Identifying an Honest EXP^NP Oracle Among Many

Shuichi Hirahara

Department of Computer Science, The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 133-8654, Japan
`hirahara@is.s.u-tokyo.ac.jp`

── **Abstract** ──────────────────────────────────

We provide a general framework to remove short advice by formulating the following computational task for a function f: given two oracles at least one of which is honest (*i.e.* correctly computes f on all inputs) as well as an input, the task is to compute f on the input with the help of the oracles by a probabilistic polynomial-time machine, which we shall call a *selector*. We characterize the languages for which short advice can be removed by the notion of selector: a paddable language has a selector if and only if short advice of a probabilistic machine that accepts the language can be removed under any relativized world.

Previously, instance checkers have served as a useful tool to remove short advice of probabilistic computation. We indicate that existence of instance checkers is a property stronger than that of removing short advice: although no instance checker for EXP^{NP}-complete languages exists unless $EXP^{NP} = NEXP$, we prove that there exists a selector for any EXP^{NP}-complete language, by building on the proof of $MIP = NEXP$ by Babai, Fortnow, and Lund (1991).

1 Introduction

Blum and Kannan [10] introduced the notion of *instance checker*. Roughly speaking, an instance checker for a function f is an efficient probabilistic machine that, given access to an oracle, checks if the oracle computes $f(x)$ correctly on a given instance x; the oracle models a possibly buggy program that purports to compute f, and an instance checker verifies whether the program works correctly on a given instance.

The notion of instance checker is intimately related to interactive proof systems: the line of work showing the power of interactive proofs [22, 24, 6] yielded instance checkers for $P^{\#P}$-, PSPACE-, and EXP-complete languages; in addition, Blum and Kannan [10] gave a characterization of the languages with an instance checker by a function-restricted interactive proof system. Since any language with an interactive proof protocol is in NEXP [17], any language with an instance checker must be in NEXP ∩ coNEXP.

In this paper, we investigate a computational task weaker than instance checking of a (Boolean) function f: we are given access to two oracles (instead of a single oracle) as well as an input x; again, both of the oracles purport to compute f; however, it is assumed that at least one of the two oracles is *honest*, *i.e.* computes $f(q)$ correctly on all inputs q; and the task is to compute $f(x)$ with the help of the oracles in polynomial time. We shall call a probabilistic machine doing the task a *(probabilistic) selector* for f.

If the answers of oracles on the input x agree, then we have only to output the answer, which is surely correct by the assumption. Thus, the task of a selector is essentially to

identify the honest oracle when two oracles disagree on x (*i.e.* one of the oracles asserts that $f(x) = 0$, whereas the other asserts that $f(x) = 1$).

Our main result shows that there exists a selector for $\mathrm{EXP}^{\mathrm{NP}}$-complete languages. We also show that the notion of selector does not change even if there are one honest oracle and polynomially many dishonest oracles. Thus, these results can be encapsulated in the following phrase: "identifying an honest oracle among many is *strictly* weaker than instance checking unless $\mathrm{EXP}^{\mathrm{NP}} = \mathrm{NEXP}$."

Although the task is weaker than instance checking, a situation in which one may assume existence of an honest oracle naturally arises out of computation with advice: Suppose, for example, that a (paddable) language L is computed by a probabilistic machine M with advice of one bit. We regard M with advice 0 and 1 as two oracles A_0 and A_1, respectively. By the definition of advice, either A_0 or A_1 is honest on all the inputs (of the same length). Thus, the advice of one bit can be removed if L has a selector. We can in fact remove advice of size $O(\log n)$, since a selector can identify an honest oracle among polynomially many oracles.

1.1 Removing Short Advice for Probabilistic Computation

In early work as to removing short advice for probabilistic computation, Trevisan and Vadhan [25] gave an insight into the potential of instance checkability: they demonstrated that instance checkability can be exploited to remove short advice. Based on the existence of an instance checker for EXP-complete languages, they showed a quantitative tradeoff from a uniform worst-case-hardness assumption (*i.e.* $\mathrm{EXP} \not\subseteq \mathrm{BPTIME}(t(n^{O(1)}))$) to average-case hardness of EXP (*i.e.* EXP contains languages that cannot be solved by probabilistic computation on a fraction better than $\frac{1}{2} + \frac{1}{t}$ of inputs in time t).

They also argued that their result cannot be obtained via *black-box uniform reductions*. Typical constructions of a worst-case to average-case connection are based on the following scheme: we convert a function f into another function f', which is an error-correcting code of f; if we have a "black-box" algorithm that computes f' on a fraction greater than $\frac{1}{2} + \epsilon$ of the inputs, then a probabilistic machine that takes advice can compute f on all inputs by decoding f'. Since it is impossible to uniquely decode f' for small ϵ, the advice is used to identify f and is provably indispensable.

Indeed, it was the instance checkability of EXP-complete languages that broke the black-box construction in the proof of Trevisan and Vadhan; the instance checkability enabled them to remove advice of logarithmic size. Therefore, it will be helpful for future research to closely understand the property that they actually exploited.

Subsequent to their work, instance checkability has since been exploited to cope with short advice for probabilistic computation: for example, Barak [7] proved the first hierarchy theorem for probabilistic computation with short advice; Buhrman, Fortnow, and Santhanam [12] unconditionally separated BPEXP from BPP with advice of subpolynomial size; and Buhrman, Fortnow, Koucký, and Loff [11] gave some evidences that a deterministic efficient computation with oracle access to the set of Kolmogorov-random strings can be simulated by a probabilistic efficient computation.

1.2 Our Results

In fact, the notion of selector captures a property of removing short advice:

▶ **Theorem 1.1.** *Let L be an arbitrary paddable language. The following are equivalent:*
1. *There exists a selector for L.*
2. *For any oracle $R \subseteq \{0,1\}^*$, it holds that $L \in \mathrm{BPP}^R /\!/ \log$ implies $L \in \mathrm{BPP}^R$.*

That is, a paddable language has a selector if and only if short advice can be removed under any relativized world. ("//" means advice that can depend on coin flips of probabilistic machines as well as input length [25].)

In addition, we construct a selector for $\mathrm{EXP^{NP}}$-complete languages, thereby indicating an essential difference between selectors and instance checkers. We also give an upper bound on the languages with a selector:

▶ **Theorem 1.2** (Main Theorem).
1. *Every* $\mathrm{EXP^{NP}}$*-complete language has a selector.*
2. *Any language with a selector is in* $\mathrm{S}_2^{\mathrm{exp}}$ *(which is an exponential-time analogue of* $\mathrm{S}_2^{\mathrm{p}}$*).*

Thus, existence of an instance checker is a property stronger than that of removing short advice (or, equivalently, existence of a selector): although no instance checker for $\mathrm{EXP^{NP}}$-complete languages exists unless $\mathrm{EXP^{NP}} = \mathrm{NEXP}$, short advice of a probabilistic machine that accepts $\mathrm{EXP^{NP}}$-complete languages can be removed.

Our Techniques

The most technical part of this paper is a proof of the main theorem (Theorem 1.2, Part 1). In order to construct a selector for $\mathrm{EXP^{NP}}$-complete languages, we build on the proof of $\mathrm{MIP} = \mathrm{NEXP}$ by Babai, Fortnow, and Lund [6]. As pointed out by Gábor Tardos in the paper [6], the complexity of honest provers of the interactive proof system for NEXP-complete languages can be bounded above by $\mathrm{EXP^{NP}}$. We crucially use this fact to check satisfiability of an exponential-sized formula with the help of an $\mathrm{EXP^{NP}}$-complete oracle. We also compare two exponential-sized strings by performing a binary search.

Thanks to plenty of machinery that has been cultivated together with interactive proof systems, program checking, and PCPs, we can prove the main theorem by careful combinations of such machinery. For example, we exploit a multilinearity test [6] and the self-correction of low-degree polynomials [8, 21].

Due to the usage of arithmetization, we suspect that our proof of the main theorem does algebrize [1] but does not relativize.

Variants of Selectors

We also investigate other variants of selectors: a *deterministic selector* and a *nonadaptive deterministic selector*. We focus on the "suprema" of the languages with a selector, namely, upper bounds on these languages and existence of a selector for languages complete for a complexity class that is close to the upper bounds. (Note that the languages with a selector are not necessarily closed downward. For example, although $\mathrm{NEXP} \subseteq \mathrm{EXP^{NP}}$, we do not know whether NEXP-complete languages have a selector or not.)

For a nonadaptive deterministic selector, we prove polynomial-time analogues of Theorem 1.2:

▶ **Theorem 1.3.**
1. *Every* $\mathrm{P^{NP}}$*-complete language has a nonadaptive deterministic selector.*
2. *Any language with a nonadaptive deterministic selector is in* $\mathrm{S}_2^{\mathrm{p}}$*.*

The proofs of this theorem will clearly illustrate the basic ideas for Theorem 1.2.

Notice that $\mathrm{P^{NP}}$ is close to the upper bound $\mathrm{S}_2^{\mathrm{p}}$ since $\mathrm{P^{NP}} \subseteq \mathrm{S}_2^{\mathrm{p}} \subseteq \mathrm{ZPP^{NP}}$ [23, 14]. (Under suitable hardness assumptions, it holds that $\mathrm{P^{NP}} = \mathrm{S}_2^{\mathrm{p}}$ by derandomization [19].)

For a deterministic selector, the supremum is PSPACE:

▶ **Theorem 1.4.**
1. *Every* PSPACE-*complete language has a deterministic selector. More generally, any downward self-reducible language has a deterministic selector.*
2. *Any language with a deterministic selector is in* PSPACE.

As with Theorem 1.1, a property of removing short advice for deterministic computation can be characterized by existence of a deterministic selector:

▶ **Theorem 1.5.** *Let L be an arbitrary paddable language. The following are equivalent:*
1. *There exists a deterministic selector for L.*
2. *For any oracle $R \subseteq \{0,1\}^*$, it holds that $L \in P^R / \log$ implies $L \in P^R$.*

1.3 Comparison with Prior Work

In seminal work by Karp and Lipton [18] as to collapses of a uniform class contained in a nonuniform class, it was shown that $NP \subseteq P / \log$ implies $NP \subseteq P$ and $PSPACE \subseteq P / \log$ implies $PSPACE \subseteq P$. These results are essentially equivalent to the existence of deterministic selectors for NP- and PSPACE-complete languages, respectively.

Fortnow and Klivans [16] observed that $NEXP \subseteq BPP // \log$ implies $NEXP = BPP$ by combining previous results. Similarly, it is folklore that $EXP^{NP} \subseteq BPP // \log$ implies $EXP^{NP} = BPP$. This follows by combining the result by Buhrman and Homer [13] stating that $EXP^{NP} \subseteq EXP / poly$ implies $EXP^{NP} = EXP$, the existence of an instance checker (or a selector) for EXP-complete languages, and $BPP // \log \subseteq P / poly$ (see [16]).

We clarify the differences between the folklore and our results in two respects. First, our results can be relativized on the right-hand side. Second, selectors can be used to quantitatively remove advice of logarithmic size: if we allow a machine to run in time t (instead of polynomial time), then advice of size $\log t$ can be removed.

▶ **Corollary 1.6** (Analogous to Proposition 5.6 in [25]). *There are an EXP^{NP}-complete language L and a constant $d \in \mathbb{N}$ such that, for any nice time bound[1] $t \colon \mathbb{N} \to \mathbb{N}$ and any oracle $R \subseteq \{0,1\}^*$, if $L \in BPTIME^R(t(n)) // \log t(n)$ then $L \in BPTIME^R(t(n^d))$.*

We mention in passing that, by substituting selectors for instance checkers in the proofs of Trevisan and Vadhan [25], one can obtain a quantitative tradeoff from a uniform worst-case-hardness assumption on EXP^{NP} to a uniform average-case hardness of EXP^{NP} (see [25, Theorem 5.7]).

1.4 Application: Random Strings vs. Randomized Computation

In Section 6, we will give another application in order to demonstrate usefulness of the notion of selector, by simply substituting selectors for instance checkers in the previous work by Buhrman, Fortnow, Koucký, and Loff [11].

They tried to show that a deterministic polynomial-time computation with oracle access to the set of Kolmogorov-random strings is, in some sense, equivalent to a probabilistic polynomial-time computation; they modeled oracle access to the set of Kolmogorov-random strings as advice strings of high nonuniform complexity. Although the nonuniform complexity of the advice strings is required to be much higher than that of Kolmogorov-random strings, they showed, as a partial result, that if a language L can be solved in deterministic polynomial

---

[1] Although the definition of a *nice time bound* is the same as in [25], we note that the condition $t(n) \leq 2^n$ is not needed here.

time with high nonuniform advice, then L is in BPP with advice of almost linear size [11, Theorem 13].

Because the goal is to show that L is in BPP *without* any advice, they further observed that one can dispense with the advice of almost linear size if there exists an instance checker for L. From this observation, they showed that, for any class $\mathcal{C} \in \{\mathrm{NP}, \mathrm{P}^{\#\mathrm{P}}, \mathrm{PSPACE}, \mathrm{EXP}\}$, if some $\mathcal{C}$-complete language can be solved in deterministic polynomial time with high nonuniform advice, then $\mathcal{C} \subseteq \mathrm{BPP}$ [11, Theorem 15].

In fact, they proved this result by analyzing the two cases: For $\mathcal{C} \in \{\mathrm{P}^{\#\mathrm{P}}, \mathrm{PSPACE}, \mathrm{EXP}\}$, they used an instance checker for $\mathcal{C}$-complete languages, whose existence was shown by [22, 24, 6]; Unfortunately, because it is not known whether NP-complete languages have instance checkers or not, they needed to prove the result in another way solely for $\mathcal{C} = \mathrm{NP}$.

The notion of selector, however, enables us to show the result in a unified way and to extend the result from $\{\mathrm{NP}, \mathrm{P}^{\#\mathrm{P}}, \mathrm{PSPACE}, \mathrm{EXP}\}$ to any classes whose complete languages have a selector. Given the fact that many languages have selectors (*e.g.* languages with instance checkers and downward self-reducible languages), it becomes more plausible that we can dispense with the advice of almost linear size; thereby we slightly strengthen the connection between Kolmogorov-random strings and randomized computation.

Organization

In Section 2, we give formal definitions, common properties of selectors, and a proof of Theorem 1.1. Sections 3, 4, and 5 are devoted to investigating nonadaptive deterministic selectors, probabilistic selectors, and deterministic selectors, respectively. We mention some possible directions for future work in Section 7.

Preliminaries and Notations

We assume that the reader is familiar with basics of computational complexity (*e.g.* [2]).

For a Turing machine M, let $M(x)$ denote the output of M on input $x \in \{0,1\}^*$. For an oracle Turing machine M and oracles $A_0, A_1 \subseteq \{0,1\}^*$, let M^{A_0,A_1} represent a machine equipped with access to oracle $A \subseteq \{0,1\}^*$ such that $A(i \cdot q) = A_i(q)$, for each $i \in \{0,1\}$ and for any $q \in \{0,1\}^*$. We identify false and true with 0 and 1, respectively. We also identify a language $L \subseteq \{0,1\}^*$ with its characteristic function from $\{0,1\}^*$ to $\{0,1\}$. For a Boolean formula φ in n variables, we abuse notation and write $\varphi \colon \{0,1\}^n \to \{0,1\}$.

We say that a language L is paddable if there exists a polynomial-time machine that, on input $(x, 1^m)$ where $x \in \{0,1\}^n$ and $n \leq m$, outputs a string y of length m such that $y \in L$ if and only if $x \in L$.

2 Definitions and Common Properties of Selectors

In this section, we give formal definitions of selectors and show common properties that all types of selectors have. First, we define a probabilistic selector:

▶ **Definition 2.1** (Probabilistic Selector). A *(probabilistic) selector* S for a language $L \subseteq \{0,1\}^*$ is a probabilistic polynomial-time oracle Turing machine which computes L with high probability, given arbitrary two oracles $A_0, A_1 \subseteq \{0,1\}^*$ such that A_0 or A_1 is equal to L. That is, for any input $x \in \{0,1\}^*$ and oracles $A_0, A_1 \subseteq \{0,1\}^*$,

$$L \in \{A_0, A_1\} \implies \Pr\left[S^{A_0,A_1}(x) = L(x)\right] \geq \frac{2}{3},$$

where the probability is taken over coin flips of S.

Note that the success probability $\frac{2}{3}$ in Definition 2.1 can be enhanced by repetitions. We often abbreviate a probabilistic selector as a selector.

An oracle equal to L is said to be *honest*; otherwise it is said to be *dishonest.*

Next, we define a deterministic selector and a nonadaptive deterministic selector:

▶ **Definition 2.2** (Deterministic Selector). A *deterministic selector* for a language L is a deterministic polynomial-time oracle machine S such that $S^{L,X}(x) = S^{X,L}(x) = L(x)$ for any oracle $X \subseteq \{0,1\}^*$ and for any input $x \in \{0,1\}^*$.

▶ **Definition 2.3** (Nonadaptive Deterministic Selector). A *nonadaptive deterministic selector* S for a language L is a deterministic polynomial-time oracle machine such that
- $S^{L,X}(x) = S^{X,L}(x) = L(x)$ for any oracle $X \subseteq \{0,1\}^*$ and any input $x \in \{0,1\}^*$, and
- S is nonadaptive, *i.e.* there exists a polynomial-time machine which, on input $x \in \{0,1\}^*$, outputs the query set $Q(x)$ of all the queries that S makes to either of the oracles.

We state a useful structural property:

▶ **Proposition 2.4.** *The class of the languages with a selector is closed under polynomial-time Turing equivalence. Namely, $L_1 \leq_T^p L_2$ and $L_2 \leq_T^p L_1$ imply that if L_1 has a selector then so does L_2.*

In particular, it is closed under complement. Moreover, for any complexity class $\mathcal{C}$, if a specific $\mathcal{C}$-complete language has a selector, then so does an arbitrary $\mathcal{C}$-complete language.

Proof. The proof is essentially the same with Beigel's theorem [10], which shows the same closure property of instance checkers. The idea is as follows: reduce a L_2 problem to a L_1 problem by using the reducibility from L_2 to L_1, and solve the L_1 problem by running a selector for L_1, while converting its query (which is an instance of L_1) into an instance of L_2.

Let M_{ij} be a polynomial-time oracle machine that witnesses the polynomial-time Turing reduction $L_i \leq_T^p L_j$ for each $(i,j) \in \{(1,2),(2,1)\}$ (that is, $M_{ij}^{L_j}(x) = L_i(x)$ for any x), and S be a selector for L_1. The following algorithm yields a selector for L_2: Given an input $x \in \{0,1\}^n$ and two oracles A_0, A_1, simulate $M_{21}(x)$ in order to compute $L_2(x)$. If M_{21} makes a query q, then we try to answer it with $L_1(q)$, by running $S(q)$. If S makes a query q' to the ith oracle ($i \in \{0,1\}$), then answer it with $M_{12}^{A_i}(q')$.

Let A_i be an honest oracle (*i.e.* $A_i = L_2$). Then, we have $M_{12}^{A_i}(q') = M_{12}^{L_2}(q') = L_1(q')$, and hence $S(q)$ is simulated under the existence of the honest oracle; thus it outputs $L_1(q)$ correctly with high probability (say, with probability at least $1 - 2^{-n}$, by running the selector $O(n)$ times). Therefore, the simulation of $M_{21}(x)$ results in outputting $L_2(x)$ with probability at least $1 - 2^{-n}n^{O(1)}$. ◀

▶ **Remark 2.5.** Similarly, the class of languages with a deterministic selector is closed under polynomial-time Turing equivalence, and the class of languages with a nonadaptive deterministic selector is closed under polynomial-time truth-table (*i.e.* nonadaptive) equivalence.

To prove Theorem 1.1, we show that the definitions of selectors are robust even if we consider a situation in which we are given polynomially many oracles.

▶ **Lemma 2.6.** *For any language $L \subseteq \{0,1\}^*$, the following are equivalent:*
1. *There exists a selector for L.*
2. *There exists a selector for L that identifies an honest oracle among polynomially many oracles.*

The latter can be formally stated as follows: for any polynomial $m \colon \mathbb{N} \to \mathbb{N}$, there exists a probabilistic polynomial-time oracle Turing machine S such that, on input length $n \in \mathbb{N}$, it holds that $\Pr\left[S^A(x) = L(x)\right] \geq \frac{2}{3}$ for any $x \in \{0,1\}^n$, where A is an arbitrary oracle such that there exists an index $i \in \{1, \cdots, m(n)\}$ that satisfies $A(i,q) = L(q)$ for all $q \in \{0,1\}^*$.

Proof. The one direction is obvious: If there exists a selector that works among $m(n)$ oracles, then letting $m(n) := 2$ yields a selector that works among two oracles.

Conversely, let S be a selector (that identifies an honest oracle among two oracles) with probability at least $1 - \frac{1}{3m(n)}$. Given an oracle A, let $A_i(q)$ denote $A(i,q)$ for any $i \in \mathbb{N}$. On input $x \in \{0,1\}^n$, we first make a query x to all the oracles $A_1, \cdots, A_{m(n)}$, and divide them into the two sets according to their answers:

$$C_0 = \{\, j \in \{1, \cdots, m(n)\} \mid A_j(x) = 0 \,\},$$
$$C_1 = \{\, k \in \{1, \cdots, m(n)\} \mid A_k(x) = 1 \,\}.$$

That is, C_α ($\alpha \in \{0,1\}$) is the set of the indices of all the oracles asserting that $L(x) = \alpha$.

Next, we repeat the following until $C_0 = \emptyset$ or $C_1 = \emptyset$: Pick arbitrary elements $j \in C_0$ and $k \in C_1$. We check which is a supposedly honest oracle by running S^{A_j, A_k} on input x. If $S^{A_j, A_k}(x) = 0$, then we doubt A_k and thus eliminate k from C_1; Otherwise we doubt A_j and eliminate j from C_0.

Finally, we output 1 if and only if $C_1 \neq \emptyset$.

Now let us analyze this algorithm. It runs in polynomial time because $|C_0| + |C_1|$ is decreased by one in each repetition.

We claim the correctness of the algorithm. For simplicity, we assume that $L(x) = 0$. Then, there exists an index $i \in \{1, \cdots, m(n)\}$ such that A_i is honest and $i \in C_0$. If $i \in C_0$ and some $k \in C_1$ are picked in a repetition, then $\Pr\left[S^{A_i, A_k}(x) = 0\right] \geq 1 - \frac{1}{3m(n)}$. That is, i remains in C_0 with probability at least $1 - \frac{1}{3m(n)}$. Since i is picked at most $|C_1|$ ($\leq m(n)$) times, the probability that i remains in C_0 is at least $1 - m(n) \cdot \frac{1}{3m(n)} = \frac{2}{3}$.   ◄

▶ **Remark 2.7.** Although Lemma 2.6 is stated only for a probabilistic selector, analogous statements hold for a deterministic selector and a nonadaptive deterministic selector. For a deterministic selector, one can easily check that the same proof works. For a nonadaptive deterministic selector, we must compute the query set in polynomial time. On input x, let $Q(x)$ denote the query (to either A_0 or A_1) set of a selector that identifies an honest oracle among two oracles. Then we can define all the set of possible queries as $Q'(x) := \{\, (i,q) \in \mathbb{N} \times \{0,1\}^* \mid 1 \leq i \leq m(|x|),\ q \in Q(x) \cup \{x\} \,\}$, which is clearly computable in polynomial time.

By using Lemma 2.6, we characterize the class of the paddable languages with a selector by the property that short advice can be removed under any relativized world. In fact, we can prove a statement stronger than Theorem 1.1:

▶ **Theorem 2.8.**
1. *For any paddable language L, if L has a selector, then $L \in \mathrm{BPP}^R /\!/ \log$ implies $L \in \mathrm{BPP}^R$ for any oracle $R \subseteq \{0,1\}^*$.*
2. *For any language L, if $L \in \mathrm{P}^R/1$ implies $L \in \mathrm{BPP}^R$ for any oracle $R \subseteq \{0,1\}^*$, then L has a selector.*

As a corollary, we immediately obtain Theorem 1.1 (note that $\mathrm{P}^R/1 \subseteq \mathrm{BPP}^R /\!/ \log$).

Proof.

Part 1. Let M be a polynomial-time oracle machine which witnesses $L \in \mathrm{BPP}^R /\!/a$, where $a(n) = O(\log n)$. That is, there exists an advice function $\alpha\colon \{0,1\}^* \to \{0,1\}^*$ such that, for every $n \in \mathbb{N}$,

$$\Pr_{r \in \{0,1\}^{t(n)}} \left[\forall x \in \{0,1\}^n,\ M^R(x, r, \alpha(r)) = L(x)\right] \geq \frac{5}{6}, \tag{1}$$

where $|\alpha(r)| = a(n)$ and t is a polynomial (see also [25, Definition 5.1]).

Let $l(n)\ \left(= n^{O(1)}\right)$ be an upper bound on the running time of a selector for L on inputs of length n. By Lemma 2.6, there exists a selector S that can identify an honest oracle among $m(n)$ oracles for $m(n) := 2^{a(l(n))} = n^{O(1)}$ with probability at least $\frac{5}{6}$. By padding, we may assume that S makes only queries of length exactly $l(n)$ on each input length $n \in \mathbb{N}$

Consider the following probabilistic algorithm: On input $x \in \{0,1\}^n$, pick a string $r \in_R \{0,1\}^{t(l(n))}$ uniformly at random, and define oracles by $A_i(q) := M^R(q, r, i)$ for any $q \in \{0,1\}^{l(n)}$, where $i \in \{1, \cdots, m(n)\}$ is identified with $i \in \{0,1\}^{a(l(n))}$. Simulate S on input x, answering its queries $q \in \{0,1\}^{l(n)}$ to A_i by computing $M^R(q, r, i)$.

If a "good" string r is picked (whose probability is at least $\frac{5}{6}$ by (1)), then we have $A_i(q) = M^R(q, r, i) = L(q)$ for any $q \in \{0,1\}^{l(n)}$, where $i = \alpha(r)$. That is, A_i is honest for some i with probability at least $\frac{5}{6}$. Thus, the algorithm computes L correctly with probability at least $1 - \frac{1}{6} - \frac{1}{6} = \frac{2}{3}$.

Part 2. We prove the contraposition. Assume that L does not have any selectors.

Recall that we regard the computation given oracle access to two oracles R_0, R_1, namely M^{R_0, R_1}, as M^R where $R(i \cdot q) = R_i(q)$ for each $i \in \{0,1\}$. Thus, the goal is to show that there exist oracles $R_0, R_1 \subseteq \{0,1\}^*$ such that $L \in \mathrm{P}^{R_0, R_1}/1$ and $L \notin \mathrm{BPP}^{R_0, R_1}$.

We use a diagonalization argument on all the probabilistic polynomial-time oracle machine $M_1, M_2, \cdots$. We construct $R_0^{(e)}, R_1^{(e)}$ at stage $e \in \mathbb{N}$, and then define $R_i := \bigcup_e R_i^{(e)}$ for each $i \in \{0,1\}$.

We will construct them so that, for each $n \in \mathbb{N}$, there exists $j_n \in \{0,1\}$ such that $R_{j_n}(q) = L(q)$ for any $q \in \{0,1\}^n$. Thus, $L \in \mathrm{P}^{R_0, R_1}/1$ holds because we can make a query x to obtain $R_{j_n}(x) = L(x)$ with advice $\{j_n\}_{n \in \mathbb{N}}$ of one bit.

Let us now construct $R_0^{(e)}, R_1^{(e)}$, and $l^{(e)} \in \mathbb{Z}$, where $l^{(e)}$ represents the maximum length of the strings that have been fixed. At stage $e = 0$, we set $R_0^{(0)} = R_1^{(0)} = \emptyset$, and $l^{(0)} := -1$.

At stage $e \geq 1$, we claim that $R_0^{(e-1)}$ and $R_1^{(e-1)}$ can be extended so that some input $x^{(e)}$ can fool M_e:

▶ **Claim 2.9.** For each $e \geq 1$, there exist oracles $A_0, A_1 \subseteq \{0,1\}^*$ and a string $x^{(e)} \in \{0,1\}^*$ such that

1. A_i agrees with $R_i^{(e-1)}$ on all the strings of length at most $l^{(e-1)}$ for each $i \in \{0,1\}$,
2. either A_0 or A_1 agrees with L on all the strings of length greater than $l^{(e-1)}$, and
3. $\Pr\left[M_e^{A_0, A_1}(x^{(e)}) = L(x^{(e)})\right] < \frac{2}{3}$.

Proof of Claim 2.9. Assume otherwise. That is, for any oracles $A_0, A_1 \subseteq \{0,1\}^*$ and string $x \in \{0,1\}^*$, we have $\Pr\left[M_e^{A_0, A_1}(x) = L(x)\right] \geq \frac{2}{3}$ if Properties 1 and 2 hold. Then, the following algorithm yields a selector for L, which contradicts the assumption: we hardwire all the strings in $R_i^{(e-1)}$ of length at most $l^{(e-1)}$ into a table; given oracles A_0, A_1 one of which agrees with L, we simulate M_e, answering its queries q to A_i ($i \in \{0,1\}$) with the content of the table if $|q| \leq l^{(e-1)}$ and with $A_i(q)$ otherwise. ◀

Define $l^{(e)}$ $\left(> l^{(e-1)} \right)$ as an upper bound on the length of the queries that $M_e^{A_0,A_1}(x^{(e)})$ makes. Then, define $R_i^{(e)}$ as $R_i^{(e)}(q) := R_i^{(e-1)}(q) = A_i(q)$ if $|q| \leq l^{(e-1)}$; $R_i^{(e)}(q) := A_i(q)$ if $l^{(e-1)} < |q| \leq l^{(e)}$; and $R_i^{(e)}(q) = 0$ otherwise, for each $q \in \{0,1\}^*$. This completes the construction of stage e.

On one hand, $x^{(e)}$ witnesses $M_e^{R_0,R_1}$ not computing L on input $x^{(e)}$ for any $e \geq 1$, by Property 3; thus, we have $L \notin \mathrm{BPP}^{R_0,R_1}$. On the other hand, for each input length $n \in \mathbb{N}$, either R_0 or R_1 agrees with L on $\{0,1\}^n$, by Property 2; thus, we have $L \in \mathrm{P}^{R_0,R_1}/1$.   ◄

▶ **Remark 2.10.** Again, the analogous statement (Theorem 1.5) holds for a deterministic selector. A proof is essentially the same and hence is omitted.

One can also prove the quantitative version (Corollary 1.6) of Part 1 of Theorem 2.8 by changing parameters in the proofs of Theorem 2.8 and Lemma 2.6.

3   Nonadaptive Deterministic Selector

In this section we prove Theorem 1.3.

We first prove Part 1 of Theorem 1.3, which states that *every* P^{NP}-complete language has a nonadaptive deterministic selector. It is sufficient to show that a *specific* P^{NP}-complete language has a selector (recall Proposition 2.4 and Remark 2.5). We construct a nonadaptive deterministic selector for the following canonical P^{NP}-complete language (see [20] for a proof of its completeness).

▶ **Definition 3.1** (Lexicographically Maximum Satisfying Assignment; Krentel [20])**.** The *lexicographically maximum satisfying assignment problem* contains all the pairs (φ, k) such that $\varphi \colon \{0,1\}^n \to \{0,1\}$ is a satisfiable Boolean formula in n variables for some $n \in \mathbb{N}$, and $a_k = 1$, where $a_1 \cdots a_n \in \{0,1\}^n$ denotes the lexicographically maximum satisfying assignment of φ.

In other words, the lexicographically maximum satisfying assignment problem is the decision version of the problem of answering, given a Boolean formula φ in n variables, the lexicographically maximum satisfying assignment if φ is satisfiable and 0^n otherwise. Note that it is implicit in the definition that the answer is 0^n for an unsatisfiable Boolean formula.

Proof of Part 1 of Theorem 1.3. We show an algorithm of a selector for the lexicographically maximum satisfying problem, together with its analysis. Let us call two oracles A_0 and A_1.

On input (φ, k), the set of all the queries that we make is $\{ (\varphi, j) \mid j \in \{1, \cdots, n\} \}$, where $n \in \mathbb{N}$ is the number of variables in φ. The (presumably) lexicographically maximum satisfying assignment asserted by each oracle A_i ($i \in \{0,1\}$) can be obtained by concatenating the answers of the oracle, namely $A_i(\varphi, 1) \cdot A_i(\varphi, 2) \cdots A_i(\varphi, n) =: v_i \in \{0,1\}^n$.

If the kth bits of v_0 and v_1 agree, then we simply output it because the oracles agree on input (φ, k).

Otherwise v_0 is not equal to v_1. Therefore, we may assume without loss of generality that $v_0 < v_1$. We check whether v_1 is a satisfying assignment or not by evaluating $\varphi(v_1)$. If $\varphi(v_1) = 1$, then we trust the oracle A_1 and output $A_1(\varphi, k)$ because A_1 showed a satisfying assignment larger than v_0; otherwise we doubt A_1 and output $A_0(\varphi, k)$ because A_1 tried to cheat us by answering an unsatisfying assignment.   ◄

Then we show that any language with a nonadaptive deterministic selector is in $\mathrm{S}_2^{\mathrm{p}}$.

Proof of Part 2 of Theorem 1.3. Let L be a language with a nonadaptive deterministic selector S. We claim that L is in S_2^p. Let $Q(x) = \{q_1, \cdots, q_m\}$ be the query set of S on input $x \in \{0,1\}^*$.

We consider the following polynomial-time machine M: Suppose that the input to M is $(x, y, z) \in \{0,1\}^n \times \{0,1\}^m \times \{0,1\}^m$. Let $y = y_1 \cdots y_m$ and $z = z_1 \cdots z_m$. M simulates the selector S on input x. If S makes a query q_i to the oracle A_0, then it is answered with y_i. Similarly, if S makes a query q_i to the oracle A_1, then it is answered with z_i.

Then, there exists $y \in \{0,1\}^m$ such that $M(x, y, z) = L(x)$ for any $z \in \{0,1\}^m$. Indeed, if y is the concatenation of $L(q_1), \cdots, L(q_m)$, then by the definition of a nonadaptive deterministic selector, $M(x, y, z)$ correctly outputs $L(x)$ for any $z \in \{0,1\}^m$, because all the queries that S makes to A_0 are answered correctly. Similarly, there exists $z \in \{0,1\}^m$ such that $M(x, y, z) = L(x)$ for any $y \in \{0,1\}^m$. ◂

4 Probabilistic Selector

In this section we investigate probabilistic selectors.

First, we show that probabilistic selectors can be constructed based on instance checkers. An instance checker is formally defined as follows:

▶ **Definition 4.1** (Instance Checker [10]). An *instance checker* C for a language L is a probabilistic polynomial-time oracle machine such that, given any oracle $A \subseteq \{0,1\}^*$,
1. if $A = L$ then C^A accepts with high probability, *i.e.* $\Pr\left[C^A(x) = 1\right] \geq \frac{2}{3}$ on all the input $x \in \{0,1\}^*$, and
2. for any input $x \in \{0,1\}^*$, if $A(x) \neq L(x)$ then $C^A(x)$ rejects with high probability, *i.e.* $\Pr\left[C^A(x) = 0\right] \geq \frac{2}{3}$,
where the probability is taken over coin flips of C.

▶ **Proposition 4.2.** *Every language with an instance checker has a selector.*

Proof. Suppose that a language L has an instance checker C. Given input $x \in \{0,1\}^*$ and two oracles $A_0, A_1 \subseteq \{0,1\}^*$, we check which is honest, A_0 or A_1, by computing $C^{A_0}(x)$. If $C^{A_0}(x)$ accepts, then we trust A_0 and output $A_0(x)$; otherwise we doubt A_0 and output $A_1(x)$.

Let us analyze the algorithm above. If $A_0 = L$, then $C^{A_0}(x)$ accepts with probability at least $\frac{2}{3}$, and hence we can output $A_0(x) = L(x)$ correctly with probability at least $\frac{2}{3}$.

Otherwise, it must hold that $A_1 = L$. If $A_0(x) = L(x)$, then we can surely output $L(x)$ correctly since $A_0(x) = A_1(x) = L(x)$. If $A_0(x) \neq L(x)$, then $C^{A_0}(x)$ rejects with probability at least $\frac{2}{3}$, and thus we can output $A_1(x) = L(x)$ correctly with probability at least $\frac{2}{3}$. ◂

Next, we show an upper bound on the languages with a probabilistic selector. For completeness, we include a definition of $S_2^{\exp}$, which is a straightforward exponential-time analogue of S_2^p:

▶ **Definition 4.3.** We say that a language L is in $S_2^{\exp}$ if there exist a time-constructible function $t(n) = 2^{n^{O(1)}}$ and a Turing machine M running in time $2^{|x|^{O(1)}}$ on input $(x, \cdot, \cdot)$ such that, for any input $x \in \{0,1\}^*$,

$$\exists y \in \{0,1\}^{t(|x|)}, \forall z \in \{0,1\}^{t(|x|)}, \ M(x, y, z) = L(x),$$
$$\exists z \in \{0,1\}^{t(|x|)}, \forall y \in \{0,1\}^{t(|x|)}, \ M(x, y, z) = L(x).$$

The proof itself is essentially a corollary of Part 2 of Theorem 1.3:

Proof of Part 2 of Theorem 1.2. Notice that a probabilistic selector can be simulated by an exponential-time nonadaptive deterministic selector. In addition, every language with an exponential-time nonadaptive deterministic selector is in $S_2^{\exp}$, which is an exponential-time analogue of Part 2 of Theorem 1.3. Combining these two facts, it follows that every language with a probabilistic selector is in $S_2^{\exp}$. ◀

4.1 Selector for $\mathrm{EXP}^{\mathrm{NP}}$-complete Languages

In this subsection we prove the main theorem (Theorem 1.2, Part 1). That is, we construct a selector for $\mathrm{EXP}^{\mathrm{NP}}$-complete languages.

Proof Sketch

We sketch the proof of the main theorem. We will construct a selector for a specific $\mathrm{EXP}^{\mathrm{NP}}$-complete language, which is a problem of finding the lexicographically maximum satisfying assignment of a succinctly described Boolean formula $F_\Phi \colon \{0,1\}^{2^n} \to \{0,1\}$. The basic strategy to construct a selector for this language is the same with that of Part 1 of Theorem 1.3: Given access to two oracles $A_0, A_1 \subseteq \{0,1\}^*$, we request them to reveal the presumably lexicographically maximum satisfying assignments $V_0, V_1 \in \{0,1\}^{2^n}$ asserted by A_0, A_1, respectively. The rest of the algorithm consists of two parts: First, we determine the larger assignment of V_0 and V_1, checking whether $V_0 < V_1$ or $V_0 > V_1$. Second, we verify whether the larger assignment satisfies the formula F_Φ or not. Obviously, the obstacle is that there can be exponentially many variables and clauses in F_Φ.

For the second part, Babai, Fortnow, and Lund [6] showed that, given access to provers (or, equivalently, an oracle), one can efficiently check that exponentially many constraints in F_Φ are satisfied: basically, by encoding an assignment as a multilinear function and using arithmetization, it holds that the assignment satisfies all the clauses in F_Φ if and only if the sum of some low-degree polynomials (that can be computed by the multilinear function and the arithmetization) over a subdomain $\{0,1\}^l$ is equal to 0, and the latter can be verified by using the sum-check protocol [22] (called the LFKN protocol in [6]). As pointed out by Gábor Tardos [6], since $\mathrm{EXP}^{\mathrm{NP}}$ is capable of finding a satisfying assignment of an exponential-sized Boolean formula, the honest oracle in the protocol above can be implemented in $\mathrm{EXP}^{\mathrm{NP}}$; thus, given access to an honest $\mathrm{EXP}^{\mathrm{NP}}$-complete oracle (which is A_0 or A_1), one can verify the satisfiability.

For the first part, we perform a binary search to obtain the lexicographically first index z such that V_0 and V_1 disagree. Thus, we need
1. to check if $V_0 = V_1$ on some range of indices, and
2. to split the range into two parts.

We observe that these can be done if we encode a satisfying assignment by the multilinear extension (as with [6]): Let $\mathbb{F}$ be a finite field. We regard the assignments $V_0, V_1 \in \{0,1\}^{2^n}$ as vectors in $\mathbb{F}^{2^n}$. There is a bijective correspondence between a vector $V \in \mathbb{F}^{2^n}$ and a multilinear function $\widetilde{V} \colon \mathbb{F}^n \to \mathbb{F}$. For example, if $n = 2$ and $V = (V_{00}, V_{01}, V_{10}, V_{11})$, then

$$\widetilde{V}(x_1, x_2) = V_{00}(1 - x_1)(1 - x_2) + V_{01}(1 - x_1)x_2 + V_{10}x_1(1 - x_2) + V_{11}x_1 x_2.$$

For Part 1, we can rely on the polynomial identity testing: indeed, since the multilinear extension is bijective, we have $V_0 \neq V_1$ if and only if these multilinear extensions $\widetilde{V}_0$ and $\widetilde{V}_1$ differ; thus, it is sufficient to check if the two low-degree polynomials $\widetilde{V}_0$ and $\widetilde{V}_1$ differ.

It is well known that, given access to two low-degree polynomials, one can efficiently check if these polynomials differ: given access to two functions $\widetilde{V}_0, \widetilde{V}_1$, pick a random point

$u \in_R \mathbb{F}^n$ and check if $\widetilde{V}_0(u) \neq \widetilde{V}_1(u)$. Assuming that the functions are low-degree (which is true if they are multilinear), the Schwartz-Zippel lemma assures that $\widetilde{V}_0$ and $\widetilde{V}_1$ disagree on a large fraction of inputs if $\widetilde{V}_0 \neq \widetilde{V}_1$. Although it is possible that a dishonest oracle tries to cheat us by storing a high-degree polynomial, we can check whether or not the function stored by an oracle is close to some multilinear function, by using the multilinearity test [6].

For Part 2, we use the following simple fact: Fixing the first variable of a multilinear extension $\widetilde{V}$ to 0 or 1, we obtain multilinear extensions that correspond to the first or second part of V. In the example above, we obtain two multilinear functions:

$$\widetilde{V}(0, x_2) = V_{00}(1 - x_2) + V_{01}x_2, \quad \widetilde{V}(1, x_2) = V_{10}(1 - x_2) + V_{11}x_2.$$

These correspond to multilinear extensions of (V_{00}, V_{01}) and (V_{10}, V_{11}), respectively, for $n = 1$. Thus, we can recursively compute the lexicographically first disagreement.

Proof of the Main Theorem

Now we move on to the proof of the main theorem. We construct a selector for the following EXP^{NP}-complete language, which is an analogue of the NEXP-complete languages called the oracle-3-satisfiability problem in [6].

▶ **Definition 4.4** (Lexicographically Maximum Oracle-3-satisfying Assignment). Let m, n be nonnegative integers, and $\Phi \colon \{0,1\}^{m+3n+3} \to \{0,1\}$ be a Boolean formula. For a Boolean function $X \colon \{0,1\}^n \to \{0,1\}$, define $F_\Phi(X)$ as the following Boolean formula:

$$\bigwedge_{w \in \{0,1\}^{m+3n}} \Phi(w, X(b_1), X(b_2), X(b_3)),$$

where $w = (y, (b_1, b_2, b_3)) \in \{0,1\}^m \times (\{0,1\}^n)^3$. A Boolean function $X \colon \{0,1\}^n \to \{0,1\}$ is said to be an assignment of F_Φ. For assignments $X, Y \colon \{0,1\}^n \to \{0,1\}$, we introduce the lexicographical ordering: X is less than Y if there exists an index $b \in \{0,1\}^n$ such that $X(b) < Y(b)$ and $X(b') = Y(b')$ for any $b' < b$. Let $V_\Phi \colon \{0,1\}^n \to \{0,1\}$ denote the lexicographically maximum assignment such that $F_\Phi(V_\Phi) = 1$ (*i.e.* the lexicographically maximum satisfying assignment of F_Φ); if there is no satisfying assignment, then define $V_\Phi(b) = 0$ for any $b \in \{0,1\}^n$.

The *lexicographically maximum oracle-3-satisfying assignment* is a problem of answering $V_\Phi(b_{\text{in}})$, given nonnegative integers m, n, a Boolean formula $\Phi \colon \{0,1\}^{m+3n+3} \to \{0,1\}$, and an index $b_{\text{in}} \in \{0,1\}^n$ as input.

We omit a proof of EXP^{NP}-completeness because this is a simple exponential-time analogue of the lexicographically maximum satisfying assignment language [20] (see also [6]).

Suppose that the input is a Boolean formula $\Phi \colon \{0,1\}^{m+3n+3} \to \{0,1\}$ and an index b_{in}, and that we have access to two oracles A_0 and A_1, one of which is honest.

Encoding Assignments by the Multilinear Extension

As with the proof of MIP = NEXP [6], we encode a satisfying assignment by the multilinear extension. Let $\mathbb{F}$ be a prime field such that $|\mathbb{F}|$ is sufficiently large (but is bounded by a polynomial in the input size). We regard $\{0,1\} \subseteq \mathbb{F}$ in the canonical way. We say that a function $f \colon \mathbb{F}^n \to \mathbb{F}$ is multilinear if it is a polynomial of degree at most 1 in each variable.

▶ **Proposition 4.5** (Multilinear Extension). *Let $f \colon \{0,1\}^n \to \mathbb{F}$ be an arbitrary function. Then, there exists a unique multilinear function $\widetilde{f} \colon \mathbb{F}^n \to \mathbb{F}$ such that f and $\widetilde{f}$ agree on $\{0,1\}^n$.*

Proof Sketch. For a complete proof, the reader is referred to [6, Proposition 4.4]. Here, we note that the extension $\widetilde{f}$ can be explicitly written as

$$\widetilde{f}(x) = \sum_{b \in \{0,1\}^n} f(b) \prod_{i=1}^{n} ((1 - x_i)(1 - b_i) + x_i b_i), \tag{2}$$

where $b = (b_1, \cdots, b_n)$ and $x = (x_1, \cdots, x_n) \in \mathbb{F}^n$. ◄

For the lexicographically maximum satisfying assignment $V_\Phi \colon \{0,1\}^n \to \{0,1\} \subseteq \mathbb{F}$, let $\widetilde{V_\Phi} \colon \mathbb{F}^n \to \mathbb{F}$ denote its multilinear extension.

We request the oracles to grant local access to $\widetilde{V_\Phi}$. Formally, we consider the following search problem: given a Boolean formula Φ, a prime $|\mathbb{F}|$, and $x \in \mathbb{F}^n$, the task is to output the value $\widetilde{V_\Phi}(x)$. We regard this problem as a decision problem in the standard way. (Specifically, given the inputs specified above and auxiliary inputs $k \in \mathbb{N}$ and $b \in \{0,1\}$, the task is to output one bit saying whether or not the kth bit of a binary representation of $\widetilde{V_\Phi}(x)$ is b.) The problem is still solvable in EXP^{NP}, by first computing V_Φ in EXP^{NP} and then computing the expression (2) straightforwardly in exponential time.

Therefore, the problem can be reduced to the original EXP^{NP}-complete problem; by using the EXP^{NP}-completeness, one can translate the problem of computing $\widetilde{V_\Phi}(x)$ into the original problem in polynomial time, and hence we can ask the oracles to output $\widetilde{V_\Phi}(x)$. Let $f_0, f_1 \colon \mathbb{F}^n \to \mathbb{F}$ denote the answers of the oracles A_0, A_1, respectively. Then, we have $f_i = \widetilde{V_\Phi}$ for an honest oracle A_i.

Although f_i is not necessarily multilinear for a dishonest oracle A_i, we can ensure that it is close to some multilinear function. This can be done by the multilinearity test, which was one of the main technical ingredients in the proof of MIP = NEXP [6]. For two functions $f, g \colon \mathbb{F}^n \to \mathbb{F}$ and a real number $\delta \in \mathbb{R}$, we say that f and g are δ-close if $\Pr_{x \in \mathbb{F}^n}[f(x) \neq g(x)] < \delta$.

▶ **Lemma 4.6** (Multilinearity Test [6]). *Let $n \in \mathbb{N}$ and $\mathbb{F}$ be a finite field. There exist a constant $\delta = n^{O(1)}/|\mathbb{F}|$ and an efficient probabilistic algorithm that, given oracle access to an arbitrary function $f \colon \mathbb{F}^n \to \mathbb{F}$,*
1. *accepts with probability 1 if f is multilinear, and*
2. *rejects with high probability if f is not δ-close to any multilinear function.*

We perform the multilinearity test for f_0 and f_1. Suppose that f_i is not δ-close to any multilinear function for a dishonest oracle A_i. Then, the multilinearity test fails and hence we can doubt A_i with high probability. Therefore, in what follows, we may assume that both f_0 and f_1 are δ-close to some multilinear functions $\hat{f}_0$ and $\hat{f}_1$, respectively (note that $\hat{f}_0$ and $\hat{f}_1$ are unique for small δ).

In reality, we have only access to f_0, f_1 instead of multilinear functions $\hat{f}_0, \hat{f}_1$. However, we may pretend to have access to the multilinear functions $\hat{f}_0, \hat{f}_1$, by using the random self-reducibility of multivariate low-degree polynomials (also known as the self-correction of the Reed-Muller code).

▶ **Lemma 4.7** (Self-correction; Beaver and Feigenbaum [8] and Lipton [21]). *There exists an efficient probabilistic algorithm that, given input $x \in \mathbb{F}^n$ and oracle access to a function $f \colon \mathbb{F}^n \to \mathbb{F}$ that is δ-close to a multilinear function $\hat{f} \colon \mathbb{F}^n \to \mathbb{F}$, outputs $\hat{f}(x)$ with probability at least $1 - \delta(n + 1)$.*

Proof. Let $a_0, \cdots, a_n$ be arbitrary distinct points in $\mathbb{F} \setminus \{0\}$. Pick a random point $y \in_R \mathbb{F}^n$. By the polynomial interpolation, find the univariate polynomial p of degree at most n such that $f(x + a_i \cdot y) = p(a_i)$ for all $i \in \{0, \cdots, n\}$, and output $p(0)$.

Since $x + a_i \cdot y$ is uniformly distributed on $\mathbb{F}^n$ for any fixed x and $a_i \neq 0$, it holds that $\hat{f}(x + a_i \cdot y) = f(x + a_i \cdot y)$ with probability at least $1 - \delta$. By the union bound, we have $p(a_i) = \hat{f}(x + a_i \cdot y)$ for each $i \in \{0, \cdots, n\}$ with probability at least $1 - \delta(n+1)$; thus we have $p(0) = \hat{f}(x)$ with probability at least $1 - \delta(n+1)$, because $\hat{f}$ is a polynomial of total degree at most n. ◄

▶ **Remark 4.8.** In the case of the proof of MIP = NEXP, the self-correcting algorithm was not needed; for the sum-check protocol, it is sufficient to evaluate a multilinear function $\hat{f}_i$ on random points $x \in_R \mathbb{F}^n$, rather than fixed points. In contrast, we need to evaluate a multilinear function $\hat{f}_i$ on points that are not uniformly distributed, during the binary search.

In the following, we pretend that the dishonest oracle A_i asserts that the satisfying assignment is $\hat{f}_i|_{\{0,1\}^n}$, instead of $f_i|_{\{0,1\}^n}$. (Note that it holds that $f_i|_{\{0,1\}^n} = \hat{f}_i|_{\{0,1\}^n} = V_\Phi$ for the honest oracle A_i.)

Identifying the Larger Assignment

We are now ready to describe how to identify the larger assignment. It is sufficient to show that we can find, with high probability, the lexicographically first index $z \in \{0,1\}^n$ such that $\hat{f}_0(z) \neq \hat{f}_1(z)$.

First, we check if $\hat{f}_0(b_{\text{in}}) = \hat{f}_1(b_{\text{in}})$: For each $i \in \{0,1\}$, run the self-correcting algorithm for f_i to obtain $\hat{f}_i(b_{\text{in}})$. If $\hat{f}_0(b_{\text{in}}) = \hat{f}_1(b_{\text{in}})$, then output it (which is surely the correct answer since $\hat{f}_i(b_{\text{in}}) = V_\Phi(b_{\text{in}})$ for the honest oracle A_i) and halt. Otherwise, perform the binary search described below.

We compute the lexicographically first disagreement $z = (z_1, \cdots, z_n) \in \{0,1\}^n$ one by one. For $j := 1$ to n, repeat the following: Suppose that we have computed $z_1, \cdots, z_{j-1}$. Pick a random point $u = (u_{j+1}, \cdots, u_n) \in_R \mathbb{F}^{n-j}$ uniformly at random. Define $x := (z_1, \cdots, z_{j-1}, 0, u_{j+1}, \cdots, u_n) \in \mathbb{F}^n$. For each $i \in \{0,1\}$, use the self-correcting algorithm for f_i to obtain $\hat{f}_i(x)$. If $\hat{f}_0(x) \neq \hat{f}_1(x)$, then set $z_j := 0$; else, set $z_j := 1$.

▶ **Claim 4.9.** Assume that $\hat{f}_0(b_{\text{in}}) \neq \hat{f}_1(b_{\text{in}})$. Let $z \in \{0,1\}^n$ denote the lexicographically first index such that $\hat{f}_0(z) \neq \hat{f}_1(z)$. Then, the binary search described above correctly computes z with probability at least $1 - \delta n(n+1) - \frac{n^2}{|\mathbb{F}|}$.

In particular, by setting $|\mathbb{F}|$ large enough, we can compute z with high probability.

Proof. Let $j \in \{1, \cdots, n\}$. Consider the jth iteration and assume that we have computed $z_1, \cdots, z_{j-1}$ correctly. For each $i \in \{0,1\}$, let $f_i' \colon \mathbb{F}^{n-j} \to \mathbb{F}$ be the multilinear function such that

$$f_i'(t_{j+1}, \cdots, t_n) = \hat{f}_i(z_1, \cdots, z_{j-1}, 0, t_{j+1}, \cdots, t_n),$$

for any $(t_{j+1}, \cdots, t_n) \in \mathbb{F}^{n-j}$. (The binary search tries to check if $f_0' \neq f_1'$ by the polynomial identity testing, and sets $z_j := 0$ if and only if $f_0' \neq f_1'$.)

If $z_j = 0$, then we have $f_0' \neq f_1'$ because $f_0'(z_{j+1}, \cdots, z_n) \neq f_1'(z_{j+1}, \cdots, z_n)$. The probability that the self-correcting algorithm outputs $\hat{f}_i(x)$ correctly is at least $1 - \delta(n+1)$ for a dishonest oracle A_i. By the Schwartz-Zippel lemma, the probability that $f_0'(u) \neq f_1'(u)$ for a random point $u \in_R \mathbb{F}^{n-j}$ is at least $1 - \frac{n-j}{|\mathbb{F}|} \geq 1 - \frac{n}{|\mathbb{F}|}$. Therefore, the algorithm sets $z_j := 0$ correctly with probability at least $1 - \delta(n+1) - \frac{n}{|\mathbb{F}|}$.

If $z_j = 1$, then it follows from the minimality of z that $f_0'(t) = f_1'(t)$ for every $t \in \{0,1\}^{n-j}$. Since f_0' and f_1' are multilinear, we have $f_0' = f_1'$ by the uniqueness of the multilinear extension

(Proposition 4.5) and hence $f_0'(u) = f_1'(u)$ holds for any $u \in \mathbb{F}^{n-j}$. Therefore, since the self-correcting algorithm outputs $\hat{f}_i(x)$ with probability at least $1 - \delta(n+1)$, the algorithm sets $z_j := 1$ correctly with probability at least $1 - \delta(n+1)$.

Overall, the algorithm computes z correctly with probability at least

$$\left(1 - \delta(n+1) - \frac{n}{|\mathbb{F}|}\right)^n \geq 1 - \delta n(n+1) - \frac{n^2}{|\mathbb{F}|}.$$

◄

We have computed the lexicographically first disagreement $z \in \{0,1\}$ such that $\hat{f}_0(z) \neq \hat{f}_1(z)$. Run the self-correcting algorithm to obtain $\hat{f}_0(z)$ and $\hat{f}_1(z)$. Without loss of generality (by swapping the oracles if $\hat{f}_0(z) > \hat{f}_1(z)$), we may assume that $\hat{f}_0(z) < \hat{f}_1(z)$.

Now we know, with high probability, that A_1 asserts the larger (presumably satisfying) assignment $\hat{f}_1|_{\{0,1\}^n} \colon \{0,1\}^n \to \mathbb{F}$.

Verifying the Satisfiability

All that remains is to verify that $\hat{f}_1|_{\{0,1\}^n}$ satisfies F_Φ, which can be done in the same way with a proof of MIP = NEXP. For completeness, we sketch a proof suggested in [6, Section 7.1] and observe that it can be done with the help of an EXP^NP-complete oracle.

Babai, Fortnow, Lund [6] used the sum-check protocol [22] to check whether or not an exponentially long assignment satisfies F_Φ. Basically, checking if an assignment $\hat{f}_1|_{\{0,1\}^n} \colon \{0,1\}^n \to \mathbb{F}$ satisfies a Boolean formula F_Φ reduces to checking if some low-degree polynomials $g \colon \mathbb{F}^l \to \mathbb{F}$ evaluate to 0 on $\{0,1\}^l$.

Let us arithmetize the Boolean formula $\Phi \colon \{0,1\}^{m+3n+3} \to \{0,1\}$ to a low-degree polynomial $\widetilde{\Phi} \colon \mathbb{F}^{m+3n+3} \to \mathbb{F}$ in the standard way, so that Φ and $\widetilde{\Phi}$ agree on $\{0,1\}^{m+3n+3}$ (see [6, Section 3.1]). Define $g^1 \colon \mathbb{F}^{m+3n} \to \mathbb{F}$ and $g^2 \colon \mathbb{F}^n \to \mathbb{F}$ as

$$g^1(w) := 1 - \widetilde{\Phi}\left(w, \hat{f}_1(b_1), \hat{f}_1(b_2), \hat{f}_1(b_3)\right), \tag{3}$$

$$g^2(b) := \hat{f}_1(b)\left(1 - \hat{f}_1(b)\right), \tag{4}$$

where $w = (y, (b_1, b_2, b_3)) \in \mathbb{F}^m \times (\mathbb{F}^n)^3$ and $b \in \mathbb{F}^n$. Note that since $\hat{f}_1$ and $\widetilde{\Phi}$ are low-degree polynomials, so are g^1 and g^2.

It is easy to see that $g^1(w) = 0$ and $g^2(b) = 0$ for any $w \in \{0,1\}^{m+3n}$ and $b \in \{0,1\}^n$ if and only if $\hat{f}_1|_{\{0,1\}^n}$ is a satisfying assignment of F_Φ. Indeed, $g^2(b) = 0$ forces $\hat{f}_1|_{\{0,1\}^n}$ to be a Boolean function (i.e. $\hat{f}_1(b) \in \{0,1\}$ for any $b \in \{0,1\}^n$), and $g^1(w) = 0$ means that $\Phi(w, \hat{f}_1(b_1), \hat{f}_1(b_2), \hat{f}_1(b_3))$ is true for any $w \in \{0,1\}^{m+3n}$.

We note that, given a random point w or b, we can compute the value of $g^1(w)$ or $g^2(b)$ with high probability by substituting f_1 for $\hat{f}_1$ in (3) or (4) (i.e. we do not need to use the self-correcting algorithm); for a random point $w \in_R \mathbb{F}^{m+3n}$, it holds that $g^1(w)$ computed by substituting f_1 in (3) and $g^1(w)$ are identical with probability at least $1 - 3\delta$.

Therefore, it is sufficient to show that we can check if each $g \in \{g^1, g^2\}$ vanishes on $\{0,1\}^l$, given access to a low-degree polynomial g. (Here, $l := m + 3n$ if $g = g^1$ and $l := n$ if $g = g^2$.) There are several ways to verify that $g \colon \mathbb{F}^l \to \mathbb{F}$ vanishes on $\{0,1\}^l$, including [6, Section 7.1] and [5, 15, 9]. Here, we follow the way of Feige, Goldwasser, Lovász, Safra, and Szegedy [15].

We reduce a task of checking if $g \colon \mathbb{F}^l \to \mathbb{F}$ vanishes on $\{0,1\}^l$ to a task of checking if a sum is equal to 0, the latter of which can be verified by the sum-check protocol (see [15,

Section 4.2.2] for more details): Pick a random point $t = (t_1, \cdots, t_l) \in_R \mathbb{F}^l$. Consider the following sum:

$$\sum_{w=(w_1, \cdots, w_l) \in \{0,1\}^l} g(w) \prod_{\{i \mid w_i = 1\}} t_i = \sum_{w \in \{0,1\}^l} g(w) \prod_{i \in \{1, \cdots, l\}} (w_i t_i + 1 - w_i). \tag{5}$$

If g vanishes on $\{0,1\}^l$, then this sum is equal to 0. Otherwise, regarding the left-hand side of (5) as a multilinear function on variables $t_1, \cdots, t_l$, the sum is not equal to 0 with probability at least $1 - \frac{l}{|\mathbb{F}|}$ by the Schwartz-Zippel lemma. Therefore, by defining a low-degree polynomial $h_t \colon \mathbb{F}^l \to \mathbb{F}$ as $h_t(w) := g(w) \prod_{i \in \{1, \cdots, l\}} (w_i t_i + 1 - w_i)$ for any $w \in \mathbb{F}^l$, it is sufficient to check if the sum of $h_t(w)$ over $w \in \{0,1\}^l$ is equal to 0, which can be done by the sum-check protocol.

We describe the sum-check protocol briefly (see [6, Section 3.2] for a detailed description): In order to check if $\sum_{w \in \{0,1\}^l} h_t(w) = 0$, pick a random point $r = (r_1, \cdots, r_l) \in_R \mathbb{F}^l$. Define a low-degree univariate polynomial $g_i \colon \mathbb{F} \to \mathbb{F}$ for each $i \in \{1, \cdots, l\}$ as

$$g_i(x) := \sum_{(w_{i+1}, \cdots, w_l) \in \{0,1\}^{l-i}} h_t(r_1, \cdots, r_{i-1}, x, w_{i+1}, \cdots, w_l)$$

and $g_0(x) := 0$. We request the oracle A_1 to reveal all the coefficients of the univariate polynomial g_i for all $i \in \{1, \cdots, l\}$. We trust A_1 if and only if $g_{i-1}(r_{i-1}) = g_i(0) + g_i(1)$ for each $i \in \{1, \cdots, l\}$ (*the Consistency Test*) and $g_l(r_l) = h_t(r)$ (*the Final Test*). Here, since r is a random point, we may evaluate $h_t(r)$ by using f_1 in place of $\hat{f}_1$ in (3) and (4).

We claim that the complexity of the honest oracle to output g_i is bounded by $\mathrm{EXP}^{\mathrm{NP}}$. Consider the following search problem: given a Boolean formula Φ, a prime $|\mathbb{F}|$, and $r, t \in \mathbb{F}^l$, the task is to output all the coefficients of g_i for all $i \in \{1, \cdots, l\}$ (which can be written in a binary representation of polynomial length), where $\widetilde{V_\Phi}$ is substituted for $\hat{f}_1$ in (3) and (4). Regarding this problem as a decision problem, one can easily show that the problem is computable in $\mathrm{EXP}^{\mathrm{NP}}$. Thus, we can request the oracle A_1 to output g_i.

Finally, we conclude the proof by analyzing the correctness (assuming that the binary search succeeded):

1. If A_1 is honest, then $\hat{f}_1 = f_1 = \widetilde{V_\Phi}$. Thus, each $g \in \{g^1, g^2\}$ vanishes on $\{0,1\}^l$, and hence the sum (5) is 0; therefore, we can trust A_1 with probability 1.

2. If A_1 is dishonest, then $\hat{f}_1$ does not constitute a satisfying assignment of F_Φ. (If it were a satisfying assignment, then $\hat{f}_1|_{\{0,1\}^n}$ would be a satisfying assignment larger than $\hat{f}_0|_{\{0,1\}^n} = V_\Phi$.) Thus, for some $g \in \{g^1, g^2\}$, the sum (5) is not 0 with probability at least $1 - \frac{l}{|\mathbb{F}|}$.

   Assume that the sum is not 0, and let $d \in \mathbb{N}$ be an upper bound on the degree of the low-degree polynomial h_t. Suppose that the dishonest oracle claimed that g_i is g_i' for each $i \in \{1, \cdots, l\}$. Assuming that the Consistency Tests pass (*i.e.* $g_{i-1}'(r_{i-1}) = g_i'(0) + g_i'(1)$ for each $i \in \{1, \cdots l\}$), it holds that $g_l'(r_l) \neq g_l(r_l) = h_t(r)$ with probability at least $1 - \frac{dl}{|\mathbb{F}|}$ (see [6, Section 3.2]). The probability that h_t can be evaluated correctly on a random point $r \in_R \mathbb{F}^l$ is at least $1 - 3\delta$. Thus, the Final Test (*i.e.* $g_l'(r_l) = h_t(r)$) fails with probability at least $1 - \frac{dl}{|\mathbb{F}|} - 3\delta$.

   Overall, we can doubt A_1 with probability at least $1 - \frac{dl}{|\mathbb{F}|} - 3\delta - \frac{l}{|\mathbb{F}|}$.

5 Deterministic Selector

This section is devoted to investigating a deterministic selector.

To prove the existence of a deterministic selector for a PSPACE-complete language (Theorem 1.4, Part 1), we show that a deterministic selector can be constructed based on downward self-reducibility:

▶ **Theorem 5.1.** *Any downward self-reducible language has a deterministic selector.*

Since there exists a downward self-reducible PSPACE-complete language, we immediately obtain a deterministic selector for any PSPACE-complete language.

Proof. Let L be a downward self-reducible language. Namely, there exists a polynomial-time oracle machine M such that

- $M^L(x) = L(x)$ for any $x \in \{0,1\}^*$, and
- M does not make any queries of length greater than or equal to $|x|$, on input $x \in \{0,1\}^*$.

The idea is to keep a string y such that $A_0(y) \neq A_1(y)$, and to run M^{A_0} and M^{A_1} to obtain another string q of length less than $|y|$ such that $A_0(q) \neq A_1(q)$. Consider the following algorithm: Given an input $x \in \{0,1\}^*$ and two oracles A_0, A_1, if $A_0(x) = A_1(x)$ then output it and halt. Else, let $y := x$ and repeat the following: Compute $M^{A_i}(y)$ for each $i \in \{0,1\}$. If $M^{A_0}(y) = M^{A_1}(y) =: b$, then we trust the oracle A_i such that $A_i(y) = b$ and output $A_i(x)$. Otherwise, let q be the first query that M^{A_0} and M^{A_1} make on input y such that $A_0(q) \neq A_1(q)$. (There exists such a q because $M^{A_0}(y) \neq M^{A_1}(y)$; moreover, it holds that $|q| < |y|$ by the definition of downward self-reducibility.) Then, we update $y := q$ and move on to the next iteration.

This algorithm runs in polynomial time, since $|y|$ decreases in each repetition.

We claim the correctness of the algorithm. It is easy to see that $A_0(y) \neq A_1(y)$ at the beginning of each repetition. Suppose that $M^{A_0}(y) = M^{A_1}(y) =: b$. Since A_0 or A_1 is equal to L, we have $b = M^{A_0}(y) = M^{A_1}(y) = M^L(y) = L(y)$, where the last equality holds by the definition of M. Moreover, there exists the unique $i \in \{0,1\}$ such that $A_i(y) = b$ because A_0 and A_1 disagree on y. Therefore, A_i is honest if and only if $A_i(y) = b \ (= L(y))$.   ◀

Then, we claim that any language with a deterministic selector is in PSPACE (Theorem 1.4, Part 2). We thereby prove that the supremum of the languages with a deterministic selector is PSPACE.

Proof of Part 2 of Theorem 1.4. Let L be a language with a deterministic selector S.

The idea is to regard a computation of S as a game played between the NO player and the YES player (which correspond to two oracles A_0 and A_1, respectively): On input $x \in \{0,1\}^*$, the YES player tries to convince the selector S that $x \in L$, whereas the NO player tries to convince S that $x \notin L$. The YES player chooses $A_1 \subseteq \{0,1\}^*$ such that $x \in A_1$, and the NO player chooses $A_0 \subseteq \{0,1\}^*$ such that $x \notin A_0$. Then, we simulate $S^{A_0,A_1}(x)$, and the YES player wins if and only if $S^{A_0,A_1}(x) = 1$.

It is easy to see that the YES player has a winning strategy if $x \in L$. Indeed, the YES player wins by setting $A_1 = L$; similarly, if $x \notin L$, then the NO player wins by setting $A_0 = L$. Therefore, it is sufficient to show that we can compute the player that has a winning strategy in PSPACE.

We may restate the game as follows: Simulate S on input x. If S makes a query x to $A_i \ (i \in \{0,1\})$, then answer it with i. If S makes a query $q \ (\neq x)$ to the oracle A_0, then the NO player gives an arbitrary answer; similarly, if S makes a query to A_1, then the YES player gives an arbitrary answer. (However, we require the players to behave in a consistent way: if S makes the same query more than once, then a player must give the same answer that the player answered in the past.)

Again, one can easily prove that the YES player has a winning strategy for this game if and only if $x \in L$.

Now we describe a polynomial-time alternating Turing machine that computes L: Simulate the game described above, while universally guessing the answers of the NO player and existentially guessing the answers of the YES player. Since a polynomial-time alternating machine can be simulated in PSPACE, it holds that $L \in \mathrm{PSPACE}$. ◀

6 Random Strings vs. Randomized Computation

In this section, we apply the notion of selector to the proof by Buhrman, Fortnow, Koucký, and Loff [11]. We thereby extend their result from $\{\mathrm{NP}, \mathrm{P}^{\#\mathrm{P}}, \mathrm{PSPACE}, \mathrm{EXP}\}$ to any classes whose complete languages have a selector (*e.g.* $\Sigma_i^{\mathrm{p}}, \Pi_i^{\mathrm{p}}, \mathrm{P}^{\#\mathrm{P}}, \mathrm{PSPACE}, \mathrm{EXP}$, and $\mathrm{EXP}^{\mathrm{NP}}$).

▶ **Theorem 6.1** (Extended Theorem 15 of [11]). *Let* $\alpha \colon \{0\}^* \to \{0,1\}^*$ *be a length preserving function,* $c > 0$ *be a constant such that* $\alpha(0^n) \notin \mathrm{i.o\text{-}EXP}/n - c\log n$, *and* $\mathcal{C}$ *be a complexity class such that there is a selector for some paddable* $\mathcal{C}$*-complete language* L. *If* $L \in \mathrm{P}/\alpha(0^{n^d})$ *for some* $d > 0$, *then* $\mathcal{C} \subseteq \mathrm{BPP}$.

Proof. Let M be a polynomial-time machine such that $L(x) = M\left(x, \alpha(0^{|x|^d})\right)$, and $G_n \subseteq \{0,1\}^{n^d}$ be the set of "good" advice:

$$G_n := \{\, r \in \{0,1\}^{n^d} \mid \forall x \in \{0,1\}^n, \ L(x) = M(x,r) \,\}.$$

Buhrman *et al.* [11] showed that $|G_n| \geq 2^{n^d}/n^{cd}$ by exploiting the high nonuniform complexity of advice $\alpha(0^{n^d})$.

As with Theorem 2.8, there exist a polynomial l and a selector S that identifies an honest oracle among $m := 2l(n)^{cd}$ oracles with probability at least $\frac{5}{6}$, and makes only queries of length exactly $l(n)$ on inputs of length n.

Consider the following probabilistic algorithm: On input $x \in \{0,1\}^n$, let l denote $l(n)$. We pick m random strings $r_1, \cdots, r_m \in_R \{0,1\}^{l^d}$ uniformly at random, and define oracles $A_i(q) = M(q, r_i)$, for any $i \in \{1, \cdots, m\}$ and for any $q \in \{0,1\}^l$. We simulate S on input x, answering its queries $q \in \{0,1\}^l$ to A_i by computing $M(q, r_i)$.

The probability that we fail to pick any "good" advice, namely $r_i \notin G_l$ for all i, is $(1 - |G_l|)^{2l^{cd}} \leq e^{-2l^{cd}/l^{cd}} < \frac{1}{6}$. Thus, we can output the correct answer with probability at least $\frac{2}{3}$ overall. ◀

7 Concluding Remarks

We state some open problems and possible directions for future work:

- Do there exist selectors for NEXP-complete languages or promise-$\mathrm{S}_2^{\mathrm{exp}}$-complete languages? In particular, it is interesting to close the gap between $\mathrm{EXP}^{\mathrm{NP}}$ and $\mathrm{S}_2^{\mathrm{exp}}$: although these classes seem "close" in some sense, $\mathrm{EXP}^{\mathrm{NP}}$ and $\mathrm{S}_2^{\mathrm{exp}}$ are very different in the known relationship with BPP; it is a notorious open problem whether $\mathrm{BPP} \neq \mathrm{EXP}^{\mathrm{NP}}$, whereas one can prove $\mathrm{BPP} \neq \mathrm{S}_2^{\mathrm{exp}}$.
- We proved that a property of removing short advice can be captured by the notion of selector. What about a property of removing advice of polynomial length?
- The result of MIP = NEXP was "scaled-down" to obtain the relationship with hardness of approximating cliques [15], and eventually the PCP theorem [4, 3] was established. Can we obtain such interesting applications of selectors, by scaling down the selector for $\mathrm{EXP}^{\mathrm{NP}}$-complete languages?

References

1   Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Trans. Comput. Theory*, 1(1):2:1–2:54, 2009.

2   Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 1st edition, 2009.

3   Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

4   Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

5   László Babai, Lance Fortnow, Leonid Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, STOC'91, pages 21–32, 1991.

6   László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Comput. Complex.*, 1:3–40, 1991.

7   Boaz Barak. A probabilistic-time hierarchy theorem for "slightly non-uniform" algorithms. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques*, RANDOM'02, pages 194–208, 2002.

8   Donald Beaver and Joan Feigenbaum. Hiding instances in multioracle queries. In *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science*, STACS'90, pages 37–48, 1990.

9   Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM J. Comput.*, 38(2):551–607, 2008.

10   Manuel Blum and Sampath Kannan. Designing programs that check their work. *J. ACM*, 42(1):269–291, 1995.

11   Harry Buhrman, Lance Fortnow, Michal Koucký, and Bruno Loff. Derandomizing from random strings. In *Proceedings of the 25th Annual Conference on Computational Complexity*, CCC'10, pages 58–63, 2010.

12   Harry Buhrman, Lance Fortnow, and Rahul Santhanam. Unconditional lower bounds against advice. In *Proceedings of the 36th International Colloquium on Automata, Languages, and Programming*, ICALP'09, pages 195–209, 2009.

13   Harry Buhrman and Steven Homer. Superpolynomial circuits, almost sparse oracles and the exponential hierarchy. In *Proceedings of the 12th Conference on Foundations of Software Technology and Theoretical Computer Science*, FSTTCS'92, pages 116–127, 1992.

14   Jin-Yi Cai. $S_2^p \subseteq ZPP^{NP}$. *J. Comput. Syst. Sci.*, 73(1):25–35, 2007.

15   Uriel Feige, Shafi Goldwasser, Laszlo Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, 1996.

16   Lance Fortnow and Adam Klivans. NP with small advice. In *Proceedings of the 20th Annual Conference on Computational Complexity*, CCC'05, pages 228–234, 2005.

17   Lance Fortnow, John Rompel, and Michael Sipser. On the power of multi-prover interactive protocols. *Theor. Comput. Sci.*, 134(2):545–557, 1994.

18   Richard Karp and Richard Lipton. Turing machines that take advice. *Enseign. Math*, 28(2):191–209, 1982.

19 Adam Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM J. Comput.*, 31(5):1501–1526, 2002.

20 Mark Krentel. The complexity of optimization problems. *J. Comput. Syst. Sci.*, 36(3):490–509, 1988.

21 Richard Lipton. New directions in testing. In Joan Feigenbaum and Michael Merritt, editors, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 2, pages 191–202. American Mathematical Society, 1991.

22 Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

23 Alexander Russell and Ravi Sundaram. Symmetric alternation captures BPP. *Comput. Complex.*, 7(2):152–162, 1998.

24 Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

25 Luca Trevisan and Salil Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Comput. Complex.*, 16(4):331–364, 2007.