Texts in Computational Complexity: Pseudorandom Generators

Oded Goldreich
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, Israel.

January 28, 2006

Indistinguishable things are identical.[1]

G.W. Leibniz (1646–1714)

A fresh view at the *question of randomness* has been 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 any 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.

At the extreme, this approach says that the question of whether the world is deterministic or allows for some free choice (which may be viewed as sources of randomness) is irrelevant. *What matters is how the world looks to us and to various computationally bounded devices.* That is, if some phenumenon looks random then we may just treat it as if it were random. Likewise, if we can generate sequences that cannot be told apart from the uniform distribution by any efficient procedure, then we can use these sequences in any efficient randomized application instead of the ideal random bits that are postulated in the design of this application.

Summary: 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*).

The archetypical case of pseudorandom generators refers to efficient generators that fool any feasible procedure; that is, the potential distinguisher is any probabilistic polynomial-time algorithm, which may be more complex than the generator itself (which, in turn, has time-complexity bounded by a fixed polynomial). These generators are called general-purpose, because their output can be safely used in an efficient application. Such (general-purpose) pseudorandom generators exist if and only if one-way functions exist.

[1]This is the *Principle of Identity of Indiscernibles*. Leibniz admits that counterexamples to this principle are conceivable but will not occur in real life because God is much too benevolent. We thus believe that he would have agreed to the theme of this text, which asserts that *indistinguishable things should be considered as identical*.

For purposes of derandomization one may use pseudorandom generators that are somewhat more complex than the potential distinguisher (which represents the algorithm to be derandomized). Following this approach, suitable pseudorandom generators, which can be constructed assuming the existence of problems in $\mathcal{E}$ that have no sub-exponential size circuits, yield a full derandomization of $\mathcal{BPP}$ (i.e., $\mathcal{BPP} = \mathcal{P}$).

It is also beneficial to consider pseudorandom generators that fool space-bounded distinguishers and generators that exhibit some limited random behavior (e.g., outputting a pair-wise independent or a small-bias sequence).

Contents

1 Introduction

The second half of this century has witnessed the development of three theories of randomness, a notion which has been puzzling thinkers for ages. The first theory (cf., [12]), initiated by Shannon [42], is rooted in probability theory and is focused at distributions that are not perfectly random. Shannon's Information Theory characterizes perfect randomness as the extreme case in which the *information contents* is maximized (i.e., 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., [28, 29]), due to Solomonov [43], Kolmogorov [25] and Chaitin [10], is rooted in computability theory and specifically in the notion of a universal language (equiv., universal machine or computing device). It measures the complexity of objects in terms of the shortest program (for a fixed universal machine) that generates the object. Like Shannon's theory, Kolmogorov Complexity is quantitative and perfect random objects appear as an extreme case. However, in this approach one may say that a single object, rather than a distribution over objects, is perfectly random. Still, Kolmogorov's approach is inherently intractable (i.e., Kolmogorov Complexity is uncomputable), and – by definition – one cannot (deterministically) generate strings of high Kolmogorov Complexity from short random seeds.

The third theory is rooted in complexity theory and is the focus of this text. This approach is explicitly aimed at providing a notion of randomness that nevertheless allows for an efficient (and deterministic) generation of random strings from shorter random seeds. The heart of this approach is the suggestion to view objects as equal if they cannot be told apart by any efficient procedure. Consequently, a distribution that cannot be efficiently distinguished from the uniform distribution will be considered as being 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 demonstrate this approach, let us consider the following mental experiment.

> Alice and Bob play "head or tail" 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 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.

We conclude that the randomness of an event is relative to the information and computing resources at our disposal. Thus, a natural concept 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 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 Section 6).

The foregoing discussion has focused at 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 is that the generation process has to be at least as efficient as the distinguisher (equiv., that the distinguisher is allowed at least as much resources as the generator). Coupled with the aforementioned strong notion of pseudorandomness, this yields the archetypical notion of pseudorandom generators – these operating in polynomial-time and producing sequences that are indistinguishable from uniform ones by *any* polynomial-time observer. Such (general-purpose) pseudorandom generators allow to reduced the randomness complexity of *any efficient application*, and are thus of great relevance to randomized algorithms and cryptography (see Section 3).

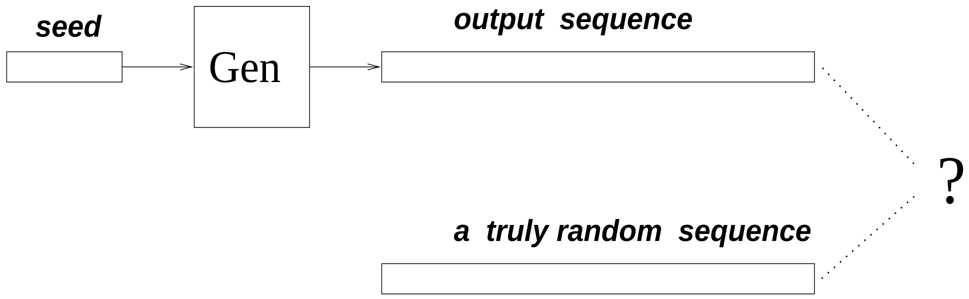


Figure 1: Pseudorandom generators – an illustration.

We stress that there are important reasons for considering also an alternative that seems less natural; 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 genewrator) 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 it tries to fool makes the task of designing pseudorandom generators easier, and enables derandomization results that are not known when using general-purpose pseudorandom generators. The usefulness of this approach is demonstrated in Sections 4 through 6.

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 saving in the randomness complexity of various applications. This saving is valuable because many applications are severly 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).

Organization. In Section 2 we present the general paradigm underlying the various notions of pseudorandom generators. The archetypical case of general-purpose pseudorandom generators is presented in Section 3. We then turn to the alternative notions of pseudorandom generators: Generators that suffice for the derandomization of complexity classes such as $\mathcal{BPP}$ are discussed in Section 4; Pseudorandom generators in the domain of space-bounded computations are discussed in Section 5; and special-purpose generators are discussed in Section 6. (For an alternative presentation, which focuses on general-purpose pseudorandom generators and provides more details on it, the reader is referred to [16, Chap. 3].)

> **Teaching note:** If you can afford teaching only one of the alternative notions of pseudorandom generators, then we suggest teaching the notion of general-purpose pseudorandom generators (presented in Section 3). Our reasons being that this notion is relevant to computer science at large and that the technical material is relatively simpler. The text is organized to facilitate this option.

Prerequisites: We assume a basic familiarity with elementary probability theory and randomized algorithms. In particular, standard conventions regarding random variables will be extensively used.

2 The General Paradigm

> **Teaching note:** 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 teacher that wishes to focus on one of the special cases may still use this section as a general motivation towards the specific definitions used later.

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*).

Stretch function: A necessary requirement from any notion of a pseudorandom generator is that it is a deterministic algorithm that stretches short strings, called seeds, into longer output sequences. Specifically, it 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). In some settings the specific stretch measure is immaterial (e.g., see Section 3.4).

Computational Indistinguishability: A necessary requirement from any notion of a pseudorandom generator is that it "fools" some non-trivial algorithms. That is, any algorithm taken from some class 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. Typically, we consider a class $\mathcal{D}$ of distinguishers and a class $\mathcal{F}$ of (threshold) functions, 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's

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

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.[2] The reader may think of such a distinguisher, D, as trying 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) 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. 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.

Complexity of Generation: The archetypical choice is that the generator has to work in polynomial-time (in length of its input – the seed). Other choices will be discussed as well. We note that placing no computational requirements on the generator (or, alternatively, putting very mild requirements such as a double-exponential running-time upper bound), yields "generators" that can fool any subexponential-size circuit family (see Exercise 30).

Notational conventions. We will consistently use k to denote the length of the seed of a pseudorandom generator, and $\ell(k)$ to denote the length of the corresponding output. In some cases, this makes our presentation a little more cumbersome (as a natural presentation may specify some other parameters and let the seed-length be a function of these). However, our choice has the advantage of focusing attention on the fundamental parameter of pseudorandom generation – 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.

Some instantiations of the general paradigm. Two important instantiations of the notion of pseudorandom generators relate to probabilistic polynomial-time observers.

1. General-purpose pseudorandom generators correspond to the case that 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.

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

[2] 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., the absolute difference between the above probabilities), 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).

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 pair-wise independence, small-bias, and hitting frequency.

3 General-Purpose Pseudorandom Generators

Randomness is playing an increasingly important role in computation: It is frequently used in the design of sequential, parallel and distributed algorithms, and it is of course central to cryptography. Whereas it is convenient to design such algorithms making free use of randomness, it is also desirable to minimize the usage of randomness in real implementations. Thus, general-purpose pseudorandom generators (as defined next) are a key ingredient in an "algorithmic tool-box" – they provide an automatic compiler of programs written with free usage of randomness into programs that make an economical use of randomness.

3.1 The basic definition

Loosely speaking, general-purpose pseudorandom generators are efficient (i.e., polynomial-time) deterministic programs that expand short randomly selected seeds into longer pseudorandom bit sequences, where the latter are defined as computationally indistinguishable from truly random sequences by *any* efficient (i.e., polynomial-time) algorithm. Thus, the distinguisher is more complex than the generator: The generator is a fixed algorithm working within *some fixed* polynomial-time, whereas a potential distinguisher is *any* algorithm that runs in polynomial-time. Thus, for example, the distinguisher *may* always run in time cubic in the running-time of the generator. Furthermore, to facilitate the development of this theory, we allow the distinguisher to be probabilistic (whereas the generator remains deterministic as above). We require that such distinguishers cannot tell the output of the generator from a truly random string of similar length, or rather that the difference that such distinguishers may detect (or sense) is negligible. Here a negligible function is one that vanishes faster than the reciprocal of any positive polynomial.

Definition 1 (general-purpose pseudorandom generator): *A deterministic polynomial-time algorithm G is called a* pseudorandom generator *if there exists a* stretch function, $\ell:\mathbb{N}\to\mathbb{N}$ (satisfying $\ell(k) > k$ for all k)*, such that for any probabilistic polynomial-time algorithm D, for any positive polynomial p, and for all sufficiently large k's*

$$|\,\mathsf{Pr}[D(G(U_k)) = 1] \;-\; \mathsf{Pr}[D(U_{\ell(k)}) = 1]\,| \;<\; \frac{1}{p(k)} \tag{2}$$

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 internal coin tosses of D.

Thus, Definition 1 is derived from the generic framework (presented in Section 2) by taking the class of distinguishers to be the set of all probabilistic polynomial-time algorithms, and taking the class of (noticeable) threshold functions to be the set of all functions that are the reciprocals of some positive polynomial.[3] The latter choice is naturally coupled with the association of efficient

[3]Definition 1 requires that the distinguishing gap of certain algorithms must be smaller than the reciprocal of any positive polynomial for all but finitely many k's. Such functions are called *negligible*; see Footnote 2. The notion of negligible probability is robust in the sense that an event which occurs with negligible probability occurs with negligible probability also when the experiment is repeated a "feasible" (i.e., polynomial) number of times.

computation with polynomial-time algorithms: An event that occurs with noticeable probability occurs almost always when the experiment is repeated a "feasible" (i.e., polynomial) number of times.

We note that Definition 1 does not make any requirement regarding the stretch function $\ell:\mathbb{N}\rightarrow\mathbb{N}$, except for the generic requirement that $\ell(k) > k$ for all k. Needless to say, the larger ℓ is the more useful is the pseudorandom generator. In Section 3.4 we show how to use any pseudorandom generator (even one with minimal stretch $\ell(k) = k+1$) in order to obtain a pseudorandom generator of any desired polynomial stretch function. But before going so, we rigorously discuss the "reduction in randomness" offered by pseudorandom generators, and the notion of computational indistinguishability underlying Definition 1.

3.2 The archetypical application

We note that "pseudo-random number generators" appeared with the first computers. However, typical implementations use generators that are not pseudorandom according to Definition 1. Instead, at best, these generators are shown to pass *some* ad-hoc statistical test (cf., [24]). We warn that the fact that a "pseudo-random number generator" passes some statistical tests, does not mean that it will pass a new test and that it will be good for a future (untested) application. Furthermore, the approach of subjecting the generator to some ad-hoc tests fails to provide general results of the form "for *all* practical purposes using the output of the generator is as good as using truly unbiased coin tosses." In contrast, the approach encompassed in Definition 1 aims at such generality, and in fact is tailored to obtain it: The notion of computational indistinguishability, which underlines Definition 1, covers all possible efficient applications guaranteeing that for all of them pseudorandom sequences are as good as truly random ones. Indeed, any efficient randomized algorithm maintains its performance when its internal coin tosses are substituted by a sequence generated by a pseudorandom generator. That is:

Construction 2 (typical application of pseudorandom generators): *Let G be a pseudorandom generator with stretch function $\ell:\mathbb{N}\rightarrow\mathbb{N}$. Let A be a probabilistic algorithm, and $\rho(n)$ denote a* (polynomial) *upper bound on its randomness complexity. Denote by $A(x,r)$ the output of A on input x and coin tosses sequence $r \in \{0,1\}^{\rho(|x|)}$. Consider the following randomized algorithm, denoted A_G:*

> *On input x, set $k = k(|x|)$ to be the smallest integer such that $\ell(k) \geq \rho(|x|)$, uniformly select $s \in \{0,1\}^k$, and output $A(x,r)$, where r is the $\rho(|x|)$-bit long prefix of $G(s)$.*

That is, $A_G(x,s) = A(x,G'(s))$, for $|s| = k(|x|) = \mathrm{argmin}_i\{\ell(i) \geq \rho(|x|)\}$, where $G'(s)$ is the $\rho(|x|)$-bit long prefix of $G(s)$.

Thus, using A_G instead of A, the randomness complexity is reduced from ρ to $\ell^{-1}\circ\rho$, while (as we show next) it is infeasible to find inputs (i.e., x's) on which the *noticeable behavior* of A_G is different from the one of A. For example, if $\ell(k) = k^2$, then the randomness complexity is reduced from ρ to $\sqrt{\rho}$. We stress that the pseudorandom generator G is *universal*; that is, it can be applied to reduce the randomness complexity of *any* probabilistic polynomial-time algorithm A.

Proposition 3 *Let A, ρ and G be as in Construction 2, and suppose that $\rho:\mathbb{N}\rightarrow\mathbb{N}$ is 1-1. Then, for every pair of probabilistic polynomial-time algorithms,* a finder F *and* a distinguisher D, *every positive polynomial p and all sufficiently long n's*

$$\sum_{x\in\{0,1\}^n} \mathsf{Pr}[F(1^n)=x]\cdot|\,\Delta_{A,D}(x)\,| \;<\; \frac{1}{p(n)} \tag{3}$$

where $\Delta_{A,D}(x) \stackrel{\rm def}{=} \Pr[D(x, A(x, U_{\rho(|x|)})) = 1] - \Pr[D(x, A_G(x, U_{k(|x|)})) = 1]$, *and the probabilities are taken over the* U_m*'s as well as over the coin tosses of* F *and* D.

Algorithm F represents a potential attempt to find an input x on which the output of A_G is distinguishable from the output of A. This "attempt" may be benign as in the case that a user employs algorithm A_G on inputs that are generated by some probabilistic polynomial-time application. However, the attempt may also be adversarial as in the case that a user employs algorithm A_G on inputs that are provided by a potentially malicious party. The potential distinguisher, denoted D, represents the potential use of the output of algorithm A_G, and captures the requirement that this output be as good as a corresponding output produced by A. Thus, D is given x as well as the corresponding output produced either by $A_G(x) \stackrel{\rm def}{=} A(x, U_{k(n)})$ or by $A(x) = A(x, U_{\rho(n)})$, and it is required that D cannot tell the difference. In the case that A is a probabilistic polynomial-time *decision procedure*, this means that it is infeasible to find an x on which A_G decides incorrectly (i.e., differently than A). In the case that A is a *search procedure for some NP-relation*, it is infeasible to find an x on which A_G outputs a wrong solution. For details, see Exercise 31.

Proof: The proposition is proven by showing that any triplet (A, F, D) violating the claim can be converted into an algorithm D' that distinguishes the output of G from the uniform distribution, in contradiction to the hypothesis. The key observation is that $\Delta_{A,D}(x)$ equals $\Pr[D(x, A(x, U_{\rho(n)})) = 1] - \Pr[D(x, A(x, G'(U_{k(n)}))) = 1]$, where $G'(s)$ is the $\rho(n)$-bit long prefix of $G(s)$. Details follow.

On input r (taken from either $U_{\ell(k(n))}$ or $G(U_{k(n)})$), algorithm D' first obtains $x \leftarrow F(1^n)$, where n can be obtained easily from $|r|$ (because ρ is 1-1 and $1^n \mapsto \rho(n)$ is computable via A). Next, D' obtains $y = A(x, r')$, where r' is the $\rho(|x|)$-bit long prefix of r. Finally D' outputs $D(x, y)$. Note that D' is implementable in probabilistic polynomial-time, and that

$$\begin{aligned} D'(U_{\rho(n)}) &= D(X_n, A(X_n, U_{\rho(n)}))\,, \text{ where } X_n \stackrel{\rm def}{=} F(1^n) \\ D'(G'(U_{k(n)})) &= D(X_n, A(X_n, G'(U_{k(n)})))\,, \text{ where } X_n \stackrel{\rm def}{=} F(1^n) \end{aligned}$$

It follows that $|\Pr[D'(U_{\ell(k(n))}) = 1] - \Pr[D'(G(U_{k(n)})) = 1]|$ equals $\mathsf{E}[\Delta_{A,D}(F(1^n))]$, which implies a weaker version of the proposition (referring to $\mathsf{E}[\Delta_{A,D}(F(1^n))]$ rather than to $\mathsf{E}[|\Delta_{A,D}(F(1^n))|]$).

In order to prove that $\mathsf{E}[|\Delta_{A,D}(F(1^n))|]$ (rather than to $\mathsf{E}[\Delta_{A,D}(F(1^n))]$) is negligible, we need to modify D' a little. We start by assuming, towards the contradiction, that $\mathsf{E}[|\Delta_{A,D}(F(1^n))|] > \varepsilon(n)$ for some non-negligible function ε. On input r (taken from either $U_{\ell(k(n))}$ or $G(U_{k(n)})$), the modified algorithm D' first obtains $x \leftarrow F(1^n)$, as before. Next, using a sample of size $\mathrm{poly}(n/\varepsilon(n))$, it approximates $p_U(x) \stackrel{\rm def}{=} \Pr[D(x, A(x, U_{\rho(n)}) = 1]$ and $p_G(x) \stackrel{\rm def}{=} \Pr[D(x, A(x, G'(U_{k(n)})) = 1]$ such that each probability is approximated to within a deviation of $\varepsilon(n)/8$ with negligible error probability (say, $\exp(-n)$). (Note that, so far, the actions of D' only depend on the length of its input r, which determines n.) If these approximations indicate that $p_U(x) \geq p_G(x)$ (equiv., that $\Delta_{A,D} \geq 0$) then D' outputs $D(x, A(x, r'))$ else it outputs $1 - D(x, A(x, r'))$, where r' is the $\rho(|x|)$-bit long prefix of r and we assume without loss of generality that the output of D is in $\{0,1\}$. It follows that

$$\begin{aligned} &\Pr[D'(U_{\rho(n)}) = 1 | F(1^n) = x] - \Pr[D'(G'(U_{k(n)})) = 1 | F(1^n) = x] \\ &\geq\ |p_U(x) - p_G(x)| - \frac{\varepsilon(n)}{2} - \exp(-n) \end{aligned}$$

where the $\varepsilon(n)/2$ term is due to the maximal typical deviation (i.e., $\varepsilon(n)/4$) of our approximation of $p_U(x) - p_G(x)$, and the $\exp(-n)$ term is due to the rare case that our approximation errs by more than $\varepsilon(n)/4$. Thus, $\Pr[D'(U_{\ell(k(n))}) = 1] - \Pr[D'(G(U_{k(n)})) = 1]$ is lower-bounded by $\mathsf{E}[|\Delta_{A,D}(F(1^n))|] - (\varepsilon(n)/2) - \exp(-n) > \varepsilon(n)/3$, and the proposition follows. ■

Conclusion. Analogous arguments are applied whenever one wishes to prove that an efficient randomized process (be it an algorithm as above or a multi-party computation) preserves its behavior when one replaces true randomness by pseudorandomness as defined above. Thus, given a pseudorandom generator with a large stretch function, *one can considerably reduce the randomness complexity in any efficient application.*

3.3 Computational Indistinguishability

In this section we spell-out (and study) the definition of computational indistinguishability that underlies Definition 1. The general definition of computational indistinguishability refers to *arbitrary* probability ensembles, where a probability ensemble is an infinite sequence of random variables $\{Z_n\}_{n\in\mathbb{N}}$ such that each Z_n ranges over strings of length bounded by a polynomial in n. We say that $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are computationally indistinguishable if for every feasible algorithm A the difference $d_A(n) \stackrel{\text{def}}{=} |\Pr[A(X_n)=1] - \Pr[A(Y_n)=1]|$ is a negligible function in n. That is:

Definition 4 (computational indistinguishability): *We say that the probability ensembles $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are* computationally indistinguishable *if for every probabilistic polynomial-time algorithm D, every positive polynomial p, and all sufficiently large n,*

$$|\Pr[D(X_n)=1] - \Pr[D(Y_n)=1]| \;<\; \frac{1}{p(n)} \tag{4}$$

where the probabilities are taken over the relevant distribution (i.e., either X_n or Y_n) *and over the internal coin tosses of algorithm D. The l.h.s. of Eq.* (4)*, when viewed as a function of n, is often called the* distinguishing gap of D *where $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ is understood from the context.*

That is, we can think of D as somebody who wishes to distinguish two distributions (based on a sample given to it), and think of 1 as D's verdict that the sample was drawn according to the first distribution. Saying that the two distributions are computationally indistinguishable means that if D is a feasible procedure then its verdict is not really meaningful (because the verdict is almost as often 1 when the input is drawn from the first distribution as when the input is drawn from the second distribution). We comment that the absolute value in Eq. (4) can be omitted without affecting the definition (see Exercise 32), and we will often do so without warning.

In Definition 1, we required that the probability ensembles $\{G(U_k)\}_{k\in\mathbb{N}}$ and $\{U_{\ell(k)}\}_{k\in\mathbb{N}}$ be computationally indistinguishable. Indeed, an important special case of Definition 4 is when one ensemble is uniform, and in such a case we call the other ensemble pseudorandom.

Non-triviality of Computational Indistinguishability. Clearly, any two probability ensembles that are statistically close[4] are computationally indistinguishable. Needless to say, this is a trivial case of computational indistinguishability, which is due to information theoretic reasons. In contrast, as noted in Section 2, there exist probability ensembles that are statistically far apart and yet are computationally indistinguishable (see Exercise 30). However, at least one of the probability

[4]Two probability ensembles, $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, are said to be statistically close if for every positive polynomial p and sufficient large n the variation distance between X_n and Y_n (i.e., $\frac{1}{2}\sum_z |\Pr[X_n = z] - \Pr[Y_n = z]|$) is bounded above by $1/p(n)$.

ensembles in Exercise 30 is *not* polynomial-time constructible. One non-trivial case of computational indistinguishability in which both ensembles are polynomial-time constructible is provided by the definition of pseudorandom generators (see Exercise 33). As we shall see (in Theorem 11), the existence of one-way functions implies the existence of pseudorandom generators, which in turn implies the existence of *polynomial-time constructible* probability ensembles that are statistically far apart and yet are computationally indistinguishable. We mention that this sufficient condition is also necessary (see Exercise 34).

Indistinguishability by Multiple Samples

The definition of computational indistinguishability (i.e., Definition 4) refers to distinguishers that obtain a single sample from one of the two probability ensembles (i.e., $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$). A more general definition refers to distinguishers that obtain several independent samples from such an ensemble.

Definition 5 (indistinguishability by multiple samples): *Let $s:\mathbb{N}\to\mathbb{N}$ be polynomially-bounded. Two probability ensembles, $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, are* computationally indistinguishable by $s(\cdot)$ samples *if for every probabilistic polynomial-time algorithm, D, every positive polynomial $p(\cdot)$, and all sufficiently large n's*

$$\left|\mathsf{Pr}\left[D(X_n^{(1)},...,X_n^{(s(n))})=1\right] - \mathsf{Pr}\left[D(Y_n^{(1)},...,Y_n^{(s(n))})=1\right]\right| < \frac{1}{p(n)}$$

where $X_n^{(1)}$ through $X_n^{(s(n))}$ and $Y_n^{(1)}$ through $Y_n^{(s(n))}$ are independent random variables, with each $X_n^{(i)}$ identical to X_n and each $Y_n^{(i)}$ identical to Y_n.

It turns out that in the most interesting cases, computational indistinguishability by a single sample implies computational indistinguishability by any polynomial number of samples. One such case is the case of polynomial-time constructible ensembles. We say that the ensemble $\{Z_n\}_{n\in\mathbb{N}}$ is polynomial-time constructible if there exists a polynomial-time algorithm S so that $S(1^n)$ and Z_n are identically distributed.

Proposition 6 *Suppose that $X \stackrel{\rm def}{=} \{X_n\}_{n\in\mathbb{N}}$ and $Y \stackrel{\rm def}{=} \{Y_n\}_{n\in\mathbb{N}}$ are both polynomial-time constructible, and s be a polynomial. Then, X and Y are computationally indistinguishable by a single sample if and only if they are computationally indistinguishable by $s(\cdot)$ samples.*

Clearly, for every polynomial s, computational indistinguishability by $s(\cdot)$ samples implies computational indistinguishability by a single sample. We now prove that, for efficiently constructible distributions, indistinguishability by a single sample implies indistinguishability by multiple samples. The proof provides a simple demonstration of a central proof technique, known as the *hybrid technique*.

Proof Sketch:[5] To prove that a sequence of independently drawn samples of one distribution is indistinguishable from a sequence of independently drawn samples from the other distribution, we consider *hybrid* sequences such that the $i^{\rm th}$ hybrid consists of i samples taken from the first distribution and the rest taken from the second distribution. The "homogeneous" sequences (which we wish to prove to be computational indistinguishable) are the extreme hybrids (i.e., the first and last hybrids considered above). The key observation is that distinguishing the extreme hybrids

[5] For more details see [16, Sec. 3.2.3].

(towards the contradiction hypothesis) means distinguishing neighboring hybrids, which in turn yields a procedure for distinguishing single samples of the two original distributions (contradicting the hypothesis that these two distributions are indistinguishable by a single sample). Details follow.

Suppose that D distinguishes $s(n)$ samples of one distribution from $s(n)$ samples of the other, with a distinguishing gap of $\delta(n)$. Denoting the $i^{\rm th}$ hybrid by H_n^i (i.e., $H_n^i = (X_n^{(1)}, ..., X_n^{(i)}, Y_n^{(i+1)}, ..., Y_n^{(s(n))})$), this means that D distinguishes the extreme hybrids (i.e., H_n^0 and $H_n^{s(n)}$) with gap $\delta(n)$. Then D distinguishes a random pair of neighboring hybrids (i.e., D distinguishes the $i^{\rm th}$ hybrid from the $i+1^{\rm st}$ hybrid, for a randomly selected i) with gap at least $\delta(n)/s(n)$. The reason being that

$$\begin{aligned}
& \mathsf{E}_{i\in\{0,...,s(n)-1\}}\left[\mathsf{Pr}[D(H_n^i)=1] - \mathsf{Pr}[D(H_n^{i+1})=1]\right] \\
&= \frac{1}{s(n)} \cdot \sum_{i=0}^{s(n)-1} \left(\mathsf{Pr}[D(H_n^i)=1] - \mathsf{Pr}[D(H_n^{i+1})=1]\right) \qquad (5) \\
&= \frac{1}{s(n)} \cdot \left(\mathsf{Pr}[D(H_n^0)=1] - \mathsf{Pr}[D(H_n^{s(n)})=1]\right) \;=\; \frac{\delta(n)}{s(n)}
\end{aligned}$$

Using D, we obtain a distinguisher D' of single samples: Given a single sample, D' selects $i \in \{0, ..., s(n)-1\}$ at random, generates i samples from the first distribution and $s(n)-i-1$ samples from the second distribution, and invokes D with the $s(n)$-samples sequence obtained when placing the input sample in location $i+1$. Thus, the construction of D' relies on the hypothesis that both proabbility ensembles are polynomial-time constructible. In analyzing D', observe that when the single sample (i.e., the input to D') is taken from the first (resp., second) distribution, algorithm D' invokes D on the $i+1^{\rm st}$ hybrid (resp., $i^{\rm th}$ hybrid). Thus, the distinguishing gap of D' is captured by Eq. (5), and the claim follows. □

The hybrid technique – a digest: The hybrid technique constitutes a special type of a "reducibility argument" in which the computational indistinguishability of *complex* ensembles is proven using the computational indistinguishability of *basic* ensembles. The actual reduction is in the other direction: efficiently distinguishing the basic ensembles is reduced to efficiently distinguishing the complex ensembles, and *hybrid* distributions are used in the reduction in an essential way. The following three properties of the construction of the hybrids play an important role in the argument:

1. *The extreme hybrids collide with the complex ensembles*: this property is essential because what we want to prove (i.e., the indistinguishability of the complex ensembles) relates to the complex ensembles.

2. *Neighboring hybrids are easily related to the basic ensembles*: this property is essential because what we know (i.e., the indistinguishability of the basic ensembles) relates to the basic ensembles. We need to be able to translate our knowledge (i.e., computational indistinguishability) of the basic ensembles to knowledge (i.e., computational indistinguishability) of any pair of neighboring hybrids. Typically, it is required to efficiently transform strings in the range of a basic distribution into strings in the range of a hybrid, so that the transformation maps the first basic distribution to one hybrid and the second basic distribution to the neighboring hybrid. (In the proof of Proposition 6, the hypothesis that both X and Y are polynomial-time constructible is instrumental for such an efficient transformation.)

3. *The number of hybrids is small* (i.e., polynomial): this property is essential in order to deduce the computational indistinguishability of extreme hybrids from the computational indistin-

guishability of each pair of neighboring hybrids. Typically, the provable "distinguishability gap" is inversely proportional to the number of hybrids. Indeed, see Eq. (5).

We remark that in the course of an hybrid argument, a distinguishing algorithm referring to the complex ensembles is being analyzed and even invoked on arbitrary hybrids. The reader may be annoyed of the fact that the algorithm "was not designed to work on such hybrids" (but rather only on the extreme hybrids). However, *an algorithm is an algorithm*: once it exists we can invoke it on inputs of our choice, and analyze its performance on arbitrary input distributions.

3.4 Amplifying the stretch function

Recall that the definition of pseudorandom generators (i.e., Definition 1) makes a minimal requirement regarding their stretch; that is, it is only required that the length of the output of such generators is longer than their input. Needless to say, we seek pseudorandom generators with a significant stretch. It turns out (see Construction 7) that pseudorandom generators of any stretch function and in particular of stretch $\ell_1(k) \stackrel{\text{def}}{=} k+1$, are easily converted into pseudorandom generators of any desired (polynomially bounded) stretch function, ℓ. (On the other hand, since pseudorandom generators are required (in Definition 1) to run in polynomial time, their stretch must be polynomially bounded.) Thus, when talking about the existence of pseudorandom generators, as in Definition 1, we may ignore the stretch function.

Construction 7 *Let G_1 be a pseudorandom generator with stretch function $\ell_1(k) = k+1$, and ℓ be any polynomially bounded stretch function that is polynomial-time computable. Let*

$$G(s) \stackrel{\text{def}}{=} \sigma_1\sigma_2\cdots\sigma_{\ell(|s|)} \tag{6}$$

where $x_0 = s$ and $x_i\sigma_i = G_1(x_{i-1})$, for $i = 1, ..., \ell(|s|)$. (That is, σ_i is the last bit of $G_1(x_{i-1})$ and x_i is the $|s|$-bit long prefix of $G_1(x_{i-1})$.)

Needless to say, G is polynomial-time computable and has stretch ℓ.

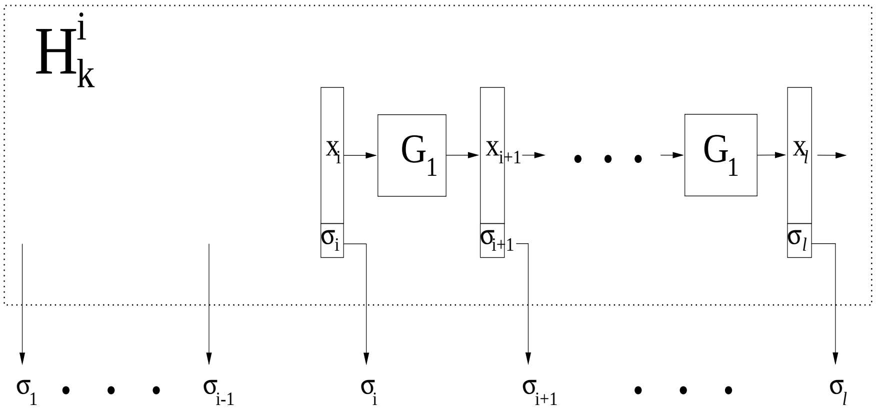


Figure 2: Analysis of stretch amplification – the i^{th} hybrid.

Proposition 8 *Let G_1 and G be as in Construction 7. Then G constitutes a pseudorandom generator.*

Proof Sketch:[6] The proposition is proven using the *hybrid technique*, presented and discussed in Section 3.3. Here (for $i = 0, ..., \ell(k)$) we consider the hybrid distributions H_k^i, defined by

$$H_k^i \stackrel{\text{def}}{=} U_i^{(1)} \cdot P_{\ell(k)-i}(U_k^{(2)}),$$

where $U_i^{(1)}$ and $U_k^{(2)}$ are independent uniform distributions (over $\{0,1\}^i$ and $\{0,1\}^k$, respectively), and $P_j(x)$ denotes the j-bit long prefix of $G(x)$. (See Figure 2.) The extreme hybrids (i.e., H_k^0 and H_k^k) correspond to $G(U_k)$ and $U_{\ell(k)}$, whereas distinguishability of neighboring hybrids can be worked into distinguishability of $G_1(U_k)$ and U_{k+1}. Details follow.

Suppose that algorithm D distinguishes H_k^i from H_k^{i+1} (with some gap $\delta(k)$). Denoting the first $|x|-1$ bits (resp., last bit) of x by $F(x)$ (resp., $L(x)$), we may write $P_j(s) \equiv (L(G_1(s)), P_{j-1}(F(G_1(s))))$ and

$$\begin{aligned}
H_k^i &= U_i^{(1)} \cdot P_{\ell(k)-i}(U_k^{(2)}) \\
&\equiv (U_i^{(1)}, L(G_1(U_k^{(2)})), P_{(\ell(k)-i)-1}(F(G_1(U_k^{(2)})))) \\
H_k^{i+1} &= U_{i+1}^{(1')} \cdot P_{\ell(k)-i-1}(U_k^{(2)}) \\
&\equiv (U_i^{(1)}, L(U_{k+1}^{(2')}), P_{(\ell(k)-i)-1}(F(U_{k+1}^{(2')}))).
\end{aligned}$$

Then, incorporating the generation of $U_i^{(1)}$ and the evaluation of $P_{\ell(k)-i-1}$ into the distinguisher D, we distinguish $(F(G_1(U_k^{(2)})), L(G_1(U_k^{(2)}))) \equiv G_1(U_k)$ from $(F(U_{k+1}^{(2')}), L(U_{k+1}^{(2')})) \equiv U_{k+1}$, in contradiction to the pseudorandomness of G_1. Specifically, on input $x \in \{0,1\}^{k+1}$, we uniformly select $r \in \{0,1\}^i$ and output $D(r \cdot L(x) \cdot P_{\ell(k)-i-1}(F(x)))$. Thus, the probability we output 1 on input $G_1(U_k)$ (resp., U_{k+1}) equals $\Pr[D(H_k^i) = 1]$ (resp., $\Pr[D(H_k^{i+1}) = 1]$). A final detail refers to the question which i to use. As usual (when the hybrid technique is used), a random i (in $\{0, ..., k-1\}$) will do. $\square$

3.5 Constructions

The constructions surveyed in this section "transform" computational difficulty, in the form of one-way functions, into generators of pseudorandomness. Recall that a *polynomial-time computable function* is called one-way if any efficient algorithm can invert it only with negligible success probability (see [16] for further discussion). We will actually use hard-core predicates of such functions, and refer the reader to their treatment in [16]. Loosely speaking, a *polynomial-time computable* predicate b is called a hard-core of a function f if any efficient algorithm, given $f(x)$, can guess $b(x)$ only with success probability that is negligible better than half. Recall that, for any one-way function f, the inner-product mod 2 of x and r is a hard-core of $f'(x,r) = (f(x), r)$. Finally, we get to the construction of pseudorandom generators.

Proposition 9 (A simple construction of pseudorandom generators): *Let b be a hard-core predicate of a polynomial-time computable 1-1 and length-preserving function f. Then, $G(s) \stackrel{\text{def}}{=} f(s) \cdot b(s)$ is a pseudorandom generator.*

Proof Sketch:[7] The $|s|$-bit long prefix of $G(s)$ is uniformly distributed, because f is 1-1 and onto $\{0,1\}^{|s|}$. Hence, the proof boils down to showing that distinguishing $f(s)b(s)$ from $f(s) \cdot \sigma$,

[6] For more details see [16, Sec. 3.3.3].
[7] For more details see [16, Sec. 3.3.4].

where σ is a random bit, yields contradiction to the hypothesis that b is a hard-core of f (i.e., that $b(s)$ is *unpredictable* from $f(s)$). Intuitively, such a distinguisher also distinguishes $f(s)b(s)$ from $f(s)\cdot\overline{b(s)}$, where $\overline{\sigma} = 1-\sigma$, but distinguishing $f(s)\cdot b(s)$ from $f(s)\cdot\overline{b(s)}$ yields an algorithm for predicting $b(s)$ based on $f(s)$. Details follow.

We start with any potential distinguisher D, and let

$$\delta(k) \stackrel{\text{def}}{=} \mathsf{Pr}[D(G(U_k))=1] - \mathsf{Pr}[D(U_{k+1})=1].$$

We may assume, without loss of generality, that $\delta(k)$ is non-negative (for infinitely many k's). Using $G(U_k) = f(U_k)\cdot b(U_k)$ and $U_{k+1} \equiv f(U_k)\cdot Z$, where $Z = b(U_k)$ with probability $1/2$ and $Z = \overline{b(U_k)}$ otherwise, we have

$$\mathsf{Pr}[D(f(U_k)b(U_k))=1] - \mathsf{Pr}[D(f(U_k)\overline{b(U_k)})=1] \;=\; 2\delta(k).$$

Consider an algorithm A that, on input y, uniformly selects $\sigma\in\{0,1\}$, invokes $D(y\sigma)$, and outputs σ if $D(y\sigma)=1$ and $\overline{\sigma}$ otherwise. Then

$$\begin{aligned}
\mathsf{Pr}[A(f(U_k)) = b(U_k)] &= \mathsf{Pr}[D(f(U_k)\cdot\sigma)=1 \wedge \sigma = b(U_k)] \\
&\quad + \mathsf{Pr}[D(f(U_k)\cdot\sigma)=0 \wedge \sigma = \overline{b(U_k)}] \\
&= \frac{1}{2}\cdot\big(\mathsf{Pr}[D(f(U_k)\cdot b(U_k))=1] \\
&\qquad + 1 \;-\; \mathsf{Pr}[D(f(U_k)\cdot\overline{b(U_k)})=1]\big)
\end{aligned}$$

which equals $(1+2\delta(k))/2$. The proposition follows. $\square$

Combining Proposition 9 and Construction 7, we obtain the following corollary.

Theorem 10 (A sufficient condition for the exsitemce of pseudorandom generators): *If there exists 1-1 and length-preserving one-way function then, for every polynomially bounded stretch function ℓ, there exists a pseudorandom generator of stretch ℓ.*

Digest. The key point in the proof of Proposition 9 is showing that the (rather obvious) unpredictability of the output of G implies its pseudorandomness. The fact that (next bit) unpredictability and pseudorandomness are equivalent, in general, is proven explicitly in the alternative proof of Theorem 10 provided next.

An alternative presentation. Let us take a closer look at the pseudorandom generators obtained by combining Construction 7 and Proposition 9. For a stretch function $\ell:\mathbb{N}\to\mathbb{N}$, a 1-1 one-way function f with a hard-core b, we obtain

$$G(s) \stackrel{\text{def}}{=} \sigma_1\sigma_2\cdots\sigma_{\ell(|s|)}\,, \tag{7}$$

where $x_0 = s$ and $x_i\sigma_i = f(x_{i-1})b(x_{i-1})$ for $i = 1,...,\ell(|s|)$. Denoting by $f^i(x)$ the value of f iterated i times on x (i.e., $f^i(x) = f^{i-1}(f(x))$ and $f^0(x) = x$), we rewrite Eq. (7) as follows

$$G(s) \stackrel{\text{def}}{=} b(s)\cdot b(f(s))\cdots b(f^{\ell(|s|)-1}(s))\,. \tag{8}$$

The pseudorandomness of G is established in two steps, using the notion of (next bit) unpredictability. An ensemble $\{Z_k\}_{k\in\mathbb{N}}$ is called unpredictable if any probabilistic polynomial-time machine obtaining a (random)[8] prefix of Z_k fails to predict the next bit of Z_k with probability non-negligibly higher than 1/2. Specifically, we need to establish the following two results.

1. A **general result** asserting that *an ensemble is pseudorandom if and only if it is unpredictable.* Recall that an ensemble is pseudorandom if it is computationally indistinguishable from a uniform distribution (over bit strings of adequate length).

   Clearly, pseudorandomness implies polynomial-time unpredictability, but here we actually need the other direction, which is less obvious. Still, using a hybrid argument, one can show that (next-bit) unpredictability implies indistinguishability from the uniform ensemble. For details see Exercise 38.

2. A **specific result** asserting that the ensemble $\{G(U_k)\}_{k\in\mathbb{N}}$ is unpredictable *from right to left.* Equivalently, $G'(U_n)$ is polynomial-time unpredictable (from left to right (as usual)), where $G'(s) = b(f^{\ell(|s|)-1}(s)) \cdots b(f(s)) \cdot b(s)$ is the reverse of $G(s)$.

   Using the fact that f induces a permutation over $\{0,1\}^n$, observe that the $(j+1)$-bit long prefix of $G'(U_k)$ is distributed identically to $b(f^j(U_k)) \cdots b(f(U_k)) \cdot b(U_k)$. Thus, an algorithm that predicts the $j+1^{\text{st}}$ bit of $G'(U_n)$ based on the j-bit long prefix of $G'(U_n)$ yields an algorithm that guesses $b(U_n)$ based on $f(U_n)$. For details see Exercise 40.

Needless to say, G is a pseudorandom generator if and only if G' is a pseudorandom generator (see Exercise 39). We mention that Eq. (8) is often referred to as the Blum-Micali Construction.[9]

A general condition for the existence of pseudorandom generators. Recall that given any one-way 1-1 length-preserving function, we can easily construct a pseudorandom generator. Actually, the 1-1 (and length-preserving) requirement may be dropped, but the currently known construction – for the general case – is quite complex.

Theorem 11 (On the existence of pseudorandom generators): *Pseudorandom generators exist if and only if one-way functions exist.*

To show that the existence of pseudorandom generators imply the existence of one-way functions, consider a pseudorandom generator G with stretch function $\ell(k) = 2k$. For $x, y \in \{0,1\}^k$, define $f(x,y) \stackrel{\text{def}}{=} G(x)$, and so f is polynomial-time computable (and length-preserving). It must be that f is one-way, or else one can distinguish $G(U_k)$ from U_{2k} by trying to invert and checking the result: Inverting f on its range distribution refers to the distribution $G(U_k)$, whereas the probability that U_{2k} has inverse under f is negligible.

The interesting direction of the proof of Theorem 11 is the construction of pseudorandom generators based on any one-way function. In general (when f may not be 1-1) the ensemble $f(U_k)$ may not be pseudorandom, and so Construction 9 (i.e., $G(s) = f(s)b(s)$, where b is a hard-core of f) cannot be used *directly.* One idea underlying the construction is to hash $f(U_k)$ to an almost uniform string of length related to its entropy, using Universal Hash Functions. (This is done after guaranteeing, that the logarithm of the probability mass of a value of $f(U_k)$ is typically close to

[8]For simplicity, we define unpredictability as referring to prefices of a random length (distributed uniformly in $\{0, ..., |Z_k| - 1\}$).

[9]Given the popularity of the term, we deviate from our convention of not specifying credits in the main text. Indeed, this construction originates in [8].

the entropy of $f(U_k)$.)[10] But "hashing $f(U_k)$ down to length comparable to the entropy" means shrinking the length of the output to, say, $k' < k$. This foils the entire point of stretching the k-bit seed. Thus, a second idea underlying the construction is to compensate for the $k - k'$ loss by extracting these many bits from the seed U_k itself. This is done by hashing U_k, and the point is that the $(k - k')$-bit long hash value does not make the inverting task any easier. Implementing these ideas turns out to be more difficult than it seems, and indeed an alternative construction would be most appreciated.

3.6 Non-uniformly strong pseudorandom generators

Recall that we said that truly random sequences can be replaced by pseudorandom ones without affecting any efficient computation. The specific formulation of this assertion, presented in Proposition 3, refers to randomized algorithms that take a "primary input" and use a secondary "random input" in their computation. Proposition 3 asserts that it is infeasible to find a primary input for which the replacement of a truly random secondary input by a pseudorandom one affects the final output of the algorithm in a noticeable way. This, however, does not mean that such primary inputs do not exist (but rather that they are hard to find). Consequently, Proposition 3 falls short of yielding a (worst-case)[11] "derandomization" of a complexity class such as $\mathcal{BPP}$. To obtain such results, we need a stronger notion of pseudorandom generators, presented next. Specifically, we need pseudorandom generators that can fool all polynomial-size circuits, and not merely all probabilistic polynomial-time algorithms.[12]

Definition 12 (strong pseudorandom generator – fooling circuits): *A deterministic polynomial-time algorithm G is called a* non-uniformly strong pseudorandom generator *if there exists a* stretch function, *$\ell:\mathbb{N}\to\mathbb{N}$, such that for any family $\{C_k\}_{k\in\mathbb{N}}$ of polynomial-size circuits, for any positive polynomial p, and for all sufficiently large k's*

$$|\,\mathsf{Pr}[C_k(G(U_k))=1] \; - \; \mathsf{Pr}[C_k(U_{\ell(k)})=1]\,| \; < \; \frac{1}{p(k)}$$

An alternative formulation is obtained by referring to polynomial-time machines that take advice. Using such pseudorandom generators, we can "derandomize" $\mathcal{BPP}$.

Theorem 13 (Derandomization of $\mathcal{BPP}$): *If there exists non-uniformly strong pseudorandom generators then $\mathcal{BPP}$ is contained in $\cap_{\varepsilon>0}\mathrm{D}\textsc{time}(t_\varepsilon)$, where $t_\varepsilon(n) \stackrel{\rm def}{=} 2^{n^\varepsilon}$.*

Proof Sketch: Given any $L \in \mathcal{BPP}$ and any $\varepsilon > 0$, we let A denote the decision procedure for L and G denote a non-uniformly strong pseudorandom generator stretching n^ε-bit long seeds into poly(n)-long sequences (to be used by A as secondary input when processing a primary input of

---

[10] Specifically, given an arbitrary one-way function f', one first constructs f by taking a "direct product" of sufficiently many copies of f'. For example, for $x_1, ..., x_{k^{2/3}} \in \{0,1\}^{k^{1/3}}$, we let $f(x_1, ..., x_{k^{2/3}}) \stackrel{\rm def}{=} f'(x_1), ..., f'(x_{k^{2/3}})$.

[11] Indeed, Proposition 3 yields an *average-case derandomization of* $\mathcal{BPP}$. In particular, for every polynomial-time constructible ensemble $\{X_n\}_{n\in\mathbb{N}}$, every $L \in \mathcal{BPP}$, and every $\varepsilon > 0$, there exists a randomized algorithm A' of randomness complexity $r_\varepsilon(n) = n^\varepsilon$ such that the probability that $A'(X_n) \neq L(X_n)$ is negligible. A corresponding deterministic ($\exp(r_\varepsilon)$-time) algorithm A'' can be obtained, as in the proof of Theorem 13, and again the probability that $A''(X_n) \neq L(X_n)$ is negligible, where here the probability is taken only over the distribution of the primary input (represented by X_n). In contrast, worst-case derandomization, as captured by the assertion $\mathcal{BPP} \subseteq \mathrm{D}\textsc{time}(2^{r_\varepsilon})$, requires that the probability that $A''(X_n) \neq L(X_n)$ is zero.

[12] Needless to say, strong pseudorandom generators in the sense of Definition 12 satisfy the basic definition of a pseudorandom generator (i.e., Definition 1); see Exercise 41.

length n). We thus obtain an algorithm $A' = A_G$ (as in Construction 2). We note that A and A' may differ in their decision on at most finitely many inputs, because otherwise we can use these inputs (together with A) to derive a (non-uniform) family of polynomial-size circuits that distinguishes $G(U_{n^\varepsilon})$ and $U_{\mathrm{poly}(n)}$, contradicting the the hypothesis regarding G. (Specifically, in terms of the proof of Proposition 3, the finder F consists of a non-uniform family of polynomial-size circuits that print the "problematic" primary inputs that are hard-wired in them, and the corresponding distinguisher D' is thus also non-uniform.) Incorporating the finitely many "bad" inputs into A', we derive a probabilistic polynomial-time algorithm that decides L while using randomness complexity n^ε.

Finally, emulating A' on each of the 2^{n^ε} possible random choices (i.e., seeds to G), we obtain a deterministic algorithm A'' as required. That is, let $A'(x,r)$ denote the output of algorithm A' on input x when using coins $r \in \{0,1\}^{n^\varepsilon}$. Then $A''(x)$ invokes $A'(x,r)$ on every $r \in \{0,1\}^{n^\varepsilon}$, and rules by majority. $\square$

We comment that stronger results regarding derandomization of $\mathcal{BPP}$ are presented in Section 4.

On constructing non-uniformly strong pseudorandom generators. Non-uniformly strong pseudorandom generators (as in Definition 12) can be constructed using any one-way function that is hard to invert by any non-uniform family of polynomial-size circuits, rather than by probabilistic polynomial-time machines. In fact, the construction in this case is simpler than the one employed in the uniform case (i.e., the construction underlying the proof of Theorem 11).

3.7 Other variants and a conceptual discussion

We first mention two stronger variants on the definition of pseudorandom generators, and conclude this section by highlighting various conceptual issues.

3.7.1 Stronger notions

The following two notion represent strengthening of the standard definition of pseudorandom generators (as presented in Definition 1). Non-uniform versions of these variants (strengthening Definition 12) are also of interest.

Fooling stronger distinguishers. One strengthening of Definition 1 amounts to explicitly quantifying the resources (and success gaps) of distinguishers. We chose to bound these quantities as a function of the length of the seed (i.e., k), rather than as a function of the length of the string that is being examined (i.e., $\ell(k)$). For a class of time bounds $\mathcal{T}$ (e.g., $\mathcal{T} = \{t(k) \stackrel{\rm def}{=} 2^{c\sqrt{k}}\}_{c\in\mathbb{N}}$) and a class of noticeable functions (e.g., $\mathcal{F} = \{f(k) \stackrel{\rm def}{=} 1/t(k) : t \in \mathcal{T}\}$), we say that a pseudorandom generator, G, is $(\mathcal{T},\mathcal{F})$-strong if for any probabilistic algorithm D having running-time bounded by a function in $\mathcal{T}$ (applied to k)[13], for any function f in $\mathcal{F}$, and for all sufficiently large k's

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

An analogous strengthening may be applied to the definition of one-way functions. Doing so reveals the weakness of the construction that underlies the proof of Theorem 11: It only implies that for some $\varepsilon > 0$ ($\varepsilon = 1/5$ will do), for any $\mathcal{T}$ and $\mathcal{F}$, the existence of "$(\mathcal{T},\mathcal{F})$-strong one-way

[13] That is, when examining a sequence of length $\ell(k)$ algorithm D makes at most $t(k)$ steps, where $t \in \mathcal{T}$.

functions" implies the existence of $(\mathcal{T}', \mathcal{F}')$-strong pseudorandom generators, where $\mathcal{T}' = \{t'(k) \stackrel{\text{def}}{=} t(k^\varepsilon)/\text{poly}(k) : t \in \mathcal{T}\}$ and $\mathcal{F}' = \{f'(k) \stackrel{\text{def}}{=} \text{poly}(k) \cdot f(k^\varepsilon) : f \in \mathcal{F}\}$. What we *would like* to have is an analogous result with $\mathcal{T}' = \{t'(k) \stackrel{\text{def}}{=} t(k)/\text{poly}(k) : t \in \mathcal{T}\}$ and $\mathcal{F}' = \{f'(k) \stackrel{\text{def}}{=} \text{poly}(k) \cdot f(k) : f \in \mathcal{F}\}$.

Pseudorandom Functions. Pseudorandom generators allow to efficiently generate long pseudorandom sequences from short random seeds. Pseudorandom functions (defined in [18]) are even more powerful: They allow *efficient direct access* to a huge pseudorandom sequence, which is not even feasible to scan bit-by-bit. Put in other words, pseudorandom functions can replace truly random functions in any efficient application (e.g., most notably in cryptography). We mention that pseudorandom functions can be constructed from any pseudorandom generator (see [18]) and found many applications in cryptography (see [16, 17]). Pseudorandom functions have been used to derive negative results in computational learning theory [46] and in complexity theory (cf., Natural Proofs [38]).

3.7.2 Conceptual Discussion

Whoever does not value preoccupation with thoughts, can skip this chapter.

Robert Musil, The Man without Qualities, Chap. 28

We highlight several conceptual aspects of the foregoing computational approach to randomness. Some of these aspects are common to other instantiation of the general paradigm (esp., the one presented in Section 4).

Behavioristic versus Ontological. The behavioristic nature of the computational approach to randomness is best demonstrated by confronting this approach with the Kolmogorov-Chaitin approach to randomness. Loosely speaking, a string is *Kolmogorov-random* if its length equals the length of the shortest program producing it. This shortest program may be considered the "true explanation" to the phenomenon described by the string. A Kolmogorov-random string is thus a string that does not have a substantially simpler (i.e., shorter) explanation than itself. Considering the simplest explanation of a phenomenon may be viewed as an ontological approach. In contrast, considering the effect of phenomena on certain objects, as underlying the definition of pseudorandomness, is a behavioristic approach. Furthermore, there exist probability distributions that are not uniform (and are not even statistically close to a uniform distribution) and nevertheless are indistinguishable from a uniform distribution (by any efficient method). Thus, distributions that are ontologically very different, are considered equivalent by the behavioristic point of view taken in the definition of computational indistinguishability.

A relativistic view of randomness. We have defined pseudorandomness in terms of its observer. Specifically, we have considered the class of efficient (i.e., polynomial-time) observers and defined as pseudorandom objects that look random to any observer in that class. In subsequent sections, we shall consider restricted classes of such observers (e.g., space-bounded polynomial-time observers and even very restricted observers that merely apply specific tests such as linear tests or hitting tests). Each such class of observers gives rise to a different notion of pseudorandomness. Furthermore, the general paradigm (of pseudorandomness) explicitly aims at distributions that are not uniform and yet are considered as such from the point of view of certain observers. Thus, our entire approach to pseudorandomness is relativistic and subjective (i.e., depending on the abilities of the observer).

Randomness and Computational Difficulty. Pseudorandomness and computational difficulty play dual roles: The general paradigm of pseudorandomness relies on the fact that putting computational restrictions on the observer gives rise to distributions that are not uniform and still cannot be distinguished from uniform. Thus, the pivot of the entire approach is the computational difficulty of distinguishing pseudorandom distributions from truly random ones. Furthermore, many of the constructions of pseudorandom generators rely on either conjectures or facts regarding computational difficulty (i.e., that certain computations that are hard for certain classes). For example, one-way functions were used to construct general-purpose pseudorandom generators (i.e., those working in polynomial-time and fooling all polynomial-time observers). Analogously, as we shall see in Section 4.3, the fact that parity function is hard for polynomial-size constant-depth circuits can be used to generate (highly non-uniform) sequences that fool such circuits.

Randomness and Predictability. The connection between pseudorandomness and unpredictability (by efficient procedures) plays an important role in the analysis of several constructions (cf. Sections 3.5 and 4.2). We wish to highlight the intuitive appeal of this connection.

4 Derandomization of time-complexity classes

Let us take a second look at the proof of Theorem 13: A pseudorandom generator was used to shrink the randomness complexity of a BPP-algorithm, and derandomization was achieved by scanning all possible seeds to the generator. A key observation regarding this process is that there is no point in insisting that the pseudorandom generator runs in time polynomial in its seed length. Instead, it suffices to require that the generator runs in time exponential in its seed length, because we are incurring such an overhead anyhow due to the scanning of all possible seeds. Furthermore, in this context, the running-time of the generator may be larger than the running time of the algorithm, which means that the generator need only fool distinguishers that take less steps than the generator. These considerations motivate the following definition.

4.1 Definition

Recall that in order to "derandomize" a probabilistic polynomial-time algorithm A, we first obtain a functionally equivalent algorithm A_G (as in Construction 2) that has (significantly) smaller randomness complexity. Algorithm A_G has to maintain A's input-output behavior on all (but finitely many) inputs. Thus, the set of the relevant distinguishers (considered in the proof of Theorem 13) is the set of all possible circuits obtained from A by hard-wiring each of the possible inputs. Such a circuit, denoted C_x, emulates the execution of algorithm A on input x, when using the circuit's input as the algorithm's internal coin tosses (i.e., $A(x,r) = C_x(r)$). Furthermore, the size of C_x is polynomial in the running-time of A on input x, and the length of the input to C_x is linear in the running-time of A (on input x).[14] For simplicity, let's say that the size of C_x is quadratic in the running-time of A on input x. Thus, the pseudorandom generator in use (i.e., G) needs to fool

[14]Indeed, if algorithm A is represented as a Turing machine then C_x has size that is at most quadratic (and in fact even almost-linear) in the running-time of A on input x, which in turn means that C_x has size that is at most quadratic (or almost linear) in the length of its own input. We note that most sources use the fictitious convention by which the circuit size equals the length of its input, which can be justified by considering a suitably padded input.

each of these possible circuits. Recalling that we may allow the generator to run in exponential time (in the length of its own input)[15], we arrive at the following definition.

Definition 14 (pseudorandom generator for derandomizing $\mathrm{BP}\textsc{time}(\cdot)$)[16]: *Let $\ell:\mathbb{N}\to\mathbb{N}$ be a 1-1 function. A* canonical derandomizer of stretch ℓ *is a deterministic algorithm G of time complexity upper-bounded by* $\mathrm{poly}(2^k\cdot\ell(k))$ *such that for every circuit D_k of size $\ell(k)^2$ it holds that*

$$|\,\mathsf{Pr}[D_k(G(U_k))=1]\;-\;\mathsf{Pr}[D_k(U_{\ell(k)})=1]\,|\;<\;\frac{1}{6} \tag{9}$$

The circuits D_k are potential distinguishers, which are given inputs of length $\ell(k)$. When seeking to derandomize an algorithm A of time-complexity t, the aforementioned $\ell(k)$-bit long inputs represent possible random-inputs of A when invoked on a generic (primary) input of length $n=t^{-1}(\ell(k))$. That is, letting $D_k(r)=A(x,r)$ for some choice of $x\in\{0,1\}^n$, where $|r|=t(n)=\ell(k)$, and Eq. (9) implies that $A_G(x)$ maintains the majority vote of $A(x)$. The straightforward deterministic emulation of A_G takes time $2^k\cdot(\mathrm{poly}(2^k\cdot\ell(k))+t(n))$, which is upper-bounded by $\mathrm{poly}(2^k\cdot\ell(k))=\mathrm{poly}(2^{\ell^{-1}(t(n))}\cdot t(n))$. The following proposition is easy to establish.

Proposition 15 *If there exists a canonical derandomizer of stretch ℓ then, for every time-constructible $t:\mathbb{N}\to\mathbb{N}$, it holds that $\mathrm{BP}\textsc{time}(t)\subseteq\mathrm{D}\textsc{time}(T)$, where $T(n)=\mathrm{poly}(2^{\ell^{-1}(t(n))}\cdot t(n))$.*

Proof Sketch: Just follow the proof of Theorem 13, noting that the adequate value of k (i.e., $k=\ell^{-1}(t(n))$) can be determined easily (e.g., by invoking $G(1^i)$ for $i=1,...,k$, using the fact that $\ell:\mathbb{N}\to\mathbb{N}$ is 1-1). Note that the complexity of the deterministic procedure is dominated by the 2^k invocations of $A_G(x,s)=A(x,G(s))$, where $s\in\{0,1\}^{\ell^{-1}(t(|x|))}$, and each of these invocations takes time $\mathrm{poly}(2^k\cdot\ell(k))+t(n)=\mathrm{poly}(2^{\ell^{-1}(t(n))}\cdot t(n))$. □

The goal. In light of Proposition 15, we seek canonical derandomizers with stretch that is as big as possible. The stretch cannot be super-exponential (in fact $\ell(k)=O(2^k)$), because a circuit of size $O(2^k\cdot\ell(k))$ may violate Eq. (9) (see Exercise 42) whereas for $\ell(k)=\omega(2^k)$ it holds that $O(2^k\cdot\ell(k))<\ell(k)^2$. Thus, our goal is to construct canonical derandomizer with stretch $\ell(k)=2^{\Omega(k)}$. Such canonical derandomizers will allow for a "full derandomization of $\mathcal{BPP}$":

Theorem 16 *If there exists a canonical derandomizer of stretch $\ell(k)=2^{\Omega(k)}$ for $\mathrm{BP}\textsc{time}(\cdot)$, then $\mathcal{BPP}=\mathcal{P}$.*

Proof: Using Proposition 15, we get $\mathrm{BP}\textsc{time}(t)\subseteq\mathrm{D}\textsc{time}(T)$, where $T(n)=\mathrm{poly}(2^{\ell^{-1}(t(n))}\cdot t(n))=\mathrm{poly}(t(n))$. ■

Reflections. We stress that a canonical derandomizer G was defined in a way that allows it to have time complexity t_G that is larger than the size of the circuits that it fools (i.e., $t_G(k)>\ell(k)^2$ is allowed). Furthermore, $t_G(k)>2^k$ was also allowed. Thus, if indeed $t_G(k)=2^{\Omega(k)}$ (as is the case in Section 4.2) then *$G(U_k)$ can be distinguished from $U_{\ell(k)}$ in time $2^k\cdot t_G(k)=\mathrm{poly}(t_G(k))$* (greater than $\ell(k)^2$), by trying all possible seeds. In contrast, for a general-purpose pseudorandom generator G (as discussed in Section 3) it holds that $t_G(k)=\mathrm{poly}(k)$, while *for every polynomial p it holds that $G(U_k)$ is indistinguishable from $U_{\ell(k)}$ in time $p(t_G(k))$.*

[15]Actually, in Definition 14 we allow the generator to run in time $\mathrm{poly}(2^k\ell(k))$, rather than $\mathrm{poly}(2^k)$. This is done in order not to rule out trivially generators of super-exponential stretch (i.e., $\ell(k)=2^{\omega(k)}$). However (see Exercise 42), the condition in Eq. (9) does not allow for super-exponential stretch, and so in retrospect the two formulations are equivalent (because $\mathrm{poly}(2^k\ell(k))=\mathrm{poly}(2^k)$ for $\ell(k)=2^{O(k)}$).

[16]Fixing a model of computation, we denote by $\mathrm{BP}\textsc{time}(t)$ the class of decision problems that are solvable by a randomized algorithm of time complexity t that has two-sided error 1/3.

4.2 Construction

The fact that canonical derandomizers are allowed to be more complex than the corresponding distinguisher makes *some* of the techniques of Section 3 inapplicable in the current context. On the other hand, the techniques developed below are inapplicable to Section 3. Amazingly enough, the pseudorandomness (or rather the next-bit unpredictability) of the following generators hold even when the "observer" is given the seed (capitalizing on the fact that the observer's time-complexity does not allow running the generator).

As in Section 3.5, the construction surveyed below transforms computational difficulty into pseudorandomness, except that here both computational difficulty and pseudorandomness are of a somewhat different form than in Section 3.5. Specifically, here we use Boolean predicates that are computable in exponential-time but are T-inapproximated for some exponential function T; that is, for constants $c, \varepsilon > 0$ and all but finitely many m, the (residual) predicate $f : \{0,1\}^m \to \{0,1\}$ is computable in time 2^{cm} but for any circuit C of size $2^{\varepsilon m}$ it holds that $\Pr[C(U_m) = f(U_m)] < \frac{1}{2} + 2^{-\varepsilon m}$. (Needless to say, $\varepsilon < c$.) Recall that such predicates exist under the assumption that $\mathcal{E}$ has (almost-everywhere) exponential circuit complexity. With these preliminaries, we turn to the construction of canonical derandomizers with exponential stretch.

Construction 17 (The Nisan-Wigderson Construction):[17] *Let $f : \{0,1\}^m \to \{0,1\}$ and $S_1, ..., S_\ell$ be a sequence of m-subsets of $\{1, ..., k\}$. Then, for $s \in \{0,1\}^k$, we let*

$$G(s) \stackrel{\text{def}}{=} f(s_{S_1}) \cdots f(s_{S_\ell}) \tag{10}$$

where s_S denotes the projection of s on the bit locations in $S \subseteq \{1, ..., |s|\}$; that is, for $s = \sigma_1 \cdots \sigma_k$ and $S = \{i_1, ..., i_m\}$, we have $s_S = \sigma_{i_1} \cdots \sigma_{i_m}$.

Letting k vary and $\ell, m : \mathbb{N} \to \mathbb{N}$ be functions of k, we wish G to be a canonical derandomizer and $\ell(k) = 2^{\Omega(k)}$. Obvious necessary conditions for this to happen include the requirement that the sets be distinct and hence $m(k) = \Omega(k)$; consequently, f must be computable in exponential-time. Furthermore, the sequence of sets $S_1, ..., S_{\ell(k)}$ must be constructible in poly(2^k) time. Intuitively, it is desirable to use a set system with small pairwise intersections (because this restricts the overlap among the various inputs to which f is applied), and a function f that is strongly inapproximable (i.e., T-inapproximible for some exponential function T). Interestingly, these conditions are essentially sufficient.

Theorem 18 (analysis of Construction 17): *Let $\alpha, \beta, \gamma, \varepsilon > 0$ be constants satisfying $\varepsilon > (2\alpha/\beta) + \gamma$, and $\ell, m, T :: \mathbb{N} \to \mathbb{N}$ satisfy $\ell(k) = 2^{\alpha k}$, $m(k) = \beta k$, and $T(n) = 2^{\varepsilon n}$. Suppose that the following two conditions hold:*

1. *There exists an exponential-time computable function $f : \{0,1\}^* \to \{0,1\}$ that is T-inapproximable.*[18]
2. *There exists an exponential-time computable function $S : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that $|S(k,i)| = m(k)$ for every k and $i = 1, ..., \ell(k)$, and $|S(k,i) \cap S(k,j)| \leq \gamma \cdot m(k)$ for every k and $i \neq j$.*

Then using G as defined in Construction 17, with $S_i = S(k,i)$, yields a canonical derandomizer with stretch ℓ.

[17] Given the popularity of the term, we deviate from our convention of not specifying credits in the main text. This construction originates in [33, 36].

[18] We say that $f : \{0,1\}^* \to \{0,1\}$ is (S,ρ)-inapproximable if for every family of S-size circuits $\{C_n\}_{n \in \mathbb{N}}$ and all sufficiently large n it holds that $\Pr[C(U_n) \neq f(U_n)] \geq \frac{\rho(n)}{2}$. We say that f is T-inapproximable if it is $(T, 1-(1/T))$-inapproximable.

For any $\gamma > 0$, a function S as in Condition 2 does exist (see Exercise 43), with some $m(k) = \Omega(k)$ and $\ell(k) = 2^{\Omega(k)}$. Combining such S with the worst-case to average-case reduction of [22] and using Theorem 18, we obtain a canonical derandomizer with exponential stretch based on the assumption that $\mathcal{E}$ has (almost-everywhere) exponential circuit complexity.[19] Combining this with Theorem 16, we get the first item of the following theorem.

Theorem 19 (Derandomization of BPP, revisited):

1. *Suppose that there exists a set $S \in \mathcal{E}$ having almost-everywhere exponential circuit complexity* (i.e., there exists a constant $\varepsilon > 0$ such that, for all but finitely many m's, any circuit that correctly decides S on $\{0,1\}^m$ has size at least $2^{\varepsilon m}$). *Then, $\mathcal{BPP} = \mathcal{P}$.*

2. *Suppose that for every polynomial p there exists a set $S \in \mathcal{E}$ having circuit complexity that is almost-everywhere greater than p. Then $\mathcal{BPP}$ is contained in $\cap_{\varepsilon>0}\mathrm{DTIME}(t_\varepsilon)$, where $t_\varepsilon(n) \stackrel{\text{def}}{=} 2^{n^\varepsilon}$.*

Part 2 is proved (in Exercise 47) by using a generalization of Theorem 18, which in turn is provided in Exercise 46. We note that Part 2 of Theorem 19 superseeds Theorem 13. The two parts of Theorem 19 exhibit two extreme cases: Part 1 (often referred to as the "high end") assumes an extremely strong circuit lower-bound and yields "full derandomization" (i.e., $\mathcal{BPP} = \mathcal{P}$), whereas Part 2 (often referred to as the "low end") assumes an extremely weak circuit lower-bound and yields weak but meaningfull derandomization. Intermediate results (relying on intermediate lower-bound assumptions) can be obtained analogous to Exercise 47, but tight trade-offs are obtained differently (cf., [45]).

Proof of Theorem 18: Using the time complexity bounds on f and S, it follows that G can be computed in exponential time. Out focus is on showing that $\{G(U_k)\}$ cannot be distinguished from $\{U_{\ell(k)}\}$ by circuits of size $\ell(k)^2$; that is, that G satisfies Eq. (9). In fact, we will prove that this holds for $G'(s) = s \cdot G(s)$; that is, G fools such circuits even if they are given the seed as auxiliary input. (Indeed, these circuits are smaller than the running time of G, and so they cannot just evaluate G on the given seed.)

We start with the intuition underlying the proof. As a warm-up suppose that the sets (i.e., $S(k,i)$'s) used in the construction are disjoint. In such a case (which is indeed impossible becuase $k < \ell(k) \cdot m(k)$), the pseudorandomness of $G(U_k)$ would follow easily from the inapproximablity of f, because in this case G consists of applying f to non-overlaping parts of the seed (see Exercise 44). In the actual construction being analyzed here, the sets (i.e., $S(k,i)$'s) are not disjoint but have relatively small pairwise intersection, which means that G applies f on parts of the seed that have relatively small overlap. Intuitively, such small overlaps guarantee that the values of f on the corresponding inputs are "computationally independent" (i.e., having the value of f at one input does not help to approximate the value of f at another input). This intuition will be backed by showing that the former values can be computed at a relatively small computational cost. With this intuition in mind, we now turn to the actual proof.

The proof that G' fools circuits of size $\ell(k)^2$ utilizes the relation between pseudorandomness and unpredictability. Specifically, as detailed in Exercise 45, any circuit that distinguishes $G'(U_k)$ from $U_{\ell(k)+k}$ with gap $1/6$, yields a next-bit predictor of similar size that succeeds in predicting the

[19] Specifically, starting with a function having circuit complexity at least $\exp(\varepsilon_0 m)$, we apply the worst-case to average-case reduction of [22], obtaining a T-inapproximble predicate for $T(m) = 2^{\varepsilon m}$, where the constant $\varepsilon \in (0, \varepsilon_0)$ depends on the constant ε_0. Next, we set $\gamma = \varepsilon/2$ and invoke Exercise 43, which determines $\alpha, \beta > 0$ such that $\ell(k) = 2^{\alpha k}$ and $m(k) = \beta k$. In fact, $\beta = \gamma/2$ and $\alpha = \gamma\beta/10$, hence $(2\alpha/\beta) + \gamma < \varepsilon$.

next bit with probability at least $\frac{1}{2}+\frac{1}{6\ell'(k)} > \frac{1}{2}+\frac{1}{7\ell(k)}$, where the $\ell'(k)=\ell(k)+k<(1+o(1))\ell(k)$ factor is introduced by the hybrid technique (cf. Eq. (5)). Furthermore, given the non-uniform setting of the current proof, we may fix a bit location $i+1$ for prediction, rather than analyzing the prediction at a random bit location. Indeed, $i \geq k$ must hold, because the first k bits of $G'(U_k)$ are uniformly distributed. In the rest of the proof, we transform such a predictor into a circuit that approximates f better than allowed by the hypothesis (regarding the inapproximability of f).

Assuming that a small circuit C' can predict the $i+1^{\rm st}$ bit of $G'(U_k)$, when given the previous i bits, we construct a small circuit C for approximating $f(U_{m(k)})$ on input $U_{m(k)}$. The point is that the $i+1^{\rm st}$ bit of $G'(s)$ equals $f(s_{S(k,j+1)})$, where $j=i-k\geq 0$, and so C' approximates $f(s_{S(k,j+1)})$ based on $s, f(s_{S(k,1)}),...,f(s_{S(k,j)})$, where $s\in\{0,1\}^k$ is uniformly distributed. This is the type of thing that we are after, except that the circuit we seek may only get $s_{S(k,j+1)}$ as input.

The first observation is that C' maintains its advantage when we fix the best choice for the bits of s that are not at bit locations $S_{j+1}=S(k,j+1)$ (i.e., the bits $s_{[k]\setminus S_{j+1}}$). That is, by an averaging argument, it holds that

$$\begin{aligned}&\max_{s'\in\{0,1\}^{k-m(k)}}\{\mathsf{Pr}_{s\in\{0,1\}^k}[C'(s,f(s_{S_1}),...,f(s_{S_j}))=f(s_{S_{j+1}})\,|\,s_{[k]\setminus S_{j+1}}=s']\}\\ &\geq\ p' \stackrel{\rm def}{=} \mathsf{Pr}_{s\in\{0,1\}^k}[C'(s,f(s_{S_1}),...,f(s_{S_j}))=f(s_{S_{j+1}})].\end{aligned}$$

Recall that by the hypothesis $p'>\frac{1}{2}+\frac{1}{7\ell(k)}$. Hard-wiring the fixed string s' into C', and letting $\pi(x)$ denote the (unique) string s satisfying $s_{S_{j+1}}=x$ and $s_{[k]\setminus S_{j+1}}=s'$, we obtain a circuit C'' that satisfies

$$\mathsf{Pr}_{x\in\{0,1\}^m}[C''(x,f(\pi(x)_{S_1}),...,f(\pi(x)_{S_j}))=f(x)]\ \geq\ p'.$$

The circuit C'' is almost what we seek. The only problem is that C'' gets as input not only x, but also $f(\pi(x)_{S_1}),...,f(\pi(x)_{S_j})$, whereas we seek an appproximator of $f(x)$ that only gets x.

The key observation is that each of the "missing" values $f(\pi(x)_{S_1}),...,f(\pi(x)_{S_j})$ depend only on a relatively small number of the bits of x. This fact is due to the hypothesis that $|S_t\cap S_{j+1}|\leq\gamma\cdot m(k)$ for $t=1,...,j$, which means that $\pi(x)_{S_t}$ is an $m(k)$-bit long string in which $m_t\stackrel{\rm def}{=}|S_t\cap S_{j+1}|$ bits are projected from x and the rest are projected from the *fixed* string s'. Thus, given x, the value $f(\pi(x)_{S_t})$ can be computed by a (trivial) circuit of size $\widetilde{O}(2^{m_t})$; that is, by a circuit implementing a look-up table on m_t bits. Using all these circuits (together with C''), we will obtain the desired approximator of f. Details follow.

We obtain the desired circuit C, which depends on the index j and the string s' that are fixed as in the foregoing analysis. On input $x\in\{0,1\}^m$, the circuit computes the values $f(\pi(x)_{S_1}),...,f(\pi(x)_{S_j})$, invokes C'' on input x and these values, and outputs the answer as a guess for $f(x)$. That is,

$$C(x)=C''(x,f(\pi(x)_{S_1}),...,f(\pi(x)_{S_j}))=C'(\pi(x),f(\pi(x)_{S_1}),...,f(\pi(x)_{S_j})).$$

By the foregoing analysis, $\mathsf{Pr}_x[C(x)=f(x)]\geq p'>\frac{1}{2}+\frac{1}{T(m)}$, where the second inequality is due to $T(m(k))=2^{\varepsilon m(k)}=2^{\varepsilon\beta k}\gg 2^{2\alpha k}\gg 7\ell(k)$. The size of C is upper-bounded by $\ell(k)^2+\ell(k)\cdot\widetilde{O}(2^{\gamma\cdot m(k)})\ll\widetilde{O}(\ell(k)^2\cdot 2^{\gamma\cdot m(k)})\ll T(m(k))$, where the second inequality is due to $T(m(k))=2^{\varepsilon m(k)}\gg\widetilde{O}(2^{2\alpha\cdot k+\gamma\cdot m(k)})$ and $\ell(k)=2^{\alpha k}$. Thus, we derived a contradiction to the hypothesis that f is T-inapproximable. ■

4.3 Variants and a conceptual discussion

The Nisan–Wigderson Construction (Construction 17) is actually a general framework, which can be instantiated in various ways. We start this section by briefly reviewing some of these instantiations and end it with a conceptual discussion regarding derandomization.

Derandomization of constant-depth circuits. Using (Construction 17 with) a different setting of parameters and the parity function in the role of the inapproximable predicate (i.e., inapproximable by "small" constant-depth circuits), one can obtain pseudorandom generators that fool "small" constant-depth circuits (see [33]). The analysis of the modified construction proceeds very much like the proof of Theorem 18. One important observation is that incorporating the (straightforward) circuits that compute $f(\pi(x)_{S_i})$ into the distinguishing circuit only increases its depth by two levels. The resulting pseudorandom generator, which use a seed of polylogarithmic length (equiv., $\ell(k) = \exp(k^{1/O(1)})$), can be used for derandomizing $\mathcal{RAC}^0$ (i.e., random $\mathcal{AC}^0$), analogously to Theorem 16. In other words, we can *deterministically* approximate, in quasi-polynomial-time and up-to an additive error, the fraction of inputs that satisfy a given (constant-depth) circuit. Specifically, for any constant d, given a depth-d circuit C, one can approximate the fraction of the inputs that satisfy C (i.e., cause C to evaluate to 1) to within any *additive constant error*[20] in time $\exp(\text{poly}(\log |C|))$, where the polynomial depends on d. Providing a deterministic polynomial-time aproximation, even in the case $d = 2$ (i.e., CNF/DNF formulae) is an open problem.

Derandomization of probabilistic proof systems. A different (and more surprising) instantiation of Construction 17 utilizes predicates that are hard for small circuits having oracle access to $\mathcal{NP}$. The result is a pseudorandom generator robust against two-move public-coin interactive proofs (which are as powerful as constant-round interactive proofs). The key observation is that the above proof provides a black-box procedure for approximating the underlying predicate when given oracle access to a distinguisher (and this procedure in valid also in case the distinguisher is a non-deterministic machine). Thus, under suitably strong (and yet plausible) assumptions, constant-round interactive proofs collapse to $\mathcal{NP}$. We note that a stronger result, which deviates from the foregoing framework, has been subsequently obtained (cf. [31]).

An even more radical instantiation of Construction 17 was used to obtain explicit constructions of randomness extractors (see [41]). In addition to the foregoing observation, one also utilizes the fact that the generator itself uses the predicate as a black-box.

A conceptual discussion regarding derandomization

Part 1 of Theorem 19 is often summarized by saying that (under some reasonable assumptions) *randomness is useless*. We believe that this interpretation is wrong even within the restricted context of traditional complexity classes, and is bluntly wrong if taken outside of the latter context. Let us elaborate.

Taking a closer look at the proof of (the underlying) Theorem 16, we note that a randomized algorithm A of time complexity t is emulated by a deterministic algorithm A' of time complexity $t' = \text{poly}(t)$. Further noting that $A' = A_G$ invokes A and the canonical derandomizer G for a number of times that must exceed t, we infer that $t' > t^2$ must hold. Thus, derandomization via (Part 1 of) Theorem 19 is not really for free.

More importantly, we note that derandomization is not possible in various distributed settings, when both parties may protect their conflicting interests by employing randomization. Notable

[20]We mention that in the special case of approximating the number of satisfying assignment of a DNF formula, *relative error* approximations can be obtained by employing a deterministic reduction to the case of additive constant error (see [15, Apdx. B.1.1]). Thus, using a pseudorandom generator that fools DNF formulae, we can deterministically obtain a relative (rather than additive) error approximation to the number of satisfying assignment in a given DNF formula.

examples include most cryptographic primitives (e.g., encryption) as well as most types of probabilistic proof systems (e.g., PCP). Additional settings where randomness makes a difference (either between impossibility and possibility or between prohibited and affordable cost) include distributed computing (see [6]), communication complexity (see [26]), parallel architectures (see [27]), sampling and property testing.

5 Space Pseudorandom Generators

In the previous two sections we have considered generators the output of which is indistinguishable by any efficient procedures. The latter were modeled by time-bounded computations; specifically, polynomial-time computations. A finer characterization of time-bounded computations is obtained by considering their space-complexity (i.e., restricting the space-complexity of time-bounded computations). In contrast to the definitions of pseudorandom generators that were considered in Sections 3 and 4, the existence of pseudorandom generators that fool space-bounded distinguishers can be established without relying on computational assumptions.

5.1 Definitional issues

Unfortunately, natural notions of space-bounded computations are quite subtle, especially when non-determinism or randomization are concerned. Two major issues are *time bounds* and *access to the random tape.*

1. Time bound: The question is whether or not one restricts the space-bounded machines to run in time-complexity that is at most exponential in the space-complexity.[21] Recall that such an upper-bound follows automatically in the deterministic case, and can be assumed without loss of generality in the non-deterministic case *but not in the randomized case.*

   Indeed, we do postulate the aforementioned time-bound.

2. Access to the random tape: The question is whether whether the space-bounded machine has one-way or two-way access to the randomness tape. (Allowing two-way access means that the randomness is recorded for free; that is, without being accounted for in the space-bound.) Recall that one-way access to the randomness tape corresponds to the natural model of on-line randomized machine (which determines its moves based on its internal coin tosses).

   Again, following most work in the area, we consider one-way access.[22]

In accordance with the resulting definition of randomized space-bounded computation, we consider space-bounded distinguishers that have a one-way access to the input sequence that they examine. Since all known constructions remain valid also when these distinguishers are non-uniform (and since non-uniform distinguishers arise anyhow in derandomization), we use this stronger notion here.[23]

[21] Alternatively, one can ask whether these machines must always halt or only halt with probability approaching 1. It can be shown that the only way to ensure "absolute halting" is to have time-complexity that is at most exponential in the space-complexity.

[22] We note that the fact that we restrict our attention to one-way access is instrumental in obtaining space-robust generators without making intractability assumptions. Analogous generators for two-way space-bounded computations would imply hardness results of a breakthrough nature in the area.

[23] We note that these non-uniform space-bounded distinguishers correspond to branching programs of width that is exponential in the space-bound. Furthermore, these branching programs read their input in a fixed predetermined order (which is determined by the designer of the generator).

In the context of non-uniform algorithms that have one-way access to their input, we may assume, without loss of generality, that the running-time of such algorithms equals the length of their input, denoted $\ell = \ell(k)$. Thus, we define a non-uniform machine of space $s:\mathbb{N}\to\mathbb{N}$ as a family, $\{D_k\}_{k\in\mathbb{N}}$, of directed layered graphs such that D_k has at most $2^{s(k)}$ vertices at each layer, and labeled directed edges from each layer to the next layer.[24] Each vertex has two (possibly parallel) outgoing directed edges, one labeled 0 and the other labeled 1, and there is a single vertex in the first layer of D_k. The result of the computation of such a machine, on an input of adequate length (i.e., length ℓ where D_k has $\ell+1$ layers), is defined as the vertex (in last layer) reached when following the sequence of edges that are labelled by the corresponding bits of the input. That is, on input $x = x_1 \cdots x_\ell$, for $i = 1, ..., \ell$, we move from the vertex reached in the i^{th} layer by using the outgoing edge labelled x_i (thus reaching a vertex in the $i+1^{\text{st}}$ layer). Using a fixed partition of the vertices of the last layer, this defines a natural notion of decision (by D_k); that is, we write $D_k(x) = 1$ if on input x machine D_k reached a vertex that belongs to the first part of the aforementioned partition.

Definition 20 (Indistinguishability by space-bounded machines):

- *For a non-uniform machine, $\{D_k\}_{k\in\mathbb{N}}$, and two probability ensembles, $\{X_k\}_{k\in\mathbb{N}}$ and $\{Y_k\}_{k\in\mathbb{N}}$, the function $d:\mathbb{N}\to[0,1]$ defined as*

  $$d(k) \stackrel{\text{def}}{=} |\mathsf{Pr}[D_k(X_k) = 1] - \mathsf{Pr}[D_k(Y_k) = 1]|$$

  is called the distinguishability-gap *of $\{D_k\}$ between the two ensembles.*

- *A probability ensemble, $\{X_k\}_{k\in\mathbb{N}}$, is called (s,ε)-*pseudorandom *if for any (non-uniform) $s(\cdot)$-space-bounded machine, the distinguishability-gap of the machine between $\{X_k\}_{k\in\mathbb{N}}$ and a uniform ensemble (of length $|X_k|$) is at most $\varepsilon(\cdot)$.*

- *A deterministic algorithm G of stretch function ℓ is called a (s,ε)-*pseudorandom generator *if the ensemble $\{G(U_k)\}_{k\in\mathbb{N}}$ is (s,ε)-pseudorandom.*

5.2 Two constructions

In contrast to the case of pseudorandom generators that fool time-bounded distinguishers, pseudorandom generators that fool space-bounded distinguishers can be established without relying on any computational assumption. The following two constructions exhibit two extreme cases of a general trade-off between the length of the seed and the stretch function of the generator.[25] We start with an attempt to maximize the stretch.

Theorem 21 (exponential stretch with quadratic length seed): *For every space constructible function $s:\mathbb{N}\to\mathbb{N}$, there exists a $(s, 2^{-s})$-pseudorandom generator of stretch function $\ell(k) = 2^{k/O(s(k))}$. Furthernore, the generator works in space linear in the length of the seed, and in time linear in the stretch function.*

[24] Note that the space bound of the machine is stated in terms of a parameter k, rather than in terms of the length of its input. In the sequel this parameter will be set to the length of a seed to a pseudorandom generator. We warn that our presentation here is indeed non-standard for this area. To compensate for this, we will also state the consequences in the standard format.

[25] These two results have been "interpolated" in [5]: There exists a parameterized family of space pseudorandom generators that includes both results as extreme special cases.

In other words, we have a generator that takes a random seed of length $k = O(t \cdot m)$ and produce sequences of length 2^t that look random to any m-space-bounded machine. In particular, using a random seed of length $k = O(m^2)$, one can produce sequences of length 2^m that look random to any m-space bounded machine. Thus, *one may replace random sequences used by any space-bounded computation, by sequences that are efficiently generated from random seeds of length quadratic in the space bound.* The common instantiation is for log-space machines. In §5.2.2, we apply Theorem 21 (and its underlying ideas) for the derandomization of space complexity classes such as $\mathcal{BPL}$ (i.e., the log-space analogue of $\mathcal{BPP}$).

We now turn to the case where one wishes to minimize the seed length. We warn that Theorem 22 only guarantees a subexponential distinguishing gap (rather than the exponential distinguishing gap guaranteed in Theorem 21). This warning is voiced because failing to recall this limitation has led to errors in the past.

Theorem 22 (polynomial stretch with linear length seed): *For any polynomial p and for $s(k) = k/O(1)$, there exists a $(s, 2^{-\sqrt{s}})$-pseudorandom generator of stretch function p. Furthermore, the generator works in linear-space and polynomial-time* (both stated in terms of the length of the seed).

In other words, we have a generator that takes a random seed of length $k = O(m)$ and produce sequences of length poly(m) that look random to any m-space-bounded machine. Thus, one may *convert any randomized computation utilizing polynomial-time and linear-space into a functionally equivalent randomized computation of similar time and space complexities that uses only a linear number of coin tosses.*

5.2.1 Overviews of the proofs of Theorems 21 and 22

In both cases, we describe the construction by starting with an adequate distinguisher and showing how the input distribution it examines can be modified (from the uniform one into a pseudorandom one) without the distinguisher noticing the difference.

Overview of the proof of Theorem 21.[26] Theorem 21 is proven by using the "mixing property" of pairwise independent hash functions. A family of functions H_n which map $\{0,1\}^n$ to itself is called *mixing* if for every pair of subsets $A, B \subseteq \{0,1\}^n$ for all but very few (i.e., $\exp(-\Omega(n))$ fraction) of the functions $h \in H_n$,

$$\mathsf{Pr}[U_n \in A \wedge h(U_n) \in B] \approx \frac{|A|}{2^n} \cdot \frac{|B|}{2^n} \tag{11}$$

where the approximation is up to an additive term of $\exp(-\Omega(n))$.

Given a $s(k)$-space distinguisher D_k as in Definition 20, we set $n \stackrel{\text{def}}{=} \Theta(s(k))$ and $\ell' \stackrel{\text{def}}{=} \ell(k)/n < 2^{s(k)}$, and consider an auxiliary "distinguisher" D'_k that is a directed layered graph with ℓ' layers and $2^{s(k)}$ vertices in each layer. In D'_k, each vertex has directed edges going to each vertex of the next layer and these edges are *labeled with* (possibly empty) *subsets of* $\{0,1\}^n$ such that these subsets form a partition of $\{0,1\}^n$. The graph D'_k simulates D_k in the obvious manner; that is, the computation of D'_k on an input of length $\ell(k) = \ell' \cdot n$ is defined by breaking the input into consecutive blocks of length n and following the path of edges that are labeled by the subsets containing the corresponding block. Now, for each pair of neighboring vertices, u and v (in layers i and $i+1$, respectively), consider the label, $L_{u,v} \subseteq \{0,1\}^n$, of the edge going from u to v. Similarly,

[26] A detailed proof appears in [34].

for a vertex w at layer $i+2$, we consider the label $L'_{v,w}$ of the edge from v to w. By Eq. (11), for all but very few few of $h \in H_n$,

$$\mathsf{Pr}[U_n \in L_{u,v} \wedge h(U_n) \in L'_{v,w}] \approx \mathsf{Pr}[U_n \in L_{u,v}] \cdot \mathsf{Pr}[U_n \in L'_{v,w}]$$

where "very few" and $\approx$ are as in Eq. (11). Thus, replacing the coins in the second block (i.e., used in transitions from layer $i+1$ to layer $i+2$) with the value of h applied to the outcomes of the coins used in the first block (i.e., in transitions from layer i to $i+1$), approximately maintains the probability that D'_k moves from u to w via v. The same (with "few" being $2^{3s(k)} \cdot \ell'$ times larger here)[27] holds for every triple of vertices in any three layers as in the foregoing discussion. The point is that we can use the same h in all these approximations. Thus, at the cost of extra $|h|$ random bits, we can reduce the number of true random coins used in transitions on D'_k by a factor of 2, without significantly affecting the final decision of D'_k. In other words, at the cost of extra $|h|$ random bits, we can effectively contract the distinguisher to half its length. That is, fixing a good h (i.e., one that provides a good approximation to all $2^{3s(k)} \cdot \ell'$ relevant pairs of sets), we can replace the 2-edge paths in D'_k by edges in a new distinguisher D''_k such that r is in the set that labels the edge u–w in D''_k if and only if, for some v, the string r is in the label of the edge u–v in D'_k and $h(r)$ is in the label of the edge v–w (also in D'_k).

Repeating the process for a logarithmic (in D'_k's length) number of times we obtain a distinguisher that only examines n bits, at which point we stop. In total, we have used $\log_2(\ell(k)/O(s(k))) < \log_2 \ell(k)$ random hash functions, which means that we can generate a sequence that fools the original D_k using a seed of length $n + \log_2 \ell(k) \cdot \log_2 |H_n|$ (see Exercise 48). Using $n = \Theta(s(k))$ and an adequate family H_n yields the claimed seed length of $O(s(k) \cdot \log_2 \ell(k)) = k$. $\square$

Overview of the proof of Theorem 22.[28] Theorem 22 is proven by using a suitable randomness extractor (as in [41]), which is indeed a much more powerful tool than hashing functions. The basic idea is that when D_k is at some distant layer, say at layer t, it typically "knows" little about the random choices that led it there. That is, D_k has only $s(k)$ bits of memory, which leaves out $t - s(k)$ bits of "uncertainty" (or randomness) regarding the previous moves. Thus, much of the randomness that led D_k to its current state may be "re-used" (or "recycled"). To re-use these bits we need to extract *almost* uniform distribution on strings of sufficient length out of the aforementioned distribution over $\{0,1\}^t$ that has entropy[29] at least $t - s(k)$. Furthermore, such an extraction requires some – yet relatively few – truly random bits. In particular, using $k' = \Omega(\log t)$ bits towards this end, the extracted bits are $\exp(-\Omega(k'))$ away from uniform.

One important point is how to use the foregoing argument repeatedly. Towards this end, we break the k-bit long seed into two parts, denoted $r' \in \{0,1\}^{k/2}$ and $(r_1, ..., r_{3\sqrt{k}})$, where $|r_i| = \sqrt{k}/6$, and set $n = k/3$. Intuitively, r' will be used for determining the first n steps, and it will be re-used (or recycled) together with r_i for determining the steps $i \cdot n + 1$ through $(i+1) \cdot n$. Looking at layer $i \cdot n$, we consider the information regarding r' that is known to D_k (at layer $i \cdot n$). Typically, the conditional distribution of r', given that we reached a specific vertex at layer $i \cdot n$ has (min-)entropy greater than $0.99 \cdot (t - s(k))$. Using r_i (as a seed of an extractor applied to r'), we can extract $0.9 \cdot ((k/2) - s(k) - o(k)) > k/3 = n$ bits that are almost-random with respect to D_k, and use these

[27] Note that "very few" means an $\exp(-\Omega(n))$ fraction and that $n = \Omega(s(k))$ and $\ell' < \exp(s(k))$.

[28] A detailed proof appears in [37].

[29] Actually, a stronger technical condition needs and can be imposed on the latter distribution. Specifically, with overwhlemingly high probability, at layer t machine D_k is at a vertex that can be reached in more than $2^{0.99 \cdot (t - s(k))}$ different ways. In this case, the distribution representing a random walk that reaches this vertex has min-entropy greater than $0.99 \cdot (t - s(k))$. The reader is referred to [41] for definitions of min-entropy and extractors.

bits for determining the next n steps. Hence, using k random bits we are produce a sequence[30] of length $(1+3\sqrt{k})\cdot n > k^{3/2}$ that fools machines of space bound, say, $s(k)=k/10$. That is, we obtained a $(s, 2^{-\Omega(\sqrt{s})})$-pseudorandom generator of stretch function $\ell(k)=k^{3/2}$.

To obtain an arbitrary polynomial stretch rather than a specific polynomial stretch (i.e., $\ell(k)=k^{3/2}$) we repeatedly apply an adequate composition, to be outlined next. Suppose that G_1 is a (s_1,ε_1)-pseudorandom generator of stretch function ℓ_1 that works in linear space, and similarly for G_2 with respect to (s_1,ε_1) and ℓ_2. Then, we consider the following construction of a generator G:

1. On input $s\in\{0,1\}^k$, obtain $G_1(s)$, and parse it into consecuetive blocks, each of length $m=s_1(k)/O(1)$, denoted $r_1,...,r_t$, where $t=\ell_1(k)/m$.
2. Output the $t\cdot\ell_2(m)$-bit long sequence $G_2(r_1)\cdots G_2(r_t)$.

Note that $|G(s)|=\ell_1(k)\cdot\ell_2(m)/m$, which for $s_1(k)=\Theta(k)$ yields $|G(s)|=\ell_1(k)\cdot\ell_2(\Omega(k))/O(k)$. We claim that G is a (s,ε)-pseudorandom generator, for $s(k)=\min(s_1(k)/2, s_2(\Omega(s_1(k)))$ and $\varepsilon(k)=\varepsilon_1(k)+\ell_1(k)\cdot\varepsilon_2(\Omega(s_1(k))$. The proof uses a hybrid argument, which focuses on the intermediate distribution $G_2(U_m^{(1)})\cdots G_2(U_m^{(t)})$. The key claim is that the intermediate distribution is $(s_1/2,\varepsilon_1)$-indistinguishable from $G(U_k)$, and it is proven by converting a potential distinguisher into a distinguisher of $U_m^{(1)}\cdots U_m^{(t)}$ and $G_1(U_k)$ by invoking G_2 on the corresponding m-bit long blocks (of the $\ell(k)$-bit long input). For this reason, it crucial that G_2 can be evaluate on m-bit long strings using space at most $s_1(k)/2$, which is guaranteed by our setting of $m=s_1(k)/O(1)$ and the hypothesis that G_2 works in linear space. $\square$

5.2.2 Derandomization of space-complexity classes

As a direct application of Theorem 21, we obtain that $\mathcal{BPL}\subseteq$ DSPACE($\log^2$), where $\mathcal{BPL}$ denotes the log-space analogue of $\mathcal{BPP}$. (Recall that $\mathcal{NL}\subseteq$ DSPACE($\log^2$), but it is not known whether or not $\mathcal{BPL}\subseteq\mathcal{NL}$.)[31] A stronger derandomization result can be obtained by a finer analysis of the proof of Theorem 21.

Theorem 23 *$\mathcal{BPL}\subseteq\mathcal{SC}$, where $\mathcal{SC}$ denotes the class of decision problems that can be solved by a deterministic machine that runs in polynomial-time and polylogarithmic-space.*

Thus, $\mathcal{BPL}\supseteq\mathcal{RL}$ is placed in a class not known to contain $\mathcal{NL}$. Another such result was subsequently obtained in [40]: Randomized log-space can be simulated in deterministic space $o(\log^2)$; specifically, in space $\log^{3/2}$. We mention that the archetypical problem of $\mathcal{RL}$ has been recently proved to be in $\mathcal{L}$ (see [39]).

Overview of the proof of Theorem 23.[32] Looking at the proof of Theorem 21, we note that the question of whether or not a specific hash function $h\in H_n$ is good for a specific D'_k can be determined in space that is linear in $n=|h|/2$ and logarithmic in the size of D'_k. Indeed, the time complexity of this decision procedure is exponential in its space complexity. It follows that we can find a good $h\in H_n$, for a given D'_k, within these complexities (by scanning through all possible $h\in H_n$). Once a good h is found, we can also construct the corresponding graph D''_k (in which

[30] Specifically, using an extractor of the form Ext : $\{0,1\}^{\sqrt{k}/6}\times\{0,1\}^{k/2}\to\{0,1\}^{k/3}$, we map the seed $(r', r_1, ..., r_{3\sqrt{k}})$ to the output sequence $(r', \mathrm{Ext}(r_1, r'), ..., \mathrm{Ext}(r_{3\sqrt{k}}, r'))$.

[31] Indeed, the log-space analogue of $\mathcal{RP}$, denoted $\mathcal{RL}$, is contained in $\mathcal{NL}\subseteq$ DSPACE($\log^2$), and thus the fact that Theorem 21 implies $\mathcal{RL}\subseteq$ DSPACE($\log^2$) is of no interest.

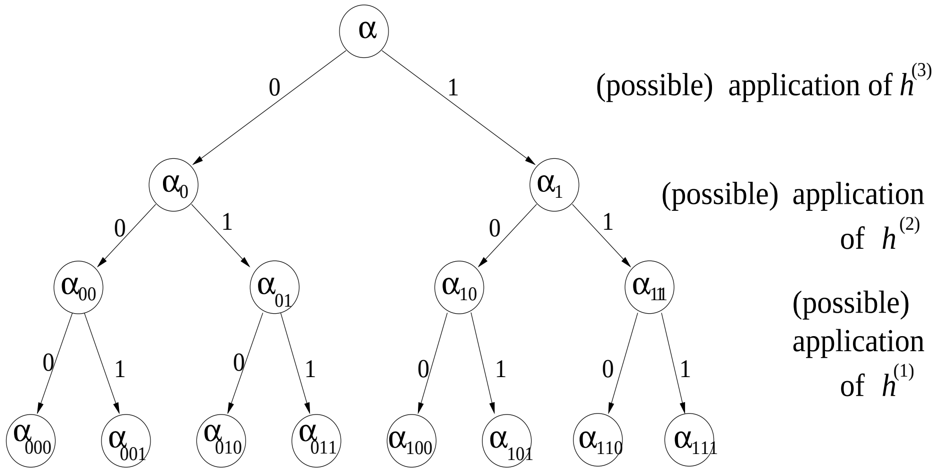
[32] A detailed proof appears in [35].

edges represent 2-edge paths in D'_k), again within the same complexity. Actually, it will be more instructive to note that we can determine a step (i.e., an edge-traversal) in D''_k by making two steps (edge-traversals) in D'_k.

The key claim is that the entire process of finding a sequence of $t \stackrel{\text{def}}{=} \log_2 \ell'(k)$ good hash functions can be performed in space $t \cdot O(n + \log |D_k|) < O(n + \log |D_k|)^2$ and time $\text{poly}(2^n \cdot |D_k|)$; that is, the time complexity is sub-exponential in the space complexity (i.e., the time complexity is significantly smaller than than the generic bound of $\exp(O(n + \log |D_k|)^2))$. Starting with $D_k^{(1)} = D'_k$, we find a good (for $D_k^{(1)}$) hashing function $h^{(1)} \in H_n$, which defines $D_k^{(2)} = D''_k$. Having found (and stored) $h^{(1)}, ..., h^{(i)} \in H_n$, which determine $D_k^{(i+1)}$, we find a good hashing function $h^{(i+1)} \in H_n$ for $D_k^{(i+1)}$ by emulating pairs of edge-traversals on $D_k^{(i+1)}$. Indeed, a key point is that we do *not* construct the sequence of graphs $D_k^{(2)}, ..., D_k^{(i+1)}$, but rather emulate an edge-traversal in $D_k^{(i+1)}$ by making 2^i edge-traversals in D'_k, using $h^{(1)}, ..., h^{(i)}$: The (edge-traversal) move $\alpha \in \{0,1\}^n$ starting at vertex v of $D_k^{(i+1)}$ translates to a sequence of 2^i moves starting at vertex v of D'_k, where the moves are determined by

$$\alpha, h^{(1)}(\alpha), h^{(2)}(\alpha), h^{(1)}(h^{(2)}(\alpha)), ..., h^{(1)}(h^{(2)}(\cdots h^{(i)}(\alpha)\cdots)).$$

(See Figure 3.) Thus, for $n = \Theta(\log |D'_k|)$, given D'_k and a pair (u, v) of source and sink in D'_k (which reside in the first and last layer, respectively), we can (deterministically) approximate the probability that a random walk starting at u reaches v in $O(\log |D'_k|)^2$-space and $\text{poly}(|D'_k|)$-time. The approximation can be made accurate up to a factor of $1 \pm (1/\text{poly}(|D'_k|))$. We conclude the proof by recalling the connection between such an approximation and the derandomization of $\mathcal{BPL}$.



The output of the generator (on seed α) consists of the concatanation of the strings denoted $\alpha_{0^i}, ..., \alpha_{1^i}$, appearing in the leaves of the tree. For every $x \in \{0,1\}^*$ it holds that $\alpha_{x0} = \alpha_x$ and $\alpha_{x1} = h^{(i-|x|)}(\alpha_x)$. In particular, for $i = 3$, we have $\alpha_{011} = h^{(1)}(\alpha_{01})$, which equals $h^{(1)}(\alpha_{01}) = h^{(1)}(h^{(2)}(\alpha))$, where $\alpha = \alpha_\lambda$.

Figure 3: Derandomization of $\mathcal{BPL}$ – the generator for $i = 3$.

The computation of a log-space probabilistic machine M, on input x, can be represented by a directed layer graph $G_{M,x}$ of size $\text{poly}(|x|)$. Specifically, the probability that M accepts x equals the probability that a random walk starting at the single vertex of the first layer of $G_{M,x}$ reaches some vertex in the last layer that represents an accepting configuration. Setting $k = \Theta(\log|x|)$ and $n = \Theta(k)$, the graph $G_{M,x}$ coincides with the graph D_k referred to at the beginning of the proof of Theorem 21, and D'_k is obtained from D_k by an "n-layer contraction" (see ibid.). Combining this with the foregoing analysis, we conclude that the probability that M accepts x can be deterministically approximated in $O(\log|x|)^2$-space and poly$(|x|)$-time. The theorem follows. $\square$

6 Special Purpose Generators

In this section we consider even weaker types of pseudorandom generators, producing sequences that can fool only very restricted types of distinguishers. Still, such generators have many applications in complexity theory and in the design of algorithms. (These applications will only be mentioned briefly.)

Our choice is to start with the simplest of these generators: the pairwise-independent generator, and its generalization to t-wise independence for any $t \geq 2$. Such generators perfectly fool any distinguisher that only observe t locations in the output sequence. This leads naturally to almost pairwise (or t-wise) independence generators, which also fool (albeit non-perfectly) such distinguishers. The latter generators are implied by a stronger class of generators, which is of independent interest: the small-bias generators. Small-bias generators fool any linear test (i.e., any distinguisher that merely considers the XOR of some fixed locations in the input sequence). We then turn to the Expander Random Walk Generator: this generator produces a sequence of strings that hit any dense subset of strings with probability that is close to the hitting probability of a truly random sequence. Related notions such as samplers, dispersers, and extractors are treated elsewhere (e.g., see [15] and [41], respectively).

Comment regarding our parameterization: To maintain consistency with prior sections, we continue to present the generators in terms of the seed length, denoted k. Since this is not the common presentation for most results presented in the sequel, we provide (in footnotes) the common presentation in which the seed length is determined as a function of other parameters.

6.1 Pairwise-Independence Generators

Pairwise (resp., t-wise) independence generators fool tests that inspect only two (resp., t) elements in the output sequence of the generator. Such loacl tests are indeed very restricted, yet they arise naturally in many settings. For example, such a test corresponds to a probabilistic analysis (of a procedure) that only relies on the pairwise independence of certain choices made by the procedure. We also mention that, in some natural range of parameters, pairwise independent sampling is as good as sampling by totally independent sample points.

A t-wise independence generator of block-size $b:\mathbb{N}\to\mathbb{N}$ (and stretch function ℓ) is an efficient deterministic algorithm (e.g., one that works in time polynomial in the output length) that expands a k-bit long random seed into a sequence of $\ell(k)/b(k)$ strings, each of length $b(k)$, such that any t blocks are uniformly and independently distributed in $\{0,1\}^{t\cdot b(k)}$. In case $t = 2$, we call the generator pairwise independent. We note that this condition holds even if the inspected t blocks are selected adaptively (see Exercise 49)

6.1.1 Constructions

In the first construction, we refer to $\mathrm{GF}(2^{b(k)})$, the finite field of $2^{b(k)}$ elements, and associate its elements with $\{0,1\}^{b(k)}$.

Proposition 24 (t-wise independence generator):[33] *Let t be a fixed integer and $b,\ell,\ell':\mathbb{N}\to\mathbb{N}$ such that $b(k)=k/t$, $\ell'(k)=\ell(k)/b(k)>t$ and $\ell'(k)\le 2^{b(k)}$. Let $\alpha_1,...,\alpha_{\ell'(k)}$ be distinct elements of this field. For $s_0,s_1,...,s_{t-1}\in\{0,1\}^{b(k)}$, let*

$$G(s_0,s_1,...,s_{t-1}) \stackrel{\rm def}{=} \left(\sum_{j=0}^{t-1} s_j\alpha_1^j \,,\, \sum_{j=0}^{t-1} s_j\alpha_2^j \,,...,\, \sum_{j=0}^{t-1} s_j\alpha_{\ell'(k)}^j\right) \tag{12}$$

where the arithmetic is that of $\mathrm{GF}(2^{b(k)})$. Then, G is a t-wise independence generator of block-size b and stretch ℓ.

That is, given a seed that consists of t elements of $\mathrm{GF}(2^{b(k)})$, the generator outputs a sequence of $\ell'(k)$ such elements. To make the above generator totally explicit, we need an explicit representation of $\mathrm{GF}(2^{b(k)})$, which requires an irreducible polynomial of degree $b(k)$ over $\mathrm{GF}(2)$. For specific values of $b(k)$, a good representation does exist: Specifically, for $d\stackrel{\rm def}{=}b(k)=2\cdot 3^e$ (with e being an integer), the polynomial $x^d+x^{d/2}+1$ is irreducible over $\mathrm{GF}(2)$. The proof of Proposition 24 is left as an exercise (see Exercise 50).

An alternative construction for the case of $t=2$ is obtained by using (random) affine transformations (as possible seeds). In fact, better performance (i.e., shorter seed length) is obtained by using affine transformations defined by Toeplitz matrices. A Toeplitz matrix is a matrix with all diagonals being homogeneous; that is, $T=(t_{i,j})$ is a Toeplitz matrix if $t_{i,j}=t_{i+1,j+1}$, for all i,j. Note that a Toeplitz matrix is determined by its first row and first column (i.e., the values of $t_{1,j}$'s and $t_{i,1}$'s).

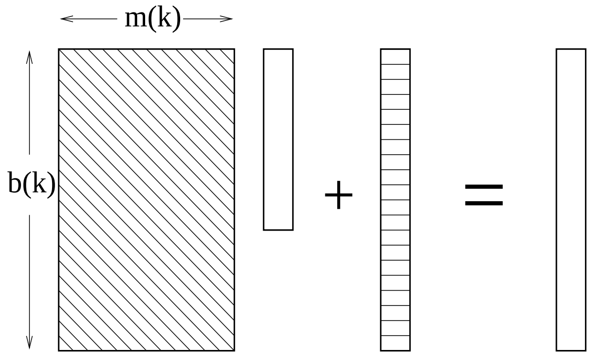


Figure 4: An affine transformation defined by a Toeplitz matrix.

Proposition 25 (Alternative pairwise independence generator, see Figure 4):[34] *Let $b,\ell,\ell',m:\mathbb{N}\to\mathbb{N}$ such that $\ell'(k)=\ell(k)/b(k)$ and $m(k)=\lceil\log_2\ell'(k)\rceil=k-2b(k)+1$. Associate $\{0,1\}^n$*

[33] The common parameterization of t-wise independence generator is as follows. Given parameters b and $\ell'\le 2^b$, and a uniformly distributed seed of length $t\cdot b$, one efficiently and deterministically generates a random sequence of ℓ' strings, each of length b, that are t-wise independent.

[34] The common parameterization of this pairwise independence generator is as follows. Given parameters b and ℓ', and a uniformly chosen seed of length $2b+\lceil\log_2\ell'\rceil-1$, one efficiently and deterministically generates a random sequence of ℓ' strings, each of length b, that are pairwise independent.

with the n-dimensional vector space over GF(2)*, and let $v_1, ..., v_{\ell'(k)}$ be distinct vectors in the $m(k)$-dimensional vector space over* GF(2)*. For $s \in \{0,1\}^{b(k)+m(k)-1}$ and $r \in \{0,1\}^{b(k)}$, let*

$$G(s,r) \stackrel{\text{def}}{=} (T_s v_1 + r \, , \, T_s v_2 + r \, , ..., \, T_s v_{\ell'(k)} + r) \tag{13}$$

where T_s is an $b(k)$-by-$m(k)$ Toeplitz matrix specified by the string s. Then G is a pairwise independence generator of block-size b and stretch ℓ.

That is, given a seed that represents an affine transformation defined by an $b(k)$-by-$m(k)$ Toeplitz matrix, the generator outputs a sequence of $\ell'(k) \leq 2^{m(k)}$ strings, each of length $b(k)$. Note that $m(k) = k - 2b(k) + 1$, and that the stretching property requires $\ell'(k) > k/b(k)$. The proof of Proposition 25 is left as an exercise (see Exercise 51).

A stronger notion of efficient generator. We note that the aforementioned constructions satisfy a stronger notion of efficient generation, which is useful in several applications. Specifically, there exists a polynomial-time algorithm that given a seed, $s \in \{0,1\}^k$, and a block location $i \in [\ell'(k)]$ (in bianry), outputs the i^{th} block of the corresponding output (i.e., the i^{th} block of $G(s)$).

6.1.2 Applications

Pairwise independence generators do suffice for a variety of applications (cf., [47, 30]). In particular, we mention the application to sampling, and the celebrated derandomization of the Maximal Independent Set algorithm. The latter uses the fact that the analysis of the "target" randomized algorithm only relies on the hypothesis that some objects are selected in pairwise independent manner. Thus, such weak generators do suffice to fool distinguishers that are derived from some natural and interesting algorithms.

Referring to Eq. (12), we remark that for constant $t \geq 2$, the cost of derandomization (i.e., going over all 2^k possible seeds) is exponential in the block-size (because $b(k) = \Omega(k)$), which in turn also bounds the number of blocks (because $\ell'(k) \leq 2^{b(k)}$). Note that if a larger number of blocks is needed, we can artificially increase the block-length in order to allow for it (i.e., allow $\ell'(k) = 2^{b(k)} = \exp(k/t)$), and in this case the cost of derandomization will be polynomial in the number of blocks. Thus, *whenever the analysis of a randomized algorithm can be based on a constant amount of independence* between (feasibly-many) random choices, each made inside a feasible domain, *a feasible derandomization is possible.*[35] On the other hand, the relationship $\ell(k) = \exp(k/t)$ is the best possible; that is, one cannot produce from a seed of length k an $\exp(k/O(1))$-long sequence of non-constantly independent random bits. In other words, t-wise independent generators of (any block-length and) stretch ℓ require a seed of length $\Omega(t \cdot \log \ell)$. In the next subsection we will see that meaningful approximations may be obtained with much shorter seeds.

6.2 Small-Bias Generators

Trying to go beyond constant-independence in derandomizations (while using seeds of length that is logarithmic in the length of the pseudorandom sequence) was the original motivation (and remain

[35] We stress that it is important to have the cost of derandomization be polynomial in the length of the produced pseudorandom sequence, because the latter is typically polynomially-related to the length of the input to the algorithm we wish to derandomize.

an important application) of the notion of small-bias generators. Still, small-bias generators are interesting for their own sake, and in particular they fool "global tests" that look at the entire output sequence and not merely at a fixed number of positions in it (as the limited independence generators). Specifically, small-bias generators generate a sequence of bits that fools any linear test (i.e., a test that computes a fixed linear combination of the bits).

For $\varepsilon:\mathbb{N}\to[0,1]$, an ε-bias generator with stretch function ℓ is an efficient deterministic algorithm (e.g., working in $\text{poly}(\ell(k))$ time) that expands a k-bit long random seed into a sequence of $\ell(k)$ bits such that for any fixed non-empty set $S\subseteq\{1,...,\ell(k)\}$ the bias of the output sequence over S is at most $\varepsilon(k)$. The bias of a sequence of n (possibly dependent) Boolean random variables $\zeta_1,...,\zeta_n\in\{0,1\}$ over a set $S\subseteq\{1,..,n\}$ is defined as

$$2\cdot\left|\Pr[\oplus_{i\in S}\zeta_i=1]-\frac{1}{2}\right| \;=\; |\Pr[\oplus_{i\in S}\zeta_i=1]-\Pr[\oplus_{i\in S}\zeta_i=0]|. \tag{14}$$

The factor of 2 was introduced so to make these biases correspond to the Fourier coefficients of the distribution (viewed as a function from $\{0,1\}^n$ to the reals). To see the correspondence replace $\{0,1\}$ by $\{\pm1\}$, and substitute XOR by multiplication. The bias with respect to set S is thus written as

$$\left|\Pr\left[\prod_{i\in S}\zeta_i=+1\right]-\Pr\left[\prod_{i\in S}\zeta_i=-1\right]\right| \;=\; \left|\mathsf{E}\left[\prod_{i\in S}\zeta_i\right]\right|,$$

which is merely the (absolute value of the) Fourier coefficient corresponding to S.

6.2.1 Constructions

Efficient small-bias generators with exponential stretch and exponentially vanishing bias are know.

Theorem 26 (small-bias generators):[36] *For some universal constant $c>0$, let $\ell:\mathbb{N}\to\mathbb{N}$ and $\varepsilon:\mathbb{N}\to[0,1]$ such that $\ell(k)\leq\varepsilon(k)\cdot\exp(k/c)$. Then, there exists an ε-bias generator with stretch function ℓ operating in time polynomial in the length of its output.*

Three simple constructions of small-bias generators that satisfy Theorem 26 are known (see [3]). One of these constructions is based on Linear Feedback Shift Registers. Loosely speaking, the first half of the seed, denoted $f_0f_1\cdots f_{(k/2)-1}$, is interpreted as a (non-degenerate) feedback rule[37], the other half, denoted $s_0s_1\cdots s_{(k/2)-1}$, is interpreted as "the start sequence", and the output sequence, denoted $r_0r_1\cdots r_{\ell(k)-1}$, is obtained by setting $r_i=s_i$ for $i<k/2$ and $r_i=\sum_{j=0}^{(k/2)-1}f_j\cdot r_{i-(k/2)+j}$ for $i\geq k/2$. (See Figure 5 and Exercise 55.)

As in Section 6.1.1, we note that the aforementioned constructions satisfy a stronger notion of efficient generation, which is useful in several applications. Specifically, there exists a polynomial-time algorithm that given a seed and a bit location $i\in[\ell(k)]$ (in bianry), outputs the i^{th} bit of the corresponding output.

[36] Here the common parameterization differs from ours merely in the point of view: Rather than saying that the functions ℓ and ε should satisfy $\ell(k)\leq\varepsilon(k)\cdot\exp(k/c)$, one says that given the desired parameters ℓ and ε the seed length k is set to $O(\log(\ell/\varepsilon))$. We also comment that using [3] the constant in the O-notation is merely 2 (i.e., $k\approx 2\log_2(\ell/\varepsilon)$), whereas using [32] $k\approx\log_2\ell+4\log_2(1/\varepsilon)$.

[37] That is, $f_0=1$ and $f(z)\stackrel{\text{def}}{=}z^{k/2}+\sum_{j=0}^{(k/2)-1}f_j\cdot z^j$ is an irreducible polynomial over GF(2).

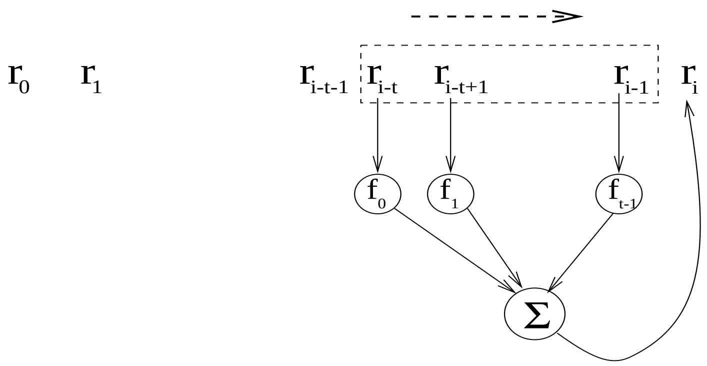


Figure 5: The LFSR small-bias generator (for $t = k/2$).

6.2.2 Applications

An archetypical application of small-bias generators is for generating random checks for fast string equality. The key obserevation is that checking whether or not $x = y$ is probabilistically reducible to checking whether the inner product modulo 2 of x and r equals the inner product modulo 2 of y and r, where r is generated by a small-bias generator. One advantage of this reduction is that only few bits (i.e., the seed of the generator and the result of the inner product) needs to be "communicated between x and y" (see Exercise 53). A related advantage (i.e., low randomness complexity) underlies the application of small-bias generators in the construction of PCPs.

Small-bias generators have been used in a variety of areas (e.g., inapproximation, structural complexity, and applied cryptography; see references in [15, Sec 3.6.2]). In addition, they seem an important tool in the design of various types of "pseudorandom" objects; see next.

Approximate independence generators. As hinted at the beginning of this section, small-bias is related to approximate limited independence.[38] Actually, even a restricted type of ε-bias (in which only subsets of size $t(k)$ are required to have bias upper-bounded by ε) implies that any $t(k)$ bits in the said sequence are $2^{t(k)/2} \cdot \varepsilon(k)$-close to $U_{t(k)}$, where here we refer to the variation distance (i.e., Norm-1 distance) between the two distributions. (The max-norm of the difference is bounded by $\varepsilon(k)$.)[39] Combining Theorem 26 and the foregoing upper-bound, and relying on the linearity of the construction in Proposition 24, we obtain generators with $\exp(k)$ stretch that are approximately $t(k)$-independent, for some non-constant $t(k)$; see Exercise 58. Specifically, for $k = O(t(k) + \log(1/\varepsilon(k)) + \log\log \ell(k))$ (equiv., for $\ell(k) = 2^{2^{k/O(1)}}$, $t(k) = k/O(1)$, and $\varepsilon(k) = 2^{-k/O(1)}$), one may obtain generators with stretch function ℓ, producing bit sequences in which any $t(k)$ positions are at most $\varepsilon(k)$-away from uniform (in variation distance). In the corresponding result for the max-norm distance, it suffices to have $k = O(\log(t(k)/\varepsilon(k) + \log\log \ell(k))$. Thus, whenever the analysis of a randomized algorithm can be based on a logarithmic amount of (almost) independence between feasibly-many binary random choices, a feasible derandomization is possible (by using an adequate generator of logarithmic seed length).

Extensions to non-binary choices were considered in various works (see references in [15, Sec 3.6.2]). Some of these works also consider the related problem of constructing small "discrepancy

[38] We warn that, unlike in the case of perfect independence, here we refer only to the distribution on fixed bit locations. See Exercise 52 for further discussion.

[39] Both bounds are derived from the Norm2 bound on the difference vector (i.e., the difference between the two probability vectors). For details, see Exercise 54.

sets" for geometric and combinatorial rectangles.

t-universal set generators. Using the aforementioned upper-bound on the max-norm, for $\varepsilon < 2^{-t}$, any ε-bias generator yields a t-universal set generator. The latter generator outputs sequences such that in every subsequence of length t all possible 2^t patterns occur (i.e., each for at least one possible seed). Such generators have many applications.

6.3 Random Walks on Expanders

In this section we review generators that produce a sequence of values by taking a random walk on a large graph (called an expander) having small degree but good "mixing" properties. Thus, producing a sequence of length ℓ over 2^b values, requires a random seed of length $b + (\ell - 1) \cdot \log_2 d$, where d is the degree of the said graph (of 2^b vertices). This should be compared against the randomness needed for generating a sequence of ℓ independent samples from $\{0,1\}^b$ (or taking a random walk on a clique of size 2^b). It will turn out that the pseudorandom sequence (generated by the said random walk on an expander) *behaves analogously to a truly random sequence with respect to hitting any fixed subset of* $\{0,1\}^b$. Let us start by defining this property, or rather define the hitting problem.

Definition 27 (the hitting problem): *A distribution on sequences over* $\{0,1\}^b$ *is* (ε, δ)-hitting *if for any* (target) *set* $T \subseteq \{0,1\}^b$ *of cardinality at least* $\varepsilon \cdot 2^b$*, with probability at least* $1 - \delta$*, at least one of the elements of a sequence drawn from this distribution hits* T*.*

Clearly, a truly random sequence of length ℓ over $\{0,1\}^b$ is (ε, δ)-hitting for $\delta = (1 - \varepsilon)^\ell$. The aforementioned "expander random walk generator" (to be described next) achieves similar behavior. Specifically, for arbitrary small $c > 0$ (which depends on the degree and the mixing property of the expander), the generator's output is (ε, δ)-hitting for $\delta = (1 - (1 - c) \cdot \varepsilon)^\ell$. To describe this generator, we need to discuss expanders.

Expanders. By expander graphs (or expanders) of degree d and eigenvalue bound $\lambda < d$, we actually mean an infinite family of d-regular graphs, $\{G_N\}_{N \in S}$ ($S \subseteq \mathbb{N}$), such that G_N is a d-regular graph over N vertices and the absolute value of all eigenvalues, save the biggest one, of the adjacency matrix of G_N is upper-bounded by λ. We will refer to such a family as to a (d, λ)-expander (for S). This technical definition is related to the aforementioned notion of "mixing" (which refers to the rate at which a random walk starting at a fixed vertex reaches uniform distribution over the graph's vertices).

We are interested in explicit constructions of such graphs, by which we mean that there exists a polynomial-time algorithm that on input N (in binary), a vertex $v \in G_N$ and an index $i \in \{1, ..., d\}$, returns the i^{th} neighbor of v. (We also require that the set S for which G_N's exist is sufficiently "tractable" – say that given any $n \in \mathbb{N}$ one may efficiently find an $s \in S$ such that $n \le s < 2n$.) Several explicit constructions of expanders are known. Below, we rely on the fact that for every $\varepsilon > 0$, there exist d and $\lambda < \varepsilon \cdot d$ such that there exists an explicit construction of a (d, λ)-expander over $\{2^b : b \in \mathbb{N}\}$.[40] The relevant (to us) fact about expanders is stated next.

Theorem 28 (Expander Random Walk Theorem): *Let* $G = (V, E)$ *be an expander graph of degree* d *and eigenvalue bound* λ*. Let* W *be a subset of* V *and* $\rho \stackrel{\text{def}}{=} |W|/|V|$*, and consider walks on* G

[40] This can be obtained with $d = \text{poly}(1/\varepsilon)$. In fact $d = O(1/\varepsilon^2)$, which is optimal, can be obtained too, albeit with graphs of sizes that are only approximately close to powers of two.

that start from a uniformly chosen vertex and take $\ell - 1$ additional random steps, where in each such step one uniformly selects one out of the d edges incident at the current vertex and traverses it. Then the probability that such a random walk stays in W is at most

$$\rho \cdot \left(\rho + (1-\rho) \cdot \frac{\lambda}{d}\right)^{\ell-1} \tag{15}$$

Thus, a random walk on an expander is "pseudorandom" with respect to the hitting property (i.e., when we consider hitting the set $V \setminus W$); that is, a set of density $\varepsilon = 1 - \rho$ is hit with probability $1 - \delta$, where $\delta = (1-\varepsilon) \cdot (1 - \varepsilon + (\lambda/d) \cdot \varepsilon)^{\ell-1} < (1 - (1 - (\lambda/d)) \cdot \varepsilon)^{\ell}$. A proof of an upper-bound that is weaker than Eq. (15) is outlined in Exercise 59. Using Theorem 28 and an explicit $(2^t, \overline{\lambda} \cdot 2^t)$-expander, we get

Proposition 29 (The Expander Random Walk Generator):[41]

- *For every constant $\overline{\lambda} > 0$, consider an explicit construction of $(2^t, \overline{\lambda} \cdot 2^t)$-expanders for $\{2^n : n \in \mathbb{N}\}$, where $t \in \mathbb{N}$ is a sufficiently latge constant. For $v \in [2^n] \equiv \{0,1\}^n$ and $i \in [2^t] \equiv \{0,1\}^t$, denote by $\Gamma_i(v)$ the vertex of the corresponding 2^n-vertex graph that is reached from vertex v when following its i^{th} edge.*

- *For $b, \ell' : \mathbb{N} \to \mathbb{N}$ such that $k = b(k) + (\ell'(k) - 1) \cdot t < \ell'(k) \cdot b(k)$, and for $v_0 \in \{0,1\}^{b(k)}$ and $i_1, ..., i_{\ell'(k)-1} \in [2^t]$, let*

$$G(v_0, i_1, ..., i_{\ell'(k)-1}) \stackrel{\text{def}}{=} (v_0, v_1, ..., v_{\ell'(k)-1}), \tag{16}$$

  where $v_j = \Gamma_{i_j}(v_{j-1})$.

Then G has stretch $\ell(k) = \ell'(k) \cdot b(k)$, and $G(U_k)$ is (ε, δ)-hitting for any $\varepsilon > 0$ and $\delta = (1 - (1 - \overline{\lambda}) \cdot \varepsilon)^{\ell'(k)}$.

The stretch of G is optimized at $b(k) \approx k/2$ (and $\ell'(k) = k/2t$), but optimizing the stretch is not necessaily the goal in all applications. Expander random-walk generators have been used in a variety of areas (e.g., PCP and inapproximability (see [7, Sec. 11.1]), cryptography (see [16, Sec. 2.6]), and the design of various types of "pseudorandom" objects.

Notes

Figure 6 depicts some of the notions of pseudorandom generators discussed in this text. We highlight a key distinction between the case of general-purpose pseudorandom generators (treated in Section 3) and the other cases (cf. Sections 4 and 5): in the former case the distinguisher is more complex than the generator, whereas in the latter cases the generator is more complex than the distinguisher. Specifically, in the general-purpose case the generator runs in (some *fixed*) polynomial-time and needs to withstand *any* probabilistic polynomial-time distinguisher. In fact, some of the proofs presented in Section 3 utilize the fact that the distinguisher can invoke the generator on seeds of its choice. In contrast, the Nisan-Wigderson Generator, analyzed in Theorem 18 (of Section 4), runs more time than the distinguishers that it tries to foil, and the proof relies on this fact in an essential manner. Similarly, the space complexity of the space-resilient generators presented in Section 5 is higher than the space-bound on the distinguishers that they foil.

[41] The common parameterization starts with parameters b and ℓ'. Given a uniformly chosen seed of length $b + O(\ell' - 1)$, one can efficiently and deterministically generate a random sequence of ℓ' strings, each of length b, which is (ε, δ)-hitting for any $\varepsilon > 0$ and $\delta = (1 - \Omega(\varepsilon))^{\ell'}$.

[42] By the OW we denote the assumption that one-way functions exists. By EvEC we denote the assumption that the class $\mathcal{E}$ has (almost-everywhere) exponential circuit complexity.

TYPE	distinguisher's resources	generator's resources	stretch (i.e., $\ell(k)$)	comments
gen.-purpose	$p(k)$-time, $\forall$ poly. p	poly(k)-time	poly(k)	Assumes OW[42]
derand. BPP	$2^{k/O(1)}$-time	$2^{O(k)}$-time	$2^{k/O(1)}$	Assumes EvEC[42]
space-bounded	$s(k)$-space	$O(k)$-space	$2^{k/O(s(k))}$	runs in time
robustness	$k/O(1)$-space	$O(k)$-space	poly(k)	poly$(k) \cdot \ell(k)$
t-wise indepen.	"t-wise"	poly$(k) \cdot \ell(k)$-time	$2^{k/O(t)}$	(e.g., pairwise)
small bias	"ε-bias"	poly$(k) \cdot \ell(k)$-time	$2^{k/O(1)} \cdot \varepsilon(k)$	
expander	"hitting"	poly$(k) \cdot \ell(k)$-time	$\ell'(k) \cdot b(k)$	
rand. walk	$(0.5, 2^{-\ell'(k)/O(1)})$-hitting for $\{0,1\}^{b(k)}$, with $\ell'(k) = ((k - b(k))/O(1)) + 1$.			

Figure 6: Pseudorandom generators at a glance

The general paradigm of pseudorandom generators. 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 technical development of the various notions was mostly unrelated, 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 development of the special-purpose generators (see Section 6) was unrelated to all of these.

General-purpose pseudorandom generators. The concept of *computational indistinguishability*, which underlies the entire computational approach to randomness, was suggested by Goldwasser and Micali [20] in the context of defining secure encryption schemes. Indeed, computational indistinguishability plays a key role in cryptography (see [16, 17]). The general formulation of computational indistinguishability is due to Yao [48]. Yao also observed (using the hybrid technique of [20]) that defining pseudorandom generators as producing sequences that are computationally indistinguishable from the corresponding uniform distribution is equivalent to defining such generators as producing unpredictable sequences. The latter definition originates in the earlier work of Blum and Micali [8].

Blum and Micali [8] pioneered the rigorous study of pseudorandom generators and, in particular, their construction based on some simple intractability assumption (in their case, the intractability of Discrete Logarithm problem over prime fields). Their work also introduces basic paradigms that were used in all subsequent improvements (cf., e.g., [48, 21]). We refer to the trasformation of computational difficulty into pseudorandomness, the use of hard-core predicates (defined in [8]), and the iteration paradigm (cf. Eq. (8)).

Theorem 11 (by which pseudorandom generators exist if and only if one-way functions exist) is due to Håstad, Impagliazzo, Levin and Luby [21], building upon the hard-core predicate of [19]. Unfortunately, the current proof of Theorem 11 is very complicated and unfit for presentation in a book of the current nature. Presenting a simpler and tighter (cf. §3.7.1) proof is indeed an important research project.

Derandomization of time-complexity classes. As observed by Yao [48], a non-uniformly strong notion of pseudorandom generators yields improved derandomization of time-complexity classes. A key observation of Nisan [33, 36] is that whenever a pseudorandom generator is used this way, it suffices to require that the generator runs in time exponential in its seed length, and so the generator may have running-time greater than the distinguisher (representing the algorithm to be derandomized). This observation underlines the construction of Nisan and Wigderson [33, 36], and is the basis for further improvements culminating in [22]. Part 1 of Theorem 19 (i.e., the so-called "high end" derandomization of $\mathcal{BPP}$) is due to Impagliazzo and Wigderson [22], whereas Part 2 (the "low end") is from [36].

The Nisan–Wigderson Generator [36] was subsequently used in several ways transcending its original presentation. We mention its application towards fooling non-deterministic machines (and thus derandomizing constant-round interactive proof systems) and to the construction of randomness extractors [44].

Space Pseudorandom Generators. As stated in the first paper on the subject of space-resilient pseudorandom generators [1][43], this research direction was inspired by the derandomization result obtained via use of general-purpose pseudorandom generators. The latter result (necessarily) depends on intractability assumptions, and so the objective was to find classes of algorithms for which derandomization is possible without relying on intractability assumptions. (This objective was achieved before for the case of constant-depth circuits.) Fundamentally different constructions of space pseudorandom generators were given in several works, but are superseeded by the two incomparable results mentioned in Section 5.2: Theorem 21 (a.k.a Nisan's Generator [34]) and Theorem 22 (a.k.a the Nisan–Zuckerman Generator [37]). These two results have been "interpolated" in [5]. Theorem 23 ($\mathcal{BPL} \subseteq \mathcal{SC}$) was proved by Nisan [35].

Special Purpose Generators. The various generators presented in Section 6 were not inspired by any of the other types of pseudorandom generator (nor even by the generic notion of pseudorandomness). Pairwise-independence generator were explicitly suggested in [11] (and are implicit in [9]). The generalization to t-wise independence (for $t \geq 2$) is due to [2]. Small-bias generators were first defined and constructed by Naor and Naor [32], and three simple constructions were subsequently given in [3]. The Expander Random Walk Generator was suggested by Ajtai, Komlos, and Szemerédi [1], who discovered that random walks on expander graphs provide a good approximation to repeated independent attempts for hitting any arbitrary fixed subset of sufficient density (within the vertex set). The analysis of the hitting property of such walks was subsequently improved, culminating in the bound cited in Theorem 28, which is taken from [23, Cor. 6.1].

(The foregoing historical notes do not mention several technical contributions that played an important role in the development of the area. For further details, the reader is referred to [15, Chap. 3]. In fact, the current text is a revision of [15, Chap. 3], providing more details for the main topics, and omitting [15, Sec. 3.6.4 and 3.6.5].)

Exercises

Exercise 30 Prove that placing no computational requirements on the generator yields "generators" that can fool any family of subexponential-size circuits. That is, 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.

[43] This paper is more frequently cited for the Expander Random Walk technique which it has introduced.

Guideline: Use the Probabilistic Method (cf. [4]). First, for any fixed circuit $C : \{0,1\}^n \to \{0,1\}$, upper-bound the probabity that for a random set $S \subset \{0,1\}^n$ of size $2^{n/2}$ the absolute value of $\Pr[C(U_n) = 1] - (|\{x \in S : C(x) = 1\}|/|S|)$ is larger than $2^{-n/100}$. 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 $2^{n/100}$ 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 $2^{-n/100}$.

Exercise 31 Let A be a probabilistic polynomial-time algorithm solving the search associated with the NP-relation R, and let A_G be as in Construction 2. Prove that it is infeasible to find an x on which A_G outputs a wrong solution; that is, assuming for simplicity that A has error probability 1/3, prove that on input 1^n it is infeasible to find an $x \in \{0,1\}^n \cap L_R$ such that $\Pr[(x, A_G(x)) \notin R] > 0.4$, where $L_R \stackrel{\rm def}{=} \{x : \exists y\, (x,y) \in R\}$.
(Hint: For x that violates the claim, it holds that $|\Pr[(x, A(x)) \notin R] - \Pr[(x, A_G(x)) \notin R]| > 0.06$.)

Exercise 32 Prove that omitting the absolute value in Eq. (4) keeps Definition 4 intact.
(Hint: consider $D'(z) \stackrel{\rm def}{=} 1 - D(z)$.)

Exercise 33 Show that the existence of pseudorandom generators implies the existence of polynomial-time constructible probability ensembles that are statistically far apart and yet are computationally indistinguishable.
(Hint: lower-bound the statistical distance between $G(U_k)$ and $U_{\ell(k)}$, where G is a pseudorandom generator with stretch ℓ.)

Exercise 34 Prove that the sufficient condition in Exercise 33 is in fact necessary.[44] Recall that $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are said to be statistically far apart if, for some positive polynomial p and all sufficiently large n, the variation distance between X_n and Y_n is greater than $1/p(n)$. Using the following three steps, prove that the existence of *polynomial-time constructible* probability ensembles that are statistically far apart and yet are computationally indistinguishable implies the existence of pseudorandom generators.

1. Show that, without loss of generality, we may assume that the variation distance between X_n and Y_n is greater than $1 - \exp(-n)$.

   (Hint: Consider $\overline{X}_n = (X_n^{(1)}, ..., X_n^{(t(n))})$ and $\overline{Y}_n = (Y_n^{(1)}, ..., Y_n^{(t(n))})$, where the $X_n^{(i)}$'s (resp., $Y_n^{(i)}$'s) are independent copies of X_n (resp., Y_n), and $t(n) = O(n \cdot p(n)^2)$.)

2. Using $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ as in Step 1, prove the existence of a *false entropy generator*, where a false entropy generator is a deterministic polynomial-time algorithm G such that $G(U_k)$ has entropy $e(k)$ but $\{G(U_k)\}_{k\in\mathbb{N}}$ is computationally indistinguishable from a polynomial-time constructible ensemble that has entropy greater than $e(\cdot) + (1/2)$.

   (Hint: Let S_0 and S_1 be sampling algorithms such that $X_n \equiv S_0(U_{{\rm poly}(n)})$ and $Y_n \equiv S_1(U_{{\rm poly}(n)})$. Consider the generator $G(\sigma, r) = (\sigma, S_\sigma(r))$, and the distribution Z_n that equals (U_1, X_n) with probability 1/2 and (U_1, Y_n) otherwise. Note that in $G(U_1, U_{{\rm poly}(n)})$ the first bit is almost determined by the rest, whereas in Z_n the first bit is statistically independent of the rest.)

3. Using a false entropy generator, obtain one in which the excess entropy is $\sqrt{k}$, and using the latter construct a pseudorandom generator.

   (Hint: Use the ideas presented at the end of Section 3.5 (i.e., the discussion of the interesting direction of the proof of Theorem 11).)

[44] This exercise follows [14], which in turn builds on [21].

Exercise 35 Prove that *if* $\{X_n\}_{n\in\mathbb{N}}$ *and* $\{Y_n\}_{n\in\mathbb{N}}$ *are computationally indistinguishable and* A *is a probabilistic polynomial-time algorithm then* $\{A(X_n)\}_{n\in\mathbb{N}}$ *and* $\{A(Y_n)\}_{n\in\mathbb{N}}$ *are computationally indistinguishable.*
(Hint: If D distinguishes the latter ensembles then D' such that $D'(z) \stackrel{\text{def}}{=} D(A(z))$ distinguishes the former.)

Exercise 36 In continuation to Exercise 35, show that the conclusion may not hold in case A is not computationally bounded. That is, show that there exists computationally indistinguishable ensembles, $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, and an exponential-time algorithm A such that $\{A(X_n)\}_{n\in\mathbb{N}}$ and $\{A(Y_n)\}_{n\in\mathbb{N}}$ are *not* computationally indistinguishable.
(Hint: For any pair of ensembles $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, consider the Boolean function f such that $f(z) = 1$ if and only if $\Pr[X_n = z] > \Pr[Y_n = z]$. Show that $|\Pr[f(X_n) = 1] - \Pr[f(Y_n) = 1]|$ equals the statistical difference between X_n and Y_n. Consider an adequate (approximate) implementation of f (e.g., approximate $\Pr[X_n = z]$ and $\Pr[Y_n = z]$ up to $\pm 2^{-2|z|}$), and use Exercise 30.)

Exercise 37 For G_1 and ℓ as in Construction 7, consider $G(s) \stackrel{\text{def}}{=} G_1^{\ell(|s|)}(s)$, where $G_1^i(x)$ denotes G_1 iterated i times on x (i.e., $G_1^i(x) = G_1^{i-1}(G_1(x))$ and $G_1^0(x) = x$). Prove that G is a pseudorandom generator of stretch ℓ. Reflect on the advantages of Construction 7 over the current construction.
(Hint: Use a hybrid argument, with the i^{th} hybrid being $G_1^i(U_{\ell(k)-i})$. Note that $G_1^{i+1}(U_{\ell(k)-(i+1)}) = G_1^i(G_1(U_{\ell(k)-i-1}))$ and $G_1^i(U_{\ell(k)-i}) = G_1^i(U_{|G_1(U_{\ell(k)-i-1})|})$, and use Exercise 35.)

Exercise 38 (pseudorandom versus unpredictability) Prove that a probability ensemble $\{Z_k\}_{k\in\mathbb{N}}$ is pseudorandom if and only if it is unpredictable. For simplicity, we say that $\{Z_k\}_{k\in\mathbb{N}}$ is (next-bit) unpredictable if for every probabilistic polynomial-time algorithm A it holds that $\Pr_i[A(F_i(Z_k)) = B_{i+1}(Z_k)]$ is negligible, where $i \in \{0, ..., |Z_k|-1\}$ is uniformly distributed, and $F_i(z)$ (resp., $B_{i+1}(z)$) denotes the i-bit prefix (resp., $i + 1^{\text{st}}$ bit) of z.

Guideline: Show that pseudorandomness implies polynomial-time unpredictability; that is, polynomial-time predictability violates pseudorandomness (because the uniform ensemble is unpredictable regardless of computing power). Use a hybrid argument to prove that unpredictability implies pseudorandomness. Specifically, the i^{th} hybrid consists of the i-bit long prefix of Z_k followed by $|Z_k| - i$ uniformly distributed bits. Thus, distinguishing the extreme hybrids (which correspond to Z_k and $U_{|Z_k|}$) implies distinguishing some neighboring hybrids, which in turn implies next-bit predictability. For the last step, use an argument as in the proof of Proposition 9.

Exercise 39 Prove that a probability ensemble is unpredictable (from left to right) if and only if it is unpredictable from right to left (or in any other canonical order).
(Hint: use Exercise 38, and note that an ensemble is pseudorandom if and only if its reverse is pseudorandom.)

Exercise 40 Let f be 1-1 and length preserving, and b be a hard-core predicate of f. For any polynomial ℓ, prove that $\{G'(U_k)\}$ is unpredictable (in the sense of Exercise 38), where $G'(s) \stackrel{\text{def}}{=} b(f^{\ell(|s|)-1}(s)) \cdots b(f(s)) \cdot b(s)$.

Guideline: Suppose towards the contradiction that, for a uniformly distributed $j \in \{0, ..., \ell(k) - 1\}$, given the j-bit long prefix of $G'(U_k)$ an algorithm A' can predict the $j + 1^{\text{st}}$ bit of $G'(U_k)$. That is, given $b(f^{\ell(k)-1}(s)) \cdots b(f^{\ell(k)-j}(s))$, algorithm A' predicts $b(f^{\ell(k)-(j+1)}(s))$, where s is uniformly distributed in $\{0,1\}^k$. Consider an algorithm A that given $y = f(x)$ approximates $b(x)$ by invoking A' on input $b(f^{j-1}(y)) \cdots b(y)$, where j is uniformly selected in $\{0, ..., \ell(k) - 1\}$. Analyze the success probability of A using the fact that f induces a permutation over $\{0,1\}^n$, and thus $b(f^j(U_k)) \cdots b(f(U_k)) \cdot b(U_k)$ is distributed identically to $b(f^{\ell(k)-1}(U_k)) \cdots b(f^{\ell(k)-j}(U_k)) \cdot b(f^{\ell(k)-(j+1)}(U_k))$.

Exercise 41 Prove that if G is a strong pseudorandom generator in the sense of Definition 12 then it a pseudorandom generator in the sense of Definition 1.
(Hint: consider a sequence of internal coin tosses that maximizes the probability in Eq. (2).)

Exercise 42 Show that there exists a circuit of size $O(2^k \cdot \ell(k))$ that violates Eq. (9), provided $\ell(k) > k$.
(Hint: The circuit may incorporate all values in the range of G and deciding by comparing its input to these values.)

Exercise 43 (constructing a set system for Theorem 18) For every $\gamma > 0$, show a construction of a set system S as in Condition 2 of Theorem 18, with $m(k) = \Omega(k)$ and $\ell(k) = 2^{\Omega(k)}$.

Guideline: We assume, without loss of generality, that $\gamma < 1$, and set $m(k) = (\gamma/2) \cdot k$ and $\ell(k) = 2^{\gamma m(k)/10}$. We construct the set system $S_1, ..., S_{\ell(k)}$ in iterations, selecting S_i as the first $m(k)$-subset of $[k]$ that has sufficiently small intersections with each of the previous sets $S_1, ..., S_{i-1}$. The existence of such a set S_i can be proved using the Probabilistic Method (cf. [4]). Specifically, for a fixed $m(k)$-subset S', the probability that a random $m(k)$-subset has intersection greater than $\gamma m(k)$ with S' is upper-bounded[45] by $2^{-\gamma m(k)/10}$, because the expected intersection size is $(\gamma/2) \cdot m(k)$. Thus, with positive probability a random $m(k)$-subset has intersection at most $\gamma m(k)$ with each of the previous $i - 1 < \ell(k) = 2^{\gamma m(k)/10}$ subsets. Note that we construct S_i in time $\binom{k}{m(k)} \cdot (i-1) \cdot m(k) < 2^k \cdot \ell(k) \cdot k$, and thus S is computable in time $k2^k \cdot \ell(k)^2 < 2^{2k}$.

Exercise 44 Suppose that the sets S_i's in Construction 17 are disjoint and that $f : \{0,1\}^m \to \{0,1\}$ is T-inapproximable. Prove that for every circuit C of size $T - O(1)$ it holds that $|\mathsf{Pr}[C(G(U_k)) = 1] - \mathsf{Pr}[C(U_\ell) = 1]| < \ell/T$.

Guideline: Prove the contrapositive using Exercise 45. Note that the values of the $i + 1^{\text{st}}$ bit of $G(U_k)$ is statistically independent of the values of the first i bits of $G(U_k)$, and thus predicting it yields an approximator for f. Indeed, such an approximator can be obtained by fixing the the first i bits of $G(U_k)$ via an averaging argument.

Exercise 45 In continuation to Exercise 38, show that if there exists a circuit of size s that distinguishes Z_n from U_ℓ with gap δ, then there exists an $i < \ell = |Z_n|$ and a circuit of size $s + O(1)$ that given an i-bit long prefix of Z_n guesses the $i + 1^{\text{st}}$ bit with success probability at least $\frac{1}{2} + \frac{\delta}{\ell}$.
(Hint: defining hybrids as in Exercise 38, note that, for some i, the given circuit distinguishes the i^{th} hybrid from the $i + 1^{\text{st}}$ hybrid with gap at least δ/ℓ.)

Exercise 46 (Theorem 18, generalized) Let $\ell, m, m', T : \mathbb{N} \to \mathbb{N}$ satisfy $\ell(k)^2 + \widetilde{O}(\ell(k)2^{m'(k)}) < T(m(k))$. Suppose that the following two conditions hold:

1. There exists an exponential-time computable function $f : \{0,1\}^* \to \{0,1\}$ that is T-inapproximable.
2. There exists an exponential-time computable function $S : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that $|S(k,i)| = m(k)$ for every k and $i = 1, ..., \ell(k)$, and $|S(k,i) \cap S(k,j)| \leq m'(k)$ for every k and $i \neq j$.

Prove that using G as defined in Construction 17, with $S_i = S(k,i)$, yields a canonical derandomizer with stretch ℓ.
(Hint: following the proof of Theorem 18, just note that the circuit constructed for approximating $f(U_{m(k)})$ has size $\ell(k)^2 + \ell(k) \cdot \widetilde{O}(2^{m'(k)})$ and success probability at least $(1/2) + (1/7\ell(k))$.)

[45] Applying the standard Chernoff Bound yields an upper-bound of $\exp(-(\gamma/2)^2 \cdot m(k))$, which suffices for $\ell(k) = 2^{\gamma^2 m(k)/10}$. However, using a Multiplicative Chernoff Bound yields an upper-bound of $\exp(-(\gamma/2) \cdot m(k)/3)$.

Exercise 47 (Part 2 of Theorem 19) Prove that if for every polynomial T there exists a T-inapproximable predicate in $\mathcal{E}$ then $\mathcal{BPP} \subseteq \cap_{\varepsilon>0}\mathrm{D}\textsc{time}(t_\varepsilon)$, where $t_\varepsilon(n) \stackrel{\rm def}{=} 2^{n^\varepsilon}$.
(Hint: For any p-time algorithm, apply Exercise 46 using $\ell(k) = p(k^{1/\varepsilon})$, $m(k) = \sqrt{k}$ and $m'(k) = O(\log k)$. Revisit Exercise 43 in order to obtain a set system as required in Exercise 46 (for these parameters), and use an adequate worst-case to average-case reduction.)

Exercise 48 Provide an explicit description of the generator outlined in the proof of Theorem 21.
(Hint: for $r \in \{0,1\}^n$ and $h^{(1)}, ..., h^{(t)} \in H_n$, the generaor outputs a 2^t-long sequence of n-bit strings such that the $i^{\rm th}$ block equals $h'(r)$, where h' is a composition of some of the $h^{(j)}$'s.)

Exercise 49 (adaptive t-wise independence tests) Prove that the output of a t-wise independence generator is indistinguishable to any test than examines t of the blocks, even if the examined blocks are selected adaptively (i.e., the location of the $i^{\rm th}$ block is determined based on the contents of the previously inspected blocks).

Guideline: First show that, without loss of generality, it suffices to consider deterministic (adaptive) tester. Next, show that the probability that such a tester sees any fixed sequence of t values at locations selected adaptively in the generator's output is $2^{-t\cdot b(k)}$, where $b(k)$ is the block length.

Exercise 50 (t-wise independence generator) Prove that G as defined in Proposition 24 produces a t-wise independent sequence over $\mathrm{GF}(2^{b(k)})$.

Guideline: For every t fixed indices $i_1, ..., i_t \in [\ell'(k)]$, consider the distribution of $G(U_k)_{i_1,...,i_t}$ (i.e., the projection of $G(U_k)$ on locations $i_1, ..., i_t$). Show that for every sequence of t possible values $v_1, ..., v_t \in \mathrm{GF}(2^{b(k)})$, there exists a unique seed $s \in \{0,1\}^k$ such that $G(s)_{i_1,...,i_t} = (v_1, ..., v_t)$.

Exercise 51 (pairwise independence generators) As a warm-up, consider a construction analogous to the one in Proposition 25, where the seed specifies an affine $b(k)$-by-$m(k)$ transformation. That is, for $s \in \{0,1\}^{b(k)\cdot m(k)}$ and $r \in \{0,1\}^{b(k)}$, where $k = b(k) \cdot m(k) + b(k)$, let

$$G(s,r) \stackrel{\rm def}{=} (A_s v_1 + r \,,\, A_s v_2 + r \,, ...,\, A_s v_{\ell'(k)} + r) \tag{17}$$

where A_s is an $b(k)$-by-$m(k)$ matrix specified by the string s. Show that G as in Eq. (17) is a pairwise independence generator of block-size b and stretch ℓ. Next, show that G as in Eq. (13) is a pairwise independence generator of block-size b and stretch ℓ.

Guideline: The following description applies to both constructions. First note that for every fixed $i \in [\ell'(k)]$, the $i^{\rm th}$ element in the sequence $G(U_k)$ is uniformly distributed in $\{0,1\}^{b(k)}$. Actually, show that for every fixed $s \in \{0,1\}^{k-b(k)}$, it holds that $G(s, U_{b(k)})_i$ is uniformly distributed in $\{0,1\}^{b(k)}$. Next note that it suffices to show that, for every $j \neq i$, conditioned on the value of the $i^{\rm th}$ element in $G(U_k)$, the $j^{\rm th}$ element is uniformly distributed in $\{0,1\}^{b(k)}$. The key technical detail is to show that for any non-zero vector $v \in \{0,1\}^{m(k)}$ it holds that $A_{U_{k-b(k)}} v$ (resp., $T_{U_{k-b(k)}} v$) is uniformly distributed in $\{0,1\}^{b(k)}$. This is easy in case of a random $b(k)$-by-$m(k)$ matrix, and can be proven also for a random Toeplitz matrix.

Exercise 52 (adaptive t-wise independence tests, revisited) In contrast to Exercise 49, we note that almost uniform distribution on any fixed t bit locations does not imply that an adaptive test that inspects t locations cannot detect "non-uniformity" (i.e., a "non random behavior" of the inspected sequence). Specifically, present a distribution over 2^{t-1}-bit long strings in which each $t-1$ fixed bit positions are $t \cdot 2^{-(t-1)}$-close to uniform, but some test that adaptively inspects t positions can distinguish this distribution from the uniform one with constant gap.
(Hint: Modify the uniform distribution over $((t-1) + 2^{t-1})$-bit long strings such that the first $t-1$ locations indicate a bit position (among the rest) that is set to zero.)

Exercise 53 Suppose that G is an ε-bias generator with stretch ℓ. Show that equality between the $\ell(k)$-bit strings x and y can be probabilistically checked by comparing the inner product modulo 2 of x and $G(s)$ to the inner product modulo 2 of y and $G(s)$, where $s \in \{0,1\}^k$ is selected uniformly. (Hint: reduce the problem to the special case in which $y = 0^{\ell(k)}$.)

Exercise 54 (bias versus statistical difference from uniform) Let X be a random variable assuming values in $\{0,1\}^t$. Prove that if X has bias at most ε over any non-empty set then the statistical difference between X and U_t is at most $2^{t/2} \cdot \varepsilon$, and that for every $x \in \{0,1\}^t$ it holds that $\Pr[X = x] = 2^{-t} \pm \varepsilon$.

Guideline: Consider the probability function $p : \{0,1\}^t \to [0,1]$ defined by $p(x) \stackrel{\text{def}}{=} \Pr[X = x]$, and let $\delta(x) \stackrel{\text{def}}{=} p(x) - 2^{-t}$ denote the deviation of p from the uniform probability function. Viewing the set of real functions over $\{0,1\}^t$ as a 2^t-dimensional vector space, we consider two orthonormal bases for this space. The first basis is of the (Kroniker) functions $\{k_\alpha\}_{\alpha \in \{0,1\}^t}$ such that $k_\alpha(x) = 1$ if $x = \alpha$ and $k_\alpha(x) = 0$ otherwise. The second basis is of the (normalize Fourier) functions $\{f_S\}_{S \subseteq [t]}$ defined by $f_S(x) \stackrel{\text{def}}{=} 2^{-t/2} \prod_{i \in S} (-1)^{x_i}$ (where $f_\emptyset \equiv 2^{-t/2}$).[46] Note that the bias of X over any $S \neq \emptyset$ equals $|\sum_x p(x) \cdot 2^{t/2} f_S(x)|$, which in turn equals $2^{t/2} |\sum_x \delta(x) f_S(x)|$. Thus, for every S (including the empty set), we have $|\sum_x \delta(x) f_S(x)| \leq 2^{-t/2}\varepsilon$, which means that the representation of δ in the normalize Fourier basis is by coefficients that have each an absolute value of at most $2^{-t/2}\varepsilon$. Thus, the Norm-2 of this vector of coefficients is bounded by $\sqrt{2^t \cdot (2^{-t/2}\varepsilon)^2} = \varepsilon$, and all claims follow by noting that they refer to norms of δ according to the Kroniker basis. In particular, Norm-2 is preserved under orthonormal bases, the max-norm is upper-bounded by Norm-2, and Norm-1 is upper-bounded by $\sqrt{2^t}$ times the value of the Norm-2.

Exercise 55 (The LFSR small-bias generator (following [3])) Using the following guidelines (and letting $t = k/2$), analyze the construction outlined following Theorem 26 (and depicted in Figure 5):

1. Prove that $r_i = \sum_{j=0}^{t-1} c_j^{(i)} \cdot s_j$, where $c_j^{(i)}$ is the coefficient of z^j in the (degree $t-1$) polynomial obtained by reducing z^i modulo the polynomial $f(z)$ (i.e., $z^i \equiv \sum_{j=0}^{t-1} c_j^{(i)} z^j \pmod{f(z)}$).

   (Hint: Recall that $z^t \equiv \sum_{j=0}^{t-1} f_j z^j \pmod{f(z)}$, and thus $z^i \equiv \sum_{j=0}^{t-1} f_j z^{i-t+j} \pmod{f(z)}$. Note the correspondance to $r_i = \sum_{j=0}^{t-1} f_j \cdot r_{i-t+j}$.)

2. For any non-empty $S \subseteq \{0, ..., \ell(k) - 1\}$, evaluate the bias of the sequence $r_0, ..., r_{\ell(k)-1}$ over S, where f is a random irreducible polynomial of degree t and $s = (s_0, ..., s_{t-1}) \in \{0,1\}^t$ is uniformly distributed. Specifically:

   (a) For a fixed f and random $s \in \{0,1\}^t$, prove that $\sum_{i \in S} r_i$ has non-zero bias if and only if $f(z)$ divides $\sum_{i \in S} z^i$.

   (Hint: Note that $\sum_{i \in S} r_i = \sum_{j=0}^{t-1} \sum_{i \in S} c_j^{(i)} s_j$, and use Item 1.)

   (b) Prove that the probability that a random irreducible polynomial of degree t divides $\sum_{i \in S} z^i$ is $\Theta(\ell(k)/2^t)$.

   (Hint: A polynomial of degree n can be divided by at most n/d different irreducible polynomials of degree d. On the other hand, the number of irreducible polynomials of degree d over GF(2) is $\Theta(2^d/d)$.)

   Conclude that for random f and s, the sequence $r_0, ..., r_{\ell(k)-1}$ has bias $O(\ell(k)/2^t)$.

---

[46] Verify that both bases are indeed orthogonal (i.e., $\sum_x k_\alpha(x) k_\beta(x) = 0$ for every $\alpha \neq \beta$ and $\sum_x f_S(x) f_T(x) = 0$ for every $S \neq T$) and normal (i.e., $\sum_x k_\alpha(x)^2 = 1$ and $\sum_x f_S(x)^2 = 1$).

Note that an implementation of the LFSR generator requires a mapping of random $k/2$-bit long seeds to almost random irreducible polynomials of degree $k/2$. Such a mapping can be constructed in $\exp(k)$ time, which is $\text{poly}(\ell(k))$ if $\ell(k) = \exp(\Omega(k))$. A more efficient mapping that uses a $O(k)$-bit long seek is described in [3, Sec. 8].

Exercise 56 (limitations on small-bias generators) Let G be an ε-bias generator with stretch ℓ, and view G as a mapping from $\text{GF}(2)^k$ to $\text{GF}(2)^{\ell(k)}$. As such, each bit in the output of G can be viewed as a polynomial in the k input variables (each ranging in GF(2)). Prove that if $\varepsilon(k) < 1$ and each of these polynomials has degree at most d then $\ell(k) \leq \sum_{i=1}^{d} \binom{k}{i}$.

Guideline: First note that, without loss of generality, all polynomials have a free term equal to zero. Then, consider the vector space spanned by all d-monomials over k variables (i.e., monomial having at most d variables). Since $\varepsilon(k) < 1$, the polynomials representing the output bits of G must correspond to a sequence of independent vectors in this space. Derive the following corollaries:

1. If $\varepsilon(k) < 1$ then $\ell(k) < 2^k$.
2. If $\varepsilon(k) < 1$ and $\ell(k) > k$ then G cannot be a linear transformation.

Note that $G(s) = (s, b(s))$, where $b(s_1, ..., s_k) = \sum_{i=1}^{k/2} s_i s_{(k/2)+i} \bmod 2$, is an ε-biased generator with $\varepsilon(k) = \exp(-\Omega(k))$.
(Hint: Focusing on bias over sets that include the last output bit, prove that without loss of generality it suffices to analyze the bias of $b(U_k)$.)

Exercise 57 (a sanity check for pseudorandomness) The following fact is suggested as a sanity check for candidate pseudorandom generators with respect to space-bounded machines. The fact (to be proven as an exercise) is that, for every $\varepsilon(\cdot)$ and $s(\cdot)$ such that $s(k) \geq 1$ for every k, if G is (s, ε)-pseudorandom (as per Definition 20), then G is an ε-bias generator.

Exercise 58 (approximate t-wise independent generators (following [32])) Combining Theorem 26 and relying on the linearity of the t-wise independent generator of Eq. (12), construct a generator producing ℓ-bit long sequences in which any t positions are at most ε-away from uniform (in variation distance), while using a seed of length $O(t + \log(1/\varepsilon) + \log\log \ell)$. (For max-norm a seed of length $O(\log(t/\varepsilon) + \log\log \ell)$ suffices.)

Guideline: First note that, for any t, ℓ' and b, the transformation of Eq. (12) can be implemented by a fixed linear (over GF(2)) transformation of a $t \cdot b$-bit seed into an ℓ-bit bit sequence, where $\ell = \ell' \cdot b$. It follows that there exists a fixed GF(2)-linear transformation T of a random seed of length $t \cdot b$, where $b = \log_2 \ell$, into a t-wise independent bit sequence of the length ℓ (i.e., $T U_{t \cdot b}$ is t-wise independent over $\{0,1\}^\ell$). Thus, every t rows of T are linearly independent. The key observation is that when we replace the aforementioned random seed by an ε'-biased sequence, every $i \leq t$ positions in the output sequence have bias at most ε' (because they define a non-zero linear test on the bits of the ε'-biased sequence). Note that the length of the new seed (used to produce ε'-biased sequence of length $t \cdot b$) is $O(\log tb/\varepsilon')$. Applying Exercise 54, we conclude that any t positions are at most $2^{t/2} \cdot \varepsilon'$-away from uniform (in variation distance). Recall that this was obtained using a seed of length $O(\log(t/\varepsilon') + \log\log \ell)$, and the claim follows by using $\varepsilon' = 2^{-t/2} \cdot \varepsilon$.

Exercise 59 (A version of the Expander Random Walk Theorem) Using notations as in Theorem 28, prove that the probability that a random walk of length ℓ stays in W is at most

$(\rho + (\lambda/d)^2)^{\ell/2}$. In fact, prove a more general claim that refers to the probability that a random walk of length ℓ intersects $W_0 \times W_1 \times \cdots \times W_{\ell-1}$. The claimed upper-bound is

$$\sqrt{\rho_0} \cdot \prod_{i=1}^{\ell-1} \sqrt{\rho_i + (\lambda/d)^2}, \tag{18}$$

where $\rho_i \stackrel{\rm def}{=} |W_i|/|V|$.

Guideline: View the random walk as the evolution of a corresponding probability vector under suitable transformations. The transformations correspond to taking a random step in the graph and to passing through a "sieve" that keeps only the entries that correspond to the current set W_i. The key observation is that the first trasformation shrinks the component that is orthogonal to the uniform distribution (which is the first eigenvalue of the adjacency matrix of the expander), whereas the second trasformation shrinks the component that is in the direction of the uniform distribution. Details follow.

View the random walk as the evolution of a corresponding probability vector under suitable transformations. Let A be a matrix representing the random walk on G (i.e., A is the adjacency matrix of G divided by the degree, d). Let $\overline{\lambda}$ denote the absolute value of the second largest eigenvalue of A (i.e., $\overline{\lambda} \stackrel{\rm def}{=} \lambda/d$), and note that $u = (|V|^{-1}, ..., |V|^{-1})$ (which represents the uniform distribution) is the eigenvector of A that is associated with the largest eigenvalue (which is 1). Let P_i be a 0-1 matrix that has 1-entries only on its diagonal, and furthermore entry (j,j) is set to 1 if and only if $j \in W_i$. Then, the probability that a random walk of length ℓ intersects $W_0 \times W_1 \times \cdots \times W_{\ell-1}$ is the sum of the entries of the vector $v \stackrel{\rm def}{=} P_{\ell-1}A \cdots P_2AP_1AP_0u$. We are interested in upper-bounding $\|v\|_1$, and use $\|v\|_1 \leq \sqrt{|V|} \cdot \|v\|$, where $\|z\|_1$ and $\|z\|$ denote the L_1 norm and L_2 norm of z, respectively (e.g., $\|u\|_1 = 1$ and $\|u\| = |V|^{-1/2}$). The key observation is that, for every z, it holds that $\|P_iAz\| \leq (\rho_i + \overline{\lambda}^2)^{1/2} \cdot \|z\|$, which is proven by decomposing $z = z_1 + z_2$ such that z_1 is the projection of z on u (the "first" eigenvector of A) and z_2 is the component orthogonal to u. Facts to be used in the proof of the forgoing observation include $\|P_iAz_1\| = \|P_iz_1\| \leq \sqrt{\rho_i}\|z_1\|$ and $\|P_iAz_2\| \leq \|Az_2\| \leq \overline{\lambda}\|z_2\|$ (i.e., P_i shrinks any uniform vector by eliminating $1-\rho_i$ of its elements, whereas A shrinks the length of any eigenvector except u by a factor of at least $\overline{\lambda}$).[47]

Exercise 60 Using notations as in Theorem 28, prove that the probability that a random walk of length ℓ visits W more than $\alpha\ell$ times is smaller than $\binom{\ell}{\alpha\ell} \cdot (\rho + (\lambda/d)^2)^{\alpha\ell/2}$. For example, for $\alpha = 1/2$ and $\lambda/d < \sqrt{\rho}$, we get an upper-bound of $(32\rho)^{\ell/4}$. We comment that much better bounds can be obtained (cf. [13]).

(Hint: Use a union bound on all possible sequences of $m = \alpha\ell$ visits, and upper-bound the probability of visiting W in steps $j_1, ..., j_m$ by applying Eq. (18) with $W_i = W$ if $i \in \{j_1, ..., j_m\}$ and $W = V$ otherwise.)

References

[1] M. Ajtai, J. Komlos, E. Szemerédi. Deterministic Simulation in LogSpace. In *19th ACM Symposium on the Theory of Computing*, pages 132–140, 1987.

[2] N. Alon, L. Babai and A. Itai. A fast and Simple Randomized Algorithm for the Maximal Independent Set Problem. *J. of Algorithms*, Vol. 7, pages 567–583, 1986.

[47] Also use the triangle inequality (for $\|P_iAz_1 + P_iAz_2\| \leq \|P_iAz_1\| + \|P_iAz_2\|$), the Cauchy-Schwartz inequality (for $\sqrt{\rho_i}\|z_1\| + \overline{\lambda}\|z_2\| \leq \sqrt{\rho_i + \overline{\lambda}^2} \cdot \sqrt{\|z_1\|^2 + \|z_2\|^2}$), and $\sqrt{\|z_1\|^2 + \|z_2\|^2} = \|z_1 + z_2\| = \|z\|$.

[3] N. Alon, O. Goldreich, J. Håstad, R. Peralta. Simple Constructions of Almost k-wise Independent Random Variables. *Journal of Random structures and Algorithms*, Vol. 3, No. 3, (1992), pages 289–304.

[4] N. Alon and J.H. Spencer. *The Probabilistic Method.* John Wiley & Sons, Inc., 1992.

[5] R. Armoni. On the derandomization of space-bounded computations. In the proceedings of *Random98*, Springer-Verlag, Lecture Notes in Computer Science (Vol. 1518), pages 49–57, 1998.

[6] H. Attiya and J. Welch: *Distributed Computing: Fundamentals, Simulations and Advanced Topics.* McGraw-Hill, 1998.

[7] M. Bellare, O. Goldreich and M. Sudan. Free Bits, PCPs and Non-Approximability – Towards Tight Results. *SIAM Journal on Computing*, Vol. 27, No. 3, pages 804–915, 1998. Extended abstract in *36th FOCS*, 1995.

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

[9] L. Carter and M. Wegman. Universal Hash Functions. *Journal of Computer and System Science*, Vol. 18, 1979, pages 143–154.

[10] G.J. Chaitin. On the Length of Programs for Computing Finite Binary Sequences. *Journal of the ACM*, Vol. 13, pages 547–570, 1966.

[11] B. Chor and O. Goldreich. On the Power of Two–Point Based Sampling. *Jour. of Complexity*, Vol 5, 1989, pages 96–106. Preliminary version dates 1985.

[12] T.M. Cover and G.A. Thomas. *Elements of Information Theory.* John Wiley & Sons, Inc., New-York, 1991.

[13] D. Gillman. A chernoff bound for random walks on expander graphs. In *34th IEEE Symposium on Foundations of Computer Science*, pages 680–691, 1993.

[14] O. Goldreich. A Note on Computational Indistinguishability. *Information Processing Letters*, Vol. 34, pages 277–281, May 1990.

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

[16] O. Goldreich. *Foundation of Cryptography: Basic Tools.* Cambridge University Press, 2001.

[17] O. Goldreich. *Foundation of Cryptography: Basic Applications.* Cambridge University Press, 2004.

[18] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *Journal of the ACM*, Vol. 33, No. 4, pages 792–807, 1986.

[19] O. Goldreich and L.A. Levin. Hard-core Predicates for any One-Way Function. In *21st ACM Symposium on the Theory of Computing*, pages 25–32, 1989.

[20] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Science*, Vol. 28, No. 2, pages 270–299, 1984. Preliminary version in *14th STOC*, 1982.

[21] J. Håstad, R. Impagliazzo, L.A. Levin and M. Luby. A Pseudorandom Generator from any One-way Function. *SIAM Journal on Computing*, Volume 28, Number 4, pages 1364–1396, 1999. Preliminary versions by Impagliazzo *et. al.* in *21st STOC* (1989) and Håstad in *22nd STOC* (1990).

[22] R. Impagliazzo and A. Wigderson. P=BPP if E requires exponential circuits: Derandomizing the XOR Lemma. In *29th ACM Symposium on the Theory of Computing*, pages 220–229, 1997.

[23] N. Kahale, Eigenvalues and Expansion of Regular Graphs. *Journal of the ACM*, Vol. 42 (5), pages 1091–1106, September 1995.

[24] D.E. Knuth. *The Art of Computer Programming*, Vol. 2 (*Seminumerical Algorithms*). Addison-Wesley Publishing Company, Inc., 1969 (first edition) and 1981 (second edition).

[25] A. Kolmogorov. Three Approaches to the Concept of "The Amount Of Information". *Probl. of Inform. Transm.*, Vol. 1/1, 1965.

[26] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1996.

[27] F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[28] L.A. Levin. Randomness Conservation Inequalities: Information and Independence in Mathematical Theories. *Information and Control*, Vol. 61, pages 15–37, 1984.

[29] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications.* Springer Verlag, August 1993.

[30] M. Luby and A. Wigderson. Pairwise Independence and Derandomization. TR-95-035, International Computer Science Institute (ICSI), Berkeley, 1995. ISSN 1075-4946.

[31] P.B. Miltersen and N.V. Vinodchandran. Derandomizing Arthur-Merlin Games using Hitting Sets. *Journal of Computational Complexity*, to appear. Preliminary version in *40th FOCS*, 1999.

[32] J. Naor and M. Naor. Small-bias Probability Spaces: Efficient Constructions and Applications. *SIAM Journal on Computing*, Vol 22, 1993, pages 838–856.

[33] N. Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica*, Vol. 11 (1), pages 63–70, 1991.

[34] N. Nisan. Pseudorandom Generators for Space Bounded Computation. *Combinatorica*, Vol. 12 (4), pages 449–461, 1992.

[35] N. Nisan. $\mathcal{RL} \subseteq \mathcal{SC}$. *Journal of Computational Complexity*, Vol. 4, pages 1-11, 1994.

[36] N. Nisan and A. Wigderson. Hardness vs Randomness. *Journal of Computer and System Science*, Vol. 49, No. 2, pages 149–167, 1994.

[37] N. Nisan and D. Zuckerman. Randomness is Linear in Space. *Journal of Computer and System Science*, Vol. 52 (1), pages 43–52, 1996.

[38] A.R. Razborov and S. Rudich. Natural Proofs. *Journal of Computer and System Science*, Vol. 55 (1), pages 24–35, 1997.

[39] O. Reingold. Undirected ST-Connectivity in Log-Space. In *37th ACM Symposium on the Theory of Computing*, pages 376–385, 2005.

[40] M. Saks and S. Zhou. RSPACE(S) $\subseteq$ DSPACE($S^{3/2}$). In 36th *IEEE Symposium on Foundations of Computer Science*, pages 344–353, 1995.

[41] R. Shaltiel. Recent Developments in Explicit Constructions of Extractors. *Bulletin of the EATCS 77*, pages 67–95, 2002.

[42] C.E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. Jour.*, Vol. 27, pages 623–656, 1948.

[43] R.J. Solomonoff. A Formal Theory of Inductive Inference. *Information and Control*, Vol. 7/1, pages 1–22, 1964.

[44] L. Trevisan. Constructions of Near-Optimal Extractors Using Pseudo-Random Generators. In *31st ACM Symposium on the Theory of Computing*, pages 141–148, 1998.

[45] C. Umans. Pseudo-random generators for all hardness. *Journal of Computer and System Science*, Vol. 67 (2), pages 419–440, 2003.

[46] L.G. Valiant. A theory of the learnable. *CACM*, Vol. 27/11, pages 1134–1142, 1984.

[47] A. Wigderson. The amazing power of pairwise independence. In *26th ACM Symposium on the Theory of Computing*, pages 645–647, 1994.

[48] A.C. Yao. Theory and Application of Trapdoor Functions. In *23rd IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.