Great Ideas in Theoretical Computer Science **Summer 2013**

Lecture 3: Randomness in Computation

Lecturer: Kurt Mehlhorn & He Sun

Randomness is one of basic resources and appears everywhere. In computer science, we study randomness due to the following reasons:

1. Randomness is essential for computers to simulate various physical, chemical and biological phenomenon that are typically random.
2. Randomness is necessary for Cryptography, and several algorithm design techniques, e.g. sampling and property testing.
3. Even though efficient deterministic algorithms are unknown for many problems, simple, elegant and fast randomized algorithms are known. We want to know if these randomized algorithms can be derandomized without losing the efficiency.

We address these in this lecture, and answer the following questions: (1) What is randomness? (2) How can we generate randomness? (3) What is the relation between randomness and hardness and other disciplines of computer science and mathematics?

1 What is Randomness?

What are *random* binary strings? One may simply answer that every 0/1-bit appears with the same probability (50%), and hence

$$00000000010000000001000010000$$

is not random as the number of 0s is much more than the number of 1s.

How about this sequence?

$$000000000011111111111$$

The sequence above contains the same number of 0s and 1s. However most people think that it is not random, as the occurrences of 0s and 1s are in a very unbalanced way. So we roughly call a string $S \in \{0,1\}^n$ random if for every pattern $x \in \{0,1\}^k$, the numbers of the occurrences of every x of the same length are almost the same. Now we look at several definitions that formalize our intuitions.

The study of randomness started in the last century through three different theories. The first one is the information theory, initiated by Shannon. Information theory focuses on distributions that are not perfect random, and considers perfect randomness as an extreme case — the distribution with the maximum entropy.

Definition 1 (Entropy). *Let X be a random variable. Then the entropy $\mathbf{H}(X)$ of X is defined as*

$$\mathbf{H}(X) \triangleq \mathbf{E}_{x\sim X}\left[-\log\left(\mathbf{Pr}[\,X = x\,]\right)\right],$$

where $x \sim X$ means that x is sampled according to X.

Definition 2 (min-Entropy)**.** *Let X be a random variable. The min-Entropy $\mathbf{H}_\infty(X)$ of X is defined as*

$$\mathbf{H}_\infty(X) \triangleq \min_{x\sim X}\left\{-\log\left(\mathbf{Pr}[\,X=x\,]\right)\right\}.$$

We call X a k-source if $\mathbf{H}_\infty(X) \geq k$.

The second theory is due to Kolmogorov. Kolmogorov theory measures the complexity of objects in terms of the shortest program (for a fixed universal program) that generates the object. Here perfect randomness appears as an extreme case since no shorter program can generate the perfect random string. In contrast to the information theory where the entropy measures the randomness for a distribution, following the Kolmogorov theory one can talk about the randomness of a single string.

The third theory is due to Blum and Yao in 1980s, who define randomness with respect to observers (computational models) and suggest to view distributions as equal if they cannot be distinguished by efficient observers. To motivate this definition, let us look at one example.

Example. [1] *Alice and Bob play "head or tail" in one of the following four ways. In all of them Alice flips a coin high in the air, and Bob is asked to guess its outcome before the coin hits the floor. The alternative ways differ by the knowledge Bob has before making his guess. In the first alternative, Bob has to announce his guess before Alice flips the coin. Clearly, in this case Bob wins with probability $1/2$. In the second alternative, Bob has to announce his guess while the coin is spinning in the air. Although the outcome is determined in principle by the motion of the coin, Bob does not have accurate information on the motion and thus we believe that also in this case Bob wins with probability $1/2$. The third alternative is similar to the second, except that Bob has at his disposal sophisticated equipment capable of providing accurate information on the coin's motion as well as on the environment effecting the outcome. However, Bob cannot process this information in time to improve his guess. In the fourth alternative, Bob's recording equipment is directly connected to a powerful computer programmed to solve the motion equations and output a prediction. It is conceivable that in such a case Bob can improve substantially his guess of the outcome of the coin.*

This example tells us that proper definitions of randomness depend on the power of observers: One string that looks random for one observer may be not random for more powerful observers. In computer science, observers with different powers correspond to algorithms under different time/space constraints (e.g. polynomial-time). To capture this intuition, we define *computational indistinguishable* which refers to pairs of distributions which cannot be distinguished by efficient procedures (i.e. polynomial-time algorithms).

Definition 3 (probability ensembles)**.** *A probability ensemble $\mathcal{X}$ is a family $\mathcal{X} = \{X_n\}_{n\geq 1}$ such that X_n is a probability distribution on some finite domain.*

Definition 4 (Computational Indistinguishability)**.** *Let $\mathcal{D}$ and $\mathcal{E}$ be probability ensembles. The success probability of randomized algorithm A for distinguishing $\mathcal{D}$ and $\mathcal{E}$ is*

$$sp_n(A) = \left|\mathbf{Pr}[\,A(X)=1\,] - \mathbf{Pr}[\,A(Y)=1\,]\right|,$$

where X has distribution $\mathcal{D}$ and Y has distribution $\mathcal{E}$. Distributions $\mathcal{D}$ and $\mathcal{E}$ are called computationally indistinguishable if for any probabilistic polynomial-time algorithm A,

[1]This example is from "A Primer on Pseudorandom Generators" by Oded Goldreich, 2010.

for any positive polynomial $p(\cdot)$, and for all sufficiently large n's, it holds that $sp_n(A) < 1/p(n)$.

We use $\mathcal{D} \sim^c \mathcal{E}$ to express that $\mathcal{D}$ and $\mathcal{E}$ are computationally indistinguishable.

2 Extractors

Randomized algorithms have access to a sequence of random strings as part of algorithms' input. This sequence of random strings is typically generated based on physical sources or certain movements (e.g. movements of the mouse, frequency of typing the keyboard or visiting webpages) that look random and contain some randomness, but is not perfect random. This motivates the study of constructing deterministic objects which can extract almost-perfect randomness from sources of weak randomness.

Definition 5. *For random variables X and Y taking values in U, their statistical difference is*

$$\Delta(X,Y) \triangleq \max_{T\subseteq U} |\mathbf{Pr}[\, X \in T\,] - \mathbf{Pr}[\, Y \in T\,]| \,.$$

We say that X and Y are ε-close if $\Delta(X,Y) \leq \varepsilon$.

In the following, let $\mathcal{U}_n$ be the uniform distribution defined on $\{0,1\}^n$.

Definition 6. *A (k,ε)-extractor is a function $\mathsf{Ext}: \{0,1\}^n \times \{0,1\}^d \mapsto \{0,1\}^m$ such that for every k-source X on $\{0,1\}^n$ the distribution $\mathsf{Ext}(X,\mathcal{U}_d)$ is ε-close to $\mathcal{U}_m$.*

We refer to the ratio k/n as *the entropy rate* of the source X and to the ratio m/k as *the fraction of randomness* extracted by Ext. The goal of constructing good extractors is to minimize d and maximize m.

Why do we need extractors? We will show that, once we have certain extractors, random strings with high-entropy suffice for most randomized algorithms. Moreover, the construction of extractors is independent with particularly randomized algorithms and algorithms can use extractors as a black-box. As an example, we assume that algorithm A uses m random bits. Since we do not know how to obtain truly random bits, algorithm A can only obtain *weak random sources X*. So instead of using this weak random source X directly, we use $\mathsf{Ext}(X,\mathcal{U}_d)$ to purify the randomness in X, and let $\mathsf{Ext}(X,\mathcal{U}_d)$ be the actual randomness used by A. Since d, the length of the truly random strings used in Ext, is typically small, we can eliminate the randomness by running all possible seeds and taking the majority value.

Lemma 7. *Let $A(w;r)$ be a randomized algorithm such that $A(w;\mathcal{U}_m)$ has error probability at most γ, and let $\mathsf{Ext}: \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a (k,ε)-extractor. Define*

$$A' \triangleq \mathrm{maj}_{y\in\{0,1\}^d} \left\{A(w, \mathsf{Ext}(x,y))\right\}.$$

Then for every k-source X on $\{0,1\}^n$, $A'(w;X)$ has error probability at most $2(\gamma+\varepsilon)$.

Now we look at a simple construction of extractors.

Definition 8 (pairwise independent hash functions)**.** *A family of pairwise hash functions is a set of functions $h: D \mapsto R$ such that for any distinct $x_1, x_2 \in D$ and all (not necessarily distinct) $y_1, y_2 \in R$, it holds that*

$$\mathbf{Pr}[\, h(x_1) = y_1 \wedge h(x_2) = y_2\,] = \frac{1}{|R|^2}.$$

Theorem 9 (Leftover Hash Lemma)**.** *If $\mathcal{H} = \{h : \{0,1\}^n \to \{0,1\}^m\}$ is a family of pairwise independent hash functions where $m = k - 2\log(1/\varepsilon)$. Then $\mathsf{Ext}(x,h) = (h, h(x))$ is a (k,ε)-extractor.*

Although the study of extractors is motivated by simulating randomized algorithms with weak random sources, recent studies show that extractors play a central role in studying various combinatorial objects, e.g. error-correcting codes and expander graphs. See [3] for extensive discussion.

3 Pseudorandom Generators

In 1982, Blum and Micali introduced the idea of designing efficient algorithms such that the outputs of the algorithms pass a polynomial time test. In the same year, Yao gave the other definition of pseudorandom generators, and proved this definition is equivalent to Blum's definition.

Loosely speaking, pseudorandom generators are defined as efficient *deterministic* algorithms which stretch *short random seeds* into longer pseudorandom sequences. There are three aspects for pseudorandom generators.

- Efficiency: The generator should be efficient, i.e. the pseudorandom generators should produce pseudorandom sequences in polynomial-time. In fact, pseudorandom generators are one kind of *deterministic polynomial-time algorithms.*
- Stretching: The generator is required to stretch its input seed to a longer output sequence. Specifically, the generator stretches an n-bit input into an $\ell(n)$-bit long output, where $\ell(n) > n$. The function ℓ is called the *stretching function* of the generator.
- Pseudorandomness: The generator's output has to look random to any efficient observer. That is, any procedure should fail to distinguish the output of a generator (on a random seed) from a truly random sequence of the same length in polynomial-time. For instance, a procedure could count the number of 0s and 1s and any pseudorandom generator need output almost the same number of 0s and 1s.

Definition 10 (Pseudorandom Generators)**.** *A deterministic polynomial-time algorithm G is called a pseudorandom generator if there exists a stretching function $\ell : \mathbb{N} \mapsto \mathbb{N}$, such that the following two probability ensembles, denoted $\{\mathcal{G}_n\}_{n\in\mathbb{N}}$ and $\{\mathcal{U}_n\}_{n\in\mathbb{N}}$, are computationally indistinguishable.*

1. *Distribution $\mathcal{G}_n$ is defined as the output of G whose length is $\ell(n)$ on a uniformly selected seed in $\{0,1\}^n$.*
2. *Distribution $\mathcal{U}_{\ell(n)}$ is defined as the uniform distribution on $\{0,1\}^{\ell(n)}$, $\ell(n) > n$.*

That is, we require that for any probabilistic polynomial-time algorithm A, for any positive polynomial $p(\cdot)$, and for all sufficiently large n's, it holds that

$$\left|\mathbf{Pr}[\,A(G(\mathcal{U}_n)) = 1\,] - \mathbf{Pr}[\,A\left(\mathcal{U}_{\ell(n)}\right) = 1\,]\right| < \frac{1}{p(n)}.$$

By definition, pseudorandomness is defined in terms of its observer. It is a distribution which cannot be told apart from the uniform distribution by any polynomial-time observer. However, pseudorandom sequences may be distinguished from truly random ones by infinitely powerful observers or more powerful observers. For instance, the pseudorandom sequence that cannot be distinguished from truly random ones by any polynomial-time algorithm could be distinguished from truly random ones by an exponential-time algorithm. So pseudorandomness is subjective to the abilities of the observer.

The Formulation of Pseudorandom Generators

The formulation of pseudorandom generators consists of three aspects: (1) The stretching measure of the generators; (2) The class of distinguishers that the generators are supposed to fool, i. e. the class of algorithms that are allowed to distinguish the output of generators and the truly uniform distributions; (3) The resources that generators are allowed to use.

Pseudorandom generators and computational difficulty are strongly related. To show the computational difficulty, we introduce the notion of one-way functions. One-way functions are a family of functions which is easy to compute and hard to compute the inverted. The formal definition is as follows:

Definition 11 (One-way Functions)**.** *A function $f : \{0,1\}^* \to \{0,1\}^*$ is one-way if f satisfies the following two conditions:*

1. *There exists a polynomial-time algorithm A to compute f, i.e. $\forall x : A(x) = f(x)$.*
2. *For all probabilistic polynomial-time algorithms A', polynomials $p(\cdot)$ and sufficiently large n's, it holds that*

$$\mathbf{Pr}[\, A'(f(\mathcal{U}_n)) = f^{-1} \circ f(\mathcal{U}_n)\,] < \frac{1}{p(n)}.$$

We call the algorithm that tries to invert a one-way function or tries to distinguish the output of a pseudorandom generator from a truly random string *an adversary.*

In 1993, Håstad, Impagliazzo, Levin and Luby proved the following definitive theorem: Starting with any one-way function, one can construct a pseudorandom generator.

Theorem 12 ([1])**.** *Pseudorandom generators exist if and only if one-way functions exist.*

Proof. We only show that the existence of PRGs implies the existence of one-way functions, and the proof of the other direction is more complicated.

Let $G : \{0,1\}^n \mapsto \{0,1\}^{2n}$ be a pseudorandom generator. Consider $f(x,y) \triangleq G(x)$, where $|x| = |y|$. If you can invert f on $f(\mathcal{U}_{2n}) = G(\mathcal{U}_n)$, then you can distinguish $G(\mathcal{U}_n)$ from $\mathcal{U}_{2n}$, which contradicts to the assumption that G is a PRG. □

There are a few problems that seem to be one-way in practice and are conjectured to be one-way. The following are two typical candidates.

Problem 13 (Factoring Problem)**.** *Let $f(p,q) = pq$. Given $f(p,q)$ for randomly chosen pairs of primes (p,q), find p and q.*

Problem 14 (Discrete Logarithm Problem)**.** *Given a prime modulus p and a generator $g \in \mathbb{Z}_p^*$, for $y = g^x \bmod p$ find the index x.*

Now we prove that once we have a PRG with stretching function $\ell(n) = n+1$, then we have PRGs with arbitrary stretching functions.

Theorem 15 (amplification of stretch function)**.** *Suppose we have a pseudorandom generator G with a stretch function $\ell(n) = n+1$, then for every polynomial $\ell(n) > n$ there exists a pseudorandom generator with stretch function $\ell(n)$.*

Proof. Let G be a PRG with a stretching function $n+1$. We construct a PRG G^i with stretching function $\ell(n) = n+i$. Define

$$G^i(\mathcal{U}_n) = \begin{cases} G(\mathcal{U}_n) & i = 1 \\ G^{i-1}\left(G(\mathcal{U}_n)_{1\cdots n}\right) \circ G(\mathcal{U}_n)_{n+1} & i > 1 \end{cases}$$

where $G(\mathcal{U}_n)_i$ is the $i$bit of $G(X_n)$, $G(\mathcal{U}_n)_{i\cdots j}$ is the substring of $G(\mathcal{U}_n)$ from the ith bit up to the jth bit, and $\circ$ represents the concatenation operator between two strings.

Define a sequence of distributions as follows:

$$\mathcal{D}_0 : \mathcal{U}_n \circ \mathcal{U}_m, \ \ \mathcal{D}_1 : G(\mathcal{U}_n) \circ \mathcal{U}_{m-1}, \ \cdots, \ \mathcal{D}_m : G^m(\mathcal{U}_n).$$

Now we show that each G^i is a PRG. It suffices to show that there is no efficient algorithm to distinguish $G^m(\mathcal{U}_n)$ from $\mathcal{D}_0$. Assume for contradiction that there is an algorithm A and polynomial p, such that

$$|\mathbf{Pr}[A(\mathcal{D}_m) = 1] - \mathbf{Pr}[A(\mathcal{D}_0) = 1]| \geq \frac{1}{p(n)}.$$

Because

$$\begin{aligned} \Big|\mathbf{Pr}[A(\mathcal{D}_m) = 1] - \mathbf{Pr}[A(\mathcal{D}_0) = 1]\Big| &= \left|\sum_{i=1}^{m} \left(\mathbf{Pr}[A(\mathcal{D}_i) = 1] - \mathbf{Pr}[A(\mathcal{D}_{i-1}) = 1]\right)\right| \\ &\leq \sum_{i=1}^{m} \Big|\mathbf{Pr}[A(\mathcal{D}_i) = 1] - \mathbf{Pr}[A(\mathcal{D}_{i-1}) = 1]\Big|, \end{aligned}$$

there is $i \in \{1, \cdots, m\}$ such that

$$\Big|\mathbf{Pr}[A(\mathcal{D}_i) = 1] - \mathbf{Pr}[A(\mathcal{D}_{i-1}) = 1]\Big| > \frac{1}{m \cdot p(n)},$$

which contradicts to the assumption that $\mathcal{D}_i \sim^c \mathcal{D}_{i+1}$. □

So far we discussed the PRGs for polynomial-time algorithms, and constructing such PRGs rely on the existence of one-way functions. It is known that if we weaken the abilities of the algorithms that PRGs need to fool, e.g. algorithms has bounded-space, then such PRGs can be constructed without assuming the existence of one-way functions. For instance, we show the PRGs that fool space-bounded computation.

Definition 16. *Let M be a randomized algorithm that on input w requires $\ell(|w|)$ random bits. The family $\{G_n\}_{n=1}^{\infty}$ of functions ($G_n : \{0,1\}^{s(n)} \to \{0,1\}^{\ell(n)}$) is an $\varepsilon-$PRG for M if for any $w \in \{0,1\}^*$, it holds that*

$$\left|\mathbf{Pr}_{r \in \{0,1\}^{\ell(|w|)}}\left[M(w,r) = 1\right] - \mathbf{Pr}_{z \in \{0,1\}^{s(|w|)}}\left[M\left(w, G_{|w|}(z)\right) = 1\right]\right| < \varepsilon.$$

Theorem 17 ([2])**.** *For any R and S there exists an (explicitly given) pseudorandom generator which converts a random seed of length $O(S \log R)$ to R bits which look random to any algorithm running in space S.*

Bibliography

[1] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[2] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[3] Salil P. Vadhan. The unified theory of pseudorandomness: guest column. *SIGACT News*, 38(3):39–54, 2007.