On Uniform Amplification of Hardness in NP

Luca Trevisan*

November 4, 2004

Abstract

We continue the study of amplification of average-case complexity within NP, and we focus on the *uniform* case.

We prove that if every problem in NP admits an efficient uniform algorithm that (averaged over random inputs and over the internal coin tosses of the algorithm) succeeds with probability at least $1/2 + 1/(\log n)^\alpha$, then for every problem in NP there is an efficient uniform algorithm that succeeds with probability at least $1 - 1/\mathrm{poly}(n)$. Above, $\alpha > 0$ is an absolute constant.

Previously, Trevisan (FOCS'03) presented a similar reduction between success $3/4 + 1/(\log n)^\alpha$ and $1 - 1/(\log n)^\alpha$. Stronger reductions, due to O'Donnell (STOC'02) and Healy, Vadhan and Viola (FOCS'04) are known in the non-uniform case.

1 Introduction

Amplification of hardness

Generally speaking, the goal of *amplification of hardness* is to start from a problem that is known (or assumed) to be hard on average in a weak sense (that is, every efficient algorithm has a noticeable probability of making a mistake on a random input) and to define a related new problem that is hard on average in the strongest possible sense (that is, no efficient algorithm can solve the problem noticeably better than by guessing a solution at random).

For decision problems, Yao's XOR Lemma [Yao82] is a very powerful result on amplification of hardnes. In the XOR Lemma, we start from a Boolean function $f : \{0,1\}^n \to \{0,1\}$ and define a new function $f^{\oplus k}(x_1, \ldots, x_k) := f(x_1) \oplus \cdots \oplus f(x_k)$, and the Lemma says that if every circuit of size $\leq S$ makes at least a δ fraction of errors in computing $f(x)$ for a random x, then every circuit of size $\leq S \cdot \mathrm{poly}(\delta\varepsilon/k)$ makes at least a $1/2 - \varepsilon$ fraction of errors in computing $f^{\oplus k}()$, where ε is roughly $\Omega((1-\delta)^k)$.

Various proofs of the XOR Lemma are known [Lev87, BL93, Imp95, GNW95, IW97], and none of them is completely *uniform*, that is, none of them describes a uniform reduction that transforms an algorithm that solves $f^{\oplus k}()$ on more than an $1/2+\varepsilon$ fraction of inputs into an algorithm that solves $f()$ on more than a $1-\delta$ fraction of inputs.[1] Due to a connection between amplification of hardness

*luca@cs.berkeley.edu. U.C. Berkeley, Computer Science Division.

[1] Some reductions, however, can be implemented uniformly provided that the distribution $(x, f(x))$ can be sampled by an efficient uniform sampler.

and coding theory discussed in [Imp02, TV02, Vio03, Tre03], no amplification result proved using uniform "black-box" reductions can start from a decision problem for which every efficient algorithm errs on at least a .2499 fraction of inputs and construct a new decision problem for which every efficient algorithm errs on at least a .2501 fraction of inputs. For specific problems, however, it is possible to prove uniform amplification results that are as strong as the XOR Lemma; for example, in [TV02] we show that this is possible for PSPACE-complete and EXP-complete problems, using the fact that such problems have *instance checkers*.[2]

Amplification of Hardness in NP

Suppose now that we want to study the average-case complexity of problems in NP, and that we would like to prove an amplification result of the following type: if L is a language in NP such that every efficient algorithm (or small family of circuits) errs on at least a $1/\mathrm{poly}(n)$ fraction of inputs of length n, then there is a language L' also in NP such that every efficient algorithm (or small circuit) errs on a $1/2 - 1/n^{\Omega(1)}$ fraction of inputs. The XOR Lemma does not help us prove a result of this kind, because if we define the language L' made of k-tuples $(x_1, \ldots, x_k)$ such that an odd number of x_i are in L, then we see that the language L' has the required average-case hardness but it is not clear that L' is in NP.[3]

In order to prove amplification of hardness results within NP, O'Donnell [O'D02] proves the following result: for every balanced Boolean function $f : \{0,1\}^n \to \{0,1\}$ (e.g. the characteristic function of an NP problem) and positive parameters ε, δ, there is an integer $k = \mathrm{poly}(1/\varepsilon, 1/\delta)$ and a *monotone* function $g : \{0,1\}^k \to \{0,1\}$ such that if every circuit of size S makes at least a δ fraction of errors in computing $f(x)$ given x, then every circuit of size $S \cdot \mathrm{poly}(\varepsilon, \delta)$ makes at least a $1/2 - \varepsilon$ fraction of errors in computing $f^{g,k} := g(f(x_1), \ldots, f(x_k))$ given $(x_1, \ldots, x_k)$. Note that if $f(\cdot)$ is the characteristic function of an NP language L on inputs of length n, then $f^{g,k}$ is the characteristic function of another NP language L' on inputs of length nk, and so this result indeed proves amplification of hardness in NP, albeit only for *balanced* decision problems, that is, for problems such that, for a random instance of a given length, there is a probability $1/2$ that the answer is YES and a probability $1/2$ that the answer is NO. For balanced problems, O'Donnell proves an amplification of hardness results from $1 - 1/\mathrm{poly}(n)$ to $1/2 + 1/n^{1/2-\varepsilon}$. He also introduces a padding argument to remove the restriction to balanced problems; for general problems the amplification goes from $1 - 1/\mathrm{poly}(n)$ to $1/2 + 1/n^{1/3-\varepsilon}$. O'Donnell's proof is based on Impagliazzo's [Imp95] result about "hard-core" distributions of inputs for problems that are weakly hard on average. Impagliazzo's results appear to use non-uniformity in an essential way.

O'Donnell's results were recently improved by Healy and others [HVV04]. For balanced problems in NP, [HVV04] prove amplification from $1 - 1/\mathrm{poly}(n)$ to $1/2 + 1/\mathrm{poly}(n)$. More generally, they are able to start from the assumption that every balanced problem in NP can be solved on a $1/2 + 1/s(n)$ fraction of inputs by circuits of size $s(n)$, and derive the conclusion that every balanced problem in NP can be solved on a $1 - 1/\mathrm{poly}(n)$ fraction of inputs by circuits of size roughly $s(n^2)$. These

[2] Such results, based on the non-uniform results of [STV01], are actually stronger than the XOR Lemma because relate *worst-case* to strong average-case complexity, instead of relating a weak form of average-case complexity to a strong form. The approach of [TV02] is unlikely to be useful within NP: on the one hand, it is considered unlikely that the kind of worst-case to average-case reductions presented in [STV01] can be generalized to NP-complete problems, and, on the other hand, it is considered unlikely that NP-complete problems have instance checkers.

[3] In fact, if L is NP-complete then L' cannot be in NP unless NP = *co*NP.

results of [HVV04] also use Impagliazzo's hard core distributions, and the reductions in [HVV04] also appear to be inherently non-uniform.

Previous Work on Uniform Amplification of Hardness in NP

A weak uniform version of O'Donnell's result appears in [Tre03].

In [Tre03], we first give an "advice efficient" presentation of Impagliazzo's proof, from which we derive an amplification from $1-\delta$ to $1/2+\varepsilon$ using only $\mathrm{poly}(1/\varepsilon, 1/\delta)$ "bits of non-uniformity." This result is stated in a precise form below as Lemma 4. We then show how to eliminate the non-uniformity and how to do amplification from $1-1/(\log n)^\alpha$ to $3/4+1/(\log n)^\alpha$, where $\alpha > 0$ is an absolute constant. As discussed above, $3/4$ is a natural barrier for uniform black box amplification results.

The Results of This Paper

In this paper we prosent a uniform amplification result from $1-1/\mathrm{poly}(n)$ to $1/2+1/(\log n)^\alpha$. That is, we break through the $3/4$ barrier in one direction and we achieve the "right" bound $1-1/\mathrm{poly}(n)$ in the other direction. Formally, our result is as follows.

Theorem 1 (Main) *Suppose that for every langyuage in* NP *there is a probabilistic polynomial time algorithm that succeeds with probability* $1/2+1/(\log n)^\alpha$ *on inputs of length* n*. Then for every language in* NP *and polynomial* p *there is a probabilistic polynomial time algorithm that succeeds with probability* $1-1/p(n)$ *on inputs of length* n*. The value* $\alpha > 0$ *is an absolute constant.*

2 Overview of the Proof

Here is an overview of our proof. For a function $f : \{0,1\}^n \to \{0,1\}$ and an odd integer k, define $f^{\mathrm{maj},k}(x_1, \ldots, x_k) = \mathrm{majority}\{f(x_1), \ldots, f(x_k)\}$.

1. We begin by showing that if $f : \{0,1\}^n \to \{0,1\}$ is balanced, then an efficient algorithm that solves $f^{\mathrm{maj},k}$ on a $1-O(\delta\sqrt{k})$ fraction of inputs yields an efficient algorithm that solves f on a $1-\delta$ fraction of inputs, provided that $k \le O(\delta^{-2/7})$.

   For example, if there is no efficient algorithm that solves f on a $1-1/n$ fraction of inputs, then there is no efficient algorithm that solves $f^{\mathrm{maj},k}$ on a $1-1/(12\cdot n^{6/7})$ fraction of inputs, where $k = n^{2/7}$.

   This is established in Lemma 5, proved in Section 4.

   This *uniform* analysis of amplification of hardness using the majority function is the main new technical result of this paper.

2. By repeated applications of Lemma 5 we show how to start from a balanced function $f : \{0,1\}^n \to \{0,1\}$ and define a new function $F : \{0,1\}^{n^{1+O(t)}} \to \{0,1\}$ such that an efficient algorithm that solves F on a $1-\varepsilon$ fraction of inputs yields an efficient algorithm that solves f on a $1-1/n^t$ fraction of inputs, where $\varepsilon > 0$ is an absolute constant. This remains true even

if f is only $1/n^{O(t)}$-close to be balanced. We prove this fact in Lemma 6 that, for simplicity, is specialized to the case $t = 1/5$.

To see, very informally, how this works, let us start from a function $f : \{0,1\}^n \to \{0,1\}$ that is hard to solve on more than a $1 - 1/n^t$ fraction of inputs: then we define $f_1 := f^{\mathrm{maj},n^{2t/7}}$, we have that f_1 has inputs of length $n^{1+2t/7}$ and it is hard to solve on more than a $1 - 1/n^{7t/8}$ fraction of inputs (assuming n is large enough). Then we can define $f_2 := f_1^{\mathrm{maj},n^{t\cdot(7/8)\cdot(2/7)}}$, and this new function, defined on inputs of length $n^{1+t\cdot(2/7)+t\cdot(2/7)\cdot(7/8)}$ is hard to solve on more than a $1 - 1/n^{t\cdot(7/8)^2}$ fraction of inputs. Repeating this process i times we get to f_i, which has inputs of length $n^{1+t\cdot\frac{2}{7}\cdot\left(1+\frac{7}{8}+\left(\frac{7}{8}\right)^2+\cdots+\left(\frac{7}{8}\right)^{i-1}\right)}$ and is hard to solve on more than a $1 - 1/n^{t\cdot(7/8)^i}$ fraction of inputs. When $i \approx \log\log n$, f_i has inputs of length $n^{1+O(t)}$ and it is hard to solve on more than a $1 - \varepsilon$ fraction of inputs, where $\varepsilon > 0$ is an absolute constant.

One must, of course, be careful in composing a super-constant number of reductions: if each reduction increased the running time by, say, a factor of n then from a polynomial time algorithm for F we would only deduce a $n^{O(\log\log n)}$ time algorithm for f. The precise statement of Lemma 5 ensures us that at step i we only need to lose a multiplicative factor of $n^{O(t\cdot(7/8)^i)}$ in efficiency.

3. We use a padding trick from [O'D02] to generalize Lemma 6 to the case of general, not necessarily almost balanced, functions, at the price of a small amount of non-uniformity.

   To summarize: the reasoning described so far shows that if for every problem in NP there is an efficient algorithm that solves the problem on a $1 - \varepsilon$ fraction of inputs of each length, then for every problem in NP and every polynomial $p()$ there is an efficient algorithm that solves the problem on a $1 - 1/p(n)$ fraction of inputs of length n *with a small amount of non-uniformity*.

   Combined with a result of [Tre03], we can reach the same conclusion under the weaker assumption that for every problem in NP there is an efficient algorithm that solves the problem on a $1/2 + 1/(\log n)^\alpha$ fraction of inputs of length n.

   Combined with a result from [BDCGL92], from that weaker assumption we can reach the stronger conclusion that for every *search* problem in NP and every $p()$ there is an efficient algorithm that solves the search problem on a $1 - 1/p(n)$ fraction of inputs of length n *with a small amount of non-uniformity*.

   Finally, we eliminate the non-uniformity by enumerating all possible advice strings, and accepting if and only if one we find a witness using one of the advice strings. This (and the use of a search-to-decision reduction) is the "non-black-box" step of our argument.

3 Preliminaries

If L is a language, then we denote by $L_n : \{0,1\}^n \to \{0,1\}$ the characteristic function of L restricted to inputs of length n.

We use the following notion of average-case tractability. We say that a (uniform, probabilistic) algorithm A *succeeds with probability* $p(n)$ on a language L if for, every n,

$$\mathbf{Pr}[A(x) = L_n(x)] \geq p(n)$$

where the probability is taken both over the choice of $x \sim \{0,1\}^n$ and over the internal coin tosses of A. We also introduce a notion of average case tractability for "slightly non-uniform" probabilistic algorithms.

Definition 2 *An probabilistic algorithm A solves a language L with agreement $p(n)$ and ambiguity D if, on input 1^n, A outputs a list of $\leq D$ circuits and, with high probability over the internal coin tosses of A, at least one of the circuits agrees with L_n on at least a $p(n)$ fraction of inputs.*

If we do not care about the size of the list, then the "high probability" in the above statement can be easily amplified by running the algorithm several times independently and then taking the union of the lists.

Note that if we have an algorithm that solves L with ambiguity 1 and agreement $1-\delta$, then we also have, for a stronger reason, an algorithm which succeeds with probability $\approx 1-\delta$: on input x, construct the circuit and evaluate x on it. The success probability is at least $1-\delta$ minus the probability of error of the algorithm, that is, the probability that the output circuit does not agree with L on a $1-\delta$ fraction of inputs of length n. (To prove the claim, note that the randomnes used to pick a random x is independent of the internal coin tosses of the algorithm, and we may think of x as being chosen after the circuit.) If we want to reduce the probability of error of the algorithm while keeping the list of size 1, we can run the algorithm several times independently, obtain circuits $C_1, \ldots, C_K$ and then define $C'(x) = \mathrm{maj}\{C_1(x), \ldots, C_K(x)$. If there was a probability, say, at least $1-\varepsilon$ that a random C_i agrees with L on a $1-\delta$ fraction of inputs, then there are at least $1-3(\delta+\varepsilon)$ inputs x such that $L(x)$ has a probability at least $2/3$ of agreeing with $C_i(x)$. Therefore there is a probability at least $1-e^{-\Omega(k\cdot(\varepsilon+\delta)^2)}$ that C', as described above, agrees with L on at least, say, a $1-4(\delta+\varepsilon)$ fraction of inputs.

Lemma 3 (Decision Versus Search, [BDCGL92]) *Let L be an* NP *language and R be an* NP *relation that defines L, let $w(n)$ be a polynomial that bounds from above the length of a witness for a YES instance of L of length n. Then there is a language L', a polynomial $l()$ and a probabilistic polynomial time algorithm A that, given in input a circuit C' that solves L' on a $1-\delta$ fraction of inputs of length $l(n)$, outputs with probability at least $1-2^{-poly(n)}$ a circuit C that solves the search version of L (with respect to R) on a $1-O(\delta\cdot(w(n))^2)$ fraction of inputs of length n.*

Lemma 3 was stated in a somewhat different form in [BDCGL92]. A statement similar to the one above appears in [BT03]. We will also need the following result from [Tre03].

Lemma 4 ([Tre03]) *For every language L in* NP *there is a language L' such that if there is a probabilistic polynomial time algorithm for L' that succeeds with probability $1/2+1/(\log n)^\alpha$, then there is a probabilistic polynomial time algorithm that solves L with poly(n) ambiguity and $1-1/(\log n)^\alpha$ agreement. The value $\alpha > 0$ is an absolute constant.*

4 The Main Reduction

In this section we prove the following result. Recall that we defined

$$f^{\mathrm{maj},k}(x_1,\ldots,x_k) = \mathrm{majority}\{f(x_1),\ldots,f(x_k)\}\ .$$

Lemma 5 (Main) *There is a probabilistic polynomial time algorithm that on input a parameter $\delta > 0$ and integer parameters n, k, where k is odd and $c \leq k \leq \delta^{-2/7}$, and a circuit C of size s with nk inputs, returns a circuit C' of size $O(s \cdot poly(1/\delta))$ such that the following is true. (c is an absolute constant.)*

If there is a balanced function $f : \{0,1\}^n \to \{0,1\}$ such that C agrees with $f^{\mathrm{maj},k}$ on a $1 - \delta\sqrt{k}/12$ fraction of inputs, then there is a probability at least $1 - 2^{-1/\delta}$ over the randomness of the algorithm that the output circuit C' agrees with f on a $1 - \delta$ fraction of inputs.

Let us start by considering to following, much simpler, scenario: we are given oracle access to $f^{\mathrm{maj},k}$ itself, and, given x, we want to compute $f(x)$. Here, the natural approach is to construct a random k-tuple $(X_1, \ldots, X_k)$ by picking at random $I \in \{1, \ldots, k\}$, setting $X_I = x$, and picking at random X_j for $j \neq I$; then we compute $f^{\mathrm{maj},k}(X_1, \ldots, X_k)$ and we output the result.

To analyze the above process, we need to consider the distribution of $|\{j \neq I : f(X_j) = 1\}|$. If there are precisely $(k-1)/2$ values of $j \neq I$ such that $f(X_j) = 1$, and precisely $(k-1)/2$ such that $f(X_j) = 0$, then $f^{\mathrm{maj},k}(X_1, \ldots, X_k) = f(X_I) = f(x)$, and we find the right value. Note that, under our assumption that f is balanced, this happens with probability $p_k := \binom{k-1}{(k-1)/2} \cdot 2^{-(k-1)}$. Otherwise, if $|\{j \neq I : f(X_j) = 1\}| \geq (k-1)/2 + 1$ then we output 1 regardless of the value of $f(x)$, and if $|\{j \neq I : f(X_j) = 1\}| \leq (k-1)/2 - 1$ then we output 0 regardless of the value of $f(x)$; note that these two events have the same probability. Overall, our output is correct conditioned on a certain event (that happens with probability p_k), and our output is a fair coin conditioned on the event not happening. Overall, we have probability $1/2 + p_k/2$ of giving the correct answer. By Stirling's approximation, the probability is at least $1/2 + (1 - o(1))/\sqrt{2 \cdot \pi \cdot (k-1)}$, which is at least $1/2 + 1/3\sqrt{k}$ if k is sufficiently large.

It is helpful to think of the above analysis as follows. Define a bipartite graph that has a node on the left for each element of $\{0,1\}^n$, and a node on the right for each k-tuple in $(\{0,1\}^n)^k$. For each $j = 1, \ldots, k$ and for each k-tuple $x_1, \ldots, x_k$ we put an edge between the vertex x_j on the left and the vertex $(x_1, \ldots, x_k)$ on the right. (Note that there are some parallel edges.) In total, there are $k \cdot 2^{nk}$ vertices, and the graph is bi-regular with degree $k \cdot 2^{n \cdot (k-1)}$ on the left and k on the right. We have proved that if we label each vertex x on the left with the label $f(x)$, and each vertex $(x_1, \ldots, x_k)$ on the right with the label $f^{\mathrm{maj},k}(x_1, \ldots, x_k)$, then each vertex on the left has a label that agrees with at least a $1/2 + 1/3\sqrt{k}$ fraction of the labels of its neighbors.

Suppose now that, instead of having oracle access to $f^{\mathrm{maj},k}$ we have oracle access to a function C that is α-close to $f^{\mathrm{maj},k}$, with $\alpha := \delta\sqrt{k}/12$. Let us go back to the graph we defined above, and let us label each vertex $(x_1, \ldots, x_k)$ on the right with $C(x_1, \ldots, x_k)$. We say that a vertex x on the left is *bad* if its label agrees with fewer than $1/2 + 1/12\sqrt{k}$ fraction of the labels of its neighbor. We will argue that only a $O(\alpha/\sqrt{k})$ fraction of vertices on the left are bad.

Let B be the set of bad vertices, define $\beta := |B|/ \cdot 2^n$, and let A, with $|A| \leq \alpha \cdot 2^{nk}$ be the set of inputs on which C differ from $f^{\mathrm{maj},k}$. For each bad vertex x in B, at least a $(1/2 - 1/3\sqrt{k}) - (1/2 - 1/12\sqrt{k}) = 1/4\sqrt{k}$ fraction of its outgoing edges are in S, and so there are at least

$$\frac{1}{4}\beta \cdot \sqrt{k} \cdot 2^{nk}$$

edges between B and A. Considering that there is a total of only $k\alpha 2^{kn}$ edges going into A, we note that this already implies $\beta < 4\sqrt{k}\alpha$.

The crucial observation is now that most vertices in the neighborhood of B are *unique neighbors*. Indeed, there are

$$\beta \cdot (1-\beta)^{k-1} \cdot k \cdot 2^{kn} > \beta(1-k\beta)k2^{kn}$$

vertices on the right having exactly one neighbor in B. To see why, consider the experiment of picking a random vertex on the right, that is, a random k-tuple, and then see what is the probability that precisely one element of the k-tuple lands in a set of density β.

Of these unique neighbors, at least $\beta(1-k\beta)k2^{kn} - |A|$ are outside A, and so at least as many edges from B land in the complement of A. Considering that there are $\beta k 2^{kn}$ edges going out of B, it follows that the number of edges between B and A is at most

$$\beta k 2^{kn} - (\beta(1-k\beta)k2^{kn} - |A|) = \beta^2 k^2 2^{kn} + \alpha 2^{kn}$$

Combining the relations that we have discovered so far we get

$$\frac{1}{4}\beta \cdot \sqrt{k} \cdot 2^{kn} < \alpha 2^{kn} + \beta^2 k^2 2^{kn} \leq \alpha 2^{kn} + 16\alpha^2 k^3 2^{kn} \ .$$

Where the last inequality follows from $\beta \leq 4\alpha\sqrt{k}$, a fact that we proved earlier. Recall that we defined $\alpha := \delta\sqrt{k}/12$, and so we get

$$\beta < \frac{1}{3}\delta + \frac{1}{9}\delta^2 k^{3.5} \leq \frac{1}{2}\delta$$

where the last inequality follows from the assumption $\delta \leq k^{-3.5}$.

Wrapping up, for at least a $1-\beta > 1-\delta/2$ fraction of ertices x on the left we have that $\mathbf{Pr}[f(x) = C(x_1, \ldots, x_k)] \geq 1/2 + 1/12\sqrt{k}$, where the probability is taken over the choice of a random neighbor $(x_1, \ldots, x_k)$ of x.

We are finally ready to describe the algorithm claimed in the lemma. The algorithm operates as follows. It fixes $t = O(k/\delta)$ and it picks at random t sequences of k strings of $\{0,1\}^n$. For each such sequence $x_1, \ldots, x_k$ it picks at random $i \in \{1, \ldots, k\}$ and then it defines a circuit that on input x gives the output of $C(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_k)$. Finally, the output circuit C' is the majority of these circuits. If we look at each $x \notin B$, the probability, over the random choices, that $C'(x) = f(x)$ is, by Chernoff bounds, $1 - 2^{-\Omega(t/k)}$ and, by Markov inequality, there is a probability at least $1 - 2^{-\Omega(t/k)} > 1 - 2^{-1/\delta}$ that C' is such that C' agrees with $f()$ on at least a $1 - 2^{-\Omega(t/k)} > 1 - \delta/2$ fraction of the inputs x that are not in B. By our choice of t and by our bound on B it follows that there is a probability at least $1 - 2^{-1/\delta}$ that C' agrees with f on a $1-\delta$ fraction of inputs. Finally, if the size of C was s, then the size of C' is $poly(1/\delta) \cdot s$.

5 Recursive Application of the Reduction

In this section, we show how to compose the reduction with itself, and prove the following result.

Lemma 6 *Let language L in* NP*. Then there is a language L', a polynomially bounded efficiently computable function $\ell(n)$, and a probabilistic polynomial time algorithm that given in input a circuit C' that solves L' on a $\geq 1-\varepsilon$ (where ε is an absolute constant) fraction of inputs of length $\ell(n)$, returns with high probability a circuit C that solves L on a $\geq 1 - 2/n^{1/5}$ fraction of inputs of length n, provided that L is $1/\sqrt{n}$-close to balanced on inputs of length n.*

We choose a such that if $\delta < a$ and we set $k = \delta^{-2/7}$, then $k \geq c$ (the constant of Lemma 5) and $\delta^{7/8} < \delta^{6/7}/20$. We fix $\varepsilon := a^{8/7}/2$.

For a boolean function $f : \{0,1\}^n \to \{0,1\}$, we inductively define functions $f_i : \{0,1\}^{n_i} \to \{0,1\}$ as follows. Define

$$\delta_0 = \frac{1}{n^{1/5}} \quad f_0 = f$$

$$k_i = \frac{1}{\delta_{i-1}^{2/7}} \quad \delta_i = \delta_{i-1}^{7/8} \quad f_i = f_{i-1}^{\mathrm{maj},k_i} \quad n_i = n_{i-1} \cdot k_i$$

Unfolding the recursion, we get

$$k_i = n^{\frac{1}{5}\cdot\frac{2}{7}\cdot\left(\frac{7}{8}\right)^{i-1}} \quad \delta_i = \frac{1}{n^{\frac{1}{5}\cdot\left(\frac{7}{8}\right)^{i}}}$$

Let r be the largest index such that $\delta_r < a$. Then $\delta_r \geq a^{8/7} = 2\varepsilon$.

The input length of f_r is

$$n \cdot k_1 \cdot k_2 \cdots k_r < n^{1+\frac{1}{5}\cdot\frac{2}{7}\cdot\left(1+\frac{7}{8}+\left(\frac{7}{8}\right)^2+\cdots\left(\frac{7}{8}\right)^r\right)} \leq n^{1+\frac{1}{5}\cdot\frac{2}{7}\cdot 8} = n^{1+\frac{16}{35}}$$

Alternatively, we can view f_r to be defined as

$$f_r(x_1, \ldots, x_K) = g(f(x_1), \ldots, f(x_K))$$

where $g()$ is a recursive majority and $K \leq n^{16/35}$.

Suppose that we are given a circuit C_r such that C_r agrees with f_r on a $1 - 2\varepsilon > 1 - \delta_r$ fraction of inputs. Then, by applying the algorithm of Lemma 5 to C_r with parameter δ_r we get, with high probability, a circuit C_{r-1} that agrees with f_{r-1} on at least a $1 - \delta_{r-1}$ fraction of inputs. By repeatedly applying the algorithm of Lemma 5, we eventually get, still with high probability, a circuit C_0 that agrees with f on at least a $1 - \delta_0 = 1 - 1/n^{1/5}$ fraction of inputs.

Let now L be a language in NP, and for every input length n define $L_n : \{0,1\}^n \to \{0,1\}$ to be the characteristic function of L restricted to inputs of length n. Define $L_{n,r} : \{0,1\}^{nK} \to \{0,1\}$ based on L_n in the same way as we define f_r based on f above. (Again, $K \leq n^{16/35}$.) Finally, let L' be the language such that $x \in L'$ if and only if $L_{n,r}(x) = 1$ for some n.

Suppose that we are given a circuit C_r that agrees with $L_{n,r}$ on a $1 - a$ fraction of inputs. Let f be a balanced function that agrees with L_n on a $1 - 1/\sqrt{n}$ fraction of inputs. Then, f_r and $L_{n,r}$ agree on at least $1 - K/\sqrt{n} = 1 - o(1)$ fraction of inputs and, in particular, f_r and C_r agree on at least a $1 - 2a = 1 - \varepsilon$ fraction of inputs if n is large enough. If we repeatedly apply the algorithm of Lemma 5 to C_r then, as explained above, we eventually construct a circuit C that agrees with f on at least a $1 - 1/n^{1/5}$ fraction of inputs. We conclude that C agrees with L_n on at least a $1 - 1/n^{1/5} - 1/\sqrt{n} > 1 - 2/n^{1/5}$ fraction of inputs.

6 The Main Result

6.1 Dealing With Problems That Are Not Balanced

As first step, we present a version of Lemma 6 that has no balance condtion, but that introduces a small amount of non-uniformity. This follows an idea from [O'D02].

Lemma 7 *For every language L in* NP *and polynomial p there is a language L' such that if there is a probabilistic polynomial time algorithm that solves L' with agreement $\geq 1-\varepsilon$ and polynomial ambiguity, then there is a probabilistic polynomial time algorithm that solves L with agreement $1-1/p(n)$ and polynomial ambiguity. The constant ε is the same as in Lemma 6*

PROOF: Let t be such that $p(n) \leq n^{t+1}/6$ for every sufficiently large n. We define a new language $L_{bal} \in$ NP such that to each input length of L there "corresponds" an input length of L_{bal} on which L_{bal} is nearly balanced.

Specifically, L_{bal} is defined as follows. Let $x = (x_0, x_1, \ldots, x_{N-1})$ be an input of length N. Let n be the largest integer such that $N \geq n^{t+1}$ and let $m = N - n^{t+1}$, then

- If $m > n^t$ then $x \notin L_{bal}$;
- If $x_0 = 0$ then x is in L_{bal} if and only $(x_1, \ldots, x_n)$ is one of the first $m \cdot 2^n/n^t$ strings of $\{0,1\}^n$ in lexicographic order;
- If $x_0 = 1$ then $x \in L_{bal}$ if and only if $(x_1, \ldots, x_n) \in L$.

To see the connection between L_{bal} and L, let n be an input length, and define $p_n := \mathbf{Pr}_{x \in B_n}[L_n(x)]$ the fraction of inputs of length n that are in L. Let $N = n^{t+1} + \lfloor n^t \cdot p_n \rfloor$. We observe the following two important points.

Claim 8 *L_{bal} is $1/n^t$-close to balanced on inputs of length N, because between a $1-p_n$ and $1-p_n+1/n^t$ fraction of inputs of the form $0zy$ are in L_{bal}, and a p_n fraction of inputs of the form $1zy$ are in L_{bal}. Considering that $N < (n+1)^{t+1}$, we have that L_{bal} is $O(1/N^{t/(t+1)})$-close to balanced and, in particular, $1/\sqrt{N}$-close to balanced if n is large enough.*

Claim 9 *Suppose that we had an algorithm that succeeds with probability $1-\delta$ on inputs of length N of L_{bal}; then we could get an algorithm that succeeds with probability $1-2\delta$ on inputs of length n of L as follows: on input z, pick y at random and pass $1zy$ to the algorithm for L_{bal}. Similarly, if we are given a circuit C' that agrees with L_{bal} on a $1-\delta$ fraction of inptus of length N, then we can find with high probability a circuit C that agrees with L on, say, a $1-6\delta$ fraction of inputs of length n.*[4]

In summary, if we were able to solve L_{bal} well on average even only on the input lengths on which it is almost balanced, and if we knew at least an approximation to the values p_n, we could solve L well on average on all input lengths.

Starting from L_{bal}, we apply Lemma 6 and we get a language L' and a length function $\ell()$. Suppose that L' can be solved with agreement $1-\varepsilon$ and polynomial ambiguity. Then we have an algorithm for L_{bal} that, given an N on which that L_{bal} is $1/\sqrt{N}$-close to balanced, returns a polynomial number of circuits such that one of them agrees with L_{bal} on a $\geq 1-1/N^{1/5}$ fraction of inputs of length N. (To see that this claim is true, given N, run the assumed algorithm for L' on $\ell(N)$, and

[4] To prove the second statement, pick $y_1, \ldots, y_k$ at random, and then define $C(x) = \text{maj}\{C'(1xy_1), \ldots, C'(1xy_k)\}$. It is easy to see that the probability that $C()$ does not have agreement $1-6\delta$ with L on inputs of length n is at most $e^{-\Omega(k)}$.

get a list of polynomially many circuits such that one of them agrees with L' on a $1-\varepsilon$ fraction of inputs of length $\ell(N)$. Then apply the algorithm of Lemma 6 to each circuit, and get a list of polynomially many circuits such that one of them agrees with L_{bal} on a $1-1/N^{1/5}$ fraction if inputs of length N, provided that L_{bal} was $1/\sqrt{N}$-close to balanced on inputs of length N.)

Now, let n be an input length for L. We "guess" the value $\lfloor \mathbf{Pr}[L_n(x)=1] \cdot n^t \rfloor$ by trying all values of $m = 0, \ldots, n^t$. For each such m, we set $N = n^{t+1} + m$ and we use the above described procedure to construct a list of polynomially many circuits for L_{bal} on inputs of length N (when we use the correct value of m, at least one circuit in the list solves L_{bal} on a $1-1/N^{1/5}$ fraction of inputs), and then we construct a circuit for L on inputs of length n from each such circuit as in Claim 9. For the correct value of m and for the correct circuit in the list, we construct a circuit for L that is correct on a $\geq 1 - 6/N^{1/5} \geq 1 - 1/p(n)$ fraction of inputs. □

6.2 Proof of Theorem 1

Suppose that for for every problem in NP there is a probabilistic polynomial time algorithm that succeeds with probability $1/2 + 1/(\log n)^{\alpha}$

From Lemma 4, we have that for every problem L in NP there is a probabilistic polynomial time algorithm that solves it with polynomial ambiguity and $1-\varepsilon$ agreement. (In fact, recall that Lemma 4 has an even stronger conclusion.)

From Lemma 7, we get that for every polynomial $p()$ and NP problem L there is a probabilistic polynomial time algorithm that solves L with polynomial ambiguity and agreement $1 - 1/p(n)$.

The previous statement and Lemma 3 imply that for every NP search problem L and polynomial $p()$ there is a probabilistic polynomial time algorithm that on input 1^n outputs a list of polynomially many circuits such that, with high probability, at least one of them solves the search problem on a $1 - 1/p(n)$ fraction of inputs. (That is, except possibly for a fraction $1/p(n)$ of inputs, the circuit, given an input x, finds a certificate for x if such a certificate exists.) As discussed in Section 3, the "high probability" could be made, say, $\geq 1 - 1/2^n$ while keeping the list of polynomial size.

After constructing such a list of circuits $C_1, \ldots, C_K$, consider the circuit C that, on input x, accepts if and only if $C_i(x)$ outputs a witness for x for some i. The new circuit is still of polynomial size, always rejects NO instances, and the only YES instances that are rejected are those on which all the circuits C_i fail to solve the search problem, including the circuit that we assumed failed on $\leq 1/p(n)$ fraction of inputs. Overall, C is correct on a $\geq 1 - 1/p(n)$ fraction of inputs.

Summing up, we have proved that, under the assumption of the theorem, for every language L in NP and every polynomial $p()$ there is a probabilistic polynomial time algorithm that solves L with ambiguity 1 and agreement $1 - 1/p(n)$, and whose probability of error is at most $1/2^n$. The conclusion of the Theorem now follows from observations in Section 3.

7 Concluding Remarks

There is a standard way (cf. [Imp02, TV02, Vio03, Tre03]) to view "black box" amplification of hardness results as methods to convert an error correcting code that corrects a small number of errors into an error correcting code that corrects a larger number of errors. Uniform reductions

give unique decoding algorithms for such codes.[5]

The results of Section 4 could be seen as a way of constructing an error-correcting code that can correct up to a $\delta\sqrt{k}/12$ fraction of errors from a "balanced" error-correcting code that can correct up to a δ fraction of errors.[6] The analysis would also give an error-reduction algorithm that given a string that has agreement $1-\delta\sqrt{k}/12$ with a codeword of the new code produces a string that has agreement $1-\delta$ with the original code. One may see some similarity between what we do and the error-reduction code used in the construction of super-concentrator codes [Spi96]. In both cases, each bit of the new code depends on a small number of bits of the old code, and the correspondence is given by a graph that is a good unique-neighbor expander. The property of being a unique-neighbour expander is used in the analsys of the error-reduction algorithm, that performs a simple local computation. The way in which we recursively compose the construction with itself in Section 5 also bears some similarity with the way the basic error-reduction code is used to construct super-concentrator codes.

Acknowledgements

I wish to thank Paul Beame for a conversation in Banff that relighted my interest in this problem. My understanding of the use of majority for hardness amplification benefitted from discussions with Andrej Bogdanov and Irit Dinur on a related problem.

References

[BDCGL92] Shai Ben-David, Benny Chor, Oded Goldreich, and Michael Luby. On the theory of average case complexity. *Journal of Computer and System Sciences*, 44(2):193–219, 1992. 4, 5

[BL93] Dan Boneh and Richard J. Lipton. Amplification of weak learning under the uniform distribution. In *Proceedings of the 6th ACM Conference on Computational Learning Theory*, pages 347–351, 1993. 1

[BT03] Andrej Bogdanov and Luca Trevisan. On wost-case to average-case reductions for NP problems. In *Proceedings of the 44th IEEE Symposium on Foundations of Computer Science*, pages 308–317, 2003. 5

[GNW95] O. Goldreich, N. Nisan, and A. Wigderson. On Yao's XOR lemma. Technical Report TR95-50, Electronic Colloquium on Computational Complexity, 1995. 1

[HVV04] Alexander Healy, Salil Vadhan, and Emanuele Viola. Using nondeterminism to amplify hardness. In *Proceedings of the 36th ACM Symposium on Theory of Computing*, 2004. 2, 3

[5] Unique decoding of binary error correcting codes is only possible if the fraction of errors is at most 1/4, and this is the source of the bottleneck that we mentioned in the introduction.

[6] For the purpose of this discussion, say that an error-correcting code is balanced if every codeword contains an equal number of zeroes and ones.

[Imp95] Russell Impagliazzo. Hard-core distributions for somewhat hard problems. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 538–545, 1995. 1, 2

[Imp02] Russell Impagliazzo. Hardness as randomness: a survey of universal derandomization. In *Proceedings of the International Congress of Mathematicians*, volume 3, pages 659–672, 2002. 2, 10

[IW97] R. Impagliazzo and A. Wigderson. $P = BPP$ unless E has sub-exponential circuits. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 220–229, 1997. 1

[Lev87] Leonid Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363, 1987. 1

[O'D02] Ryan O'Donnell. Hardness amplification within np. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 751–760, 2002. 2, 4, 8

[Spi96] Daniel Spielman. Linear time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1732, 1996. 11

[STV01] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001. 2

[Tre03] Luca Trevisan. List-decoding using the XOR Lemma. In *Proceedings of the 44th IEEE Symposium on Foundations of Computer Science*, pages 126–135, 2003. 2, 3, 4, 5, 10

[TV02] Luca Trevisan and Salil Vadhan. Pseudorandomness and average-case complexity via uniform reductions. In *Proceedings of the 17th IEEE Conference on Computational Complexity*, pages 129–138, 2002. 2, 10

[Vio03] Emanuele Viola. Hardness vs. randomness within alternating time. In *Proceedings of the 18th IEEE Conference on Computational Complexity*, 2003. 2, 10

[Yao82] A.C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23th IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982. 1