Theory Comput Syst (2012) 51:248–265
DOI 10.1007/s00224-011-9365-0

Collapsing and Separating Completeness Notions Under Average-Case and Worst-Case Hypotheses

Xiaoyang Gu · John M. Hitchcock · A. Pavan

Published online: 8 October 2011
© Springer Science+Business Media, LLC 2011

Abstract This paper presents the following results on sets that are complete for NP.

(i) If there is a problem in NP that requires $2^{n^{\Omega(1)}}$ time at almost all lengths, then every many-one NP-complete set is complete under length-increasing reductions that are computed by polynomial-size circuits.
(ii) If there is a problem in co-NP that cannot be solved by polynomial-size nondeterministic circuits, then every many-one NP-complete set is complete under length-increasing reductions that are computed by polynomial-size circuits.
(iii) If there exist a one-way permutation that is secure against subexponential-size circuits and there is a hard tally language in NP $\cap$ co-NP, then there is a Turing complete language for NP that is not many-one complete.

Preliminary version appeared at 27th International Symposium on Theoretical Aspects of Computer Science.

Part of the research of X.G. was done while the author was at Iowa State University. Research supported in part by NSF grants 0652569 and 0728806.

Research of J.M.H. supported in part by NSF grants 0515313 and 0652601 and by an NWO travel grant. Part of this research was done while this author was on sabbatical at CWI.

Research of A.P. supported in part by NSF grants 0830479 and 0916797.

X. Gu
Linked In Corporation, Mountain View, USA
e-mail: xiaoyang@cs.iastate.edu

J.M. Hitchcock
Department of Computer Science, University of Wyoming, Laramie, USA
e-mail: jhitchco@cs.uwyo.edu

A. Pavan (✉)
Department of Computer Science, Iowa State University, Ames, USA
e-mail: pavan@cs.iastate.edu

Our first two results use worst-case hardness hypotheses whereas earlier work that showed similar results relied on average-case or almost-everywhere hardness assumptions. The use of average-case and worst-case hypotheses in the last result is unique as previous results obtaining the same consequence relied on almost-everywhere hardness results.

Keywords Computational complexity · NP-completeness · Turing completeness · Length-increasing reductions · Approximable sets

1 Introduction

It is widely believed that many important problems in NP such as satisfiability, clique, and discrete logarithm are exponentially hard to solve. Existence of such intractable problems has a bright side: research has shown that we can use this kind of intractability to our advantage to gain a better understanding of computational complexity, for derandomizing probabilistic computations, and for designing computationally-secure cryptographic primitives. For example, if there is a problem in EXP (such as any of the aforementioned problems) that has $2^{n^{\Omega(1)}}$-size worst-case circuit complexity (i.e., that for all sufficiently large n, no subexponential size circuit solves the problem correctly on all instances of size n), then it can be used to construct pseudorandom generators [9, 32]. Using these pseudorandom generators, BPP problems can be solved in deterministic quasipolynomial time. Similar average-case hardness assumptions on the discrete logarithm and factoring problems have important ramifications in cryptography. While these hardness assumptions have been widely used in cryptography and derandomization, more recently Agrawal [3] and Agrawal and Watanabe [5] showed that they are also useful for improving our understanding of NP-completeness. In this paper, we provide further applications of such hardness assumptions.

1.1 Length-Increasing Reductions

A language in NP is NP-complete if every language in NP is *reducible* to it. While there are several ways to define the notion of reduction, the most common definition uses polynomial-time computable many-one functions. Many natural problems that arise in practice have been shown to be NP-complete using polynomial-time computable many-one reductions. However, it has been observed that all known NP-completeness results hold when we restrict the notion of reduction. For example, SAT is complete under polynomial-time reductions that are one-to-one and length-increasing. In fact, all known many-one complete problems for NP are complete under this type of reduction [11]. This raises the following question: are there languages that are complete under polynomial-time many-one reductions but not complete under polynomial-time, one-to-one, length-increasing reductions? Berman [7] showed that every many-one complete set for E is complete under one-to-one, length-increasing reductions. Thus for E, these two completeness notions coincide. A weaker result is known for NE. Ganesan and Homer [19] showed that all NE-complete sets are complete via one-to-one reductions that are exponentially honest.

For NP, until recently there had not been any progress on this question. Agrawal [3] showed that if one-way permutations exist, then all NP-complete sets are complete via one-to-one, length-increasing reductions that are computable by polynomial-size circuits. Hitchcock and Pavan [23] showed that NP-complete sets are complete under length-increasing P/poly reductions under the measure hypothesis on NP [30]. Recently Buhrman et al. improved the latter result to show that if the measure hypothesis holds, then all NP-complete sets are complete via length-increasing, polynomial-time computable functions that require $O(\log\log n)$ bits of advice [13]. More recently, Agrawal and Watanabe [5] showed that if regular one-way functions exist, then all NP-complete sets are complete via one-one, length-increasing, P/poly-computable reductions. All the hypotheses used in these works require the existence of an *almost-everywhere hard* language or an *average-case hard* language in NP.

In the first part of this paper, we consider hypotheses that only concern the *worst-case hardness* of languages in NP. Our first hypothesis concerns the deterministic time complexity of languages in NP. We show that if there is a language in NP for which every correct algorithm spends more than 2^{n^ϵ} time at almost all lengths, then NP-complete languages are complete via length-increasing P/poly-computable reductions. The second hypothesis concerns nondeterministic circuit complexity of languages in co-NP. We show that if there is a language in co-NP that cannot be solved by nondeterministic polynomial-size circuits, then all NP-complete sets are complete via length-increasing P/poly-computable reductions. For more formal statements of the hypotheses, we refer the reader to Sect. 3. We stress that these hypotheses require only worst-case hardness. The worst-case hardness is of course required at every length, a technical condition that is necessary in order to build a reduction that works at every length rather than just infinitely often.

1.2 Turing Reductions versus Many-One Reductions

In the second part of the paper we study the completeness notion obtained by allowing a more general type of reduction—Turing reduction. Informally, with Turing reductions an instance of a problem can be solved by asking polynomially many (adaptive) queries about the instances of the other problem. A language in NP is Turing complete if there is a polynomial-time Turing reduction to it from every other language in NP. Though many-one completeness is the most commonly used completeness notion, Turing completeness also plays an important role in complexity theory. Several properties of Turing complete sets are closely tied to the separation of complexity classes. For example, Turing complete sets for EXP are sparse if and only if EXP has polynomial-size circuits. Moreover, to capture our intuition that a complete problem is easy, then the entire class is easy, Turing reductions seem to be the "correct" reductions to define completeness. In fact, the seminal paper of Cook [15] used Turing reductions to define completeness, though Levin [28] used many-one reductions.

This raises the question of whether there is a Turing complete language for NP that is not many-one complete. Ladner, Lynch and Selman [29] posed this question in 1975, thus making it one of the oldest problems in complexity theory. This question is completely resolved for exponential time classes such as EXP and NEXP [14, 37]. We know that for both these classes many-one completeness differs from Turing-completeness. However progress on the NP side has been very slow. Lutz and Mayordomo [31] were the first to provide evidence that Turing completeness differs from many-one completeness. They showed that if the measure hypothesis holds, then the completeness notions differ. Since then a few other weaker hypotheses have been used to achieve the separation of Turing completeness from many-one completeness [4, 24, 33–35].

All the hypotheses used in the above works are considered "strong" hypotheses as they require the existence of an *almost everywhere hard* language in NP. That is, there is a language L in NP and every algorithm that decides L takes exponential-time on *all but finitely many* strings. A drawback of these hypotheses is that we do not have any candidate languages in NP that are believed to be almost everywhere hard.

It has been open whether we can achieve the separation using more believable hypotheses that involve average-case hardness or worst-case hardness. None of the proof techniques used earlier seem to achieve this, as they crucially depend on the almost everywhere hardness.

In this paper, for the first time, we achieve the separation between Turing completeness and many-one completeness using average-case and worst-case hardness hypotheses. We consider two hypotheses. The first hypothesis states that there exist 2^{n^ϵ}-secure one-way permutations and the second hypothesis states that there is a language in $\mathrm{NEEE} \cap \mathrm{coNEEE}$ that can not be solved in triple exponential time with logarithmic advice, i.e., $\mathrm{NEEE} \cap \mathrm{coNEEE} \not\subseteq \mathrm{EEE}/\log$. We show that if both of these hypothesis are true, then there is a Turing complete language in NP that is not many-one complete.

The first hypothesis is an average-case hardness hypothesis and has been studied extensively in past. The second hypothesis is a worst-case hardness hypothesis. At first glance, this hypothesis may look a little esoteric, however, it is only used to obtain hard tally languages in $\mathrm{NP} \cap \text{co-NP}$ that are sufficiently sparse. Similar hypotheses involving double and triple exponential-time classes have been used earlier in the literature [8, 16, 17, 22].

We use length-increasing reductions as a tool to achieve the separation of Turing completeness from many-one completeness. We first show that if one-way permutations exist then NP-complete sets are complete via length-increasing, polynomially-bounded, quasipolynomial-time computable reductions. We then show that if the second hypothesis holds, then there is a Turing complete language for NP that is not many-one complete via length-increasing, polynomially-bounded, quasipolynomial-time computable reductions. Combining these two results we obtain our separation result.

2 Preliminaries

In the paper, we use the binary alphabet $\Sigma = \{0, 1\}$. Given a language A, A_n denotes the characteristic sequence of A at length n. We also view A_n as a boolean function from Σ^n to Σ. We use $A^{\leq n}$ to denote the set of strings in A whose length is at most n, and use $A^{=n}$ to denote the set of strings in A whose length is n.

For languages A and B, we say that $A = \mathrm{io}\, B$, if $A_n = B_n$ for infinitely many n. For a complexity class $\mathcal{C}$, we say that $A \in \mathrm{io}\,\mathcal{C}$ if there is a language $B \in \mathcal{C}$ such that $A = \mathrm{io}\, B$.

For a boolean function $f : \Sigma^n \to \Sigma$, $CC(f)$ is the smallest number s such that there is circuit of size s that computes f. A function f is quasipolynomial time computable (QP-computable) if can be computed deterministically in time $O(2^{\log^{O(1)} n})$. We will use the triple exponential time class EEE $=$ DTIME$(2^{2^{2^{O(n)}}})$, and its nondeterministic counterpart NEEE.

A language A is in NP/poly if there is a polynomial-size circuit family $C = (C_1, C_2, \dots)$ and a polynomial p such that for every x, x is in A if and only if there is a y of length $p(|x|)$ such that $C_{|x|}(x, y) = 1$. Let $f : \mathbb{N} \to \mathbb{N}$ be a function such that $f(n) < n$.

Definition A language L is in EEE$/f(n)$ if there is a function $g : 0^* \to \Sigma^*$ such that $|g(0^n)| \le f(n)$ and a triple exponential time algorithm M such that x is in L if and only if M accepts $\langle x, g(0^{|x|})\rangle$.

We define EEE$/O(\log n)$ to be $\bigcup_{c>0}$ EEE$/c \log n$.

Definition Given a language A, let *fold*(A) be a language such that for every n, *fold*$(A)_n = A_0 A_1 \cdots A_{n-1} 0$.

Our proofs make use a variety of results from approximable sets, instance compression, derandomization and hardness amplification. We mention the results that we need.

Definition 1 A language A is *$t(n)$-time 2-approximable* [6] if there is a function f computable in time $t(n)$ such that for all strings x and y, $f(x, y) \neq A(x)A(y)$.

A language A is *io-lengthwise $t(n)$-time 2-approximable* if there is a function f computable in time $t(n)$ such that for infinitely many n, for every pair of n-bit strings x and y, $f(x, y) \neq A(x)A(y)$. We say that f is the *approximator* of A.

Let A be a io-lengthwise $t(n)$-time 2-approximable and let f be its approximator. Given a natural number $m > 0$, we say that A *is 2-approximable at length m*, if for every pair of m-bit strings x and y $f(x, y) \neq A(x)A(y)$.

Amir, Beigel and Gasarch [2] proved that every polynomial-time 2-approximable set is in P/poly. Their proof also implies the following extension.

Theorem 2.1 [2] *If A is io-lengthwise $t(n)$-time 2-approximable, then for infinitely many n, $CC(A_n) \le t^2(n)$.*

If a language is disjunctively self-reducible and is 2-approximable, then it is in P. See the survey paper by Buhrman, Fortnow and Thierauf [10] for a proof. Though this proof carries overs to other time-bounds, it does not carry over to the infinitely-often setting. This is because the proof makes use of the fact that the approximator has the ability to compare strings at different lengths. However, if we use folded language A, then the proof carries through.

Theorem 2.2 *Let A be a disjunctive self-reducible language. If fold(A) is io-lengthwise $t(n)$-time 2-approximable, then there is an algorithm M that decides A and M runs in time $t^2(n)$ for infinitely many n. In addition, if fold(A) is 2-approximable at length n, then M runs in $t^2(n)$ time on all strings of length $n-1$.*

Given a language H' in co-NP, let H be $\{\langle x_1, \dots, x_n\rangle \mid |x_1| = \dots = |x_n| = n, x_i \in H', 1 \le i \le n\}$. Observe that a n-tuple consisting of strings of length n can be encoded by a string of length n^2. From now we view a string of length n^2 as an n-tuple of strings of length n.

Theorem 2.3 [12, 18] *Let H and H' be defined as above. Suppose there is a language L, a polynomial-size circuit family $\{C_m\}$, and a polynomial p such that for infinitely many n, for every $x \in \Sigma^{n^2}$, x is in H if and only if there is a string y of length $p(n)$ such that $C(x, y)$ is in $L^{\le n}$. Then H' is in* io NP/poly.

The proof of Theorem 2.3 is similar to the proofs in [12, 18]. The difference is rather than having a polynomial-time many-one reduction, here we have a NP/poly many-one reduction which works infinitely often. The nondeterminism and advice in the reduction can be absorbed into the final NP/poly decision algorithm. The NP/poly decision algorithm correctly works at infinitely many lengths, corresponding to the lengths at which the NP/poly reduction works.

Definition A function $f : \{0,1\}^n \to \{0,1\}^m$ is *s-secure* if for every $\delta < 1$, every $t \le \delta s$, and every circuit $C : \{0,1\}^n \to \{0,1\}^m$ of size t, $\Pr[C(x) = f(x)] \le 2^{-m} + \delta$. A function $f : \{0,1\}^* \to \{0,1\}^*$ is *$s(n)$-secure* if it is $s(n)$-secure at all but finitely many lengths n.

Definition An *$s(n)$-secure one-way permutation* is a polynomial-time computable bijection $\pi : \{0,1\}^* \to \{0,1\}^*$ such that $|\pi(x)| = |x|$ for all x and π^{-1} is $s(n)$-secure.

Under widely believed average-case hardness assumptions about the hardness of the RSA cryptosystem or the discrete logarithm problem, there is a secure one-way permutation [21].

Definition A *pseudorandom generator (PRG) family* is a collection of functions $G = \{G_n : \{0,1\}^{m(n)} \to \{0,1\}^n\}$ such that G_n is uniformly computable in time $2^{O(m(n))}$ and for every circuit of C of size n,

$$\left| \Pr_{x \in \{0,1\}^n}[C(x) = 1] - \Pr_{y \in \{0,1\}^{m(n)}}[C(G_n(y))] = 1 \right| \le \frac{1}{n}.$$

There are many results that show that the existence of hard functions in exponential time implies PRGs exist. We will use the following.

Theorem 2.4 [26, 32] *If there is a language A in* E *such that $CC(A_n) \ge 2^{n^{\epsilon}}$ for all sufficiently large n, then there exist a constant k and a PRG family $G = \{G_n : \{0,1\}^{\log^k n} \to \{0,1\}^n\}$.*

3 Length-Increasing Reductions

In this section we provide evidence that many-one complete sets for NP are complete via length-increasing reductions. We use the following hypotheses.

Hypothesis 1 There is a language L in NP and a constant $\epsilon > 0$ such that L is not in $\mathrm{io}\,\mathrm{DTIME}(2^{n^\epsilon})$.

Informally, this means that every algorithm that decides L takes more than 2^{n^ϵ}-time on at least one string at every length.

Hypothesis 2 There is a language L in co-NP such that L is not in $\mathrm{io}\,\mathrm{NP/poly}$.

This means that every nondeterministic polynomial size circuit family that attempts to solve L is wrong on at least one string at each length.

We will first consider the following variant of Hypothesis 1.

Hypothesis 3 There is a language L in NP and a constant $\epsilon > 0$ such that for all but finitely many n, $CC(L_n) > 2^{n^\epsilon}$.

We will first show that if Hypothesis 3 holds, then NP-complete sets are complete via length-increasing, P/poly reductions. Then we describe how to modify the proof to derive the same consequence under Hypothesis 1. We do this because the proof is much cleaner with Hypothesis 3 and contains all essential ideas. To use Hypothesis 1 we have to fix encodings of boolean formulas and work with folded languages.

3.1 If NP has Subexponentially Hard Languages

Theorem 3.1 *If there is a language L in* NP *and an $\epsilon > 0$ such that for all but finitely many n, $CC(L_n) > 2^{n^\epsilon}$, then all* NP-*complete sets are complete via length-increasing*, P/poly *reductions.*

Proof Let A be a NP-complete set that is decidable in time 2^{n^k}. Let L be a language in NP that requires 2^{n^ϵ}-size circuits at every length. Since SAT is complete via polynomial-time, length-increasing reductions, it suffices to exhibit a length-increasing, P/poly-reduction from SAT to A.

Let $\delta = \frac{\epsilon}{2k}$. Consider the following intermediate language

$$S = \{\langle x, y, z\rangle \mid |x| = |z| = \lceil |y|^{1/\delta}\rceil, \mathtt{MAJ}[L(x), \mathrm{SAT}(y), L(z)] = 1\},$$

where $\mathtt{MAJ}$ is the majority function.

Clearly S is in NP. Since A is NP-complete, there is a many-one reduction f from S to A. We will first show that at every length n there exist strings on which the reduction f must be length-increasing. Let

$$T_n = \{\langle x, z\rangle \mid |x| = |z| = \lceil n^{1/\delta}\rceil, L(x) \neq L(z),\ \forall y \in \{0,1\}^n |f(\langle x, y, z\rangle)| > n\}. \quad \Box$$

Lemma 3.2 *For all but finitely many n, $T_n \neq \emptyset$.*

Assuming that the above lemma holds, we complete the proof of the theorem. Given a length n, fix an (arbitrary) ordering of tuple from T_n. Let $\langle x_n, z_n \rangle$ be the first tuple from T_n. Consider the following reduction from SAT to A: Given a string y of length n, the reduction outputs $f(\langle x_n, y, z_n \rangle)$. Given x_n and y_n as advice, this reduction can be computed in polynomial time. Since $|x_n|$ and $|z_n|$ are polynomial in n, this is a P/poly reduction.

By the definition of T_n, $L(x_n) \neq L(z_n)$. Thus $y \in$ SAT if and only if $\langle x_n, y, z_n \rangle \in S$, and so y is in SAT if and only if $f(\langle x_n, y, z_n \rangle)$ is in A. Again, by the definition of T_n, for every y of length n, the length of $f(\langle x_n, y, z_n \rangle)$ is bigger than n. Thus there is a P/poly-computable, length-increasing reduction from SAT to A. This, together with the proof of Lemma 3.2 we provide next, completes the proof of Theorem 3.1. □

Proof of Lemma 3.2 Suppose $T_n = \emptyset$ for infinitely many n. We will show that this yields a length-wise 2-approximable algorithm for L at infinitely many lengths. This enables us to contradict the hardness of L. Consider the following algorithm:

1. Input x, z with $|x| = |z| = m$.
2. Find n such that $\lceil n^{1/\delta} \rceil = m$. If no such n exists, then output 10.
3. Find a y of length n such that $|f(\langle x, y, z \rangle)| \leq n$.
4. If no such y is found, Output 10.
5. If y is found, then solve the membership of $f(\langle x, y, z \rangle)$ in A. If $f(\langle x, y, z \rangle) \in A$, then output 00, else output 11.

We first bound the running time of the algorithm. Step 3 takes $O(2^{m^\delta})$ time. In Step 5, we decide the membership of $f(\langle x, y, z \rangle)$ in A. This step is reached only if the length of $f(\langle x, y, z \rangle)$ is at most n. Thus the time taken for this step is $(2^n)^k \leq 2^{m^{\epsilon}/2}$ time. Thus the total time taken by the algorithm is bounded by $2^{m^\epsilon/2}$.

Consider a length n at which $T_n = \emptyset$. Let x and z be any two strings of length $m = \lceil n^{1/\delta} \rceil$. Suppose for every y of length n, the length of $f(\langle x, y, z \rangle)$ is at least n. Then it must be the case that $L(x) = L(z)$, otherwise the tuple $\langle x, z \rangle$ belongs to T_n. Thus if the above algorithm fails to find y in Step 3, then $L(x)L(z) \neq 10$.

Suppose the algorithm succeeds in finding a y in Step 3. If $f(\langle x, y, z \rangle) \in A$, then at least one of x or z must belong to L. Thus $L(x)L(z) \neq 00$. Similarly, if $f(\langle x, y, z \rangle) \notin A$, then at least one of x or z does not belong to L, and so $L(x)L(z) \neq 11$.

Thus L is 2-approximable at length $m = \lceil n^{1/\delta} \rceil$. If there exist infinitely many lengths n, at which T_n is empty, then L is infinitely-often, length-wise, $2^{m^\epsilon/2}$-time approximable. By Theorem 2.1, L has circuits of size 2^{m^ϵ} at infinitely many lengths. □

We will now describe how to modify the proof if we assume that Hypothesis 1 holds. Let L be the hard language guaranteed by the hypothesis. This means that there exists an $\epsilon > 0$ such that L is not in io DTIME(2^{n^ϵ}). Fix an encoding of 3CNF formulas such that formulas with same numbers of variables can be encoded as strings of same length. We will work with *fold*(3-SAT).

Trivially, *fold*(3-SAT) is also NP-complete and there is a reduction f and a constant $r \geq 1$ from L to *fold*(3-SAT) such that all strings of length n are mapped to strings of length n^r. Let

$$H = \mathit{fold}(\text{3-SAT}) \cap \bigcup_n \Sigma^{n^r}.$$

The following observation is easy to prove.

Observation 3.3 *If there is an algorithm $\mathcal{A}$ that decides H such that for infinitely many n the algorithm $\mathcal{A}$ runs in 2^{n^ϵ} time on all strings of length n^r, then L is in* $\mathrm{io\,DTIME}(2^{n^\epsilon})$.

Now H plays the role of L from previous theorem and the proof proceeds same as before except that we use H instead of L. Our intermediate language will be

$$S = \{\langle x, y, z\rangle \mid |x| = |z| = (\lceil |y|^{1/\delta}\rceil)^r, \text{ and } \mathtt{MAJ}[H(x), \mathrm{SAT}(y), H(z)] = 1\}.$$

Clearly S is in NP and since A is NP-complete there is a reduction g from S to A. Consider the set T_n as before.

$$T_n = \{\langle x, z\rangle \mid H(x) \neq H(z), |x| = |z| = (\lceil n^{1/\delta}\rceil)^r, \forall y \in \Sigma^n |g(\langle x, y, z\rangle)| > n\}.$$

We can show the following.

Lemma 3.4 *For all but finitely many n, $T_n \neq \emptyset$.*

As before, we can show that if T_n is empty for infinitely many n, then H is infinitely-often, length-wise, $2^{n^{\epsilon/2}}$-time approximable. More precisely, we can show that for every length n at which T_n is empty, H is 2-approximable in time $2^{n^{\epsilon/2}}$ at length n^r. Since 3-SAT is disjunctive self-reducible and H coincides with *fold*(3-SAT) at lengths of form n^r, it follows from Theorem 2.2 that H can be decided in time 2^{n^ϵ}-time at infinitely many lengths of the form n^r. By Observation 3.3 this implies that L is in $\mathrm{io\,DTIME}(2^{n^\epsilon})$.

If T_n is not empty at all but finitely many lengths, then there is a P/poly, length-increasing reduction from SAT to A. Thus we have the following theorem.

Theorem 3.5 *If there is a language in* NP *that is not in* $\mathrm{io\,DTIME}(2^{n^\epsilon})$, *then all* NP-*complete sets are complete via length-increasing* P/poly *reductions.*

3.2 If co-NP is Hard for Nondeterministic Circuits

In this subsection we show that Hypothesis 2 also implies that all NP-complete sets are complete via length-increasing reductions.

Theorem 3.6 *If there is a language in* co-NP *that is not in* io NP/poly, *then* NP-*complete sets are complete via* P/poly-*computable*, *length-increasing reductions.*

Proof We find it convenient to work with co-NP rather than NP. We will show that all co-NP-complete languages are complete via P/poly, length-increasing reductions.

Let H' be a language in co-NP that is not in io NP/poly. Let H be

$$\{\langle x_1, \ldots, x_n\rangle \mid \forall i\ 1 \le i \le n, [x_i \in H' \text{ and } |x_i| = n]\}.$$

Note that every n-tuple that may potentially belong to H can be encoded by a string of length n^2. From now we view a string of length n^2 as an n-tuple of n bit strings.

Let $S = 0H' \cup 1\overline{\mathrm{SAT}}$. Observe that S is in co-NP and S is not in io NP/*poly*. Since $1\overline{\mathrm{SAT}}$ is many-one complete for co-NP via length-increasing reductions, S is also co-NP-complete via length-increasing reductions. Let A be any co-NP-complete language. It suffices to exhibit a length-increasing reduction from S to A.

Consider the following intermediate language:

$$L = \{\langle x, y, z\rangle \mid |x| = |z| = |y|^2, \mathtt{MAJ}[x \in H, y \in S, z \in H] = 1\}.$$

Clearly the above language is in co-NP. Let f be a many-one reduction from L to A. As before we will first show at every length n that there exits strings x and z such that for every y in S the length of $f(\langle x, y, z\rangle)$ is at least n. □

Lemma 3.7 *For all but finitely many n, there exist two strings x_n and z_n of length n^2 with $H(x_n) \ne H(z_n)$ and for every $y \in S^{=n}$, $|f(\langle x_n, y, z_n\rangle)| > n$.*

Proof Suppose not. Then there exist infinitely many lengths n at which for every pair of strings (of length n^2) x and z with $H(x) \ne H(z)$, there exist a y of length n such that $y \in S$ and $|f(x, y, z)| \le n$.

From this we obtain a NP/poly-reduction from H to A such that for infinitely many n, for every x of length n^2, $|f(x)| \le n$. By Theorem 2.3, this implies that H' is in io NP/poly. We now describe the reduction. Given n let z_n be a string (of length n^2) that is not in H.

1. Input x, $|x| = n^2$. Advice: z_n.
2. Guess a string y of length n.
3. If $|f(\langle x, y, z_n\rangle)| > n$, the output $\perp$.
4. Output $f(\langle x, y, z_n\rangle)$.

Suppose $x \in H$. Since $z_n \notin H$, there exists a string y of length n such that $y \in S$ and $|f(\langle x, y, z_n\rangle)| \le n$. Consider a path that correctly guesses such a y. Since $z_n \notin H$, and $y \in S$, $\langle x, y, z_n\rangle \in A$. Thus $f(\langle x, y, z_n\rangle) \in A^{\le n}$. Thus there exists at least one path on which the reduction outputs a string from $A^{\le n}$. Now consider the case $x \notin H$. On any path, the reduction either outputs $\perp$ or outputs $f(\langle x, y, z_n\rangle)$. Since both z_n and x are not in H, $\langle x, y, z_n\rangle \notin A$. Thus $f(\langle x, y, z_n\rangle) \notin A$ for any y.

Thus there is a NP/poly many-one reduction from H to A such that for infinitely many n, the output of the reduction, on strings of length n^2, on any path is at most n. By Theorem 2.3, this places H' in io NP/poly.

Thus for all but finitely many lengths n, there exist strings x_n and z_n of length n^2 with $H(x_n) \ne H(z_n)$ and for every $y \in S^n$, the length of $f(\langle x_n, y, z_n\rangle)$ is at least n. □

This suggests the following reduction h from S to A. The reduction will have x_n and z_n as advice. Given a string y of length n, the reductions outputs $f(\langle x_n, y, z_n\rangle)$. This reduction is clearly length-increasing and is length-increasing on every string from S. Thus we have the following lemma.

Lemma 3.8 *Consider the above reduction h from S to A, for all $y \in S$, $|h(y)| > |y|$.*

Now we show how to obtain a length-increasing reduction on all strings. We make the following crucial observation.

Observation 3.9 *For all but finitely many n, there is a string y_n of length n such that $y_n \notin S$ and $|f(\langle x_n, y_n, z_n\rangle)| > n$.*

Proof Suppose not. This means that for infinitely many n, for every y from $\overline{S} \cap \Sigma^n$, the length of $f(\langle x_n, y, z_n\rangle)$ is less than n. Now consider the following algorithm that solves S. Given a string y of length n, compute $f(\langle x_n, y, z_n\rangle)$. If the length of $f(\langle x_n, y, z_n\rangle) > n$, then accept y else reject y.

The above algorithm can be implemented in P/poly given x_n and z_n as advice. If $y \in S$, then we know that the length of $f(\langle x_n, y, z_n\rangle)$ is bigger than n, and so the above algorithm accepts. If $y \notin S$, then by our assumption, the length of $f(\langle x_n, y, z_n\rangle)$ is at most n. In this case the algorithm rejects y. This shows that S is in io P/poly which in turn implies that H' is in io P/poly. This is a contradiction. □

Now we are ready to describe our length increasing reduction from S to A. At length n, this reduction will have x_n, y_n and z_n as advice. Given a string y of length n, the reduction outputs $f(\langle x_n, y, z_n\rangle)$ if the length of $f(\langle x_n, y, z_n\rangle)$ is more than n. Else, the reduction outputs $f(\langle x_n, y_n, z_n\rangle)$.

Since $H(x_n) \neq H(z_n)$, $y \in S$ if and only if $f(\langle x_n, y, z_n\rangle) \in A$. Thus the reduction is correct when it outputs $f(\langle x_n, y, z_n\rangle)$. The reduction outputs $f(\langle x_n, y_n, z_n\rangle)$ only when the length of $f(\langle x_n, y, z_n\rangle)$ is at most n. We know that in this case $y \notin S$. Since $y_n \notin S$, $f(\langle x_n, y_n, z_n\rangle \notin A$.

Thus we have a P/poly-computable, length-increasing reduction from S to A. Thus all co-NP-complete languages are complete via P/poly, length-increasing reductions. This immediately implies that all NP-complete languages are complete via P/poly-computable, length-increasing reductions. □

4 Separation of Completeness Notions

In this section we consider the question of whether the Turing completeness differs from many-one completeness for NP under two plausible complexity-theoretic hypotheses:

(1) There exists a 2^{n^ϵ}-secure one-way permutation.
(2) $\mathrm{NEEE} \cap \mathrm{coNEEE} \not\subseteq \mathrm{EEE}/\log$.

It turns out that the first hypothesis implies that every many-one complete language for NP is complete under a particular kind of length-increasing reduction, while the second hypothesis provides us with a specific Turing complete language that is not complete under the same kind of length-increasing reductions. Therefore, the two hypotheses together separate the notions of many-one and Turing completeness for NP.

Theorem 4.1 *If both of the above hypotheses are true, there is a language that is polynomial-time Turing complete for* NP *but not polynomial-time many-one complete for* NP.

Theorem 4.1 is immediate from Lemma 4.2 and Lemma 4.3 below.

Lemma 4.2 *Suppose 2^{n^ϵ}-secure one-way permutations exist. Then for every* NP-*complete language A and every $B \in$* NP, *there is a quasipolynomial-time computable, polynomially-bounded, length-increasing reduction from B to A.*

A function f is *polynomially-bounded* if there is a polynomial p such that the length of $f(x)$ is at most $p(|x|)$ for every x.

Lemma 4.3 *If* NEEE ∩ coNEEE $\not\subseteq$ EEE/ log, *then there is a polynomial-time Turing complete set for* NP *that is not many-one complete via quasipolynomial-time computable, polynomially-bounded, length-increasing reductions.*

To show Lemma 4.2 we use results and ideas from Agrawal [3].

Let $\gamma > 0$ and let $S \subseteq \{0,1\}^n$. A function g is *γ-sparsely many-one* on S if for every $x \in S$,

$$|g^{-1}(g(x)) \cap \{0,1\}^n| \le \frac{2^n}{2^{n^\gamma}}.$$

Lemma 4.4 (Agrawal [3]) *Suppose 2^{n^ϵ}-secure one-way permutations exists. For every NP-complete language L for every set S in* NP, *there is a reduction from f from S to L that is $\frac{\epsilon}{2}$-sparsely many-one on $\{0,1\}^n$ for all $n \in \mathbb{N}$.*

Proof of Lemma 4.2 Let $\delta = \epsilon/3$. Consider the following set

$$S = \{\langle x, y\rangle \mid x \in B \text{ and } |y| = n^{1/\delta}\}.$$

For $|x| = n$, let $m = |\langle x, y\rangle|$. By Lemma 4.4, there is a reduction f from S to A that is $\frac{\epsilon}{2}$-sparsely many-one on Σ^m for every m. For any x of length n, let $S_x = \{y \in \Sigma^{n^{1/\delta}} \mid |f(\langle x, y\rangle)| \le n\}$. It follows that the size of S_x is at most $\frac{2^{n+m+1}}{2^{m^{\epsilon/2}}}$. Thus for at least $1 - 2^{-2n^{3/2}}$ fraction of strings y from $\{0,1\}^{n^{1/\delta}}$, $|f(\langle x, y\rangle)| > n$. If we randomly pick a $y \in \{0,1\}^{n^{1/\delta}}$, then with very high probability $|f(\langle x, y\rangle)| > n$.

We can derandomize the above process. If 2^{n^ϵ}-secure one-way permutations exist, then EXP does not have 2^{n^ϵ}-size circuits. Thus by Theorem 2.4, there exists a pseudorandom generator family $\{G_n\}$ that map strings of length $\log^k n$ to strings of length n. These pseudorandom generators are computable in $O(2^{\log^d n})$ time for some constant d.

Now the length-increasing reduction from B works as follows. Given x as input of length n, let $t = n^{1/\delta}$. Cycle through all seeds s of length $\log^k t$ till the length of $f(\langle x, G_t(s)\rangle)$ is bigger than n. Output $f(\langle x, G_t(s)\rangle)$. Since G_t is a pseudorandom generator, it follows that there exists at least one s for which the length $f(\langle x, G_t(s)\rangle)$ is bigger than n.

Clearly, the above reduction can be computed in time quasipolynomial in n. The output of the reduction is bounded by the length of $f(\langle x, y\rangle)$, where $|y| = n^{1/\delta}$. Since f is polynomial-time computable, the reduction is polynomial-bounded. □

The remainder of this section is devoted to proving Lemma 4.3. For this we first prove two auxiliary results. The first result establishes a connection between worst-case and average-case complexities for languages in NEEE ∩ coNEEE. The second result extends a well known equivalence between languages and their tally-encodings to the average-case world.

We know several results that establish worst-case to average-case connections for classes such as EXP and PSPACE [9, 25, 26, 36, 38]. The following lemma establishes a similar connection for triple exponential time classes. This lemma can be proved using classical hardness amplification results [9, 20, 25, 38] or using list decoding techniques of error correcting codes. (See [1] for a good review.) The technique is now fairly standard in the area of hardness amplification and the situation is only made easier since we are concerned with deterministic and nondeterministic computation in time $O(2^{2^{2^{O(n)}}})$. For the sake of completeness, we include a proof in the Appendix.

Lemma 4.5 *If* NEEE ∩ coNEEE $\not\subseteq$ EEE/log, *then there is language L in* NEEE ∩ coNEEE *such that no* EEE/log *algorithm can decide L, at infinitely many lengths n, on more than $\frac{1}{2} + \frac{1}{n}$ fraction of strings from* $\{0, 1\}^n$.

It is well known that any set A over Σ^* can be encoded as a tally set T_A such that A is worst-case hard for exponential-time if and only if T_A is worst-case hard for polynomial-time. For our purposes, we need an average-case version of the this equivalence. Below we describe particular encoding of languages using tally sets that is helpful for us and prove the average-case equivalence.

Let $t_0 = 2$, $t_{i+1} = t_i^2$ for all $i \in \mathbb{N}$. Observe that $t_i = 2^{2^i}$. Let $\mathcal{T} = \{0^{t_i} \mid i \in \mathbb{N}\}$. For each $l \in \mathbb{N}$, let $\mathcal{T}_l = \{0^{t_i} \mid 2^l - 1 \le i \le 2^{l+1} - 2\}$. Observe that $\mathcal{T} = \bigcup_{l=0}^{\infty} \mathcal{T}_l$. Given a set $A \subseteq \{0, 1\}^*$, let $T_A = \{0^{2^{2^{r_x}}} \mid x \in A\}$, where r_x is the rank index of x in the standard enumeration of $\{0, 1\}^*$. It is easy to verify that for all $l \in \mathbb{N}$ and every x,

$$x \in A \cap \{0, 1\}^l \iff 0^{t_{r_x}} \in T_A \cap \mathcal{T}_l. \tag{4.1}$$

Lemma 4.6 *Let A and T_A be as above. Suppose there is a quasipolynomial time algorithm $\mathcal{A}$ such that for every l, on an ϵ fraction of strings from $\mathcal{T}_l$, this algorithm correctly decides the membership in T_A, and on the rest of the strings the algorithm* *outputs "I do not know". There is a $2^{2^{2^{k(l+1)}}}$-time algorithm $\mathcal{A}'$ for some constant k that takes one bit of advice and correctly decides the membership in A on $\frac{1}{2}+\frac{\epsilon}{2}$ fraction of the strings at every length l.*

Proof Consider the following algorithm $\mathcal{A}''$ for A: For each x, run the algorithm $\mathcal{A}$ on $0^{t_{r_x}}$ and output the same answer.

If the length of x is l, then $|0^{t_{r_x}}| \le 2^{2^{2^{l+1}-2}}$. Thus the time algorithm $\mathcal{A}''$ takes is at most $2^{2^{2^{k(l+1)}}}$ for some constant k. Algorithm $\mathcal{A}$ decides T_A correctly on ϵ fraction of strings from $\mathcal{T}_l$. Thus by (4.1), the algorithm $\mathcal{A}''$ correctly decides A on ϵ fraction strings from $\{0,1\}^l$, and say "I do not know" on the rest.

For algorithm $\mathcal{A}'$, Let S_l be the set of strings of length l on which the algorithm $\mathcal{A}''$ says "I do not know", if $|S_l \cap A| > \frac{1}{2}|S_l|$, then set the advice be 1, otherwise let the advice be 0. The algorithm $\mathcal{A}'$ simulates the algorithm $\mathcal{A}''$ and gives the advice bit as the answer when it says "I do not know". The fraction of strings of length l on which $\mathcal{A}'$ is correct is at least $\frac{1-\epsilon}{2}+\epsilon = \frac{1}{2}+\frac{\epsilon}{2}$. □

Now we are ready to prove Lemma 4.3.

Proof of Lemma 4.3 By Lemma 4.5, there is a language $L \in (\mathrm{NEEE} \cap \mathrm{coNEEE}) - \mathrm{EEE}/\log$ such that no $\mathrm{EEE}/\log$ algorithm can decide L correctly on more than a $\frac{1}{2}+\frac{1}{n}$ fraction of the inputs for infinitely many lengths n.

Without loss of generality, we can assume that $L \in \mathrm{NTIME}(2^{2^{2^n}}) \cap \mathrm{coNTIME}(2^{2^{2^n}})$ Let

$$T_L = \left\{0^{2^{2^{r_x}}} \,\middle|\, x \in L\right\}.$$

Clearly, $T_L \in \mathrm{NP} \cap \mathrm{coNP}$.

Recall that $t_0 = 2$ and $t_i = 2^{2^i}$. Define $\tau : \mathbb{N} \to \mathbb{N}$ such that $\tau(n) = \max\{i \mid t_i \le n\}$. Observe that $\tau(n)$ is at most $\log\log n$. Thus for every n, $t_{\tau(n)} \le n$. Now we will define our Turing complete language. Let

$$\mathrm{SAT}_0 = \{0x \mid 0^{t_{\tau(|x|)}} \notin T_L \text{ and } x \in \mathrm{SAT}\},$$
$$\mathrm{SAT}_1 = \{1x \mid 0^{t_{\tau(|x|)}} \in T_L \text{ and } x \in \mathrm{SAT}\}.$$

Let $A = \mathrm{SAT}_0 \cup \mathrm{SAT}_1$. Since L is in $\mathrm{NP} \cap \text{co-NP}$ and $t_{\tau(|x|)}$ is at most $|x|$, A is in NP. The following is a Turing reduction from SAT to A: Given a formula x, ask queries $0x$ and $1x$, and accept if and only if exactly one of them is in A. Thus A is polynomial-time 2-tt complete for NP. □

Suppose A is complete via length-increasing, polynomially-bounded, quasipolynomial-time reductions. Then there is such a reduction f from $\{0\}^*$ to A. Since f is polynomially-bounded, there is a constant d such that for every x, the length of $f(x)$ is less that $|x|^d$.

The following observation is easy to prove.

Observation 4.7 *Let $y \in \{0,1\}^*$ and $b \in \{0,1\}$ be such that $f(0^{t_i}) = by$. Then $0^{t_{\tau(|y|)}} \in T_L$ if and only if $b = 1$.*

Proof Fix a length l. We will describe a quasipolynomial-time algorithm that will decide the membership in T_L on at least $\frac{1}{\log d}$ fraction of strings from $\mathcal{T}_l$, and says "I do not know" on other strings. By Lemma 4.6, this implies that there is EEE/1 algorithm that decides L on more than $\frac{1}{2} + \frac{1}{2\log d}$ fraction of strings from $\{0,1\}^l$. This contradicts the hardness of L and completes the proof. □

Let $s = 2^l - 1$ and $r = 2^{l+1} - 2$. Recall that $\mathcal{T}_l = \{0^{t_i} \mid s \le i \le r\}$. Partition $\mathcal{T}_l$ in sets $T_0, T_2, \ldots, T_m$ where

$$T_k = \{0^{t_i} \mid s + k\log d \le i < s + (k+1)\log d \text{ and } i \le r\}.$$

This gives at least $\frac{2^l}{\log d}$ sets. We make the following observation.

Observation 4.8 *Fix k. If $f(0^{t_{s+k\log d}}) = by$, then $0^{t_{\tau(|y|)}} \in T_k$.*

Proof Recall that the length of $f(0^{t_{s+k\log d}})$ is less than $t^d_{s+k\log d}$. Since $t_i = 2^{2^i}$, $t^d_{s+k\log d} \le t_{s+(k+1)\log d}$. Since f is length-increasing, we have the following bound on the length of y.

$$t_{s+k\log d} \le |y| < t_{s+(k+1)\log d}.$$

By definition of τ, we have

$$s + k\log d \le \tau(|y|) < s + (k+1)\log d.$$

By the definition of T_k, we have $0^{t_{\tau(|y|)}} \in T_k$. □

This suggests the following algorithm for T_L: Let 0^{t_j} be its input. Say it lies in T_k. Compute $f(0^{t_{s+k\log d}}) = by$. If $t_{\tau(|y|)} \ne t_j$, then output "I do not know". Otherwise, accept 0^{t_j} if and only if $b = 1$. By Observation 4.7 this algorithm never errs. Since f is computable in quasipolynomial time, this algorithm runs in quasipolynomial time.

By Observation 4.8, $0^{t_{\tau(|y|)}}$ belongs to T_k and on this input the algorithm does not output "I do not Know". Thus there is at least one string in T_k on which this algorithm does not output "I do not Know".

Thus for every k, $0 \le k \le m$, there is at least one string from T_k on which the above algorithm correctly decides T_L. Thus the above algorithm correctly decides T_L on at least $\frac{1}{\log d}$ fraction of strings from $\mathcal{T}_l$, and never errs.

Acknowledgements We thank anonymous reviewers for several comments and suggestions that led to a vastly improved presentation of the paper.

Appendix

Here we provide a proof of Lemma 4.5

Definition Let $x, y \in \{0,1\}^m$. $\Delta(x,y) = \frac{1}{m}|\{i \mid x[i] \neq y[i]\}|$. We say that $E : \{0,1\}^n \to \{0,1\}^m$ is an *error correcting code with distance* δ if for every $x \neq y \in \{0,1\}^n$, $\Delta(E(x), E(y)) \geq \delta$.

Theorem 5.1 (Johnson Bound [27]) *If* $E : \{0,1\}^n \to \{0,1\}^m$ *is an ECC with distance at least* $\frac{1}{2} - \epsilon$, *then for every* $x \in \{0,1\}^m$, *and* $\eta \geq \sqrt{\epsilon}$, *there exist at most* $l \leq 1/(2\eta^2)$ *strings* $y_1, \ldots, y_l \in E(\{0,1\}^n)$ *such that* $\Delta(x, y_i) \leq \frac{1}{2} - \eta$ *for every* $i \in [l]$.

Consider an ECC $E : \{0,1\}^{2n \log n} \to \{0,1\}^{n^4}$ with distance $1/2 - /2n$. An explicit example of such ECC is the code obtained by concatenating the Walsh-Hadamard code with the Reed-Solomon code. We will view strings of length $2n \log n$ and n^4 truth tables of languages at certain lengths. For this, both $2n \log n$ and n^4 must be powers of 2. If n is of the form $2^{2^{k-1}}$, then both $2n \log n$ and n^4 are powers of two.

For any language L'', let L' be *fold*(L''). It is clear that $L'' \in \text{NEEE} \cap \text{coNEEE}$ if and only if $L' \in \text{NEEE} \cap \text{coNEEE}$. The following fact is easy to verify.

Lemma 5.2 *If* $L'' \notin \text{EEE}/O(\log n)$, *then* $L' \notin \text{EEE}/O(\log n)$ *and for every* $\text{EEE}/O(\log n)$ *algorithm* $\mathcal{A}$, *there are infinitely many lengths* n *of the form* $2^{k+1} + k$ *such that* $\mathcal{A}$ *fails to decide* L' *correctly at length* n.

Lemma 4.5 *If there is a language in* $L'' \in \text{NEEE} \cap \text{coNEEE} - \text{EEE}/O(\log n)$, *then there is a language* $L \in \text{NEEE} \cap \text{coNEEE}$ *such that for every* $\text{EEE}/O(\log n)$ *algorithm* $\mathcal{A}$, *there exist infinitely many* n *and* $\mathcal{A}$ *correctly decides* L *on at most* $\frac{1}{2} + \frac{1}{n}$ *fraction of strings from* Σ^n.

Proof Let $L'' \in \text{NEEE} \cap \text{coNEEE} - \text{EEE}/O(\log)$. Let $L' = $ *fold*(L''). Let L be the language such that for every $k \in \mathbb{N}$, $L_{2^{k+1}} = E(L'_{2^{k-1}+k})$ where E is the WH-RS code. On other lengths, we set L to be empty. It is clear that $L \in \text{NEEE} \cap \text{coNEEE}$ since we can actually compute the truth table of L' in $\text{NEEE} \cap \text{coNEEE}$.

Now, for the sake of contradiction, assume that there is an $\text{EEE}/O(\log n)$ algorithm $\mathcal{A}$ that correctly decides L on more than $\frac{1}{2} + \frac{1}{n}$ fraction of inputs at every length n. We will first show that this yields a $\text{EEE}/O(\log n)$ algorithm for L. Since L is empty at lengths that are not of the form 2^{k+1}, we can decide L easily at those lengths. Thus we will concentrate on lengths of the form 2^{k+1}.

Consider n of the form 2^{k+1}. By the definition of L, we know that $L_n = E(L'_{2^{k-1}+k})$. Let $N = 2^n$. Recall that the distance of the code E is $\frac{1}{2} - \epsilon$, where $\epsilon = \frac{1}{2\sqrt[4]{N}}$. Let $\eta = \frac{1}{n} = \frac{1}{\log N} > \sqrt{\epsilon}$.

By running $\mathcal{A}$ on all strings of length n compute a candidate sequence, $x \in \{0,1\}^N$, for L_n. By our assumption, A is correct on at least $\frac{1}{2} + \frac{1}{n}$ of strings from Σ^n. Thus $\Delta(x, L_n) \leq \frac{1}{2} - \frac{1}{n}$. By the Johnson bound 5.1, there exist at most $l \leq \frac{n^2}{2}$ strings $y_1, \ldots, y_l$ in $E(\Sigma^{2^{k-1}+k})$ such that $\Delta(x, y_i) \leq \frac{1}{2} - \frac{1}{n}$. By running E on all strings from $\Sigma^{2^{k-1}+k}$, we can compute these y_is. Since WH-RS code is computable in polynomial time, this computation can be done in triple exponential time. Since $L_n = E(L'_{2^{k-1}+k})$, there exists a j, $1 \leq j \leq l$ such that $L_n = y_j$. Thus $\log l = O(\log n)$ bits of information is enough to identify j for which $L_n = y_j$. Thus $L \in \text{EEE}/O(\log n)$.

Recall that $L_n = E(L'_{2^{k-1}+k})$ when $n = 2^k$. Since $L \in \mathrm{EEE}/O(\log n)$, by inverting the WH-RS code E, we can decide L' correctly on all lengths of the form $2^{k+1} + k$, using an $\mathrm{EEE}/O(\log n)$ algorithm. This contradicts Lemma 5.2. □

References

1. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press, Cambridge (2009)
2. Amir, A., Beigel, R., Gasarch, W.: Some connections between bounded query classes and non-uniform complexity. Inf. Comput. **186**, 104–139 (2003)
3. Agrawal, M.: Pseudo-random generators and structure of complete degrees. In: Proceedings of the Seventeenth Annual IEEE Conference on Computational Complexity, pp. 139–147 (2002)
4. Ambos-Spies, K., Bentzien, L.: Separating NP-completeness notions under strong hypotheses. J. Comput. Syst. Sci. **61**(3), 335–361 (2000)
5. Agrawal, M., Watanabe, O.: One-way functions and the isomorphism conjecture. Technical report TR09-019, Electronic Colloquium on Computational Complexity, 2009
6. Beigel, R.: Query-limited reducibilities. PhD thesis, Stanford University, 1987
7. Berman, L.: Polynomial reducibilities and complete sets. PhD thesis, Cornell University, 1977
8. Beigel, R., Feigenbaum, J.: On being incoherent without being very hard. Comput. Complex. **2**(1), 1–17 (1992)
9. Babai, L., Fortnow, L., Nisan, N., Wigderson, A.: BPP has subexponential time simulations unless EXPTIME has publishable proofs. Comput. Complex. **3**, 307–318 (1993)
10. Buhrman, H., Fortnow, L., Torenvliet, L.: Six hypotheses in search of a theorem. In: IEEE Conference on Computational Complexity, pp. 2–12 (1997)
11. Berman, L., Hartmanis, J.: On isomorphism and density of NP and other complete sets. SIAM J. Comput. **6**, 305–322 (1977)
12. Buhrman, H., Hitchcock, J.M.: NP-hard sets are exponentially dense unless NP $\subseteq$ coNP/poly. In: Proceedings of the 23rd Annual IEEE Conference on Computational Complexity, pp. 1–7. IEEE Comput. Soc., Los Alamitos (2008)
13. Buhrman, H., Hescott, B., Homer, S., Torenvliet, L.: Non-uniform reductions. Theory Comput. Syst. **47**(2), 317–341 (2010)
14. Buhrman, H., Homer, S., Torenvliet, L.: Completeness for nondeterministic complexity classes. Math. Syst. Theory **24**, 179–200 (1991)
15. Cook, S.A.: The complexity of theorem proving procedures. In: Proceedings of the Third ACM Symposium on the Theory of Computing, pp. 151–158 (1971)
16. Feigenbaum, J., Fortnow, L., Laplante, S., Naik, A.: On coherence, random-self-reducibility, and self-correction. Comput. Complex. **7**, 174–191 (1998)
17. Feigenbaum, J., Fortnow, L., Lund, C., Spielman, D.: The power of adaptiveness and additional queries in random-self-reductions. Comput. Complex. **4**, 158–174 (1994)
18. Fortnow, L., Santhanam, R.: Infeasibility of instance compression and succinct PCPs for NP. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, pp. 133–142 (2008)
19. Ganesan, K., Homer, S.: Complete problems and strong polynomial reducibilities. SIAM J. Comput. **21**(4), 733–742 (1992)
20. Goldreich, O., Levin, L.A.: A hard-core predicate for all one-way functions. In: Proceedings of the Twenty-First ACM Symposium on the Theory of Computing, pp. 25–32 (1989)
21. Goldreich, O., Levin, L., Nisan, N.: On constructing 1-1 one-way function. Technical report TR95-029, ECCC, 1995
22. Hemaspaandra, E., Naik, A., Ogiwara, M., Selman, A.: P-selective sets and reducing search to decision vs. self-reducibility. J. Comput. Syst. Sci. **53**(2), 194–209 (1996)
23. Hitchcock, J.M., Pavan, A.: Comparing reductions to NP-complete sets. Inf. Comput. **205**(5), 694–706 (2007)
24. Hitchcock, J.M., Pavan, A., Vinodchandran, N.V.: Partial Bi-immunity, scaled dimension, and NP-completeness. Theory Comput. Syst. **42**(2), 131–142 (2008)
25. Impagliazzo, R.: Hard-core distributions for somewhat hard problems. In: Proceedings of the 36th Annual Conference on Foundations of Computer Science, pp. 538–545 (1995)

26. Impagliazzo, R., Wigderson, A.: P = BPP if E requires exponential circuits: derandomizing the XOR lemma. In: Proceedings of the 29th Symposium on Theory of Computing, pp. 220–229 (1997)
27. Johnson, S.: A new upper bound for error correcting codes. IRE Trans. Inf. Theory **8**(3), 203–207 (1962)
28. Levin, L.: Universal sorting problems. Probl. Inf. Transm. **9**, 265–266 (1973). English translation of original in Problemy Peredaci Informacii
29. Ladner, R., Lynch, N., Selman, A.: A comparison of polynomial time reducibilities. Theor. Comput. Sci. **1**, 103–123 (1975)
30. Lutz, J.H., Mayordomo, E.: Measure, stochasticity, and the density of hard languages. SIAM J. Comput. **23**(4), 762–779 (1994)
31. Lutz, J.H., Mayordomo, E.: Cook versus Karp-Levin: separating completeness notions if NP is not small. Theor. Comput. Sci. **164**(1–2), 141–163 (1996)
32. Nisan, N., Wigderson, A.: Hardness vs randomness. J. Comput. Syst. Sci. **49**, 149–167 (1994)
33. Pavan, A.: Comparison of reductions and completeness notions. SIGACT News **34**(2), 27–41 (2003)
34. Pavan, A., Selman, A.: Separation of NP-completeness notions. SIAM J. Comput. **31**(3), 906–918 (2002)
35. Pavan, A., Selman, A.: Bi-immunity separates strong NP-completeness notions. Inf. Comput. **188**, 116–126 (2004)
36. Sudan, M., Trevisan, L., Vadhan, S.: Pseudorandom generators without the XOR lemma. J. Comput. Syst. Sci. **62**(2), 236–266 (2001)
37. Watanabe, O.: A comparison of polynomial time completeness notions. Theor. Comput. Sci. **54**, 249–265 (1987)
38. Yao, A.: Theory and applications of trapdoor functions. In: Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, pp. 80–91 (1982)