Independence of $P$ vs. $NP$ in regards to oracle relativizations.

JERRALD MEEK

This is the third article in a series of four articles dealing with the $P$ vs. $NP$ question. The purpose of this work is to demonstrate that the methods used in the first two articles of this series are not affected by oracle relativizations. Furthermore, the solution to the $P$ vs. $NP$ problem is actually independent of oracle relativizations.

Categories and Subject Descriptors: F.2.0 [Analysis of Algorithms and Problem Complexity]: General
General Terms: Algorithms, Theory
Additional Key Words and Phrases: P vs NP, Oracle Machine

1. INTRODUCTION.

Previously in “$P$ is a proper subset of $NP$” [Meek Article 1 2008] and “Analysis of the Deterministic Polynomial Time Solvability of the 0-1-Knapsack Problem” [Meek Article 2 2008] the present author has demonstrated that some $NP$-complete problems are likely to have no deterministic polynomial time solution. In this article the concepts of these previous works will be applied to the relativizations of the $P$ vs. $NP$ problem.

It has previously been shown by Hartmanis and Hopcroft [1976], that the $P$ vs. $NP$ problem is independent of Zermelo-Fraenkel set theory. However, this same discovery had been shown by Baker [1979], who also demonstrates that if $P = NP$ then the solution is incompatible with $ZFC$.

The purpose of this article will be to demonstrate that the oracle relativizations of the $P$ vs. $NP$ problem do not preclude any solution to the original problem. This will be accomplished by constructing examples of five out of the six relativizing oracles from Baker, Gill, and Solovay [1975], it will be demonstrated that inconsistent relativizations do not preclude a proof of $P \neq NP$ or $P = NP$. It will then become clear that the $P$ vs. $NP$ question is independent of oracle relativizations.

2. PRELIMINARIES.

Baker, Gill, and Solovay [1975] have previously shown that there are six types of oracles that can be constructed to examine the $P$ vs. $NP$ problem. These include

(1) Oracle $A$ such that $P^A = NP^A$.
(2) Oracle $B$ such that $P^B \neq NP^B$.
(3) Oracle $C$ such that $NP^C$ is not closed under complementation.

People who wish to remain anonymous have offered comments and suggestions which have improved this work. The author wishes to express his appreciation for their assistance.

Jerrald Meek Copyright ©2008
(4) Oracle $D$ such that $P^D \neq NP^D$ but $NP^D$ is closed under complementation.

(5) Oracle $E$ such that $P^E \neq NP^E$ and $P^E = NP^E \cap \text{co-NP}^E$.

(6) Oracle $F$ such that $P^F \subseteq [NP^F \cap \text{co-NP}^F] \subseteq NP$.

Theorem 4.4 from P is a proper subset of NP. [Meek Article 1 2008] 2.1.
P = NP Optimization Theorem.

The only deterministic optimization of a NP-complete problem that could prove $P = NP$ would be one that can always solve a NP-complete problem by examining no more than a polynomial number of input sets for that problem.

Definition 2.1. Oracle Machine

An oracle machine can be deterministic or non-deterministic. The difference between an oracle machine and a regular Turing Machine is that an oracle Machine has a yes state, a no state, and a query state. When the oracle machine is placed in the query state after passing a string to the oracle, if that string is an element of the oracle set, then the oracle will place the Turing Machine in the yes state, otherwise it will place the Turing Machine in the no state. The processing time required by the oracle can be considered instantaneous.

2.1 Encoding Methods.

In this article, two encoding methods will be used. One encoding method will encode the input sets for a problem and is identical to that used by Baker, Gill, and Solovay [1975]. This will be identified as “input encoding.”

The second method of encoding will be used to identify a partition of the set of all possible input sets for a problem. This method of encoding will be identified as “partition encoding.”

2.1.1 Partition Encoding. The concept of partition encoding is not to encode the input sets for a problem, but an entire partition of the set of all possible input sets. However, for this method to work the partitioning must be identified with a specific problem.

The method of partition encoding used in this work will divide partitions by the number of literals in an input set which have a true value. If a problem has $n$ literals, then there will be $n + 1$ partitions (one partition has no true literals).

Notice that the problem $[a \lor b \lor c]$ has different input sets that result in true evaluations from the problem $[\neg a \lor b \lor c]$ even though the cardinality of these input sets are equivalent.

It will then be necessary that the problem which a partition applies to will need to be encoded. This can be accomplished by the following method:

—Let $P$ be a NP-Complete problem with $k$ literals.
—Let $g$ be the Gödel Number calculated from $P$.
—Let $\sigma$ be a set of strings representing each of the $k + 1$ partitions for $P$.
—Let $1 \leq i \leq k + 1$.
—Let $S$ be the set of all possible input sets for $P$.
—$T(x) =$ the number of true literals in input set $x$. 

ArXiv, Vol. V, No. N, Month 20YY.
Independence of $P$ vs. $NP$ in regards to oracle relativizations.

then

$$\forall i : (\sigma_i \equiv \langle i - 1, g \rangle)$$

and the partition encoding of $S_i$ is represented by:

$$\sigma_{T(S_i)}$$

For each partition of $P$ there will be one string consisting of:

1. a number in the range of 0 to $k$ representing the number of \textit{true} literals in $S_i$;
2. and the Gödel Number of $P$.

3. THE RELATIVIZING ORACLES.

Baker, Gill, and Solovay [1975] have shown that the classes of $P$ and $NP$ do not relativize in a consistent manner. The inconsistency of the relativizations of $P$ and $NP$ has often been used to justify arguments that the $P$ vs. $NP$ problem is unusually difficult at best, or maybe even impossible.

By slightly altering the machine model, we can obtain differing answers to the relativized question. This suggests that resolving the original question requires careful analysis of the computational power of machines. It seems unlikely that ordinary diagonalization methods are adequate for producing an example of a language in $NP$ but not in $P$; such diagonalizations, we would expect, would apply equally well to the relativized classes, implying a negative answer to all relativized $P = ? NP$ questions, a contradiction. [Baker et al. 1975]

Although it is generally accepted that diagonalization is not an effective method for solving the $P$ vs. $NP$ problem, it is the purpose of this article to demonstrate that oracle relativizations in no way prevent a solution to the $P$ vs. $NP$ problem.

It is also worthwhile to mention that the six oracles of Baker, Gill, and Solovay [1975] do not represent an exclusive set of all oracle relativizations. There has been extensive work to find other unusual relativizations of complexity classes in regards to the $P$ vs. $NP$ question. For example, Bennett and Gill [1981] have found that random oracles can produce the relativization $P^A \neq NP^A \neq co-NP^A$. However, the scope of this work will be limited to the six oracles from Baker, Gill, and Solovay [1975] with the expectation that what is learned from these six oracles may be applied to any other relativizing oracle.

3.1 A $P = NP$ Oracle.

The $P = NP$ Optimization Theorem requires that no more than a polynomial number of input sets can be examined by a deterministic polynomial time search algorithm. By the same logic it is obvious that a deterministic oracle machine can not request more than a polynomial number of queries in polynomial time.

Baker, Gill, and Solovay [1975] define the oracle $A$ such that $A = \{ \langle i, x, 0^n \rangle : NP^A \text{ accepts } x \text{ in } < n \text{ steps} \}$.

This definition produces an oracle such that $P^A = NP^A$. However, this definition does not really say anything about how the oracle functions. In this work, the oracle will be created by partition encoding strings, a string will be included in oracle $A$
if there exists at least one element of the partition represented by the string which results in the problem associated to the string evaluating \textit{true}.

—Let \( L(X) = \{ x : \text{there exists } y \in X \text{ such that } |y| = |x| \} \) for any oracle \( X \).

—Let \( x \) be the set of all problems in \( L(X) \).

—Let \( S \) be the set such that \( S_i \) is the set of all possible input sets for \( x_i \) when \( 1 \leq i \leq |x| \).

The oracle set is created by the following algorithm.

(1) Let \( i = 1 \).

(2) Let \( e = 1 \).

(3) If \( x_i \) evaluates \textit{true} with input \( S_i \), then add the partition encoding of \( S_i \rightarrow x_i \) to set \( A \), and let \( e = |S_i| \).

(4) If \( e < |S_i| \), then increment \( e \) and continue at step 3.

(5) If \( e = |S_i| \) and \( i < |x| \), then increment \( i \) and continue at step 2.

(6) If \( e = |S_i| \) and \( i = |x| \), then the set \( A \) has been found.

The goal was to produce an example of an oracle set such that \( P^A = NP^A \). Here, the oracle \( A \) is actually such that \( P^A = NP \). Such an oracle would also create the condition that \( P^A = NP^A \). Another method could be used to create an oracle such that \( P^X = \text{PSPACE} \), which would also satisfy \( P^X = NP^X \), except that \( NP \neq NP^X \). There are multiple such oracles, however according to Mehlhorn [1973] the set of computable oracles such that \( P^A = NP^A \) is meager.

Notice that in this example the set of all possible input sets will be exponential in size, and therefore some partitions for any problem will be exponential in size. It is then easy to see that the problem of creating the oracle set (even when the oracle is created to only work for one problem) is a member of the \( \text{FNP} \) complexity class (when \( P^A = NP \)).

The process that a Deterministic Oracle Machine with oracle set \( A \) uses to solve problem \( x_i \) which is in \( NP \) and has \( n \) literals is the algorithm.

(1) Let \( e = 0 \).

(2) Encode the partition for problem \( x_i \) for input sets having \( e \) \textit{true} literals.

(3) Pass the partition encoding from step 2 to the oracle and enter the query state.

(4) If the machine is in the yes state then halt in an accepting state.

(5) If the machine is in the no state and \( e < n \) then increment \( e \) and continue at step 2.

(6) If the machine is in the no state and \( e = n \) then halt in a non-accepting state.

The algorithm of the oracle machine solves any \( NP \) problem in polynomial time on a Deterministic Oracle Machine. A Non Deterministic Oracle Machine could also use this algorithm to solve the same set of problems in polynomial time. Therefore, \( P^A = NP^A \).

This method is dependant on an oracle set creation algorithm requiring deterministic exponential time for each problem that the oracle is capable of solving. If so desired, the time required to compute the oracle set could be treated as separate to the process of solving the problem.
Another option is that the Deterministic Oracle Machine could be networked to both an oracle and a Non Deterministic Turing Machine. In this model the non deterministic machine generates the oracle set which it sends to the oracle. This method will allow the entire process to require polynomial time.

A third method is that we simply assume that the oracle set somehow creates itself magically.

Regardless of the method used to generate the oracle set, all three methods involve somehow concealing the majority of the work. This does not in any way reduce the complexity of the problem; it only removes the difficulty of the problem by placing the majority of the computational burden upon the process of oracle set creation.

In this example the oracle $A$ was created such that $P^A = NP$. It would have been valid to create oracle set $A$ such that $P^A = PSPACE$. However, it can not be expected that such an oracle would be any easier to create.

3.2 A $P \neq NP$ Oracle.

The method of producing oracle set $B$ as described by Baker, Gill, and Solovay [1975] is similar to the algorithm:

1. Let $L(X) = \{x : \text{there exists } y \in X \text{ such that } |y| = |x|\}$ for any oracle $X$.
2. Let $x$ be the set of all problems in $L(X)$.
3. Let $S$ be the set such that $S_i$ is the set of all possible input sets for $x_i$ when $1 \leq i \leq |x|$.
4. Let $B(i)$ represent the elements of $B$ placed in $B$ prior to stage $i$.
5. Let $p_i(n) < |S_i|$ be a polynomial function of $n$ that represents the number of computations preformed by $P_i^X$ and $NP_i^X$ for all oracles $X$. If $p_i(n)$ computations have been preformed, and no accepting element of $S_i$ has been found, then the algorithm halts in the non-accepting state without examining all elements of $S_i$.

Create the oracle with the algorithm.

1. Let $i = 1$.
2. If $P_i^B(i)$ rejects $x_i$ then the next unexamined element of $S_i$ is an element of $B$.
3. If $i < |x|$ then increment $i$ and continue at step 2.
4. If $i = |x|$ then the oracle set $B$ has been found.

The process used by an oracle machine with oracle set $B$ for finding a solution to a problem $x_i$ in $NP$ with $n_i$ literals is preformed by the following algorithm.

1. Search for an element of $S_i$ which results in $x_i$ evaluating true. Terminate the search if the time limit $p_i(n)$ has been reached.
2. If a solution to $x_i$ was found then, halt in an accepting state.
3. If no solution to $x_i$ was found and all elements of $S_i$ were examined in step 1, then halt in a non accepting state.
4. If no solution to $x_i$ has been found and not all elements of $S_i$ have been examined, then query the oracle $B$ with an unexamined element of $S_i$.
5. If the machine is in the yes state then halt in an accepting state.
(6) If the machine is in the no state then halt in a non accepting state.

This algorithm will allow for a polynomial time solution on a Non Deterministic Oracle Machine $NP^B$ because such a machine will always find a solution in step one for any problem in $NP$ if one exists, and halt in step 2 or 3.

However, a Deterministic Oracle Machine may not find the solution in step 1. When a deterministic machine queries the oracle the oracle only contains one set of the proper cardinality which may or may not be an accepting input set (because the result of the input set is not a requirement for membership in $B$). The oracle will then place the machine in the yes state if the input set passed to the oracle is equivalent to the one input set of proper cardinality that is an element of $B$. It is then the case that the Deterministic Oracle Machine may terminate in an accepting state when there is no accepting input to the problem. On the other hand if the deterministic oracle machine fails to find a solution in step 1, and the input set passed to the oracle is not an element of $B$, then the machine may terminate in a non-accepting state when an accepting input set may exist. It is then clear that oracle $B$ does not produce a reliable polynomial time solution to the problem $x_i$.

It is then true that $P^B \neq NP^B$. However this is only because the oracle $B$ is dysfunctional. Oracle $B$ does not actually help a Non Deterministic Oracle Machine to solve a $NP$ problem, because the non deterministic machine has the ability to solve the problem without entering the query state. The oracle is then never queried, and the dysfunctionality of the oracle does not interfere with the operation of the Non Deterministic Oracle Machine. However, a Deterministic Oracle Machine requires the assistance of the oracle to overcome the $P = NP$ Optimization Theorem. It is then the case that a dysfunctional oracle will prevent a Deterministic Oracle Machine from solving a problem in polynomial time while not affecting the performance of a Non Deterministic Oracle Machine.

It then becomes easy to see why $P^A = NP^A$ oracle sets are meager [Mehlhorn 1973]. The oracle set must be carefully fabricated in order to create such a condition. If great care has not been taken to ensure that the oracle will be of assistance to a deterministic machine, then the oracle will be dysfunctional.

3.3 An Oracle such that $NP$ is not closed under complementation.

A complexity class is closed under complementation if the complements of all problems in that class are members of the same class. Baker, Gill, and Solovay [1975] have shown that there exists an oracle such that one or more problems in $NP^C$ have a complement that is not in $NP^C$.

— Let $L(X) = \{x : \text{there exists } y \in X \text{ such that } |y| = |x|\}$ for any oracle $X$.
— Let $x$ be the set of all problems in $L(X)$.
— Let $S$ be the set such that $S_i$ is the set of all possible input sets for $x_i$ when $1 \leq i \leq |x|$.
— Let $C(i)$ represent the elements of $C$ placed in $C$ prior to stage $i$.
— Let $p_i(n) < |S_i|$ be a polynomial function of $n$ that represents the maximum number of computations preformed by $P_i^X$ and $NP_i^X$ for all oracles $X$. If $p_i(n)$ computations have been preformed, and no accepting element of $S_i$ has been
found, then the algorithm halts in the non-accepting state without examining all elements of $S_i$.

Create the oracle with the algorithm.

1. Let $i = 1$.
2. If $NP^C(i)$ accepts $x_i$, then any one element of $S_i$ that is accepted by $x_i$ is added to the set $C$.
3. If $i < |x|$ then increment $i$ and continue at step 2.
4. If $i = |x|$ then the oracle set $C$ has been found.

The set $C$ is a set containing exactly one accepting input set for each problem described by $L(X)$ that has an accepting input set.

The process of finding a solution to a problem $x_i$ which is in $NP$ with oracle set $C$ is preformed with the following algorithm.

1. Pass each element of $S_i$ to the oracle and enter the query state.
2. If the machine is in the yes state for any query, then halt in an accepting state.
3. If the machine is in the no state for all queries, then halt in a non accepting state.

For a Non Deterministic Oracle Machine, all elements of $S_i$ can be evaluated simultaneously. However, a Deterministic Oracle Machine must iterate threw them one at a time. It is then easy to see that $PC \neq NPC$.

To create the oracle set $\bar{C}$ the compliment of $C$, use the algorithm.

1. Let $i = 1$.
2. If $NP^{\bar{C}}(i)$ rejects $x_i$, then add all elements of $S_i$ to the set $\bar{C}$.
3. If $i < |x|$ then increment $i$ and continue at step 2.
4. If $i = |x|$ then the oracle set $\bar{C}$ has been found.

With the oracle set $\bar{C}$ a Deterministic Oracle Machine can solve the compliment of any $NP$ problem by performing one query. This is because $\bar{C}$ contains all input sets for any problem in $co-NP$ that has at least one input set that results in the problem evaluating true. If the oracle places the machine in the yes state for any input set then an accepting input set exists for the $co-NP$ problem, although the accepting input set may not be represented by the string queried. If the oracle places the machine in the no state then no accepting input set exists for the $co-NP$ problem. It is then the case that $PC \supset co-NP$.

However, this situation requires that the oracle sets $C$ and $\bar{C}$ are created non deterministically. It is then the case that the work of solving these problems has been done by the creation of the oracle sets, and this allows the Deterministic Oracle Machine to avoid the limitations of the $P = NP$ Optimization Theorem when solving $co-NP$ problems.

3.4 A $P \neq NP$ oracle such that $NP$ is closed under complementation.

—Let $L(X) = \{x : \text{there exists } y \in X \text{ such that } |y| = |x|\}$ for any oracle $X$.
—Let $x$ be the set of all problems in $L(X)$. 

ArXiv, Vol. V, No. N, Month 20YY.
—Let $S$ be the set such that $S_i$ is the set of all possible input sets for $x_i$ when $1 \leq i \leq |x|$
—Let $D(i)$ represent the elements of $D$ placed in $D$ prior to stage $i$. 
—Let $\bar{D}(i)$ represent the elements of $\bar{D}$ placed in $\bar{D}$ prior to stage $i$. 
—Let $p_i(n) < |S_i|$ be a polynomial function of $n$ that represents the maximum number of computations performed by $P^X_i$ and $NP^X_i$ for all oracles $X$. If $p_i(n)$ computations have been performed, and no accepting element of $S_i$ has been found, then the algorithm halts in the non-accepting state without examining all elements of $S_i$.

Construct the oracle sets $D$ and $\bar{D}$ with the algorithm.

(1) Let $i = 1$.
(2) Let $n = 1$.
(3) If $n$ is even, then let $e = 1$.
(4) If $n$ is even and $S_n \notin D(i)$ then find the prefix $u$ of $S_n$ with length $|S_n|/2$.
(5) If $n$ is even and $S_n \notin \bar{D}(i)$ and $u$ is an input set for $x_u$ and $NP_i^{D(n)}$ rejects $x_u$, then $S_n \in D$.
(6) If $n$ is even and $S_n \notin \bar{D}(i)$ and $e < |S_n|$ then increment $e$ and continue at step 4.
(7) If $n$ is even and $S_n \notin D(i)$ and $e = |S_n|$ then break to step 10.
(8) If $n$ is odd and all elements of $\bar{D}(n)$ have cardinality less than $n$, and $p_i(n) < 2^{(n-1)/2}$, then add to $\bar{D}$ all strings queried by $P_i^{D(n)}$ for problem $x_n$. If $P_i^{D(n)}$ rejects $x_n$, then add to $D$ the next element of $S_n$ not queried.
(9) If $n$ is odd, then increment $i$.
(10) If $n < |x|$, then increment $n$ and continue at step 3.
(11) If $n = |x|$, then the sets $D$ and $\bar{D}$ have been found.

In steps 4 and 5, elements are added to $D$ if their prefix is rejected. Notice that if $f(x)$ has no input set resulting in a true evaluation, then $f(x) \land g(x)$ has no input set resulting in a true evaluation. It is then the case that in step 4 and 5, elements are only added to $D$ if they represent a problem that has no accepting input set.

In step 8, an input set is added to $D$ if $P_i^{D(n)}$ rejects the problem which that input set belongs to. However, on the first iteration, $D(n)$ will be an empty set. An empty oracle set is equivalent to having no oracle, and so the polynomial time algorithm will not check all possible input sets for the first problem examined. For the third problem examined, $D(n)$ may contain an input set if it belongs to a problem that always evaluates false. It is then easy to see that step 8 adds elements to $D$ that may or may not cause a problem to evaluate true.

Because the elements of $D$ do not have any consistent representation, it then follows that the set $D$ will result in a dysfunctional oracle. A deterministic machine requires a functional oracle set to be able to solve all $NP$ problems, but a non-deterministic machine does not rely upon the oracle set when solving $NP$ problems. It then follows that $P^D \neq NP^D$.

In step 8, the requirement that all elements of $\bar{D}(n)$ must have cardinality less than $n$ will always evaluate true. The statement $p_i(n) < 2^{(n-1)/2}$ will evaluate true
for the first values of \( n \) up to some limit. For problems less than this limit, a polynomial number of elements will be added to \( \hat{D} \). The elements will be added regardless of weather or not they result in a true evaluation of the co-NP problem associated to them. It is then clear to see that the set \( \hat{D} \) will also result in a dysfunctional oracle, and \( P^D \) will not produce reliable results for any co-NP problem.

3.5 A \( P \neq NP \) oracle such that \( P = [NP \cap co-NP] \).

Baker, Gill, and Solovay [1975] construct the oracle set \( E \) by adding elements to the oracle set \( A \) which was previously constructed such that \( P^A = NP^A \). The additional elements in \( E \) will result in \( P^E \neq NP^E \). The trick is to arrange the additional elements such that if a problem is in \( NP \), and the compliment of the problem is also in \( NP \), then both of these problems are solvable in polynomial time by the oracle \( P^E \).

In the description of creating oracle \( E \), it is allowed to start with “any oracle \( A \) such that \( P^A = NP^A \).” Here, the oracle \( A \) that was previously constructed in this article will be used. However, remember that oracle \( A \) did not contain strings representing input sets, but instead partition encoding was used. It will then be the case that the same encoding specification will need to be used for the additional elements of \( E \).

—Let \( L(X) = \{ x : \text{there exists } y \in X \text{ such that } |y| = |x| \} \) for any oracle \( X \).
—Let \( x \) be the set of all problems in \( L(X) \).
—Let \( S \) be the set such that \( S_i \) is the set of all possible input sets for \( x_i \) when \( 1 \leq i \leq |x| \).
—Let \( e(n) \) be a function such that \( e(0) = 0 \) and \( e(x) = 2^{2e(x-1)} \leftrightarrow x > 1 \).
—Let \( E(0) = A \) and let \( E(n) \) be the set of elements placed in \( E \) prior to stage \( n \).
—Let \( r \) be the set of all possible values for \( \langle j, k \rangle \leftarrow j \in N, k \in N, j \neq k \).
—Let \( r_{j(i)} \) denote the \( j \) element of set \( r_i \), and \( r_{k(i)} \) denote the \( k \) element of \( r_i \).
—Let \( p_i(n) < |S_i| \) be a polynomial function of \( n \) that represents the maximum number of computations preformed by \( P^X_i \) and \( NP^X_i \) for all oracles \( X \). If \( p_i(n) \) computations have been preformed, and no accepting element of \( S_i \) has been found, then the algorithm halts in the non-accepting state without examining all elements of \( S_i \).

Create the oracle set \( E \) with the algorithm.

1. Let \( n = 1 \).
2. Let \( i = 1 \).
3. Let \( l \in S_n \).
4. If \( e(n-1) < (\log_2 |l|) \leq e(n) \leq \max \{ p_{r_{j(i)}}(l), p_{r_{k(i)}}(l) \} < e(n+1) \) and neither \( NP^E_{r_{j(i)}} \) or \( NP^E_{r_{k(i)}} \) accepts \( x_n \), then \( \langle j, k \rangle \) is unsatisfied.
5. If \( p_i(e(n)) \geq 2^{e(n)} \), then \( \langle i, i \rangle \) is unsatisfied.
6. If \( \langle j, k \rangle \) was not determined unsatisfied in step 4, and \( \langle i, i \rangle \) was not determined unsatisfied in step 5, then increment \( i \).
(7) If \((j, k)\) was not determined unsatisfied in step 4, and \((i, i)\) was determined unsatisfied in step 5, and \(F_i^{E(n)}\) rejects \(x_n\), then add to \(E\) the encoding of \(x_n\) with the next unevaluated element of \(S_n\).

(8) If \((j, k)\) was not determined unsatisfied in step 4, and \((i, i)\) was determined unsatisfied in step 5, then increment \(i\).

(9) If \(i < |N|\) then increment \(n\) and continue at step 3.

(10) If \(i = |N|\) then the set \(E\) has been found.

Notice that when \((i, i)\) is unsatisfied, then \(P_i^{E(n)}\) will be run on \(x_n\). The set \(E\) \((n)\) will only contain elements relevant to \(x_n\) if it was inherited from set \(A\). It is then the case that \(P_i^{E(n)}\) will only reject \(x_n\) when \(x_n\) has no accepting input set. In step 7, if \((i, i)\) is unsatisfied, and \(P_i^{E(n)}\) rejects \(x_n\), then the input set encoding for \(x_n\) will always be an encoding that represents a partition containing no accepting input set. It will then be the case that \(P^E\) will accept \(x_n\) when a Non Deterministic Oracle Machine would halt in a non accepting state without querying the oracle. Therefore, \(P^E \neq NP^E\) because the oracle \(E\) is dysfunctional when handling some problems.

Let \(\kappa\) be the set of all problems in \(NP\) which have complements in \(NP\). When \(x_n \in \kappa\), then \((j, k)\) will be satisfied. When \((j, k)\) is satisfied then no strings applicable to \(x_n\) will be added to \(E\). It is then the case that the subset of \(E\) that applies to all elements of \(\kappa\) will be identical to the subset of \(A\) that applies to all elements of \(\kappa\). So \(P^E = NP^E\) when the problem being evaluated is an element of \(\kappa\).

### 3.6 An oracle set such that \(P^F \subseteq [NP^F \cap co-NP^F] \subseteq NP\).

Baker, Gill and Solovay [1975] do not lay out the specific method for creating oracle \(F\). However they do provide an outline which includes creating two languages \(L_1(F)\) and \(L_2(F)\) such that \(L(F) = [L_1(F) \cup L_2(F)]\). Two oracle sets must also be created by two different methods, and \(F\) is the union of these two oracle sets.

However, notice that if \(P^F \subseteq [NP^F \cap co-NP^F]\), and \([NP^F \cap co-NP^F] \subseteq NP\), then \(P^F \subseteq NP\).

It has already been demonstrated that an oracle set can be created such that \(P^A = NP\). In other words the set of problems solvable by a Deterministic Oracle Machine with oracle set \(A\) in polynomial time is equivalent to the set of problems solvable by a Non Deterministic Turing Machine in polynomial time.

It is then the case that the only new condition of oracle set \(F\) is that \([NP^F \cap co-NP^F]\) contains all problems in \(NP\). This is the same thing as saying that \(NP\) is strictly contained within \(co-NP\).

Earlier, it was shown that an oracle set \(C\) could be created such that \(NP^C\) is not closed under complementation. The reason that oracle set \(C\) allowed this condition was because it resulted in all problems in \(co-NP\) being solvable in polynomial time by \(P^C\), while problems in \(NP\) where not solvable in polynomial time by \(P^C\).

It is then easy to see that an oracle \(F\) could be created such that all problems in \(NP\) are solvable in polynomial time by \(P^F\), and all problems in \(co-NP\) are solvable in polynomial time by \(P^F\). Under such an oracle the complexity classes \(NP\), and \(co-NP\) are both contained by \(P\).

If \(co-NP \subset P^F\), and \(NP \subseteq P^F\), then \(P^F \subseteq [NP \cap co-NP]\). All problems in
both \( NP \) and \( co-NP \) have polynomial time solutions on a Deterministic Oracle Machine with oracle set \( F \). It then follows that all problems in \( NP \) and \( co-NP \) have polynomial time solutions on a Non Deterministic Oracle Machine with oracle set \( F \).

4. ORACLE RELATIVIZATIONS AND P VS. NP SOLUTION METHODS.

Here, the relation of oracle relativizations to the \( P \) vs. \( NP \) question will be examined by creating an analog of the \( P \) vs. \( NP \) question. Two complexity classes will be created, \( P_{\Lambda} \) and \( NP_{\Lambda} \); the question will be asked, is \( P_{\Lambda} = NP_{\Lambda} \)?

To create the analog complexity classes, one problem that is obviously in \( P \) will be excluded from \( P_{\Lambda} \). Any problem in \( P \) will due, here the problem used will be the \textit{set-sum} problem. The \textit{set-sum} problem will be defined as:

—Let \( S \) be a set of real numbers.
—Let \( r \) be the number of elements in \( S \).
—Let \( M \) be a real number.

\[
\sum_{i=1}^{r} S_i = M
\]

Given the value of \( S \) and \( M \), determine if the equality evaluates \textit{true}.

Notice that the \textit{set-sum} problem is similar to the \textit{0-1-Knapsack} problem, and could be solved by finding the sum of all subsets of \( S \), but only evaluate \textit{true} if the one subset of \( S \) that contains all elements of \( S \) sums to \( M \). In the analog complexity model, it will be assumed that this is the only known method of solving \textit{set-sum}.

An analog complexity model will be created which ignores the existence of some \( NP \) problems (such as \textit{NP-complete} problems). The analog complexity classes will be defined as:

—Let \( NP_{\Lambda} \) contain all problems known to be in \( P \) assuming \( P \neq NP \).
—Let \( P_{\Lambda} \) contain all problems in \( NP_{\Lambda} \) except the \textit{set-sum} problem.

In the analog computational model \( NP_{\Lambda} \) contains all problems known to be solvable in non deterministic polynomial time, while \( P_{\Lambda} \) contains all problems known to be solvable in deterministic polynomial time. In the analog question \( P_{\Lambda} = NP_{\Lambda} \) assume that no method has yet been found to prove that \textit{set-sum} is a member of \( P_{\Lambda} \), yet \textit{set-sum} has not been conclusively excluded from \( P_{\Lambda} \).

Now the following questions may be asked.

—Does there exist an oracle such that \( P_{\Lambda}^{A} = NP_{\Lambda}^{A} \)?
—Does there exist an oracle such that \( P_{\Lambda}^{B} \neq NP_{\Lambda}^{B} \)?
—Does there exist an oracle such that \( NP_{\Lambda}^{C} \) is not closed under complementation?
—Does there exist an oracle such that \( P_{\Lambda}^{D} \neq NP_{\Lambda}^{D} \) but \( NP_{\Lambda}^{D} \) is closed under complementation?
—Does there exist an oracle such that \( P_{\Lambda}^{E} \subseteq [ NP_{\Lambda}^{E} \cap co-NP_{\Lambda}^{E} ] \subseteq NP_{\Lambda} \)?
It should be easy to see that the answer to all of these questions is yes. If inconsistent relativizations are an indication of the difficulty of the $P_\Lambda$ vs. $NP_\Lambda$ problem, then the problem of proving the set-sum problem solvable in deterministic polynomial time must be at least as difficult as proving that $NP$-complete problems are solvable in deterministic polynomial time.

Is the set of computable oracles such that $P_\Lambda^X = NP_\Lambda^X$ meager? If so, then does that indicate a greater likelihood that $P_\Lambda \neq NP_\Lambda$?

With this analog it happens to be the case that $P_\Lambda = NP_\Lambda$. If the $P$ vs. $NP$ question is dependant on oracle relativizations, then the analog $P_\Lambda$ vs. $NP_\Lambda$ question should indicate that $P = NP$. If $P$ vs. $NP$ is dependant on oracle relativizations, then it should be expected that $P = NP$ could be proven by finding the relation between the polynomial time solvability of set-sum and the relativizations of $P_\Lambda$ and $NP_\Lambda$. If such a relation exists, then it should provide an indication of how to prove $P = NP$. However, if no such relation exists, then there must exist no dependancy between oracle relativizations and the solution to the $P$ vs. $NP$ question.

Obviously, the proof that $P_\Lambda = NP_\Lambda$ can be accomplished by eliminating the evaluation of all subsets of $S$ for the set-sum problem and only evaluating the one relevant subset. It is known that the only relevant subset is the subset equivalent to $S$. This is known because the original definition of the problem says it is so. Is there some way that the oracle relativizations also identify the one relevant subset of $S$? If so then the relativizations should identify a limited partition of relevant subsets for the Knapsack problem.

It is then easy to see that oracle relativizations only tell us that the complexity of any given problem can be hidden by oracle set creation. Essentially, using an oracle to solve a $NP$-complete problem in deterministic polynomial time is the reverse operation of taking a problem from the $P$ complexity class and pretending that it has no deterministic polynomial time solution. When applied to a $NP$-complete problem, which may actually have no deterministic polynomial time solution, an oracle such that $P^A = NP^A$ allows us to pretend that $NP$-complete problems have deterministic polynomial time solutions.

5. CONCLUSION.

In this article all six oracles form Baker, Gill and Solovay [1975] have been examined. As a result, the following things can be said about the oracles.

— Oracle set $A$ such that $P^A = NP^A$ is a functional oracle which allows a Deterministic Oracle Machine to solve any $NP$ problem with a polynomial number of queries.

— Oracle set $B$ such that $P^B \neq NP^B$ is a dysfunctional oracle which fails to allow a Deterministic Oracle Machine to solve all $NP^B$ problems with a polynomial number of queries.

— Oracle set $C$ such that $NP^C$ is not closed under complementation is a dysfunctional oracle such that a Deterministic Oracle Machine can not solve all $NP$ problems with a polynomial number of queries, although a Deterministic Oracle Machine could solve any co-$NP$ problem with one query.

— Oracle set $D$ such that $P^D \neq NP^D$ but $NP^D$ is closed under complementation is a dysfunctional oracle with a dysfunctional complement. It is then the case that
Independence of $P$ vs. $NP$ in regards to oracle relativizations.

A Deterministic Oracle Machine can not solve any $NP$ problem or any co-$NP$ problem in polynomial time.

- Oracle set $E$ such that $P^E \neq NP^E$ and $P^E = [NP^E \cap co-NP^E]$ is a dysfunctional oracle that allows a Deterministic Oracle Machine to solve a $NP$ problem in polynomial time when the compliment of the problem is also in $NP$.

- Oracle set $F$ such that $P^F \subseteq [NP^F \cap co-NP^F] \subseteq NP$ is a functional oracle which allows a Deterministic Oracle Machine to solve all problems in $NP$ and all problems in co-$NP$ in polynomial time.

If $P = NP$ then all problems in $NP$ will be solvable in deterministic polynomial time with or without an oracle set. It would then be the case that a Deterministic Oracle Machine would only be dependant on an oracle set to solve a problem in $NP$ or co-$NP$ if the machine is not using an optimal algorithm. It is possible to create an algorithm for any $NP$ problem such that a solution would require exponential time on a Non Deterministic Turing Machine. This however does not disqualify the problem’s membership in $NP$. Likewise, just because a $NP$ problem has a deterministic exponential time solution, this does not disqualify that problem’s membership in $P$. Therefore, the oracle relativizations are no indicator of $P$ vs. $NP$.

If $P \neq NP$ then any problem not ordinarily in $P$ will be solvable in polynomial time by a Deterministic Oracle Machine only when the oracle set is functional for that problem. In this case if $P \neq NP$ then the Baker, Gill, and Solovay model works exactly as would be expected.

The meagerness of computable $P^X = NP^X$ oracle sets also seems not to indicate anything about the $P$ vs. $NP$ problem. That is, we should expect a functional oracle for any purpose to be meager. All of these observations lead to the conclusion that the $P$ vs. $NP$ question is independent of oracle relativizations.

6. VERSION HISTORY.

The author wishes to encourage further feedback which may improve, strengthen, or perhaps disprove the content of this article. For that reason the author does not publish the names of any specific people who may have suggested, commented, or criticized the article in such a way that resulted in a revision, unless premission has been granted to do so.

**arXiv Current Version**
3Sep08 Submitted to arXiv.

- Revision of partition encoding method.
- Refrance to [Nagel and Newman 2001] was added.

**arXiv Version 5**
22Aug08 Submitted to arXiv.

- Minor addition.

**arXiv Version 4**
21May08 Submitted to arXiv.

- Minor error correction.

ArXiv, Vol. V, No. N, Month 20YY.
arXiv Version 3
20May08 Submitted to arXiv.

—Clarifications were made for some statements.
—The Oracle relativizations and \( P \) vs. \( NP \) solution methods section was added.
—Reference to [Bennett and Gill 1981] and [Mehlhorn 1973] were added.

arXiv Version 2
16May08 The article was withdrawn due to misleading statements and incomplete research.

arXiv Version 1
14May08 Submitted to arXiv.

REFERENCES
Baker, T. 1979. On “Provable” Analogs of \( P \) and \( NP \). Mathematical Systems Theory 12, 213-218.
Baker, T., Gill, J., and Solovay, R. 1975. Relativizations of the \( P \) = \( \not{=} \) \( NP \) question. SIAM J. Comput. 4, (Dec.), 431-442. MR395311. Zbl 0323.68033.
Bennett, C., and Gill, J. 1981. Relative to a random oracle \( A \), \( P^A \neq NP^A \neq co-NP^A \) with probability 1. SIAM J. Comput. 10, (Feb.), 96-113. MR 0605606. Zbl 0454.68030.
Nagel, E., and Newman, J. 2001. Gödel’s Proof New York University Press.
Hartmanis, J., and Hopcroft, J. 1976. Independence results in computer science. ACM SIGACT News 8, 4.
Meek, J. 2008. \( P \) is a proper subset of \( NP \). arXiv:0804.1079 Article 1 in series of 4.
Meek, J. 2008. Analysis of the deterministic polynomial time solvability of the \( 0 \)-\( 1 \)-Knapsack problem. arXiv:0805.0517 Article 2 in series of 4.
Mehlhorn, K. 1973. On the size of sets of computable functions. 14th Annual Symp. on Automata and Switching Theory (Univ. Iowa, Iowa City, Iowa), 190-196. MR429513.

Received xx/2008; revised xx/2008; accepted xx/2008