of length n and of length less than or equal to n, respectively.

For any set of strings C, the *census function* of C, $\text{Census}_C(n)$, is the number of strings in C of length less than or equal to n. A set is *sparse* if its census function is bounded by a polynomial.

For any two sets B and C, the *disjoint union* of B and C, written $B \oplus C$, is the set $\{1x \mid x \in B\} \cup \{0y \mid y \in C\}$.

If a language L is in $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$, then there exists a constant k and a deterministic polynomial time oracle Turing machine M such that $\mathrm{L}(M^{\mathrm{SAT}}) = L$, and for all inputs x and oracles C, $M^C(x)$ makes at most $k \log |x|$ queries. Given x, we can map out M's query behavior on x in polynomial time—without access to any oracle. We can simulate M on x trying both possible answers to each query and see whether M accepts or rejects with each sequence of query answers.

The query behavior of M on x can be represented by a *query tree*. A query tree is a binary tree with query strings for internal nodes and leaves that are labeled either ACCEPT or REJECT. The left branch from a node, q, leads to a node that describes M's behavior if the oracle answer to q is "no." Thus if the next query M would ask is q', the left child of q would be labeled q'. If an answer of "no" to q would make M reject or accept without any more queries, the left child of q would be a leaf labeled accordingly. The right child of q similarly describes the behavior of M if the answer to q is "yes." Since M asks only $O(\log n)$ queries, the height of the tree is $O(\log n)$, and the number of nodes is bounded by a polynomial in the length of x.

We say a path from the root to a leaf is an *accepting path* if the leaf is labeled ACCEPT. A path is a *rejecting path* if it ends at a leaf labeled REJECT. There may be many accepting paths and many rejecting paths in the tree for $M(x)$, but there is only one path from the root to a leaf for which all the queries are answered correctly (relative to a particular oracle). This path is called the *valid path*. $M^{\mathrm{SAT}}(x)$ accepts if and only if the valid path (relative to SAT) is an accepting path.

From the preceding discussion, it should be clear that the following theorem is true.

THEOREM 2.1. *The set*

$$T_{\log} \stackrel{\text{def}}{=} \{T\,\#^n \mid T \text{ is a query tree of SAT queries of height} \leqslant \log n, \text{ and the valid path of } T \text{ is an accepting path}\}$$

is $\leqslant_{\mathrm{m}}^{\mathrm{P}}$*-complete for* $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$.

3. SPARSENESS RESULTS

Much research has been focused on questions about sparse sets and NP. One of the major areas of concern is whether NP or co-NP is reducible to a sparse set under any of the standard polynomial time reducibilities. For Turing reducibilities, the results in this area show that the existence of a sparse Turing-hard set implies

that the PH collapses. The depth of the collapse varies depending on the assumptions made about the complexity of the sparse set and the power of the Turing reducibility (deterministic or nondeterministic).

For nondeterministic Turing reducibility, a result by Yap implies that if there exists a sparse S with co-$\mathrm{NP} \subseteq \mathrm{NP}^S$, then $\mathrm{PH} \subseteq \Sigma_3^{\mathrm{P}} \cap \Pi_3^{\mathrm{P}}$ [28].

Turning to deterministic Turing reducibility, Meyer made an important connection by showing that there exists a sparse S such that $\mathrm{NP} \subseteq \mathrm{P}^S$ if and only if there exist polynomial size circuits for every language in NP. This result appears in [2].

Using the circuit formulation, Karp, Lipton, and Sipser showed that if there exists a sparse S with $\mathrm{NP} \subseteq \mathrm{P}^S$, then $\mathrm{PH} \subseteq \Sigma_2^{\mathrm{P}} \cap \Pi_2^{\mathrm{P}}$ [15].

Mahaney took things a step further by proving that if the sparse oracle is itself a set in NP, then the PH collapses down to Δ_2^{P} ($\mathrm{P}^{\mathrm{SAT}}$). His proof involves showing that a $\mathrm{P}^{\mathrm{SAT}}$ machine can actually enumerate (write down) the strings in the oracle set [19].

Long generalized Mahaney's result by showing that if the oracle set is anywhere in Δ_2^{P}, the the collapse occurs down to Δ_2^{P}. Long's theorem relies on the fact that if there exists a sparse oracle in Δ_2^{P}, then all the relevant strings in the oracle up to a certain length can be enumerated by a $\mathrm{P}^{\mathrm{SAT}}$ machine [18].

Hence both Long's and Mahaney's results depend upon enumerating the strings of the oracle set with algorithms that make polynomially many queries.

We show that if there is a sparse $S \in \mathrm{NP}$ such that co-$\mathrm{NP} \subseteq \mathrm{NP}^S$ (which is a weaker assumption than $\mathrm{NP} \subseteq \mathrm{P}^S$), then there is no need to enumerate the oracle in order to prove that the PH collapses. Instead, the census function of the sparse oracle is all that is needed, and that can be computed—by binary search—with $O(\log n)$ queries. Therefore the PH collapses to $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$.

The proof uses two applications of a technique called *oracle replacement*. Let N_1 be an NP^B machine. If we have two NP^C machines, one that accepts B and one that accepts $\bar{B}$, then we can construct an NP^C machine N_2 such that

$$\mathrm{L}(N_1^B) = \mathrm{L}(N_2^C).$$

N_2^C simulates N_1^B. When N_1 would query B, N_2 runs the B and $\bar{B}$ acceptors on the query string to determine the oracle answer. Thus we can replace the oracle for B by an oracle for C. This technique is used implicitly in the sparse oracle results mentioned above.

THEOREM 3.1. *If there exists a sparse set* $S \in \mathrm{NP}$ *such that* co-$\mathrm{NP} \subseteq \mathrm{NP}^S$, *then* $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{SAT}[O(\log n)]}$.

Proof. Suppose $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$, where S is sparse. We will show $\mathrm{NP}^{\mathrm{SAT}} \subseteq \mathrm{P}^{\mathrm{SAT}[O(\log n)]}$; this implies $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{SAT}[O(\log n)]}$.

Let $L = \mathrm{L}(N_0^{\mathrm{SAT}})$ be any language in $\mathrm{NP}^{\mathrm{SAT}}$. The essence of our proof is that a $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ machine can use $O(\log n)$ queries to compute the census function of S, and this census value can be used to generate an NP machine that accepts an initial segment of L. The $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ machine can then tell if a given string is in L by

making one more query to determine if the generated NP machine accepts the string.

First, the $P^{SAT[O(\log n)]}$ machine uses the oracle replacement technique to construct an NP^S machine N_1 such that $L = (N_1^S)$. N_1 is just N_0 with its SAT oracle "replaced" by an NP acceptor for SAT and an NP^S acceptor for $\overline{SAT}$.

Then using oracle replacement again, a $P^{SAT[O(\log n)]}$ machine can generate an NP machine, N_2, that accepts an input x if and only if $x \in L(N_1^S)$. $N_1^S(x)$ only asks queries whose lengths are bounded by a polynomial in the length of x. The $P^{SAT[O(\log n)]}$ machine will produce an NP machine N_{no}, described below, which accepts strings in S up to the length of the queries made by $N_1^S(x)$. Since $S \in NP$, there exists a fixed NP machine, N_{yes}, that accepts S. Then using N_{no} and N_{yes} to "replace" the S oracle, the $P^{SAT[O(\log n)]}$ machine can convert N_1^S into the desired NP machine, N_2, such that

$$N_2(x) \text{ accepts} \Leftrightarrow N_1^S(x) \text{ accepts.}$$

The $P^{SAT[O(\log n)]}$ machine can tell if $N_2(x)$ accepts with one query.

N_{no} is the NP machine that accepts the pseudo-complement of S (the pseudo-complement was an important concept used by Mahaney in [19]). On input $(0^i, 1^j, x)$, N_{no} guesses i strings of length at most j, verifies that the strings are in S, and accepts if x is not one of the guessed strings. If $|x| \leqslant j$, and $i = \text{Census}_S(j)$, then N_{no} accepts $(0^i, 1^j, x)$ if and only if $x \in \bar{S}$.

Hence the $P^{SAT[O(\log n)]}$ machine need only compute a polynomial bound on the length of the queries $N_1^S(x)$ can make, call that number m, and then compute $\text{Census}_S(m)$ to be able to use N_{no} as desired. $\text{Census}_S(m)$ can be computed by a binary search using $O(\log |x|)$ queries. These queries take the form

$$\text{does } N_{\text{census}}(1^m, 0^k) \text{ accept?,}$$

where N_{census} on input $(1^m, 0^k)$ guesses k strings of length at most m and accepts if they are all in S. Thus N_{census} accepts if and only if $\text{Census}_S(m) \geqslant k$. The binary search ranges over $0 \leqslant k \leqslant p(m)$, where $p(\cdot)$ is a polynomial that bounds Census_S. ∎

The following corollary sharpens Mahaney's result and reasserts the distinction between sparse oracles inside NP and oracles outside NP that was blurred by Long's result.

COROLLARY 3.2. *If there exists a sparse* $\leqslant_T^P$*-complete set for NP, then* $PH \subseteq P^{SAT[O(\log n)]}$.

Observe that this proof relativizes easily. For any oracle B, if NP^B contains a sparse set S such that $\text{co-NP}^B \subseteq [NP^B]^S$, then $PH^B \subseteq [P^B]^{NP^B[O(\log n)]}$. Here $[NP^B]^S \stackrel{\text{def}}{=} NP^{B \oplus S}$, and $[P^B]^{NP^B[O(\log n)]}$ is the class of languages accepted by machines that make polynomially many queries to B and $O(\log n)$ queries to a language in NP^B.

Another aspect of this proof is that it ties the depth of the collapse directly to the density of the sparse oracle. The sparser the oracle is, the further down the PH falls. For instance, if S contains at most $\log n$ strings of length n, then $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{NP}[O(\log \log n)]}$.

Using $O(\log n)$ queries to compute how many of something and then passing that number to an NP machine in one final query is an important technique that finds its way into almost every aspect of the study of $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$. In particular, Hemachandra used it to prove that $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ contains $\mathrm{P}^{\mathrm{NP}\|}$, the class of languages recognizable by deterministic polynomial time machines that make (polynomially many) parallel queries to an NP oracle [9].

This technique also yields easy proofs that $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ has many other natural characterizations. Let TT[NP] be the languages polynomial time truth-table reducible to NP, where the truth-tables are expressed as Boolean circuits. This type of truth-table reducibility is equivalent to polynomial time Turing reducibility where the queries must be made in parallel [17]. Therefore $\mathrm{TT[NP]} = \mathrm{P}^{\mathrm{NP}\|}$. A restricted truth-table reducibility is *Boolean formula* reducibility: Bf[NP] is the class of languages polynomial time truth-table reducible to NP, where the truth-tables are expressed as Boolean formulas [25]. It is not hard to prove that

$$\mathrm{P}^{\mathrm{NP}[O(\log n)]} \subseteq \mathrm{Bf[NP]} \subseteq \mathrm{TT[NP]} = \mathrm{P}^{\mathrm{NP}\|}.$$

(The only inclusion that is not trivial is the first one; it follows from the fact that query trees can be encoded as Boolean formulas. See Lemma 5.1 for an example encoding.) Hemachandra's result then implies the equality of all these classes. Buss and Hay [4] and Wagner [26] give direct proofs that $\mathrm{Bf[NP]} = \mathrm{TT[NP]}$. Related results are proved by Beigel [1]. The binary search technique can also be used to show that $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ equals the class of languages deterministic log-space Turing reducible to SAT, a result due to Wagner [26] (he proves a variety of other equivalences in this paper).

Returning our attention to Corollary 3.2, it seems that the value of the census function is precisely the information that a $\mathrm{P}^{\mathrm{SAT}}$ machine needs to recognize $\mathrm{NP}^{\mathrm{SAT}}$ languages. We use this idea in the proof of our next result. We show that there exist relativized worlds where NP has sparse $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-complete sets, but the PH collapses no further than $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$. Therefore our previous result is in some sense optimal.

THEOREM 3.3. *For any function $f(n)$ that is $o(\log n)$, there exists a set B for which NP^B has a sparse $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-complete set and*

$$[\mathrm{P}^B]^{\mathrm{NP}^B[O(\log n)]} \not\subseteq [\mathrm{P}^B]^{\mathrm{NP}^B[f(n)]}.$$

Proof. The basic idea is to make sure that there is a sparse set in NP^B whose census function cannot be computed with only $f(n)$ queries. We will actually prove a slightly stronger claim. We will construct B by diagonalizing over $[\mathrm{P}^B]^{\mathrm{NP}^B[\log n]}$ machines to show

$$[\mathrm{P}^B]^{\mathrm{N}^B[O(\log n)]} \not\subseteq [\mathrm{P}^B]^{\mathrm{NP}^B[\log n]}.$$

Then since exactly $\log n$ queries is not enough to recognize all the languages in $[\mathrm{P}^B]^{\mathrm{NP}^B[O(\log n)]}$, $f(n)$ is also not enough.

B will be of the form $\mathrm{QBF} \oplus A$, where A will be sparse. QBF, the set of quantified Boolean formulas, is well known to be $\leqslant_m^{\mathrm{P}}$-complete for PSPACE [23]. Consider

$$\mathrm{Prefix}(A) \stackrel{\mathrm{def}}{=} \{ y \#^m \mid \exists x \in A \text{ with } y \text{ a prefix of } x, \text{ and } |x| = |y| + m \}.$$

$\mathrm{Prefix}(A)$ is in $\mathrm{NP}^{\mathrm{QBF} \oplus A}$ and is sparse if A is sparse. We will show that $\mathrm{Prefix}(A)$ is $\leqslant_T^{\mathrm{P}}$-complete for $\mathrm{N}^{\mathrm{QBF} \oplus A}$.

Suppose we are given some $\mathrm{NP}^{\mathrm{QBF} \oplus A}$ machine and an input string x. Using an oracle for $\mathrm{Prefix}(A)$, a deterministic polynomial time machine can enumerate the strings in A up to the length of the queries the NP machine can make. Given the list of strings, x, and the NP machine, a PSPACE machine can determine if the NP machine accepts x when using oracle $\mathrm{QBF} \oplus A$. Thus a polynomial time machine can tell if the NP machine accepts x with one query to QBF. Therefore as long as A is sparse, $\mathrm{Prefix}(A)$ will be $\leqslant_T^{\mathrm{P}}$-complete for $\mathrm{NP}^{\mathrm{QBF} \oplus A}$; that is,

$$\mathrm{NP}^{\mathrm{QBF} \oplus A} \subseteq [\mathrm{P}^{\mathrm{QBF} \oplus A}]^{\mathrm{Prefix}(A)}.$$

So by Theorem 3.1,

$$\mathrm{PH}^{\mathrm{QBF} \oplus A} \subseteq [\mathrm{P}^{\mathrm{QBF} \oplus A}]^{\mathrm{NP}^{\mathrm{QBF} \oplus A}[O(\log n)]}.$$

The rest of the proof involves showing how to define A so that it is sparse and yet

$$[\mathrm{P}^{\mathrm{QBF} \oplus A}]^{\mathrm{NP}^{\mathrm{QBF} \oplus A}[O(\log n)]} \not\subseteq [\mathrm{P}^{QBF \oplus A}]^{\mathrm{NP}^{\mathrm{QBF} \oplus A}[\log n]}.$$

For notational convenience, we will write B for $\mathrm{QBF} \oplus A$.

A will be such that $\forall n$, $\mathrm{Census}_A(n) \leqslant n^3$. Let

$$L_{\mathrm{odd}} \stackrel{\mathrm{def}}{=} \{ 1^n \mid \mathrm{Census}_A(n) \text{ is odd} \}.$$

$L_{\mathrm{odd}} \in [\mathrm{P}^B]^{\mathrm{NP}^B[O(\log n)]}$, since $\mathrm{Census}_A(n)$ can be computed with $O(\log n)$ queries to NP^B on input 1^n.

We will diagonalize over all $[\mathrm{P}^B]^{\mathrm{NP}^B[\log n]}$ machines to prevent L_{odd} from being in $[\mathrm{P}^B]^{\mathrm{NP}^B[\log n]}$.

A will be constructed in stages. At stage i, we will put strings into A to thwart M_i, the ith $[\mathrm{P}^B]^{\mathrm{NP}^B[\log n]}$ machine. A will consist of all the strings put into A during any stage.

Stage i. Let $g(n)$ be a polynomial bound on the running time of M_i. We will pick an integer n_i and make 1^{n_i} a witness that

$$\mathrm{L}(M_i^{B \oplus \mathrm{NP}^B[\log n]}) \neq L_{\mathrm{odd}}.$$

The n_i we pick must be big enough such that:

1. None of the previous diagonalization steps dealt with strings of length n_i (so no diagonalization steps interfere with each other).

2. $2^{n_i} > n_i^2 + n_i\, g(n_i) + n_i^2 g(n_i)$ (so there are lots of potential strings of length n_i to put into A).

We will run $M_i(1^{n_i})$ and force it to accept or reject and make $\text{Census}_A(n_i)$ even or odd to force $\text{L}(M_i^{B \oplus \text{NP}^B[\log n]}) \neq L_{\text{odd}}$. In a nutshell, our method will be to see what $M_i(1^{n_i})$ does with the strings already put into A. If it accepts and $\| A^{\leqslant n_i} \|$ is even or it rejects and $\| A^{\leqslant n_i} \|$ is odd, then 1^{n_i} is our witness, and we are done with M_i. Otherwise, we add a string of length n_i to A and see what $M_i(1^{n_i})$ does relative to this new definition of A. If this new string in A does not change M_i's acceptance behavior, then we are done because we have changed $\| A^{\leqslant n_i} \|$ from even to odd or vice versa. The strings added to A will be carefully chosen so that we can argue that the acceptance behavior of $M_i(1^{n_i})$ will only be able to change (i.e., flip flop) n_i^2 times. Hence we can thwart M_i while keeping $\text{Census}_A(n) \leqslant n^3$.

$M_i(1^{n_i})$ gets to make at most $g(n_i)$ queries to B and at most $\log n_i$ queries to NP^B. We assume without loss of generality that the queries to NP^B are to the set

$$U_{\text{NP}} \stackrel{\text{def}}{=} \{ \langle N, x, \text{pad} \rangle \mid N \text{ is an NP oracle machine, and } N^B(x) \text{ accepts in } |x| + |\text{pad}| \text{ steps} \}$$

which is $\leqslant_{\text{m}}^{\text{P}}$-complete for NP^B.

Consider T_{M_i}, the query tree for $M_i(1^{n_i})$ generated as follows:

1. For any query made directly to QBF, answer it correctly.

2. For any query made directly to A, answer it "yes" if the string has already been put into A and "no" otherwise.

3. For any query to U_{NP}, try both possible answers.

Since M_i can only make $\log n_i$ queries to U_{NP}, the number of splits along any path through T_{M_i} is at most $\log n_i$. Therefore the number of leaves is no more than n_i, and the number of U_{NP} queries in the tree is no more than $n_i \log n_i$, which is less than n_i^2.

We will run through the tree several times adding strings of length n_i to A and building another list of strings of length n_i, A_0. The strings in A_0 will not be in A and will never be put into A during later passes through the tree. Throughout the cycles, we will be sure to keep

$$\| A^{=n_i} \| < n_i^2$$

$$\| A_0 \| \leqslant n_i\, g(n_i) + n_i^2\, g(n_i).$$

Hence we will keep $\text{Census}_A(n) \leqslant n^3$, and at any time during the cycles, we will be able to find a string of length n_i which is not in A_0 that we can add to $A^{=n_i}$.

As we go through the tree we will mark queries to U_{NP} as "frozen" or "unfrozen." A frozen query is one whose answer is "yes" with the current status of A and will remain "yes" for any extensions we will make to A.

Start out with $A^{=n_i}$ and A_0 empty. Mark all U_{NP} queries as "unfrozen."

Initial pass. Run through the tree and add to A_0 any strings of length n_i that were queried directly to A (to which we answered "no"). This will make those "no" answers correct. There are at most n_i different paths through the tree and at most $g(n_i)$ queries to A on each path, so this adds only $n_i\, g(n_i)$ strings to A_0.

Cycle step. Run through the tree and look at each query to U_{NP} of the form $\langle N_j, x_j, \mathrm{pad}_j\rangle$, that has not been marked as "frozen." Consider each computation path of $N_j^B(x_j)$ using correct answers for the queries to QBF and the strings already put into A to answer the queries to A.

If there is no sequence of guesses that causes acceptance, then

$$\langle N_j, x_j, \mathrm{pad}_j\rangle \notin U_{\mathrm{NP}}$$

with our current status of A. At least until we change A, the answer to this U_{NP} query is "no."

If there is some sequence of guesses that causes acceptance, then

$$\langle N_j, x_j, \mathrm{pad}_j\rangle \in U_{\mathrm{NP}},$$

and the answer to the query is "yes" with our current status of A. We will mark $\langle N_j, x_j, \mathrm{pad}_j\rangle$ as "frozen" and "freeze" some accepting path so that any future changes to A will not change the answer to this query from "yes" to "no." To freeze the accepting path, look at all the queries of length n_i to A along the path. Take all the strings that are answered "no" (i.e., the ones that have not yet been put into A) and put them into A_0. Since we never put a string in A_0 into A, we will never change the query answers along this path, so the answer to this U_{NP} question will remain "yes." M_i must write down $\langle N_j, x_j, \mathrm{pad}_j\rangle$ in order to make the query, thus $|x_j| + |\mathrm{pad}_j| \leqslant g(n_i)$. Then since N_j accepts in $|x_j| + |\mathrm{pad}_j|$ steps, there can be at most $g(n_i)$ queries along any computation path. So to freeze the accepting path, we have to add at most $g(n_i)$ strings to A_0.

Now look at the path through T_{M_i} with the queries answered correctly according to the current status of A. If this path ends with acceptance, and the number of strings in $A^{\leqslant n_i}$ is even, then 1^{n_i} is our witness, and we do not need to add any more strings to $A^{=n_i}$. Similarly, if this path is a rejecting path and $\| A^{\leqslant n_i} \|$ is odd, then we are done.

Otherwise, we have to change the number of strings in $A^{\leqslant n_i}$ from even to odd or vice versa, so we pick a new string of length n_i that is not in A_0 and add it to A. Then go back and do the cycle step again.

Since there are fewer than n_i^2 U_{NP} queries in the tree, n_i^2 is an upper bound on the number of nodes that can be frozen. Since freezing requires adding at most $g(n_i)$ strings to A_0, the total number of strings added for freezing purposes is less than

$n_i^2\, g(n_i)$. Therefore, including the strings added to A_0 during the initial pass, we have

$$\| A_0 \| \leqslant n_i\, g(n_i) + n_i^2\, g(n_i).$$

Similarly, the fact that there are fewer than $n_i^2\, U_{\mathrm{NP}}$ queries in the tree implies that we will succeed in forcing 1^{n_i} to be our witness in fewer than n_i^2 cycles. The following facts should make this clear:

1. Each time we add a string to $A^{=n_i}$, if the valid path through T_{M_i} does not change, then 1^{n_i} is our witness because we will have changed the oddness or evenness of $\| A^{\leqslant n_i} \|$.

2. Since we freeze "yes" answers to U_{NP}, if the valid path changes when we add a string to $A^{=n_i}$, it is because some U_{NP} query answer is changed from "no" to "yes."

3. There are fewer than $n_i^2\, U_{\mathrm{NP}}$ queries in the tree, and hence fewer than $n_i^2\, U_{\mathrm{NP}}$ queries can be switched from "no" to "yes."

Since we make fewer than n_i^2 cycles, and we add one string to $A^{=n_i}$ on each cycle, $\| A^{=n_i} \| \leqslant n_i^2$. Thus we know n_i was chosen large enough that we can always find another string not in A_0 to add to A. ∎

Immerman and Mahaney have proven a similar optimality result for the Karp, Lipton, Sipser collapse. They showed that there exists a relativized world where NP has polynomial size circuits, which implies the $\mathrm{PH} \subseteq \Sigma_2^{\mathrm{P}} \cap \Pi_2^{\mathrm{P}}$, and yet $\mathrm{NP} \neq \Sigma_2^{\mathrm{P}}$ [11]. Wilson and Heller have independently taken this work a step further by exhibiting relativized worlds where NP has polynomial size circuits and yet $\Delta_2^{\mathrm{P}} \neq \Sigma_2^{\mathrm{P}}$ [27, 7]. Therefore a proof that the existence of a sparse oracle for NP collapses the PH down to Δ_2^{P} would be a result that does not relativize.

It would be interesting to see if similar relativizations can be achieved for Long's result and Yap's result.

4. Complete Problems

We have shown that $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ captures the PH if there exists a sparse $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-complete set for NP because it is the class that embodies the technique of binary search over polynomially many NP questions. We have also pointed out that this technique can be used to show that $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ has many natural characterizations. Since the solution size of many NP optimization problems is bounded by a polynomial in the size of the problem representation, this same technique puts many problems related to solution sizes into $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$. For example, UOCLIQUE, the set of graphs with a unique maximum size clique, is in $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$. A host of other similar "unique optimum" languages are also easily seen to be in $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$, to name a few:

UOASAT (unique optimum assignment satisfiability) is the set of CNF formulas that have exactly one assignment that satisfies the maximum number of clauses, and all other assignments satisfy fewer clauses.

UOBTSP(k) (unique optimum bounded traveling salesperson problem) is the set of undirected graphs $G=(V, E)$ with edge costs no more than $\|V\|^k + k$ such that there is a unique optimum Hamiltonian circuit through the graph.

UOCOLORING (unique optimum graph coloring).

UOVCOVER (unique optimum vertex cover).

All these languages can be recognized by using $O(\log n)$ queries to NP to do a binary search to compute the optimum solution size and then making one more query to determine uniqueness.

Papadimitriou and Zachos observed that languages of this type are in $P^{NP[O(\log n)]}$, and they asked specifically if UOCLIQUE is $\leqslant_m^P$-complete [22]. This question became more interesting when Papadimitriou proved UOTSP is $\leqslant_m^P$-complete for P^{NP} [20]. Because of the possibility of huge edge costs, the cost of the maximum solution to a traveling salesperson problem can be exponential in the size of the problem representation. Thus a binary search to find the optimum solution cost can take polynomially many queries instead of logarithmically many queries.

Krentel observed that the optimum solution size of all these problems is what is important and that the uniqueness question is just one way to force a language recognizer to compute the solution size [16]. He showed the following two problems $\leqslant_m^P$-complete for $P^{NP[O(\log n)]}$:

SAT-MOD-k is the set of pairs $\langle F, \#^m \rangle$ such that F is a CNF formula, and the maximum number of simultaneously satisfiable clauses of F is equal to 0 mod m.

CLIQUE-MOD-k is the set of pairs $\langle G, \#^m \rangle$ such that G is an undirected graph whose maximum clique size is equal to 0 mod m.

A nice special case of these problems is ODDSAT and ODDCLIQUE "is the optimum solution size odd?" Wagner showed ODDSAT and ODDCLIQUE are $\leqslant_m^P$-complete for $P^{NP[O(\log n)]}$ (actually he proved that they are log-space many–one complete) [26]. He also characterizes other complete languages this way [25].

While these results show that the solution size is really the major necessary ingredient for completeness, the question of uniqueness is such a natural one that we pursue it further in this section.

In trying to prove that UOCLIQUE or UOASAT is complete for $P^{NP[O(\log n)]}$, we found that the uniqueness condition causes difficulties, but by weakening the definition of uniqueness, we could prove completeness.

DEFINITION. UOCSAT (unique optimum clause satisfiability) is the set of CNF formulas with the property that all the assignments that satisfy the maximum number of clauses satisfy the same set of clauses.

The uniqueness condition in UOASAT, that one assignment satisfy more clauses than all the others, is stronger than the condition that all the best solutions satisfy the same clauses.

THEOREM 4.1. UOCSAT *is* $\leqslant_m^P$*-complete for* $P^{SAT[O(\log n)]}$.

The proof is quite involved and can be found in [12]. The basic idea is to reduce $T_{\log}$, defined in Section 2, to UOCSAT by encoding query trees as CNF Boolean formulas. We start by taking each query q_i in the tree and reducing it to two formulas $q_{i,y}$ and $q_{i,n}$ such that

$$q_i \in \text{SAT} \Leftrightarrow q_{i,y} \in \text{UOCSAT}$$

and

$$q_i \notin \text{SAT} \Leftrightarrow q_{i,n} \in \text{UOCSAT}.$$

Next, each accepting path p_j in the tree is encoded as a formula F_{p_j}. F_{p_j} is basically the ANDing together of $q_{i,y}$'s and $q_{i,n}$'s that represent the queries and answers on path p_j. Thus $F_{p_j} \in$ UOCSAT if and only if p_j is the valid path. Finally we show that the F_{p_j}'s can be combined into a single CNF formula that will be in UOCSAT if and only if the original query tree has a valid accepting path.

The major difficulty in adapting this proof for UOASAT is that, while it is not hard to reduce $\overline{\text{SAT}}$ to UOASAT, it is not clear how to reduce SAT to UOASAT. UOASAT is very similar to USAT, the set of Boolean formulas that have exactly one satisfying assignment. USAT also has the property that $\overline{\text{SAT}}$ can easily be reduced to it, but it is not clear how to reduce SAT. Blass and Gurevich have shown that SAT $\leqslant_m^P$ USAT if and only if USAT is $\leqslant_m^P$-complete for D^P (see [21] for a definition of D^P). Furthermore, they have proven that D^P can be relativized so that the relativized versions of USAT can be made complete or incomplete for D^P [3]. This indicates that resolving whether or not USAT is $\leqslant_m^P$-complete for D^P is probably going to be very difficult. Since UOASAT and USAT are so similar, we believe that resolving whether or not UOASAT is $\leqslant_m^P$-complete for $P^{NP[O(\log n)]}$ is also going to be very difficult. An interesting open problem is whether we can prove the same type of relativization result for UOASAT and $P^{NP[O(\log n)]}$.

Returning to the clique problem, we can also relax the uniqueness condition here to achieve completeness.

DEFINITION. UOGCLIQUE (unique optimum grouped clique) is the set of undirected graphs whose vertices are partitioned into groups, with no edges between vertices in the same group, with the property that all the maximum cliques contain vertices from the same set of groups.

THEOREM 4.2. UOGCLIQUE *is* $\leqslant_m^P$*-complete for* $P^{NP[O(\log n)]}$.

Proof. The standard reduction from SAT to Clique in [10] reduces UOCSAT to UOGCLIQUE. ∎

The standard reduction also reduces UOASAT to UOCLIQUE, and so a proof that UOASAT is complete would also imply that UOCLIQUE is complete.

5. Many–One versus $P^{C[O(\log n)]}$

Understanding the relative power of the various polynomial time reducibilities is an important goal of complexity theory. The central questions in the field can usually be phrased in terms of reducibilities. For instance, NP $\neq$ co-NP $\Leftrightarrow \exists L$ such that $L \leqslant_T^P$ SAT and $L \not\leqslant_m^P$ SAT. Hence a complete characterization of the sets C for which $\leqslant_m^P$-reductions to C have the same power as $\leqslant_T^P$-reductions to C could be very useful.

In this section we consider $\leqslant_m^P$ and $\leqslant_{T[O(\log n)]}^P$. We observe that for sets C that have natural combining functions, $m-1[C] = P^{C[O(\log n)]}$. This observation leads to a surprising relationship between D^P and $P^{SAT[O(\log n)]}$.

A set has natural combining functions if there are polynomial time functions for combining membership questions about the set.

Definition. A set C has an AND function if there exists a polynomial time "any–ary" function $AND_C(\ )$ such that $\forall n, \forall x_1, ..., x_n$,

$$x_1 \in C \wedge \cdots \wedge x_n \in C \Leftrightarrow AND_C(x_1, ..., x_n) \in C.$$

Observe that a set C has an AND function if and only if $m-1[C]$ is closed under polynomial time conjunctive reducibility (conjunctive and disjunctive reducibilities are defined in [17]).

Definition. A set C has an OR function if there exists a polynomial time "any–ary" function $OR_C(\ )$ such that $\forall n, \forall x_1, ..., x_n$,

$$x_1 \in C \vee \cdots \vee x_n \in C \Leftrightarrow OR_C(x_1, ..., x_n) \in C.$$

A set C has an OR function if and only if $m-1[C]$ is closed under polynomial time disjunctive reducibility.

Definition. A set C has a NOT function if there exists a polynomial time function $NOT_C(\cdot)$ such that $x \notin C \Leftrightarrow NOT_C(x) \in C$.

That is, C has a NOT function if and only if $C \leqslant_m^P \bar{C}$. In other words, C has a NOT function if and only if $m-1[C]$ is closed under complementation.

Note that by De Morgan's laws, any set that has AND (OR) and NOT functions also has an OR (AND) function.

It is easy to see that SAT has AND and OR functions. Another easy observation is that if two sets are $\leqslant_m^P$-reducible to each other and the first set has an AND (OR) function, then the second set has an AND (OR) function too. Therefore all

the NP-complete sets have AND and OR functions. In other words, NP is closed under conjunctive and disjunctive reducibilities.

The next lemma states that for sets that have all three logical combining functions, $\leqslant^{\mathrm{P}}_{\mathrm{T}[O(\log n)]}$ is no more powerful than $\leqslant^{\mathrm{P}}_{\mathrm{m}}$ (this lemma was incorrectly stated in the preliminary version of this paper that appeared in [13]).

LEMMA 5.1. *For all sets C, if C has* AND, OR, *and* NOT *functions, then* $\mathrm{m}-1[C] = \mathrm{P}^{C[O(\log n)]}$.

Proof. Using C's AND, OR, and NOT functions, a query tree can be reduced to a single string that is in C if and only if the valid path of the tree is accepting.

For each accepting path p_i in the tree, let

$$X_i \stackrel{\text{def}}{=} \mathrm{AND}(x_1, ..., x_m, \mathrm{NOT}(y_1), ..., \mathrm{NOT}(y_l)),$$

where $x_1, ..., x_m$ are the queries answered "yes" on the path, and $y_1, ..., y_l$ are the queries answered "no." $X_i \in C \Leftrightarrow p_i$ is the valid path of the tree (relative to C). Therefore, $\mathrm{OR}(X_1, ..., X_k) \in C \Leftrightarrow$ one of the accepting paths $p_1, ..., p_k$ is valid. ∎

With this lemma we can show that if D^{P} is closed under complementation, then $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$ collapses to D^{P}. Recall the canonical $\leqslant^{\mathrm{P}}_{\mathrm{m}}$-complete language for D^{P} is SATUNSAT [21],

$$\mathrm{SATUNSAT} \stackrel{\text{def}}{=} \{(x, y) \mid x \in \mathrm{SAT} \text{ and } y \in \overline{\mathrm{SAT}}\}.$$

We will show that if D^{P} is closed under complementation, then SATUNSAT has AND, OR, and NOT functions. The existence of these functions will cause $\mathrm{P}^{\mathrm{SAT}[O(\log n)]} = \mathrm{D}^{\mathrm{P}}$.

THEOREM 5.2. *If* $\mathrm{D}^{\mathrm{P}} = \mathrm{co\text{-}D}^{\mathrm{P}}$, *then* $\mathrm{P}^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{D}^{\mathrm{P}}$.

Proof. First observe that, without any assumptions, SATUNSAT has an AND function:

$$\mathrm{AND}((x_1, y_1), ..., (x_k, y_k)) \stackrel{\text{def}}{=} \left(\bigwedge_{i=1}^{k} x_i, \bigvee_{i=1}^{k} y_i \right).$$

If $\mathrm{D}^{\mathrm{P}} = \mathrm{co\text{-}D}^{\mathrm{P}}$, then SATUNSAT also has a NOT function. So by De Morgan's laws, SATUNSAT has an OR function. Therefore by Lemma 5.1,

$$\mathrm{m}-1[\mathrm{SATUNSAT}] = \mathrm{P}^{\mathrm{SATUNSAT}[O(\log n)]}.$$

Then since $\mathrm{D}^{\mathrm{P}} = \mathrm{m}-1[\mathrm{SATUNSAT}]$, and $\mathrm{P}^{\mathrm{SATUNSAT}[O(\log n)]} = \mathrm{P}^{\mathrm{SAT}[O(\log n)]}$, $\mathrm{D}^{\mathrm{P}} = \mathrm{P}^{\mathrm{SAT}[O(\log n)]}$. ∎

COROLLARY 5.3. *If* $P^{SAT[1]} = P^{SAT[2]}$, *then* $P^{SAT[1]} = P^{SAT[O(\log n)]}$.

Proof. $P^{SAT[1]} \subseteq D^P \subseteq P^{SAT[2]}$. Therefore if $P^{SAT[1]} = P^{SAT[2]}$, then $P^{SAT[1]} = D^P$ which implies $D^P = \text{co-}D^P$. ∎

D^P and co-D^P make up the second level of the *Boolean hierarchy* (BH) [5]. $P^{SAT[1]}$ and $P^{SAT[2]}$ respectively make up the first and second levels of the *query hierarchy* (QH) [14]. The BH and QH intertwine to form a rich structure within $P^{SAT[O(\log n)]}$. It has been shown in [14] that if the BH or QH collapses, then the PH collapses to its third level. The results of this section tell us that if the QH and BH collapse at their bottom levels, then we also see a collapse just above these hierarchies within P^{NP}.

The proof of Theorem 5.2 does not seem to generalize to higher levels of the BH. Chang has pointed out that the existence of an AND or an OR function for complete languages at higher levels of the BH is enough by itself to collapse the BH and $P^{SAT[O(\log n)]}$ [6].

6. CONCLUSION

There is now strong evidence that $P^{NP[O(\log n)]}$ is an important class whose structure is related to the structure of D^P, P^{NP}, and NP itself. We have shown that $P^{NP[O(\log n)]}$ is closely tied to the question of the existence of sparse $\leq_T^P$-complete sets for NP. It is a natural breaking point since the existence of such a set collapses the PH to $P^{NP[O(\log n)]}$, but techniques that relativize will not prove any deeper collapse. $P^{NP[O(\log n)]}$ contains many natural languages and has natural complete languages. It is a surprisingly robust class with a wide variety of different yet equivalent definitions. Finally, it is interesting to note that the technique of using the NP oracle and binary search to compute census functions is a common thread throughout all these results.

One issue for further research is resolving whether UOCLIQUE and similar languages are $\leq_m^P$-complete for $P^{NP[O(\log n)]}$.

Another question to explore is what the collapse of P^{SAT} to $P^{SAT[O(\log n)]}$ would imply. It seems that such a collapse would say something about the relationship between deterministic polynomial time machines and NP. Hemachandra has proven some interesting results along these lines in [8].

ACKNOWLEDGMENTS

I am grateful to Professor Hartmanis for helpful discussions and support, and particularly for suggesting that Theorem 3.3 might be true. Thanks also to Steve Mitchell and Richard Chang for carefully reading versions of this paper. Discussions with Richard Chang and Richard Beigel were also very helpful in sorting out subtle entanglements.

REFERENCES

1. R. BEIGEL, Bounded queries to SAT and the Boolean hierarchy, *Theoret. Comput. Sci.* 1988, in press.
2. L. BERMAN AND J. HARTMANIS, On isomorphisms and density of NP and other complete sets, *SIAM J. Comput.* **6**, No. 2 (1977), 305–322.
3. A. BLASS AND Y. GUREVICH, On the unique satisfiability problem, *Inform. and Control* **55** (1982), 80–88.
4. S. BUSS AND L. HAY, On truth-table reducibility to SAT and the difference hierarchy over NP, *in* "Proceedings, 3rd Structure in Complexity Theory Conference," 1988, pp. 224–233.
5. J. CAI, T. GUNDERMANN, J. HARTMANIS, L. HEMACHANDRA, V. SEWELSON, K. WAGNER, AND G. WECHSUNG, The Boolean hierarchy I. Structural properties, *SIAM J. Comput.* **17** (1988).
6. R. CHANG, 1988, private communication.
7. H. HELLER, On relativized exponential and probabilistic complexity classes, *Inform. and Control* **71** (1986), 231–243.
8. L. A. HEMACHANDRA, "Can P and NP manufacture randomness?," Technical Report TR 86–795, Cornell Department of Computer Science, December 1986.
9. L. A. HEMACHANDRA, The strong exponential hierarchy collapses, *in* "Proceedings, 19th ACM Symposium on Theory of Computing," 1987, pp. 110–122.
10. J. HOPCROFT AND J. ULLMAN, "Introduction to Automata Theory, Languages, and Computation," Addison–Wesley, Reading, MA, 1979.
11. N. IMMERMAN AND S. MAHANEY, Relativizing relativized computations, *Theoret. Comput. Sci.* 1988.
12. J. KADIN, "Deterministic Polynomial Time with $O(\log n)$ Queries," Technical Report TR 86–771, Cornell Department of Computer Science, August 1986.
13. J. KADIN, $P^{NP[\log n]}$ and sparse Turing complete sets for NP, *in* "Proceedings, 2nd Structure in Complexity Theory Conference," 1987, pp. 33–40.
14. J. KADIN, The polynomial hierarchy collapses if the Boolean hierarchy collapses, *SIAM J. Comput.* **17** (1988).
15. R. KARP AND R. LIPTON, Turing machines that take advice, *Enseign. Math.* **28** (1982), 191–209.
16. M. KRENTEL, The complexity of optimization problems, *J. Comput. System Sci.* **36** (1988), 490–509.
17. R. LADNER, N. LYNCH, AND A. SELMAN, A comparison of polynomial time reducibilities, *Theoret. Comput. Sci.* **1**, No. 2 (1975), 103–124.
18. T. LONG, A note on sparse oracles for NP, *J. Comput. System Sci.* **24** (1982), 224–232.
19. S. MAHANEY, Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis, *J. Comput. System Sci.* **25**, No. 2 (1982), 130–143.
20. C. PAPADIMITRIOU, On the complexity of unique solutions, *J. Assoc. Comput. Mach.* **31**, No. 2 (1984), 392–400.
21. C. PAPADIMITRIOU AND M. YANNAKAKIS, The complexity of facets (and some facets of complexity), *J. Comput. System Sci.* **28**, No. 2 (1984), 244–259.
22. C. PAPADIMITRIOU AND S. ZACHOS, Two remarks on the power of counting, *in* "Sixth GI Conference on Theoretical Computer Science," pp. 269–275, Lecture Notes in Computer Science, Vol. 145, Springer-Verlag, New York/Berlin, 1983.
23. L. STOCKMEYER, "The Complexity of Decision Problems in Automata Theory and Logic," Technical Report MAC TR–133, MIT, 1974; Project MAC.
24. L. STOCKMEYER, The polynomial-time hierarchy, *Theoret. Comput. Sci.* **3** (1977), 1–22.
25. K. WAGNER, More complicated questions about maxima and minima, and some closures of NP, *in* "Proceedings, 13th International Colloquium on Automata, Languages, and Programming (ICALP)," pp. 434–443, Lecture Notes in Computer Science, Vol. 226, Springer-Verlag, New York/Berlin, 1986.
26. K. WAGNER, "Bounded Query Classes," Technical Report 157, University of Augsburg, October 1987.
27. C. WILSON, Relativized circuit complexity, *J. Comput. System Sci.* **31**, No. 2 (1985), 169–181.
28. C. YAP, Some consequences of non-uniform conditions on uniform classes, *Theoret. Comput. Sci.* **26** (1983), 287–300.