Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 429-440
www.stacs-conf.org

COLLAPSING AND SEPARATING COMPLETENESS NOTIONS UNDER AVERAGE-CASE AND WORST-CASE HYPOTHESES

XIAOYANG GU [1] AND JOHN M. HITCHCOCK [2] AND A. PAVAN [3]

[1] LinkedIn Corporation

[2] Department of Computer Science, University of Wyoming

[3] Department of Computer Science, Iowa State University

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 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∩co-NP, then there is a Turing complete language for NP that is not many-one complete.

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.

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

Key words and phrases: computational complexity, NP-completeness.

Gu's research was supported in part by NSF grants 0652569 and 0728806.

Hitchcock's research was 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.

Pavan's research was supported in part by NSF grants 0830479 and 0916797.

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2462

n), then it can be used to construct pseudorandom generators. Using these pseudorandom generators, BPP problems can be solved in deterministic quasipolynomial time [23]. 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 [1] and Agrawal and Watanabe [2] 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 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 [9]. 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 [8] 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 [17] 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 [1] 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 [20] showed that NP-complete sets are complete under length-increasing P/poly reductions under the measure hypothesis on NP [26]. 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, P/-computable functions with $\log \log n$ bits of advice [10]. More recently, Agrawal and Watanabe [2] showed that if there exist regular one-way functions, 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 P/poly-computable, length-increasing 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 Section 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 notion 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 contains 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 [13] used Turing reductions to define completeness, though Levin [25] 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 [24] 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 [33, 12]. 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 [27] 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 [3, 30, 31, 21, 29].

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 an *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 the 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 NEEE$\cap$coNEEE that can not be solved in triple exponential time with logarithmic advice, i.e, NEEE $\cap$ coNEEE $\not\subseteq$ 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 NP $\cap$ co-NP that are sufficiently sparse. Similar hypotheses involving double and triple exponential-time classes have been used earlier in the literature [7, 15, 19, 14].

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, 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 complete via quasi polynomial-time, length-increasing 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. For languages A and B, we say that $A =$ io$B$, if $A_n = B_n$ for infinitely many n. For a complexity class $\mathcal{C}$, we say that $A \in$ io$\mathcal{C}$ if there is a language $B \in \mathcal{C}$ such that $A =$ 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 L is in NP/poly if there is a polynomial-size circuit C and a polynomial p such that for every x, x is in L if and only if there is a y of length $p(|x|)$ such that $C(x, y) = 1$.

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 2.1. 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, $f(x, y) \neq A(x)A(y)$.

Amir, Beigel, Gasarch [4] proved that every polynomial-time 2-approximable set is in P/poly. Their proof also implies the following extension for a superpolynomial function $t(n)$.

Theorem 2.2 ([4])**.** *If A is io-lengthwise $t(n)$-time 2-approximable, then for infinitely many n, $CC(A_n) \leq t^2(n)$.*

Given a language H' in co-NP, let H be $\{\langle x_1, \cdots, x_n\rangle \mid |x_1| = \cdots = |x_n| = n, x_i \in H'\}$. 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 ([16, 11])**.** *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 $L^{\leq n}$. Then H' is in* io$\mathrm{NP}/poly$.

The proof of Theorem 2.3 is similar to the proofs in [16, 11]. 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 works infinitely often, corresponding to when the NP/poly reduction works.

Definition 2.4. A function $f : \{0,1\}^n \to \{0,1\}^m$ is *s-secure* if for every $\delta < 1$, every $t \leq \delta s$, and every circuit $C : \{0,1\}^n \to \{0,1\}^m$ of size t, $\Pr[C(x) = f(x)] \leq 2^{-m} + \delta$. A function $f : \{0,1\}^* \to \{0,1\}^*$ is $s(n)$-*secure* if it is $s(n)$-secure at all but finitely many length n.

Definition 2.5. 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 [18].

Definition 2.6. 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,

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

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

Theorem 2.7 ([28, 23])**.** *If there is a language A in* E *such that $CC(A_n) \geq 2^{n^\epsilon}$ 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 $_{\text{io}}\text{DTIME}(2^{n^\epsilon})$.

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 $_{\text{io}}$NP/poly.

This means that every nondeterministic polynomial size circuit family that attempts to solve L is wrong on 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 Hypothesis 3 holds, then NP-complete sets are complete via length-increasing 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. To use Hypothesis 1 we have to fix encodings of boolean formulas with certain properties.

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

$$S = \left\{ \langle x, y, z\rangle \;\middle|\; |x| = |z|, |y| = |x|^\delta, \texttt{MAJ}[L(x), \mathrm{SAT}(y), L(z)] = 1 \right\}.$$

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 honest. Let

$$T_n = \left\{ \langle x, z\rangle \in \{0,1\}^n \times \{0,1\}^n \;\middle|\; L(x) \neq L(z),\ \forall y \in \{0,1\}^{n^\delta}\ |f(\langle x, y, z\rangle)| > n^\delta \right\}$$

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

Assuming that the above lemma holds, we complete the proof of the theorem. Given a length m, let $n = m^{1/\delta}$. 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 m, 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 n is polynomial in m, 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 m, the length of $f(\langle x_n, y, z_n\rangle)$ is bigger than $n^\delta = m$. 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, complete the proof of Theorem 3.1. ∎

Proof of Lemma 3.2. Suppose $T_n = \varnothing$ 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 $|x| = |z| = n$.
(2) Find a y of length n^δ such that $|f(\langle x, y, x\rangle)| \leq n^\delta$.
(3) If no such y is found, Output 10.
(4) 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 2 takes $O(2^{n^\delta})$ time. In Step 4, 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^δ. Thus the time taken to for this step is $(2^{n^\delta})^k \leq 2^{n^\epsilon/2}$ time. Thus the total time taken by the algorithm is bounded by $2^{n^\epsilon/2}$.

Consider a length n at which $T_n = \varnothing$. Let x and z be any strings at this length. Suppose for every y of length n^δ, the length of $f(\langle x, y, z\rangle)$ is at least n^δ. 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 2, then $L(x)L(z) \neq 10$.

Suppose the algorithm succeeds in finding a y in Step 2. 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 n. If there exist infinitely many lengths n, at which T_n is empty, then L is infinitely-often, length-wise, $2^{n^\epsilon}/2$-time approximable. By Theorem 2.2, L has circuits of size 2^{n^ϵ} at infinitely many lengths. ∎

Now we will describe how to modify the proof if we assume that Hypothesis 1 holds. Let L be the hard language guaranteed by the hypothesis. We will work with 3-SAT. Fix an encoding of 3CNF formulas such that formulas with same numbers of variables can be encoded as strings of same length. Moreover, we require that the formulas $\phi(x_1, \cdots, x_n)$ and $\phi(b_1, \cdots, b_i, x_{i+1}, \cdots, x_n)$ can be encoded as strings of same length, where $b_i \in \{0, 1\}$. Fix a reduction f from L to 3-SAT such that all strings of length n are mapped to formulas with n^r variables, $r \geq 1$. Let 3-SAT$' =$ 3-SAT $\cap \cup_r \Sigma^{n^r}$. It follows that that if there is an algorithm that decides 3-SAT$'$ such that for infinitely many n the algorithm runs in 2^{n^ϵ} time on all formulas with n^r variables, then L is in ioDTIME(2^{n^ϵ}).

Now the proof proceeds exactly same as before except that we use 3-SAT$'$ instead of L, i.e, our intermediate language will be

$$\{\langle x, y, z\rangle \mid \texttt{MAJ}[\text{3-SAT}'(x), \text{SAT}(y), \text{3-SAT}'(z)]\} = 1.$$

Consider the set T_n as before. It follows that if T_n is empty at infinitely many lengths, then for infinitely many n, 3-SAT$'$ is 2-approximable on formulas with n^r variables. Now we can use the disjunctive self-reducibility of 3-SAT$'$ to show that there is a an algorithm that solves 3-SAT$'$ and for infinitely many n, this algorithm runs in DTIME(2^{n^ϵ})-time on formulas with n^r variables. This contradicts the hardness of L. This gives the following theorem.

Theorem 3.3. *If there is a language in* NP *that is not in* ioDTIME(2^{n^ϵ})*, 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 3.4. *If there is a language L in* co-NP *that is not in* ioNP/*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 ioNP/poly. Let H be

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

Note that every n-tuple that may potentially belong to H can be encoded by a string of length n^2.

Let $S = 0H' \cup 1\overline{SAT}$. It is easy to show that S is in co-NP and S is not in ioNP/*poly*. Observe that S is 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 $\overline{A}$. 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 3.5. *For all but finitely many n, there exist two strings x_n and z_n of length n^2 with $H(x_n) \neq H(z_n)$ 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 $H(x) \neq H(z)$, there exist a y of length n such that $|f(x, y, z)| \leq n$.

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, $|f(x)| \leq n$. By Theorem 2.3, this implies that H' is in ioNP/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 $f(\langle x, y, z_n)$.

Suppose $x \in H$. Since $z_n \notin H$, there exists a string y of length n such that $y \in S$ and $|f(\langle x, y, z_n\rangle)| \leq n$. Consider a path that correctly guesses such a y. Since $z_n \notin H$, and $y \in S$, $\langle x, y, z_n\rangle \in L$. Thus $f(\langle x, y, z_n\rangle) \in A^{\leq n}$. Thus there exists at least one path on which the reduction outputs a string from $L \cap \Sigma^{\leq n}$. 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, $\langle x, y, z\rangle \notin L$. 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 L 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 ioNP/poly.

Thus for all but finitely many lengths n, there exist strings x_n and z_n of length n^2 with $H(x_n) \neq H(z_n)$ 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 3.6. *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 3.7. 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 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 $\text{io}\mathrm{P}/\mathrm{poly}$ which in turn implies that H' is in $\text{io}\mathrm{P}/\mathrm{poly}$. 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 $y_n \notin S$, $f(\langle x_n, y_n, z_n) \notin A$.

Thus we have a P/poly-computable, length-increasing 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 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 reduction. Therefore, the two hypotheses together separate the notions of many-one and Turing completeness for NP as stated in the following theorem.

Theorem 4.1. *If both of the above hypotheses are true, there is 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, polynomial-bounded, length-increasing reduction reduction f from B to A.*

A function f is *polynomial-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* $\mathrm{NEEE} \cap \mathrm{coNEEE} \not\subseteq \mathrm{EEE}/\log$*, then there is a polynomial-time Turing complete set for* NP *that is not many-one complete via quasipolynomial-time computable, polynomial-bounded, length-increasing reductions.*

The proof of Lemma 4.2 will appear in the full paper. The remainder of this section is devoted to proving Lemma 4.3. 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 if and only if T_A is worst-case hard. 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 $i \in \mathbb{N}$. Let $\mathcal{T} = \{0^{t_i} \mid i \in \mathbb{N}\}$. For each $l \in \mathbb{N}$, let $\mathcal{T}_l = \{0^{t_i} \mid 2^l - 1 \leq i \leq 2^{l+1} - 2\}$. Observe that $\mathcal{T} = \bigcup_{l=0}^{\infty} \mathcal{T}_l$. Given a set $A \subseteq \{0,1\}^*$, let $T_A = \left\{ 0^{2^{2^{r_x}}} \,\middle|\, x \in A \right\}$, 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 4.4. *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.*

We know several results that establish worst-case to average-case connections for classes such as EXP and PSPACE [34, 5, 22, 23, 32]. The following lemma establishes a similar connection for triple exponential time classes, and can be proved using known techniques.

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 $\{0,1\}^n$.*

Now we are ready to prove Lemma 4.3.

Proof of Lemma 4.3. By Lemma 4.5, there is a language $L \in (\text{NEEE} \cap \text{coNEEE}) - \text{EEE/log}$ such that no 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 $L \in \text{NTIME}(2^{2^{2^n}}) \cap \text{coNTIME}(2^{2^{2^n}})$ Let

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

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

Define $\tau : \mathbb{N} \to \mathbb{N}$ such that $\tau(n) = \max\{i \mid t_i \leq n\}$. Now we will define our Turing complete language. Let

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

Let $A = \text{SAT}_0 \cup \text{SAT}_1$. Since L is in NP ∩ co-NP, 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 at least one them is in A. Thus A is polynomial-time 2-tt complete for NP.

Suppose A is complete via length-increasing, polynomial-bounded, quasipolynomial-time reductions. Then there is such a reduction f from $\{0\}^*$ to A. There is a constant d such that f is n^d-bounded and runs in quasipolynomial time.

The following observation is easy to prove.

Observation 4.6. 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 $b = 1$.

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 the Lemma 4.4, 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 $\mathcal{T}_l = \left\{0^{t_i} \mid s \leq i \leq r\right\}$. Divide $\mathcal{T}_l$ in sets $T_0, T_2, \cdots T_r$ where $T_k = \left\{0^{t_i} \mid s + k \log d \leq r + (k+1) \log d\right\}$. This gives at least $\frac{2^l}{\log d}$ sets. Consider the following algorithm that decides T_L on strings from $\mathcal{T}_l$: Let 0^{t_j} be the input. Say, it lies in the set T_k. Compute $f(0^{t_{s+k \log d}}) = by$. If $t_{\tau(|y|)} \neq t_j$, then output "I do not know". Otherwise, accept 0^{t_j} if and only if $b = 1$. By Observation 4.6 this algorithm never errs. Since f is computable in quasipolynomial time, this algorithm runs in quasipolynomial time. Finally, observe that $t_{\tau(|y|)}$ lies between $t_{s+k \log d}$ and $t_{s+(k+1) \log d}$. Thus for every k, $0 \leq k \leq r$, there is at least one string from 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. ∎

References

[1] M. Agrawal. Pseudo-random generators and structure of complete degrees. In *Proceedings of the Seventeenth Annual IEEE Conference on Computational Complexity*, pages 139–147, 2002.

[2] M. Agrawal and O. Watanabe. One-way functions and the isomorphism conjecture. Technical Report TR09-019, Electronic Colloquium on Computational Complexity, 2009.

[3] K. Ambos-Spies and L. Bentzien. Separating NP-completeness notions under strong hypotheses. *Journal of Computer and System Sciences*, 61(3):335–361, 2000.

[4] A. Amir, R. Beigel, and W. Gasarch. Some connections between bounded query classes and non-uniform complexity. *Information and Computation*, 186:104–139, 2003.

[5] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

[6] R. Beigel. *Query-limited reducibilities*. PhD thesis, Stanford University, 1987.

[7] R. Beigel and J. Feigenbaum. On being incoherent without being very hard. *Computational Complexity*, 2(1):1–17, 1992.

[8] L. Berman. *Polynomial Reducibilities and Complete Sets*. PhD thesis, Cornell University, 1977.

[9] L. Berman and J. Hartmanis. On isomorphism and density of NP and other complete sets. *SIAM Journal on Computing*, 6:305–322, 1977.

[10] H. Buhrman, B. Hescott, S. Homer, and L. Torenvliet. Non-uniform reductions. *Theory of Computing Systems*. To appear.

[11] H. Buhrman and J. M. Hitchcock. NP-hard sets are exponentially dense unless NP $\subseteq$ coNP/poly. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity*, pages 1–7. IEEE Computer Society, 2008.

[12] H. Buhrman, S. Homer, and L. Torenvliet. Completeness for nondeterministic complexity classes. *Mathematical Systems Theory*, 24:179–200, 1991.

[13] S. A. Cook. The complexity of theorem proving procedures. In *Proceedings of the Third ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[14] J. Feigenbaum, L. Fortnow, S. Laplante, and A. Naik. On coherence, random-self-reducibility, and self-correction. *Computational Complexity*, 7:174–191, 1998.

[15] J. Feigenbaum, L. Fortnow, C. Lund, and D. Spielman. The powwer of adaptiveness and additional queries in random-self-reductions. *Computational Complexity*, 4:158–174, 1994.

[16] L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 133–142, 2008.

[17] K. Ganesan and S. Homer. Complete problems and strong polynomial reducibilities. *SIAM J. Comput.*, 21(4):733–742, 1992.

[18] O. Goldreich, L. Levin, and N. Nisan. On constructing 1-1 one-way function. Technical Report TR95-029, ECCC, 1995.
[19] E. Hemaspaandra, A. Naik, M. Ogiwara, and A. Selman. P-selective sets and reducing search to decision vs. self-reducibility. *Journal of Computer and System Sciences*, 53(2):194–209, 1996.
[20] J. M. Hitchcock and A. Pavan. Comparing reductions to NP-complete sets. *Information and Computation*, 205(5):694–706, 2007.
[21] J. M. Hitchcock, A. Pavan, and N. V. Vinodchandran. Partial Bi-immunity, Scaled Dimension, and NP-Completeness. *Theory of Computing Systems*. To appear.
[22] R. Impagliazzo. Hard-core distributions for somewhat hard problems. In *Proceedings of the 36th Annual Conference on Foundations of Computer Science*, pages 538–545, 1995.
[23] R. Impagliazzo and A. Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th Symposium on Theory of Computing*, pages 220–229, 1997.
[24] R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1:103–123, 1975.
[25] L. Levin. Universal sorting problems. *Problems of Information Transmission*, 9:265–266, 1973. English translation of original in *Problemy Peredaci Informacii*.
[26] J. H. Lutz and E. Mayordomo. Measure, stochasticity, and the density of hard languages. *SIAM Journal on Computing*, 23(4):762–779, 1994.
[27] J. H. Lutz and E. Mayordomo. Cook versus Karp-Levin: Separating completeness notions if NP is not small. *Theoretical Computer Science*, 164(1–2):141–163, 1996.
[28] N. Nisan and A. Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.
[29] A. Pavan. Comparison of reductions and completeness notions. *SIGACT News*, 34(2):27–41, June 2003.
[30] A. Pavan and A. Selman. Separation of NP-completeness notions. *SIAM Journal on Computing*, 31(3):906–918, 2002.
[31] A. Pavan and A. Selman. Bi-immunity separates strong NP-completeness notions. *Information and Computation*, 188:116–126, 2004.
[32] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *JCSS: Journal of Computer and System Sciences*, 62, 2001.
[33] O. Watanabe. A comparison of polynomial time completeness notions. *Theoretical Computer Science*, 54:249–265, 1987.
[34] A. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.