CS 710: Complexity Theory 4/6/2010

Lecture 18: Randomness Extraction

Instructor: Dieter van Melkebeek Scribe: Beth Skubak, Jeff Kinne

In the previous lecture, we gave some evidence that randomness is not very powerful in terms of reducing the complexity of solving decision problems. We have seen an unconditional pseudorandom generator that fools space-bounded computations, and a conditional pseudorandom generator that fools time-bounded computations under the plausible hypothesis that there exists a language in E requiring linear exponential size circuits. In fact, it is conjectured that using randomness can only lead to a polynomial factor savings in time and a constant factor savings in space. This does not mean, however, that randomness is useless in practice. On the contrary, for example even a quadratic speedup achieved with randomness may be practically very attractive. Additionally, many randomized algorithms are simpler and easier to implement than deterministic algorithms for the same problems.

We now turn to a different question. Most randomized algorithms assume access to a perfect source of unbiased, and more importantly uncorrelated (independent), random bits. How do we run such algorithms with access to an imperfect random source? The goal of randomness extraction is to take samples from a weak random source — one where samples may not be uniformly distributed, but do have some inherent randomness — and generate samples that are close to being uniformly distributed. Such weak random sources will be our models for physical sources of randomness, such as keystrokes or delays over networks.

An *extractor* is an efficient deterministic procedure for taking a sample from such an imperfect source and "extracting" the randomness from it, producing an output string that is shorter but much closer to being uniformly distributed. Such a procedure can be used to run randomized algorithms with weak random sources. The fact that the output distribution of an extractor is not exactly uniform will have only a small effect on the output distribution of the randomized algorithm, and we can have confidence in the answer just as we would if we had access to perfect random sources. Today we formally define randomness extractors, explore some applications, and give a few constructions.

1 Entropy

Before we embark on the task of constructing an extractor, we need to formalize what we mean by the "amount of randomness" contained in our weak random source, and by "closeness to uniform" of the output distribution obtained by applying our extractor to our weak random source. For the latter, we will use the standard measure of statistical distance. For the former, one idea is to use the measure of entropy from physics.

Definition 1. *The* entropy *(or Shannon entropy) of a discrete random variable X is*

$$H(X) = E\Big[\log \frac{1}{p_i}\Big] = \sum_i p_i \log \frac{1}{p_i}$$

where the sum is over the range of X, and $p_i = Pr[X = i]$.

For example, if X is a uniform random variable on a string of r bits, each $p_i = \frac{1}{2^r}$, so that $log\frac{1}{p_i} = r$ and so $H(X) = r$ as expected.

This measure of randomness, however, does not work in our setting. Indeed, suppose that the range of X is $\{0,1\}^m$, that with probability one half X is 0^m, and that the other half of the time X is uniform over the remaining nonzero strings. Then the entropy measure indicates that X has a fair amount of randomness: $log\frac{1}{p_{0^m}} = 1$, while for nonzero strings i, we have $log\frac{1}{p_i} \simeq m - 1$, and so $H(X) \simeq \frac{r}{2}$.

But note that X is useless for simulating a bounded-error randomized algorithm — if 0^m is in the bad set for a particular input, then the probability of error on that input is greater than half!

Instead, we will require that for X to have a large "amount of randomness", it must be the case that no string is given too much weight. This suggests the following measure.

Definition 2. *The* min-entropy *of a discrete random variable X is*

$$H_\infty(X) = \min_i \log \frac{1}{p_i}.$$

Equivalently, $H_\infty(X)$ is the largest value of k such that all outcomes have probability at most 2^{-k} under X.

We will say that a source X with $H_\infty(X) \geq k$ has at least k bits of randomness.

Our goal is to construct extractors such that given a source with min-entropy at least k, the output distribution of the extractor is statistically close to the uniform distribution on strings of length k.

2 Weak Random Sources

We will show that, given a source with $H_\infty(X) \geq n^{\Omega(1)}$, we can in polynomial time use the source to run polynomial time randomized algorithms. In addition, it can be shown that any source that can be used for randomness extraction, which we define formally in Definition 3, is close to a source with min-entropy $n^{\Omega(1)}$ (see Exercise 2). That is, not only is this min-entropy necessary, it is sufficient. Hence, min-entropy is the correct notion of randomness for our goal. Therefore, we call a source a *weak random source* if $H_\infty(X) \geq n^{\Omega(1)}$.

There are a number of sources with high min-entropy with simple descriptions.

- *Bit fixing sources.* These are sources where a certain number of bits are fixed adversarially, and the remaining k bits are perfectly random. For such a source, $H_\infty(X) = k$. If $k = n^{\Omega(1)}$, this gives a weak random source.

- *Unpredictable or "Somewhat random" sources.* These are sources where each bit is unpredictable given previous bits, namely for each i, $\Pr[X_{i+1} = 0 | X_0, X_1, ..., X_i] \in [1/2 - \delta, 1/2 + \delta]$ for some $\delta < 1/2$. That is, given any prefix of bits, the probability that the next bit is 0, say, is not too strongly biased. For $\delta \leq 1/2 - 1/n^{1-\epsilon}$ for any constant $\epsilon < 1$ this gives a weak random source.

- *Flat sources.* These sources correspond to the uniform distribution over some subset S of the range $\{0,1\}^n$. It follows that $H_\infty(X) = \log |S|$, and such a source is a weak random source if $|S| = 2^{n^{\Omega(1)}}$. The following exercise shows that in fact flat sources are a base case for all

sources with high min-entropy, and therefore if we can handle flat sources then we can handle all random sources with high min-entropy.

Exercise 1. *Let X be a random source with $H_\infty(X) \geq k$. Then X is a convex combination of flat sources each on a subset of size at least 2^k.*

3 Randomness Extractors

Given a source of randomness with high min-entropy, we wish to output a distribution that is statistically close to uniform. Ideally, we would like an extractor to be able to convert weak randomness into near perfect randomness without using any additional perfect randomness. However, the following shows this is not possible.

Proposition 1. *Let $E : \{0,1\}^n \to \{0,1\}^m$ be a function taking input from a weak random source. There is a weak random source X with $H_\infty(X) = n - 1$ so that when $m = 1$, $E(X)$ is a constant function.*

Proof. In this setting, E outputs a single bit and so must output either 0 or 1 with probability $\geq 1/2$; suppose 0. Define X to be the flat distribution on $S = \{x | E(x) = 0\}$. Then X has min-entropy at least $n - 1$, yet $\Pr[E(X) = 0] = 1$ meaning that the output distribution of E is a constant. □

We think of E in the above as taking a weak random source as input and attempting to output bits that are close to uniform. The proposition shows this cannot be done so simply. We therefore augment E with an additional input that comes from a perfect random source.

Definition 3 (Extractor)**.** *The function $E : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^m$ is a (k, ϵ) extractor if for all X on $\{0,1\}^n$ with $H_\infty(X) \geq k$,*

$$\| E(X, U_\ell) - U_m \|_1 < 2\epsilon \tag{1}$$

where U_ℓ is a uniform variable on ℓ bits and U_m is uniform on m bits.

Note that (1) is equivalent to the following: for every event $A \subseteq \{0,1\}^m$,

$$|\Pr[E(X, U_\ell) \in A] - \Pr[U_m \in A]| < \epsilon. \tag{2}$$

Although we need some additional true randomness to define the extractor, we often will be in the setting where $\ell = O(\log n)$, meaning the amount of true randomness needed is very small. In fact, we will see uses of extractors in the next section that eliminate the need for any true randomness in an algorithm by cycling over all of the possible additonal random strings in $\{0,1\}^\ell$ that could be input to the extractor.

There are two parameters of Definition 3 that we wish to optimize: we want to use as few true random bits as possible, and we want to have as many near-uniform bits as output. That is, we want to minimize ℓ and maximize m. The limits we can do this are described by the following bounds, which can be proved for any possible extractor.

- $\ell \geq \log n + 2 \log \frac{1}{\epsilon} - O(1)$

- $m \leq k + \ell - 2\log\frac{1}{\epsilon} + O(1)$

We think of these bounds intuitively as: we need at least enough perfect random bits to specify an index into the weak random source, and we can extract out at most as many random bits contained in the combination of the weak source and perfect random source. It can be shown that picking a function at random with $\ell = \log n + 2\log\frac{1}{\epsilon} + O(1)$ and $m = k + \ell - 2\log\frac{1}{\epsilon} - O(1)$ with high probability satisfies Definition 3. However, this does not help us in the application we are interested in as the act of picking an extractor at random requires a large amount of perfect randomness. For our application, we want to develop extractors that are computable in deterministic polynomial time.

Exercise 2. *Show that if there is a combination of random source X and function E satisfying (1), then the source must be $O(\epsilon)$ close to a source with min-entropy $H_\infty(X) \geq m - \ell$.*

This fact justifies our choice of H_∞ as the notion of weak randomness.

4 Applications

We give two applications of extractors: to achieve our original goal of simulating randomized algorithms with weak random sources, and to give another alternate proof that BPP $\subseteq \Sigma_2^p$. In each application, we eliminate the need for the perfect randomness by using an extractor where $\ell = O(\log n)$ and cycling over all possible seeds. There are other areas where this is not feasible. For example, in many cryptographic settings, we do not have this luxury. There do exist extractors, called seedless extractors, that can be used in these settings. For these, the extractor takes input from two independent weak random sources and outputs a distribution close to uniform. We do not discuss seedless extractors but only mention their existence.

4.1 Simulating Randomized Algorithms

If we had an extractor that did not have the second input from a perfect random source, simulating a randomized algorithm with the extractor would be trivial; as we have shown, however, such an extractor does not exist. So instead, we describe a simulation that removes the need for the perfect random source while giving a simulation that is correct with high probability. The main idea is to choose a sample from the weak source and run the extractor with this sample on all possible strings of the truly random input. We then test the algorithm with the each output of the extractor, and take the majority vote.

Suppose we have a randomized algorithm M that needs m random bits and an extractor $E : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^m$. Given an input z, we simulate $M(z)$ as follows:

(1) Set $count = 0$.
(2) Let x be a sample from a random source X on $\{0,1\}^n$.
(3) **foreach** $y \in \{0,1\}^\ell$
(4) Let $\rho_y = E(x,y)$.
(5) **if** $M(z,\rho_y) = 1$ **then** $count = count + 1$
(6) **if** $count \geq 2^\ell/2$ **then** Output 1
(7) **else** Output 0

Let us consider the probability this simulation errs. Let B_z be the bad set for z on algorithm M, i.e., $B_z = \{\rho | M(z,\rho) \neq \text{maj}_r(M(z,r))\}$. Then the bad set for our simulation, given we have chosen the fixed source x, is

$$B'_z = \{x | \Pr_y[E(x,y) \in B_z] \geq 1/2\}. \tag{3}$$

Claim 1. *If E is a $(k, 1/6)$ extractor, then $|B'_z| < 2^k$.*

Proof. Suppose $|B'_z| \geq 2^k$, and let X be the flat source on B'_z. Notice that X has min-entropy at least k. Also, by our assumtion on $|B'_z|$, $\Pr[E(X, U_\ell) \in B_z] \geq 1/2$, while $\Pr[U_m \in B_z] \leq 1/3$ since M decides a language with bounded error. So we have a set B_z where the difference in probability assigned between the extractor and uniform is at least $1/6$, contradicting E being a $(k, 1/6)$ extractor via (2). □

Given this claim, we compute the probability our simulation errs (because of our choice of x), assuming that E is a $(k, 1/6)$ extractor:

$$\Pr[\text{Simulation errs}] = \Pr_{x \leftarrow X}[x \in B'_z] \leq |B'_z| \cdot 2^{-H_\infty(X)} < 2^{k - H_\infty(X)}.$$

If we use a source X with $H_\infty(X)$ slightly larger than k, this probability will be at most $1/3$ (for example, $H_\infty(X) \geq k + 2$ suffices).

Now consider the efficiency of the simulation. We hope that the simulation incurs only a $\text{poly}(m)$ factor overhead in time so that simulating a polynomial time algorithm still takes polynomial time. The time to complete the simulation is the time to compute E, plus the product of 2^ℓ and the time of the original algorithm. Given a $\text{poly}(n)$ computable extractor, the time to compute E is $\text{poly}(m)$ if $H_\infty(X) \geq n^{\Omega(1)}$ because then $n = \text{poly}(m)$. The 2^ℓ term is $\text{poly}(m)$ if $\ell = O(\log m)$, or equivalently $\ell = O(\log n)$ since $n = \text{poly}(m)$. Thus in fact the overhead is only polynomial in the number of random bits m the algorithm requires.

Recall that the min-entropy condition on X is precisely what we stated earlier as the requirement to be a weak random source. So, we see that our choice of weak random sources corresponds precisely with the random sources for which this analysis yields a correct simulation of M in polynomial time. We also remark that extractors with the parameters stated here do exist, and we demonstrate these later in the lecture.

4.2 Alternate Proof of BPP $\subseteq \Sigma_2^p$

For this application, we assume the existence of a $(n/2, 1/6)$ extractor $E : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^m$ computable in polynomial time and with $\ell = O(\log n)$. The existence of such an extractor is proven in the next section.

Given a BPP machine M requiring m random bits and input z, we wish to give a Σ_2^p formula equivalent to the acceptance of $M(z)$. We start by considering the simulation given in the previous section using E on a perfectly random source (one with $H_\infty(X) = n$). We view a sample x from this source as two components of equal length: $x = (x_1, x_2)$ where $|x_1| = |x_2| = n/2$. The number of x on which the simulation fails on a sample from X is $< 2^{n/2}$ by Claim 1. By a counting argument, there is a choice of x_1 so that the simulation when given (x_1, x_2) results in the correct answer for all x_2. Stated formally, for an input z, we have the following

$$z \in L(M) \Rightarrow \exists x_1 \forall x_2 (\Pr_y[M(z; E(x_1, x_2, y)) = 1] \geq 1/2).$$

Because $|y| = O(\log n)$ and assuming $m = n^{\Omega(1)}$, the inside predicate is computable in polynomial time. If we can show that $z \notin L(M)$ implies the negation of the RHS, we will be done, for then the above implication is in fact an equivalence, and so the Σ_2^p predicate does exactly decide the language of M. With this goal in mind, we switch the roles of x_1 and x_2, and note that the simulation outputs 0 only when the appropriate probability is less than 1/2, and get the following:

$$\begin{aligned} z \notin L(M) \Rightarrow & \quad \exists x_2 \forall x_1 (\Pr_y[M(z; E(x_1, x_2, y)) = 1] < 1/2) \\ \Rightarrow & \quad \forall x_1 \exists x_2 \neg (\Pr_y[M(z; E(x_1, x_2, y)) = 1] \geq 1/2). \end{aligned}$$

The first line implies the second because $\exists x \forall y$ always implies $\forall y \exists x$ and $(\Pr_y[M(z; E(x_1, x_2, y)) = 1] < 1/2) = \neg(\Pr_y[M(z; E(x_1, x_2, y)) = 1] \geq 1/2)$.

5 Constructions

We give three different constructions of polynomial time computable extractors.

1. The first construction is based on the construction of pseudorandom generators secure against circuits that use functions with high circuit complexity. This construction gives $\ell = O(\log n)$ and $m = k^{\Omega(1)}$ for sources with $k = n^{\Omega(1)}$. Note that these parameters are good enough for the applications of the previous section, but the dependence of the number of output bits to the entropy of the input distribution is still far from optimal.

2. The second construction is based on a random walk on an expander graph. For any constant $\alpha > 0$, this construction gives a $\delta > 0$ and an extractor such that for sources with entropy $k = (1 - \delta)n$, $m = (1 - \alpha)k$, and $\ell = O(\log n)$. This construction is within constant factors of being optimal but only works for sources with very high min-entropy.

3. The final construction is based on polynomials over a finite field and uses the second construction as a building block. The parameters achieved are the same as the second construction, but it works for any value of k.

The first two we give today, and for the third we refer the reader to the notes from the previous run of this course.

5.1 Hardness based Extractor

The basic idea is to take a weak random source and view it as the truth table of a function. Most functions have high circuit complexity and so can be used as the hard function to the pseudorandom generator based on hard functions. We then use the perfect random seed as the seed for this generator. We now elaborate these ideas and derive the extractor.

Recall the pseudorandom generator that we gave in a previous lecture that is secure against circuits; that is, the generator is indistinguishable to circuits of size n. Given a function $f : \{0,1\}^{\Theta(\log r)} \to \{0,1\}$ with non-uniform circuit complexity $C_f \geq r^{\Omega(1)}$, we were able to construct a generator $G_f : \{0,1\}^{\Theta(\log r)} \to \{0,1\}^r$ such that for all circuits D of size at most r,

$$|\Pr_\sigma[D(G_f(\sigma)) = 1] - \Pr_\rho[D(\rho) = 1]| \leq 1/r.$$

An examination of the proof reveals that the proof relativizes, so that if the function has circuit complexity $C_f^A \geq r^{\Omega(1)}$ even for circuits that are given an A oracle (i.e., circuits may have gates

that query A), then the output is indistinguishable even for circuits D^A with oracle access to A. In particular, the output of the generator given such a hard function is indistinguishable to the oracle circuit that just queries the oracle[1]. That is, if f has $C_f^A \geq r^{\Omega(1)}$, then

$$|\Pr_{\sigma}[A(G_f(\sigma)) = 1] - \Pr_{\rho}[A(\rho) = 1]| \leq 1/r. \tag{4}$$

This already looks similar to the condition of (2) for G_f being an extractor. This inequality (4) already makes use of a short random seed, but only works for a fixed f. So to view G_f as an extractor, we use the weak random source to pick a function f at random, allowing the sample to specify the truth table of f. Since f is a function on $O(\log r)$ bits, this can be specified in $r^{O(1)}$ bits.

Claim 2. *For any fixed oracle A, most functions $f : \{0,1\}^{\Theta(\log r)} \to \{0,1\}$ have $C_f^A \geq r^{\Omega(1)}$.*

We do not prove this claim or quantify what "most" means.

We are now ready to define the extractor. We view x as the truth table of a function f from $\Theta(\log r)$ bits to 1 bit, and view y as the seed σ for the pseudorandom generator G_f. Then

$$E(x, y) = G_f(\sigma).$$

Now consider the difference in probability assigned to a set A by this extractor and the uniform distribution. By (4) and the fact that x comes from a weak random source with min-entropy at least k, we split the difference in probability assigned to A based on whether or not f has high circuit complexity, to get

$$\begin{aligned} |\Pr[E(x, y) \in A] - \Pr[U_r \in A]| &\leq \tfrac{1}{r} \cdot (\Pr_{x \leftarrow X}[C_f^A \text{ large}]) + 1 \cdot (\Pr_{x \leftarrow X}[C_f^A \text{ small}]) \\ &\leq \tfrac{1}{r} + 1 \cdot 2^{-k} \cdot (\#\text{ of } f \text{ with } C_f^A \text{ small}). \end{aligned}$$

The term (# of f with C_f^A small) can be quantified, but intuitively can be bounded by considering the number of small circuits. We want $C_f^A = r^{\Omega(1)}$. The number of circuits of size $r^{o(1)}$ is $2^{r^{o(1)}}$, so the number of f with small circuit complexity is bounded by this value. This means we only need to have $k = r^{\Omega(1)}$ to ensure the above probability is less than some given constant ϵ, showing that $E(x, y)$ is a (k, ϵ) extractor.

5.2 Expander based Extractor

For this construction, we view the sample from the weak random source as describing a random walk on a d-regular expander graph. Let x be a sample from source X. We view the first portion as indicating an initial vertex x_0 within an expander, and the remaining part of x specifies a path to follow in the graph from this vertex. The perfect random seed y specifies a vertex along this path to output. Therefore, the first m bits of x specify x_0 and the next $t \log d$ specify a path of length t starting from x_0. To index a vertex along this path, $|y| = \log t$. To summarize, we let

$$E(x, y) = y^{th} \text{ point on random walk specified by } x.$$

[1]Note that for an oracle circuit, the size of a query to the oracle is charged as part of the size of the circuit. Otherwise, even very large queries to the oracle would count as only one gate, making even "constant sized" circuits powerful.

Before proving this is an extractor, let us specify the relationship of the parameters. By definition, $n \geq m + t \log d$. We said previously that the construction should work for $m = (1-\alpha)k$ for any $\alpha > 0$, and we can pick a δ to set $k = (1-\delta)n$. Then for the δ we pick, we will have $n \geq (1-\alpha)(1-\delta)n + t \log d$. Setting $t = \frac{\alpha n}{\log d}$, we get that $\ell = |y| = \log(t) = O(\log n)$.

We must show that we can pick a δ so that (2) is satisfied. Let $A \subseteq \{0,1\}^m$. We use the Chernoff Bound for random walks on expanders to bound the difference in probability assigned to A by $E(x,y)$ and U_m. Recall that the Chernoff Bound for random walks states that if ρ_i is the i^{th} vertex visited on a random walk in an expander with second largest eigenvalue λ, then

$$\Pr_{x \leftarrow U_n}[\underbrace{|\frac{1}{t}\sum_{i}^{t} \chi[\rho_i \in A] - \mu(A)| \geq \epsilon}_{(*)}] \leq \underbrace{\exp(-b(1-\lambda)\epsilon^2 t)}_{(**)}. \tag{5}$$

Splitting the difference in probability assigned to A based off whether (*) holds for a particular x and assuming a weak random source with min-entropy k, we get

$$\leq \underbrace{|\Pr_{x \leftarrow X, y \leftarrow U_\ell}[E(x,y) \in A] - \Pr_{\rho \leftarrow U_m}[\rho \in A]|}_{\text{contribution of good } x \text{ breaking (*)}} + 1 \cdot (\#\ x \text{ satisfying (*)}) \cdot 2^{-k}.$$

To finish, we bound $(\#\ x \text{ satisfying (*)}) \cdot 2^{-k}$:

$$(\#\ x \text{ satisfying (*)}) \cdot 2^{-k} = (2^n \cdot (**)) \cdot 2^{-k} = 2^{\delta n} \cdot (**) = 2^{\delta n} 2^{-\beta n}$$

for some constant β depending on (**). If we pick δ sufficiently smaller than β, we get that $E(x,y)$ is a $(k, 2\epsilon)$ extractor.