0368.4163 Randomness and Computation March 11, 2009

Lecture 2

Lecturer: Ronitt Rubinfeld *Scribe: Meir-Jonathan Dahan, Tomer Levinboim*

Today

- Randomness Sources, Deterministic Randomness Extractors
- Seed-Extractors
- Derandomization

1 Preliminaries

Definition: Recall that a *language* L is any subset of $\{0,1\}^*$.
Examples:

- L={ x | x is a set with a "good" 2-coloring } (see last proof in Lecture 1)
- L={ x | x is a description of a graph that contains a Hamilton cycle }

Definition: P (*Polynomial Time*) = the class of languages L for which there exists a deterministic poly-time algorithm A s.t.:

$$x \in L \Rightarrow A(x) = 1$$

$$x \notin L \Rightarrow A(x) = 0$$

Definition: RP (*Randomized Polynomial Time*) = the class of languages L for which there exists a probabilistic poly-time algorithm A s.t.:

$$x \in L \Rightarrow Pr[A(x) = 1] \geq \frac{1}{2}$$

$$x \notin L \Rightarrow Pr[A(x) = 1] = 0$$

In other words, RP is the class of languages for which membership can be decided by a probabilistic poly-time (bounded) one-sided-error TM, where the completeness is at least $\frac{1}{2}$ (we might reject $x \in L$), and we have full soundness (always reject $x \notin L$). It can easily be shown that if we perform k independent repetitions of such an algorithm A, we reduce the error rate (of rejecting $x \in L$) to 2^{-k}. This is also known as *amplification.*

Definition: BPP (*Bounded Probabilistic Polynomial Time*) = the class of languages L for which there exists a probabilistic poly-time algorithm A s.t.:

$$x \in L \Rightarrow Pr[A(x) = 1] \geq \frac{2}{3}$$

$$x \notin L \Rightarrow Pr[A(x) = 1] \leq \frac{1}{3}$$

In other words, BPP is the class of languages for which membership can be decided by a probabilistic poly-time (bounded) two-sided-error TM.
BPP can also be amplified, but this time we need to take the majority vote of the independent repetitions:

Theorem 1 [Amplification of BPP]: For any constant $\beta > 0$, at most $O(log\frac{1}{\beta})$ independent repetitions are needed to decreases the error rate to β.

Before we proceed with the proof, we recall *Chernoff's bound* which roughly says that, the probability of a sum of independent variables will deviate from its expected value μ, decreases exponentially in the shift from μ**.**

Chernoff's Bound: Let $X_1, X_2...X_k \in \{0, 1\}$ i.i.d (independently, identically distributed) random variables with $Pr(X_i = 1) = p$. Let $X = \sum_{i=1}^{k} X_i$ and $\mu = E[X]$. Then $\forall \delta \in [0, 1]$ the following inequality holds:

$$Pr[X \geq (1 + \delta)\mu] \leq e^{-\delta^2 \mu/4}$$

Chernoff's bound has many variations, but this form will fit our needs.

Proof [Theorem 1]: Let A be a probabilistic polytime TM that decides $L \in BPP$. We construct A', which runs A exactly k times (where the runs are independent) and takes the majority vote as its answer. We now analyze the probability A' errs.
Let X_i denote an indicator variable which is 1 if on the i'th iteration of A', A accepted, and otherwise, 0 (There are k such indicators), and let $X = \sum_{i=1}^{k} X_i$.
Clearly (by linearity of expectation and the definition of BPP): (1) if $x \in L$ then $E[X] \geq \frac{2k}{3}$ and (2) if $x \notin L$ then $E[X] \leq \frac{k}{3}$.
Suppose $x \notin L$, then A' returns a wrong answer in the event that the majority over the k answers of A is 1. This happens when $X > \frac{k}{2}$.
Therefore, If $X \notin L$ then Pr[A' fails] $= Pr[X > \frac{k}{2}] = Pr[X > (1 + .5)\frac{k}{3}] \leq Pr[X > (1 + .5)E[X]] \leq e^{-\frac{1/2^2 E[X]}{4}}$
Where the last inequality is due to Chernoff. If we wish this term to be at most β we obtain:

$$e^{-\frac{1/2^2 EX}{4}} \leq \beta \Longrightarrow E[X] \geq 16ln\frac{1}{\beta}$$

But, when $x \notin L$ we have $E[X] \leq \frac{k}{3}$, therefore:

$$\frac{k}{3} \geq 16ln\frac{1}{\beta} \Longrightarrow k \geq 48ln\frac{1}{\beta}$$

Therefore, when $x \notin L$, taking at least $48ln\frac{1}{\beta}$ suffices to decrease the error rate to at most β. We omit the analysis for the case where $x \in L$, as it is exactly

the same, with respect to the random variable $(n-X)$. This concludes the proof.

Concluding this section, we state that it is easy to show (i.e., follows from the definitions) that:

$$P \subseteq RP \subseteq BPP$$

A central question in complexity theory that remains unanswered is whether $P = BPP$ (although many believe this to be true).

2 Randomness Sources and Deterministic Randomness Extractors

Real-life computers have a built-in "random number generators", which usually work by some combination of sampling the internal clock, user input (e.g., interval between keystrokes), network traffic etc. These sources of randomness cannot be considered truly random and are usually called *pseudo-random*. A well known approach to get a somewhat random source is to digitize the noise produced by a Zener diode, another might be to sample cloud formations (not so good on a sunny day), or even a lava-lamp (see www.lavarnd.org), however, these sources are considered as *weak random sources* - that is, not truly random. In spite of that, weak random sources may not be completely useless - given a weak random source of bits, it might be possible to extract truly random bits out of it. This is exactly the idea behind an *Extractor* - it is a function that purifies a weak source of randomness into a (nearly) perfect one.

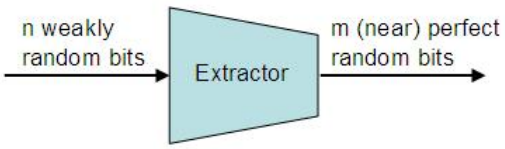

Figure 1: Deterministic Extractor

Once we have such an extractor, our requirements from the source of randomness are relaxed - it allows as to construct a probabilistic algorithm that relies on weakly random bits, instead of truly random bits. This is depicted in the following diagram:

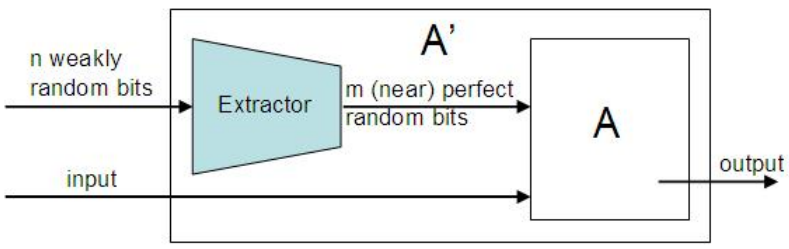

Figure 2: Extractor in use

Given input x, A' uses the extractor on the n weakly random bits, and supplies A with x and the the m (almost) truly random bits.

We shall now proceed to discuss several types of randomness sources, where our assumptions on these sources will be weakened further and further.

2.1 Von-Neumann's "Trick"

We begin with a simple example, where the source is only slightly weakened, in that, it is not unbiased.
Claim: $\forall \rho \in (0,1)$ a ρ-biased coin (i.e., $Pr[\text{"}Heads\text{"}] = \rho$, $Pr[\text{"}Tails\text{"}] = 1-\rho$) can be used to simulated a truly random coin, in expected time $O(\frac{1}{\rho(1-\rho)})$.
Proof: Consider the following algorithm A:

- Get input $X = (X_1, Y_1, X_2, Y_2...)$. (where all X_i, Y_i are independent and $\rho - biased$)
- At the i'th iteration, if $X_i \neq Y_i$
  - stop and output X_i
  - else proceed to next iteration

Notice that at each iteration $Pr[H,H] = \rho^2$, $Pr[T,T] = (1-\rho)^2$, and more importantly, $Pr[H,T] = Pr[T,H] = \rho(1-\rho)$, therefore, given that A stops, it outputs "Heads" or "Tails" with equal probability. The probability to stop at each iteration is exactly $\alpha = 2\rho(1-\rho)$, so the probability not to stop after n iterations is $(1-\alpha)^n$ which rapidly decays to 0. Moreover, we expect to stop after $\frac{1}{\alpha} = \frac{1}{2\rho(1-\rho)}$ iterations.
For example, for $\rho = \frac{1}{3}$, we have have $2\rho(1-\rho) = \frac{4}{9}$, and we therefore expect to stop after $\frac{9}{4}$ iterations on average.

2.2 δ-source

A source of bits $X = (X_1X_2, X_3...X_n)$ where each X_i is an independent random variable over $\{0,1\}$ is called a δ-*source* when

$$\forall i: \; Pr[X_i = 1] = \delta_i$$

Where for each δ_i, $0 < \delta \leq \delta_i \leq 1 - \delta < 1$ for some constant δ.
Compared to the Von-Neumann model, the bits are still chosen independently, however might not be identically distributed.
Due to the independence, we can still extract an (almost) truly random bit using the parity function. That is, we can show:

$$\left| Pr\left[\bigoplus_{i=1}^{n} X_i = 1\right] - \frac{1}{2} \right| = 2^{-\Omega(n)}$$

Which is exactly what we desire.

2.3 SV-source (Santha-Vazirani)

A sequence $X = (X_1, X_2...X_n)$ is called an an *SV-source* if:
$\forall i \in \{1..n\}$, $\forall \alpha_1, \alpha_2...\alpha_{n-1} \in \{0,1\}$ and constant δ it holds that:

$$\delta \leq Pr[X_i = 1 | X_1 = \alpha_1, X_2 = \alpha_2, ...X_{i-1} = \alpha_{i-1}] \leq 1 - \delta$$

Under these settings, the random variables can be dependent, so the parity function we previously employed will not work. In fact, the situation is much worse as it is known that for any deterministic randomness extractor there is an SV-source that foils its operation (makes it output a biased bit with high probability).
Remark: one way to overcome the difficulty of deterministically extracting randomness from SV-sources is to modify the model. It turns out that if we use a set of sources S (where $|S| > 1$) there are $\epsilon-$extractors for this set.

2.4 k-source

A source of n bits is called a $k - source$ if the probability of any output is at most 2^{-k}. Formally:

$$\forall x \in \{0,1\}^n \ : \ Pr[X = x] \leq 2^{-k}$$

Usually we consider $k \ll n$.
Examples:

- Bit Fixing sources - e.g., the first k bits of X are i.i.d with $Pr[x_i = 1] = 1/2$, $\forall i \in \{1..n\}$, and the next $(n-k)$ bits are (say) all zeros. This can be generalized to any k bit-positions, and letting the remaining $(n-k)$ bits be any fixed function of the random k bits.
- An SV-source with $k = log\frac{1}{(1-\delta)^n}$
- A "flat" source - a uniform distribution over any $S \subseteq \{0,1\}^n$ such that $|S| = 2^k$

3 Seeded Randomness Extractors

3.1 Notation

- Denote U_n as the *uniform distribution* over $\{0,1\}^n$ (that is - $\forall x \in \{0,1\}^n$ we have $Pr_{X \sim U_n}[X = x] = 2^{-n}$).
- The *total variation distance* between two probability distributions P and Q on finite domain D is

$$\delta(P,Q) = \frac{1}{2} \sum_{x \in D} |P(x) - Q(x)|$$

- P and Q are said to be ϵ-*close* if $\delta(P,Q) < \epsilon$.

3.2 Seed-Extractors

Seed-extractors are a relaxation of deterministic extractors. While the extractors discussed in the previous section were all deterministic, seeded extractors use some (hopefully small) source of randomness (a seed):

Definition: $f : \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^m$ is a (k,ϵ)-*seed-extractor* if for any k-source X on $\{0,1\}^n$, $f(X, U_d)$ is ϵ-close to U_m.
(note the slight abuse of notation, f's domain is binary strings, while X and U_n are distributions)

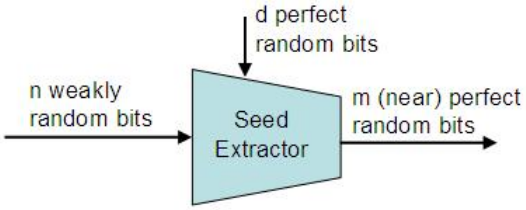

Figure 3: A seed extractor

Our goal when extracting such seeded extractors, is to maximize m (the number of truly random bits the extractor outputs) while minimizing d (length of the seed). Consider the following options:

1. $m < d$ - Useless, as we ended up with less bits than we started with.
2. $m > d+n$ - Impossible, this means we somehow conjured random bits out of thin air.
3. $m = d + k$ - The following theorem states a close result.

Theorem 2: $\forall n, k$ such that $k \leq n$ and $\forall \epsilon > 0$, there exists a (k,ϵ)-*seed-extractor* $f : \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^m$ such that:

1. $m = k + d - 2log(\epsilon^{-1}) - O(1)$,
2. $d = log(n-k) + 2log(\epsilon^{-1}) + O(1)$

A few remarks are in order:

- A constant gap ($2log(\epsilon)+O(1)$) remains between the output length m and the desired output $d + k$.
- The seed's length $d < log(n)$.
- The theorem does not provide an explicit construction of such an effective extractor (Although, current constructions come close to these parameters).
- Using such a (k,ϵ)-extractor, we can reduce our dependency on truly random bits. We can transform any probabilistic TM that depends on m truly random bits and has error rate β (say, $\frac{1}{3}$ as in BPP) to one that has error rate at most$\beta+\epsilon$, but depends only on $d = log(n)$ truly random bits (and the weak source).

4 Derandomization

Consider a probabilistic polynomial algorithm A for some language $L \in BPP$, can we derandomize it? That is, can we somehow produce a deterministic algorithm A' that always outputs the correct answer without the requirement for a random source? One naive way to achieve this is *derandomization via enumeration*, in which we simply iterate over all possible random strings A could consider, and output the majority answer. A formal presentation follows:

Let A a probabilistic polynomial time TM, and let $p(n)$ a polynomial bounding the running time of A on input x of length $|x| = n$.
It follows that A uses at most $p(n)$ random bits (otherwise its running time would not have been $p(n)$). Consider a TM A' that on input x runs as follows:

1. For each $r_i \in \{0,1\}^{p(n)}$, compute $s_i = A(x, r_i)$ - (*The Enumeration*)
2. Output the majority of all s_i $MAJ(s_1, ...s_{2^{p(n)}})$

Now, if $x \in L \Rightarrow Pr[s_i = 1] \geq 2/3$, therefore $MAJ(s_1, ...s_{2^{p(n)}}) = 1$,
Otherwise, if $x \notin L \Rightarrow Pr[s_i = 1] \leq 1/3$, therefore $MAJ(s_1, ...s_{2^{p(n)}}) = 0$.
Therefore, A' always outputs the correct answer.

Notice that the running time of A' is bounded from above by $2^{p(n)}p(n)$ which is exponential. This also implies that $BPP \subseteq EXP$ (recall that $EXP = DTIME(\bigcup_c 2^{n^c})$).

Generally, if A uses r random bits, we can construct an A' by derandomization via enumeration and "pay" up to $2^r p(n)$ in running time. Now, if $r = O(log(n))$ then the running time of A' is $2^{O(log(n))}p(n) = n^c p(n)$ for some constant $c > 0$. In this special case, derandomization via enumeration yields a deterministic polynomial time algorithm.

This last note motivates us to be as parsimonious as we can with our source of randomness. Suppose we can show our algorithm A does not need its m-random bits to be completely independent, but only *pairwise independent* (p.i. henceforth). In the lecture to follow we will show how to generate m p.i. bits given $O(log(m))$ truly random ones. Let G be a process that does exactly that (produces m p.i bits out of m truly random ones), then, derandomizing such an A can be done as follows:

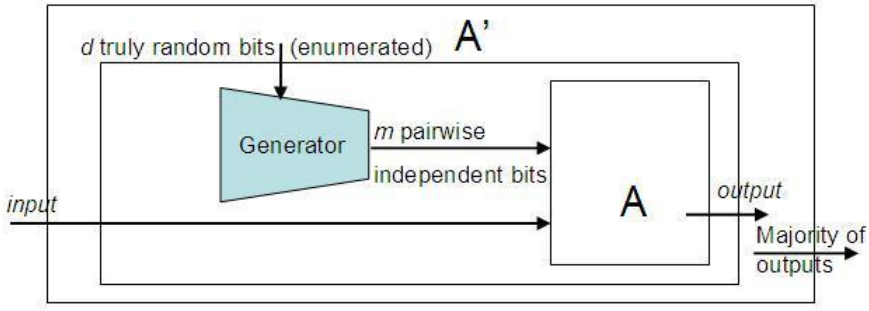

Figure 4: Derandomization using pairwise independence

1. For each $d_i \in \{0,1\}^{d=O(log(m))}$ (*The Enumeration*)
   (a) Compute $\hat{d_i} = Generator(d_i)$ ($\hat{d}$ is a sequence of m p.i bits)
   (b) Compute $s_i = A(x, \hat{d_i})$
2. Return $MAJ(s_1, ...s_{2^d})$

This yields a deterministic polytime algorithm, which always computes the correct answer (because A expects only m p.i. bits).
For further details, see lecture 3.