Relativized Questions Involving Probabilistic Algorithms

CHARLES RACKOFF

University of Toronto, Toronto, Ontario, Canada

ABSTRACT Let $R \subseteq$ NP be the collection of languages L such that for some polynomial-time-computable predicate $P(x, y)$ and constant k,

$$L = \{x \mid \exists y, |y| = |x|^k, P(x, y)\}$$
$$= \{x \mid \exists \text{ at least } 2^{|x|^k - 1} \text{ values of } y, |y| = |x|^k, P(x, y)\}$$

Let $U \subseteq$ NP be the collection of languages L such that for some polynomial-time-computable predicate $P(x, y)$ and constant k,

$$L = \{x \mid \exists y, |y| = |x|^k, P(x, y)\}$$
$$= \{x \mid \exists \text{ unique } y, |y| = |x|^k, P(x, y)\}$$

Let R^A, U^A, P^A, NP^A, and CO-NP^A be the relativization of these classes with respect to an oracle A. Then for some oracles E and F,

$$(\text{NP}^E \cap \text{CO-NP}^E) = R^E = P^E \neq \text{NP}^E \quad \text{and} \quad (\text{NP}^F \cap \text{CO-NP}^F) = U^F = P^F \neq \text{NP}^F,$$

while for some other oracle D,

$$\text{CO-NP}^D = \text{NP}^D = U^D = R^D \neq P^D$$

Categories and Subject Descriptors F 1 2 [**Computation by Abstract Devices**]: Modes of Computation—*probabilistic computation, relativized computation*, F 1 3 [**Computation by Abstract Devices**]. Complexity Classes

General Terms Algorithms, Languages, Theory

Additional Key Words and Phrases NP

1. *Introduction*

The area of computational complexity is concerned with what amount of resources (such as space or time) is intrinsically needed to perform various computational tasks. Since [5] the two most important models of computation considered have been the deterministic and the nondeterministic machine. The former model (or class of models) is important because it reflects most closely the computers which we are actually able to build. The latter is important because nondeterminism appears to be much faster at deciding membership in certain sets, yet one is unable (at present) to prove that nondeterminism ever actually helps.

More recently, a good deal of interest has been shown in probabilistic machines. These are deterministic machines which are able, in effect, to "flip coins" in order to make decisions randomly. One can build computers in such a way, and the question

This work was supported in part by the National Research Council of Canada

An earlier version of this paper was presented at the 10th Annual ACM Symposium on the Theory of Computing [10]

Author's address: Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A7.

arises whether such machines are more powerful than deterministic machines if one has an appropriate notion of acceptance and running time. Gill [6] presents a number of different notions of what it could mean for a language to be acceptable in polynomial time by a probabilistic machine.

Recently [1, 11] a new definition of this concept was proposed. The definition of the class R (for random) of languages is best viewed as an adjustment of the definition of NP (the class of languages acceptable in polynomial time by a nondeterministic Turing machine). A language L is in NP if there is a notion of "proof" such that a string $x \in L$ iff there is a "short proof" that $x \in L$. If, in addition, for every $x \in L$ we are guaranteed the existence of "many" short proofs (for instance, at least half the short strings are proofs), then we say that $L \in R$. R is defined carefully in the abstract.

It is not hard to see that if $L \in R$, then there is a probabilistic algorithm which always halts in polynomial time, such that for any string x, if $x \notin L$, then the algorithm halts saying $x \notin L$; if $x \in L$, then with probability $>1 - 1/1000$ (an algorithm could be found for any probability <1) the algorithm halts saying $x \in L$. The most well-known example of a set known to be in R but not known to be in P (the class of languages acceptable by a deterministic Turing machine in polynomial time) is the set of nonprimes [11]. (See [9] for related material.) If such an algorithm answers "yes" on an input, we can have absolute confidence in its correctness; if the answer is "no," we can have a large degree of confidence. If $L \in R$ and $\bar{L} \in R$, then we can have a probabilistic algorithm which *always* gives the right answer and which for any input very probably halts in polynomial time. (Gill calls the class of such languages ZPP.)

It seems not unlikely that R contains languages not in P. In [1] the question is posed whether one can prove $P \neq \text{NP} \Rightarrow R \neq P$. It is reasonable to try to use merely the assumption that $P \neq \text{NP}$ to prove something interesting, since a result of this type has already been shown by Ladner [7]. He shows that $P \neq \text{NP} \Rightarrow$ there exist sets in $\text{NP} - P$ which are not NP-complete. This author first tried to do a variation on this proof in order to show that $P \neq \text{NP} \Rightarrow R \neq P$ but was unable to succeed. He then realized that one reason for this difficulty is that Ladner's proof *relativizes*, as defined below. For a class of languages C defined with respect to some model of a computer and for a set A of strings, define C^A to be the same as C, only now the model is allowed to ask about membership in A of any string and receive the answer in one step. For instance, NP^A is the class of languages which can be accepted by some nondeterministic machine in polynomial time, using A as an oracle. When we say that Ladner's proof relativizes, we mean that it can be easily adjusted to show that for any A, $P^A \neq \text{NP}^A \Rightarrow \text{NP}^A - P^A$ contains sets which are not NP^A complete.

Hence it seems likely that any natural adjustment of this proof to show that $P \neq \text{NP} \Rightarrow R \neq P$ would show as well that $P^A \neq \text{NP}^A \Rightarrow R^A \neq P^A$ for every oracle A. But this is not true! We show in Section 2 that for some oracle A, $P^A \neq \text{NP}^A$ and yet $R^A = P^A$; in fact, for some other oracle A, $P^A \neq \text{NP}^A$ and $R^A = \text{NP}^A$.

Almost all the proofs we know for showing relations between complexity classes relativize. (There are some counterexamples in [8], but these involve log space, and it is not clear what the "right" definition of relativized log space is.) This implies that it is improbable that known proof techniques will succeed in determining whether or not $R = P$, even assuming that $P \neq \text{NP}$. This complements the result of [2] that for some oracle A, $P^A = \text{NP}^A$, while for some other oracle A, $P^A \neq \text{NP}^A$; this result implies that it is unlikely that known techniques will succeed in determining whether or not $P = \text{NP}$.

It is interesting to note that a stronger version of our second theorem about relativized R has since been proved by apparently very different means. Bennett and Gill [4] show that for "almost all" oracles A, $\mathrm{NP}^A \neq \mathrm{CO\text{-}NP}^A \neq P^A = R^A$.

Now let U (called UP in [13]) be defined as in the abstract. Intuitively, a set $L \in \mathrm{NP}$ is also in U if the notion of "proof" that is used to show it is in NP is such that $x \in L \Leftrightarrow$ there is a *unique* proof that $x \in L$. Some people feel that the sets in U might have easier membership problems than NP-complete sets (this is implicit, for instance, in [13]). The reasoning is that if a nondeterministic machine is guaranteed to have at most one way of accepting an input, then perhaps the operation of the machine is sufficiently simple that one can simulate it with a deterministic machine in polynomial time. In any case, we show that for some oracle A, $P^A \neq \mathrm{NP}^A$ and $U^A = P^A$, while for some other oracle A, $P^A \neq \mathrm{NP}^A$ and $U^A = \mathrm{NP}^A$.

The constructions of oracles which put NP into R and U can be accomplished by slight adjustments to the proof of [2, Th. 5] which shows that for some oracle D, $P^D \neq \mathrm{NP}^D = \mathrm{CO\text{-}NP}^D$. For the sake of completeness, we include proofs of these results. The construction of oracles putting R or U into P uses some very clever ideas from the proof of [2, Th. 6]. This theorem states that for some oracle E, $(\mathrm{NP}^E \cap \mathrm{CO\text{-}NP}^E) = P^E \neq \mathrm{NP}^E$; our proofs have to extend these ideas a good deal.

Section 2 contains proofs of results about R. Section 3 contains proofs of results about U. Section 4 considers some results about R *and* U, and raises some open questions. Although the proofs that follow are self-contained, they will be easier to follow if the reader has already understood the proofs of [2].

2. *Theorems About R*

Recall that R is the set of languages L such that for some polynomial-time-computable predicate $P(x, y)$ and some constant k,

$$\begin{aligned} L &= \{x \mid \exists y, |y| = |x|^k, P(x, y)\} \\ &= \{x \mid \exists \text{ at least } 2^{|x|^k - 1} \text{ values of } y, |y| = |x|^k, P(x, y)\}. \end{aligned}$$

Let $P_1, P_2, P_3, \ldots$ be an enumeration of the polynomial-time-bounded oracle predicates. That is, we let P_i be the ith deterministic Turing machine which asks questions of oracles, and we assume without loss of generality that for any oracle A and input x, P_i^A on x halts in $\leq |x|^i$ steps. We also denote by NP_i the ith nondeterministic Turing machine. P_i^A and NP_i^A represent P_i and NP_i using oracle A.

THEOREM 1. *For some oracle* D, $P^D \neq NP^D = R^D$.

PROOF. We will construct D such that $L(D) \notin P^D$, where $L(D) = \{x \mid \exists y, |y| = |x|, y \in D\}$. Since $L(D) \in \mathrm{NP}^D$, this ensures that $P^D \neq \mathrm{NP}^D$. We also will construct D with the property that $K(D) \in R^D$, where $K(D) = \{0^i 10^j 1x \mid NP_i^D$ accepts x within j steps$\}$; $K(D)$ is the Karp complete set in NP^D, so $K(D) \in R^D \Rightarrow \mathrm{NP}^D \subseteq R^D$. We will put $K(D)$ in R^D by ensuring that

$$\begin{aligned} K(D) &= \{0^i 10^j 1x \mid \exists y, |y| = |0^i 10^j 1x|, 0^i 10^j 1xy \in D\} \\ &= \{0^i 10^j 1x \mid \exists \text{ at least } 2^{n-1} \text{ values of } y, \\ &\qquad |y| = n = |0^i 10^j 1x|, 0^i 10^j 1xy \in D\}. \end{aligned}$$

To ensure this, we have to fulfill conditions α of the form $0^i 10^j 1x$; the *length* of such a condition is $2n$, $n = |0^i 10^j 1x|$, and to fulfill such a condition we have to make sure that $\alpha \in K(D) \Leftrightarrow \exists y, |y| = n, \alpha y \in D \Leftrightarrow \exists 2^{n-1} y, |y| = n, \alpha y \in D$. In order to ensure

that $L(D) \notin P^D$ we have to fulfill condition i for each $i \in N$; to fulfill condition i, we make sure that for some j, $0^j \in L(D) \Leftrightarrow P_i^D$ does not accept 0^j.

Before stage n we will have determined whether or not $x \in D$ for all x, $|x| < n$; call the set of such strings in D, $D(n)$. In addition, we will have reserved for $\bar{D}$ at most $2^{n/2-1}$ strings of length $\geq n$. Before stage 0, no strings have been reserved for $\bar{D}$. Stage n is as follows.

If n is even, fulfill all conditions of length n. To fulfill condition $\alpha = 0^i 1 0^j 1 x$, see if $\mathrm{NP}_i^{D(n)}$ accepts x in $\leq j$ steps; if so, put $\alpha y \in D$ for each y such that αy is not already reserved for $\bar{D}$; if not, add nothing to D. Note that NP_i^D operating in j steps can only ask the oracle about strings of length $\leq j < n$, so this ensures that condition α is truly fulfilled. After all conditions of length n have been fulfilled in this way, add no other strings of length n to D and reserve no further bigger strings for $\bar{D}$.

If n is odd and i is the smallest unfulfilled condition of the other type, then we will fulfill condition i at this stage if it is possible to do so while retaining the induction hypothesis. If there are any strings of length $\geq n$ reserved for $\bar{D}$, or if $n^i > 2^{(n+1)/2-1}$, then add no strings of length n to D and reserve no bigger strings for $\bar{D}$; otherwise, fulfill condition i. To fulfill condition i, run $P_i^{D(n)}$ on 0^n, reserving for $\bar{D}$ all strings of length $\geq n$ queried about. If $P_i^{D(n)}$ accepts 0^n, add no strings of length n to D. If $P_i^{D(n)}$ does not accept 0^n, add some string of length n not reserved for $\bar{D}$ (there must be one) to D.

It is not hard to see that all conditions are eventually fulfilled. □

THEOREM 2. *There exists an oracle E such that $NP^E \neq P^E = R^E$.*

PROOF. To begin with, let A be an oracle such that $\mathrm{PSPACE}^A = P^A$; for instance, we can let A be any complete member of PSPACE, the class of sets which can be accepted in polynomial space by a Turing machine (see [2]). Clearly, we can also choose A to contain no string of even length. We will form E by adding to A at most one string of length $e(n)$ for every $n \in N$, where we define $e(0) = 2$ and $e(n+1) = 2^{2^{e(n)}}$. Since $e(n)$ is always even, we see that A contains no string of length $e(n)$ for any n. Define $L(E) = \{x \mid \exists y, |x| = |y|, y \in E\}$. E will have the property that $L(E) \notin P^E$, and hence $\mathrm{NP}^E \neq P^E$. For every condition $i \in N$ we wish to fulfill condition i by ensuring that P_i^E does not accept $L(E)$. At stage $n \in N$ in the construction we will put at most one string of length $e(n)$ into E. Let $E(n)$ be the contents of E before stage n, so that initially $E(0) = A$.

Stage n is as follows. Let i be the smallest unfulfilled condition. If $(e(n))^i \geq 2^{e(n)}$, then do nothing; otherwise, fulfill condition i. To fulfill condition i, run $P_i^{E(n)}$ on $0^{e(n)}$; if this accepts, do nothing; if this rejects, put into E some string of length $e(n)$ which has not been queried about. Since no string of length $e(n+1)$ or bigger is queried about, condition i is fulfilled.

Eventually all conditions are fulfilled, so $P^E \neq \mathrm{NP}^E$.

It remains to show that $R^E \subseteq P^E$, and it is sufficient to do this for the case $k = 1$ in the definition of R. (This is because any set in R^E is polynomially transformable to a set satisfying this property.) Let i be such that

$$\begin{aligned} S &= \{x \mid \exists y, |y| = |x|, P_i^E \text{ accepts } xy\} \\ &= \{x \mid \exists 2^{|x|-1} \text{ values of } y, |y| = |x|, P_i^E \text{ accepts } xy\}. \end{aligned}$$

Then $S \in P^E$, as will now be proved.

Let $x \in \{0,1\}^*$, $|x| = m$. Let n be such that $e(n-1) \leq \log_2 m \leq e(n)$. Assume that x is sufficiently large so that $(2m)^i < 2^m$, and so P_i^E on xy never queries about strings as long as $e(n+1)$.

Begin by computing which strings were added to E before stage n by querying E about all strings of length $e(j)$, $j < n$; there are only $O(m)$ such queries since $e(n-1) \leq \log_2 m$. This set is $E(n) - A$.

Definition. Define a string w, $|w| = e(n)$, to be *critical* (for x) if there exist at least 2^{m-1} different values of y, $|y| = m$, such that $P_i^{E(n)}$ on xy queries about w.

LEMMA 2.1. *There are at most* $2(2m)^i$ *critical strings.*

PROOF OF LEMMA. If there are c critical strings, then there are at least $c \cdot 2^{m-1}$ queries in $P_i^{E(n)}(xy)$ as y ranges over strings of length m. The total number of such queries is at most $2^m \cdot (2m)^i$. So $c \cdot 2^{m-1} \leq 2^m \cdot (2m)^i$, and the lemma follows. □

We now wish to compute the set W (of size ≤ 1) of critical strings in E. Assume $e(n) \leq (2m)^i$, since otherwise W is empty. The fact that $\text{PSPACE}^A = P^A$ means that there is a procedure which, given a pair $(M, 0^j)$ where M is a deterministic oracle machine, decides using oracle A in *time* polynomial in $|M| + j$ if M^A accepts λ (the empty string) in *space* j. Now let M be a machine which uses oracle A to decide in a small amount of space if there is a critical string. For each string w of length $e(n)$, M tests if w is critical; to test if w is critical, M simulates $P_i^{E(n)}$ on xy for each y of length m, the simulation taking place using oracle A and the set $E(n) - A$. The machine M has x as well as the set $E(n) - A$ built into it and has size less than some polynomial in m^i. The space used by M (on λ) is also less than some polynomial in m^i. Hence $\text{PSPACE}^A = P^A$ implies that using oracle A (and hence E), one can determine in *time* polynomial in m^i if there is a critical string.

Actually, we wish to compute all the critical strings. For each k, $1 \leq k \leq 2(2m)^i$, and j, $1 \leq j \leq e(n)$, let $M_{k,j}$ be an oracle machine which uses oracle A (together with $E(n) - A$) to accept λ iff there are at least k critical strings and the jth bit of the kth critical string (in lexicographical order) is 1. There is a fixed polynomial in m^i bigger than the space used by any of these machines and bigger than the size of any of these machines. So $\text{PSPACE}^A = P^A$ implies that all of the critical strings can be computed using oracle A (and hence E) in time polynomial in m^i. We can now compute W by asking which critical strings are in E.

If W is nonempty (and hence of size one), then $E(n+1) = E(n) \cup W$, so $x \in S \Leftrightarrow \exists y, |y| = |x|, P_i^{E(n) \cup W}$ accepts xy. There exists a machine M using oracle A which runs in space polynomial in m^i and is of size polynomial in m^i, which accepts $\lambda \Leftrightarrow \exists y, |y| = |x|, P_i^{E(n) \cup W}$ accepts xy; M does the obvious thing, using x and $(E(n) \cup W) - A$ which are built into it. Hence $\text{PSPACE}^A = P^A$ implies that using A (and hence E) we can compute if $\exists y, |y| = |x|, P_i^{E(n) \cup W}$ accepts xy, in time polynomial in m^i. (Note that it was sufficient here to assume that $\text{NP}^A = P^A$.)

So let us assume that W is empty.

LEMMA 2.2. *If* $W = \varnothing$ *and* $x \in S$, *then* $\exists y, |y| = |x|, P_i^{E(n)}$ *accepts* xy.

PROOF OF LEMMA. This is obvious unless there exists a string $t \in E$ of length $e(n)$ (if not, $E(n+1) = E(n)$). t must be noncritical. $x \in S \Rightarrow$ there are at least 2^{m-1} values of y, $|y| = m$, P_i^E accepts xy. Since t is not critical,

$$\exists y, |y| = m, P_i^E \text{ accepts } xy \text{ without querying about } t$$
$$\Rightarrow \exists y, |y| = |x|, P_i^{E(n)} \text{ accepts } xy.$$

□

As above, since $P^A = \text{NP}^A$, we can quickly test if $\exists y, |y| = |x|, P_i^{E(n)}$ accepts xy. If not, we know from Lemma 2.2 that $x \notin S$. So assume that for some y, $|y| = |x|$ and $P_i^{E(n)}$ accepts xy. Using the fact that $P^A = \text{NP}^A$, we can compute such a y quickly

using oracle E. To determine whether or not $x \in S$, we now determine which elements of length $e(n)$ are queried about in the computation of $P_i^{E(n)}$ on xy and determine if one of those is in E. If not, P_i^E accepts xy and $x \in S$. If so, we now know $E(n+1) - A$; $x \in S \Leftrightarrow \exists y, |y| = |x|, P_i^{E(n+1)}$ accepts xy, and this can be computed using E in time polynomial in m^t. □

3. *Theorems About U*

Recall that U is the set of languages L such that for some polynomial-time-computable predicate $P(x, y)$ and constant k,

$$\begin{aligned} L &= \{x \mid \exists y, |y| = |x|^k, P(x, y)\} \\ &= \{x \mid \exists \text{ unique } y, |y| = |x|^k, P(x, y)\}. \end{aligned}$$

THEOREM 3. *There is an oracle D such that $P^D \neq NP^D = U^D$.*

PROOF. Construct D as in the proof of Theorem 1, but instead of putting all strings αy into D that have not been reserved for $\bar{D}$, put exactly one such string in. □

THEOREM 4. *There is an oracle E such that $NP^E \neq P^E = U^E$.*

PROOF. As in the proof of Theorem 2, we let A be an oracle such that $P^A = \text{PSPACE}^A$ and A contains no strings of length $e(n)$ for any n, where $e(n)$ is as in the proof of Theorem 2. To construct E, we start with A, and at stage n add zero or more strings of length $e(n)$. This time there are two types of conditions we wish to fulfill. To fulfill condition i, we have to ensure that P_i^E does not accept $L(E) = \{x \mid \exists y, |y| = |x|, y \in E\}$. To fulfill condition i' (*if possible*), we have to ensure that P_i^E fails to have the *uniqueness property*; P_i^E has the uniqueness property if for every x there is at most one y, $|y| = |x|$, P_i^E accepts xy. Linearly order all the conditions in some arbitrary way (for instance, 0, 0′, 1, 1′, . . .). We denote by $E(n)$ the contents of E before stage n; $E(0) = A$. Stage n is as follows.

At stage n, put zero or more strings of length $e(n)$ into E in order to fulfill the *first* unfulfilled condition it is possible to fulfill (if any). In case $(e(n))^t < 2^{e(n)}$, we can fulfill condition i as in the proof of Theorem 2. We can fulfill condition i' if there is some string x and some set V of strings of length $e(n)$ such that

(1) $\exists$ at least two values of y, $|x| = |y|$, $P_i^{E(n) \cup V}$ accepts xy, and
(2) $(2|x|)^t < e(n+1)$.

Condition i' can be fulfilled by adding V to E.

Eventually condition i gets fulfilled for every $i \in N$, so $P^E \neq \text{NP}^E$. It remains to show that $U^E \subseteq P^E$. Let P_i^E have the uniqueness property. Let

$$\begin{aligned} S &= \{x \mid \exists y, |y| = |x|, P_i^E \text{ accepts } xy\} \\ &= \{x \mid \exists \text{ unique } y, |y| = |x|, P_i^E \text{ accepts } xy\}. \end{aligned}$$

We wish to show that $S \in P^E$. Let x be a string, $|x| = m$, such that we wish to decide if $x \in S$.

Let n be such that $e(n-1) \leq \log_2 m \leq e(n)$, and assume that m is large enough that $(2m)^t < e(n+1)$. Since condition i' was never fulfilled, condition i' was never fulfillable except at some finite number of stages. So assume as well that m is sufficiently large that i' was not fulfillable at stage n. As in the proof of Theorem 2, we begin by calculating $E(n) - A$ using oracle E.

To decide if $x \in S$, we will use the fact that $P^A = \text{PSPACE}^A$ to compute a set of "special" strings and then ask which of these are in E. On the basis of these answers we compute new special strings, etc. After a "small" number of such steps we will know if $x \in S$.

Say that W and W' are two disjoint sets of strings of length $e(n)$ such that $W \subseteq E$ and $W' \cap E = \emptyset$. Let y be a string of length m, and consider an accepting computation of P_i on xy which answers "yes" to oracle questions about $E(n) \cup W$, "no" to questions about W', "no" to questions about strings not in $E(n) \cup W$ and not of length $e(n)$, and consistently to questions about strings of length $e(n)$ not in $W \cup W'$. Call the set of pairs of oracle questions and answers about strings of length $e(n)$ not in $W \cup W'$ *an accepting computation with respect to* (W, W').

LEMMA 4.1. *Every two distinct, nontrivial (i.e., nonempty) such computations must contain an oracle question in common.*

PROOF. Assume otherwise, Then we must have (distinct) y_1, y_2 of length m such that there are accepting computations with respect to (W, W') for y_1 and y_2 with no common oracle question. Then there is some set V of strings of length $e(n)$ such that $P_i^{E(n) \cup W \cup V}$ accepts both xy_1 and xy_2. This contradicts the fact that condition i' was not fulfillable at stage n and proves the lemma. □

To decide if $x \in S$, do the following: Set $W = W' = \emptyset$; GO TO L.

L. Using $P^A = \text{NP}^A$, see if there are any accepting computations with respect to (W, W'); if not, $x \notin S$, if so, see if there is a trivial such computation, if so, $x \in S$, if not, compute (using $P^A = \text{NP}^A$) one such computation (All this can be done in time polynomial in $m' + |W| + |W'|$, as in the proof of Theorem 2) For each of the oracle questions in this computation, test each of the elements for membership in E, call V the set found to be in E, and call V' the set found not to be in E. Set $W = W \cup V$ and $W' = W' \cup V'$ GO TO L □

Lemma 4.1 implies that the loop will be performed at most $(2m)^i$ times, since each time through the loop the length of the longest accepting computation with respect to (W, W') decreases by at least one.

5. *Other Results and Open Questions*

THEOREM 5. *For some oracle D,*

$$CO\text{-}NP^D = NP^D = U^D = R^D \neq P^D.$$

Theorem 5 can easily be proved by combining the proofs of Theorems 1 and 3 of this paper and [2, Th. 5]. The author announced earlier a similar result combining Theorems 2 and 4, but an error was found in the proof, leaving the question open. However the known results can easily be combined to show

THEOREM 6. *For some oracles E and F,*

$$(NP^E \cap CO\text{-}NP^E) = R^E = P^E \neq NP^E$$

and

$$(NP^F \cap CO\text{-}NP^F) = U^F = P^F \neq NP^F.$$

The most interesting open question about relativization that we know of is whether or not there exists some oracle A such that $\text{CO-NP}^A = \text{NP}^A \neq \text{PSPACE}^A$; this would mean that the entire Meyer–Stockmeyer PSPACE^A hierarchy [12] collapses into NP^A, except that PSPACE^A remains different from NP^A.

The best result in the other direction, where one attempts to construct an oracle such that the hierarchy does not collapse, is by Baker and Selman [3]. They construct an oracle A such that $\Sigma_2^{P,A} \neq \Pi_2^{P,A}$.

Gill [6] provides further relativization results about other notions of probabilistic acceptance.

ACKNOWLEDGMENT. The author would like to thank Dana Angluin for pointing out and correcting an error in an earlier version of the proof of Theorem 2.

REFERENCES

1 ADLEMAN, L., AND MANDERS, K. Reducibility, randomness, and intractibility 9th Ann ACM Symp on Theory of Computing, Boulder, Colo., 1977, pp. 151–153
2. BAKER, T., GILL, J., AND SOLOVAY, R. Relativizations of the P = NP? question *SIAM J Comput 4* (1975), 431–442.
3 BAKER, T., AND SELMAN, A A second step toward the polynomial hierarchy. 17th Ann IEEE Symp on Foundations of Computer Science, Houston, Texas, 1976, pp. 71–75
4 BENNETT, C H., AND GILL, J Relative to a random oracle A, $P^A \neq NP^A \neq CO\text{-}NP^A$ with probability 1 *SIAM J Comput. 10* (1981), 96–113
5 COOK, S The complexity of theorem proving procedures 3rd. Ann ACM Symp on Theory of Computing, Shaker Heights, Ohio, 1971, pp. 151–158.
6. GILL, J Computational complexity of probabilistic Turing machines. *SIAM J. Comput. 6* (1977), 675–695. Also unpublished manuscript, Forschungsinstitut für Mathematik, Eidgenossische Technische Hochschule, Zurich, 1976.
7. LADNER, R On the structure of polynomial time reducibility *J. ACM 22*, 1 (Jan, 1975), 155–171
8. LADNER, R., AND LYNCH, N Relativization of questions about log space computability *Math Syst. Theory 10* (1976), 19–32
9 MILLER, G L Riemann's hypothesis and tests for primality *J Comput. Syst. Sci., 13* (1976), 300–317
10 RACKOFF, C Relativized questions involving probabilistic algorithms. 10th Ann ACM Symp. on Theory of Computing, San Diego, Calif., 1978, pp 338–342
11 SOLOVAY, R., AND STRASSEN, V A fast Monte-Carlo test for primality *SIAM J Comput 6* (1977), 84–85
12 STOCKMEYER, L J. The polynomial time hierarchy *Theor Comput Sci 3* (1976), 1–22
13. VALIANT, L. Relative complexity of checking and evaluating *Inf Proc 5* (1976), 20–23

RECEIVED AUGUST 1979, REVISED APRIL 1981, ACCEPTED MAY 1981