Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 370 (2007) 293–298
Theoretical Computer Science
www.elsevier.com/locate/tcs

Note

Improved hardness amplification in NP

Chi-Jen Lu[a,*], Shi-Chun Tsai[b], Hsin-Lung Wu[b]

[a] *Institute of Information Science, Academia Sinica, Taipei, Taiwan*
[b] *Department of Computer Science, National Chiao Tung University, Hsinchu 30050, Taiwan*

Received 17 March 2005; received in revised form 26 September 2006; accepted 9 October 2006

Communicated by A. Razborov

Abstract

We study the problem of hardness amplification in NP. We prove that if there is a balanced function in NP such that any circuit of size $s(n) = 2^{\Omega(n)}$ fails to compute it on a $1/\text{poly}(n)$ fraction of inputs, then there is a function in NP such that any circuit of size $s'(n)$ fails to compute it on a $1/2 - 1/s'(n)$ fraction of inputs, with $s'(n) = 2^{\Omega(n^{2/3})}$. This improves the result of Healy et al. (STOC'04), which only achieves $s'(n) = 2^{\Omega(n^{1/2})}$ for the case with $s(n) = 2^{\Omega(n)}$.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Computational complexity; Hardness amplification; Pseudorandom generator; NP

1. Introduction

We study the problem of transforming a hard function into a harder function. We say that a Boolean function f is ε-*hard* for circuits of size s if any such circuit fails to compute f on at least an ε fraction of the input. A function on n bits is called average-case hard, mildly hard, and worst-case hard, respectively, when ε is $1/2 - 2^{-\Omega(n)}$, $1/\text{poly}(n)$, and $1/2^n$. A central question in complexity theory is to understand the relationship among such hardness conditions in complexity classes, which has played an important part in the research on derandomization. Given a complexity class, can we transform any function in it which is worst-case hard into one in it which is average-case hard? After a long series of work, this has been shown to be possible for high complexity classes, such as $\mathsf{DTIME}(2^{O(n)})$ [6,11]. However, it remains open for lower complexity classes. In fact, there are results showing that the same techniques used for high complexity classes can not be used for the class NP to obtain average-case hardness when starting from worst-case hardness [1,12,13] or even starting slightly below mild hardness [12,8,13].[1]

In this paper, we focus on the task of transforming mild hardness to average-case hardness for the complexity class NP. One attempt is to use Yao's XOR lemma [14,3], which transforms a given function $f : \{0,1\}^n \to \{0,1\}$ into a function $f' : (\{0,1\}^n)^k \to \{0,1\}$ defined by $f'(x_1, \ldots, x_k) = f(x_1) \oplus \cdots \oplus f(x_k)$. However, we do not know if

* Corresponding author.
E-mail addresses: cjlu@iis.sinica.edu.tw (C.-J. Lu), sctsai@csie.nctu.edu.tw (S.-C. Tsai), hsinlung@csie.nctu.edu.tw (H.-L. Wu).

[1] Although [12,13] only addressed explicitly the case of starting from worst-case hardness, it seems that the techniques used there can be extended to case of starting below mild hardness.

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.10.009

this works here, since we do not know if NP is closed under the XOR operation. O'Donnell [10] gave the first result along this line, showing how to convert any balanced function $f \in \mathsf{NP}$ which is mildly hard for polynomial-size circuits into another $f' \in \mathsf{NP}$ which is $(1/2 - 1/n^{1/2-\alpha})$-hard for polynomial-size circuits, for any constant $\alpha > 0$. He considered transformations of the form: $f'(x_1, \ldots, x_k) = C(f(x_1), \ldots, f(x_k))$, where C is a polynomial-time computable *monotone* function. Then he used the "tribes" function and the "recursive majority" function, and took their composition as the function C. Recently, Healy et al. [4] were able to amplify hardness beyond $1/2 - 1/\mathrm{poly}(n)$, showing how to convert any balanced function in NP which is mildly hard for circuits of size $s(n)$ into one in NP which is $(1/2 - 1/s'(n))$-hard for circuits of size $s'(n)$, with $s'(n) = s(n^{1/2})^{\Omega(1)}$. In particular, $s'(n) = n^{\omega(1)}$ when $s(n) = n^{\omega(1)}$, $s'(n) = 2^{n^{\Omega(1)}}$ when $s(n) = 2^{n^{\Omega(1)}}$, and $s'(n) = 2^{\Omega(n^{1/2})}$ when $s(n) = 2^{\Omega(n)}$. A key source of their improvement came from derandomizing O'Donnell's proof (the other source being the use of nondeterminism in computing the new function). They observed that the function C used by O'Donnell can be computed by a small-size read-once branching program and thus can be fooled by the pseudorandom generator of Nisan [9]. Unfortunately, this generator becomes the bottleneck of their approach when $s(n) = 2^{\Omega(n)}$, which prevents them from achieving the goal of having $s'(n) = 2^{\Omega(n)}$.

In this note, we make a further progress towards this goal, at the high end of the spectrum:

Theorem 1. *Suppose there is a balanced function in* NP *which is mildly hard for circuits of size* $s(n) = 2^{\Omega(n)}$. *Then there is a function in* NP *which is* $(1/2 - 1/s'(n))$*-hard for circuits of size* $s'(n)$, *with* $s'(n) = 2^{\Omega(n^{2/3})}$.

Our improvement comes from a closer look into the structure of the function C used by Healy et al., which enables us to construct a better pseudorandom generator to fool C. More precisely, we observe that the function C, which is the composition of the tribes function (a DNF) with the recursive majority function, can be seen as some kind of combinatorial rectangle, though the range in each dimension is large. This suggests that we fool each dimension by a separate copy of Nisan's generator and provide their seeds using the output of Lu's pseudorandom generator for rectangles [7]. Our generator then is the composition of these two generators.

2. Preliminaries

We will basically follow the notations and definitions of [4]. For $n \in \mathbb{N}$, let $[n]$ denote the set $\{1, \ldots, n\}$, and let U_n denote the uniform distribution over $\{0,1\}^n$. For $q \in \mathbb{N}$, we identify the set $\{0,1\}^q$ with $[2^q]$. For a set R, we also use R to denote its membership function. We say that a function $G : \{0,1\}^\ell \to \{0,1\}^m$ is *explicitly computable* if given $x \in \{0,1\}^\ell$ and $i \in [m]$, the ith bit of $G(x)$ can be computed in time $\mathrm{poly}(\ell, \log m)$. A function $f : \{0,1\}^n \to \{0,1\}$ is called *balanced* if $\Pr[f(U_n) = 1] = 1/2$. For a function $f : \{0,1\}^n \to \{0,1\}$, let $f^{\otimes k} : \{0,1\}^{kn} \to \{0,1\}^k$ be the function defined by $f^{\otimes k}(x_1, \ldots, x_k) = (f(x_1), \ldots, f(x_k))$, for $x_1, \ldots, x_k \in \{0,1\}^n$.

Next, we define what we mean by a hard function.

Definition 1. For $0 \le \delta \le 1/2$ and $s(n) \le 2^{O(n)}$, we say that a function $f : \{0,1\}^n \to \{0,1\}$ is δ-hard for size $s(n)$ if every circuit of size $s(n)$ fails to compute f on at least a δ fraction of inputs.

As shown by Impagliazzo [5], one can basically see a δ-hard function as a δ-random function defined below, as they cannot be distinguished by circuits of size slightly smaller than $s(n)$.

Definition 2. A probabilistic function $g : \{0,1\}^n \to \{0,1\}$ is called δ-random if it is balanced and there is a subset $H \subset \{0,1\}^n$ with $|H| = 2\delta 2^n$ such that $g(x)$ is an independent random bit for $x \in H$ and $g(x)$ is deterministic for $x \notin H$.

Note that any probabilistic function g can also be seen as a deterministic function with respect to a random string y, and we will use g_y to denote this deterministic function.

2.1. Hardness amplification

Given a hard function f, one would like to transform it into a harder function f'. One typical way is to apply a function $C : \{0,1\}^k \to \{0,1\}$ to the function $f^{\otimes k}$ to get the function $f' = C \circ f^{\otimes k} : \{0,1\}^{nk} \to \{0,1\}$, defined as $(C \circ f^{\otimes k})(x_1, \ldots, x_k) = C(f(x_1), \ldots, f(x_k))$. For hardness amplification within NP, to ensure that $C \circ f^{\otimes k} \in \mathsf{NP}$

whenever $f \in \mathsf{NP}$, O'Donnell [10] chose C to be a polynomial-time computable *monotone* function. In particular, he considered the functions TRIBES and RMAJ, defined as follows.

Definition 3. Define the function $\text{TRIBES}_t : \{0,1\}^t \to \{0,1\}$ as

$$\text{TRIBES}_t(x_1, \ldots, x_t) = (x_1 \wedge \cdots \wedge x_b) \vee (x_{b+1} \wedge \cdots \wedge x_{2b}) \vee \cdots \vee (x_{\lceil t/b \rceil b-b+1} \wedge \cdots \wedge x_t),$$

where b is the largest integer such that $(1-2^{-b})^{t/b} \geq 1/2$. Note that this makes $b = O(\log t)$. Also, let MAJ be the majority function, and define the function $\text{RMAJ}_r : \{0,1\}^{3^r} \to \{0,1\}$ recursively as:

$$\begin{aligned} \text{RMAJ}_1(x_1, x_2, x_3) &= \text{MAJ}(x_1, x_2, x_3), \\ \text{RMAJ}_r(x_1, \ldots, x_{3^r}) &= \text{RMAJ}_{r-1}(\text{MAJ}(x_1, x_2, x_3), \ldots, \text{MAJ}(x_{3^r-2}, x_{3^r-1}, x_{3^r})). \end{aligned}$$

Given any $\delta \geq 1/\text{poly}(n)$, to amplify from a δ-hard function $f : \{0,1\}^n \to \{0,1\}$, O'Donnell used the composition

$$\text{AMP}_k^\delta = \text{TRIBES}_t \circ \text{RMAJ}_r^{\otimes t}$$

as the function for C, with $k = t3^r$, $t \in \mathbb{N}$, and $r = O(\log(1/\delta))$. He showed that the resulting function $f' = C \circ f^{\otimes k} : \{0,1\}^{n'} \to \{0,1\}$ has hardness $1/2 - 1/k^c$ for some constant c. Note that the new function f' now has an input length $n' = kn$, so its hardness when expressed in terms of the input length n' is only $1/2 - 1/\text{poly}(n')$. That is, even if a super-polynomial k is used, the resulting hardness does not go beyond $1/2 - 1/\text{poly}(n')$.

To overcome this bottleneck, Healy et al. [4] showed that one can take a super-polynomial k while keeping the input size of f' small (polynomial in n) if the k inputs to $f^{\otimes k}$ are generated in some pseudorandom way, in the following sense.[2] To simplify our presentation, we state things in a slightly different way from [4].

Definition 4 (*[4]*). For a probabilistic function $h : \{0,1\}^n \to \{0,1\}$, define its expected collision probability as $\text{EXPCP}[h] = \text{E}_x[2 \cdot \Pr_{y,y'}[h_y(x) = h_{y'}(x)] - 1]$.

Definition 5. We say that a generator $G : \{0,1\}^\ell \to (\{0,1\}^n)^k$ ε-fools the δ-EXPCP of a function $C : \{0,1\}^k \to \{0,1\}$ if for any δ-random function $g : \{0,1\}^n \to \{0,1\}$,

$$\left| \text{EXPCP}\left[(C \circ g^{\otimes k})\right] - \text{EXPCP}\left[(C \circ g^{\otimes k}) \circ G\right] \right| \leq \varepsilon.$$

Given such a generator G for $C = \text{AMP}_k^\delta$, the amplified function f' is defined as $f' = C \circ f^{\otimes k} \circ G$. The seed length of the generator G now becomes the input length of the new function f'. The following lemma states that the task of hardness amplification can be reduced to that of constructing such a generator.

Lemma 1 (*[4]*). *Suppose for any $\delta \geq 1/\text{poly}(n)$ and any $k = t3^r \leq 2^{O(n)}$, with $t \in \mathbb{N}$ and $r = O(\log(1/\delta))$, there exists an explicitly computable generator $G : \{0,1\}^{\ell(n)} \to (\{0,1\}^n)^k$ which $2^{-\Omega(n)}$-fools the δ-EXPCP of the function AMP_k^δ. Then for any $\delta \geq 1/\text{poly}(n)$ and any $s(n) \geq 2^{\Omega(n)}$, if there exists a balanced function in NP which is δ-hard for size $s(n)$, one can convert it into a function in NP which is $(1/2 - 1/s'(n))$-hard for size $s'(n)$, for some $s'(n) \geq 2^{\Omega(\ell^{-1}(n))}$.*

Remark 1. Lemma 1 does not appear explicitly in [4] but can be derived from arguments therein. In fact, Healy et al. [4] used two kinds of generators: one for preserving indistinguishability and one for fooling the δ-EXPCP of the function AMP_k^δ. In the case with $s(n) \geq 2^{\Omega(n)}$, the bottleneck lies in that for AMP_k^δ. They observed that the function AMP_k^δ can be computed by a read-once branching program of small size, and they showed that a pseudorandom generator fooling such branching programs (see Definitions 6 and 7) can fool the δ-EXPCP of the function AMP_k^δ. Therefore, they reduced the task of hardness amplification to that of finding a pseudorandom generator to fool such branching programs. (See for example Theorem 6.2 in the journal version of [4].) However, using currently available generators for branching programs, they were only able to obtain a generator of seed length $\Omega(n^2)$ for fooling the δ-EXPCP of AMP_k^δ. Instead of fooling branching programs, we will show that it suffices to fool a simpler class of tests: combinatorial rectangles, for which a better generator can indeed be found.

[2] Another issue when using a super-polynomial k is that the function AMP_k^δ is no longer computable in time $\text{poly}(n)$. As shown in [4], this can be handled using non-determinism.

2.2. PRGs for branching programs and rectangles

Our generator will be based on pseudorandom generators that fool read-once branching programs and combinatorial rectangles, respectively.

Definition 6. A function $G : \{0,1\}^\ell \to \{0,1\}^t$ is called an ε-PRG for a class of functions from $\{0,1\}^t$ to $\{0,1\}$ if for any T in this class, $|\Pr[T(U_t)] - \Pr[T(G(U_\ell))]| \le \varepsilon$.

Definition 7. A *read-once branching program* of size s with block-length n is a finite state machine of s states, with each edge labelled by a subset of $\{0,1\}^n$. The computation proceeds as follows. The input is read sequentially in one pass, one block of n bits at a time. When the machine reads a block $\beta \in \{0,1\}^n$, it goes from the current state to the state reached by the edge labelled with β. Let $\mathsf{BP}(s,n)$ denote the class of functions computed by such read-once branching programs.

Definition 8. For $m, d \in \mathbb{N}$, let $\mathsf{R}(m,d)$ denote the collection of rectangles $R = R_1 \times \cdots \times R_d \subseteq [m]^d$, with $R_i \subseteq [m]$ for all $i \in [d]$.

To fool these two classes of functions, we will use the PRGs of Nisan [9] and Lu [7], respectively.

Lemma 2 ([9]). *For any $n \in \mathbb{N}$ and any $s \le 2^n$, there exists an explicitly computable $2^{-\Omega(n)}$-PRG $G_N : \{0,1\}^\ell \to \{0,1\}^{sn}$ for $\mathsf{BP}(s,n)$ with $\ell = O(n \log s)$.*

Lemma 3 ([7]). *For any $m, d \in \mathbb{N}$ and any $\varepsilon \in (0,1)$, there exists an explicitly computable ε-PRG $G_L : \{0,1\}^\ell \to [m]^d$ for $\mathsf{R}(m,d)$ with $\ell = O(\log m + \log d + \log^{3/2}(1/\varepsilon))$.*

3. Proof of Theorem 1

Consider any $\delta \ge 1/\mathrm{poly}(n)$ and any $k = t3^r \le 2^{O(n)}$, with $t \in \mathbb{N}$ and $r = O(\log(1/\delta))$. Given Lemma 1, our task is to construct a generator with a short seed which fools the δ-ExpCP of the function Amp_k^δ. Let Or_d denote the OR function on d bits and let And_b denote the AND function on b bits. Consider an arbitrary δ-random function $g : \{0,1\}^n \to \{0,1\}$, and let $A : \{0,1\}^{kn} \to \{0,1\}$ be the function

$$A = \mathrm{Amp}_k^\delta \circ g^{\otimes k} = \mathrm{Or}_d \circ \left(\mathrm{And}_b \circ \mathrm{Rmaj}_r^{\otimes b} \circ g^{\otimes b3^r}\right)^{\otimes d},$$

where $k = db3^r = 2^{O(n)}$, $b = \mathrm{poly}(n)$, $d = 2^{O(n)}$, and $r = O(\log(1/\delta)) = O(\log n)$. Note that A is a probabilistic function (because of g), and let A_y denote the function A taking the random string y. Observe that each $A_y^{-1}(0)$ can be seen as a rectangle in $\mathsf{R}(2^{b3^r n}, d)$. As we will see, to fool the δ-ExpCP of Amp_k^δ, it suffices to have a good PRG for rectangles in $\mathsf{R}(2^{b3^r n}, d)$. However, the range in each dimension of such rectangles is too large for us to apply Lemma 3 effectively. To resolve this, we use d copies of Nisan's PRGs to fool the d functions in the d dimensions respectively, with the d seeds coming from the output of Lu's PRG. Formally, we use the following two generators, with $\varepsilon = 2^{-\Omega(n)}$:

- Let $G_N : \{0,1\}^q \to \{0,1\}^{b3^r n}$ be Nisan's (ε/d)-PRG for $\mathsf{BP}(n^c, n)$, for some large enough constant c. From Lemma 2, one can have $q = O(n \log n)$.
- Let $G_L : \{0,1\}^\ell \to \{0,1\}^{dq}$ be Lu's ε-PRG for $\mathsf{R}(2^q, d)$. From Lemma 3, one can have $\ell = O(n \log n) + O(n^{3/2}) = O(n^{3/2})$.

Then define our generator $G : \{0,1\}^\ell \to \{0,1\}^{db3^r n}$ as

$$G(u) = \left(G_N^{\otimes d} \circ G_L\right)(u) = G_N^{\otimes d}(G_L(u)).$$

It is easy to see that G is explicitly computable since both G_N and G_L are. To show that G is a good generator, we shall bound the value $|\mathrm{ExpCP}[A] - \mathrm{ExpCP}[A \circ G]|$. Let $C_{y,y'}$ be the function defined as $C_{y,y'}(x) = 1$ if

$A_y(x) = A_{y'}(x)$ and $C_{y,y'}(x) = 0$ otherwise. Then

$$\begin{aligned}|\textsc{ExpCP}[A] - \textsc{ExpCP}[A \circ G]| &= 2\left|\mathop{\mathrm{E}}_{x}\left[\Pr_{y,y'}\left[C_{y,y'}(x)=1\right]\right] - \mathop{\mathrm{E}}_{u}\left[\Pr_{y,y'}\left[C_{y,y'}(G(u))=1\right]\right]\right| \\ &\le 2\mathop{\mathrm{E}}_{y,y'}\left[\left|\Pr_{x}\left[C_{y,y'}(x)=1\right] - \Pr_{u}\left[C_{y,y'}(G(u))=1\right]\right|\right].\end{aligned}$$

It remains to show that for any y and y', G fools the function $C_{y,y'}$. However, neither $C^{-1}_{y,y'}(0)$ nor $C^{-1}_{y,y'}(1)$ appears to be a rectangle, so some twist is needed.

In fact, $C^{-1}_{y,y'}(0)$ is the symmetric difference of the two rectangles $R^y = A_y^{-1}(0)$ and $R^{y'} = A_{y'}^{-1}(0)$ in $\mathsf{R}(2^{b3^r n}, d)$, denoted as $R^y \ominus R^{y'}$. That is,

$$C^{-1}_{y,y'}(0) = R^y \ominus R^{y'} = (R^y \setminus R^{y'}) \cup (R^{y'} \setminus R^y).$$

Then $\Pr_x[x \in R^y \ominus R^{y'}] = \Pr_x[x \in R^y] + \Pr_x[x \in R^{y'}] - 2\Pr_x[x \in R^y \cap R^{y'}]$ and similarly for $\Pr_u[G(u) \in R^y \ominus R^{y'}]$. Thus

$$\begin{aligned}&\left|\Pr_x\left[C_{y,y'}(x)=1\right] - \Pr_u\left[C_{y,y'}(G(u))=1\right]\right| \\ &= \left|\Pr_x\left[C_{y,y'}(x)=0\right] - \Pr_u\left[C_{y,y'}(G(u))=0\right]\right| \\ &= \left|\Pr_x\left[x \in R^y \ominus R^{y'}\right] - \Pr_u\left[G(u) \in R^y \ominus R^{y'}\right]\right| \\ &\le \left|\Pr_x\left[x \in R^y\right] - \Pr_u\left[G(u) \in R^y\right]\right| + \left|\Pr_x\left[x \in R^{y'}\right] - \Pr_u\left[G(u) \in R^{y'}\right]\right| \\ &\quad + 2\left|\Pr_x\left[x \in R^y \cap R^{y'}\right] - \Pr_u\left[G(u) \in R^y \cap R^{y'}\right]\right|.\end{aligned}$$

Note that R^y, $R^{y'}$, and $R^y \cap R^{y'}$ are all rectangles in $\mathsf{R}(2^{b3^r n}, d)$. Furthermore, they all satisfy the property that the membership function of the set in each dimension can be computed in $\mathsf{BP}(n^c, n)$ for some constant c, according to [4].[3] Therefore, it remains to show the following.

Lemma 4. *For any $R = R_1 \times \cdots \times R_d \in \mathsf{R}(2^{b3^r n}, d)$ such that $R_i \in \mathsf{BP}(n^c, n)$ for any $i \in [d]$, $|\Pr_x[x \in R] - \Pr_u[G(u) \in R]| \le 2\varepsilon$.*

Proof. Observe that $|\Pr_x[x \in R] - \Pr_u[G(u) \in R]|$ is at most

$$\left|\Pr_x[x \in R] - \Pr_v\left[G_N^{\otimes d}(v) \in R\right]\right| + \left|\Pr_v\left[G_N^{\otimes d}(v) \in R\right] - \Pr_u\left[G_N^{\otimes d}(G_L(u)) \in R\right]\right|, \tag{1}$$

where v is sampled from U_{dq} with $dq = O(d \cdot n \log n)$. It remains to bound the two terms above.

First, note that G_N is an (ε/d)-PRG for $\mathsf{BP}(n^c, n)$ and can fool each R_i. Using a standard hybrid argument (see e.g. [2]), one can show that $G_N^{\otimes d}$ is an $(d \cdot \varepsilon/d)$-PRG for such a rectangle R. Thus, the first term in (1) above is at most ε.

[3] Recall that $A = \textsc{Amp}_k^\delta \circ g^{\otimes k} = \textsc{Or}_d \circ (\textsc{And}_b \circ \textsc{Rmaj}_r^{\otimes b} \circ g^{\otimes b3^r})^{\otimes d}$. From [4], the function $\textsc{And}_b \circ \textsc{Rmaj}_r^{\otimes b}$ is in $\mathsf{BP}(\mathrm{poly}(n), 1)$, and the probabilistic function $\textsc{And}_b \circ \textsc{Rmaj}_r^{\otimes b} \circ g^{\otimes b3^r}$ can be computed by a probabilistic $\mathsf{BP}(\mathrm{poly}(n), n)$. Thus by fixing the random string of A to any string y, the set $R^y = A_y^{-1}(0)$ seen as a rectangle in $\mathsf{R}(2^{b3^r n}, d)$ has the property that each of its d dimensions has its membership function in $\mathsf{BP}(\mathrm{poly}(n), n)$. Next, we argue that for any y and y', each dimension of the rectangle $R^y \cap R^{y'}$ also has its membership function in $\mathsf{BP}(\mathrm{poly}(n), n)$. To see this, first observe that the ith dimension of $R^y \cap R^{y'}$ is exactly $R_i^y \cap R_i^{y'}$, where R_i^y and $R_i^{y'}$ are the ith dimension of R^y and $R^{y'}$, respectively. Suppose R_i^y and $R_i^{y'}$ are recognized by read-once branching programs B and B' with state spaces S and S', respectively. Then the set $R_i^y \cap R_i^{y'}$ is recognized by the read-once branching program B'' with state space $S \times S'$, which goes from state (s, s') to state (t, t') when reading an input block x *if and only if* B goes from s to t and B' goes from s' to t', respectively, when reading x. The initial state of B'' is the state (s_0, s_0') where s_0 and s_0' are the initial states of B and B', respectively, while the accepting states of B'' are exactly those states (t, t') where t and t' are the accepting states of B and B', respectively. Thus, if B and B' are both in $\mathsf{BP}(\mathrm{poly}(n), n)$, so is B''.

To bound the second term, consider the rectangle $R' = \{v : G_N^{\otimes d}(v) \in R\} \in \mathsf{R}(2^q, d)$ with $q = O(n \log n)$. That is, $R' = R'_1 \times \cdots \times R'_d$, with $R'_i = G_N^{-1}(R_i) \subseteq \{0,1\}^q$ for $i \in [d]$. Since G_L is an ε-PRG for $\mathsf{R}(2^q, d)$, we have $|\Pr_v[v \in R'] - \Pr_u[G_L(u) \in R']| \leq \varepsilon$. Thus, the second term in (1) above is also at most ε. Therefore we have the lemma. □

Combining the lemma and the discussion above, we have

$$|\textsc{ExpCP}[A] - \textsc{ExpCP}[A \circ G]| \leq 2(2\varepsilon + 2\varepsilon + 4\varepsilon) = 16\varepsilon = 2^{-\Omega(n)}.$$

That is, our generator G, which uses a seed of length $\ell(n) = O(n^{3/2})$, is able to $2^{-\Omega(n)}$-fool the δ-ExpCP of the function $\textsc{Amp}_k^\delta$. Then by Lemma 1, we have our main theorem.

Discussion

Our generator uses a seed of length $O(n^{3/2})$, and as a result, we can only amplify hardness to $1/2 - 1/s'(n)$ against size $s'(n)$ with $s'(n) = 2^{\Omega(n^{2/3})}$. The main bottleneck is the generator for rectangles. However, to achieve the goal of having $s'(n) = 2^{\Omega(n)}$ using our approach, we need to improve both the generator for branching programs and the generator for rectangles. Without improving Nisan's PRG, even if we could have an optimal ε-PRG for $\mathsf{R}(m, d)$, with seed length $\Theta(\log m + \log d + \log(1/\varepsilon))$, the resulting generator would still need a seed of length $\Theta(n \log n) + O(n) = \Omega(n \log n)$ (see the definition of the generator G and the calculation of its seed length on the previous page), and we would only be able to achieve $s'(n) = 2^{\Omega(n/\log n)}$.

References

[1] Andrej Bogdanov, Luca Trevisan, On worst-case to average-case reductions for NP problems, in: Proceedings of the 44th Annual Symposium on Foundations of Computer Science, 2003, pp. 11–14.

[2] Oded Goldreich, Foundations of Cryptography, Basic Tools, volume 1, Cambridge University Press, Cambridge, 2001.

[3] Oded Goldreich, Noam Nisan, Avi Wigderson, On Yao's XOR lemma, Technical Report TR95–050, Electronic Colloquium on Computational Complexity, 1995.

[4] Alexander Healy, Salil P. Vadhan, Emanuele Viola, Using nondeterminism to amplify hardness, in: Proceedings of the 36th ACM Symposium on Theory of Computing, 2004, pp. 192–201, SIAM Journal on Computing 35 (4) (2006) 903–931 (in press) (Final version).

[5] Russel Impagliazzo, Hard-core distributions for somewhat hard problems, in: Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, 1995, pp. 538–545.

[6] Russel Impagliazzo, Avi Wigderson, P = BPP if E requires exponential circuits: Derandomizing the XOR lemma, in: Proceedings of the 29th ACM Symposium on Theory of Computing, 1997, pp. 220–229.

[7] Chi-Jen Lu, Improved pseudorandom generators for combinatorial rectangles, Combinatorica 22 (3) (2002) 417–434.

[8] Chi-Jen Lu, Shi-Chun Tsai, Hsin-Lung Wu, On the complexity of hardness amplification, in: Proceedings of the 20th Annual IEEE Conference on Computational Complexity, 2005, pp. 170–182.

[9] Noam Nisan, Pseudorandom generators for space-bounded computation, Combinatorica 12 (4) (1992) 449–461.

[10] Ryan O'Donnell, Hardness amplification within NP, Journal of Computer and System Sciences 69 (1) (2004) 68–94.

[11] Madhu Sudan, Luca Trevisan, Salil Vadhan, Pseudorandom generators without the XOR lemma, Journal of Computer and System Sciences 62 (2) (2001) 236–266.

[12] Emanuele Viola, The complexity of constructing pseudorandom generators from hard functions, Computational Complexity 13 (3–4) (2004) 147–188.

[13] Emanuele Viola, On constructing parallel pseudorandom generators from one-way functions, in: Proceedings of the 20th Annual IEEE Conference on Computational Complexity, 2005, pp. 183–197.

[14] Andrew Chi-Chih Yao, Theory and applications of trapdoor functions, in: Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science, 1982, pp. 80–91.