CS 710: Complexity Theory 11/22/2011

Lecture 22: Randomness Extraction

Instructor: Dieter van Melkebeek Scribe: Prathmesh Prabhu

In the previous lecture, we saw a pseudorandom generator that fools time-bounded computations under the plausible hypothesis that there exists a language in E requiring linear exponential size circuits. We left off by proving the result under a somewhat stronger hypothesis that there is a language in E with high average-case hardness. In the first part of this lecture, we wrap up this discussion by showing that worst-case hardness at length m, $C_L(m)$, can be substituted for average-case hardness, $H_L(m)$. Towards this, we make use of Error Correcting Codes described in previous lectures.

In the latter part of this lecture, we turn to another major pursuit in the study of randomness and psuedorandomness. Most randomized algorithms assume access to a perfect source of unbiased and, more importantly, uncorrelated (independent) random bits. But in practice, we must run these algorithms given access to only 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.

1 Worst-Case to Average-Case reduction

In the previous lecture, we enumerated some properties needed in an ECC so that it can be employed for this reduction. Further, we hinted at a construction of a local list decoder with these properties by concatenating the Hadamard decoder and the decoder for Reed-Muller code. The following theorem, stated without proof, asserts the existence of the required code.

Theorem 1. *Given length of the information word K, and $\epsilon > 0$, we can construct an error correcting code $E\{0,1\}^K \to \{0,1\}^N$ such that*

1. *E is computable in time poly$(K, \frac{1}{\epsilon})$.*

2. *<u>Local list decodability:</u> There exists a randomized oracle Turing Machine M such that M^r runs in time poly$(\log(K), \frac{1}{\epsilon})$ and outputs a list of randomized oracle TMs $D_1, D_2, \ldots$, such that for all received words r and information words x, if r agrees with the encoding of x in at least $\frac{1}{2} + \epsilon$ positions, then there is some machine D_i that computes x. Formally,*

$$\forall x \in \{0,1\}^N \quad \left[\Delta(E(x), r) \leq \frac{1}{2} + \epsilon\right] \Rightarrow \exists i \quad \forall j \in N \quad D_i^r(j) = x_j$$

where Δ is the hamming distance between the two words. Each of the machines D_i also run in time poly$(\log(K), \frac{1}{\epsilon})$.

This Error Correcting Code can now be used to obtain worst-case to average-case reductions.

Theorem 2. *For every $L \in \mathrm{E}$ there exists a language $L' \in \mathrm{E}$ such that*

$$H_{L'}(m) \geq \frac{C_L(m)^{\Omega(1)}}{m^{O(1)}}.$$

Proof. Let L be a language in E. Let $\chi_{L|_m}$ be the characteristic function of L. We construct an ECC guaranteed by Theorem 1 with the following parameters: $K = 2^m$; $\epsilon = \frac{1}{H_L(m)}$; and $N = 2^{m'}$ where $m' = O(m)$. We denote the obtained string of length N by $\chi_{L|_{m'}}$. We can solve L' in E by taking an input of length m', computing the corresponding length $m < m'$, explicitly writing out $\chi_{L|_m}$, encoding it, and extracting the bit corresponding to our input.

Condition (1) from Theorem 1 guarantees that if L is in E, then L' is also in E. Writing out $\chi_{L|_m}$ takes time $2^{O(m')}$ because there are 2^m positions, each of which can be computed in $2^{O(m)}$ time since $L \in \mathrm{E}$. Encoding $\chi_{L|_m}$ also takes $2^{O(m')}$ time. The ECC from Theorem 1 encodes in time $poly(K, \frac{1}{\epsilon})$. Here, $K = 2^m$ and $\frac{1}{\epsilon} = H_L(m) \leq 2^m$ where the last equality follows because any boolean function on m inputs can be computed with a circuit of size 2^m. Therefore, there is only a linear exponential time overhead for solving L' over L.

We will next show that

$$H_{L'}(m') \geq \frac{C_L(m)^{\Omega(1)}}{m^{O(1)}},$$

and the theorem will follow from the fact that $m' = O(m)$ (and the fact that $C_L(m)$ is at most exponential, and so changing the input length by a constant factor only makes $C_L(m)$ change by a polynomial factor).

We know that there exists a circuit C' of size $H_{L'}(m')$ such that

$$\Delta(C', \chi_{L'}(m')) \leq \frac{1}{2} + \frac{1}{H_{L'}(m')}$$

(Note: The definition of average case hardness says that there are no circuits for larger values of $H_{L'}(m')$ that satisfy this inequality. Here, we assume that some circuit exists at the minimum value as well.)

We can now construct a circuit C for L using item (2) of Theorem 1. Using the random oracle machine M^r with the characteristic string of the circuit C' as the oracle, we obtain a list of oracle machines such that at least one of the machines produces the characteristic function of L given access to C' as the oracle, i.e.,

$$\forall j \quad D_i(j) = (\chi_L(m))_j$$

We can obtain a circuit C corresponding to this randomized oracle machine D_i. We can do this by employing error reduction techniques to amplify the probability of correct results from D_i and then encoding in C a random bit sequence on which D_i gives the correct answer on all inputs. Moreover, since D_i is generated by a uniform algorithm running in time poly in $\log(k)$ and $\frac{1}{\epsilon}$ (and hence the description of D_i is of the same complexity) and C' is employed to answer oracle queries in C.

$$size(C) = poly(\log(k), \frac{1}{\epsilon}) \cdot |C'|$$

Therefore,

$$\begin{aligned} C_L(m) &\leq poly(m \cdot \frac{1}{\epsilon})^c H_{L'}(m') \\ &\leq poly(m \cdot H_{L'}(m'))^c H_{L'}(m') \\ &\leq poly(m \cdot H_{L'}(m'))^c \end{aligned}$$

for some constant c.

$$H_{L'}(m') \geq \frac{(C_L(m))^{\frac{1}{c}}}{m}$$

□

This worst-case to average-case reduction immediately implies the following corollaries.

Corollary 1. *If there exists a language $L \in \mathrm{E}$ with $C_L(m) \geq m^{\omega(1)}$ then there exists a language $L' \in \mathrm{E}$ with $H_{L'}(m) \geq m^{\omega(1)}$ and thus there exists a quick PRG with subpolynomial seed length, implying that* BPP $\subseteq$ SUBEXP.

Corollary 2. *If there exists a language $L \in \mathrm{E}$ with $C_L(m) \geq 2^{m^{\Omega(1)}}$ then there exists a language $L' \in \mathrm{E}$ with $H_{L'}(m) \geq 2^{m^{\Omega(1)}}$ and thus there exists a quick PRG with polylogarithmic seed length, implying that* BPP $\subseteq$ DTIME$(n^{\log^{O(1)} n})$.

Corollary 3. *If there exists a language $L \in \mathrm{E}$ with $C_L(m) \geq 2^{\Omega(m)}$ then there exists a language $L' \in \mathrm{E}$ with $H_{L'}(m) \geq 2^{\Omega(m)}$. and thus there exists a quick PRG with logarithmic seed length, implying that* BPP $=$ P.

On the other hand, the existence of similar reductions for smaller classes is still an open problem. We saw that there is a linear exponential overhead in the scheme employed in the proof of Theorem 2. Hence, the same proof does not work when L is in, say, NP. For instance, suppose that it is known that SAT needs large circuits in the worst case, it is not known whether it implies that there is a language in NP that needs large circuits in the average case. Such an implication would have interesting ramifications for cryptography, because then, P $\neq$ NP would imply the average case hardness needed in cryptographic algorithms. Currently, stronger assumptions are needed for many results in cryptography.

2 Randomness Extraction

We have seen 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 a plausible hypothesis. Indeed, 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 that randomness is useless in practice. On the contrary, a quadratic speedup achieved with randomness may be very attractive in practice. 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 uncorellated random bits. But physical sources of randomness, such as keystrokes or delays, are far from perfect. These sources are formally modeled by the notion of weak sources. The goal of randomness extraction can be stated as – "Given a sequence of bits from a weak source of randomness, extract a shorter, but almost uniformly random, sequence."

An *extractor* is an efficient 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 only "close" to uniform will have only a small effect on the output distribution of the randomized algorithm.

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 *of a discrete random variable X is*

$$H(X) = E\left[\log \frac{1}{\Pr[X=x]}\right] = \sum_x \Pr[X=x] \cdot \log \frac{1}{\Pr[X=x]}$$

where the sum is over the range of X.

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

However, This measure of randomness captures the average case behaviour of the source, and 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: $log\frac{1}{\Pr[X=0^m]} = 1$, while for nonzero strings x, we have $log\frac{1}{\Pr[X=x]} \simeq m-1$, 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 that focuses on the worst case behaviour of the source over all output strings.

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

$$H_\infty(X) = \min_x \log \frac{1}{\Pr[X=x]}$$

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.

With this measure of randomness in place, we are in a position to define the notion of weak sources of randomness.

2.1 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 $n^{\Omega(1)}$ 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 $H_\infty(X) \geq n^{\Omega(1)}$. If the entropy were exactly r, where r is the length of the bit sequence obtained, the source would be perfectly uniform. Weak sources can have slightly lower min-entropy, say a constant fraction of r.

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. The min-entropy for this source is k.

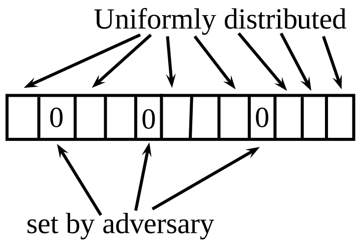

Figure 1: A bit fixing source

- *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. The min-entropy for this source, $H_\infty = O\left(\log\left[\frac{1}{(1/2+\delta)^r}\right]\right) = O(r)$. Therefore, the min-entropy for this source is also linear in r for small δ.

- *Flat sources.* These sources are a generalization of Bit fixing sources. The output is a 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 $|S| = 2^{n^{\Omega(1)}}$. These flat sources are generic sources to look at when we want to model a weak random source with a given min-entropy k. This is the case because all weak sources of the given min-entropy can be obtained as a convex combination of flat sources.

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

2.2 Randomness Extractors

Our goal is to construct extractors that can extract the maximum amount of randomness from a given source with min entropy at least k. Although the distribution of the extracted strings can not be expected to be perfectly uniform, we want it to be statistically close to uniform. Also, we want to extract a large number of random bits from the source string. The maximum number of bits that can be extracted equals the min-entropy, because that captures the total amount of randomness in the source sequence. Finally, we would hope that the extractor is a deterministic procedure that outputs the nearly uniformly distributed strings given input from the weak source. This last requirement is too strong to be feasible. Indeed it is impossible to extract even a single truly uniform bit deterministically.

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 with seed length l if for all X on $\{0,1\}^n$ with $H_\infty(X) \geq k$,*

$$\parallel E(X, U_\ell) - U_m \parallel_1 < 2\epsilon \tag{1}$$

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 $A \subseteq \{0,1\}^m$,

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

Note that this definition of an extractor places a very strong requirement that the extractor algorithm, which is independent of the weak source of randomness under consideration, should work for all sources with the stated min-entropy.

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 additional 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 within which we can do this are described by the following bounds, which we state without proof.

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

The intuition behind the first bound is that we need at least enough perfect random bits to specify an index into the weak random source. The lower bound on l once again implies that it is not possible to have a deterministic extractor with the given properties. Indeed, the proof of this bound is a generalization of the construction we saw earlier as an argument for the need of a seed. The second bound intuitively means that we can extract out at most as many random bits as contained in the combination of the weak source and perfect random source; And as ϵ decreases, the output distribution gets statistically closer to a uniform source, at the cost of a reduction in the number of bits extracted.

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. We will see some constructions in the next lecture. In the rest of this lecture, we discuss a couple applications of extractors.

2.3 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.

2.3.1 Simulating Randomized Algorithms

We always assumed that a source of perfectly random sequences was available in our discussion of randomized algorithms. We now show how extractors can be used to run these algorithms using weak random sources. If we had an extractor that did not need the seed of length ℓ 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 r random bits. Because our simulation should run in polynomial time, we can use only a polynomial number of bits from the weak random source in the simulation. This means that we should be able to extract r truly random bits from $poly(r)$ bits from the weak random source. This is only possible if the min-entropy of the source, $H_\infty \geq n^\gamma, \gamma > 0$. This bounds the minimum randomness needed in the source for this simulation to work. For instance, if the source had $H_\infty = o(1)$, the time needed to extract r bits would be super-polynomial in r, and the simulation would not run in polynomial time. We next describe a simulation for which a source with $H_\infty = n^\gamma, \gamma > 0$ suffices.

Given a randomized algorithm M that needs r random bits and an extractor $E : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^r$. 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 that this simulation errs. Let B_z be the bad set for z on algorithm M,

$$\begin{aligned} B_z &= \{\rho | M(z, \rho) \text{ errs}\} \\ &= \{\rho | M(z, \rho) \neq \mathrm{maj}_r(M(z, r))\} \end{aligned}$$

Then the bad set for our simulation, given we have chosen the fixed source x, is

$$\begin{aligned} B'_z &= \{x | \mathrm{maj}_y(M(z, E(x, y)) \text{ errs}\} \\ &= \{x | \Pr_y[E(x, y) \in B_z] \geq 1/2\} \end{aligned}$$

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. □

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. 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 poly(n) computable extractor, the time to compute E is poly(m) because $n =$ poly(m). The 2^ℓ term is poly(m) if $\ell = O(\log m)$, or equivalently $\ell = O(\log n)$ since $n =$ poly(m). Thus, the overhead is only polynomial in the number of random bits m the algorithm requires.

2.3.2 Alternate Proof of $\mathrm{BPP} \subseteq \Sigma_2^p$

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 $(\Pr_y[M(z; E(x_1, x_2, y)) = 1] < 1/2) = \neg(\Pr_y[M(z; E(x_1, x_2, y)) = 1] \geq 1/2)$.

Acknowledgements

In writing the notes for this lecture, I perused the notes by Matt Elder and Beth Skubak from the Spring 2007 offering of CS 810, and the notes by Brian Rice and Jeff Kinne for lectures 19 and 21 from the Spring 2010 offering of CS 710.