Chapter 17

Derandomization and Extractors

[1] The concept of a *randomized algorithm*, though widespread, has both a logical and a practical difficulty associated with it.

The logical difficulty is best encapsulated in Einstein's famous claim: "God does not play dice." Do random events (such as the unbiased coin flip, that mainstay of randomized algorithms) truly exist in the world, or is the world deterministic?

The practical difficulties have to do with actually generating random bits. A randomized algorithm running on a modern computer could need hundreds of billions of random bits a second. Even if the world contains randomness —say, the ups and downs of the stock market — it may not have enough randomness to provide billions of uncorrelated random bits every second inside a tiny microprocessor. Current computing environments rely on shortcuts such as taking a small "fairly random looking" bit sequence—e.g., interval between the programmer's keystrokes measured in microseconds—and applying a deterministic generator to turn them into a longer sequence of "sort of random looking" bits. Recently-marketed devices use quantum phenomena to try to do better. But in all the above cases it is unclear how random and uncorrelated those bits really are.

Such logical and practical difficulties look deterring; the philosophical aspect alone has been on the philosophers' table for centuries. The results in the current chapter may be viewed as complexity theory's contribution to these questions.

First, we present a research area called *derandomization* that suggests, at least in the realm of polynomial-time computations, that randomness may not provide any benefit over determinism. If certain plausible complexity theoretic conjectures are true, then P = BPP. This further deepens the mystery about the role of randomness (or lack thereof) in our world.

At a practical level, we will show the existence of an algorithm that is an *extractor*: it allows us to extract provably random bits from any *weakly random* device. Of course, we cannot resolve the question of whether such weakly random devices exist; this is presumably a subject for physics or philosophy.

A central result in both areas is a beautiful construction of Nisan and Wigderson.

Example 17.1 We describe derandomization at an intuitive level using polynomial identity testing (Example 7.2 in Chapter 7), a main element of PCPs and Interactive proofs. Given

[1] This chapter written jointly with Luca Trevisan.

two degree d multivariate polynomials p, q, the test picks a random vector $\overline{a}$ and checks whether $p(\overline{a}) \neq q(\overline{a})$.

To derandomize this test we will show how to produce a small set of vectors $\overline{a_1}, \overline{a_2}, \ldots, \overline{a_N}$ such that for every input (p, q) if $p \neq q$ then $p(\overline{a_i}) \neq q(\overline{a_i})$ for at least one i. (A more general derandomization algorithm could pick $\overline{a_i}$'s after examining the input (p, q), but the algorithms of this chapter use the same $\overline{a_i}$'s for all inputs. In fact they use the same pseudorandom strings $\overline{a_i}$'s for all BPP problems; in this sense the derandomization is "black box.")

Note that the *existence* of the $\overline{a_i}$'s is easy to show; we did it while proving BPP $\subseteq$ P/poly (Theorem 7.2). An explicit, efficient construction is more difficult, and the one given here assumes the truth of certain complexity theoretic conjectures.

A mathematician thinking about polynomial identity testing may be forgiven for thinking that resolving complexity theoretic conjectures may be overkill for this derandomization. After all, the digits of transcendental numbers such as π seem to follow no pattern. It is conceivable that such deterministically generated bits (there are known algorithms that compute the first t bits of π in poly(t) time) would suffice to derandomize.

However, a recent result (Section 17.5) dashes such hopes: designing any deterministic polynomial time algorithm for polynomial identity testing requires resolving complexity-theoretic conjectures.

17.1 Pseudorandom Generators and Derandomization

The main tool in derandomization is a pseudorandom generator. We defined it earlier in the chapter on cryptography, but its use for derandomization presents new twists.

DEFINITION 17.1 *A mapping $G : \{0,1\}^t \to \{0,1\}^m$ is a (S, ϵ)-pseudorandom generator if for every circuit D of size at most S we have*

$$\left| \mathbf{Pr}_{r \in \{0,1\}^m}[D(r) = 1] - \mathbf{Pr}_{z \in \{0,1\}^t}[D(G(z)) = 1] \right| \leq \epsilon \tag{17.1}$$

If both t and $\mathcal{S}$ are mappings from integers to integers, then a $(\mathcal{S}(m), \epsilon)$-pseudorandom generator family is a family of generators $(G_m)_{m \geq 1}$ where each $G_m : \{0,1\}^{t(m)} \to \{0,1\}^m$ is a $(S(m), \epsilon)$-pseudorandom generator. The family is said to be computable in time $T(m)$ if there is a Turing machine that given $x \in \{0,1\}^{t(m)}$ computes $G_m(x)$.

REMARK 17.1 This definition breaks convention by defining running time as a function of output length and not of input length. This will be more convenient in derandomization.

The next Lemma is the starting point of most work on derandomization.

LEMMA 17.1
If there is a $(m^2, 1/10)$-pseudorandom generator family $G_m : \{0,1\}^{t(m)} \to \{0,1\}^m$ that is computable in $2^{O(t(m))}$ time, then BPP $\subseteq$ DTIME$(2^{O(t(n^{O(1)}))})$.

PROOF: A language L is in BPP if there is a polynomial time algorithm A and a polynomial $q()$ such that for every $x \in \{0,1\}^n$ we have

$$\mathbf{Pr}_{r\in\{0,1\}^{q(n)}}[A(x,r)=\chi_L(x)]\geq\frac{3}{4},$$

where $\chi_L(x)=1$ if $x\in L$ and 0 otherwise.

Using our usual transformation from Turing machine computations to circuits, we can for every input x, construct a circuit $C_x(r)$ that for any input r outputs $A(x,r)$. The size of C_x is some polynomial in n, say $p(n)$. Without loss of generality we assume $q(n)\geq\sqrt{p(n)}$ (i.e., $p(n)\leq(q(n))^2$) since if $q(n)$ is smaller we can input an r of length $\sqrt{p(n)}$ to the circuit anyway and it could choose to ignore some of the bits.

Let $m=q(n)$. Suppose we were to offer circuit C_x a pseudorandom input produced by applying G_m on a random string $y\in\{0,1\}^{t(m)}$. Since C_x has size at most m^2, its acceptance probability on these pseudorandom bits will be essentially the same as on truly random bits:

$$\left|\Pr_{r\in\{0,1\}^m}[C_x(r)=\chi_L(x)]-\Pr_{y\in\{0,1\}^{t(m)}}[C_x(G_m(y))=\chi_L(x)]\right|\leq\frac{1}{10},$$

implying that

$$\Pr_{y\in\{0,1\}^{t(m)}}[C_x(G_m(y))=\chi_L(x)]\geq\frac{3}{4}-\frac{1}{10}=\frac{13}{20}.$$

Thus by trying all $y\in\{0,1\}^{t(m)}$ —there are only $2^{t(m)}$ of them— we can in $2^{O(t(m))}$ time determine if $\Pr_{y\in\{0,1\}^{t(m)}}[C_x(G_m(y))]\geq 13/20$ and hence whether or not $x\in L$. □

The next Section will use complexity assumptions of varying strengths to construct pseudorandom generators satisfying the above condition with $t()$ ranging from $m^{o(1)}$ to $O(\log m)$.

This is a good place to contrast the pseudorandom generators of this chapter with those used in cryptography. For cryptographic applications, the seed length and the output length are polynomially related (i.e., $t(m)=m^\epsilon$ for some fixed $\epsilon>0$). The generators is computable in polynomial time and should fool circuits of superpolynomial size. For derandomization using Lemma 17.1, it suffices to have generators that run in time exponential in the seed length $t(m)$. The reason is that the procedure in the proof of Lemma 17.1 would take $2^{t(m)}$ time to try all seeds $y\in\{0,1\}^{t(m)}$ anyway, so there is no harm in letting it take $2^{t(m)}$ time to compute each $G_m(y)$ as well. In many constructions this running time $2^{t(m)}$ is superpolynomial in the output length m, whereas the generator is only required to "fool" circuits whose size is some fixed polynomial m^c. Thus the generator's running time considerably exceeds the size of the adversary circuit, which makes no sense in the cryptographic setting. For this reason the Nisan-Wigderson construction of this chapter will be different from the Blum-Micali-Yao generators encountered in the cryptography chapter.

17.1.1 Hardness and Derandomization

We will need the following definition of hardness.

DEFINITION 17.2 (HARDNESS) *A boolean function f is (S,ϵ) hard if for every circuit of size $\leq S$, $\mathbf{Pr}_x[C(x)=f(x)]\leq 1/2+\epsilon$. The* hardness *of f, denoted $H(f)$, is the largest S*

such that for every circuit C of size $\leq S$ we have $\mathbf{Pr}_x[f(x) = C(x)] \leq 1/2 + 1/S$. (In other words, $H(f) = \max\{S : f \text{ is } (S, 1/S)\text{-hard}\}$.)

EXAMPLE 17.2 If $g_n : \{0,1\}^n \to \{0,1\}^n$ is a one-way permutation, then the Goldreich-Levin Theorem (Theorem 9.5) implies that function $f_{2n} : \{0,1\}^{2n} \to \{0,1\}^{2n}$ defined as $f(x,r) = (g^{-1}(x) \odot r)$ is $(n^c, 1/n^c)$ hard for every $c > 1$.

Earlier, while discussing Yao's XOR lemma (Theorem 16.2), we had defined hardness in a more general way. However, Yao's Lemma shows how to turn a weakly hard function into a strongly hard function, so we can now simplify notation to deal only with strongly hard functions. In particular, note that the notion of $H(f)$ makes little sense for a weakly hard function, when it may be as low as 2.

The main result on derandomization in this chapter is the following.

THEOREM 17.2
Suppose there is a decision problem $L \in$ DTIME$(2^{O(n)})$ such that, for every n, L restricted to inputs of length n has hardness at least $h(n)$. Then for every l there is a $(m^2, 1/m)$ pseudorandom generator $G : \{0,1\}^t \to \{0,1\}^m$ computable in $2^{O(t)}$ time, where $m = \Omega(\sqrt{h(l)})$ and $t = O(l^2/\log m)$.

Together with Lemma 17.1 this theorem yields complexity assumptions under which BPP has nontrivial derandomizations.

COROLLARY 17.3
Suppose there is a problem $L \in$ EXPTIME having superpolynomial hardness. Then BPP $\subseteq$ DTIME$(2^{n^{o(1)}})$.

COROLLARY 17.4
Suppose there is a problem $L \in$ DTIME$(2^{O(n)})$ and some constant $\delta > 0$, such that for all sufficiently large n we have $H(L_n) \geq 2^{\delta n}$. Then P = BPP.

17.2 The Nisan-Wigderson Construction

How can we use a function $f : \{0,1\}^n \to \{0,1\}$ with hardness $S \gg n$ to produce pseudorandom bits that fool all circuits of size $S^{1/10}$ (say)? Here is an idea that has the right intuition, though it uses too much randomness.

Suppose the circuit needs $S^{1/10}$ random bits. Take $k = S^{1/10}$ random strings $x_1, x_2, \ldots, x_k$ in $\{0,1\}^n$ and offer $f(x_1), \ldots, f(x_k)$ to the circuit. We claim that the circuit will have difficulty distinguishing this sequence from a random sequence of k bits. Suppose not. By Yao's hybrid argument (see the proof of Theorem 9.3) the circuit can be modified so as to predict (with good odds) $f(x_i)$ given $f(x_1), f(x_2), \ldots, f(x_{i-1})$, for some $i \leq k$. By averaging there must be a way to fix $f(x_1), f(x_2), \ldots, f(x_{i-1})$ such that the remaining circuit —which has size $S^{1/10}$ and input only x_i— predicts f with reasonable odds on a random x_i, thus contradicting the assumption $H(f) > S$.

The problem with the above construction is, of course, that it uses nk random bits to produce k pseudorandom bits, and is thus useless. The key idea of the Nisan-Wigderson

construction is to generate $x_1, x_2, \ldots, x_k$ using a *combinatorial design* —a form of limited independence reminiscent of pairwise independence discussed earlier— so that they can be generated with much fewer than k bits, thus making the generator useful. The proof of correctness will follow the above sketch. The first step involving Yao's hybrid argument still works, and the second step involving the fixing of $f(x_1), \ldots, f(x_{i-1})$ needs a more careful argument since $x_1, \ldots, x_{i-1}$ depend upon x_i, though only weakly.

Now we define combinatorial designs. This is a broad term, but in this chapter refers to systems of sets with small pairwise intersections.

DEFINITION 17.3 (COMBINATORIAL DESIGN) *A family of subsets $(S_1, \ldots, S_m)$ of a universe U is a (l, α)-design if, for every i, $|S_i| = l$, and, for every $i \neq j$, $|S_i \cap S_j| \leq \alpha$.*

Combinatorial designs can be constructed in a relatively efficient way; the proof of the next Theorem appears in Section 17.6.

THEOREM 17.5
There is an algorithm that, for every integer l, $m \leq 2^l$, outputs an $(l, \log m)$ design $(S_1, \ldots, S_m)$ over the universe $[t]$, where $t = O(l^2/\log m)$; the running time is $O(2^t t m^2)$.

17.2.1 The Pseudorandom Generator

To present the Nisan-Wigderson construction, we will use the following notation: if z is a string in $\{0,1\}^t$ and $S \subset [t]$, then we denote by $z_{|S}$ the string of length $|S|$ obtained from z by selecting the bits indexed by S. For example if $z = (0,0,1,0,1,0)$ and $S = \{1,2,3,5\}$ then $z_{|S} = (0,0,1,1)$.

For a Boolean function $f : \{0,1\}^l \to \{0,1\}$, and a design $\mathcal{S} = (S_1, \ldots, S_m)$ over $[t]$, the Nisan-Wigderson generator is a function $NW_{f,\mathcal{S}} : \{0,1\}^t \to \{0,1\}^m$ defined as follows (see Figure 17.1):

$$NW_{f,\mathcal{S}}(z) = f(z_{|S_1}) \cdot f(z_{|S_2}) \cdots f(z_{|S_m}) \tag{17.2}$$

The intuition is that for each $i \neq j$, the bits $f(z|_{S_i})$ and $f(z|_{S_j})$ look fairly independent to small circuits when z is a random string. Imagine that the circuit is told $f(z|_{S_i})$ and asked to predict $f(z|_{S_j})$. Since $S_i \cap S_j$ is small, $f(z|_{S_i})$ holds information about only a small fraction of bits in $z|_{S_j}$. Furthermore, f is a hard function, so knowing a few bits in the input should give very little advantage in guessing the output $f(z|_{S_j})$. This intuition is formalized in the following lemma which says that if $NW_{f,\mathcal{S}}$ is not a pseudorandom generator, then f has low hardness.

LEMMA 17.6
Let $f : \{0,1\}^l \to \{0,1\}$ be a Boolean function and $\mathcal{S} = (S_1, \ldots, S_m)$ be a $(l, \log m)$ design over $[t]$. Suppose circuit $D : \{0,1\}^m \to \{0,1\}$ has size m^2 and it distinguishes the output of $NW_{f,\mathcal{S}}$ from truly random bits as follows

$$|\mathbf{Pr}_z[D(NW_{f,\mathcal{S}}(z)) = 1] - \mathbf{Pr}_r[D(r) = 1]| > \epsilon. \tag{17.3}$$

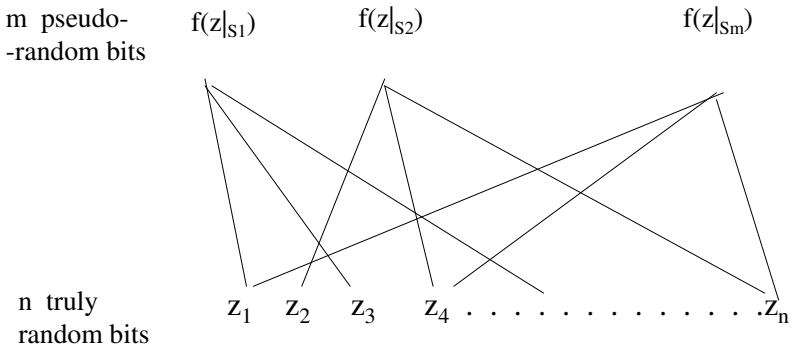

Figure 17.1: The Nisan-Wigderson generator takes a set of n bits, selects m subsets of these using a combinatorial design $\mathcal{S} = (S_1, \ldots, S_m)$ and applies the hard function f to each subset to get m pseudorandom bits.

Then there exists a circuit C of size $O(m^2)$ that has reasonable success in predicting f on a random input:

$$\mathbf{Pr}_x[C(x) = f(x)] \geq \frac{1}{2} + \frac{\epsilon}{m}$$

(Thus $H(f) \leq \max\left\{O(m^2), m/\epsilon\right\}$.)

PROOF: First, by putting a NOT gate on top of D if necessary, assume that the inequality in (17.3) is true even if we drop the "absolute value" sign from the left hand side:

$$\mathbf{Pr}_z[D(NW_{f,\mathcal{S}}(z)) = 1] - \mathbf{Pr}_r[D(r) = 1] > \epsilon.$$

We use Yao's hybrid argument from the proof of Theorem 9.3. We define a sequence of distributions such that the first one is the output of the generator, the last one is a random input, and consecutive distributions differ on just one bit position.

$$\begin{array}{lcl}
H_m & = & f(z|_{S_1})f(z|_{S_2})\ldots f(z|_{S_i})f(z|_{S_{i+1}})\ldots f(z|_{S_{m-1}})f(z|_{S_m}) \quad = NW(z) \text{ for random } z \\
H_{m-1} & = & f(z|_{S_1})f(z|_{S_2})\ldots f(z|_{S_i})f(z|_{S_{i+1}})\ldots f(z|_{S_{m-1}}) \quad r_m \\
\vdots & \vdots & \\
H_i & = & f(z|_{S_1})f(z|_{S_2})\ldots f(z|_{S_i}) \quad r_{i+1} \quad \ldots \quad r_{m-1} \quad r_m \\
H_{i-1} & = & f(z|_{S_1})f(z|_{S_2})\ldots \quad r_i \quad r_{i+1} \quad \ldots \quad r_{m-1} \quad r_m \\
\vdots & & \\
H_1 & = & f(z|_{S_1}) \quad r_2 \quad \ldots \quad r_i \quad r_{i+1} \quad \ldots \quad r_{m-1} \quad r_m \\
H_0 & = & r_1 \quad r_2 \quad \ldots \quad r_i \quad r_{i+1} \quad \ldots \quad r_{m-1} \quad r_m \quad = \text{ uniform}
\end{array}$$

The Lemma hypothesis may be rephrased as

$$\mathbf{Pr}[D(H_m) = 1] - \mathbf{Pr}[D(H_0) = 1] > \epsilon.$$

Rewriting the left hand side as the telescoping sum $\mathbf{Pr}[D(H_m) = 1] - \mathbf{Pr}[D(H_{m-1}) = 1] + \mathbf{Pr}[D(H_{m-1})] - \mathbf{Pr}[D(H_{m-2})] + \cdots - \mathbf{Pr}[D(H_0) = 1]$, we conclude by averaging there is an i such that

$$\mathbf{Pr}[D(H_i) = 1] - \mathbf{Pr}[D(H_{i-1}) = 1] > \frac{\epsilon}{m} \tag{17.4}$$

Now we want to construct a predictor for f. Let α denote $\Pr_x[f(x) = 0]$. Consequently, when z is random, the probability that $f(z|_{S_i}) = 0$ is also α. For $b_1, b_2 \in \{0, 1\}$ denote by p_{b_1,b_2} the probability (over choice of z and $r_1, r_2, \ldots$) that D outputs 1 on an input chosen from H_{i-1} in which $r_i = b_2$, and z is such that $f(z|_{S_i}) = b_1$. We have four possible disjoint events given by all 4 combinations of values of $f(z|_{S_i})$ and r_i, so

$$\mathbf{Pr}[D(H_i) = 1] = \alpha p_{00} + (1 - \alpha)p_{11} \tag{17.5}$$

$$\mathbf{Pr}[D(H_{i-1}) = 1] = \frac{\alpha}{2}(p_{00} + p_{01}) + \frac{1 - \alpha}{2}(p_{10} + p_{11}). \tag{17.6}$$

From (17.4) we conclude that

$$\frac{\alpha}{2}(p_{00} - p_{01}) + \frac{1 - \alpha}{2}(p_{11} - p_{10}) \geq \frac{\epsilon}{m}. \tag{17.7}$$

The random events in (17.4) and (17.7) depend upon z and $r_i, r_{i+1}, \ldots, r_m$. For notational ease assume that set S_i is $\{1, 2, \ldots, l\}$, namely, $z|_{S_i} = z_1 z_2 \cdots z_l$. Then averaging implies that there is a way to fix the remaining $t - l$ bits, say to $z_{l+1} = b_{l+1}, z_{l+2} = b_{l+2}, \ldots, z_t = b_t$ and also fix $r_i, r_{i+1}, \ldots, r_m$ so that the advantage in (17.7) is maintained[2]. Now the only random choice remaining is that of $z_1, z_2, \ldots, z_l \in \{0, 1\}$.

Each of the strings $z|_{S_j}$ for $j = 1, \ldots, i-1$ has been largely fixed, except for the parts that come from $S_j \cap S_i$, namely, at most $\log m$ bits. Thus each of $f(z|_{S_1}), f(z|_{S_2}), \ldots, f(z|_{S_{i-1}})$ has been reduced to some function on a subset of $\log m$ bits out of $z_1 z_2 \cdots z_l$, and is therefore computable by a circuit of size at most $2^{\log m}/\log m < m$. Let $C_1, C_2, \ldots, C_{i-1}$ be these circuits; note that they all take $z_1 z_2 \cdots z_l$ as input.

Now we describe a prediction circuit for f. We actually describe two circuits and show by averaging that at least one is a good predictor. Each circuit has size at most $O(m^2)$.
Circuit 1: Given an l-bit string x, the circuit applies $C_1, C_2, \ldots, C_{i-1}$ on x to get an $(i-1)$-bit string u. Then it applies circuit D to the input $u\ 0 r_{i+1} \cdots r_m$, and flips the answer (i.e., outputs 1 iff D outputs 0).
Circuit 2: Similar to Circuit 1 but it uses 1 instead of 0 (i.e., it runs D on string $u\ 1 r_{i+1} \cdots r_m$) and does not flip the answer of D.

If x is random, the probability that the first circuit computes $f(x)$ correctly is

$$\alpha p_{01} + (1 - \alpha)(1 - p_{10})$$

and the probability for the second circuit is

$$\alpha(1 - p_{01}) + (1 - \alpha)p_{11}.$$

[2]We are using the obvious fact that if Y is a real-valued function of two random variables x, y then there is a choice c for y such that $E[Y(x, c)] \geq E[Y(x, y)]$. *Maybe this principle should be identified earlier in the book as an example of averaging.*

The sum of the two is

$$1 + \alpha(p_{00} - p_{01}) + (1 - \alpha)(p_{11} - p_{10}) \qquad \geq 1 + 2\frac{\epsilon}{m},$$

whence it follows that one of the two probabilities is at least $1/2 + \epsilon/m$. □

Actually by examining the proof of Lemma 17.6 it is easy to extract a slightly stronger statement as a corollary. It says that regardless of the complexity of $D()$, there is always an efficient reduction from the task of computing $D()$ to the task of computing a function having a non-trivial correlation with $f()$.

COROLLARY 17.7
Let $f : \{0,1\}^l \to \{0,1\}$ be a Boolean function and $\mathcal{S} = (S_1, \ldots, S_m)$ be a $(l, \log m)$ design over $[t]$. Suppose $D : \{0,1\}^m \to \{0,1\}$ is such that

$$|\mathbf{Pr}_r[D(r) = 1] - \mathbf{Pr}_z[D(NW_{f,\mathcal{S}}(z)) = 1]| > \epsilon \ .$$

Then there exists a circuit C of size $O(m^2)$ that maps l bits to m bits and satisfies

$$|\mathbf{Pr}_x[D(C(x)) = f(x)] - \frac{1}{2}| \geq \frac{\epsilon}{m}$$

PROOF: The construction of C follows from the proof of Lemma 17.6. It involves fixing an appropriate i, and fixing $r_{i+1}, r_{i+2}, \ldots, r_m$, constructing circuits $C_1, C_2, \ldots, C_{i-1}$ and finally, deciding upon one of Circuit 1 and Circuit 2 to do the computation. □

17.3 Weak Random Sources and Extractors

Thus far, our definition of probabilistic Turing machines assumes the existence of a source of uncorrelated random bits. Now we consider whether such a machine can be run using a weaker random source, whose output bits, though not completely random, nevertheless have significant "unpredictability".

Such a source, when asked to produce a string of length N, outputs a string according to some distribution on $\{0,1\}^N$. We will blur the distinction between a source, and this associated distribution. We will need to quantify how random such a source is. A popular method is *Shannon entropy*, defined as $\sum_{x \in \{0,1\}^N} p_x \log 1/p_x$, where $p_x = \mathbf{Pr}[x \text{ is output}]$ and by convention $p_x \log 1/p_x = 0$ when $p_x = 0$. However, in our setting, Shannon entropy is not very useful. Consider a distribution that outputs $(0, 0, \cdots, 0)$ with probability $1 - 1/\sqrt{N}$, and otherwise outputs a string picked uniformly from the remaining $2^N - 1$ strings of length N. The Shannon entropy is approximately $\frac{1}{2^N\sqrt{N}} \times 2^N \log \sqrt{N} 2^N$, which is about $\sqrt{N}$. Thus the entropy is fairly high, yet the output is almost always deterministic: $(0, 0, \ldots, 0)$.

DEFINITION 17.4 (MIN-ENTROPY) *The* min-entropy *of a distribution $\{p_x\}$ on $\{0,1\}^N$ is the largest real number k such $p_x \leq \frac{1}{2^k}$ for all $x \in B^N$.*

A source that produces such a distribution is called an (N, k)-source.

An equivalent phrasing of min-entropy is as $\min_x \{\log 1/p_x\}$. To get intuition about this measure, think of the amount of randomness "contained" in the outcome x of a random variable X as $\log 1/p_x$ bits. (After all, the distribution could place a probability p_x on $1/p_x$ strings; drawing a random sample would then need $\log 1/p_x$ random bits.) This gives another perspective on Shannon entropy: Shannon entropy is the *expected* randomness content of a string obtained from the source. It should now be clear why the bad example described above happens: the expected randomness content may not be achieved very often. (This is a well-known shortcoming of the —misleadingly named!— expected value.) By contrast, the output of an (N, k)-source always contains k bits of randomness.

Extracting random bits from an (N, k) source involves using the output of the source to produce a string of m bits that is uniformly (or almost uniformly) distributed in $\{0,1\}^m$. If possible, this lets us run any Turing machine computation that requires m random bits. How large can m be as a function of k? It is possible to argue (see exercises) that if the Turing machine is simulated in "black box" fashion (i.e., without looking at the working of the algorithm) and it runs in time T, then m cannot exceed $k + O(\log T)$. Furthermore, we give a simulation below in Section 17.4.2 that allows $m = k^{1/3}$. The proof can be tightened to work for $m = k^{1-\epsilon}$ for any $\epsilon > 0$.

These results give strong evidence that min-entropy is the "right" measure for the randomness content of a weak random source: if we want to simulate all probabilistic algorithms using such a source, it is necessary and sufficient that the source have sufficiently high min-entropy. Whether or not such weak sources exist is ultimately a question for physicists.

Below, we will often need the following easy fact about min-entropy:

LEMMA 17.8
If $X \in \{0,1\}^N$ is a random variable with min-entropy k then for any set $B \subseteq \{0,1\}^N$, $\Pr_{x \in X}[x \in B] \leq |B| \cdot 2^{-k}$.

The following measure will be used to quantify when two distributions are close.

DEFINITION 17.5 *For two random variables X and Y with range $\{0,1\}^m$, their* variational distance *is defined as $||X - Y|| = \max_{S \subseteq \{0,1\}^m} \{|\mathbf{Pr}[X \in S] - \mathbf{Pr}[Y \in S]|\}$. We say that two random variables are ϵ-close if their variational distance is at most ϵ.*

This concept has several restatements that are illuminating (see the exercises). First, $||X - Y|| \geq \epsilon$ iff there is a boolean function $D : \{0,1\}^m \to \{0,1\}$ such that $|\Pr_{x \in X}[D(x) = 1] - \Pr_{y \in Y}[D(y) = 1]| \geq \epsilon$. This will be useful in Lemma 17.9. Furthermore, $||X - Y|| = 1/2 \sum_x |\mathbf{Pr}[X = x] - \mathbf{Pr}[Y = x]|$, which is the usual ℓ_1 norm. (This is why we called $||X - Y||$ a "distance.")

17.3.1 Randomness Extractors

An extractor is a (deterministic) function that transforms a (N, k) source into an almost uniform distribution. The transformation is done by using a small number of additional truly random bits.

DEFINITION 17.6 *A function $Ext : \{0,1\}^N \times \{0,1\}^t \to \{0,1\}^m$ is a (k, ϵ) extractor if for any (N, k) source X, the distribution $Ext(X, U_t)$ is ϵ-close to U_m. Here U_l for $l = t, m$ denotes the uniform distribution over $\{0,1\}^l$.*

Equivalently, if $Ext : \{0,1\}^N \times \{0,1\}^t \to \{0,1\}^m$ is a (k,ϵ) extractor, then for every distribution X ranging over $\{0,1\}^N$ of min-entropy k, and for every $S \subseteq \{0,1\}^m$, we have

$$|\mathbf{Pr}_{a\in X, z\in\{0,1\}^t}[Ext(a,z) \in S] - \mathbf{Pr}_{r\in\{0,1\}^m}[r \in S]| \leq \epsilon$$

17.3.2 An extractor based upon Nisan-Wigderson

Now we describe an elegant construction of extractors due to Trevisan.

Suppose we are given a string x obtained from an (N,k)-source. How can we extract k random bits from it, given $O(\log N)$ truly random bits? Let us check that the trivial idea fails. Using $2\log N$ random bits we can compute a set of k (even $N-1$) indices that are uniformly distributed and pairwise independent. Maybe we should just output the corresponding bits of x? Unfortunately, the source is allowed to set $N-k$ bits (deterministically) to 0 so long as it makes the remaining k bits completely random. In that case the expected number of random bits in our sample is at most k^2/N, which is less than even 1 if $k < \sqrt{N}$.

This suggests an important idea: we should first apply some transformation on x to "smear out" the randomness, so it is not localized in a few bit positions. For this, we will use error-correcting codes. Recall that such codes are used to introduce error-tolerance when transmitting messages over noisy channels. Thus intuitively, the code must have the property that it "smears" every bit of the message all over the transmitted message.

Having done this, the construction uses a better sampling method than pairwise independent sampling, namely, the Nisan-Wigderson combinatorial design.

Nisan-Wigderson as a sampling method:

In (17.2) we defined a function $NW_{f,\mathcal{S}}(z)$ using any function $f : \{0,1\}^l \to \{0,1\}$ and a combinatorial design $\mathcal{S}$. Note that the definition does not require f to be a hard function.

Think of f as a string of length 2^l, namely, its truth table. (Likewise, we can think of any circuit with l-bit inputs and with 0/1 outputs as computing a string of length 2^l.) Given any z ("the seed"), $NW_{f,\mathcal{S}}(z)$ is just a method to use z to sample a sequence of m bits from f. This is completely analogous to pairwise independent sampling considered above.

Furthermore, Corollary 17.7 now has an information-theoretic interpretation. Suppose f is some string and D is some test that, given a string of m bits, outputs 0/1. (Let us not care about the complexity of D.) Suppose D distinguishes a random string of m bits from $NW_{f,\mathcal{S}}(z)$ where z is a random seed. Then there is a way to turn D (by combining it with a small circuit) into a black-box that computes f approximately: the black-box outputs a string of length 2^l that agrees with f in $1/2 + \epsilon/m$ fraction of bits.

Below, f will be of the form $\mathcal{C}(x)$, where $\mathcal{C}$ is an an error-correcting code and x is a string. If we have a black box computing f approximately then its output has hamming distance $1/2 + \epsilon/m$ to $\mathcal{C}(x)$. This is almost as good as computing $\mathcal{C}(x)$ exactly, since the errors can be corrected. (Warning: this last line is not completely accurate.)

Constructing the extractor

As mentioned above, every string $x \in \{0,1\}^{\bar{N}}$ may be viewed as a boolean function $< x >: \{0,1\}^{\log \bar{N}} \to \{0,1\}$ whose truth table is x. Let $l = \log \bar{N}$, and let $\mathcal{S} = (S_1, \ldots, S_m)$ be a $(l, \log m)$ design over $[t]$.

Let $\mathcal{C} : \{0,1\}^N \to \{0,1\}^{\bar{N}}$ be a polynomial time computable error-correcting code such that any ball of radius at most $1/2 - \delta$ (in the usual Hamming distance) contains at most $1/\delta^2$ codewords. Such codes exist with $\bar{n} = \text{poly}(n, 1/\delta)$.

The extractor $ExtNW : \{0,1\}^N \times \{0,1\}^t \to \{0,1\}^m$ is defined as

$$ExtNW_{\mathcal{C},\mathcal{S}}(x,z) = NW_{<\mathcal{C}(x)>,\mathcal{S}}(z) \ .$$

That is, $ExtNW$ encodes its first ("weakly random") input x using an error-correcting code, then uses Nisan-Wigderson sampling on the resulting string using the second ("truly random") input z as a seed.

LEMMA 17.9
For sufficiently large m and for $\epsilon > 2^{-m^2}$, $ExtNW_{\mathcal{C},\mathcal{S}}$ is a $(m^3, 2\epsilon)$-extractor.

PROOF: Fix a random variable X of min-entropy m^3. We wish to show that the distribution $ExtNW(a,z)$ where $a \in X, z \in \{0,1\}^t$ has variational distance at most ϵ to the uniform distribution. From our remarks after Definition 17.5, it suffices to show for each function $D : \{0,1\}^m \to \{0,1\}$ that

$$\left| \mathbf{Pr}_r[D(r) = 1] - \mathbf{Pr}_{a \in X, z \in \{0,1\}^t}[D(ExtNW(a,z)) = 1] \right| \leq 2\epsilon.$$

Fix any such function D. We say string a is *bad for D* if

$$\left| \mathbf{Pr}[D(r) = 1] - \mathbf{Pr}_{z \in \{0,1\}^t}[D(ExtNW(a,z)) = 1] \right| > \epsilon.$$

Let B be the set of bad a's for this D.

We show that B is small using a counting argument: we exhibit a 1-1 mapping from the set of bad a's to another set G, and prove G is small. Actually, here is G:

$$G = \left\{ \text{circuits of size } O(m^2) \right\} \times \{0,1\}^{2\log(m/\epsilon)} \times \{0,1\} \, .$$

The number of circuits of size $O(m^2)$ is $2^{O(m^2 \log m)}$, so $|G| \leq 2^{O(m^2 \log m)} \times 2(m/\epsilon)^2 = 2^{O(m^2 \log m)}$.

Let us exhibit a 1-1 mapping from B to G. When a is bad, Corollary 17.7 implies that there is a circuit C of size $O(m^2)$ such that either the circuit $D(C())$ or its negation agrees with a on a fraction $1/2 + \epsilon/m$ of its entries. (The reason we have to allow either $D(C())$ or its complement is the $|\cdot|$ sign in the statement of Corollary 17.7.) Let $w \in \{0,1\}^{\bar{N}}$ be the string computed by this circuit. Then the encoding of a, namely, $\mathcal{C}(a)$, lies inside a ball of radius $1/2 - \epsilon/m$ around w. By the assumed property of the code $\mathcal{C}$, at most other $(m/\epsilon)^2$ other codewords lie inside this ball. Hence a is completely specified by C, together with $2\log(m/\epsilon) + 1$ additional bits which specify whether we use $D(C())$ or its complement to compute w, and which of the $(m/\epsilon)^2$ codewords in the ball around w is $\mathcal{C}(a)$. Thus we have described the mapping from B to G.

We conclude that for any fixed D, there are at most $2^{O(m^2 \log m)}$ bad strings. The probability that an element a taken from X belongs to B is (by Lemma 17.8) at most

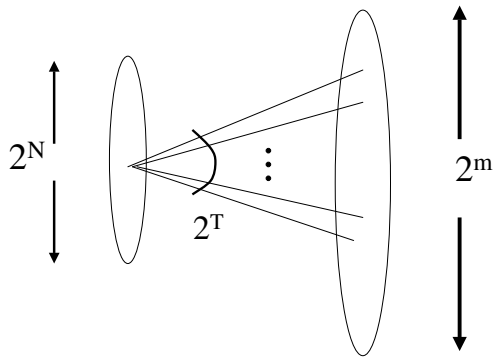

Figure 17.2: An extractor $Ext : \{0,1\}^N \times \{0,1\}^T \to \{0,1\}^m$ defines a bipartite graph where every node on the left has degree 2^T.

$2^{-m^3} \cdot 2^{O(m^2 \log m)} < \epsilon$ for sufficiently large m. We then have

$$\begin{aligned} & \left|\mathbf{Pr}_r[D(r)=1] - \mathbf{Pr}_{a\in X, z\in\{0,1\}^t}[D(ExtNW(a,z))=1]\right| \\ \leq & \sum_a \mathbf{Pr}[X=a]\left|\mathbf{Pr}[D(r)=1] - \mathbf{Pr}_{z\in\{0,1\}^t}[D(ExtNW(a,z))=1]\right| \\ \leq & \mathbf{Pr}[X\in B] + \epsilon \leq 2\epsilon, \end{aligned}$$

where the last line used the fact that if $a \notin B$, then by definition of B,
$\left|\mathbf{Pr}[D(r)=1] - \mathbf{Pr}_{z\in\{0,1\}^t}[D(ExtNW(a,z))=1]\right| \leq \epsilon$. □

The following theorem is an immediate consequence of the above lemma.

THEOREM 17.10
Fix a constant ϵ; for every N and $k = N^{\Omega(1)}$ there is a polynomial-time computable (k,ϵ)-extractor $Ext : \{0,1\}^N \times \{0,1\}^t \to \{0,1\}^m$ where $m = k^{1/3}$ and $t = O(\log N)$.

17.3.3 Extractors based upon hash functions

to be written

17.4 Applications of Extractors

Extractors are deterministic objects with strong pseudorandom properties. Many uses exist for them already, and many more will undoubtedly be found in future.

17.4.1 Graph constructions

An extractor is essentially a graph-theoretic object; see Figure 17.2. (In fact, extractors have been used to construct expander graphs.) Think of a (k,ϵ) extractor $Ext : \{0,1\}^N \times \{0,1\}^t \to \{0,1\}^m$ as a bipartite graph whose left side is $\{0,1\}^N$ and the right side $\{0,1\}^m$.

Each node on the left side has 2^t edges, labelled with strings in $\{0,1\}^t$. The source is allowed to pick any distribution on the left side with min-entropy at least k. The extractor's definition implies that picking a node according to this distribution and a random outgoing edge gives a node on the right that is essentially uniformly distributed.

17.4.2 Running randomized algorithms using weak random sources

We now describe how to use extractors to simulate probabilistic algorithms. Suppose that $A(\cdot,\cdot)$ is a probabilistic algorithm that on an input of length n uses $m = m(n)$ random bits, and suppose that for every x we have $\mathbf{Pr}_r[A(x,r) = \text{ right answer }] \geq 3/4$. (If A's answers are 0/1, then such algorithms can be viewed as defining a BPP language.) Suppose $Ext : \{0,1\}^N \times \{0,1\}^t \to \{0,1\}^m$ is a $(k, 1/4)$-extractor.

Consider the following algorithm A': on input $x \in \{0,1\}^n$ and given a string $a \in \{0,1\}^N$ from the weakly random source,

$$A'(x,a) = \text{majority value in } \left\{A(x, Ext(a,z)) : z \in \{0,1\}^t\right\} \tag{17.8}$$

The running time of A' is approximately 2^t times that of A. We show that if a comes from an $(n, k+2)$ source, then A' outputs the correct answer with probability at least 3/4.

Fix the input x. Let $R = \{r \in \{0,1\}^m : A(x,r) = \text{ right answer }\}$, and thus $|R| \geq \frac{3}{4}2^m$. Let B be the set of strings $a \in \{0,1\}^N$ for which the majority answer computed by algorithm A' is incorrect, namely,

$$\begin{aligned} B &= \left\{a : \mathbf{Pr}_{z\in\{0,1\}^t}[A(x, Ext(a,z)) = \text{ right answer}] < 1/2\right\} \\ &= \left\{a : \mathbf{Pr}_{z\in\{0,1\}^t}[Ext(a,z) \in R] < 1/2\right\} \end{aligned}$$

We show that $|B| \leq 2^k$. Let random variable Y correspond to picking an element uniformly at random from B. By definition, Y has min-entropy $\log B$, and thus may be viewed as a $(N, \log B)$-source. By definition of B,

$$\mathbf{Pr}_{a\in Y, z\in\{0,1\}^t}[Ext(a,z) \in R] < 1/2.$$

But $|R| = \frac{3}{4}2^m$, so we have

$$\left|\mathbf{Pr}_{a\in Y, z\in\{0,1\}^t}[Ext(a,z) \in R] - \mathbf{Pr}_{r\in\{0,1\}^m}[r \in R]\right| > 1/4,$$

which implies that the variational distance between the uniform distribution and $Ext(Y,z)$ is at least 1/4. Since Ext is a $(k, 1/4)$-extractor, Y must have min-entropy less than k. Hence $|B| \leq 2^k$.

The correctness of the simulation now follows since

$$\begin{aligned} \mathbf{Pr}_{a\in X}[A'(x,a) = \text{ right answer }] &= 1 - \mathbf{Pr}_{a\in X}[a \in B] \\ &\geq 1 - 2^{-(k+2)} \cdot |B| \geq 3/4, \quad \text{(by Lemma 17.8).} \end{aligned}$$

Thus we have shown the following.

THEOREM 17.11
Suppose A is a probabilistic algorithm running in time $T_A(n)$ and using $m(n)$ random bits on inputs of length n. Suppose we have for every $m(n)$ a construction of a $(k(n), 1/4)$-extractor $Ext_n : \{0,1\}^N \times \{0,1\}^{t(n)} \to \{0,1\}^{m(n)}$ running in $T_E(n)$ time. Then A can be simulated in time $2^t(T_A + T_E)$ using one sample from a $(N, k+2)$ source.

17.4.3 Pseudorandom generators for spacebounded computation

To be written

17.5 Derandomization requires circuit lowerbounds

We saw that proving strong circuit lowerbounds allows us to derandomize BPP (fully or partially). Now we prove a result in the reverse direction: derandomizing BPP requires circuit lowerbounds.

to be written: the Impagliazzo-Kabanets result.

17.6 Technical Details

PROOF:(Theorem 17.5) We describe a simple algorithm for the above. We divide the interval $[1, t]$ into l intervals of length $O(l/\log m)$. The algorithm picks the sets so that each set contains exactly one element from each interval. Let k be a constant to be determined later.

Input: l, m
Output: Family $S_1, S_2, \ldots, S_m$ satisfying the above constraints.

$t := k\frac{l}{\log m}$
Divide $\{1, 2, \ldots, t\}$ into l intervals of equal size.
Pick $S_1 \subseteq \{1, 2, \ldots, t\}$ arbitrarily so that it contains exactly one element from each interval.
for $i := 2$ to m **do**
    Pick $S_i \subseteq \{1, 2, \ldots, t\}$ arbitrarily such that there is exactly one element from each interval, and further $|S_i \cap S_j| \leq \log m$ for $j = 1, 2, \ldots, i-1$.
Output $S_1, S_2, \ldots, S_m$.

This is a greedy algorithm. If it reaches the end without failing, we are done. To prove that the algorithm never fails, it suffices to prove the following claim.

CLAIM 17.12
Let $S_1, S_2, \ldots, S_i$ for $i \leq m-1$ be subsets of $\{1, 2, \ldots, t\}$ with one element per interval, where $t = k\frac{l^2}{\log m}$. Then there is $S \subseteq \{1, 2, \ldots, t\}$ containing one element per interval so that $|S \cap S_j| \leq \log m$ for all $j = 1, 2, \ldots, i$.

PROOF: We show the existence of such a set S by showing a random experiment that gives such a set with non-zero probability. Construct S by choosing, for each interval, one element uniformly at

random from the interval. Now for a fixed j, consider the random variable $|S \cap S_j|$. This is the sum of l independent 0/1 random variables each with expectation $\frac{l}{t}$. Thus

$$E[|S \cap S_j|] = l\frac{l}{t} = \frac{\log m}{k}$$

Thus for an appropriate k the probability that $|S \cap S_j| \geq \log m$ can be made strictly smaller than $\frac{1}{m}$ by using Chernoff bounds. Thus, with a non-zero probability, $|S \cap S_j| \geq \log m$ for all j. □ This shows that the algorithm never fails. Further, we can verify that the algorithm has the required time complexity. □ *Need constants*

Problems

§1 Show how to go change (S, ϵ)-hard functions into functions with $H(f) >$?? using Yao's Lemma.

§2 Show that variational distance satisfies

- $||X - Y|| \geq \epsilon$ iff there is a boolean function $D : \{0,1\}^m \to \{0,1\}$ such that

$$\left| \Pr_{x \in X}[D(x) = 1] - \Pr_{y \in Y}[D(y) = 1] \right| \geq \epsilon.$$

- $||X - Y|| = 1/2 \sum_x |\mathbf{Pr}[X = x] - \mathbf{Pr}[Y = x]|$.

§3 Show that there is a constant $c > 0$ such that if an algorithm runs in time T and requires m random bits, then it is not possible in general to simulate it in a blackbox fashion using an (N, k) source if $m > k + c \log T$. (Hint???)

§4 Suppose boolean function f is (S, ϵ)-hard and let D be the distribution on m-bit strings defined by picking inputs $x_1, x_2, \ldots, x_m$ uniformly at random and outputting $f(x_1)f(x_2) \cdots f(x_m)$. Show that the variational distance between D and the uniform distribution is at most ϵm.