Length-Increasing Reductions for PSPACE-Completeness

John M. Hitchcock[1,⋆] and A. Pavan[2,⋆⋆]

[1] Department of Computer Science, University of Wyoming
jhitchco@cs.uwyo.edu
[2] Department of Computer Science, Iowa State University
pavan@cs.iastate.edu

Abstract. Polynomial-time many-one reductions provide the standard notion of completeness for complexity classes. However, as first explicated by Berman and Hartmanis in their work on the isomorphism conjecture, all natural complete problems are actually complete under reductions with stronger properties. We study the length-increasing property and show under various computational hardness assumptions that all PSPACE-complete problems are complete via length-increasing reductions that are computable with a small amount of nonuniform advice.

If there is a problem in PSPACE that requires exponential time, then polynomial size advice suffices to give li-reductions to all PSPACE-complete sets. Under the stronger assumption that linear space requires exponential-size NP-oracle circuits, we reduce the advice to logarithmic size. Our proofs make use of pseudorandom generators, hardness versus randomness tradeoffs, and worst-case to average-case hardness reductions.

Keywords: computational complexity, completeness, length-increasing reductions, PSPACE.

1 Introduction

Completeness is arguably the single most important notion in computational complexity theory. Many natural problems that arise in practice turn out be complete for appropriate complexity classes. Informally, a set A is complete for a class $\mathcal{C}$ if A belongs to $\mathcal{C}$ and every set from $\mathcal{C}$ "polynomial-time reduces" to A. In his seminal paper, Cook [Coo71] used *Turing reductions* to define completeness. However, Karp [Kar72] used a much more restrictive notion, *many-one reductions*, to define completeness. Since then polynomial-time many-one reductions have been considered as the most natural reductions to define completeness.

It has been observed that most problems remain complete under more stringent notions of reducibility. Perhaps the most restrictive notion of a polynomial-time reduction is that of *isomorphic reduction.* Two sets A and B are p-*isomorphic*

⋆ This research was supported in part by NSF grants 0652601 and 0917417.
⋆⋆ This research was supported in part by NSF grant 0916797.

if there exists a polynomial-time computable, one-one, onto, and polynomial-time invertible reduction from A to B. Berman and Hartmanis [BH77] observed that all known natural NP-complete sets are indeed p-isomorphic and this led to their famous “isomorphism conjecture”—all NP-complete sets are p-isomorphic to SAT. Berman and Hartmanis characterized isomorphism in terms of one-one, length-increasing reductions. They showed that two sets A and B are p-isomorphic if they are reducible to each other via one-one, polynomial-time invertible *length-increasing* reductions. We write “li-reduction” as an abbreviation for “length-increasing reduction.” Thus the isomorphism conjecture is equivalent to the following statement: All NP-complete sets are complete via one-one, polynomial-time invertible li-reductions.

Though the original isomorphism conjecture concerns the class NP, a similar conjecture can be formulated for classes such as E, NE, and PSPACE. In spite of many years of research we do not have concrete evidence for or against the isomorphism conjecture for any complexity class. This has led researchers to ask weaker questions such as: Do complete sets for a class remain complete under one-one reductions? Do complete sets for a class remain complete under li-reductions? Even these weaker questions are not completely resolved and we only know of some partial answers.

Berman [Ber77] showed that all complete sets for E are complete under one-one, li-reductions. Ganesan and Homer [GH92] showed that all NE-complete sets are complete under one-one reductions. For quite sometime, there had been no progress on NP and the first major result for NP is due to Agrawal [Agr02]. He showed that if one-way permutations exist, then all NP-complete sets are complete via one-one, P/poly-computable li-reductions. Since then there have been several results of this nature. Hitchcock and Pavan [HP07] showed that if NP does not have p-measure zero, then all NP-complete sets are complete via P/poly li-reductions. Buhrman, Hescott, Homer, and Torenvliet [BHHT10] improved this result to show that under the same hypothesis, NP-complete sets are complete via li-reductions that use a logarithmic amount of advice. Next, Agrawal and Watanabe [AW09] showed that if regular one-way functions exist, then NP-complete sets are complete via one-one, P/poly li-reductions. Most recently, Gu, Hitchcock, and Pavan [GHP12] showed that if NP contains a language that requires time $2^{n^{\Omega(1)}}$ at almost all lengths, then NP-complete sets are complete via P/poly li-reductions. All of the known results till date concern the complexity classes NP, E, and NE.

In this paper, we consider the question of whether PSPACE-complete sets are complete via li-reductions. It should be noted that the proofs of many of the aforementioned results go through if one replaces NP with PSPACE. For example, Agrawal's proof shows that if one-way permutations exist, then all complete sets for PSPACE are complete via, one-one, P/poly li-reductions. Similarly, Hitchcock and Pavan's proof shows that if PSPACE does not have p-measure zero, then PSPACE-complete sets are complete via P/poly li-reductions. However, Gu, Hitchcock, and Pavan's proof does not go through if one replaces NP with PSPACE.

In this paper we establish new results regarding PSPACE-complete sets. Using ideas from [GHP12], we first give evidence that PSPACE-complete sets are complete via non uniform, li-reductions. Our first main result is the following.

Theorem I. If PSPACE contains a language that requires $2^{n^{\epsilon}}$ time at almost all lengths, then PSPACE-complete sets are complete via li-reductions that use a polynomial amount of advice.

We note that the hypothesis used in this result is a *worst-case hardness* hypothesis (as opposed to average-case or almost-everywhere hardness hypotheses used in the works of [Agr02, HP07, BHHT10]). Next we address the question of whether we can eliminate or reduce the amount of nonuniformity used. We establish two sets of results. Our first result on this shows that nonuniformity in the reductions can be traded for nondetermisnism. We show that if NP contains a language that requires $2^{n^{\Omega(1)}}$ time at almost all lengths, then PSPACE-complete sets are complete via strong nondeterministic (SNP) li-reductions.

Next we show that using stronger hypotheses the amount of nonuniformity can be reduced. Our second main contribution is the following.

Theorem II. If there is a language in linear space that requires exponential size NP-oracle circuits, then PSPACE-complete sets are complete via li-reductions that use a logarithmic amount of advice.

The proof of this theorem is nonstandard. All known proofs that establish length-increasing completeness are of the following form: Say A is a complete language and we wish to prove that it is complete via li-reductions. All known proofs first define an intermediate language S and show that a standard complete language (such as SAT or K) length-increasing reduces to S, and there is a length-increasing reduction from S to A. We note that this approach may not work for our case (see the discussion after the statement of Theorem 3.4). Our proof proceeds by constructing two intermediate languages S_1 and S_2. We show both S_1 and S_2 length-increasing reduce to A. Our final length-increasing reduction from K to A goes via S_1 on some strings, and via S_2 on other strings. We use tools from pseudorandomness and hardness amplification to establish this result.

The following table compares some of the main results of this paper.

class	hardness assumption	li-reduction type
PSPACE	$2^{n^{\Omega(1)}}$ time	P/poly
NP	$2^{n^{\Omega(1)}}$ time	SNP
LINSPACE	$2^{\Omega(n)}$ NP-oracle circuits	P/ log
PSPACE E	$2^{n^{\Omega(1)}}$ circuits $2^{\Omega(n)}$ NP-oracle circuits	P/ log

The interpretation of a line in the table is that if the class (or pair of classes for the last line) satisfies the hardness assumption, then PSPACE-complete sets are complete under li-reductions of the stated type.

2 Preliminaries

Let $\mathcal{H}$ be a class of length bound functions mapping $\mathbb{N} \to \mathbb{N}$. A function $f : \Sigma^* \to \Sigma^*$ is P/$\mathcal{H}$-computable if there exist a polynomial-time computable $g : \Sigma^* \times \Sigma^* \to \Sigma^*$ and an $l(n) \in \mathcal{H}$ so that for every n, there is an advice string $a_n \in \Sigma^{\leq l(n)}$ such that for all $x \in \Sigma^n$, $f(x) = g(x, a_n)$. We will use the length bound classes poly $= \{l : l(n) = n^{O(1)}\}$ and log $= \{l : l(n) = O(\log n)\}$.

Given a language L, $L^{=n}$ denotes the set of strings of length n that belong to L. For a language L, we denote the characteristic function of L with L itself. That is, $L(x)$ is 1 if $x \in L$, otherwise $L(x)$ equals 0. Given two languages A and B, we say that A and B are *infinitely often equivalent*, $A =_{\text{io}} B$, if for infinitely many n, $A^{=n} = B^{=n}$. Given a complexity class $\mathcal{C}$, we define ${}_{\text{io}}\mathcal{C}$ as

$$ {}_{\text{io}}\mathcal{C} = \{A \mid \exists B \in \mathcal{C}, A =_{\text{io}} B\}. $$

In this paper we will use strong nondeterministic reductions [AM77]. A language A is SNP*-reducible* to a language B if there is a polynomial-time bounded nondeterministic machine M such that for every x the following holds:

- Every path of $M(x)$ outputs a string y or outputs a special symbol $\perp$. Different paths of $M(x)$ may output different strings.
- If a path outputs a string y, then $x \in A \Leftrightarrow y \in B$.

We say that an SNP reduction is length-increasing if the length of every output (excluding $\perp$) is greater than the length of the input.

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, and $CC^{\text{NP}}(f)$ is the smallest number s such that there is a size s, NP-oracle circuit that computes f. The Boolean function f is (s, ϵ)-hard if for every circuit of size at most s, $\Pr_{x \in \Sigma^n}[C(x) \neq f(x)] \geq \epsilon$.

Definition 2.1. *A* pseudorandom generator (PRG) family *is a collection of functions* $G = \{G_n : \Sigma^{m(n)} \to \Sigma^n\}$ *such that* G_n *is uniformly computable in time* $2^{O(m(n))}$ *and for every circuit of* C *of size* $O(n)$*,*

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

A pseudorandom generator is secure against NP-oracle circuits *if the above statement holds when the circuits have access to an* NP*-oracle.*

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

Theorem 2.2 ([KvM02]). *If there is a language* A *in* E *and an* $\epsilon > 0$ *such that* $CC^{\text{NP}}(A_n) \geq 2^{\epsilon n}$ *for all sufficiently large* n*, then there is a constant* k *and a PRG family* $G = \{G_n : \Sigma^{k \log n} \to \Sigma^n\}$ *that is secure against* NP*-oracle circuits.*

3 PSPACE-Complete Sets

We will first prove that if PSPACE contains a worst-case hard language, then PSPACE-complete sets are complete via P/poly li-reductions. We will use ideas from [GHP12]. As noted before, the proof of the analogous result in [GHP12] does not go through if we replace NP with PSPACE. This is because that proof uses the fact that NP contains complete languages that are disjunctively self-reducible. Since every disjunctively self-reducible language is in NP, we cannot hope that PSPACE has a disjunctively self-reducible complete set unless NP equals PSPACE. To get around this problem, we will use the fact that PSPACE is closed under complementation.

Theorem 3.1. *If there is language L in* PSPACE *that is not in* $_{\mathrm{io}}\mathrm{DTIME}(2^{n^{\epsilon}})$ *for some $\epsilon > 0$, then all* PSPACE*-complete sets are complete via* P/poly *li-reductions.*

Proof. Let A be a PSPACE-complete set that can be decided in time 2^{n^k}, and let K be the standard PSPACE-complete set that can be decided in time 2^{cn}, for some constants k and c. We define the following intermediate language S, where $\delta = \frac{\epsilon}{ck}$.

$$S = \left\{ \langle x, y \rangle \;\middle|\; |y| = |x|^{\delta}, L(x) \oplus K(y) = 1 \right\}$$

Since S is in PSPACE, there is a many-one reduction f from S to A.

Let

$$T_n = \left\{ x \in \Sigma^n \;\middle|\; x \in L,\ \forall y \in \Sigma^{n^{\delta}},\ |f(\langle x, y \rangle)| > n^{\delta} \right\}.$$

We will first show that T_n is not an empty set.

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

Proof. Suppose there exist infinitely many n for which $T_n = \emptyset$. We will exhibit an algorithm that decides L correctly in time $2^{n^{\epsilon}}$ for every n at which $T_n = \emptyset$. Consider the following algorithm for L.

1. Input $x, |x| = n$.
2. Cycle through all y of length n^{δ} and find a y such that $|f(\langle x, y \rangle)| \leq n^{\delta}$. If no such y is found reject x.
3. Suppose such a y is found. Compute $K(y)$ and $A(f(\langle x, y \rangle))$.
4. Accept x if and only if $A(f(\langle x, y \rangle)) \oplus K(y) = 1$.

Consider a length n at which $T_n = \emptyset$. Let x be an input of length n. We first consider the case when the above algorithm finds a y in Step 2. Since f is many-one reduction from S to A, we have $L(x) \oplus K(y) = A(f(\langle x, y \rangle))$. Thus $L(x) = K(y) \oplus A(f(\langle x, y \rangle))$. Therefore the algorithm is correct in this case. Now consider the case when the algorithm does not find a y. Since $T_n = \emptyset$, for every x (of length n) in L, it must be the case that the length of $f(\langle x, y \rangle)$ is at most n^{δ} for some y of length n^{δ}. Thus if the algorithm does not find a y in Step 2, then it must be the case that $x \notin L$. In this case, the algorithm correctly rejects x. Therefore the algorithm is correct on all strings of length n.

The time taken find a y in Step 2 is bounded by 2^{2n^δ}. The time taken to compute $K(y)$ is 2^{cn^δ}. The time taken to compute $A(f(\langle x, y\rangle))$ is at most 2^{m^k}, where $m = |f(\langle x, y\rangle)|$. Since the length of $f(\langle x, y\rangle)$ is at most n^δ and $\delta = \epsilon/ck$, the total time taken by the above algorithm is bounded by 2^{n^ϵ}.

Thus if $T_n = \emptyset$ for infinitely many n, then L is in $_{\text{io}}\text{DTIME}(2^{n^\epsilon})$ and this contradicts the hardness of L. □

We will now describe the P/poly many-one reduction from K to A. Let z_n be the lexicographically smallest string from T_n. It exists because of the previous lemma. On input y of length n^δ, the reduction outputs $f(\langle z_n, y\rangle)$. Since $z_n \in L$, $y \in K$ if and only if $\langle z_n, y\rangle$ is in S. Since f is a many-one reduction from S to A, this is a many-one reduction from K to A. By the definition of z_n, we have that the length of $f(\langle z_n, y\rangle)$ is bigger than n^δ. Since y is of length n^δ, this is a li-reduction from K to A. The advice for the reduction is z_n. Thus, this is a P/poly reduction. □

Theorem 3.3. *If there is language L in* NP *that is not in* $_{\text{io}}\text{DTIME}(2^{n^\epsilon})$ *for some $\epsilon > 0$, then all* PSPACE-*complete sets are complete via* SNP *li-reductions.*

The proof of Theorem 3.3 uses the same setup as Theorem 3.1. Consider S and T_n as before. We have that T_n is not empty for all but finitely many lengths. The reduction will use nondeterminism to find string in T_n. Let y be an input of length n^δ. Nondeterministically guess a string z of length n and verify that such that z is in L and $|f(\langle z, y\rangle)| > n^\delta$. If the verification is successful, then output $f(\langle z, y\rangle)$. Otherwise, output $\perp$. Since T_n is not empty, there exist at least one path that guesses a z from T_n and on this path the reduction is correct. Note that any path that fails to guess a $z \in T_n$ will output $\perp$. Thus there is no path on which the reduction outputs a wrong answer. Thus S is SNP-complete via li-reductions.

We now show how to reduce the number of advice bits from polynomial to logarithmic with a stronger hypothesis. We will show that if the worst-case NP-oracle circuit complexity of LINSPACE is $2^{\Omega(n)}$, then PSPACE-complete sets are complete via li-reductions that are P/ log-computable. We will first assume that PSPACE has a language that is hard on average, and then use a known worst-case to average-case connection for PSPACE.

Theorem 3.4. *Suppose there is a language L in* LINSPACE *such that for every n, L is $(2^{\epsilon n}, 3/8)$-hard for* NP-*oracle circuits, then* PSPACE-*complete sets are complete via* P/ log *li-reductions.*

Before we present the proof, we will mention the idea behind the proof. Our goal is to proceed as in the proof of Theorem 3.1. Consider T_n—in the proof of Theorem 3.1, we have shown that T_n is not empty. Suppose, we could show a stronger claim and establish that T_n contains many (say $> 3/4$ fraction) strings. Then a randomly chosen string will be a good advice with high probability. If LINSPACE is hard on average, then E is also hard on average and pseudorandom generators exist. Thus we can derandomize the process of "randomly picking a string from T_n" and instead generate a small list of strings such that at least one string from this list belongs to T_n. Now, a small advices suffices to identify the good string from T_n.

However, this idea does not work for two reasons. Note that every string in T_n must be in L. Since L is in PSPACE, this places T_n in PSPACE. We would like to derandomize the process of "randomly picking a string from a language in PSPACE." For this to work, we need a pseudorandom generator that is secure against PSPACE-oracle circuits. For this, we need a language that is hard for PSPACE-oracle circuits. However, the hard language L that is guaranteed by our hypothesis is in linear space, and thus cannot be hard for PSPACE. The second reason is that it is not clear that T_n will contain a 3/4 fraction of strings from Σ^n. Since L is hard on average for circuits, L contains roughly 1/2-fraction of strings from Σ^n (at every length n). Since T_n is a subset of L, we cannot hope that the size of T_n will be bigger than $\frac{3}{4}2^n$.

We overcome these difficulties by considering two intermediate sets S_1 and S_2 instead of one intermediate set S. Say f and g are many-one reductions from S_1 and S_2 to the complete set A. Then we define T_n as the set of all strings x from Σ^n such that for every y, the length of $f(\langle x, y\rangle)$ and $g(\langle x, y\rangle)$ are both large enough. This will place T_n in coNP and we can pseudorandomly pick strings from T_n provided we have a pseudorandom generator that is secure against NP-oracle circuits. Depending on the string that we picked, we will either use reduction f or reduction g. Now, we present the details.

Proof. Let A be a PSPACE-complete language that can be decided in time 2^{n^k}, and let K be the standard complete language for PSPACE. Observe that K can be decided in time 2^{cn} for some constant $c > 0$. Let $\delta = \epsilon/k$. Consider the following two languages

$$S_1 = \left\{\langle x, y\rangle \;\middle|\; K(x) \oplus L(y) = 1, |x| = \frac{\epsilon}{2c}|y|\right\},$$

$$S_2 = \left\{\langle x, y\rangle \;\middle|\; K(x) \oplus L(y) = 0, |x| = \frac{\epsilon}{2c}|y|\right\}.$$

Both languages are in PSPACE, thus there is a many-one reduction f from S_1 to A and many-one reduction g from S_2 to A. We first show that these reductions must be honest for most strings.

Claim 3.5. *Under our hardness assumption of L, there is a polynomial time algorithm $\mathcal{A}$ such that for all but finitely many m, $\mathcal{A}(1^m)$ outputs polynomially many strings $y_1, y_2, \cdots y_t$ of length $n = \frac{2cn}{\epsilon}$, such that for some y_i $1 \le i \le t$, and for every $x \in \{0,1\}^m$, the lengths of both $f(\langle x, y_i\rangle)$ and $g(\langle x, y_i\rangle)$ are at least n^δ.*

Assuming that Claim 3.5 holds, we complete the proof of the theorem. We will describe a honest reduction h from K to A. Given a length m, let $y_1, \cdots, y_t$ be the strings output by the algorithm $\mathcal{A}$. By the lemma, there is y_i such that for every x the lengths of both $f(\langle x, y_i\rangle)$ and $g(\langle x, y_i\rangle)$ are at least n^δ. The reduction gets i and $L(y_i)$ as advice. Note that the length is the advice is $O(\log m)$.

Let x be a string of length m. The reduction first computes the list $y_1, \cdots, y_t$. If the $L(y_i) = 0$, then $h(x) = f(\langle x, y_i \rangle)$, else $h(x) = g(\langle x, y_i \rangle)$.

Thus h is P$/O(\log n)$ computable. If $y_i \notin L$, then $x \in K$ if and only if $\langle x, y_i \rangle \in S_1$. Similarly if $y_i \in L$, then $x \in K$ if and only if $\langle x, y_i \rangle \in S_2$. Since f is a reduction from S_1 to A and g is a reduction S_2 to A, h is a valid reduction from K to A. Since the lengths of both $f(\langle x, y_i \rangle)$ and $g(\langle x, y_i \rangle)$ are at least n^δ, h is an honest reduction from K to A. Since K is paddable, there is a length-increasing P$/O(\log n)$-reduction from K to A. This, together with the forthcoming proofs of Claims 3.6 and 3.5, complete the proof of Theorem 3.4. □

To prove Claim 3.5, we need the following result.

Claim 3.6. *Let*

$$T_n = \left\{ y \in \{0,1\}^n \;\middle|\; \forall x \in \{0,1\}^{\frac{\epsilon n}{2c}}, |f(\langle x, y \rangle)| \geq n^\delta, \text{ and } |g(\langle x, y \rangle)| \geq n^\delta \right\}.$$

For all but finitely many n, $|T_n| \geq \frac{3}{4} 2^n$.

Proof. Suppose not. There exist infinitely many n for which T_n has at most $\frac{3}{4}2^n$ strings. We will show that this implies L must be not be average-case hard at infinitely many lengths. Consider the following algorithm for L.

1. Input $y, |y| = n$.
2. Cycle through all strings of length $\frac{\epsilon n}{2c}$ to find a string x such that at least one of $f(\langle x, y \rangle)$ or $g(\langle x, y \rangle)$ has length less than n^δ.
3. If no such string is found output $\perp$.
4. If $|f(\langle x, y \rangle)| \leq n^\delta$, then output $\overline{A(f(\langle x, y \rangle)) \oplus K(x)}$.
5. If $|g(\langle x, y \rangle)| \leq n^\delta$, then output $\overline{A(g(\langle x, y \rangle)) \oplus K(x)}$.

Consider a length n for which the cardinality of T_n is less than $\frac{3}{4}2^n$. We will first show that the above algorithm correctly solves L on $1/4$ fraction of strings from $\{0,1\}^n$.

A string y does not belong to T_n, if there is a string x of length $\frac{\epsilon n}{2c}$ such that at least one of the strings $f(\langle x, y \rangle)$ or $g(\langle x, y \rangle)$ has length less than n^δ. For all such string y the above algorithm halts in either Step 4 or in Step 5. It is clear that the decision made by the algorithm in these steps is correct. Thus if $y \notin T_n$, then the above algorithm correctly decide the membership of y in L. Since the size of T_n is at most $\frac{3}{4}2^n$ for many strings, the above algorithm correctly decides L on at least $1/4$ fraction of strings at length n.

The running time of the above algorithm can be bounded as follows. It takes $2^{\frac{n\epsilon}{2c}}$ time to search for x. Computing $K(x)$ takes at most $2^{\frac{n\epsilon}{2}}$ time. If $|f(\langle x, y \rangle)| < n^\delta$, computing $A(f(\langle x, y \rangle))$ takes at most 2^{n^ϵ} time. Thus Step 4 take at most $O(2^{\frac{\epsilon n}{2}})$ time. Similarly Step 5 also takes at most $O(2^{\frac{\epsilon n}{2}})$ time. Thus the running time of the above algorithm is bounded by $O(2^{\frac{\epsilon n}{2}})$.

Observe that the above algorithm never errs. On any string y it either outputs $\perp$ or correctly decides L, and for at least $1/4$ fraction of the strings the algorithm does not output $\perp$. By providing one bit of advice, we can make the algorithm to correctly decide L on at least $5/8$ fraction of inputs from Σ^n.

We can convert this modified algorithm into a family of circuits of size at most $2^{\epsilon n}$. If the size of T_n is less than $\frac{3}{4}2^n$ for infinitely many n, then this circuit family correctly computes L on at least 5/8 fraction of strings from $\{0,1\}^n$ for infinitely many n. This contradicts the hardness of L. □

Now we return to the proof of Claim 3.5.

Proof of Claim 3.5. In the following we fix m and so n. Recall that $n = \frac{2cm}{\epsilon}$. There is a polynomial p such that the computation of both f and g on strings of form $\langle x, y\rangle$, $|y| = n$, $|x| = m$ is bounded by $p(n)$. Let $r = p^2(n)$. Since LINSPACE is hard on average for $2^{\epsilon n}$-size NP-oracle circuits, and LINSPACE $\subseteq$ E, by Theorem 2.2 there is a PRG family G_r that maps $d \log r$ bits to r bits. The algorithm $\mathcal{A}$ on input 1^m behaves as follows: For each $d \log r$ bit string u compute $G_r(u)$ and output its n-bit prefix. This generates at most r^d strings. Since r is a polynomial in m, the number of strings output are polynomial in m.

We have to show that there exists a string y_i from the output of $\mathcal{A}(1^m)$ such that for every $x \in \{0,1\}^m$, the lengths of both $f(\langle x, y_i\rangle)$ and $g(\langle x, y_i\rangle)$ are bigger than n^δ. Suppose not. Consider the following algorithm $\mathcal{B}$ that has SAT as oracle.

Given a string of length r as input, let y be its n-bit prefix. By making queries to the SAT find if there is a string x of length m such that one of $f(\langle x, y\rangle)$ or $g(\langle x, y\rangle)$ have length less than n^δ. If no such x is found accept, else reject.

This algorithm runs in time $p(n)$, and so can be converted into a circuit C of size at most r. By our assumption,

$$\Pr_{z \in \{0,1\}^{d \log r}}[C(G_r(z)) = 1] = 0$$

However, by Claim 3.6

$$\Pr_{z \in \{0,1\}^r}[C(z) = 1] \geq 3/4$$

This contradicts the fact that G is a pseudorandom generator against NP-oracle circuits. □ ***Claim 3.5***

We can weaken the average-case hardness assumption in the above theorem to a worst-case hardness assumption. It is known that if LINSPACE requires $2^{\epsilon n}$-size NP-circuits at every length, there is a language in LINSPACE that is $(2^{\epsilon' n}, 3/8)$ hard for NP-oracle circuits at all lengths of the form t^2 [IW97, KvM02]. The proof of Theorem 3.4 requires average-case hardness of a language L at all lengths. However, the proof can be easily modified to work even when the language is average-case hard only at lengths of the form t^2. Thus we have the following theorem.

Theorem 3.7. *Suppose there is a language L in* LINSPACE *such that for every n, the worst-case* NP-*oracle circuit complexity of L is $2^{\epsilon n}$. Then all* PSPACE-*complete sets are complete via* P/log *li-reductions.*

We conclude with an improvement of Theorem 3.7. Consider the proof of Theorem 3.4. The hardness assumption "LINSPACE is $(2^{\epsilon n}, 3/8)$ hard for NP-oracle circuits" is used at two places. First in the proof of Claim 3.6. Note that the proof of this lemma only needs a weaker assumption, namely "PSPACE is $(2^{\epsilon n}, 3/8)$ hard for circuits." By a very slight modification of the proof, we can further weaken the hypothesis needed to establish Claim 3.6. Consider the following definitions of S_1 and S_2.

$$S_1 = \left\{ \langle x, y \rangle \;\middle|\; K(x) \oplus L(y) = 1, |x| = \frac{|y|^\epsilon}{2c} \right\},$$

$$S_2 = \left\{ \langle x, y \rangle \;\middle|\; K(x) \oplus L(y) = 0, |x| = \frac{|y|^\epsilon}{2c} \right\}.$$

We define T_n as

$$T_n = \left\{ y \in \{0,1\}^n \;\middle|\; \forall x \in \{0,1\}^{\frac{n^\epsilon}{2c}}, |f(\langle x, y \rangle)| \geq n^\delta, \text{ and } |g(\langle x, y \rangle)| \geq n^\delta \right\}.$$

With this definition of S_1, S_2, and T_n, the proof proceeds exactly as before, except that to establish Claim 3.6, we only need that "PSPACE is $(2^{n^\epsilon}, 3/8)$ hard for circuits". Again, it is known that if PSPACE is does not have 2^{n^ϵ}-size circuits, then PSPACE is $(2^{n^\epsilon}, 3/8)$-hard for circuits.

The second place where the hardness of LINSPACE is used is in the proof of Claim 3.5. Note that the proof of this claim goes through if we merely have the assumption "E has a language with $2^{\epsilon n}$-size NP-oracle circuit complexity". These observations yield the following improvement.

Theorem 3.8. *Suppose there is a language L in* PSPACE *such that for every n, the worst-case circuit complexity of L is 2^{n^ϵ} for some $\epsilon > 0$. Further assume that* E *has a language whose worst-case* NP*-oracle circuit complexity is $2^{\delta n}$ for some $\delta > 0$. All* PSPACE*-complete sets are complete via* P/log *li-reductions.*

References

[Agr02] Agrawal, M.: Pseudo-random generators and structure of complete degrees. In: Proceedings of the Seventeenth Annual IEEE Conference on Computational Complexity, pp. 139–147. IEEE Computer Society (2002)

[AM77] Adleman, L., Manders, K.: Reducibility, randomness, and intractability. In: Proceedings of the 9th ACM Symposium on Theory of Computing, pp. 151–163 (1977)

[AW09] Agrawal, M., Watanabe, O.: One-way functions and the isomorphism conjecture. Technical Report TR09-019, Electronic Colloquium on Computational Complexity (2009)

[Ber77] Berman, L.: Polynomial Reducibilities and Complete Sets. PhD thesis, Cornell University (1977)

[BH77] Berman, L., Hartmanis, J.: On isomorphism and density of NP and other complete sets. SIAM Journal on Computing 6(2), 305–322 (1977)

[BHHT10] Buhrman, H., Hescott, B., Homer, S., Torenvliet, L.: Non-uniform reductions. Theory of Computing Systems 47(2), 317–341 (2010)

[Coo71] 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)

[GH92] Ganesan, K., Homer, S.: Complete problems and strong polynomial reducibilities. SIAM Journal on Computing 21(4), 733–742 (1992)

[GHP12] Gu, X., Hitchcock, J.M., Pavan, A.: Collapsing and separating complete notions under worst-case and average-case hypotheses. Theory of Computing Systems 51(2), 248–265 (2012)

[HP07] Hitchcock, J.M., Pavan, A.: Comparing reductions to NP-complete sets. Information and Computation 205(5), 694–706 (2007)

[IW97] 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)

[Kar72] Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–104. Plenum Press (1972)

[KvM02] Klivans, A., van Melkebeek, D.: Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. SIAM Journal on Computing 31(5), 1501–1526 (2002)