Analysis of Boolean Functions **(CMU 18-859S, Spring 2007)**

Lecture 17: Hardness Amplification

March 20, 2007

Lecturer: Ryan O'Donnell *Scribe: Eric Blais*

1 Hardness amplification

In this lecture, we examine how hard it is to compute, or even to approximately compute, boolean functions with small circuits. We begin by defining what we mean exactly by "hard to compute".

Definition 1.1. The function $g : \{-1,1\}^m \to \{-1,1\}$ is *θ-hard for circuits of size s* if for every circuit C of size at most s, $\mathbf{Pr}[C(\boldsymbol{x}) = g(\boldsymbol{x})] \leq \theta$.

The definition is valid for $1/2 \leq \theta \leq 1$. Somewhat counter-intuitively, smaller values of θ denote harder functions. When $\theta = 1$, there may be a circuit of size s that correctly computes g. When $\theta = 1 - 2^{-m}$, the function g is "worst-case hard", and no circuit of size s can exactly compute g. At the other extreme of the range, if a function is $1/2$-hard for circuits of size s, then no circuit of that size can do better than guessing. Note that no function can be more than $1/2$-hard for any circuit size, since one of the constant circuits returning 1 or -1 must correctly predict at least half of the outputs of g.

In this lecture, we will mostly concentrate on the case where s is polynomial in m. We will also focus on topics that revolve around the following general question.

> **Question:** Given a function g that is hard for circuits of size s, how can we construct some other function h, much harder for circuits of size $\approx s$?

The original motivations for this question came from the fields of cryptography and complexity theory. In cryptography, methods to construct very hard functions from slightly hard functions can be used to convert weak hard-core predicates into stronger hard-core predicates.

In complexity theory, methods of constructing very hard functions from slightly hard functions can be used to show that certain complexity classes are very hard-on-average, assuming they are slightly hard. This is in fact our main motivation for this lecture. In particular, we would like to show that, assuming $\mathbf{NP}$ can not be computed by polynomial-size circuits, there are languages in $\mathbf{NP}$ that can not be computed for even $1/2 + \epsilon$ of their inputs by small circuits.

The various notions of hardness of complexity classes can be associated with our definition of θ-hardness of functions.

- "$\mathbf{NP} \not\subseteq \mathbf{P/poly}$": $\exists L \in \mathbf{NP}$ such that if g_m is the indicator function of $L \cap \{-1,1\}^m$ for all $s = \mathrm{poly}(m)$, then for sufficiently large m, g_m is $1 - 2^{-m}$-hard for circuits of size s.

- "$\mathbf{NP}$ is slightly hard-on-average": $\exists L \in \mathbf{NP}$ such that if g_m is the indicator function of $L \cap \{-1,1\}^m$ for all $s = \mathrm{poly}(m)$, then for sufficiently large m, g_m is $1 - \frac{1}{poly(m)}$-hard for circuits of size s.
- "$\mathbf{NP}$ is very hard-on-average"' : $\exists L \in \mathbf{NP}$ such that if g_m is the indicator function of $L \cap \{-1,1\}^m$ for all $s = \mathrm{poly}(m)$, then for sufficiently large m, g_m is $\frac{1}{2} + o_m(1)$-hard for circuits of size s.

Using these definitions, we can now restate our goal theorem as follows:

Goal Theorem. *If* $\mathbf{NP}$ *is slightly hard-on-average, then* $\mathbf{NP}$ *is very hard-on-average.*

It would be even better if we could show that $\mathbf{P} \neq \mathbf{NP}$ implies that $\mathbf{NP}$ is very hard-on-average, but for our result we have to start with the stronger assumption that $\mathbf{NP}$ is slightly hard-on-average. Actually, this stronger statement is known for other complexity classes, such as $\mathbf{PSPACE}$ and $\mathbf{P}^{\#\mathbf{P}}$. For example, it is known that if $\mathbf{PSPACE}$ is not computable by polynomial-size circuits, then $\mathbf{PSPACE}$is slightly hard-on-average, by a random self-reducibility argument.

2 Impagliazzo's hard-core set theorem

In order to prove our goal theorem, we would like to understand *why* certain functions may be hard to compute for circuits of polynomial size. Let us consider a couple of examples of boolean functions that may be hard to compute, and analyze why these functions are hard. For our first example, let

$$g(x) = \begin{cases} \text{some hard function} & \text{, if } x_1 = x_2 = \cdots = x_{\log(1/\epsilon)} = 1 \\ 1 & \text{, otherwise .} \end{cases}$$

The function g is easy to compute for most of its inputs, but very hard to compute when $x_1 = x_2 = \cdots = x_{\log(1/\epsilon)} = 1$. So g is $(1-\epsilon) \cdot 1 + \epsilon \cdot \frac{1}{2} \approx 1 - \frac{\epsilon}{2}$-hard for $\mathrm{poly}(m)$ size.

As a second example, let us consider $g(x) = \text{permanent}(x)$. It's hard to say, but maybe no single input seems harder than any other input in g.

So with these two examples, we see that maybe there are two different reasons why a function can be hard: perhaps it only is hard for a subset of its input, or it may be the case that a function is hard for all of its inputs.

It turns out – and this is a key theorem in what we want to prove – that the hardness of every function lies in a subset of its inputs. To state this theorem precisely, we introduce the definition of hard-core sets.

Definition 2.1. $H \subseteq \{-1,1\}^m$ *is a* γ*-hard-core set against size* s *for* $g : \{-1,1\}^m \to \{-1,1\}$ if for every circuit C of size at most s, $\mathbf{Pr}_{\boldsymbol{x} \sim H}[C(\boldsymbol{x}) = g(\boldsymbol{x})] \leq \frac{1}{2} + \gamma$.

In other words, H is a hard-core set for g if no circuit of size s can do much better than guessing for predicting the output of g on any input chosen from H.

Theorem 2.2 (Impagliazzo '95)**.** *If $g : \{-1,1\}^m \to \{-1,1\}$ is $1-\epsilon$-hard for size s, it has a γ-hard-core set H against size s', with $|H| \geq \epsilon \cdot 2^m$, $s' = \Omega(\gamma^2/\log(1/\epsilon\gamma)) \cdot s$.*

There are multiple ways to prove the theorem. The proof method that yields the optimal parameters in the theorem uses boosting, but we will instead use a quicker proof, due to Nisan.

Proof. In order to prove the theorem, we will first work on getting a "hard-core measure" $\mathcal{H}$, where $\mathcal{H}$ is a probability distribution over sets H of size at least $\epsilon \cdot 2^m$, such that for all circuits C of size at most s,

$$\mathop{\mathbf{E}}_{H\sim\mathcal{H}}\left[\mathop{\mathbf{Pr}}_{\boldsymbol{x}\sim H}[C(\boldsymbol{x}) = g(\boldsymbol{x})]\right] \leq \tfrac{1}{2} + \gamma \ .$$

In order to get $\mathcal{H}$, let us consider a zero-sum game between "set player" and "circuit player". The game works as follows: first, the set player picks a set H with size $|H| \geq \epsilon \cdot 2^m$. Then, the circuit player picks a circuit C of size at most s'. Finally, the set player pays the circuit player $\mathbf{Pr}_{\boldsymbol{x}\sim H}[C(\boldsymbol{x}) = g(\boldsymbol{x})]$.

The von Neumann Minimax theorem implies there exists a value V such that: (1) the set player has a randomized strategy such that for all strategies of the circuit player, the expected amount that the set player pays is $\mathbf{E}[\text{amount set player pays}] \leq V$; and (2) the circuit player has a randomized strategy such that for all strategies of the set player $\mathbf{E}[\text{amount set player pays}] \geq V$. In our search for $\mathcal{H}$, there are two cases for the possible values of V that we need to consider:

- Case 1: $V \leq \frac{1}{2} + \gamma$. In this case, we can take the set player's strategy for $\mathcal{H}$.
- Case 2: $V > \frac{1}{2} + \gamma$.

We want to argue that Case 2 can never happen. We will do this by showing that $V > \frac{1}{2} + \gamma$ contradicts the fact that g is $1-\epsilon$-hard for circuits of size s.

In Case 2, the circuit player has a distribution $\mathcal{C}$ on circuits of size at most s' that achieves average correctness (for g) of at least $\frac{1}{2} + \gamma$ on every set of size $\geq \epsilon \cdot 2^m$. Let

$$S = \{x : \text{average success probability of } \mathcal{C} \text{ on } x < \frac{1}{2} + \frac{\gamma}{10}\} \ .$$

For any set of size $\epsilon \cdot 2^m$, the average correctness is supposed to be at least $\epsilon + \gamma$. So in particular, S can not be of size $\epsilon \cdot 2^m$. In fact, S can not even be of size $\epsilon \cdot 2^m$ minus some small amount. Precisely, we have the following claim.

Claim 2.3. $|S| \leq \epsilon(1 - \frac{\gamma}{2})2^m$.

Proof. Assume the contrary. Then there exists a set S' of size $|S'| \leq \frac{\epsilon\gamma}{2}2^m$ such that $|S \cup S'| \geq \epsilon \cdot 2^m$. The circuit player's distribution $\mathcal{C}$ has average correctness at most 1 on S', and at most $\frac{1}{2} + \frac{\gamma}{10}$ on S, by the definition of S. But $\mathcal{C}$ is supposed to have average correctness greater than $\frac{1}{2} + \gamma$ on $S \cup S'$, which is impossible because $\frac{|S'|}{|S|} \leq \frac{\gamma}{2}$. □

For any $x \notin S$, pick t circuits $C_1, \ldots, C_t$ independently from $\mathcal{C}$, and set $D = \mathrm{Maj}(C_1, \ldots, C_t)$. The size of the circuit D is at most $ts' + O(t)$, since each circuit C_i is of size s', and we can build a linear size circuit to take the majority of their outputs.

So now we want to show that D does very well on g with high probability. Since each C_i is correct with probability $\geq \frac{1}{2} + \frac{\gamma}{10}$, by Chernoff bound $D(x) = g(x)$ except with probability $\leq \exp(-\Omega(\gamma^2 t))$. Take $t = O(\frac{1}{\gamma^2}\log(\frac{2}{\gamma\epsilon}))$. Then $D(x) \neq g(x)$ with probability at most $\frac{\gamma\epsilon}{2}$.

So if we look at all $x \in \{-1,1\}^m$, D has error with respect to g of at most $\epsilon(1-\frac{\gamma}{2})2^m + \frac{\gamma\epsilon}{2}2^m < \epsilon \cdot 2^m$. But D has size $O(ts') \leq s$, so we have contradicted the fact that g is $1-\epsilon$-hard for circuits of size s.

By the above, we have shown that there always is a "hard-core measure" $\mathcal{H}$. To complete the proof, we need to get a hard-core set H_0 from $\mathcal{H}$. To do this, choose each $x \in \{-1,1\}^m$ to be in H_0 independently with probability $\mathbf{Pr}_{H\sim\mathcal{H}}[x \in H]$. The expected cardinality of H_0 is $\geq \epsilon \cdot 2^m$, and with probability at least $\frac{1}{2}$, it is at least $\epsilon \cdot 2^m$. With very high probability, H_0 is 2γ-hard-core for g. (We omit the proof of this statement, but it can be obtained easily: simply apply the union bound over all circuits.) □

Remark 2.4. If H is a γ-hard-core set for g, then g is almost balanced on H. By playing with H slightly, can get g exactly balanced on H, losing only a factor of 2 on γ.

3 Yao's XOR Lemma

Having gained more understanding into the reasons why functions can be hard, let us return to our original question: given a hard function, how can we get an even harder function? As we mentioned above, there has been much work on this question in the field of cryptography, in order to turn hard-core predicates into strong hard-core predicates. A classic result in this area is Yao's XOR Lemma, which basically says that if you have a hard function g, then you can take the XOR of many copies of g; since g was hard to compute originally, then the resulting function is even harder to compute.

Theorem 3.1 (Yao's XOR Lemma)**.** *If $g : \{-1,1\}^m \to \{-1,1\}$ is $(1-\epsilon)$-hard for size s, then $h : \{-1,1\}^{n\cdot m} \to \{-1,1\}$ defined by $h(x^1, \ldots, x^n) = g(x^1) \cdot \cdots \cdot g(x^n)$ is $\frac{1}{2} + (1-\epsilon)^n + \gamma$-hard for size $\Omega(\gamma^2 \log(\frac{1}{\epsilon\gamma})) \cdot s$.*

Let's examine what we can obtain from this theorem. Suppose for example that (g_m) is a function family, $g_m : \{-1,1\}^m \to \{-1,1\}$ which is $1 - \frac{1}{m^{100}}$-hard for circuits of size $\mathrm{poly}(m)$. Take $n = m^{200}$, $\gamma = \frac{1}{m^{100000}}$. Then $h : \{-1,1\}^{m^{300}} \to \{-1,1\}$, when h is obtained by taking the XOR of n functions g_m, is $\frac{1}{2} + (1 - \frac{1}{m^{100}})^{m^{200}} + \frac{1}{m^{100000}} \approx \frac{1}{2} + \frac{1}{m^{100000}}$-hard for circuits of size $\mathrm{poly}(m)$. And m^{100000} is an extremely large polynomial in the input length m^{300}. Therefore, we get a family (h) which is $\frac{1}{2} + \frac{1}{\mathrm{poly}(N)}$-hard for circuits of size $\mathrm{poly}(N)$.

The above example shows that the XOR Lemma accomplishes a lot: we can start with a function family that is slightly hard for polynomial circuits, and create a function family that is tremendously hard for polynomial circuits. Then again, that fact by itself is no big deal: we can just

pick a random function, and it is extremely hard for polynomial size circuits. Yao's XOR Lemma becomes more interesting if we have some sort of upper bound on the complexity of the original function family (g_m). What we want to achieve is to be able to show that, supposing we have a function family in some complexity class that is slightly hard, then we can get some other function family that is extremely hard for some other related complexity class.

For example, as a corollary of the previous example.

Corollary 3.2. *If* $\mathbf{PSPACE}$ *is* $1 - \frac{1}{n^{100}}$*-hard for polynomial circuits, then it's* $\frac{1}{2} + \frac{1}{\mathrm{poly}(n)}$*-hard.*

This result is especially interesting because we also know that if $\mathbf{PSPACE}$ is $1 - 2^{-n}$-hard for polynomial circuits, then $\mathbf{PSPACE}$ is $1 - \frac{1}{n}$-hard. This is an old result due to Lipton (although Lipton actually showed this result for $\mathbf{P}^{\#\mathbf{P}}$ instead of $\mathbf{PSPACE}$).

Let us now turn back to our favorite complexity class: can we obtain a corollary equivalent to Corollary 3.2 for $\mathbf{NP}$ instead of $\mathbf{PSPACE}$? Not with Yao's XOR Lemma, since $(g_m) \in \mathbf{NP}$ does not imply that the function $h = g_m \oplus g_m \oplus \cdots \oplus g_m$ is also in $\mathbf{NP}$. (In fact, if that were the case, then it would imply $\mathbf{NP} = \mathbf{coNP}$.) So if we want to obtain a similar corollary for $\mathbf{NP}$, we need to find a different approach. A natural direction to take is to see if we can replace the XOR function in Yao's XOR Lemma with other functions.

Definition 3.3. Let $f : \{-1,1\}^n \to \{-1,1\}$, and let $g : \{-1,1\}^m \to \{-1,1\}$. Then $f \otimes g : \{-1,1\}^{nm} \to \{-1,1\}$ is the function defined by $f \otimes g(x^1, \ldots, x^n) = f(g(x^1), \ldots, g(x^n))$.

Since we are interested in identifying functions f that will help us amplify hardness in $\mathbf{NP}$, our first task is to determine which functions f can be composed with some function family $(g_m) \in \mathbf{NP}$ to obtain a function $(f \otimes g_m) \in \mathbf{NP}$.

Proposition 3.4. *Suppose* $(g_m) \in \mathbf{NP}$*,* f *is monotone and* $(f) \in \mathbf{NP}$*. Then* $(f \otimes g_m)$ *is in* $\mathbf{NP}$*.*

The plan to obtain hardness amplification of $\mathbf{NP}$ is the following: we will look at various monotone functions f's in $\mathbf{NP}$ and try to understand how hard $f \otimes g$ is as a function of: (1) g's hardness, and (2) intrinsic properties of f. As we will see in the next lecture, what we will need is that f has high "noise sensitivity".