Constructive Hardness Amplification via Uniform Direct Product

Preetum Nakkiran

Jul 07, 2016

This post was motivated by trying to understand the recent paper "Learning Algorithms from Natural Proofs", by Carmosino-Impagliazzo-Kabanets-Kolokolova [CIKK16]. They crucially use the fact that several results in hardness amplification can be made constructive. In this post, we will look at the Uniform Direct Product Theorem of Impagliazzo-Jaiswal-Kabanets-Wigderson [IJKW10]. We will state the original theorem and algorithm of [IJKW10], then we will present a simpler analysis for a (weaker) non-uniform version of their algorithm, which contains some of the main ideas.

For a given function $f : \{0,1\}^n \to \{0,1\}^\ell$, say a circuit C "ε-computes f" if C computes f correctly on at least ε-fraction of inputs. That is, $\Pr_x[C(x) = f(x)] \geq \varepsilon$. We are interested in the following kind of direct product theorem (informally): "If function f cannot be ε-computed by any small circuit C, then the direct-product $f^{\otimes k}(x_1, x_2, \dots x_k) := (f(x_1), f(x_2), \dots, f(x_k))$ cannot be computed better than roughly ε^k by any similarly small circuit." [1]

This is usually proved[2] in contrapositive, by showing: If there exists a circuit C' that ε^k-computes $f^{\otimes k}$, then there exists a similarly-sized circuit C that ε-computes f. The very interesting part is, this amplification can be made fully constructive, by a simple algorithm.

Theorem 1 ([IJKW10], and Theorem 4.1 [CIKK16])**.** *Let $k \in \mathbb{N}, \varepsilon > 0$. There is a (uniform) PPT algorithm $\mathcal{A}$ with the following guarantees:*

- **Input:** *A circuit C' that ε-computes $f^{\otimes k}$ for some function $f : \{0,1\}^n \to \{0,1\}^\ell$.*
- **Output:** *With probability $\Omega(\varepsilon)$, output a circuit C that $(1-\delta)$-computes f.*

for $\delta = O(\log(1/\varepsilon)/k)$. In particular, $(1-\delta) = \varepsilon^{O(1/k)}$. The circuit C is of size $|C'|\,\mathrm{poly}(n, k, \log(1/\delta), 1/\varepsilon)$.

Note that we can only hope to construct the good circuit with probability $\Omega(\varepsilon)$, since unique decoding is impossible: the circuit C' may ε-compute up to $(1/\varepsilon)$ different functions f (agreeing with a different function on each ε-fraction of its inputs).

1 Uniform Version

The algorithm for Theorem 1 is:

$\mathcal{A}(C')$:

Input: A circuit C' that ε-computes the direct-product $f^{\otimes k}$.

1. Pick k iid random inputs $x_i \in \{0,1\}^n$, let $\vec{b} = (x_1, \dots, x_k)$, and evaluate $C'(\vec{b})$.
2. Pick a random subset $A \subset \{x_1, \dots, x_k\}$ of size $k/2$. Record $v := C'(\vec{b})|_A$ as the answers of C' on the inputs in A.

[1] If this seems trivial, consider the $k = 2$ case. We want to show that if $\Pr_x[C(x) = f(x)] \leq \varepsilon$ for all small circuits C, then $\Pr_{x,y}[C'(x,y) = (f(x), f(y))] \lesssim \varepsilon^2$ for all similarly small circuits C'. This is clearly true if the circuit C' operates independently on its inputs, but not as clear otherwise (eg, the correctness of C'-s two outputs could be highly correlated). Indeed, proofs of the direct-product theorem take advantage of this correlation.

[2] See the last section for good references to prior proofs.

3. Output the circuit $C_{A,v}$ defined below (with the values v on the subset A hardcoded).

$C_{A,v}$ is defined as the randomized circuit:

$C_{A,v}(x)$:

On input $x \in \{0,1\}^n$, check if $x \in A$, in which case output $v|_x$ (the hardcoded value of x according to v). Otherwise, repeat the following $T = O(\log(1/\delta)/\varepsilon)$ times.

1. Sample $(k/2 - 1)$ additional iid random strings $\{y_j\}$, each $y_j \in \{0,1\}^n$, and let $\vec{b} := (x, A, \{y_j\})$ be the tuple of k strings.
2. Evaluate $C'(\pi(\vec{b}))$ for a random permutation π of the k inputs.
3. If the answers of C' restricted to A agree with the hardcoded values v, then output $C'(\pi(\vec{b}))|_x$, (the answer C' gave for x), and stop.

Output an error if no output is produced after T iterations.

Intuition: Suppose the values v returned when the Algorithm queries $C'(b)$ are actually correct. That is, $v|_x = f(x)$ for all $x \in A$. Then, the circuit $C_{A,v}$ evaluates C' on input $\vec{b} = (b_1, \dots b_k)$, and it knows the correct value of $f(b_i)$ is on half of these coordinates. So, $C_{A,v}(x)$ tries to estimate whether a random point $C'(\vec{b})$ is correct or not, based on if it agrees on the known subset of coordinates. The idea is that a value of $C'(\vec{b})$ that is wrong on many coordinates is unlikely to pass this test. (See [IJKW10] for the full proof).

Now, in the remainder of this note, we will develop and prove a simpler (weaker) version.

2 Symmetrizing

The direct-product as defined above has a permutation symmetry:

$$f^{\otimes k}(\pi(x_1, \dots x_k)) = \pi(f^{\otimes k}(x_1, \dots x_k))$$

for any permutation π.

The algorithm of Theorem 1 strongly takes advantage of this symmetry (indeed, the algorithm would not work as promised if we omitted the random permutations).[3] To simplify presentation, it helps to define the direct-product f^k as a function over k-**multisets** of inputs, instead of over k-tuples of inputs. Following [IJKW10], for the remainder of this note, we will work in the setting of k-multisets, and denote the k-multiset direct product as f^k. That is, f^k takes as input an (unordered) k-multiset $B = \{x_1, x_2, \dots, x_k\}$, and returns the k-tuple

$$f^k(\{x_1, x_2, \dots, x_k\}) := (f(x_1), f(x_2), \dots, f(x_k))$$

We consider the probability measure induced by the uniform measure over tuples. That is, "pick a random k-multiset of U" means to generate a multiset by picking k iid random elements from the universe U, and forming the (unordered) multiset containing them.[4]

[3] Consider a $C'(x_1, \dots x_k)$ that is correct if x_1 lies in some ε-density set, and random otherwise. Without the random permutations, $C_{A,v}(x)$ will always evaluate $C'(x, \dots)$, and produce no output for $(1-\varepsilon)$-fraction of inputs x.

[4] So for example, for $k = 3$ the multiset $\{a, a, a\}$ has lower probability of being drawn than $\{a, a, b\}$ for $a \neq b$.

The notion of ε-computing remains the same:[5] A circuit $C'(B)$ ε-computes f^k if

$$\Pr_{B \sim \text{random } k\text{-multiset}}[C'(B) = f^k(B)] \geq \varepsilon$$

Note that C' is allowed to give different answers for the same element in a multiset, e.g. if $C'(\{a, a, a\}) = (y_1, y_2, y_3)$, the y_is may all be distinct – we don't take advantage of this symmetry.

3 Oracle Version

Here we present and prove a simpler version of the algorithm, in the case when we also have access to an oracle for f. (This can be seen as a non-uniform version).

Theorem 2. *Let $k \in \mathbb{N}, \varepsilon > 0$, and $f : \{0,1\}^n \to \{0,1\}^\ell$. There is a PPT algorithm $\mathcal{A}^f$ with oracle access to f, with the following guarantees:*

- **Input:** *A circuit C' that ε-computes f^k.*
- **Output:** *With probability 0.99, output a circuit C that $(1-\delta)$-computes f.*

for $\delta = O(\log(k)/(\varepsilon k))$. The circuit C is of size $|C'| \operatorname{poly}(n, k, \log(1/\delta), 1/\varepsilon)$.

The idea is, in Step 2 of Algorithm $\mathcal{A}$, we can generate the correct values v for the inputs in set A, by querying the oracle. That is, we set $v := f(A)$ directly, instead of using our approximate circuit C'. In fact, if we have a perfect oracle for f we can simplify the algorithm even further.

The algorithm is:

$\mathcal{A}^f(C')$:

1. Pick $T = O(\log(k)/\varepsilon)$ random $(k-1)$-multisets $A_1, \ldots A_T$, each A_i containing $(k-1)$ random inputs from $\{0,1\}^n$.
2. Query the f-oracle, and record the values of $v_{A_i} := \{f(x) : x \in A_i\}$ for all sets A_i.
3. Output the circuit $C_{A,v}$ defined below (with the values v_{A_i} on the subsets A_i hardcoded).

$C_{A,v}$ is defined as the circuit:

$C_{A,v}(x)$:

For each $i = 1 \ldots T = O(\log(k)/\varepsilon)$:

1. Let $B_i := \{x\} \cup A_i$.
2. Evaluate $C'(B_i)$.
3. If the answers of $C'(B_i)$ restricted to A_i agree with the hardcoded values $v_{A_i} = f(A_i)$, then output $C'(B_i)|_x$, (the answer C' gave for x), and stop.

[5] For our purposes, having a randomized circuit that ε-computes $f^{\otimes k}$ is essentially equivalent to having a randomized circuit that ε-computes f^k. The proofs will extend to randomized circuits, where we say C ε-computes f if $\Pr_{C,x}[C(x) = f(x)] \geq \varepsilon$, taken over randomness of C as well as x.

Output an error if no output is produced after T iterations.

Proof of Theorem 2. **Parameters:** We will have $\delta = 10000\log(k)/(\varepsilon k)$ and $T = 100\log(k)/\varepsilon$. (Think of aiming for $\delta \approx 1/k$).

We will argue that

$$\Pr_{\mathcal{A},C,x}[C_{A,v}(x) \neq f(x)] \leq \delta/100 \tag{1}$$

Where the probability is over the randomness of algorithm $\mathcal{A}^f$ (random choice of sets A_i), and random input $x \in \{0,1\}^n$. Then, by Markov

$$\Pr_{\mathcal{A}}\left[\Pr_{C,x}[C_{A,v}(x) \neq f(x)] > \delta\right] \leq 1/100$$

so the algorithm $\mathcal{A}^f$ will produce a good circuit $C_{A,v}$ except with probability 1/100.

In the execution of circuit $C_{A,v}(x)$, let us say "iteration i fails" if Step 3 of the circuit at iteration i outputs a wrong answer. That is, iteration i fails if $C'(B_i)$ is correct on the $(k-1)$ values in $A_i = B_i \setminus \{x\}$, but wrong on x.

Consider the probability that iteration 1 fails. Notice that the distribution of (x, A_1, B_1) is equivalently generated as:

$$\begin{array}{ccc} \{(x, A_1, B_1)\} & \equiv & \{(x, A_1, B_1)\} \\ A_1 \sim \text{random } (k-1)\text{-multiset} & & B_1 \sim \text{random } k\text{-multiset} \\ x \in \{0,1\}^n & & x \in B_1 \\ B_1 := \{x\} \cup A_1 & & A_1 := B_1 \setminus \{x\} \end{array}$$

That is, we can think of first sampling a random k-multiset B_1, then sampling a random $x \in B_1$. Iteration 1 only returns an output when $C'(B_1)$ has at most 1 wrong answer (since it checks correctness on the $(k-1)$ values of A_1). Thus iteration 1 only fails if the random $x \in B_1$ falls on this 1 (of k) answers. So

$$\Pr_{x,A_1,B_1}[\text{ Iteration 1 fails }] \leq \frac{1}{k} \tag{2}$$

Now, we just union bound:

$$\begin{aligned} \Pr[\text{error}] = &\Pr_{\mathcal{A},C,x}[C_{A,v}(x) \neq f(x)] \\ &\leq \Pr[\text{no output produced after } T \text{ iterations, or some iteration fails}] \\ &\leq \Pr[\text{no output produced}] + T \cdot \Pr[\text{Iteration 1 fails}] \\ &\leq \Pr[\text{no output produced}] + \frac{T}{k} \end{aligned}$$

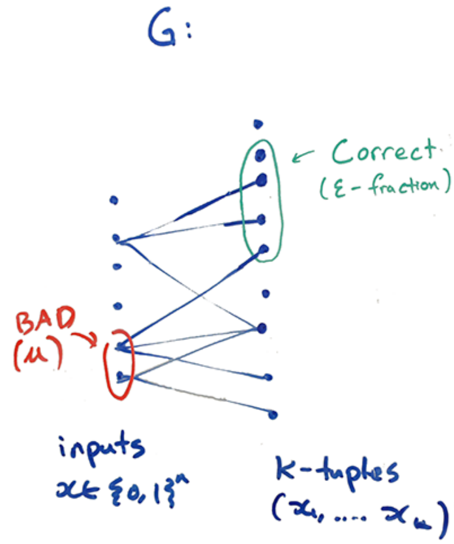
For our choice of T, δ, the second term is $\frac{T}{k} \leq \delta/200$. We will show the first term is $\leq \delta/200$ as well, completing the proof.

Produces output w.h.p.

It remains to show that the circuit $C_{A,v}$ produces an output with high probability. In Step 3 of the circuit $C_{A,v}$, notice that if C' is queried on a correct input B_i, it will pass the test and output a value.

The idea is: since C' is correct on ε-fraction of inputs, if we try $T = \Omega(\log(1/\delta)/\varepsilon)$ iid random inputs, we will be sure to hit a correct input, except with probability $O(\delta)$. This doesn't quite work, since the inputs B_i are not iid random (they all contain the input x) – but this dependence is minimal, so it still works out.

Following [IJKW10], it helps to think in term of this bipartite graph. Define G as a biregular bipartite graph between inputs $x \in \{0,1\}^n$, and k-**tuples**[6] $B \in (\{0,1\}^n)^k$, with an edge (x, B) if $x \in B$. We can think of the circuit $C_{A,v}(x)$ as picking up to T random neighbors of x in the graph G, until hitting an input B where $C'(B)$ is correct on all $B \setminus \{x\}$. We know that ε-fraction of k-tuples B are correct, and in fact we will show that almost all inputs x have close to ε-fraction of their neighbors as correct.



Lemma 3. *There are at most $O(\delta)$-fraction of "BAD" inputs $x \in \{0,1\}^n$ for which*

$$\Pr_{B \in N(x)}[C'(B) \text{ is correct}] \leq \varepsilon/10$$

This is sufficient to show that Pr[no output produced] $\leq O(\delta)$, since for inputs x that are not BAD, sampling $T = \Omega(\log(k)/\varepsilon)$ iid neighbors of x will hit a correct neighbor, except with probability $O(1/k) \leq O(\delta)$. [7]

It is easier to show the related property:

Lemma 4 (Mixing Lemma)**.** *Let $H \subseteq \{0,1\}^n$ be a set of inputs on the left of G, with the density of H at least μ. Then, except for some $2e^{-\Omega(\mu k)}$-fraction of tuples B, all tuples B on the right of G have*

$$\Pr_{x \in N(B)}[x \in H] = \mu \pm \mu/2$$

Proof of Lemma 4. Drawing a uniformly random tuple B on the right is exactly drawing k iid samples of inputs $B := (x_1, x_2, \dots, x_k)$. Then, by definition of G, picking a random neighbor $x \in N(B)$ is just picking a random $x \in B$. Thus, it is sufficient to show that if we draw k iid inputs $x_1, x_2, \dots, x_k$, the fraction of inputs that fall in H is within a multiplicative factor $(1 \pm 1/2)$ of its expectation μ (with high probability). This follows immediately from Chernoff bounds. ■

From this, the above Lemma 3 follows easily:

[6]Going back to tuples just to simplify the notation, so we can deal with the uniform measure.
[7] $(1 - \varepsilon/10)^T \leq e^{-T\varepsilon/10} \leq 1/k \leq \delta$.

Proof of Lemma 3. Let BAD be the set of "bad" inputs x, where $\Pr_{B \in N(x)}[C'(B) \text{ is correct}] \leq \varepsilon/10$. Suppose the density of BAD is μ. Let us count fraction of total edges in G that go between BAD, and the set of correct tuples (which we call GOOD). By the mixing lemma, there are at least $(\varepsilon - 2e^{\Omega(\mu k)})$ fraction of tuples B^* with $\Pr_{x \in N(B^*)}[x \text{ is bad}] \geq \mu/2$. So there are at least $(\varepsilon - 2e^{\Omega(\mu k)})(\mu/2)$ fraction of edges between the BAD and GOOD sets.

But, each bad input x has at most $\varepsilon/10$ fraction of edges into GOOD by definition, so the fraction of BAD $\leftrightarrow$ GOOD edges is at most $\mu(\varepsilon/10)$.

Thus we must have

$$(\varepsilon - 2e^{-\Omega(\mu k)})(\mu/2) \leq \mu(\varepsilon/10)$$
$$\implies \mu \leq O(\log(1/\varepsilon)/k)$$

This gives $\mu \leq \delta/200$ for our choice of δ. ■

This concludes the proof of correctness of the oracle version (Theorem 2). ■

4 Closing Remarks

- Note that in the oracle version, we were able to output a good circuit with probability 0.99, instead of w.p. $\Theta(\varepsilon)$ as in the fully uniform version. This makes sense because if we have an f-oracle, we can "check" if our circuit is actually computing the desired f, so we don't run into the unique decoding problem. (Indeed, we can construct an optimal version of algorithm $\mathcal{A}^f$ of Theorem 2 from the algorithm $\mathcal{A}$ of Theorem 1 in a black-box way, by checking if the output circuit of $\mathcal{A}$ mostly agrees with f on enough random inputs).
- There were several simplifications we made from $\mathcal{A}$ to $\mathcal{A}^f$.
  (1) We queried the oracle for the hardcoded values v, instead of the circuit.
  (2) We hardcoded $(k-1)$-multisets instead of $(k/2)$-multisets.
  (3) We hardcoded T iid multisets $\{A_i\}$, instead of just one multiset A.
  Note that we could not have done (2) without also doing (3) – otherwise there would not have been enough mixing (the circuit would fail with probability close to ε). Also, (3) would not have worked in the fully uniform case ($\mathcal{A}$, without the oracle) – because then all the hardcoded sets will be correct with only very small probability.
- The reason Theorem 2 has suboptimal parameters (eg, compare the setting of δ to Theorem 1) is because our analysis used the loose union bound, instead of using the fact that circuit $C_{A,v}$, by only outputting values that pass a test, is doing rejection-sampling on a certain conditional probability space. The tight analysis in [IJKW10] takes advantage of this fact.
- In the proof of Thereom 2, we used a property of the graph G that was essentially like an "Expander Mixing Lemma". We may hope that if we replace G with something sufficiently expander-like, we could get a derandomized direct-product theorem. Indeed, something like this is done in [IJKW10] ("Uniform direct product theorems: simplified, optimized, and *derandomized*").
- I think the oracle version is sufficient for the applications in [CIKK16], since there we have query access to the function f we are trying to learn/compress.
- For a good survey on direct-product for non-uniform hardness amplification, and the related "Yao's XOR Lemma", see [GNW11] (which includes at least 3 different proofs of the non-uniform XOR lemma). For a clean proof of Impagliazzo's Hardore Set theorem, which is used in some proofs of the XOR lemma, see for example Arora-Barak.

References

[CIKK16] Marco L Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 50. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016. URL: `http://drops.dagstuhl.de/opus/volltexte/2016/5855/pdf/34.pdf`.

[GNW11] Oded Goldreich, Noam Nisan, and Avi Wigderson. On yao's xor-lemma. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 273–301. Springer, 2011. URL: `http://www.wisdom.weizmann.ac.il/~oded/COL/yao.pdf`.

[IJKW10] Russell Impagliazzo, Ragesh Jaiswal, Valentine Kabanets, and Avi Wigderson. Uniform direct product theorems: simplified, optimized, and derandomized. *SIAM Journal on Computing*, 39(4):1637–1665, 2010. URL: `http://www.cs.columbia.edu/~rjaiswal/IJKW-Full.pdf`.