Chapter 1

Introduction

The "question of randomness" has been puzzling thinkers for ages. Aspects of this question range from philosophical doubts regarding the existence of randomness (in the world) and reflections on the meaning of randomness (in our thinking) to technical questions regarding the measuring of randomness. Among many other things, the second half of the twentieth century has witnessed the development of three theories of randomness, which address different aspects of the foregoing question.

The first theory (cf., [16]), initiated by Shannon [63], views randomness as representing *uncertainty*, which in turn is modeled by a probability distribution on the possible values of the missing data. Indeed, Shannon's Information Theory is rooted in probability theory. Information Theory focuses on distributions that are not perfectly random (i.e., encode information in a redundant manner), and characterizes perfect randomness as the extreme case in which the uncertainty is maximized (i.e., in this case there is no redundancy at all). Thus, perfect randomness is associated with a unique distribution– the uniform one. In particular, by definition, one cannot (deterministically) generate such perfect random strings from shorter random seeds.

The second theory (cf., [41, 42]), initiated by Solomonoff [64], Kolmogorov [38], and Chaitin [14], views randomness as representing the lack of structure, which in turn is reflected in the length of the most succinct (effective) description of the object. The notion of a succinct and *effective description* refers to a process that transforms the succinct description to an explicit one. Indeed, this theory of randomness is rooted in computability theory and specifically in the notion of a universal language (equiv., universal machine or computing device). It measures the randomness (or complexity) of objects in terms of the shortest program (for a fixed universal machine) that generates the object.[1] Like Shannon's theory, Kolmogorov Complexity is quantitative and perfect random objects appear as an extreme case. However, following Kolmogorov's approach one may say that a single object, rather than a distribution over objects, is perfectly random. Still, by definition, one cannot (deterministically) generate strings of high Kolmogorov Complexity from short random seeds.

[1] We mention that Kolmogorov's approach is inherently intractable (i.e., Kolmogorov Complexity is uncomputable).

1.1 The Third Theory of Randomness

The third theory, which is the focus of the current primer, views randomness as an effect on an observer and thus as being relative to the *observer's abilities* (of analysis). The observer's abilities are captured by its computational abilities (i.e., the complexity of the processes that the observer may apply), and hence this theory of randomness is rooted in complexity theory. This theory of randomness is explicitly aimed at providing a notion of randomness that, unlike the previous two notions, allows for an efficient (and deterministic) generation of random strings from shorter random seeds. The heart of this theory is the suggestion to view objects as equal if they cannot be distinguished by any efficient procedure. Consequently, a distribution that cannot be efficiently distinguished from the uniform distribution will be considered random (or rather called pseudorandom). Thus, randomness is not an "inherent" property of objects (or distributions) but is rather relative to an observer (and its computational abilities). To illustrate this perspective, let us consider the following mental experiment.

> Alice and Bob play "heads or tails" in one of the following four ways. In each of them, Alice flips an unbiased coin 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. 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 substantially improve his guess of the outcome of the coin.

We conclude that the randomness of an event is relative to the information and computing resources at our disposal. At the extreme, even events that are fully determined by public information may be perceived as random events by an observer who lacks the relevant information and/or the ability to process it. Our focus will be on the lack of sufficient processing power, and not on the lack of sufficient information. The lack of sufficient processing power may be due either to the formidable amount of computation required (for analyzing the event in question) or to the fact that the observer happens to be very limited.

A natural notion of pseudorandomness arises: a distribution is *pseudorandom* if no efficient procedure can distinguish it from the uniform distribution, where efficient procedures are associated with (probabilistic) polynomial-time algorithms. This specific notion of pseudorandomness is indeed the most fundamental one, and much of this text is focused on it. Weaker notions of pseudorandomness arise as well – they refer to indistinguishability by weaker procedures such as space-bounded algorithms, constant-depth circuits, etc. Stretching this approach even further one may consider algorithms that are designed (on purpose so) not to distinguish even weaker forms of "pseudorandom" sequences from random ones. Such algorithms arise naturally when trying to convert some natural randomized algorithm into deterministic ones; see Chapter 5.

The preceding discussion has focused on one aspect of the pseudorandomness question – the resources or type of the observer (or potential distinguisher). Another important aspect is whether such pseudorandom sequences can be generated from much shorter ones, and at what cost (or complexity). A natural approach requires the generation process to be efficient, and furthermore to be fixed before the specific observer is determined. Coupled with the aforementioned strong notion of pseudorandomness, this yields the archetypical notion of pseudorandom generators – those operating in (fixed) polynomial-time and producing sequences that are indistinguishable from uniform ones by *any* polynomial-time observer. In particular, this means that the distinguisher is allowed more resources than the generator. Such (general-purpose) pseudorandom generators (discussed in Chapter 2) allow one to decrease the randomness complexity of *any efficient application*, and are thus of great relevance to randomized algorithms and cryptography. The term *general-purpose* is meant to emphasize the fact that the same generator is good for all efficient applications, including those that consume more resources than the generator itself.

Although general-purpose pseudorandom generators are very appealing, there are important reasons for considering also the opposite relation between the complexities of the generation and distinguishing tasks; that is, allowing the pseudorandom generator to use more resources (e.g., time or space) than the observer it tries to fool. This alternative is natural in the context of derandomization (i.e., converting randomized algorithms to deterministic ones), where the crucial step is replacing the random input of an algorithm by a pseudorandom input, which in turn can be generated based on a much shorter random seed. In particular, when derandomizing a probabilistic polynomial-time algorithm, the observer (to be fooled by the generator) is a fixed algorithm. In this case employing a more complex generator merely means that the complexity of the derived deterministic algorithm is dominated by the complexity of the generator (rather than by the complexity of the original randomized algorithm). Needless to say, allowing the generator to use more resources than the observer that it tries to fool makes the task of designing pseudorandom generators potentially easier, and enables derandomization results that are not known when using general-purpose pseudorandom generators. The usefulness of this approach is demonstrated in Chapters 3 through 5.

We note that the goal of all types of pseudorandom generators is to allow the generation of "sufficiently random" sequences based on much shorter random seeds. Thus, pseudorandom generators offer significant savings in the randomness complexity of various applications (and in some cases eliminating randomness altogether). Saving on randomness is valuable because many applications are severely limited in their ability to generate or obtain truly random bits. Furthermore, typically, generating truly random bits is significantly more expensive than standard computation steps. Thus, randomness is a computational resource that should be considered on top of time complexity (analogously to the consideration of space complexity).

1.2 Organization of the Primer

We start by presenting some standard conventions (see Section 1.3). Next, in Section 1.4, we present the general paradigm underlying the various notions of pseudorandom generators. The archetypical case of general-purpose pseudorandom generators is presented in Chapter 2. We then turn to alternative notions of pseudorandom generators: generators that suffice for the derandomization of complexity classes such as $\mathcal{BPP}$ are discussed in Chapter 3; pseudorandom generators in the domain of space-bounded computations are discussed in Chapter 4; and several notions of special-purpose generators are discussed in Chapter 5.

The text is organized to facilitate the possibility of focusing on the notion of general-purpose pseudorandom generators (presented in Chapter 2). This notion is most relevant to computer science at large, and consequently it is most relevant to other sciences. Furthermore, the technical details presented in Chapter 2 are relatively simpler than those presented in Chapters 3 and 4.

The appendices. For the benefit of readers who are less familiar with computer science, we augment the foregoing material with six appendices. Appendix A provides a basic treatment of hashing functions, which are used in Section 4.2 and are related to the limited-independence generators discussed in Section 5.1. Appendix B provides a brief introduction to the notion of randomness extractors, which are of natural interest as well as being used in Section 4.2. Appendix C provides a proof of a key result that is closely related to the material of Section 2.5. Appendix D provides three illustrations to the use of randomness in computation. Appendix E presents a couple of basic cryptographic applications of pseudorandom functions, which are treated in Section 2.7.2. Appendix F provides definitions of some basic complexity classes.

Relation to complexity theory. The study of pseudorandom generators is part of complexity theory, and the interested reader is encouraged to further explore the connections between pseudorandomness and complexity theory at large (cf. e.g., [24]). In fact, the current primer is an abbreviated (and revised) version of [24, Chap. 8].

Preliminaries. We assume a basic familiarity with computational complexity; that is, we assume that the reader is comfortable with the notion of efficient algorithms and their association with polynomial-time algorithms (see, e.g., [24]). We also assume that the reader is aware that very basic questions about the nature of efficient computation are wide open (e.g., most notably, the P-vs-NP Question).

We also assume a basic familiarity with elementary probability theory (see any standard textbook or brief reviews in [46, 47, 24]) and randomized algorithms (see, e.g., either [47, 46] or [24, Chap. 6]). In particular, standard conventions regarding random variables (presented next) will be extensively used.

1.3 Standard Conventions

Throughout the entire text we refer only to *discrete* probability distributions. Specifically, the underlying probability space consists of the set of all strings of a certain length ℓ, taken with uniform probability distribution. That is, the sample space is the set of all ℓ-bit long strings, and each such string is assigned probability measure $2^{-\ell}$. Traditionally, *random variables* are defined as functions from the sample space to the reals. Abusing the traditional terminology, we use the term random variable also when referring to functions mapping the sample space into the set of binary strings. We often do not specify the probability space, but rather talk directly about random variables. For example, we may say that X is a random variable assigned values in the set of all strings such that $\mathsf{Pr}[X=00]=\frac{1}{4}$ and $\mathsf{Pr}[X=111]=\frac{3}{4}$. (Such a random variable may be defined over the sample space $\{0,1\}^2$ such that $X(11)=00$ and $X(00)=X(01)=X(10)=111$.) One important case of a random variable is the output of a randomized process (e.g., a probabilistic polynomial-time algorithm).

All of our probabilistic statements refer to random variables that are defined beforehand. Typically, we may write $\mathsf{Pr}[f(X)=1]$, where X is a random variable defined beforehand (and f is a function). An important convention is that *all occurrences of the same symbol in a probabilistic statement refer to the same* (unique) *random variable*. Hence, if $B(\cdot,\cdot)$ is a Boolean expression depending on two variables, and X is a random variable, then $\mathsf{Pr}[B(X,X)]$ denotes the probability that $B(x,x)$ holds when x is chosen with probability $\mathsf{Pr}[X=x]$. For example, for every random variable X, we have $\mathsf{Pr}[X=X]=1$. We stress that if we wish to discuss the probability that $B(x,y)$ holds when x and y are chosen independently with identical probability distribution, then we will define *two* independent random variables each with the same probability distribution. Hence, if X and Y are two independent random variables, then $\mathsf{Pr}[B(X,Y)]$ denotes the probability that $B(x,y)$ holds when the pair (x,y) is chosen with probability $\mathsf{Pr}[X=x]\cdot\mathsf{Pr}[Y=y]$. For example, for every two independent random variables, X and Y, we have $\mathsf{Pr}[X=Y]=1$ only if both X and Y are trivial (i.e., assign the entire probability mass to a single string).

Throughout the entire text, U_n denotes a random variable uniformly distributed over the set of all strings of length n. Namely, $\mathsf{Pr}[U_n=\alpha]$ equals 2^{-n} if $\alpha\in\{0,1\}^n$ and equals 0 otherwise. We often refer to the distribution of U_n as the uniform distribution (neglecting to qualify that it is uniform over $\{0,1\}^n$). In addition, we occasionally use random variables (arbitrarily) distributed over $\{0,1\}^n$ or $\{0,1\}^{\ell(n)}$, for some function $\ell:\mathbb{N}\to\mathbb{N}$. Such random variables are typically denoted by X_n, Y_n, Z_n, etc. We stress that in some cases X_n is distributed over $\{0,1\}^n$, whereas in other cases it is distributed over $\{0,1\}^{\ell(n)}$, for some function ℓ (which is typically a polynomial). We often talk about probability ensembles, which are infinite sequences of random variables $\{X_n\}_{n\in\mathbb{N}}$ such that each X_n ranges over strings of length bounded by a polynomial in n.

Statistical difference. The statistical distance (a.k.a variation distance) between the random variables X and Y is defined as

$$\frac{1}{2}\cdot\sum_v|\mathsf{Pr}[X=v]-\mathsf{Pr}[Y=v]| \;=\; \max_S\{\mathsf{Pr}[X\in S]-\mathsf{Pr}[Y\in S]\} \tag{1.1}$$

(see Exercise 1.1). We say that X is δ-close (resp., δ-far) to Y if the statistical distance between them is at most (resp., at least) δ.

1.4 The General Paradigm

We advocate a unified view of various notions of pseudorandom generators. That is, we view these notions as incarnations of a general abstract paradigm, to be presented in this section. A reader who is interested only in one of these incarnations may still use this section as a general motivation towards the specific definitions used later. On the other hand, some readers may prefer reading this section after studying one of the specific incarnations.

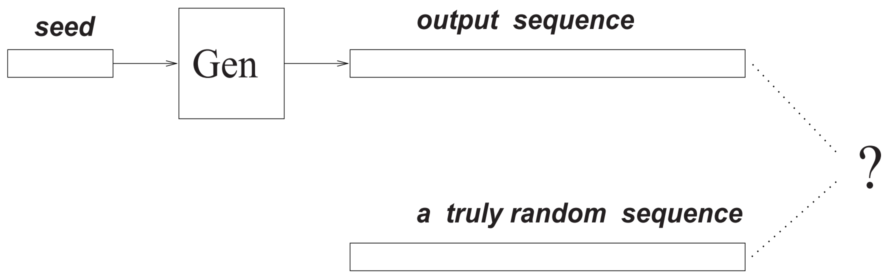

Figure 1.1: Pseudorandom generators – an illustration.

1.4.1 Three fundamental aspects

A generic formulation of pseudorandom generators consists of specifying three fundamental aspects – the *stretch measure* of the generators; the class of distinguishers that the generators are supposed to fool (i.e., the algorithms with respect to which the *computational indistinguishability* requirement should hold); and the resources that the generators are allowed to use (i.e., their own *computational complexity*). Let us elaborate.

Stretch function: A necessary requirement from any notion of a pseudorandom generator is that the generator is a *deterministic algorithm* that stretches short strings, called seeds, into longer output sequences.[2] Specifically, this algorithm stretches k-bit long seeds into $\ell(k)$-bit long outputs, where $\ell(k) > k$. The function $\ell:\mathbb{N}\to\mathbb{N}$ is called the stretch measure (or stretch function) of the generator. In some settings the specific stretch measure is immaterial (e.g., see Section 2.4).

Computational Indistinguishability: A necessary requirement from any notion of a pseudorandom generator is that the generator "fools" some non-trivial algorithms. That is, it is required that any algorithm taken from a predetermined class

[2]Indeed, the seed represents the randomness that is used in the generation of the output sequences; that is, the randomized generation process is decoupled into a deterministic algorithm and a random seed. This decoupling facilitates the study of such processes.

of interest cannot distinguish the output produced by the generator (when the generator is fed with a uniformly chosen seed) from a uniformly chosen sequence. Thus, we consider a class $\mathcal{D}$ of distinguishers (e.g., probabilistic polynomial-time algorithms) and a class $\mathcal{F}$ of (threshold) functions (e.g., reciprocals of positive polynomials), and require that the generator G satisfies the following: For any $D \in \mathcal{D}$, any $f \in \mathcal{F}$, and for all sufficiently large k it holds that

$$|\,\mathsf{Pr}[D(G(U_k)) = 1] \;-\; \mathsf{Pr}[D(U_{\ell(k)}) = 1]\,| \;<\; f(k)\,, \tag{1.2}$$

where U_n denotes the uniform distribution over $\{0,1\}^n$, and the probability is taken over U_k (resp., $U_{\ell(k)}$) as well as over the coin tosses of algorithm D in case it is probabilistic. The reader may think of such a distinguisher, D, as an observer who tries to tell whether the "tested string" is a random output of the generator (i.e., distributed as $G(U_k)$) or is a truly random string (i.e., distributed as $U_{\ell(k)}$). The condition in Eq. (1.2) requires that D cannot make a meaningful decision; that is, ignoring a negligible difference (represented by $f(k)$), D's verdict is the same in both cases.[3] The archetypical choice is that $\mathcal{D}$ is the set of all probabilistic polynomial-time algorithms, and $\mathcal{F}$ is the set of all functions that are the reciprocal of some positive polynomial.

We note that there is a clear tension between the stretching and the computational indistinguishability conditions. Indeed, as shown in Exercise 1.2, the output of any pseudorandom generator is "statistically distinguishable" from the corresponding uniform distribution. However, there is hope that a restricted class of (computationally bounded) distinguishers cannot detect the (statistical) difference; that is, be fooled by some suitable generators. In fact, placing no computational requirements on the generator (or, alternatively, imposing very mild requirements such as upper-bounding the running-time by a double-exponential function), yields "generators" that can fool any subexponential-size circuit family (see Exercise 1.3). However, we are interested in the complexity of the generation process, which is the aspect addressed next.

Complexity of Generation: This aspect refers to the complexity of the generator itself, when viewed as an algorithm. That is, here we refer to the resources used by the generator (e.g., its time and/or space complexity). The archetypical choice is that the generator has to work in polynomial-time (i.e., make a number of steps that is polynomial in the length of its input – the seed). Other choices will be discussed as well.

1.4.2 Notational conventions

We will consistently use k for denoting the length of the seed of a pseudorandom generator, and $\ell(k)$ for denoting the length of the corresponding output. In some cases, this makes our presentation a little more cumbersome, where in these cases

[3]The class of threshold functions $\mathcal{F}$ should be viewed as determining the class of noticeable probabilities (as a function of k). Thus, we require certain functions (i.e., those presented on the l.h.s of Eq. (1.2)) to be smaller than any noticeable function *on all but finitely many integers*. We call the former functions negligible. Note that a function may be neither noticeable nor negligible (e.g., it may be smaller than any noticeable function on infinitely many values and yet larger than some noticeable function on infinitely many other values).

it is more natural to focus on a different parameter (e.g., the length of the pseudorandom sequence) and let the seed-length be a function of the latter. However, our choice has the advantage of focusing attention on the fundamental parameter of pseudorandom generation process – the length of the random seed. We note that whenever a pseudorandom generator is used to "derandomize" an algorithm, n will denote the length of the input to this algorithm, and k will be selected as a function of n.

1.4.3 Some instantiations of the general paradigm

Two important instantiations of the notion of pseudorandom generators relate to polynomial-time distinguishers.

1. General-purpose pseudorandom generators correspond to the case where the generator itself runs in polynomial-time and needs to withstand *any probabilistic polynomial-time distinguisher*, including distinguishers that run for more time than the generator. Thus, the same generator may be used safely in any efficient application. (This notion is treated in Chapter 2.)

2. In contrast, pseudorandom generators intended for derandomization may run for more time than the distinguisher, which is viewed as a fixed circuit having size that is upper-bounded by a fixed polynomial. (This notion is treated in Chapter 3.)

In addition, the general paradigm may be instantiated by focusing on the space-complexity of the potential distinguishers (and the generator), rather than on their time-complexity. Furthermore, one may also consider distinguishers that merely reflect probabilistic properties such as pairwise independence, small-bias, and hitting frequency.

Notes

Our presentation, which views vastly different notions of pseudorandom generators as incarnations of a general paradigm, has emerged mostly in retrospect. We note that, while the historical study of the various notions was mostly unrelated at a technical level, the case of general-purpose pseudorandom generators served as a source of inspiration to most of the other cases. In particular, the concept of computational indistinguishability, the connection between hardness and pseudorandomness, and the equivalence between pseudorandomness and unpredictability, appeared first in the context of general-purpose pseudorandom generators (and inspired the development of "generators for derandomization" and "generators for space bounded machines"). Indeed, the study of the special-purpose generators (see Chapter 5) was unrelated to all of these.

We mention that an alternative treatment of pseudorandomness, which puts more emphasis on the relation between various techniques, is provided in [68]. In particular, the latter text highlights the connections between information theoretic and computational phenomena (e.g., randomness extractors and canonical derandomizers), while the current text tends to decouple the two.

Exercises

Exercise 1.1 Prove the equality in Eq. (1.1).

Guideline: Let S be the set of strings having a larger probability under the first distribution.

Exercise 1.2 Show that the output of any pseudorandom generator is "statistically distinguishable" from the corresponding uniform distribution; that is, show that, for any stretch function ℓ and any generator G of stretch ℓ, the statistical difference between $G(U_k)$ and $U_{\ell(k)}$ is at least $1 - 2^{-(\ell(k)-k)}$.

Exercise 1.3 Show that placing no computational requirements on the generator enables unconditional results regarding "generators" that fool any family of subexponential-size circuits. That is, making no computational assumptions, prove that there exist functions $G : \{0,1\}^* \to \{0,1\}^*$ such that $\{G(U_k)\}_{k\in\mathbb{N}}$ is (strongly) pseudorandom, while $|G(s)| = 2|s|$ for every $s \in \{0,1\}^*$. Furthermore, show that G can be computed in double-exponential time.

Guideline: Use the Probabilistic Method (cf. [6]). First, for any fixed circuit $C : \{0,1\}^n \to \{0,1\}$, upper-bound the probability that for a random set $S \subset \{0,1\}^n$ of size $2^{n/2}$ the absolute value of $\mathsf{Pr}[C(U_n) = 1] - (|\{x \in S : C(x) = 1\}|/|S|)$ is larger than $2^{-n/8}$. Next, using a union bound, prove the existence of a set $S \subset \{0,1\}^n$ of size $2^{n/2}$ such that no circuit of size $2^{n/5}$ can distinguish a uniformly distributed element of S from a uniformly distributed element of $\{0,1\}^n$, where distinguishing means with a probability gap of at least $2^{-n/8}$.