

1. Introduction

A central issue in computational complexity theory is to distinguish computational problems that are solvable using efficient resources from those that are inherently intractable. Computer scientists have succeeded in establishing upper bounds on the amount of time needed to solve most computational problems, but for the most part lower bounds remain elusive. There are important classes of problems for which exponential time algorithms are known, but it is not known whether these problems can be solved efficiently.

In particular, much attention has been focused on the complexity classes P and NP, and the underlying Turing machine model of computation. Garey and Johnson [6] have catalogued dozens of NP-complete problems which have efficient algorithms if and only if $P = NP$. Initial attempts to show $P \neq NP$ used diagonalization, a technique which typically relativizes to any oracle. The construction by Baker, Gill and Solovay [1] of oracles B and B' such that $P.T(B) \neq NP.T(B)$ and $P.T(B') = NP.T(B')$ cast some doubt on the applicability of diagonalization for solving $P = ?NP$ and the use of relativization for gaining insight about nonrelativized problems.

A deterministic polynomially time-bounded reducibility \leq_r^P and its nondeterministic analogue \leq_r^{NP} provide a *positive relativization* of the $P = ?NP$ problem if $P = NP$ is equivalent to $P.r(B) = NP.r(B)$ for all oracle sets B . Although polynomially time-bounded Turing reducibility is not known to provide a positive relativization of the $P = ?NP$ problem, Book, Long and Selman [3] have found a restriction of this reducibility which does positively relativize the problem. Their restrictions are called quantitative since they limit the number of queries that a nondeterministic polynomially time-bounded oracle Turing machine can make.

This paper examines several restrictions of polynomially time-bounded Turing reducibility, both quantitative and non-quantitative, for their ability to positively relativize the $P = ?NP$ question. In §2, we establish notational conventions and define the basic notions of relative complexity theory. In §3, we define reducibilities of various strengths between polynomially time-bounded many-one and Turing reducibility, and show that each is distinct. In §4, we determine which of the reducibilities provide positive relativizations of the $P = ?NP$ problem.

2. Notation and Relative Complexity Notions

We assume the reader is familiar with the basic concepts of computability and language theory, particularly the Turing machine model of computation and the complexity classes P and NP. N denotes the set of nonnegative integers. Strings are taken over the alphabet $\Sigma = \{0, 1\}$. The following notation is used: $i, j, k, m, n \in N$; $q, w, x, y \in \Sigma^*$; $A, B, C \subseteq \Sigma^*$; $\#$ is a symbol not in Σ ; M is a Turing machine of some sort; and p is a polynomial. $|x|$ denotes the length of x and $\|A\|$ denotes the cardinality of A . Concatenation is implied by writing two or more strings adjacently (e.g., $w1$ or $x\#y$). The symbols in a string y of length n are denoted y_1, y_2, \dots, y_n and \bar{y}_k is the complement of y_k . Finally, $\langle i, j \rangle: N \times N \rightarrow N$ is a pairing function computable in polynomial time.

We turn now to relative complexity notions. An augmented type of Turing machine called an oracle Turing machine can be used to measure the relative complexity of two sets.

Definition 2.1. An *oracle Turing machine* is a Turing machine with a distinguished query tape and distinguished states Q, YES, and NO. When the machine enters state Q, the next state will be YES or NO depending on whether or not the string currently written on the query tape is in an associated *oracle set*.

Polynomially time-bounded oracle Turing machines are used to classify problems that are tractable with respect to some oracle problem. This relative complexity notion is made precise by defining a polynomially time-bounded Turing reducibility relation. The result of applying polynomial time bounds to Turing reducibility is sometimes called Cook's reducibility [5].

Definition 2.2. $A \leq_T^P B$ ($A \leq_T^{NP} B$) (A is (non)deterministically Turing reducible to B in polynomial time) if there is a (non)deterministic oracle Turing machine using oracle set B that makes at most a polynomial number of moves in the length of an input x , and accepts if and only if $x \in A$.

Another reducibility relation is the polynomially time-bounded restriction of many-one reducibility introduced by Karp. An augmented type of Turing machine called a Turing transducer can be used to compute partial recursive functions and characterize the many-one reducibilities.

Definition 2.3. A *Turing transducer* is a Turing machine with a distinguished output tape. A Turing transducer computes a value y given input x if there is a halting computation that leaves y on the output tape.

Notice that functions computed by nondeterministic Turing transducers are in general multivalued. By considering Turing transducers which operate in polynomial time, several subsets of the partial recursive functions arise.

Definition 2.4. PSV is the set of single-valued partial functions computed by polynomially time-bounded deterministic Turing transducers. NPSV is the set of single-valued partial functions computed by polynomially time-bounded nondeterministic Turing transducers. NPMV is the set of multivalued partial functions computed by polynomially time-bounded nondeterministic Turing transducers.

Karp's relative complexity notion [7], which is important in the theory of NP-completeness, is made precise as follows.

Definition 2.5. $A \leq_m^P B$ ($A \leq_m^{\text{NP}} B$) (A is (non)deterministically many-one reducible to B in polynomial time) if there is a Turing transducer that computes a function in PSV(NPSV) such that $x \in A$ if and only if $f(x)$ is defined and is in B .

Please note that we have strayed from the definition of nondeterministic polynomially time-bounded many-one reducibility (\leq_m^{NP}) in Ladner, Lynch and Selman [8]. Our intent is to give new definitions of nondeterministic reducibilities which have different properties.

3. Reducibilities between Karp's and Cook's

The following proposition and its proof are well known; we present it to illustrate and compare restrictions of Turing reducibility that are less restrictive than many-one reducibility.

Proposition.

- (1) $A \leq_m^P B \implies A \leq_T^P B$.
- (2) $A \leq_m^{\text{NP}} B \implies A \leq_T^{\text{NP}} B$.

Proof. Every many-one reduction can be simulated by a Turing reduction. Simulate the former exactly except that anything written on the transducer's output tape is written on the oracle machine's query tape. When the transducer halts, enter the query state Q and accept if and only if the oracle machine responds by entering state YES. Any time bound or determinism will be inherited by the oracle Turing machine. ■

The many-one simulation makes only one query; moreover, it is the last move made and the oracle machine must accept or reject as the query response dictates. Since many-one reducibility is actually a restricted form of Turing reducibility, it is said to be stronger than Turing reducibility. Formally, a reducibility r is *stronger* than a reducibility s if for every set A and B , if A is r -reducible to B , then A is s -reducible to B . Reducibility r is *properly stronger* than s if r is stronger than s , but s is not stronger than r . For example, polynomially time-bounded many-one reducibility is stronger than polynomially time-bounded Turing reducibility by the proposition. In fact, Karp's reducibility is properly stronger than Cook's reducibility [8]. In this section, polynomially time-bounded reducibilities of varying strengths between these two reducibilities are defined by placing natural restrictions on the behavior of oracle Turing machines.

Definition 3.1. $Q(M, B, x)$ is the set of strings queried in every computation of oracle Turing machine M on input x using oracle set B . $Q(M, x) = \bigcup_{B \subseteq \Sigma^*} Q(M, B, x)$ is called the *query set* of M on input x .

Notice that for a given oracle, a polynomially time-bounded deterministic oracle Turing machine can make at most a polynomial number of queries while its nondeterministic counterpart can make an exponential number. That is, if M is nondeterministic, then $\|Q(M, B, x)\|$ as a function of $|x|$ may not be bounded above by any polynomial. For a deterministic machine M , $\|Q(M, x)\|$ may not be bounded by any polynomial in $|x|$. Such a machine is intuitively “adaptive” since the set of strings queried by the machine depends on the oracle set used, and later queries depend on earlier ones. Deterministic machines that are not adaptive are called non-adaptive. The key feature of a non-adaptive oracle Turing machine is that there is a function f such that for every input x , $f(x) = Q(M, x)$. Now we extend this consideration to nondeterministic oracle Turing machines.

Definition 3.2. A (non)deterministic oracle Turing machine is *non-adaptive* if there is a function $f \in \text{PSV}(\text{NPSV})$ such that for every x , $f(x) = q_1 \# q_2 \# \dots \# q_k$ where $Q(M, x) = \{q_1, q_2, \dots, q_k\}$.

The function f enumerates the query set of M on input x independent of the choice of oracle set. There is an implicit polynomial bound on the size of $Q(M, x)$, since the length of the output of a polynomially time-bounded Turing transducer is polynomially bounded. The oracle Turing machine that simulates a polynomially time-bounded many-one reduction is non-adaptive since its single query is enumerated by the original Turing transducer and is independent of the chosen oracle set.

Now consider restricting the acceptance behavior of an oracle Turing machine. Recall that the many-one simulation must accept or reject as the query response dictates. This “positiveness” restriction can be generalized by defining an oracle Turing machine to be positive if its acceptance behavior with a particular oracle set is unchanged by adding strings to that oracle.

Definition 3.3. An oracle Turing machine M is *positive* if for every input x and oracle sets B and C , if x is accepted by M with oracle set B and $B \subseteq C$, then x is accepted by M with oracle set C .

We consider reducibilities of varying strengths between polynomially time-bounded many-one and Turing reducibility which are witnessed by non-adaptive and/or positive oracle Turing machines.

Definition 3.4.

- (1) $A \leq_{\text{tt}}^{\text{P}} B$ ($A \leq_{\text{tt}}^{\text{NP}} B$) (A is (non)deterministically truth-table reducible to B) if $A \leq_{\text{T}}^{\text{P}} B$ ($A \leq_{\text{T}}^{\text{NP}} B$) via a non-adaptive oracle Turing machine.
- (2) $A \leq_{\text{pos}}^{\text{P}} B$ ($A \leq_{\text{pos}}^{\text{NP}} B$) (A is (non)deterministically positive reducible to B) if $A \leq_{\text{T}}^{\text{P}} B$ ($A \leq_{\text{T}}^{\text{NP}} B$) via a positive oracle Turing machine.
- (3) $A \leq_{\text{ptt}}^{\text{P}} B$ ($A \leq_{\text{ptt}}^{\text{NP}} B$) (A is (non)deterministically positive truth-table reducible to B) if $A \leq_{\text{T}}^{\text{P}} B$ ($A \leq_{\text{T}}^{\text{NP}} B$) via a positive non-adaptive oracle Turing machine.
- (4) $A \leq_{k\text{-tt}}^{\text{P}} B$ ($A \leq_{k\text{-tt}}^{\text{NP}} B$) (A is (non)deterministically k -question truth-table reducible to B) if $A \leq_{\text{tt}}^{\text{P}} B$ ($A \leq_{\text{tt}}^{\text{NP}} B$) via an oracle Turing machine M such that for every input x , $\|Q(M, x)\| = k$.

- (5) $A \leq_{\text{btt}}^{\text{P}} B$ ($A \leq_{\text{btt}}^{\text{NP}} B$) (A is (non)deterministically bounded truth-table reducible to B) if there is a constant k such that $A \leq_{k\text{-tt}}^{\text{P}} B$ ($A \leq_{k\text{-tt}}^{\text{NP}} B$).
- (6) $A \leq_{k\text{-ptt}}^{\text{P}} B$ ($A \leq_{k\text{-ptt}}^{\text{NP}} B$) (A is (non)deterministically k -question positive truth-table reducible to B) if $A \leq_{\text{ptt}}^{\text{P}} B$ ($A \leq_{\text{ptt}}^{\text{NP}} B$) via an oracle Turing machine M such that for every input x , $\|Q(M, x)\| = k$.
- (7) $A \leq_{\text{bptt}}^{\text{P}} B$ ($A \leq_{\text{bptt}}^{\text{NP}} B$) (A is (non)deterministically bounded positive truth-table reducible to B) if there is a constant k such that $A \leq_{k\text{-ptt}}^{\text{P}} B$ ($A \leq_{k\text{-ptt}}^{\text{NP}} B$).

The first study of polynomially time-bounded reducibilities is Ladner, Lynch and Selman [8]. Many of the reducibilities defined here were introduced in that paper; however, we have taken a different approach by defining reducibilities characterized by non-adaptive and positive oracle machines instead of truth-table conditions. We use the same notation scheme as in [8] and the deterministic reducibilities are the same as in [8], but caution that the nondeterministic reducibilities are not equivalent. We repeat, the reducibility $\leq_{\text{m}}^{\text{NP}}$ defined here is not the same as in Ladner, Lynch and Selman. Our nondeterministic reducibilities have the advantage that they form a proper hierarchy and yield positive relativizations of the $\text{P} = ?\text{NP}$ problem. In general, the nondeterministic reducibilities defined here are obtained by replacing NPMV by NPSV in the definition schemes of [8]. Definition 3.5 gives a scheme for denoting the reduction classes of these reducibilities.

Definition 3.5.

$$\text{P.r}(B) = \{ A \mid A \leq_r^{\text{P}} B \}.$$

$$\text{NP.r}(B) = \{ A \mid A \leq_r^{\text{NP}} B \}.$$

For every oracle B , Figure 1 shows the trivial inclusions among the polynomially time-bounded reducibilities defined heretofore.

We now show that every reduction class on the top face of Figure 1 is distinct and every reduction class on the bottom face is distinct. In each of the proofs, an effective enumeration of the polynomially time-bounded oracle Turing machines will be needed to facilitate diagonalization. Since oracle Turing machines and polynomials are syntactic entities, they are both effectively enumerable. By pairing them, effective enumerations of the polynomially time-bounded oracle Turing machines, $\{P_i(\text{NP}_i)\}_{i \in \mathbb{N}}$, are obtained. P_i^B (NP_i^B) denotes machine P_i (NP_i) using oracle set B , and $p_i(n)$ denotes the polynomial running time of P_i (NP_i).

Theorem 3.1. There exist recursive sets A and B such that $A \leq_{\text{pos}}^{\text{P}} B$ and $A \not\leq_{\text{tt}}^{\text{NP}} B$.

Proof. Consider the following reduction procedure using oracle set B .

$$\forall B[X(B) \subseteq Y(B)]$$

Figure 1

```

input  $x$ ;
 $w := x\#$ ;
for  $k := 1$  to  $|x|$  do
    if  $w0 \in B$  then
        if  $w1 \in B$  then accept  $x$ 
        else  $w := w0$ 
    else
        if  $w1 \in B$  then  $w := w1$ 
        else reject  $x$ ;
reject  $x$ 

```

This reduction procedure is certainly deterministic; it also requires a polynomial amount of time in the length of the input. In addition to these properties, the procedure is positive since adding strings to the oracle set cannot cause a previously accepted input string to be rejected. Define set A in terms of oracle set B to be $\{x \mid x \text{ is accepted by the above procedure using oracle } B\}$. By definition, $A \leq_{\text{pos}}^P B$. To guarantee $A \not\leq_{\text{tt}}^{\text{NP}} B$, an oracle set B will be effectively constructed so that no polynomially time-bounded non-adaptive oracle Turing machine correctly witnesses the reduction. That is, for each polynomially time-bounded oracle Turing machine NP_i and polynomial p_j , some input string x will witness either $\|Q(\text{NP}_i, x)\| > p_j(|x|)$ or $x \in A$ if and only if x is not accepted by NP_i^B . Each condition precludes a correct truth-table reduction. Intuitively, such an input can be provided since the deterministic reduction is adaptive and its query set size is exponential while any truth-table reduction is constrained to a polynomial query set

size. The construction of B will proceed in stages. At each stage $m = \langle i, j \rangle$, a finite initial segment B_m of B will be constructed such that $B_m \supseteq B_{m-1}$.

stage 0:

$$\begin{aligned} B_0 &:= \phi; \\ n_0 &:= 0; \end{aligned}$$

stage $m = \langle i, j \rangle > 0$:

- (1) $B_m := B_{m-1}$;
 - (2) $n :=$ the least integer such that $n > n_{m-1}$ and $p_j(n) < 2^n$;
 - (3) $n_m := p_i(n)$;
 - (4) **if** $\|Q(\text{NP}_i, 0^n)\| \leq p_j(n)$ **then**
begin
 - (5) $y :=$ some string such that $|y| = n$ and $0^n \# y \notin Q(\text{NP}_i, 0^n)$;
 - (6) $B_m := B_m \cup \{0^n \# y_1 \dots y_k \mid 0 < k < n\} \cup \{0^n \# y_1 \dots y_{n-1} \bar{y}_n\}$;
 - (7) **if** 0^n is not accepted by $\text{NP}_i^{B_m}$ **then**
 - (8) $B_m := B_m \cup \{0^n \# y\}$
- end;**

First, the construction must be effective to guarantee that sets A and B are recursive. In (4), the query set of a time-bounded machine with any oracle contains only strings of finite length, so the condition is decidable. In (5), there are 2^n strings of length n , but $\|Q(\text{NP}_i, 0^n)\| \leq p_j(n) < 2^n$ by (2) and (4). In (7), B_m is a finite oracle, so the condition is decidable. Second, at each stage $m = \langle i, j \rangle$, either $\|Q(\text{NP}_i, 0^n)\| > p_j(n)$ by (4) or $0^n \in A$ if and only if 0^n is not accepted by NP_i^B . The latter is true because $0^n \# y$ can be added in (8) without affecting the result in (7) since $0^n \# y$ is not queried; lines (2) and (3) guarantee that no strings added at any later stage will affect the current stage; and the net effect of adding strings to B_m in (6) and (8) is to cause 0^n to be in or out of set A as desired. If B is constructed as above and $A = \{x \mid x \text{ is accepted by the deterministic positive reduction procedure using oracle } B\}$, then $A \not\leq_{\text{tt}}^{\text{NP}} B$. ■

Corollary 3.1. There exists a recursive set B such that

- (1) $\text{P.ptt}(B) \subset \text{P.pos}(B)$.
- (2) $\text{P.tt}(B) \subset \text{P.T}(B)$.
- (3) $\text{NP.ptt}(B) \subset \text{NP.pos}(B)$.
- (4) $\text{NP.tt}(B) \subset \text{NP.T}(B)$.

Proof. Construct sets A and B as in Theorem 3.1. $A \in \text{P.pos}(B)$ and $A \notin \text{NP.tt}(B)$. ■

Theorem 3.2. For every $k \geq 1$ there exist recursive sets A and B such that $A \leq_{k+1\text{-ptt}}^{\text{P}} B$ and $A \not\leq_{k\text{-tt}}^{\text{NP}} B$.

Proof. Consider the following reduction procedure using oracle set B .

```

input  $x$ ;
for  $i := 0$  to  $k$  do
    if  $x\#1^i \in B$  then accept  $x$ ;
reject  $x$ 

```

This reduction procedure is deterministic, polynomially time-bounded, and positive; moreover, the query set of this procedure on any input contains exactly $k + 1$ strings. If $A = \{x \mid x \text{ is accepted by the above procedure using oracle } B\}$, then $A \leq_{k+1\text{-ptt}}^P B$. To guarantee $A \not\leq_{k\text{-tt}}^{\text{NP}} B$, an oracle set B must be effectively constructed so that no polynomially time-bounded k -question non-adaptive oracle Turing machine correctly witnesses the reduction. Since the construction is similar to Theorem 3.1, an exact construction is omitted. Intuitively, such an effective construction is possible since any oracle Turing machine that makes at most k queries can be “fooled” by placing an unqueried string in or out of B so that the procedure above is affected but the oracle Turing machine is not affected. ■

Corollary 3.2. For every $k \geq 1$ there exists a recursive set B such that

- (1) $\text{P}.k\text{-ptt}(B) \subset \text{P}.k+1\text{-ptt}(B)$.
- (2) $\text{P}.k\text{-tt}(B) \subset \text{P}.k+1\text{-tt}(B)$.
- (3) $\text{NP}.k\text{-ptt}(B) \subset \text{NP}.k+1\text{-ptt}(B)$.
- (4) $\text{NP}.k\text{-tt}(B) \subset \text{NP}.k+1\text{-tt}(B)$.

Proof. Construct sets A and B as in Theorem 3.2. $A \in \text{P}.k+1\text{-ptt}(B)$ and $A \notin \text{NP}.k\text{-tt}(B)$. ■

Theorem 3.3. There exist recursive sets A and B such that $A \leq_{1\text{-ptt}}^P B$ and $A \not\leq_m^{\text{NP}} B$.

Proof. Let $A = \Sigma^*$ and $B = \emptyset$. $A \leq_{1\text{-ptt}}^P B$ via a machine that makes one query then accepts, ignoring the query. If the oracle set of a many-one reduction is empty, then no strings are accepted; therefore, only the empty set is many-one reducible to the empty set in polynomial time. ■

Corollary 3.3. There exists a recursive set B such that

- (1) $\text{P}.m(B) \subset \text{P}.1\text{-ptt}(B)$.
- (2) $\text{NP}.m(B) \subset \text{NP}.1\text{-ptt}(B)$.

Theorem 3.4. There exist recursive sets A and B such that $A \leq_{1\text{-tt}}^P B$ and $A \not\leq_{\text{pos}}^{\text{NP}} B$.

Proof. Consider the following reduction procedure using oracle set B .

```

input  $x$ ;
if  $x \in B$  then reject  $x$ 
else accept  $x$ 

```

This reduction procedure is deterministic, polynomially time-bounded, and 1-question non-adaptive. Define set A in terms of oracle set B to be $\{x \mid x \text{ is accepted by the above procedure using oracle } B\}$. By definition, $A \leq_{1\text{-tt}}^P B$. To guarantee $A \not\leq_{\text{pos}}^{\text{NP}} B$, an oracle set B will be effectively constructed so that no polynomially time-bounded positive

oracle Turing machine correctly witnesses the reduction. That is, for each polynomially time-bounded oracle Turing machine NP_i , either NP_i is not positive or some input string x will witness $x \in A$ if and only if x is not accepted by NP_i^B . Each condition precludes a correct positive reduction. The construction of B will proceed in stages. At each stage i , a finite initial segment B_i of B will be constructed such that $B_i \supseteq B_{i-1}$.

stage 0:

$$\begin{aligned} B_0 &:= \phi; \\ n_0 &:= 0; \end{aligned}$$

stage $i > 0$:

- (1) $B_i := B_{i-1}$;
- (2) $n := n_{i-1} + 1$;
- (3) $n_i := p_i(n)$;
- (4) $B_{full} := \{y \mid |y| \leq n_i\}$;
- (5) **if** 0^n is accepted by $NP_i^{B_i}$ **then**
- (6) **if** $\exists C, B_i \subseteq C \subseteq B_{full}$ and 0^n is not accepted by NP_i^C **then**
- (7) $B_i := B_i \cup \{0^n\}$;

First, the construction must be effective to guarantee that sets A and B are recursive. In (5) and (6), B_i and its supersets C to be considered are finite, so the condition is decidable. Second, at each stage i , either NP_i is not positive if the condition in (6) is false or $0^n \in A$ if and only if 0^n is not accepted by NP_i^B . The latter can happen in two ways; Either the condition in (5) is false and $0^n \in A$ since it is never added to B , or (7) is executed causing $0^n \notin A$. 0^n can be added in (7) without affecting the result in line (5) since NP_i is “positive” (over the supersets of B_i that can possibly be queried on input 0^n) by (6). Lines (2) and (3) ensure that no strings added at any later stage will affect the current stage. If B is constructed as above and $A = \{x \mid x \text{ is accepted by the 1-question truth-table reduction procedure using oracle } B\}$, then $A \not\leq_{\text{pos}}^{\text{NP}} B$. ■

Corollary 3.4. There exists a recursive set B such that

- (1) $P.k\text{-ptt}(B) \subset P.k\text{-tt}(B)$, all $k \geq 1$.
- (2) $P.\text{bptt}(B) \subset P.\text{btt}(B)$.
- (3) $P.\text{ptt}(B) \subset P.\text{tt}(B)$.
- (4) $P.\text{pos}(B) \subset P.T(B)$.
- (5) $NP.k\text{-ptt}(B) \subset NP.k\text{-tt}(B)$, all $k \geq 1$.
- (6) $NP.\text{bptt}(B) \subset NP.\text{btt}(B)$.
- (7) $NP.\text{ptt}(B) \subset NP.\text{tt}(B)$.
- (8) $NP.\text{pos}(B) \subset NP.T(B)$.

Proof. Construct sets A and B as in Theorem 3.4. $A \in P.1\text{-tt}(B)$ and $A \notin NP.\text{pos}(B)$. ■

$$\begin{array}{l}
 \text{—————} \quad \forall B[X(B) \subseteq Y(B)] \\
 \text{—————} \quad \exists B[X(B) \subset Y(B)]
 \end{array}$$

Figure 2

Each of the theorems in §3 separates two deterministic or two nondeterministic relative complexity classes and Figure 2 summarizes these results. Theorem 3.1 separates the non-adaptive reduction classes (the leftmost four slices) from those which may be adaptive (the rightmost face). Theorems 3.2 and 3.3 separate the leftmost four slices. Theorem 3.4 separates the positive reduction classes (the back face) from those which may not be positive (the front face).

4. Positive Relativizations

The results in §3 distinguish either deterministic or nondeterministic reducibilities. Here we study whether or not a given deterministic reducibility and its nondeterministic analogue are distinct. That is, for a reducibility r , is there an oracle set B such that $P.r(B) \neq NP.r(B)$? We will show for various reducibilities r that there is such an oracle if and only if $P \neq NP$. Such relativizations originate in Book [2,4] and are called positive relativizations. This notion should not be confused with positive reducibility; the term positive relativization is used here solely for describing a reducibility r such that $P = NP \iff \forall B[P.r(B) = NP.r(B)]$. We will show that the non-adaptive reducibilities lead to positive relativizations, but the adaptive reducibilities are distinct, regardless of whether or not $P \neq NP$. The reduction procedure in Theorem 4.1 was used by Baker, Gill and Solovay [1] to prove the equivalent of our Corollary 4.1(2). We use it to show that the deterministic and nondeterministic positive reducibilities are distinct as well.

Theorem 4.1. There exist recursive sets A and B such that $A \leq_{\text{pos}}^{\text{NP}} B$ and $A \not\leq_{\text{T}}^{\text{P}} B$.

Proof. Consider the following reduction procedure using oracle set B .

```

input  $x$ ;
 $w := x\#$ ;
for  $k := 1$  to  $|x|$  do
    if true  $\rightarrow w := w0$ 
    ■ true  $\rightarrow w := w1$ 
    fi;
if  $w \in B$  then accept  $x$ 
    else reject  $x$ 

```

This reduction procedure is polynomially time-bounded; but it is nondeterministic and x is accepted if any computation on input x accepts. It is also positive since adding strings to B will not cause any input to be rejected that was previously accepted. If $A = \{x \mid x \text{ is accepted by the above procedure using oracle } B\}$, then $A \leq_{\text{pos}}^{\text{NP}} B$. To guarantee $A \not\leq_{\text{T}}^{\text{P}} B$, an oracle set B must be effectively constructed so that no polynomially time-bounded deterministic oracle Turing machine correctly witnesses the reduction. Intuitively, this is possible because the nondeterministic reduction has an exponentially large query set while any polynomially time-bounded deterministic oracle Turing machine has at most a polynomially large query set size for a specific oracle set. The construction of B will proceed in stages. At each stage i , a finite initial segment B_i of B will be constructed such that $B_i \supseteq B_{i-1}$.

stage 0:

$B_0 := \phi$;
 $n_0 := 0$;

stage $i > 0$:

- (1) $B_i := B_{i-1}$;
- (2) $n :=$ the least integer such that $n > n_{i-1}$ and $p_i(n) < 2^n$;
- (3) $n_i := p_i(n)$;
- (4) $y :=$ some string such that $|y| = n$ and $0^n \# y \notin Q(P_i, B_i, 0^n)$;
- (5) **if** 0^n is not accepted by $P_i^{B_i}$ **then**
- (6) $B_i := B_i \cup \{0^n \# y\}$;

First, the construction must be effective to guarantee that sets A and B are recursive. In (4), there are 2^n strings of length n , but $\|Q(P_i, B_i, 0^n)\| \leq p_i(n) < 2^n$ by (2). In (5), B_i is a finite oracle, so the condition is decidable. Second, at each stage i , $0^n \# y$ can be added without affecting the result in (5) since $0^n \# y$ is not queried; lines (2) and (3) guarantee that no strings added at any later stage will affect the current stage. If B is constructed as above and $A = \{x \mid x \text{ is accepted by the nondeterministic reduction procedure using oracle } B\}$, then $A \not\leq_{\text{T}}^{\text{P}} B$. ■

Corollary 4.1. There exists a recursive set B such that

- (1) $\text{P.pos}(B) \subset \text{NP.pos}(B)$.
- (2) $\text{P.T}(B) \subset \text{NP.T}(B)$.

Proof. Construct sets A and B as in Theorem 4.1. $A \in \text{NP.pos}(B)$ and $A \notin \text{P.T}(B)$. ■

Definition 4.1. A reducibility r is *non-adaptive* if for every set A and B , $A \leq_r^{\text{P}} B$ and $A \leq_r^{\text{NP}} B$ are witnessed by non-adaptive oracle Turing machines, if at all.

The technique used in the following theorem is from Theorem 5.3(A) of Book, Long and Selman [3], which is equivalent to our Corollary 4.2(7). We apply the technique to establish that each of the non-adaptive reducibilities provides a positive relativization of the $\text{P} = ?\text{NP}$ problem.

Theorem 4.2. Let r be a polynomially time-bounded non-adaptive reducibility. If there exists a set B such that $\text{P}.r(B) \subset \text{NP}.r(B)$, then $\text{P} \subset \text{NP}$.

Proof. The method is to show that every polynomially time-bounded nondeterministic r -reduction can be deterministically simulated under the assumption $\text{P} = \text{NP}$. Given non-adaptive oracle Turing machine NP_i , its query set $Q(\text{NP}_i, x)$ on input x is enumerated by a function $f \in \text{NPSV}$. If $\text{P} = \text{NP}$, then $f \in \text{PSV}$ since every nondeterministic transducer can be deterministically simulated. NP_i^B on input x can be simulated deterministically in polynomial time as follows:

- (1) Compute $f(x)$ deterministically in polynomial time and write the output $Q(\text{NP}_i, x)$ on a work tape.
- (2) On another work tape, list the words in $Q(\text{NP}_i, x) \cap B$ by reading from work tape (1) and querying oracle B .
- (3) Simulate NP_i^B without further queries by replacing potential queries with a search of work tape (2).

Step (3) can be achieved deterministically under the assumption $\text{P} = \text{NP}$ because no queries are made. ■

Corollary 4.2. $\text{P} \subset \text{NP}$ if and only if there exists a set B such that

- (1) $\text{P.m}(B) \subset \text{NP.m}(B)$.
- (2) $\text{P}.k\text{-ptt}(B) \subset \text{NP}.k\text{-ptt}(B)$, any $k \geq 1$.
- (3) $\text{P}.k\text{-tt}(B) \subset \text{NP}.k\text{-tt}(B)$, any $k \geq 1$.
- (4) $\text{P.bptt}(B) \subset \text{NP.bptt}(B)$.
- (5) $\text{P.btt}(B) \subset \text{NP.btt}(B)$.
- (6) $\text{P.ptt}(B) \subset \text{NP.ptt}(B)$.
- (7) $\text{P.tt}(B) \subset \text{NP.tt}(B)$.

Proof. Each of these reducibilities is non-adaptive, so the implication from right to left is proven by Theorem 4.2. Conversely, assume $\text{P} \subset \text{NP}$. $\text{P}.r(\phi) \subset \text{NP}.r(\phi)$ for each reducibility r in (2)–(7). Pathologically, ϕ (or Σ^*) will not do for many-one reducibility, but any finite set B will suffice. ■

Figure 3 summarizes the results in §3 and §4.

$$\begin{array}{l}
 \text{—————} \quad \exists B[X(B) \subset Y(B)] \\
 \text{———} \text{———} \quad \exists B[X(B) \subset Y(B)] \iff P \subset NP
 \end{array}$$

Figure 3

References

- [1] T. Baker, J. Gill and R. Solovay, “Relativizations of the P =? NP question,” *SIAM J. Comput.* 4 (1975), 431–442.
- [2] R. V. Book, “Bounded query machines: on NP and PSPACE,” *Theor. Comput. Sci.* 15 (1981), 27–39.
- [3] R. V. Book, T. J. Long and A. L. Selman, “Quantitative relativizations of complexity classes,” *SIAM J. Comput.* 13 (1984), 461–487.
- [4] R. V. Book and C. Wrathall, “Bounded query machines: on NP() and NPQUERY(),” *Theor. Comput. Sci.* 15 (1981), 41–50.
- [5] S. A. Cook, “The complexity of theorem-proving procedures,” *Proc. 3rd Annual ACM Symposium on Theory of Computing* (1971), 151–158.
- [6] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., San Francisco, 1979.
- [7] R. M. Karp, “Reducibility among combinatorial problems,” in *Complexity of Computer Computations* (Miller and Thatcher, eds.), Plenum Press, New York, 1973.
- [8] R. E. Ladner, N. A. Lynch and A. L. Selman, “A comparison of polynomial time reducibilities,” *Theor. Comput. Sci.* 1 (1975), 103–123.
- [9] A. L. Selman, “Reductions on NP and P-selective sets,” *Theor. Comput. Sci.* 19 (1982), 287–304.