The Computational Complexity Column

by

Vikraman Arvind

Institute of Mathematical Sciences, CIT Campus, Taramani
Chennai 600113, India
arvind@imsc.res.in
http://www.imsc.res.in/~arvind

We revisit *robust machines* and *helping oracles* introduced by Uwe Schöning [23] three decades ago. A *robust* oracle machine always accepts the same language, regardless of the oracle. An oracle A is said to *help* a robust machine if oracle access to A "speeds up" the machine and makes it polynomial-time bounded. Robust machines with helping oracles actually models interactive computation, and can be seen as a precursor to interactive proofs. We discuss these connections and point out how robust oracle machines relate to some recently defined classes like oblivious NP and oblivious MA [15].

As we keep pace with new developments in the field, it is also worthwhile recalling some of the older ideas and results. Robust machines and helping oracles are a nice example.

Robust Oracle Machines revisited

V. Arvind *

1 Introduction

Let M be an *oracle Turing machine* and let $L(M^A) \subseteq \Sigma^*$ denote the language accepted by the machine M with oracle A, where Σ is a fixed alphabet and inputs are encoded as strings in Σ^*.

Definition 1. [23]

- *An oracle machine M is* robust *if $L(M^A) = L(M^\emptyset)$ for every oracle $A \subseteq \Sigma^*$.*
- *An oracle set $A \subseteq \Sigma^*$* helps *a robust oracle machine M if M^A is polynomial-time bounded on all inputs.*

What is the motivation for studying robust machines and helping oracles? We quote excerpts from the article [23, Section 1]:

> *Typical algorithms for computationally hard problems like NP-complete problems usually involve backtrack search ... consider the situation that the algorithm is allowed to query an oracle during the computation to receive information that might lead to faster search for the solution. This situation can be thought of as some kind of man-machine interaction. ... This model only makes sense if we do not allow the algorithm to rely on the oracle information such that changing the oracle ... would result in changing the final outcome of the algorithm.*

This is an entirely new take on oracle machine computations, quite different from relativization results that were prevalent in structural complexity theory in the 1980's. It provides an algorithmic motivation for the idea of *interactive computation*, and it is perhaps the first time a notion of interactive computation was formulated and studied. The notion of interactive proof systems [16], discovered soon after, has its motivation in cryptography.

Once we have Definition 1, the natural question is which languages have robust oracle machines with oracles that help? We can define the following complexity class [23]:

*Institute of Mathematical Sciences, Chennai, India arvind@imsc.res.in

$$\mathsf{P}_{\mathsf{help}} = \{L^A(M) \mid M \text{ is a robust deterministic Turing machine and } A \text{ helps } M\}.$$

It turns out that $\mathsf{P}_{\mathsf{help}}$ coincides with $\mathsf{NP} \cap \mathsf{coNP}$.

Theorem 2. [23] $\mathsf{P}_{\mathsf{help}} = \mathsf{NP} \cap \mathsf{coNP}$.

Proof Sketch. For the forward inclusion notice that it suffices to show $\mathsf{P}_{\mathsf{help}}$ is contained in NP since $\mathsf{P}_{\mathsf{help}}$ is closed under complements. Suppose $L \in \mathsf{P}_{\mathsf{help}}$. Then $L = L(M^A)$ where M is a robust machine and A is a helping oracle. An NP machine can be easily obtained from M by nondeterministically guessing the answers of oracle A and aborting all computation paths at a suitable polynomial length. Since M is robust this NP machine will not accept $x \notin L$. On the other hand, if $x \in L$ then the path of correct guesses to the queries to A will result in an accepting computation of polynomial length.

For the reverse inclusion, suppose $L \in \mathsf{NP} \cap \mathsf{coNP}$. Let N be an NP machine for L and N' for $\overline{L}$. We design an oracle

$$A = \{\langle x, y\rangle \mid \text{ if } x \in L \text{ then } y \text{ is a prefix of an accepting path of } N(x) \text{ and if } x \notin L \text{ then } y \text{ is a prefix of an accepting path of } N'(x)\}.$$

The robust machine M is a polynomial-time procedure that on input x prefix searches for accepting paths of either $N(x)$ or $N'(x)$ by making oracle queries $\langle x, y\rangle$. In the end, if neither $N(x)$ nor $N'(x)$ accepts, M does a brute-force search to solve x.

□

Remark 3. *Schöning's work on robust machines sparked a lot of interest and related research. An (incomplete) list of papers on this and related topics is [18, 6, 24, 7, 17, 10, 14, 21, 3]. In this article, we consider only some aspects of the topic.*

1.1 *A Proof System Definition*

We can give an equivalent definition of $\mathsf{P}_{\mathsf{help}}$ that is based on the idea of proof systems.

A language L is in $\mathsf{P}_{\mathsf{help}}$ if and only if there is a *polynomial time bounded* oracle Turing machine M such that for any input x and any oracle A the output $M^{(}x)$ is in $\{Acc, Rej, ?\}$ and the following conditions of "soundness" and "completeness" hold:

Soundness: For every $x \in \Sigma^*$ and every oracle A:

$$M^A(x) = Acc \implies x \in L$$
$$M^A(x) = Rej \implies x \notin L$$

Completeness: There is an oracle $\hat{A}$ such that $M^{\hat{A}}(x) \in \{Acc, Rej\}$ for every input x.

The equivalence with the original definition is straightforward. In this course of this article we will see that robustn machines and helping oracles are, in fact, quite closely connected to interactive proof systems.

2 One-sided help

The notion of one-sided helping oracles was introduced and studied by Ker-I Ko [18] as a generalization of helping.

Definition 4. [18] *A language L is in $\mathsf{P}_{1-\mathsf{help}}$ if there is a robust oracle machine M accepting L, an oracle A and a polynomial time bound $p(n)$ such that for all $x \in \Sigma^*$: if $x \in L$ then $M^{(}x)$ accepts x in $p(|x|)$ time.*

Notice that the helping oracle A helps only accepts "yes" instances in polynomial time. Indeed, it is easy to see from the definition that the following holds.

Theorem 5. [18] $\mathsf{P}_{1-\mathsf{help}} = \mathsf{NP}$.

In his paper, Ko also studied the subclass $\mathsf{P}_{1-\mathsf{help}}[C]$ of $\mathsf{P}_{1-\mathsf{help}}$ when the helping oracle is restricted to some language class C. For instance $\mathsf{P}_{1-\mathsf{help}}[\mathsf{NP}]$ is the class of languages in $\mathsf{P}_{1-\mathsf{help}}$ that have helping oracles in NP. It is easy to see that this is no restriction as $\mathsf{P}_{1-\mathsf{help}} = \mathsf{P}_{1-\mathsf{help}}[\mathsf{NP}]$.

It is interesting to investigate the power of helping oracles that are known to be not NP-hard. The class $\mathsf{P}_{1-\mathsf{help}}[A]$ for such oracles A would yield uniform subclasses of NP that do not contain NP-complete sets. We note two further results from Ko's article here. He showed that the class $\mathsf{P}_{1-\mathsf{help}}[\mathrm{Sparse}]$, where Sparse is the collection of all polynomially sparse languages, does not contain NP-complete languages unless the polynomial-time hierarchy collapses to the second level. He also showed that log^*-sparse sets are "no helpers", in the sense that they are powerless as helping oracles to robust machines. A language S is said to be log^*-sparse if the number of strings in S of length between n and 2^n is bounded by a fixed constant k for every $n > 0$.

Theorem 6. [18]

- *The class* $\mathsf{P}_{1-\mathsf{help}}[\mathrm{Sparse}]$ *does not contain* NP*-complete languages unless* $\mathsf{PH} = \Sigma_2^p$.
- $\mathsf{P}_{1-\mathsf{help}}[S] = \mathsf{P}$ *for every log^*-sparse language S.*

It is also interesting to consider the power of helping oracles that come from complexity classes not known to contain NP. Recall that a language L is in $\mathsf{Mod}_k\mathsf{P}$ if there is an NP machine M such that

$$x \in L \iff acc_M(x) \not\equiv 0 (mod\ k),$$

where $acc_M(x)$ denotes the number of accepting paths of M on input x.

Ogihara [21] has examined the classes $\mathsf{P}_{1-\mathsf{help}}[\mathsf{Mod}_k\mathsf{P}]$, for prime k, and obtained the following curious result.

Theorem 7. [21] *A language L is in $\mathsf{P}_{1-\mathsf{help}}[\mathsf{Mod}_k\mathsf{P}]$, k prime, if and only if there is an NP machine M that accepts L such that $x \in L \iff acc_M(x) \not\equiv 0 (mod\ k)$.*

Remark 8. *In general, it would be interesting to further investigate the classes $\mathsf{P}_{1-\mathsf{help}}[C]$ and $\mathsf{P}_{\mathsf{help}}[C]$ for different language classes C.*

3 Interactive proof systems

In this section we will start with observations from [24] on *probabilistic robust machines*. Then we will discuss its equivalence with MIP (where MIP stands for the class of languages with multi-prover interactive protocols). Also, compare with the definition of IP (the class of languages that have single prover interactive protocols). Then we will discuss the [3] paper of interactive proof systems with bounded prover complexity and also include comparisons with helping machines.

Noticing the connection to interactive proof systems, in [24] Schöning generalized the class $\mathsf{P}_{1-\mathsf{help}}$ by allowing randomized robust computations. We recall the formal definition.

Definition 9. [24] *A languages L is in $\mathsf{BPP}_{1-\mathsf{help}}$ if there is a randomized polynomial-time Turing machine M and an oracle A such that for all inputs $x \in \Sigma^*$:*

- *If $x \in L$ then*
$$\text{Prob}[M^A(x)\ accepts] \geq 3/4.$$
- *If $x \notin L$ then for all oracles B*
$$\text{Prob}[M^B(x)\ accepts] \leq 1/4.$$

We now recall a quick definition of interactive proof systems. The reader can find more details in complexity theory textbooks such as [1].

Definition 10. *Let V be a probabilistic polynomial-time machine and p a polynomial. A language L is in the class MIP if there is* multiprover interactive protocol *such that for every positive integer n and $x \in \Sigma^n$:*

- *If $x \in L$ then there exist provers $P_1, P_2, \ldots, P_{p(n)}$ such that*
$$\text{Prob}[P_1, \ldots, P_{p(n)}\ make\ V\ accept] \geq 3/4.$$

- *If $x \notin L$ then for all provers $P'_1, P'_2, \ldots, P'_{p(n)}$ such that*

$$\text{Prob}[P'_1, \ldots, P'_{p(n)} \textit{ make } V \textit{ accept}] \leq 1/4.$$

We note that IP is the subclass of MIP when the interactive protocol allows only a single prover. Two seminal results in the area of interactive proof systems are: IP = PSPACE [25, 20] and MIP = NEXP [5].

It turns out that the class $\mathsf{BPP}_{1-\mathsf{help}}$ coincides with MIP, which is the class of languages that have multiprover interactive proof systems. This was actually discovered in [5] as an "oracle" characterization of MIP in the course of the proof that MIP = NEXP [5].

Theorem 11. [5] $\mathsf{BPP}_{1-\mathsf{help}} = \mathsf{MIP}$.

An interesting point that arises in interactive protocols is the complexity of the "honest provers" $P_i, 1 \leq i \leq p(n)$ for a given language L. Let us denote by $\mathsf{IP}[C]$ and $\mathsf{MIP}[C]$, the subclasses of IP and MIP, respectively, consisting of languages with honest provers that are polynomial-time Turing reducible to some set in C. With this notation we recall some known results here:

- PSPACE = IP[PSPACE] [11, 25].
- $\mathsf{P}^{\mathsf{PP}} \subseteq \mathsf{IP}[\mathsf{PP}]$ [20].
- NEXP = $\mathsf{MIP}[\mathsf{EXP}^{\mathsf{NP}}]$ [5].
- $\oplus\mathsf{P}$ and PH are contained in $\mathsf{IP}[\oplus\mathsf{P}]$ [4].

Regarding the fourth containment above more can be said. In fact, since $\mathsf{BPP}_{1-\mathsf{help}}[\oplus\mathsf{P}] = \mathsf{MIP}[\oplus\mathsf{P}] = \mathsf{IP}[\oplus\mathsf{P}]$, it follows that $\mathsf{BPP}_{1-\mathsf{help}}[\oplus\mathsf{P}] = \mathsf{BPP}^{\oplus\mathsf{P}} = \mathsf{BP}\cdot\oplus\mathsf{P}$. It is interesting to compare this with Theorem 7, which is about deterministic robust machines helped by $\oplus\mathsf{P}$ oracles.

3.1 *Interactive proof systems with provers in* P/poly

The class MIP[P/poly] is an interesting uniform subclass of P/poly. It was investigated in [3], and we recall observations from that paper in this subsection.

Proposition 12. $\mathsf{BPP} \subseteq \mathsf{MIP}[\mathsf{P/poly}] \subset \mathsf{P/poly}$.

As noted in [14, 24], for any class of languages C we have $\mathsf{BPP}_{1-\mathsf{help}}[C] = \mathsf{MIP}[C]$. In particular, we have:

Proposition 13. $\mathsf{BPP}_{1-\mathsf{help}}[\mathsf{P/poly}] = \mathsf{MIP}[\mathsf{P/poly}]$.

It follows as a consequence that $\mathsf{P}_{1-\mathsf{help}}[\mathsf{P/poly}]$ is contained in $\mathsf{MIP}[\mathsf{P/poly}]$. One might wonder if $\mathsf{MIP}[\mathsf{P/poly}]$ could contain more languages than BPP. However, it turns out that $\mathsf{MIP}[\mathsf{P/poly}]$ contains all sparse sets in NP.

Theorem 14. *All sparse sets in* NP *are contained in* $\mathsf{P}_{1-\mathsf{help}}[\mathsf{P/poly}]$.

Proof Sketch. et $S \in \mathsf{NP}$ be a sparse set. Suppose N is an NP machine accepting S. Let $S^{=n}$ denote the strings of length n in S. For each $s \in S^{=n}$, let w_s denote the leftmost accepting path in $N(s)$. We can ensure by padding that $|w_x| = p(|x|)$ for each input $x \in S$, for a fixed polynomial p. We include the string $\langle s, y, 0^n \rangle$ in the helping oracle $\hat{A}$ for every prefix y of the string w_s, for each $s \in S^{=n}$ and each n. As S is sparse, it follows that $\hat{A} \in \mathsf{P/poly}$.

Now, let x be an input instance for S. We construct a deterministic robust Turing machine M that queries a given oracle A, for strings of the kind $\langle x, y, 0^{|x|} \rangle$, to guide a prefix search for the leftmost accepting path w_x of $N(x)$. Once w_x is constructed, M accepts iff $N(x)$ accepts along w_x. Clearly, $\hat{A}$ is a helping oracle for M which proves that S is in $\mathsf{P}_{1-\mathsf{help}}[\mathsf{P/poly}]$. □

Now, since $\mathsf{P}_{1-\mathsf{help}}[\mathsf{P/poly}] \subseteq \mathsf{MIP}[\mathsf{P/poly}]$, the existence of sparse NP sets not in BPP would imply that BPP is a proper subclass of $\mathsf{MIP}[\mathsf{P/poly}]$. Applying a standard translation technique [9] we can obtain the following.

Theorem 15. *If* NE *is* not *contained in* BPE *then* BPP *is a proper subset of* $\mathsf{MIP}[\mathsf{P/poly}]$.

On the other hand, we stress that $\mathsf{MIP}[\mathsf{P/poly}]$ is a small complexity class. It is shown in [3] that, like BPP, $\mathsf{MIP}[\mathsf{P/poly}]$ is also *low* for Σ_2^p: i.e. sets in $\mathsf{MIP}[\mathsf{P/poly}]$ are powerless as oracle to Σ_2^p. More precisely,

Theorem 16. $\Sigma_2^A = \Sigma_2^p$ *for all* $A \in \mathsf{MIP}[\mathsf{P/poly}]$.

3.2 *Oblivious NP*

In [15], in their study of fixed circuit lower bounds for different complexity classes, the authors considered the notion of oblivious classes and specifically considered oblivious versions of NP and MA.

Definition 17. [15] *A language L is in oblivious* NP *if there is a polynomial-time computable binary relation R, a polynomial $p(n)$, and for all n there is a witness $w_n \in \Sigma^{p(n)}$ such that $x \in \Sigma^n$: $x \in L$ if and only if $R(x, w_n) = 1$.*

The class oblivious MA is similarly defined when the binary relation R is relaxed to be computable in randomized polynomial time with one-sided error.

Theorem 18. *Oblivious* NP *coincides with* $\mathsf{P}_{1-\mathsf{help}}[\mathsf{P/poly}]$.

Proof Sketch. uppose L is in $\mathsf{P}_{1-\mathsf{help}}[\mathsf{P/poly}]$ witnessed by a polynomial-time bounded robust machine M and helping oracle $A \in \mathsf{P/poly}$. For inputs of length n the robust machine M makes queries to the oracle of length bounded by $p(n)$ for a fixed polynomial p. Let w_n denote the concatenation of all advice strings upto length $p(n)$. Then we can define the required binary relation R from the robust machine M such that $x \in L \cap \Sigma^n$ if and only if $R(x, w_n) = 1$.

Conversely, suppose L is in oblivious NP. We define the "helping" oracle $\hat{A}$ consisting of all pairs $\langle 0^n, i, b\rangle$ where b is the i^{th} bit of w_n for $1 \leq i \leq p(n)$. The corresponding robust machine M on input $x \in \Sigma^n$ will query the oracle A to extract all the bits of the string w_n and then compute $R(x, w_n)$ to decide if $x \in L$. Clearly, M is robust with $\hat{A}$ as one-sided helping oracle for accepting L. □

Likewise, we can easily show that oblivious MA coincides with $\mathsf{BPP}_{1-\mathsf{help}}[\mathsf{P/poly}]$.

4 Self-helpers

We now consider the notion of *self-helping* defined by Ko [18] and discuss its connection to *program checkers*. Program result checking is another idea emanating from work on interactive proofs.

Definition 19. [18] *A language $L \subseteq \Sigma^*$ is a self-helper if L is in $\mathsf{P}_{\mathsf{help}}[L]$.*

Integer factorization provides a nice example in this context. For every positive integer n we are interested in computing the function $\mathrm{fact}(n) = \langle (p_1, e_1), (p_2, e_2), \ldots, (p_k, e_k)\rangle$, such that $p_1 < p_2 \cdots < p_k$ are distinct primes and $n = \prod_i p_i^{e_i}$. Furthermore, we can assume some standard binary encoding of $\mathrm{fact}(n)$ such that $|\mathrm{fact}(n)| = c.\lceil \log n\rceil$ for a fixed constant c. We associate with $\mathrm{fact}(n)$ the language:

$$L_{\mathrm{fact}} = \{\langle n, b, i\rangle \mid 1 \leq i \leq c.\lceil \log n\rceil \text{ and the } i^{th} \text{ bit of } \mathrm{fact}(n) \text{ is } b\}.$$

Clearly, by construction, integer factorization and L_{fact} are polynomial-time equivalent.

Lemma 20. *The language L_{fact} is a self-helper.*

Proof Sketch. he polynomial-time robust machine M on input $\langle n, \alpha, i\rangle$ will query the given oracle, say A, for each $\langle n, b, j\rangle$, where $b \in \{0, 1\}$ and $1 \leq j \leq c.\lceil \log n\rceil$. For any i, if both $\langle n, 0, j\rangle$ and $\langle n, 1, j\rangle$ are in A or both are not in A then the oracle A is bad and the machine M will output ? and stop. Otherwise, from the answers to all queries the machine M can construct $\mathrm{fact}(n) = \langle (p_1, e_1), (p_2, e_2), \ldots, (p_k, e_k)\rangle$ and verify that $n = \prod_i p_i^{e_i}$ and $p_1 < p_2 < \cdots < p_k$. If the verification fails the machine will output ? and stop. If it succeeds the machine M will accept $\langle n, \alpha, i\rangle$ iff it agrees with the

oracle A's answer. Clearly the machine M is robust and accepts L_{fact}. Furthermore, the language L_{fact} clearly helps M. □

The notion of self-helping is connected to an interesting philosophical question: can there be a *nonconstructive* proof of P = NP? In other words, is it possible that we obtain a "mathematical proof" that SAT is polynomial-time solvable, but we do not have the actual algorithm? The first place where this question was studied is attributed to Levin [19] (in [13, 12]). The following definition is helpful.

Definition 21. [13] *By* P *constructively equal to* NP *is meant that an algorithm is known that computes, from the index and time-bound of an* NP *machine M, the index of a deterministic polynomial-time Turing machine for L(M). Furthermore, a language A is* constructively NP*-hard if a polynomial-time many-one reduction from* SAT *to A is known.*

More generally, for a computational problem X (which is not known to be polynomial-time computable nor is any superpolynomial time lower bounds known for it) can we prove a theorem of the kind "If X is polynomial-time computable then it is *constructively* polynomial-time computable"?

This is made precise in [12] as an algorithm design problem: A computational problem X is P*-time self-witnessing* if we can design an algorithm $\mathcal{A}$ for X with the property that if there exists some polynomial-time algorithm $\mathcal{B}$ for X then $\mathcal{A}$ is a polynomial-time algorithm for X.

Levin [19] first noticed that integer factorization has such an algorithm. Indeed, it is a simple consequence of the fact that L_{fact} is a self-helper. In general, we have the following easy to prove observation.

Theorem 22. [2] *If we have designed a robust oracle machine M for a set A such that A is a self-helper w.r.t. M then A has the* P*-time self-witnessing property defined above.*

It is an interesting open problem if an NP-complete set has the P-time self-witnessing property. One way to prove this would be by showing that NP-complete sets are self-helpers, but that would imply NP = coNP because self-helpers are in NP ∩ coNP. However, it is easy to show that every NP-complete set A is a "one-sided" self-helper: i.e. $A \in \mathsf{P}_{1-\mathsf{help}}[A]$ for NP-complete sets A [18].

4.1 *Graph Minor Theorem*

The Graph Minor Theorem due to Robertson and Seymour [22] is a celebrated monumental result in graph theory. We recall the statement and discuss how it is connected to self-helping oracles.

A graph H is a *minor* of a graph G (denoted $H \leq_m G$), if is H can be obtained from G by a sequence of zero or more edge-contractions, edge-deletions, or vertex-deletions of G. Clearly, $\leq_m$ forms a preorder on graphs (it is not a partial order because antisymmetry is only upto isomorphism).

A class $\mathcal{K}$ of graphs is *minor-closed* if for every $G \in \mathcal{K}$ every minor of G also belongs to $\mathcal{K}$. For instance, given a family of graphs $\{G_i\}_{i\geq 0}$ we can define $\mathcal{K}$ to consist of all graphs G such that $G_i \not\leq_m G$ for all $i \geq 0$. Clearly, $\mathcal{K}$ is a minor-closed family of graphs and is defined by the list G_i of *excluded minors*.

Conversely, if $\mathcal{K}$ is any minor-closed family of graphs then it can be characterized by a list of excluded minors $\{G_i\}_{i\geq 0}$: we can simply let $G_i, i \geq 0$ be the set of all graphs not in $\mathcal{K}$.

The Graph Minor Theorem asserts that every minor-closed family of graphs can be characterized by a *finite* list of excluded minors.

Theorem 23 (Graph Minor Theorem)**.** [22] *Every minor-closed family of graphs can be characterized by a finite list of excluded minors.*

In the course of their proof (spread over 500 pages and twenty articles), Robertson and Seymour also gave an $O(n^3)$ time algorithm for checking if H is a minor of G, for every *fixed* graph H. Here n is the number of vertices in G, and the big-Oh hides a superpolynomial constant depending on H.

Consequently, for any minor-closed family $\mathcal{K}$ of graphs, there is an $O(n^3)$ algorithm to check if $G \in \mathcal{K}$. We only need to check for each excluded minor H that $H \not\leq_m G$. A curious aspect of this algorithm is that the list of excluded minors for $\mathcal{K}$ may be unknown! Moreover, the list, though finite, might be astronomically large!

This question was studied in [13] who show how to deal with this problem in many cases. Specifically, we explain an algorithm adapted from [13], in the context of self-helping oracles, that works correctly for certain minor-closed families $\mathcal{K}$, *without* knowing explicitly the list of excluded minors.

Let $G_1 \prec G_2 \prec \ldots$ be a total ordering on all undirected graphs such that $G_i \prec G_j$ iff either (i) $|V(G_i)| < |V(G_j)|$, or (ii) $|V(G_i)| = |V(G_j)|$ and $|E(G_i)| < |E(G_j)|$, or (iii) $|V(G_i)| = |V(G_j)|$ and $|E(G_i)| = |E(G_j)|$ and G_i lexicographically precedes G_j.

Theorem 24. [13] *Suppose $\mathcal{K}$ is a minor-closed family of graphs such that*

- $\mathcal{K} \in \mathsf{P}_{\mathsf{help}}[\mathcal{K}]$ *witnessed by a robust oracle machine M.*
- *The robust machine M has the additional property that for any input graph G the machine M queries the oracle only for graphs G' such that $G' \prec G$.*

Then, from M we can design a deterministic polynomial-time decision procedure for $\mathcal{K}$ (which does not require finding the entire list of excluded minors for $\mathcal{K}$).

Proof Sketch. We can assume M is polynomial-time bounded and outputs one of $Acc, Rej, ?$. We describe the polynomial-time decision procedure, call it $\mathcal{A}$, for the minor-closed family $\mathcal{K}$. Let G be an input instance whose membership in $\mathcal{K}$ we need to decide. As $\mathcal{K}$ is a minor-closed family, by the Graph Minor Theorem it is characterized by a finite (but unknown) list of excluded minors. In the course of its execution the decision procedure will discover a subset E of these excluded minors. To begin

with we can assume $E = \emptyset$. If at any stage $H \leq_m G$ for some $H \in E$ then we can reject G and stop (because H is an excluded minor).

If $H \not\leq_m G$ for any H in the current set E, the decision procedure simulates the robust machine M on input G answering the oracle queries with oracle $\mathcal{K}_E$ defined as follows:

$$\mathcal{K}_E = \{G' \mid H \not\prec G' \text{ for all } H \in E\}.$$

Notice that $\mathcal{K}_E \supset \mathcal{K}$. Thus, $G' \notin \mathcal{K}_E$ implies $G' \notin \mathcal{K}$ but not, in general, the other direction. In the end, if M accepts the input G then the decision procedure $\mathcal{A}$ also accepts (because M is robust and $\mathcal{K}$ is a self-helper which means the "yes" outputs are correct). Now, suppose M with oracle $\mathcal{K}_E$ outputs "no". If all the queries are correctly answered by oracle $\mathcal{K}_E$ then the "no" answer is correct because the machine is robust. That means E is not the complete list of excluded minors. On the other hand, if some "yes" answers by oracle $\mathcal{K}_E$ are incorrect that also implies E is not the complete list of excluded minors. In either case, the algorithm $\mathcal{A}$ will run an enumeration of all graphs in $\prec$ order and look for the first graph G' such that $G' \notin \mathcal{K}$. This is the first graph $G' \notin E$ in the $\prec$ order rejected by the machine M with $\emptyset$ as oracle. By the Graph Minor Theorem we know that E is a fixed set of constant size. If the maximum number of vertices in a graph in E is c then the above enumeration takes time $2^{c^2}\dot{\text{poly}}(n)$. Hence the overall running time is polynomial in n.

□

4.2 *Program Checkers*

We briefly mention connections to the notion of program result checking defined by Blum and Kannan [8]. We first recall the formal definition.

Definition 25. [8] *A computational problem π is said to be* checkable *if there is a randomized polynomial-time oracle Turing machine C_π such that for any deterministic program P that halts on all inputs and purports to solve π, and input instance x of π and security parameter 1^k the following hold*

- *If $P(y) = \pi(y)$ for all inputs y then $C_\pi(x, 1^k) = 1$ with probability* 1.
- *If $P(x) \neq \pi(x)$ then with probability at least $1 - 1/2^k$ $C_\pi(x, 1^k)$ outputs P is incorrect.*

The problem π is deterministically checkable *if C_π is a deterministic oracle Turing machine.*

A host of natural computational problems is shown to have efficient program checkers in [8] and many subsequent papers. Standard complexity-theoretic examples include the Permanent, complete problems for $\oplus$P, PSPACE, and EXP.

It is easy to prove that integer factorization has a deterministic checker using the fact that L_{fact} is a self-helper. Indeed, it can be shown that a decision problem π is deterministically checkable iff it is a self-helper.

References

[1] S. Arora, B. Barak. *Computational Complexity: A modern approach.* Cambridge University Press, New York, USA, 2009.

[2] V. Arvind. Constructivizing membership proofs in complexity classes. *Intl Journal of Foundations of Computer Science,* 8(4):433-442, 1997.

[3] V. Arvind, J. Köbler, R. Schuler. On helping and interactive proof systems. *Intl. Journal of Foundations of Computer Science,* 6(2): 137-153, 1995.

[4] L. Babai, L. Fortnow. Arithmetization: a new method in structural complexity. *Computational Complexity,* 1:41-66, 1991.

[5] L. Babai, L. Fortnow, C. Lund. Nondeterministic exponential time has two-prover interactive protocols. *Computational Complexity,* 1:1-40, 1991.

[6] J. Balcázar. Only smart oracles help. Technical report LSI-88-9, Universitat Politècnica de Catalunya, 1988.

[7] J. Balcázar. Self-reducibility structures and solutions of NP problems. *Revista Matematica de la Universidad Complutense de Madrid,* 2:175-184, 1989.

[8] M. Blum, S. Kannan. Designing programs that check their work. *Journal of the ACM,* 42(1):269-291, 1995.

[9] R. Book. Tally languages and complexity classes. *Information and Control,* 26:186-193, 1974.

[10] J.Y. Cai, L. Hemachandra, J. Vyskoc. Promises and fault-tolerant database access, In *Complexity Theory,* volume edited by K. Ambos-Spies, S. Homer, U. Schöning, pages 101-146, CUP, 1993.

[11] P. Feldman. The optimal prover lives in PSPACE. manuscript, 1986.

[12] M.R. Fellows, N. Koblitz. Self-witnessing polynomial time complexity and prime factorization. *Proc. 6th Annual IEEE Structure in Complexity Conference,* 107-110, 1992.

[13] M.R. Fellows, M.A. Langston. Nonconstructive tools for proving polynomial-time decidability. *Journal of the ACM,* 35(3):727-739, 1988.

[14] L. Fortnow, J. Rompel, M. Sipser. On the power of multiprover interactive protocols. *Proc. 3rd Structure in Complexity Theory Conference,* 156-161, 1988.

[15] L. Fortnow, R. Santhanam, R. Williams. Fixed-Polynomial Size Circuit Bounds. *24th Annual IEEE Conference on Computational Complexity,* 19-26, 2009.

[16] S. Goldwasser, S. Micali, C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal of Computing,* 18:186-208, 1989.

[17] J. Hartmanis, L. Hemachandra. Robust machines accept easy sets. *Theoretical Computer Science,* 74(2):217-226, 1984.

[18] Ker-I Ko. On helping by robust oracle machines. *Theoretical Computer Science,* 52:15-36, 1987.

[19] L.A. Levin. Universal Enumeration problems. *Problemy Peredachi Informatsii,* IX, 115-116, 1972.

[20] C. Lund, L. Fortnow, H. Karloff, N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM,* 39(4):859-868, 1992.

[21] M. Ogihara. On helping by parity-like languages. *Theoretical Computer Science,* 54:41-43, 1995.

[22] N. Robertson, P. Seymour. Graph Minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory,* Series B, 92(2):325ÎšâĆňâĂIJ357, 2004.

[23] U. Schöning. Robust algorithms: a different approach to oracles. *Theoretical Computer Science,* 40:57-66, 1985.

[24] U. Schöning. Robust oracle machines. *Proc. 13th Mathematical Foundations of Computer Science,* LNCS, Springer, 93-106, 1988.

[25] A. Shamir. IP=PSPACE. *Journal of the ACM,* 39(4):869-877, 1992.