THE COMPLEXITY OF PROMISE PROBLEMS

by

Alan L. Selman and Yacov Yacobi

Technical Report #222

October 1981

1 On sabbatical leave from Department of Computer Science, Iowa State University, Ames, Iowa 50011, with funds provided by the United States-Israel Educational Foundation (Fulbright Award). This research was supported in part by the National Science Foundation and grant MCS77-23493 AQ2.

2 On leave from the Electrical Engineering Department of the Technion, Haifa, Israel. Current address: Electrical Engineering and Computer Sciences Dept., University of California at San Diego, La Jolla, CA. 92093, USA.
1. INTRODUCTION

This paper is concerned with several complexity issues about certain kinds of partial decision problems. The nature of these partial decision problems can be best explained by contrasting them with ordinary decision problems. A decision problem is given as a predicate \( P(x) \). The question, of course, is to determine whether there exists an algorithm \( A \) that solves the problem, i.e., such that \( A(x) \) converges for all meaningful input instances \( x \) and such that

\[
\forall x[A(x) = "yes" \iff P(x)].
\]

In practice, one often encounters problems for which only a subclass of the domain of all instances is of concern. Such problems are here called promise problems. Informally, a promise problem has structure

input \( x \),

promise \( Q(x) \),

property \( R(x) \),

where \( Q \) and \( R \) are predicates. Formally, a promise problem is a pair of predicates \((Q, R)\). The predicate \( Q \) is called the promise. A deterministic Turing machine \( M \) solves the promise problem \((Q, R)\) if

\[
\forall x[Q(x) \rightarrow [M(x) \land (M(x) = "yes" \iff R(x))]].
\]

If a Turing machine \( M \) solves \((Q, R)\), then the language \( L(M) \) accepted by \( M \) is a solution to \((Q, R)\).

The study of problems with this format are certainly not new. For partial recursive functions one wants a program that computes correctly on its domain. And, techniques for establishing correctness of a program
are typically distinct from halting issues. Problems of this kind have also arisen in context-free language theory [3].

Complexity issues about promise problems arise from Even and Yacobi's work in public-key cryptography [1]. In [1], a public-key cryptosystem is described and the basic question of whether there exist systems with NP-hard cracking problem is raised. The cracking problem is describable as a promise problem \((Q, R)\), with the additional added feature that both \((Q, R)\) and the complementary problem \((Q, \sim R)\) have solutions in NP. They then introduce a new hypothesis that extends the popular conjecture that NP is not closed under complements, and observe on the basis of this hypothesis that there exist no public-key cryptosystems with NP-hard cracking problem.

We show here that this hypothesis is false. Related results are obtained also. These results indicate that complexity classes of promise problems have different structural properties than do complexity classes of decision problems.
2. BASIC CONCEPTS

We required notation for promise problems that have solutions in NP. One possibility is to extend the definition of NP to include promise problems. However, in order to keep NP sacrosant for decision problems (encoded as languages), the notation of the polynomial hierarchy is borrowed instead.

**Definition 1:** $\Sigma_1^{PP}$ is the class of all promise problems $(Q,R)$ such that $(Q,R)$ has a solution in NP. $\Pi_1^{PP}$ is the class of all promise problems $(Q,R)$ such that $(Q,\neg R)$ is in $\Sigma_1^{PP}$. $\Delta_1^{PP} = \Sigma_1^{PP} \cap \Pi_1^{PP}$.

These definitions can clearly be extended but there is no reason to do so here. Every set $S$ in NP may be considered to be a promise problem $(U,S)$ with the trivial promise $U$, the universal set. In this way, $\Sigma_1^{PP}$ is a proper extension of NP, $\Pi_1^{PP}$ is a proper extension of co-NP, and $\Delta_1^{PP}$ is a proper extension of $\Sigma_1^{PP}$.

Recall that a language $L$ is NP-complete if $L$ is in NP and every set in NP is $\leq^P_m$-reducible to $L$. A language $L$ is NP-hard if every set in NP is $\leq^P_T$-reducible to $L$.

**Definition 2:** A promise problem $(Q,R)$ is NP-hard if every solution $L$ of $(Q,R)$ is NP-hard.

It follows from the definition that if an NP-hard promise problem has a tractable solution (i.e., in P), then $P = NP$. For an oracle Turing machine $M$ with oracle set $A$, let $L(M,A)$ denote the language accepted by $M$ with oracle $A$. According to Definition 2, $(Q,R)$ is NP-hard if and only if, for every set $S$ in NP and for every solution $A$
of \((Q,R)\), there is an oracle Turing machine \(M\) that operates in polynomial time so that \(S = L(M,A)\).

**Definition 3:** A promise problem \((Q,R)\) is **uniformly NP-hard** if, for every set \(S\) in NP, there is an oracle Turing machine \(M\) that operates in polynomial time such that, for all solutions \(A\) of \((Q,R)\), \(S = L(M,A)\).

Uniformly NP-hard obviously implies NP-hard. The converse can be expected to be false but no proof is yet known. In any case, the main results of the next section will hold for both notions.

The concepts thus far defined can now be used as bases for definitions of reductions and uniform reductions between promise problems.

**Definition 4:** A promise problem \((Q,R)\) is **Turing reducible in polynomial time** to a promise problem \((S,T)\), in symbols, \((Q,R) \leq_{UT}^{PP} (S,T)\), if, for every solution \(A\) of \((S,T)\), there is an oracle Turing machine \(M\) that operates in polynomial time such that, \(M\) with oracle \(A\) solves \((Q,R)\).

**Definition 5:** A promise problem \((Q,R)\) is **uniformly Turing reducible in polynomial time** to a promise problem \((S,T)\), \((Q,R) \leq_{UT}^{PP} (S,T)\), if there is an oracle Turing machine \(M\) that operates in polynomial time such that, for every solution \(A\) of \((S,T)\), \(M\) with oracle \(A\) solves \((Q,R)\).

**Lemma 1:**

(i) \(\leq_{UT}^{PP}\) and \(\leq_{UT}^{PP}\) are transitive relations.

(ii) \((Q,R) \leq_{UT}^{PP} (S,T)\) implies \((Q,R) \leq_{UT}^{PP} (S,T)\).

(iii) \((Q,R)\) is NP-hard if and only if, for every set \(S\) in NP, \((S,L) \leq_{UT}^{PP} (Q,R)\).
(iv) \( (Q,R) \) is uniformly NP-hard if and only if, for every set \( S \) in NP, \( (U,S) \leq_{\text{UT}}^{\text{PP}} (Q,R) \).

Whether \( \leq_{\text{UT}}^{\text{PP}} \) implies \( \leq_{\text{T}}^{\text{PP}} \) is an open question.
3. ELEMENTARY RESULTS

Since we are focusing on polynomial-time complexity issues, let us assume henceforth that for all promise problems \((Q,R)\) mentioned, both \(Q\) and \(R\) are recursive predicates. Furthermore, if \(M\) is a Turing machine that solves \((Q,R)\), then \(M\) halts on every input. Therefore, every solution to a promise problem is a recursive set. The solution criterion becomes

\[
Q(x) \rightarrow (M(x) \equiv \text{"yes"}) \leftrightarrow R(x)).
\]

The recursive solutions to a promise problem \((Q,R)\) can be completely characterized by the set theoretically: \(A\) is a recursive solution to \((Q,R)\) if and only if \(A = (Q \cap R) \cup B\), where \(Q \cap B = \emptyset\) and \(B\) is recursive. In particular, \(Q \cap R\) and \(R\) are both solutions, and \(Q \cap R\) is the smallest solution. If \((Q,R)\) is an NP-hard promise problem, then \(R\) and \(Q \cap R\) are NP-hard sets.

Promise problems with tractable promise can be analyzed rather completely. First of all, \(R\) is NP-hard and \(Q\) is in P does not imply that \((Q,R)\) is NP-hard. (Take \(Q\) to be empty or finite and observe that \(Q \cap R\) is either empty or finite.) To obtain a non-trivial example, use Ladner's result [2] to obtain an NP-complete set \(R\) and a set \(Q\) in \(P\) such that \(Q \cap R\) is not NP-hard (although \(Q \cap R\) is in NP-P assuming \(P \neq NP\)). Thus, \((Q,R)\) is not NP-hard.

Theorem 1: If \(R\) is NP-hard, \(Q\) is in \(P\), and \((Q,R)\) has a solution in \(P\), then \((\sim Q,R)\) is NP-hard.

Proof: Let \(R\) be NP-hard, \(Q\) in \(P\), and let \(A\) in \(P\) be a solution to \((Q,R)\). Let \(B\) be an arbitrary solution to \((\sim Q,R)\). To show that \((\sim Q,R)\) is NP-hard, it suffices to show \(R \leq_P B\). This is accomplished
Theorem 2: If \( Q \) is in \( P \) and \( Q \cap R \) is an \( \neg P \)-hard set, then
\[
\{ Q(x) \rightarrow (x \in A \leftrightarrow R(x)) \}
\]

Lemma 2: If \( Q \) is in \( P \) and \( A \) is a solution to \( (Q,R) \), then
\[
Q \cap R \leq^P A.
\]

Proof: Let \( Q \), belong to \( P \) and let \( A \) be any solution to \( (Q,R) \).

The reduction is given by the following algorithm

```plaintext
input x;
if Q(x)
    then if x \in A then accept else reject
        \{Q(x) \rightarrow (x \in A \leftrightarrow R(x))\}
    else if x \in B then accept else reject
        \{\neg Q(x) \rightarrow (x \in B \leftrightarrow R(x))\}.
```

As an immediate consequence we have the following theorem.

Theorem 2: If \( Q \) is in \( P \) and \( Q \cap R \) is an \( \text{NP} \)-hard set, then
\( (Q,R) \) is an \( \text{NP} \)-hard promise problem.

Theorem 2 can be used to generate many interesting examples of
\( \text{NP} \)-hard promise problems in \( \Sigma_1^{PP} \). The technique is this: let \( R \) be
any known \( \text{NP} \)-complete problem and let \( Q \cap R \) be a refinement that is
still \( \text{NP} \)-complete, where \( Q \) belongs to \( \widetilde{P} \). Then, \( (Q,R) \) is \( \text{NP} \)-hard.

For example, let \( \text{SAT} \) be an encoding of the satisfactory formulas of
propositional logic, and let \( \exists \) be an encoding of all formulas with
three literals per clause. \( \exists \cap \text{SAT} \) is the well-known \( \text{NP} \)-complete
set 3SAT. Since 3 is in P, Theorem 2 applies. Hence, (3, SAT) is an NP-hard promise problem in $\Pi_1^p$.

**Theorem 3:** If $Q$ belongs to P and $(Q,R)$ is an NP-hard promise problem, then $(Q,R)$ is uniformly NP-hard.

**Proof:** Since $(Q,R)$ is NP-hard, $Q \cap R$ is an NP-hard solution. Let $M$ be an oracle Turing machine that operates in polynomial time and that implements the algorithm given in the proof of Lemma 2. Let $S$ be any set in NP. Then, the machine which $P^M$ reduces $S$ to $Q \cap R$ followed by $M$ uniformly reduces $S$ to each solution of $(Q,R)$. $\square$
4. ON THE CLASS $\Delta^P_1$

It is well-known that $\text{NP} \cap \text{co-NP}$ contains an NP-hard set if and only if NP is closed under complements. Hence, the former property is unlikely to be true. It is hypothesized in [1] that there exist no NP-hard promise problems in $\Delta^P_1$. By the following lemma it is certainly reasonable to conjecture that $\Sigma^P_1 \not= \Pi^P_1$.

**Lemma 3:** $\Sigma^P_1 \not= \Pi^P_1$ if and only if $\text{NP} = \text{co-NP}$.

**Proof:** Since $\Sigma^P_1$ and $\Pi^P_1$ are extensions of NP and co-NP, respectively, the proof from left to right is trivial. The converse implication follows from the observation that if A is a solution to $(Q,R)$, then $\overline{A}$ is a solution to $(Q,\overline{R})$. \hfill $\square$

However, we now provide an example of an NP-hard promise problem in $\Delta^P_1$. Let $\oplus$ denote the logical operator "exclusive or". Let SAT denote the NP-complete satisfiability problem. We will take the liberty also of writing SAT as a predicate, so that SAT(x) asserts that x is satisfiable. Let EX denote the predicate defined by

$$EX(x,y) \iff SAT(x) \oplus SAT(y).$$

**Theorem 4:**

(i) $(EX, SAT) \in \Delta^P_1$.

(ii) $(EX, SAT)$ is NP-hard.

**Proof:** (i) Let $x$ and $y$ be input words and suppose the promise EX(x,y) is true. Then $\neg SAT(x)$ is equivalent to the predicate SAT(y). Thus, it is evident that both $(EX, SAT)$ and $(EX, \neg SAT)$ belong to $\Sigma^P_1$.

(ii) Let A be any solution to $(EX, SAT)$. (Technically, A is a language consisting of encoded ordered pairs; we will write $<x,y> \in A$.
to denote membership in $A$.) If $\text{SAT}(x) \iff \text{SAT}(y)$, then $<x,y> \in A \iff \text{SAT}(x)$.

To show that $A$ is $\text{NP}$-hard it suffices to show that $\text{SAT} \leq \text{P}^A$. This is accomplished by the following iterative algorithm with oracle set $A$.

Let $\psi$ be a program variables that ranges over propositional formulas. Let $\varphi(\sigma_1, \ldots, \sigma_n)$ be an input formula with Boolean variables $\sigma_1, \ldots, \sigma_n$.

$$\psi := \varphi(\sigma_1, \ldots, \sigma_n);$$

for $i := 1$ to $n$ do

{ $\psi$ has free variables $\sigma_1, \ldots, \sigma_n$ }

if $<\psi(0, \sigma_1, \ldots, \sigma_n), \psi(1, \sigma_1, \ldots, \sigma_n)> \in A$

then $\psi := \psi(0, \sigma_1, \ldots, \sigma_n)$;

else $\psi := \psi(1, \sigma_1, \ldots, \sigma_n)$;

{ $\psi$ is a variable-free Boolean expression }

if $\psi$ has value 1

then accept $<\varphi$ is satisfiable $>$

else reject $<\varphi$ is not satisfiable $>$.

The algorithm clearly operates in polynomial time. If the accept state is reached, then a satisfying assignment for $\varphi$ has been found. Conversely, suppose $\varphi$ is satisfiable. We claim that the loop preserves satisfiability of $\psi$. At each execution of the loop body, if $\psi(\sigma_1, \ldots, \sigma_n)$ is satisfiable, then $\psi(0, \sigma_1, \ldots, \sigma_n)$ is satisfiable or $\psi(1, \sigma_1, \ldots, \sigma_n)$ is satisfiable. If exactly one of these is satisfiable, then the promise is true and the oracle query provides correct information. If both of these are true, then $\psi$ remains true independent of the value of the oracle query. Hence the algorithm correctly reduces $\text{SAT}$ to the solution of $(\text{EX}, \text{SAT})$ in polynomial time. 

$\square$
Since the algorithm just given does not depend on choice of solution, we have the following corollary.

**Corollary:** (EX, SAT) is uniformly NP-hard.

**Open Question:** Define a promise problem \((Q, R)\) to be \(\leq_m^P\)-hard if, for each solution \(A\) of \((Q, R)\) and each set \(L\) in NP, \(L \leq_m^P A\). Is there an \(\leq_m^P\)-hard promise problem in \(\Delta_1^{PP}\)?
REFERENCES

