Pseudorandomness

(Notes for an overview talk)

Oded Goldreich
Department of Computer Science
Weizmann Institute of Science, Israel.
oded.goldreich@weizmann.ac.il

July 5, 2006

Abstract

A fresh view at the *question of randomness* was taken in the theory of computing: It has been postulated that a distribution is pseudorandom if it cannot be told apart from the uniform distribution by an efficient procedure. The paradigm, originally associating efficient procedures with polynomial-time algorithms, has been applied also with respect to a variety of limited classes of such distinguishing procedures. Starting with the general paradigm, we present the case of general-purpose pseudorandom generators (running in polynomial-time and withstanding any polynomial-time distinguisher), as well as derandomization-aimed generators (used towards the derandomization of complexity classes such as $\mathcal{BPP}$), generators withstanding space-bounded distinguishers, and some special-purpose generators.

This document contains preparation notes for a one-hour talk on the subject. References and further details can be found in the author's texts [2, Chap. 3] and [3, Chap. 8].

Keywords: Pseudorandom generators, Derandomization, One-way functions, Hard-core predicates, Worst-case vs Average-case, Hardness vs Randomness, Pairwise independence.

Contents

1 **Randomness and Computation** 1

2 **Computational view of randomness** 2

3 **Computational Indistinguishability** 3

4 **Notions of pseudorandom generators** 4

5 **On constructing general-purpose PRGs** 5

6 **OWF imply PRG** 6

7 **OWF imply Hardcore** 7
Detail: how to generate the samples 7

8 **Hardness vs Randomness, Act 2** 8

9 **On constructing canonical derandomizers** 9
Detail: unpredictability implies pseudorandomness 9

10 **PRGs for space-bounded distinguishers** 10

11 **Special-purpose PRGs** 11

12 **Credits** 12

1 Randomness and Computation

Contrasting two points of view:

1. Randomness as a tool, used in computation.

   Essential uses include

   - Cryptography and distributed computing.
   - Probabilistic Proof Systems (e.g., IP, ZK, and PCP).
   - Sampling and Property Testing.

   Arguably these have an information theoretic flavour. I have omitted, on purpose, the applications to standard algorithms (as these may be non-essential).

2. Randomness as an object, viewed by (resource-bounded) computations.

   Computational perspective of randomness: Computational Indistinguishability.

   The question is *how are different random phenomena viewed by computational-bounded observers*. In particular, can such observers tell these phenomena apart?

   Leads to saving (and sometimes even elimination) of randomness in computation settings.

2 Computational view of randomness

A computational view of randomness leads to the notion of *computational indistinguishability*. Consider the following three cases:

1. Identically distributed random variables (or probability ensembles), denoted $X \equiv Y$.

2. Statistically indistinguishable random variables, denoted $X \overset{s}{=} Y$. This means that the variation distance between these probability ensembles is negligible.

3. Computationally indistinguishable random variables, denoted $X \overset{c}{=} Y$. This means that no algorithm, in a specified class of algorithms, can tell these probability ensembles apart.

   The classes we shall consider are: probabilistic polynomial-time algorithms, (non-uniform) polynomial-size circuits, (non-uniform) quadratic-size [sic] circuits, space-bounded algorithms, syntactically restricted algorithms (e.g., projection tests, linear tests, hitting tests).

The potential benefit of the relaxed notion of indistinguishability is that it may allow for the generation of pseudorandom sequences (i.e., computationally indistinguishable from uniform) using less randomness. [Indeed, wait and see...]

3 Computational Indistinguishability

The formulation refers to probability ensembles of the type $Z = \{Z_k\}$, where $Z_k \in \{0,1\}^k$ (or $Z_k \in \{0,1\}^{\mathrm{poly}(k)}$.

For an algorithm (potential distinguisher) D, we consider the probability that D outputs 1 (indicating that the sample is taken from X) when given a sample of X_k versus the probability that D outputs 1 when given a sample of Y_k. If these two probabilities are fairly close (i.e., $\Pr[D(X_k) = 1] \approx \Pr[D(Y_k) = 1]$) then this indication is meaningless; that is, D does not distinguish X from Y.

Formally, we require that the difference $\delta(k) \stackrel{\mathrm{def}}{=} |\Pr[D(X_k) = 1] - \Pr[D(Y_k) = 1]|$ is negligible; that is, δ is a negligible function, where typically negligible means being inversely proportional to the complexity (e.g., running-time) of the distinguisher.

We consider various classes of distinguishers. In particular:

- probabilistic polynomial-time algorithms, and (non-uniform) polynomial-size circuits,
- (non-uniform) quadratic-size [sic] circuits,
- space-bounded algorithms,
- syntactically restricted algorithms (e.g., projection tests, linear tests, hitting tests).

4 Notions of pseudorandom generators

A (generic) notion of a pseudorandom generator (PRG) refers to three issues:

1. *Stretch*: $G : \{0,1\}^k \to \{0,1\}^{\ell(k)}$, for $\ell(k) > k$ (and actually $\ell(k) \gg k$). Typically, we also upper-bound the stretch analogously to the next item (i.e., "efficient generation"):
   (a) ℓ is polynomially bounded (i.e., $\ell(k) \leq \text{poly}(k)$)
   (b) ℓ is exponentially bounded (i.e., $\ell(k) \leq \exp(O(k))$)
2. *Efficient generation*:
   (a) G produces each output bit in polynomial-time
   (b) G produces each output bit in exponential-time

   Time is stated as a function of the generator's input, called its seed.
3. *Pseudorandomness*; that is, computational indistinguishability from the uniform probability ensemble (i.e., $\{G(U_k)\} \stackrel{c}{\equiv} \{U_{\ell(k)}\}$). Two central notions of computational indistinguishability are:
   (a) computational indistinguishability by probabilistic polynomial-time algorithms
   (b) computational indistinguishability by (non-uniform) quadratic-size circuits,

Two famous (popular) incarnations follow.

General-purpose PRG: Taking the first option of each item (i.e., 1a+2a+3a). This yields a PRG that works in polynomial-time, stretches its seed by a polynomial amount, and produces sequences that are as good as random ones with respect to any (feasible) application. That is, this PRG is universal, and it can be used to shrink the randomness consumption of any efficient procedure. In particular, the output looks random also to observers that use more resources than were used in the generation process, which is essential for cryptographic applications.

Canonical derandomizers: Taking the second option of each item (i.e., 1b+2b+3b). In contrast to the general-purpose PRG, here the generation may (and typically does) take more resources than available to the observer. Although this seems "unfair", such construct are useful (esp., in the context of derandomization (e.g., emulating $\mathcal{BPP}$ by $\mathcal{P}$)).

5 On constructing general-purpose PRGs

Recall $G : \{0,1\}^k \to \{0,1\}^{\ell(k)}$, where $\ell(k) > k$ (or ℓ is any polynomial), is polynomial-time computable and $\{G(U_k)\} \overset{\text{ppt}}{=} \{U_{\ell(k)}\}$. Indeed, having a PRG with some stretch (even $\ell(k) = k+1$) yields PRGs with arbitrary polynomial stretch (e.g., by "simple" or "sophisticated" iterations).[1]

THM ("Randomness vs Hardness"): Such PRGs exists if and only if one-way functions exist.

DEF (one-way functions (OWF)): $f : \{0,1\}^k \to \{0,1\}^k$ is a OWF if

1. it is polynomial-time computable
2. it hard to invert on the average-case; that is, for every probabilistic polynomial-time algorithm A,
$$\Pr_{x \leftarrow U_k}[A(f(x)) \in f^{-1}(f(x))] = \text{negl}(k)$$

PRG implies OWF: Let $G : \{0,1\}^k \to \{0,1\}^{2k}$ be a PRG. Consider $f(x,y) = G(x)$, where $|x| = |y|$. If you can invert f on $f(U_{2k}) = G(U_k)$ then you can distinguish $G(U_k)$ from U_{2k}, since the probability that the latter has a f-preimage (at all!) is negligible.

OWF implies PRG: Far more complicated. We'll see a special case next.

[1] The simple iteration yields $G'(s) = G^{\ell(|s|)-|s|}(s)$, where $G^{i+1}(x) = G(G^i(x))$ and $G^0(x) = x$. In the alternative ("fixed-length" iteration) method we have $G'(s) = \sigma_1 \cdot \sigma_{\ell(|s|)}$, where $s_0 = s$ and $\sigma_i s_i = G(s_{i-1})$ for $i = 1, ..., \ell(|s|)$. In both cases, the analysis relies on the distinguisher's ability to apply G.

6 OWF imply PRG

An "intermediate" notion is that of a hardcore of a function f.

DEF (hardcore): The predicate $b : \{0,1\}^k \to \{0,1\}$ is a hardcore of $f : \{0,1\}^k \to \{0,1\}^k$ if

1. b is polynomial-time computable
2. $b(x)$ hard to predict from $f(x)$, in the average-case sense; that is, for every probabilistic polynomial-time algorithm A,

$$\Pr_{x \leftarrow U_k}[A(f(x)) = b(x)] < \frac{1}{2} + \mathrm{negl}(k)$$

The claim that $b(U_k)$ is hard to predict from $f(U_k)$ is related to the claim that $f(U_k)b(U_k)$ is computationally indistinguishable from $f(U_k)U_1$. See more below. But first note that

1. An individual bit of the input of a OWF f need NOT be a hardcore of f. Consider, for example, $f(x, y) = (f'(x), y)$.
2. If f is 1-1 and polynomial-time invertible then it has no hardcore, because $f(x) \mapsto x \mapsto b(x)$ is easy to compute.

In the case that a OWF f is 1-1, any hardcore of f yields a PRG. For a hardcore b of f, we set $G(s) = f(s)b(s)$. Note that $G(U_k) = f(U_k)b(U_k)$ is computationally indistinguishable from $f(U_k)U_1 \equiv U_{k+1}$ (by the 1-1 property).

Thus, we merely need to show that and (1-1) OWF has a hardcore. This is done next.

7 OWF imply Hardcore

Given an arbitrary OWF f_0, we claim that $b(x,r) = \sum_{i=1}^{k} x_i r_i \bmod 2$ is a hardcore of the OWF $f(x,r) = (f_0(x), r)$. That is, if given $f_0(x)$ is hard to obtain x then it is hard to predict $b(x,r)$ when given $f_0(x)$ and a random r (i.e., it is hard to predict the XOR of a random subset of the bits of x).

Analysis of the counter-positive. *Suppose we are given oracle access to a function* $B : \{0,1\}^k \to \{0,1\}$ *such that for some* x *it holds that*

$$\mathrm{Pr}_{r \leftarrow U_k}[B(r) = b(x,r)] \geq \frac{1}{2} + \epsilon$$

Then, in $\mathrm{poly}(k/\epsilon)$*-time, we can guess* x *correctly with probability at least* $\mathrm{poly}(\epsilon/k)$.
[Indeed, think of B as an algorithm that violates the claim that b is a hardcore of f. Actually, B is derived from such an algorithm A by setting $B_x(r) = A(f(x), r)$.]

Warm-up. Suppose $p_x \stackrel{\mathrm{def}}{=} \mathrm{Pr}_{r \leftarrow U_k}[B(r) = b(x,r)] \geq \frac{3}{4} + \epsilon$. Recover x_j with probability at least $1 - 2 \cdot (1 - p_x) \geq 0.5 + 2\epsilon$ by selecting uniformly $r \in \{0,1\}^k$ and outputting $B(r) \oplus B(r \oplus e^j)$.[2] Repeating this experiment $m = O(k/\epsilon^2)$ times, and ruling by majority we are correct with probability at least $1 - (1/2k)$, even if the samples are only pairwise independent.

Eliminating the error-doubling phenomenon. Suppose we can generate uniformly distributed and pairwise independent $r^{(1)}, ..., r^{(m)} \in \{0,1\}^k$ such that we know the value of each $b(x, r^{(i)})$. Then, with probability at least $1 - (1/2k)$, the majority vote of $b(x, r^{(i)}) \oplus B(r^{(i)} \oplus e^j)$ yields x_j.

So it is left to provide a procedure for generating such $r^{(i)}$'s. More accurately, generate uniformly distributed and pairwise independent $r^{(1)}, ..., r^{(m)} \in \{0,1\}^k$ such that, with probability $\mathrm{poly}(\epsilon/k)$, we correctly guess the value of all $b(x, r^{(i)})$'s.

Detail: how to generate the samples

For $\ell = \log_2(m+1)$, select uniformly $s^{(1)}, ..., s^{(\ell)} \in \{0,1\}^k$.

Guess $b(x, s^{(1)}), ..., b(x, s^{(\ell)}) \in \{0,1\}$. The guess is correct with probability $2^{-\ell} = 1/(m+1)$.

Compute $(r^{(I)})_{\emptyset \neq I \subset [\ell]}$ such that $r^{(I)} = \oplus_{i \in I} s^{(i)}$. Compute $b(x, r^{(I)})$ as

$$b(x, \oplus_{i \in I} s^{(i)}) = \oplus_{i \in I} b(x, s^{(i)}).$$

These $(r^{(I)})_{\emptyset \neq I \subset [\ell]}$ are pairwise independent and uniformly distributed in $\{0,1\}^k$. If the guesses for all $b(x, s^{(i)})$'s are correct then the values of all $b(x, r^{(I)})$'s are correct.

[2]Note that $b(x,r) \oplus b(x, r \oplus e^j)$ equals $(\sum_{i=1}^{k} x_i r_i) + (x_j + \sum_{i=1}^{k} x_i r_i) = x_j$.

8 Hardness vs Randomness, Act 2

Recall $G : \{0,1\}^k \to \{0,1\}^{\ell(k)}$ is called a canonical derandomizer if G is exponential-time computable and $\{G(U_k)\} \overset{\text{quad-size}}{=} \{U_{\ell(k)}\}$.

Using such G we derandomize a probabilistic polynomial-time A (viewed as a two-input algorithm with a second input having length that equals the running-time as a function of first input) as follows.

1. $A'(x,s) = A(x, G(s))$.

   This reduces randomness from $\text{poly}(|x|) = \ell(k)$ to $k = \ell^{-1}(\text{poly}(|x|))$. For $\ell(k) = 2^{\Omega(k)}$, we get $k = \ell^{-1}(\text{poly}(|x|)) = O(\log |x|)$.

   The existence of x for which $|\Pr[A(x, U_{\ell(k)}) = 1] - \Pr[A(x, G(U_k)) = 1]| > 0.1$ implies a quadratic-size ($\ell(k)^2$-size) circuit distinguishing $U_{\ell(k)}$ from $G(U_k)$.

2. $A''(x) = \text{maj}_{s \in \{0,1\}^k}\{A'(x,s)\}$.

   The running-time of A'' is $2^k \cdot (\text{time}_A(|x|) + \text{time}_G(k))$.

Our aim is constructing canonical derandomizers with as large as possible stretch function (ideally, exponential stretch function).

THM: *A canonical derandomizer with stretch $\ell(k) = 2^{\Omega(k)}$ implies $\mathcal{BPP} = \mathcal{P}$.*

THM: *If $\mathcal{E} = \text{Dtime}(2^{O(n)})$ contains a set that requires exponential size circuits*[3] *then there exists a canonical derandomizer with stretch $\ell(k) = 2^{\Omega(k)}$.*

[3] That is, for all but finitely many n's deciding this set on $\{0,1\}^n$ requires circuits of size $2^{\Omega(n)}$. Note that we refer to worst-case complexity, but in the almost-everywhere (a.e.) sense.

9 On constructing canonical derandomizers

Not shown: worst-case hardness (for $\mathcal{E}$) implies average-case hardness (for $\mathcal{E}$). Thus, our starting point is a function f (in $\mathcal{E}$) such that for every $2^{\Omega(m)}$-size circuit C_m it holds that

$$\Pr_{x \leftarrow U_m}[C_m(x) = f(x)] < \frac{1}{2} + 2^{-\Omega(m)}$$

The construction follows, where s_I denotes the projection of $s \in \{0,1\}^k$ on the coordinates in $I \subseteq [k]$.

$$G(s) = f(s_{I_1}) f(s_{I_2}) \cdots f(s_{I_{\ell(k)}}),$$

where $I_j \subset [k]$ such that $|I_j| = m$ and $|I_j \cap I_{j'}| \leq m' < m$ (for any $j' \neq j$).
Computing G involves computing $I_1, I_2, ..., I_{\ell(k)}$, and evaluating f on $\ell(k) < 2^k$ points. The time for computing G is $2^{O(m)} > 2^k$, which will be smaller than $\ell(k)$ and even $\ell(k)^2$. [Thus, by construction, computing G is infeasible for the distinguisher, and indeed we shall show that not only that $G(U_k)$ is indistinguishable from $U_{\ell(k)}$ (by $\ell(k)^2$-size circuits) but $U_k G(U_k)$ is indistinguishable from $U_{k+\ell(k)}$.]

Pseudorandomness versus unpredictability. [This theme has already appeared wrt hard-core.]

- Pseudorandomness implies unpredictability, simply because the uniform ensembles is unpredictable.
- But the direction we need here is *unpredictability implies pseudorandomness.* [See details in next slide.]

Thus, we focus on establishing the unpredictability of $G(U_k)$.

A warm-up and beyond. Suppose that the I_j's are disjoint (which is impossible as this implies $\ell(k) \leq k/m$). Then unpredictability is straightforward. The intuition is that small intersections bound the gain towards guessing $f(s_{I_{j+1}})$ obtained from having $f(s_{I_1}) \cdots f(s_{I_j})$. The point is that, for $j' \in [j]$, the value of $f(s_{I_{j'}})$ depends on at most m' bits of $s_{I_{j+1}}$.

Detail: unpredictability implies pseudorandomness

Suppose that $\{Z_k\}$ is not pseudorandom; that is, for some adequate distinguisher D, it holds that $\{Z_k\} \stackrel{\text{D}}{\not\equiv} \{U_{\ell(k)}\}$.

Consider the following hybrids, $i = 0, 1, ..., \ell(k)$. The i^{th} hybrid, denoted $H_k^{(i)}$, consists of an i-bit long prefix of Z_k followed by $\ell(k) - i$ uniformly distributed bits (i.e., an $(\ell(k) - i)$-bit long suffix of $U_{\ell(k)}$). Note that $H_k^{(0)} \equiv U_{\ell(k)}$ whereas $H_k^{(i)} \equiv Z_k$. Thus, $\{Z_k\} \stackrel{\text{D}}{\not\equiv} \{U_{\ell(k)}\}$ implies $\{H_k^{(i)}\} \stackrel{\text{D}}{\not\equiv} \{H_k^{(i+1)}\}$ for some $i \in \{0, 1..., \ell(k) - 1\}$ (with a possible loss of a factor of $\ell(k)$ in the gap).

This means that we can distinguish the $(i+1)$-bit long prefix of $H_k^{(i)}$ from the $(i+1)$-bit long prefix of $H_k^{(i+1)}$, which can be translated to predicting the $i + 1^{\text{st}}$ bit of $H_k^{(i+1)}$ (equiv., predicting the $i + 1^{\text{st}}$ bit of Z_k) based on the preceding i bits.

10 PRGs for space-bounded distinguishers

Typical probabilistic polynomial-time applications have space complexity that is significantly smaller than their time complexity. This motivates the study of PRGs that fool the corresponding space-bounded distinguishers. A useful result to bear in mind follows.

THM: *Every probabilistic polynomial-time algorithm can be emulated by a probabilistic polynomial-time algorithm that uses randomness that is linear in the sum of the space complexity* (of the original algorithm) *and the length of the input.*

Note: This THM is based on a construction of a PRG that (in contrast to the previous PRGs) does not rely on any computational assumptions. That is, the corresponding hardness needed here can be proved (rather than needs to be assumed).

CONJ: Every probabilistic polynomial-time algorithm can be emulated by a probabilistic polynomial-time algorithm that uses randomness that is linear in the sum of the space complexity (of the original algorithm) and the *logarithm of the length of the input.*

Evidence in support of CONJ is provided by a PRG that uses randomness that is quadratic in the space complexity, by $\mathcal{BPL} \subseteq \mathcal{SC}$, and by the deterministic log-space algorithm for undirected graph connectivity (which for a couple of decades was only known to be in $\mathcal{RL}$).

11 Special-purpose PRGs

Special-purpose PRGs are useful in many applications. These PRGs should only withstand restricted (syntactic) tests, and can be constructed without relying on any assumptions.

Projection tests and t-wise PRGs. These require that the projection of any t coordinates (in the tested distribution) is uniformly distributed. Known constructions include random univariate polynomials of degree $t-1$, and random affine transformations.

Linear tests and small-bias PRGs. These require that any (non-zero) linear combination of the bits (of the tested distribution) is almost-uniformly distributed. One of the known construction is a random LFSR (i.e., one with a random feedback rule). A typical application of is for the construction of PCPs for the satisfiability of systems of equations.

Hitting tests and expander walk PRGs. We refer to a distribution on sequences of length ℓ over $\{0,1\}^n$. The criteria is that for any set $S \subset \{0,1\}^n$ of density 1/2, the probability that all blocks in the distribution miss S is $\exp(-\Omega(\ell))$. That is, some block hits S with probability at least $1-\exp(-\Omega(\ell))$. One of the known construction is a random walk of length ℓ over a 2^n-vertex expander graph.

12 Credits

The concept of computational indistinguishability [5, 13]
The notion of (general-purpose) pseudorandom generator [1, 13]

Constructions of (general-purpose) pseudorandom generator (PRG): hardcore and iterations [1], hardcore for any one-way function [4], PRG based on any one-way function [6].

Notion and construction of canonical derandomizers [10]. The conditional full derandomization of $\mathcal{BPP}$ [7].

PRG for space-bounded computations: reducing randomness to linear in space and length [11], towards reducing it to linear in space [8, 9, 12].

Special-purpose PRGs: see credits in [2, Chap. 3] and [3, Chap. 8].

References

[1] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SICOMP*, Vol. 13, pages 850–864, 1984. Preliminary version in *23rd FOCS*, 1982.

[2] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Algorithms and Combinatorics series (Vol. 17), Springer, 1999.

[3] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. In preparations. Drafts available at `http://www.wisdom.weizmann.ac.il/~oded/cc-book.html`

[4] O. Goldreich and L.A. Levin. Hard-core Predicates for any One-Way Function. In *21st STOC*, pages 25–32, 1989.

[5] S. Goldwasser and S. Micali. Probabilistic Encryption. *JCSS*, Vol. 28, No. 2, pages 270–299, 1984. Preliminary version in *14th STOC*, 1982.

[6] J. Håstad, R. Impagliazzo, L.A. Levin and M. Luby. A Pseudorandom Generator from any One-way Function. *SICOMP*, Vol. 28, No. 4, pages 1364–1396, 1999. Preliminary versions by Impagliazzo *et. al.* in *21st STOC* (1989) and Håstad in *22nd STOC* (1990).

[7] R. Impagliazzo and A. Wigderson. P=BPP if E requires exponential circuits: Derandomizing the XOR Lemma. In *29th STOC*, pages 220–229, 1997.

[8] N. Nisan. Pseudorandom Generators for Space Bounded Computation. *Combinatorica*, Vol. 12 (4), pages 449–461, 1992. Preliminary version in *22nd STOC*, 1990.

[9] N. Nisan. $\mathcal{RL} \subseteq \mathcal{SC}$. *J. of Computat. Complex.*, Vol. 4, pages 1-11, 1994. Preliminary version in *24th STOC*, 1992.

[10] N. Nisan and A. Wigderson. Hardness vs Randomness. *JCSS*, Vol. 49, No. 2, pages 149–167, 1994. Preliminary version in *29th FOCS*, 1988.

[11] N. Nisan and D. Zuckerman. Randomness is Linear in Space. *JCSS*, Vol. 52 (1), pages 43–52, 1996. Preliminary version in *25th STOC*, 1993.

[12] O. Reingold. Undirected ST-Connectivity in Log-Space. In *37th STOC*, pages 376–385, 2005.

[13] A.C. Yao. Theory and Application of Trapdoor Functions. In *23rd FOCS*, pages 80–91, 1982.