CS 710: Complexity Theory 3/16/2010

Lecture 17: Time-Bounded Derandomization

Instructor: Dieter van Melkebeek Scribe: Tom Watson and Andrew Bolanowski

In the last lecture we introduced the notion of a pseudorandom generator (PRG), showed how PRGs can be used for derandomization, and developed a construction of a PRG that fools space-bounded computations. In particular, we developed a PRG with seed length $O(\log^2 n)$ that fools BPL computations. In the time-bounded setting, there is the trivial simulation for a PRG in EXP time, but nothing better is known unconditionally. There are constructions that are known to work under certain reasonable complexity-theoretic hypotheses. In this lecture, we will present one such construction, due to Nisan and Wigderson [1]. Under a sufficiently strong (but still reasonable) hypothesis, this PRG allows us to show that BPP = P.

1 Pseudorandom Generators for Time-Bounded Computations

1.1 Distinguishability

Recall that a PRG takes a truly random seed of length $\ell(r)$ and produces a "pseudorandom" string of length r. To be useful, a PRG should be efficiently computable by a deterministic machine. A PRG is called *quick* if it can be computed in time $2^{O(\ell(r))}$, i.e. in time linear exponential in its seed length. We will show that if there exists a language in E with large average-case circuit complexity, then there exists a quick PRG with short seed length that fools time-bounded randomized computations. We will formalize the notion of average-case complexity in Section 2.1. In the next lecture, we will see how to use error-correcting codes to relax our hypothesis from the existence of an average-case hard language to the existence of a worst-case hard language.

The notion of "quickness" may not seem to be efficient enough, and indeed in the cryptographic setting PRGs are typically required to be computable in time polynomial in the seed length. Also, this may not be efficient enough if our goal is merely to reduce the amount of randomness needed by a computation. However, our present focus is full derandomization, achieved by trying all possible seeds and explicitly computing the probability that our algorithm accepts under the pseudorandom distribution. In this setting, we need $2^{\ell(r)}$ time just to look at all possible seeds, and so the factor $2^{O(\ell(r))}$ overhead in computing the PRG's output is sub-exponential and just a polynomial overhead in time if $\ell(r)$ is logarithmic.

To guage the quality of our PRG construction, we will need measures of how powerful the computations we are trying to fool are allowed to be, and how well we fool these computations. These measures are formalized by the parameters r and ϵ in the following definition.

Definition 1. *An* ϵ-PRG *for circuits of size* r *is a family of functions* $(G_r)_{r\in\mathbb{N}}$ *where* $G_r : \{0,1\}^{\ell(r)} \to \{0,1\}^r$ *such that for all circuits* C *that take* r *inputs and are of size at most* r,

$$\left| Pr_{\sigma\in\{0,1\}^{\ell(r)}}\left[C(G_r(\sigma))=1\right] - Pr_{\rho\in\{0,1\}^r}\left[C(\rho)=1\right] \right| < \epsilon$$

where σ *and* ρ *are chosen uniformly at random.*

Intuitively, this definition means that every circuit of size at most r will have trouble distinguishing whether its input was sampled from the uniform distribution or from the pseudorandom distribution. Since we typically have the error rate of a random algorithm be up to $1/3$, ϵ should be less than $1/6$ for this definition to be useful.

There are a few questions about the above definition that naturally present themselves.

(1) Why do we require that our PRG fool circuits when we're really interested in fooling uniform computations? Since BPTIME(t) computations can be mimicked by circuits of size polynomial in t, we will also be able to use such a PRG to fool the uniform computations. We want our PRG to succeed in fooling the computations on all but finitely many inputs, and this is easily captured in the nonuniform setting by constructing a different circuit for each input where the input is hard-wired and the random bits are left as inputs to the circuit. Also, it turns out that our arguments critically use the nonuniformity of the circuits. There are also results that start from a uniform hardness assumption, but those results aren't as strong.

(2) Why do we only require that our PRG fool circuits of linear size? Using the same parameter r for the size of the circuit and its number of inputs will keep the arguments cleaner, and there's no harm in allowing the circuit to take more random bits than it needs. Mimicking a uniform computation with a circuit may yield a circuit that's larger than the number of random bits it needs, but the computation won't be affected by allowing the PRG to provide more random bits.

Recall from the last lecture that a PRG can be used for full derandomization by trying all possible seeds and explicity computing the probability of acceptance of the algorithm under the pseudorandom distribution. The running time becomes the time to run the PRG on a given seed plus the time to run a simulation of the algorithm, times $2^{\ell(r)}$ seeds. Thus if we can get a quick PRG with $O(\log r)$ seed length, then this full derandomization runs in polynomial time, implying that BPP = P. Our ultimate goal is to show that if a sufficiently hard function exists, then such a PRG exists.

Recall from last lecture the discussion about the seemingly weaker unpredictability requirement. Recall that unpredictability means that no algorithm from the class can predict the next bit of the pseudorandom sequence given a prefix significantly better than half of the time. Getting half right is trivial. We had shown that these definitions are actually equivalent with the exception that the ϵ changes by a factor of r. We will use the weaker version.

2 The Nisan-Wigderson Construction

2.1 Average-Case Circuit Complexity

We want to construct an ϵ-PRG for circuits of size r. We argued above that if some circuit of size r distinguishes the pseudorandom distribution from the uniform distribution by at least an ϵ amount, then there exists an index i and another circuit of size r that succeeds in predicting the ith bit of a sample from the first $i-1$ bits with advantage at least ϵ/r over the trivial bound of $1/2$. Thus our task is reduced to constructing a PRG such that no small circuit can gain a significant advantage in predicting any bit of a sample from the previous bits.

Intuitively, this suggests that our PRG should generate each bit of its output by applying some hard function, so that we can argue that having a small predictor circuit for some index i would allow us to construct a circuit evaluating the function that was used to generate the ith bit. We now formalize the precise notion of hardness we will need.

Definition 2. *For a language L, the average-case hardness of L at input length m, denoted $H_L(m)$, is the largest s such that no circuit of size at most s can compute L correctly on at least a $\frac{1}{2} + \frac{1}{s}$ fraction of the inputs of length m.*

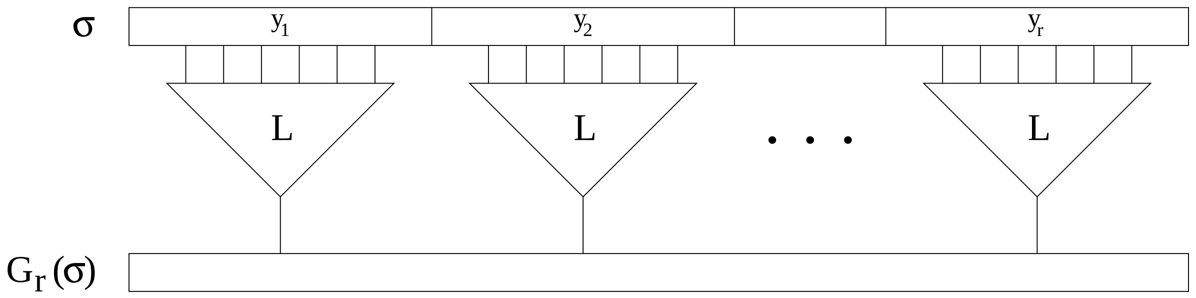
Note that computing L correctly on at least a $1/2$ fraction of the inputs at a given length is trivial—either constant 0 or constant 1 will do the job. An average-case hard function is one that is not only hard to compute exactly, but also hard to compute correctly on noticeably more than half the inputs.

One might wonder why the above definition uses s to refer to both the size of the circuits under consideration and the degree of hardness. The main reason is to keep the number of parameters small, so that our analysis works out cleanly. We combine the constraints of having the chance of being correct be very close to 1/2 and the size of the circuit be very large. One might also wonder why we are measuring hardness against *nonuniform* circuits. There is a very good reason for this—our arguments will crucially use this nonuniformity.

Let us gain some intuition about Definition 2. As mentioned in a previous lecture, every predicate on m bits can be computed exactly by a circuit of size at most 2^m, so $H_L(m) < 2^m$ for all L. As s gets smaller, the condition of Definition 2 gets easier to satisfy: the circuits under consideration become more computationally restricted, *and* they're required to compute L on more inputs. Thus $H_L(m)$ serves as a measure of the average-case hardness of L at input length m.

2.2 PRG Construction

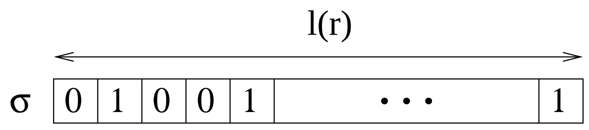
Suppose we have a language L such that $H_L(m) \geq r/\epsilon$. Then no circuit of size r/ϵ, and in particular no circuit of size r, can compute L with probability at least $\frac{1}{2} + \frac{\epsilon}{r}$ over inputs of length m chosen uniformly at random. This suggests the following approach for constructing a PRG: given $y_1, \ldots, y_r \in \{0,1\}^m$ chosen independently and uniformly at random, apply L (viewed as a function producing a bit) to each y_i, yielding an output string of length r. Intuitively, if some bit of the output distribution of this PRG were predictable, then since the samples y_i are chosen independently, such a predictor would have an advantage in computing L. Thus the output distribution of this PRG would be unpredictable and hence indistinguishable from the uniform distribution, as desired.



We will not proceed to formalize this very vague idea because it is seriously flawed. One issue that naturally arises is that computing the output of such a function would require computing L, which is assumed to be hard to compute. However, the length m at which we would be computing

L would ideally be much less than the output length r, so the complexity of computing L might not be prohibitive. A much more critical problem is that this construction takes a seed of length mr but only outputs r bits! We want our seed length to be much smaller than r, and certainly not larger. It is trivial to build a PRG when the seed length is at least as large as the output length — we can just output some of the bits of the seed, yielding a uniform output distribution. The reason the above construction requires such a long seed is that all y_i's are to be chosen independently. We would like to show that by sacrificing some independence of the y_i's, we can drastically reduce the seed length without the quality of the output distribution deteriorating by too much. Say we let the y_i's overlap a little bit thus reducing the incoming number of random bits. If they overlap in a logarithmic number of positions, they will still be random enough.

We will accomplish this by taking a seed σ of length $\ell(r) > m$ and selecting r subsets S_i $(i = 1, \dots, r)$ of the bit positions of the seed, and letting $y_i = \sigma|_{S_i}$ be the bits of the seed indexed by S_i. For example, if $S_1 = \{1, 3, \ell(r)\}$ and the seed σ is as illustrated below, then $y_1 = 001$.



The desired subset construction is formalized in the following definition.

Definition 3. *An* (m, a)-design of size r over $[\ell] = \{1, \dots, \ell\}$ *is a sequence of subsets* $S_1, \dots, S_r \subseteq [\ell]$ *such that* $|S_i| = m$ *for all* i, *and* $|S_i \cap S_j| \leq a$ *for all* $i \neq j$.

We want our PRG output length r to be large, but at the same time we want the pairwise intersections of the S_i's to be small so that the y_i's are "as independent as possble." These two goals are at odds with each other, but the following lemma shows that, in fact, not only do there exist such designs with large r, but the subsets can be efficiently computed.

Lemma 1. *For all* r *and* $m \geq \log r$, *there exists an efficiently computable* $(m, \log r)$*-design of size* r *over* $[\ell]$ *where* $\ell = O(m^2)$.

Proof. Assume m is a prime power so that GF(m) is a field. If it's not, we can pad to the next prime power to make it a field. Let the S_i's be the graph of a univariate polynomial of degree at most n over $GF(m)$. The graph here mean the pairs $(x, f(x))$. $S_i = \{(x, q_i(x)) : x \in GF(m)\}$ where q_i is the ith univariate polynomial. So the universe is $GF(m) \times GF(m)$, that is, with x in the first is $GF(m)$ and $f(x)$ in the second. $|S_i| = m$ for all i Then $|S_i \cap S_j| \leq n$ follows from the fact that the polynomials of degree less than n represent distinct functions and can have at most n points in common. We still need enough polynomials, so we need m large enough. That is, we need $m^n \geq r$. This however is implied by $m \geq \log r$ To do this efficiently, we can assume that m is prime, not just a prime power, to make the calculations easier. The calculations can then be done in $2^{O(\ell(n))}$, which is fast enough.

□

Note that under a different construction, we can make $\ell = O(\log r)$ instead of $\ell = O(\log r^2)$.

The condition that $m \geq \log r$ is no problem for us since we will want m to be such that $H_L(m) \geq \frac{r}{\epsilon} \geq r$, and since $H_L(m) \leq 2^m$, we get the constraint $m \geq \log r$ anyway.

We are now in a position to fully specify our PRG. For a given r and m, we can set $\ell(r) = O(m^2)$ and obtain an $(m, \log r)$-design $S_1, \ldots, S_r$ via Lemma 1. Then our PRG G_r will be

$$G_r(\sigma) = L(\sigma|_{S_1})L(\sigma|_{S_2}) \cdots L(\sigma|_{S_r}).$$

It is straightforward to verify that if L is computable in linear exponential time, then this PRG is quick. All that remains is to show that this construction fools circuits of size r if L is sufficiently hard.

Theorem 1. *If $H_L(m) \geq \frac{r}{\epsilon}$ and $\epsilon \leq \frac{1}{r}$, then the above construction is an ϵ-PRG for circuits of size r.*

Proof. We will prove the theorem by contradiction. Suppose that for some circuit C of size r, we have

$$\Big| Pr_{\sigma \in \{0,1\}^{\ell(r)}}\big[C(G_r(\sigma)) = 1\big] - Pr_{\rho \in \{0,1\}^r}\big[C(\rho) = 1\big] \Big| \geq \epsilon.$$

We will show that then $H_L(m) < \frac{r}{\epsilon}$ by exhibiting a circuit of size at most $\frac{r}{\epsilon}$ that solves L at length m on at least a $\frac{1}{2} + \frac{\epsilon}{r}$ fraction of the inputs, thus contradicting the assumed hardness of L.

There exists an $i \in \{1, \ldots, r\}$ and a circuit P of size at most r such that

$$Pr\big[P\big((G_r(\sigma))_1, \ldots, (G_r(\sigma))_{i-1}\big) = (G_r(\sigma))_i\big] \geq \frac{1}{2} + \frac{\epsilon}{r}.$$

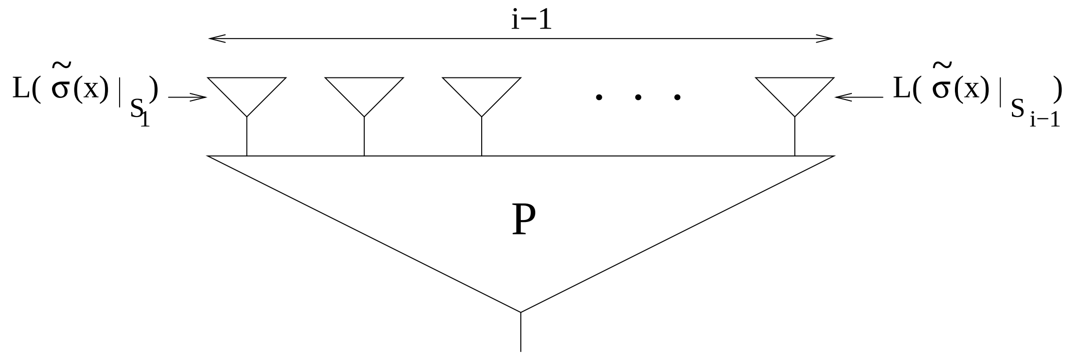
We would like to use P to construct a small circuit that will approximate L well at length m. Intuitively, P seems to be approximating L on input $\sigma|_{S_i}$, and in fact, by another averaging argument we can fix some setting to the bits of σ other than those indexed by S_i such that the predictor P maintains its ϵ/r advantage. Here we are again critically use the fact that we are working with nonuniform circuits. Renaming $\sigma|_{S_i}$ to x and letting $\widetilde{\sigma}(x)$ denote the $\ell(r)$ bits where x fills the positions indexed by S_i and the rest of the positions are fixed as above, we have

$$Pr\big[P\big((G_r(\widetilde{\sigma}(x)))_1, \ldots, (G_r(\widetilde{\sigma}(x)))_{i-1}\big) = L(x)\big] \geq \frac{1}{2} + \frac{\epsilon}{r}$$

where the probability is over x chosen uniformly at random from $\{0,1\}^m$. This is exactly the sort of behavior we would like, but we need to construct a circuit that takes input x, whereas P takes the first $i-1$ bits of G_r's output. We cannot just attach a circuit computing G_r to P, since computing G_r involves computing L exactly. But now we come to the most critical observation of the entire argument: each input to P depends only on at most $\log r$ bits of x, since $|S_i \cap S_j| \leq \log r$ for $j \neq i$, and can thus be computed from x by a circuit of size at most $2^{\log r} = r$. Having the limited pairwise intersections of the subsets in the design is the key to avoiding the inherent complexity of computing L, allowing us to get a contradiction. To the jth input of P, we can attach a circuit of size at most r computing $(G_r(\widetilde{\sigma}(x)))_j = L(\widetilde{\sigma}(x)|_{S_j})$ from x, as illustrated below.

Since $i < r$, we have thus obtained a circuit of size at most $r^2 \leq r/\epsilon$ that succeeds in computing $L(x)$ with probability at least $\frac{1}{2} + \frac{\epsilon}{r}$ over the choice of x. Since $|x| = m$, we conclude that $H_L(m) < \frac{r}{\epsilon}$. We have our contradiction, and the theorem is proved. □

To summarize the above proof, we assumed that our PRG's output distribution was distinguishable from the uniform distribution by a small circuit, obtained a small predictor circuit for some bit of the pseudorandom distribution, and connected some additional small circuitry to this predictor to convert it into a small circuit that approximated L well, thus showing that L cannot be too hard. We can eliminate the ϵ parameter by setting $\epsilon = 1/r$ to obtain the following clean corollary.



Corollary 1. *If $L \in \mathrm{E}$, m, and r are such that $H_L(m) \geq r^2$, then there exists a quick $\frac{1}{r}$-PRG for circuits of size r with seed length $O(m^2)$.*

We remark that no languages in E are known to satisfy the hardness condition $H_L(m) \geq r^2$ on arbitrary circuits for interesting values of m, say $m = r^{o(1)}$. We list some results with reasonable assumptions.

If $L \in \mathrm{E}$, m, and r are such that $H_L(m) \geq m^{\omega(1)}$, a superpolynomial, then $\ell(r) = r^{o(1)}$ We need $H_L(m) \geq r^2$. with $\ell = m^2$. So r^ϵ is enough. This gives us subexponential time.

If $L \in \mathrm{E}$, m, and r are such that $H_L(m) \geq 2^{m^{\Omega(1)}}$ then $\ell(r) = \log r^{O(1)}$ Quasi-polynomial time

The most we can hope for is that $L \in \mathrm{E}$, m, and r are such that $H_L(m) \geq 2^{\Omega(m)}$ In which case $\ell(r) = \log r^2$ which while doesn't give us the polynomial time, we can do the other setup to get $\ell(r) = \log r$

Again we don't know if any such L actually exist.

However, we will see in the next lecture that if there exist languages in E requiring large circuits in the worst case, which is conjectured to be true, then such average-case hard languages do exist. We will use error correction codes. Notice also that the results give us a relationship between hardness and randomness. Intuitively, if there exists a hard problem, then randomness is useless. We can do a similar construction with branching problems. Derandomization gives BPP lower bounds.

2.3 Extensions

2.3.1 Constant-Depth Circuits

In a previous lecture we saw average-case hardness results for parity on constant-depth circuits. With such hardness results, the Nisan-Wigderson construction yields unconditional PRGs for constant depth circuits. Supposing we have a distinguisher for the PRG's output distribution, we can obtain a predictor without increasing the depth. The final circuit approximating parity is obtained by adding depth-2 circuits computing the first $i-1$ bits of the PRG's output in DNF or CNF. Thus this PRG succeeds in fooling constant-depth circuits. It has polylogarithmic seed length, and only involves computing the parity of short strings.

2.3.2 Space-Bounded Derandomization

A similar construction to the one discussed today yields PRGs for branching programs that can be used for space-bounded derandomization. In particular, if there exists a language in DSPACE(n)

that is average-case hard for linear exponential size branching programs, then this construction yields a (conditional) PRG for space-bounded computations.

2.3.3 Worst-Case to Average-Case Reductions

In the next lecture, we will see how to use error-correcting codes to relax our hardness requirement from average-case hardness to worst-case hardness. We will show that encoding the characteristic sequence of a worst-case hard function with a good error-correcting code can yield an average-case hard function, since a circuit that approximates the "encoded" function could be combined with an efficient decoder to compute the original function on every input. Combining this technique with the results of today's lecture, we can obtain a PRG that fools polynomial-size circuits and has logarithmic seed length, leading to the following result.

Theorem 2. *If there exists a language in* E *requiring linear exponential size circuits, then* $\mathrm{BPP} = \mathrm{P}$.

References

[1] N. Nisan and A. Wigderson. Hardness vs. Randomness. *Journal of Computer and System Sciences*, 49:149-167, 1994.