JOURNAL OF COMPUTER AND SYSTEM SCIENCES 24, 224–232 (1982)

A Note on Sparse Oracles for *NP**

TIMOTHY J. LONG

Department of Computer Science, New Mexico State University, Las Cruces, New Mexico 88003

Received December 8, 1980; revised December 22, 1981

We prove that if S is a sparse oracle for NP, then S is a sparse oracle for the polynomial-time hierarchy. Consequently, if NP has either a sparse or co-sparse $\leqslant_T^P$-complete set, then the polynomial-time hierarchy collapses to Δ_2^P. S. Mahaney (Proceedings, 21st IEEE Symposium, Foundations of Computer Science, 1980) had previously shown that if NP has a sparse $\leqslant_T^P$-complete set, then the polynomial-time hierarchy collapses to Δ_2^P. Our single construction yields Mahaney's result for sparse $\leqslant_T^P$-complete sets for NP and also the corresponding result for co-sparse $\leqslant_T^P$-complete sets for NP. Both results strengthen the earlier work of Karp, Lipton, and Sipser, Proceedings, Twelfth ACM Symposium Theory of Computing, 1980, who proved that if NP has a sparse $\leqslant_T^P$-complete set, then the polynomial-time hierarchy collapses to $\Sigma_2^P \cap \Pi_2^P$.

INTRODUCTION

In this paper we shall prove that if S is a sparse oracle for every set in NP, then S is a sparse oracle for every set in the polynomial-time hierarchy (PH). Consequently, if NP has either a sparse or co-sparse $\leqslant_T^P$-complete set, then PH collapses to Δ_2^P. It is not likely, therefore, that NP has such $\leqslant_T^P$-complete sets.

These results represent a technical improvement of earlier work done by Karp *et al.* [7]. They proved that if NP has polynomial-size circuits, then PH collapses to $\Sigma_2^P \cap \Pi_2^P$. It is known from the work of Meyer (appearing in [2]) that if NP has a sparse oracle, then NP has polynomial-size circuits. Thus, as a corollary to their work, Karp *et al.* concluded that if NP has a sparse $\leqslant_T^P$-complete set, then $\mathrm{PH} = \Sigma_2^P \cap \Pi_2^P$. Mahaney [9] strengthened this result by proving that if NP has a sparse $\leqslant_T^P$-complete set, then PH actually collapses to Δ_2^P. This last result leads to the question of wether $\mathrm{PH} = \Delta_2^P$ if NP has a co-sparse $\leqslant_T^P$-complete set. Our work shows that this is the case and establishes, from a single construction, both Mahaney's improvement and the companion to Mahaney's improvement of the Karp, *et al.* result.

All of this work can also be viewed as being related to a conjecture of Berman and Hartmanis [2] that NP-complete sets cannot be sparse unless $P = NP$. The first significant step towards establishing this conjecture was made by Berman [3] who

* This paper was written while the author was a member of the faculty of the Department of Computer and Information Science at The Ohio State University.

0022-0000/82/020224-09$02.00/0

proved that if *NP* has an *NP*-complete set in $\{0\}^*$, then $P = NP$. (Note that any subset of $\{0\}^*$ is both sparse and co-sparse.) Fortune [5] improved Berman's result by showing that if *NP* has a co-sparse *NP*-complete set then $P = NP$. (A proof of this also appears in Meyer and Paterson [10] along with other related results.) Mahaney [9] then settled the Berman–Hartmanis conjecture by proving that if *NP* has a sparse *NP*-complete set, then $P = NP$.

This paper is concerned with the implications of having a (co-)sparse $\leqslant_T^P$-complete set for *NP* as opposed to having a (co-)sparse $\leqslant_m^P$-complete set for *NP*. The known consequences of having a (co-)sparse $\leqslant_T^P$-complete set for *NP*, that $\mathrm{PH} = \Delta_2^P$, are not as strong as the $P = NP$ consequence of having a (co-)sparse $\leqslant_m^P$-complete set for *NP*. However, assuming the existence of a (co-)sparse $\leqslant_m^P$-complete set for *NP* is stronger than assuming the existence of a (co-)sparse $\leqslant_T^P$-complete set for *NP* since $\leqslant_T^P$-reducibility is more general than $\leqslant_m^P$-reducibility.

PRELIMINARIES

All sets are assumed to be over a fixed alphabet Σ, of size greater than one, with λ denoting the string of length 0. If $T \subseteq \Sigma^*$, then the complement of T is denoted by $\bar{T}$; i.e., $\bar{T} = \{x \mid x \in \Sigma^* - T\}$.

For any finite set, say T, over Σ, we let $c(T)$ denote an encoding of T over Σ; i.e., $c(T) \in \Sigma^*$. (We will often refer to finite sets as tables.) It is assumed that for strings $y \in \Sigma^*$ and for finite sets $T \subseteq \Sigma^*$, computing $c(T \cup \{y\})$ from $c(T)$ and y and deciding if $y \in T$ from $c(T)$ and y can both be done in time polynomial in $|y| + |c(T)|$. For any $T \subseteq \Sigma^*$ and any $n \in N$ (N denotes the natural numbers) let $T^{\leqslant n} = \{x \mid x \in T \text{ and } |x| \leqq n\}$. Then, $c(T^{\leqslant n})$ denotes an encoding of an initial segment of T. A set $T \subseteq \Sigma^*$ is sparse if there is a polynomial p such that $|T^{\leqslant n}| \leqq p(n)$ for all $n \in N$.

$\{M_i, p_i \mid i \in N\}$ is an effective enumeration of the nondeterministic oracle Turing machines which run in nondeterministic-polynomial time, with polynomial p_i bounding the running time of M_i. For any oracle Turing machine M (either deterministic or nondeterministic) and any set $S \subseteq \Sigma^*$, the notation M^S denotes that M is using S as the oracle set. Letting T range over the finite subsets of Σ^*, define the set $K = \{\langle i, x, c(T), 0^n\rangle \mid M_i^T(x) \text{ accepts in, at most, } n \text{ steps}\}$. It is important to note that $K \in NP$.

The three types of reducibilities which we use are polynomial-time many-one reducibility from Karp [6], polynomial-time Turing reducibility from Cook [4], and nondeterministic polynomial-time Turing reducibility from Meyer and Stockmeyer [11] and also Ladner *et al.* [8]. Set A is many-one reducible to set B in polynomial time ($A \leqslant_m^P B$) if there is a function f, computable in polynomial time, such that for all $x \in \Sigma^*$, and $x \in A$ if and only if $f(x) \in B$. Set A is Turing reducible to set B in polynomial time ($A \leqslant_T^P B$) if there is a deterministic oracle Turing machine which runs in polynomial time and which recognizes A when using oracle set B. Set A is nondeterministically Turing reducible to set B in polynomial time ($A \leqslant_T^{NP} B$) if there is a nondeterministic oracle Turing machine which runs in nondeterministic-polynomial time and which accepts A when using oracle set B.

For any oracle Turing machine M and any sets $A, B \subseteq \Sigma^*$, the notation $A^{\leqslant n} \leqslant_T^P B$ via M denotes that $\forall x \in \Sigma^*(|x| \leqslant n \Rightarrow (M^B(x) \text{ accepts} \Leftrightarrow x \in A^{\leqslant n}))$. If $|x| > n$, we do not care what the result of $M^B(x)$ is. The notation $A^{\leqslant n} \leqslant_T^{NP} B$ via M is defined similarly.

The polynomial-time hierarchy (PH) was introduced by Meyer and Stockmeyer [11] and further developed in Stockmeyer [12].

Definition 1. The PH is $\{\Sigma_i^P, \Pi_i^P, \Delta_i^P \mid i \in N\}$, where

(A) $\Sigma_0^P = \Pi_0^P = \Delta_0^P = P$ and

(B) for $i \geqslant 0$,

$$\Sigma_{i+1}^P = \{A \mid A \leqslant_T^{NP} B \text{ for some } B \in \Sigma_i^P\}$$
$$\Pi_{i+1}^P = \text{co-}\Sigma_{i+1}^P = \{\bar{A} \mid A \in \Sigma_{i+1}^P\}$$
$$\Delta_{i+1}^P = \{A \mid A \leqslant_T^P B \text{ for some } B \in \Sigma_i^P\}.$$

The proof of the following proposition is straightforward.

Proposition 2. *For $i \geqq 0$, if $A \in \Sigma_{i+1}^P$ and B is $\leqslant_T^P$-complete in Σ_i^P, then $A \leqslant_T^{NP} B$.*

PH is a set hierarchy. However, as a matter of convenience, we will talk about functions other than 0/1 valued functions and about how they relate to PH. In particular, when we say that a procedure (computing a function) is a "Δ_2^P-procedure" we mean that the procedure can be implemented via a deterministic oracle Turing transducer, with polynomial-running time, using an oracle set from Σ_1^P.

Finally, let $\{NP_j \mid j \in N\}$ be an effective enumeration of the nondeterministic Turing machines which run in nondeterministic-polynomial time. Without discussing details, we simply state that there is, for each $j \in N$, a Δ_2^P-procedure, call it COMP_NP_j, which, on input x, outputs an accepting computation of $NP_j(x)$ if NP_j accepts x, and outputs 0 if NP_j does not accept x. The details of COMP_NP_j can easily be gleaned from a similar construction appearing in Lemma 2 of Baker *et al.* [1].

Main Theorem

We begin with two results from [7]. (Proposition 3 has been reworded here.)

Proposition 3. *If NP has polynomial-size circuits then* $\text{PH} = \Sigma_2^P \cap \Pi_2^P$.

As noted in the introduction, if NP has a sparse oracle, then NP has polynomial-size circuits [2]. This proves the next corollary (also from [7]).

Corollary 4. *If NP has a sparse $\leqslant_T^P$-complete set, then* $\text{PH} = \Sigma_2^P \cap \Pi_2^P$.

The proof of Corollary 4 only requires NP to have a sparse $\leqslant_T^P$-hard set. Thus, Lemma 5 follows from the same argument.

LEMMA 5. *If NP has a sparse $\leqslant_T^P$-hard set, then* $\mathrm{PH} = \Sigma_2^P \cap \Pi_2^P$.

We now proceed directly to the main theorem.

THEOREM 6. *If S is a sparse $\leqslant_T^P$-hard set for NP, then S is a sparse $\leqslant_T^P$-hard set for* PH.

Remark. Theorem 6 was motivated in part by a similar result (Lemma 6.2) in [7]. Restating that result in terms of sparse oracles, it says that if every set in NP has a sparse oracle, then every set in PH has a sparse oracle. Different sets in PH may have different sparse oracles. Theorem 6 says that if a single set S is a sparse oracle for every set in NP, then S is also a sparse oracle for every set in PH.

Proof. Assuming the hypothesis of the theorem, let S be a sparse set such that $A \leqslant_T^P S$ for all $A \in NP$. Then, since $\mathrm{PH} = \Sigma_2^P \cap \Pi_2^P$ by Lemma 5, we need only prove that $A \leqslant_T^P S$ for all $A \in \Sigma_2^P$ to conclude that A is a sparse $\leqslant_T^P$-hard set for PH. Thus, let A be an arbitrary set in Σ_2^P. We want to conclude that $A \leqslant_T^P S$.

Proposition 2 and the fact that SAT is $\leqslant_T^P$-complete in NP ($SAT = \{w \in \Sigma^* \mid w$ encodes a satisfiable Boolean formula$\}$) imply that $A \leqslant_T^{NP} SAT$ via some nondeterministic oracle Turing machine, say $M_{A/SAT}$. Let p be a monotone increasing polynomial which bounds the running time of $M_{A/SAT}$. Our assumption about S and the fact that $SAT \in NP$ imply that $SAT \leqslant_T^P S$ via some deterministic oracle Turing machine, say $M_{SAT/S}$. Let p' be a monotone-increasing polynomial bounding the running time of $M_{SAT/S}$. These two reductions can be combined to get $A \leqslant_T^{NP} S$ via a nondeterministic oracle Turing machine, say $M_{A/S}$, which operates as follows:

> On input x, begin a nondeterministic simulation of $M_{A/SAT}$ on input x. Whenever this simulation queries the oracle about a string, say y, then begin simulating $M_{SAT/S}$ on input y with oracle set S. If this new simulation accepts, then resume the simulation of $M_{A/SAT}$ from the YES state, and if the new simulation rejects, then resume the simulation of $M_{A/SAT}$ from the NO state. Accept input x if and only if some simulated computation of $M_{A/SAT}$ accepts.

Since the running time of $M_{A/SAT}$ is bounded above by p, the longest string which $M_{A/SAT}$ can write on its oracle tape when started with input x has, at most, length $p(|x|)$. This implies that $A^{\leqslant |x|} \leqslant_T^{NP} SAT^{\leqslant p(|x|)}$ via $M_{A/SAT}$. Similarly, since the running time of $M_{SAT/S}$ is bounded above by p', $SAT^{\leqslant p(|x|)} \leqslant_T^P S^{\leqslant p'(p(|x|))}$ via $M_{SAT/S}$. Considering the specific way in which $M_{A/S}$ combines the computations of $M_{A/SAT}$ and $M_{SAT/S}$, it follows that $A^{\leqslant |x|} \leqslant_T^{NP} S^{\leqslant p'(p|x|)}$ via $M_{A/S}$. Therefore, letting $M_{A/S} = M_i$ for some $i \in N$, $x \in A \Leftrightarrow \langle i, x, c(S^{\leqslant p'(p|x|)}), 0^{p_i(|x|)}\rangle \in K$, for all $x \in \Sigma^*$. Our assumption about S and the fact that $K \in NP$ imply that $K \leqslant_T^P S$ via some deterministic oracle Turing machine, say $M_{K/S}$. Thus, for all $x \in \Sigma^*$, $x \in A \Leftrightarrow M_{K/S}^S(\langle i, x, c(S^{\leqslant p'(p(|x|))}), 0^{p_i(|x|)}\rangle)$ accepts. If we could, on input x, construct the polynomial-size table $T = S^{\leq p'(p(|x|))}$ in polynomial time using S as an oracle, then $A \leq_T^p S$ via a deterministic oracle Turing machine which operates as follows:

On input x:

(1) use S as an oracle to obtain $T = S^{\leq p'(p(|x|))}$ in polynomial time and

(2) run $M_{K/S}$ on input $\langle i, x, c(T), 0^{p_i(|x|)}\rangle$ with oracle set S.

Hence, we now develop an appropriate algorithm for computing the table T.

The construction of the table $T = S^{\leq p'(p(|x|))}$ is actually done by constructing a series of tables $T_0, T_1, \ldots, T_m$ such that $\varnothing = T_0 \subsetneq T_1 \subsetneq \cdots \subsetneq T_{m-1} = T_m \subseteq S^{\leq p'(p(|x|))}$ and such that $SAT^{\leq p(|x|)} \leq_T^p T_m$ via $M_{SAT/S}$. T is then taken to be T_m. Although it may be that $T \subsetneq S^{\leq p'(p(|x|))}$ for this choice of T, T will contain enough of $S^{\leq p'(p(|x|))}$ to guarantee that $M_{SAT/S}$, when using T as an oracle, accepts $SAT^{\leq p(|x|)}$. Considering once again just how it is that M_i operates, this choice of T also guarantees that $A^{\leq |x|} \leq_T^{NP} T$ via M_i so that, as before, $x \in A \Leftrightarrow M_{K/S}^S(\langle i, x, c(T), 0^{p_i(|x|)}\rangle)$ accepts.

In constructing the series of tables, we make use of an idea which appears in Karp and Lipton [7]. They construct a Π_1^p predicate which says that a polynomial-size circuit works correctly on an initial segment of SAT. Their predicate is based on a very simple inductive definition of SAT. For our purposes, we use this same idea in defining a co-NP set which contains tables which $M_{\mathrm{SAT}/S}$ can use in order to recognize initial segments of SAT.

Some notation is helpful in developing these ideas. Let

$$\mathrm{BF} = \{w \in \Sigma^* \mid w \text{ encodes a Boolean formula } F(x_1, \ldots, x_n) \text{ of } n \geq 0 \text{ variables } x_1, x_2, \ldots, x_n\}$$

and let

$$\mathrm{BF}' = \{w \in \mathrm{BF} \mid w \text{ encodes a Boolean formula of no variables}\}.$$

Note that $\mathrm{BF}' \subseteq \mathrm{BF}$ and if $w \in \mathrm{BF}'$, then the formula which w encodes reduces to *true* or *false*. For each $w \in \mathrm{BF}$ which encodes a formula $F(x_1, x_2, \ldots, x_n)$ of $n > 0$ variables, let w_0 denote the encoding of $F(0, x_2, \ldots, x_n)$ and w_1 denote the encoding of $F(1, x_2, \ldots, x_n)$. SAT can be defined inductively (based on the number of variables appearing in a Boolean formula) as follows:

BASIS CLAUSE. *$w \in BF' \Rightarrow (w \in SAT \Leftrightarrow$ the formula which w encodes reduces to true$)$.*

INDUCTIVE CLAUSE. *$w \in BF - BF' \Rightarrow (w \in SAT \Leftrightarrow w_0 \in SAT \vee w_1 \in SAT)$.*

Now define $\mathrm{OKFORSAT} = \{\langle 0^n, c(T)\rangle \mid SAT^{\leq n} \leq_T^p T \text{ via } M_{SAT/S}\}$. Note that $\langle 0^n, c(T)\rangle \in \mathrm{OKFORSAT}$ if and only if $M_{SAT/S}$, when using T as an oracle set, recognizes $SAT^{\leqslant n}$. Mimicking the inductive definition of SAT, we see that $\langle 0^n, c(T)\rangle \in$ OKFORSAT $\Leftrightarrow$

$$\forall w_{|w|\leqslant n} \left[\begin{array}{l} (w \in BF' \Rightarrow (M^T_{SAT/S}(w) \text{ accepts} \Leftrightarrow \text{the formula which } w \\ \qquad\qquad\qquad\qquad\qquad \text{encodes reduces to } \mathit{true})) \\ \wedge\ (w \in BF\text{—}BF' \Rightarrow (M^T_{SAT/S}(w) \text{ accepts} \Leftrightarrow M^T_{SAT/S}(w_0) \text{ accepts} \\ \qquad\qquad\qquad\qquad\qquad\qquad \vee\ M^T_{SAT/S}(w_1) \text{ accepts})) \\ \wedge\ (w \notin BF \Rightarrow M^T_{SAT/S}(w) \text{ rejects}) \end{array}\right]$$

Using the quantifier characterization of PH in Stockmeyer [12], we see that OKFORSAT $\in \Pi^P_1 =$ co-NP. Letting NP_j accept $\overline{\text{OKFORSAT}}$ (which is in NP), there is a Δ^P_2-procedure COMP_NP_j which, on input $x \in \overline{\text{OKFORSAT}}$, produces an accepting computation of NP_j on input x and which, on input $x \in$ OKFORSAT, outputs 0. If NP_j works in the obvious way by guessing, on input $\langle 0^n, c(T)\rangle$, a w such that either

(1) $w \in BF' \wedge \neg(M^T_{SAT/S}(w)$ accepts $\Leftrightarrow$ the formula which w encodes reduces to *true*) or

(2) $w \in BF - BF' \wedge \neg(M^T_{SAT/S}(w)$ accepts $\Leftrightarrow M^T_{SAT/S}(w_0)$ accepts $\vee\ M^T_{SAT/S}(w_1)$ accepts),

and then verifying that either (1) or (2) holds, we can say more about any accepting computation which COMP_NP_j produces. (We are assuming that if $w \notin$ BF, then $M^T_{SAT/S}(w)$rejects by simple syntactic checking of w and not by asking any questions of the oracle. This implies that if $\langle 0^n, c(T)\rangle \notin$ OKFORSAT, then either (1) or (2) does hold.) Namely, included in this computation will be a simulation of $M^T_{SAT/S}(w)$ and possibly of $M^T_{SAT/S}(w_0)$ and $M^T_{SAT/S}(w_1)$, as well. Thus, assuming that NP_j does work in the obvious way, we let COMP_NP_j' be the Δ^P_2-procedure which operates as follows:

> On input $\langle 0^n, c(T)\rangle$, compute COMP_$NP_j(\langle 0^n, c(T)\rangle)$. If the output produced is 0, then output a dummy value and halt. If the output produced is a computation of NP_j on input $\langle 0^n, c(T)\rangle$, then find the simulation of $M^T_{SAT/S}(w)$ (and of $M^T_{SAT/S}(w_0)$ and $M^T_{SAT/S}(w_1)$ if they also appear) which appears in the computation. From these simulations, output the set of strings queried by $M_{SAT/S}$ which are not in the finite set T and halt.

One final observation about the Δ^P_2-procedure COMP_NP_j'. COMP_NP_j' runs in polynomial time using an oracle set from NP. Since we have assumed that $A \leqslant^P_T S$ for all $A \in NP$, we can assume that COMP_NP_j' runs in polynomial time and uses S as its oracle set.

Using our assumption about S and the fact that $\overline{\text{OKFORSAT}} \in NP$, let $\overline{\text{OKFORSAT}} \leqslant^P_T S$ via $M_{\overline{\text{OKFORSAT}}/S}$. We now present a procedure named BUILD_T which, on input 0^n, outputs a table $T \subseteq S^{\leqslant p'(n)}$ such that $\langle 0^n, c(T)\rangle \in$ OKFORSAT; that is, BUILD_T outputs a table $T \subseteq S^{\leqslant p'(n)}$ such that $SAT^{\leqslant n} \leqslant^P_T T$ via $M_{SAT/S}$. BUILD_T runs in polynomial time and uses S as an oracle set.

```
procedure BUILD_$T(0^n)$;
begin
    $T' \leftarrow \varnothing$;
    while $M^S_{\overline{\mathrm{OKFORSAT}}/S}(\langle 0^n, c(T')\rangle)$ accepts do
        begin
            compute COMP_$NP'_j$ using oracle set $S$ on
            input $\langle 0^n, c(T')\rangle$ to obtain a set of
            strings $\{y_1, y_2, ..., y_k\}$;
            for $i := 1$ to $k$ if $y_i \in S$ then $T' \leftarrow T' \cup \{y_i\}$
        end /* while-do */
    $T \leftarrow T'$
end /* procedure BUILD_$T$ */
```

To prove that BUILD_T runs in polynomial time (using S as its oracle set), note that each execution of the body of the while-do takes polynomial time. Thus, we need only argue that the number of times the body of the while-do is executed is polynomial. Initially, $T' \leftarrow \varnothing$ and new strings added to T' in the for-loop are in S. Therefore, $T' \subseteq S$ throughout the execution of BUILD_T. Furthermore, $T' \subseteq S^{\leqslant p'(n)}$ since the running time of $M_{SAT/S}$ is bounded by p'. We claim that T' increases in size after each execution of the body of the while-do. If this is the case, then the number of executions of the body of the while-do is polynomial since S is sparse.

To see that T' increases in size each time through the loop, note that the body of the loop is entered if and only if $\langle 0^n, c(T')\rangle \in \overline{\mathrm{OKFORSAT}}$. In this case, NP_j does accept $\langle 0^n, c(T')\rangle$ and any accepting computation of NP_j will contain a simulation of $M^{T'}_{SAT/S}(w)$ and possibly of $M^{T'}_{SAT/S}(w_0)$ and $M^{T'}_{SAT/S}(w_1)$ as well. Also, at least one of these $M^{T'}_{SAT/S}$ computations must be yielding an incorrect answer. Because $M_{SAT/S}$ does witness $SAT \leqslant^P_T S$, it must be that any incorrect $M^{T'}_{SAT/S}$ computation is using an incorrect oracle answer. However $M^{T'}_{SAT/S}$ computations use a YES answer to a query if and only if the string being queried is in T', and these answers are correct since we know that $T' \subseteq S$. Therefore, some string, say y, is queried and an incorrect NO answer is used; that is, $y \in S$. y will be among the strings $y_1, y_2, ..., y_k$ produced by COMP_NP'_j on input $\langle 0^n, c(T')\rangle$ since the use of a NO answer to the query about y implies that $y \notin T'$. y will then be added to T' in the for-loop increasing the size of T'.

Having established that BUILD_T runs in polynomial time, it now follows that BUILD_T is correct. This is so because BUILD_T leaves the while-do loop only when $\langle 0^n, c(T')\rangle \in$ OKFORSAT. T is then immediately set to T' so that $\langle 0^n, c(T)\rangle \in$ OKFORSAT.

The proof of the theorem is now concluded by noting that $A \leqslant^P_T S$ via a deterministic oracle Turing machine which operates as follows:

On input x:

(1) Run BUILD_T on input $0^{p(|x|)}$ using oracle set S to obtain a table $T \subseteq S^{\leqslant p'(p|x|)}$ such that $SAT^{\leqslant p(|x|)} \leqslant^P_T T$ via $M_{SAT/S}$.

(2) Run $M^S_{K/S}(\langle i, x, c(T), 0^{p_i(|x|)}\rangle)$. Accept if this computation accepts and reject if this computation rejects. Q.E.D.

COROLLARY 7. *If NP has a sparse* $\leqslant^P_T$-hard *set S and* $S \in \Delta^P_2$, *then* $\mathrm{PH} = \Delta^P_2$.

Proof. Assuming the hypothesis of the corollary and using Theorem 6, $A \leqslant^P_T S$ for all $A \in \mathrm{PH}$. Since $S \in \Delta^P_2$, $S \leqslant^P_T SAT$. By transitivity of $\leqslant^P_T$-reducibility, $A \leqslant^P_T SAT$ for all $A \in \mathrm{PH}$. Therefore, $A \in \Delta^P_2$ for all $A \in \mathrm{PH}$ and $\mathrm{PH} = \Delta^P_2$. Q.E.D.

COROLLARY 8. *If NP has a* $\leqslant^P_T$-complete *set which is either sparse or co-sparse, then* $\mathrm{PH} = \Delta^P_2$.

Proof. Immediate from Corollary 7. Q.E.D.

As stated in the introduction, Mahaney [9] has previously proved that if *NP* has a sparse $\leqslant^P_T$-complete set, then $\mathrm{PH} = \Delta^P_2$.

Finally, we point out that Theorem 6 could be proved without the use of Lemma 5. The proof would proceed by induction on the levels of PH. The base of the induction is established by the assumption that S is a sparse $\leqslant^P_T$-hard set for $NP = \Sigma^P_1$. The inductive step would be to prove, for arbitrary $k \geqslant 1$, that S is a sparse $\leqslant^P_T$-hard set for Σ^P_{k+1} based on the inductive assumption that S is a sparse $\leqslant^P_T$-hard set for Σ^P_K.

To prove that S is a sparse $\leqslant^P_T$-hard set for Σ^P_{k+1} using the assumption that S is a sparse $\leqslant^P_T$-hard set for Σ^P_k, let $A \in \Sigma^P_{k+1}$ and show that $A \leqslant^P_T S$. The proof proceeds just as the proof of Theorem 6 except that the $\leqslant^P_m$-complete set $B_k \in \Sigma^P_k$ is used instead of SAT. (B_k is the set of Boolean formulas which are satisfied under $k-1$ alternations of quantifiers beginning with $\exists$. SAT is just B_1. B_k is formally defined in [12].) The important point is that the sets B_k have simple inductive definitions as does SAT so that the sets $\mathrm{OKFORB}_k \in$ co-*NP* for $k \geqslant 1$. Inductive definitions of B_k are presented in [7].

CONCLUDING REMARKS

We know from Corollary 7 that if S is a sparse $\leqslant^P_T$-hard set for *NP* and $S \in \Delta^P_2$ (or lower in PH), then $\mathrm{PH} = \Delta^P_2$. If $S \notin \Delta^P_2$, the techniques used here do not yield that $\mathrm{PH} = \Delta^P_2$. The strongest known result for the case when $S \notin \Delta^P_2$ is that of Karp *et al.* [7] which says that $\mathrm{PH} = \Sigma^P_2 \cap \Pi^P_2$. Whether or not the existence of a sparse $\leqslant^P_T$-hard set for *NP* would imply that $\mathrm{PH} = \Delta^P_2$ remains an open question.

A possible approach to this open problem is to show that if *NP* has a sparse $\leqslant^P_T$-hard set, then *NP* has a sparse $\leqslant^P_T$-hard set in Δ^P_2. The strongest known result of this type says that if *NP* has a sparse $\leqslant^P_T$-hard set, then *NP* has a sparse $\leqslant^P_T$-hard set in Σ^P_2. It is interesting to note that the existence of a sparse $\leqslant^P_m$-hard set for *NP* does imply that *NP* has a sparse $\leqslant^P_m$-complete set (Mahaney [9]).

Other important open questions include:

(1) Does the existence of a sparse oracle for *NP* imply that $P = NP$?

(2) Does the existece of a sparse oracle for *NP* imply $NP =$ co-*NP*?

Since NP-complete sets have polynomial-size circuits if and only if they have sparse oracles, a positive answer to (1), for example, would prove that NP-complete sets do not have polynomial-size circuits (assuming $P \neq NP$).

Acknowledgment

I thank Uwe Schöning for his comments on an earlier version of this paper. His remarks, in part, motivated Theorem 6.

References

1. T. Baker, J. Gill, and R. Solovay, Relativizations of the $P = NP$? question, *SIAM J. Comput.* **4** (4) (1975), 431–442.
2. L. Berman and J. Hartmanis, On isomorphisms and density of NP and other complete sets, *SIAM J. Comput.* **6** (2) (1977), 305–322.
3. P. Berman, Relationship between density and deterministic complexity of NP-complete languages, *in* "Fifth International Colloquium on Automata, Languages, and Programming," Lecture Noes in Computer Science, Vol. 62, pp. 63–71, Springer-Verlag, Berlin, 1978.
4. S. Cook, The complexity of theorem proving procedures, *in* "Proceedings, third Annual ACM Symposium on Theory of Computing, 1971", pp. 151–158.
5. S. Fortune, A note on sparse complete sets, *SIAM J. Comput.* **8** (3) (1979), 431–433.
6. R. Karp, Reducibility among combinatorial problems, *in* "Complexity of Computer Computations" (Miller and Thatcher, Eds.), pp. 85–103, Plenum, New York, 1972.
7. R. Karp and R. Lipton, Some connections between nonuniform and uniform complexity classes, *in* "Proceedings, 12th ACM Symposium on Theory of Computing, 1980," pp. 302–309.
8. R. Ladner, N. Lynch, and A. Selman, A comparison of polynomial time reducibilities, *Theoret. Comput. Sci.* **1** (1975), 103–123.
9. S. Mahaney, Sparse complete sets for NP: Solution of a conjecture of Berman and Harmanis, *in* "Proceedings, 21st IEEE Symposium, Foundations of Computer Science, (1980)," pp. 54–60.
10. A. Meyer and M. Paterson, With what frequency are apparently intractable problems difficult, *MIT Tech. Rep.* MIT/LCS/TM-126 (1979).
11. A. Meyer and L. Stockmeyer, The equivalence of regular expressions with squaring requires exponential space, *in* "Proceedings, 13th IEEE SWAT 1972," pp. 125–129.
12. L. Stockmeyer, The polynomial time hierarchy, *Theoret. Comput. Sci.* **3** (1977), 1–22.