Contents

IAS/Park City Mathematics Series
Volume 00, 0000

Pseudorandomness – Part II

Luca Trevisan
Scribe: Kumar Saurabh

Introduction

About this Part

This series of lectures shows how pseudorandom generators can be applied to the task of deterministically (and efficiently) simulating probabilistic algorithms, and to the task of converting biased distributions into almost uniform distributions.

The applicability of pseudorandom generators to the deterministic simulation of probabilistic algorithms was already noticed by Yao [**Yao82**]. Recall that, in Part I, a pseudorandom generator was defined as a polynomial time procedure whose output is indistinguishable from uniform by adversaries running in polynomial time (where the degree of the polynomial can be arbitrarily large). As observed by Nisan and Wigderson [**NW94**], this definition imposes requirements that are not necessary in derandomization applications. In fact, it is sufficient to construct generators that are secure against adversaries running in some specified polynomial time (of fixed degree) and, more dramatically, it is admissible that the generator runs in time *exponential* in its input seed length. This relaxed requirements liberated the construction of pseudorandom generators from the framework of one-way functions, and Nisan and Wigderson [**NW94**] were able to present a construction that gave very efficient derandomization under complexity assumption about the class EXP of decision problems solvable in exponential time. A series of subsequent works, most notably [**BFNW93, Imp95, IW97, IW98**] showed how to use weaker complexity assumptions to achieve the same derandomization results of [**NW94**].

The task of converting biased distributions into almost uniform distributions is performed by procedures called *randomness extractors* [**Zuc96, NZ96**]. When interpreted in the proper way, the pseudorandom generator construction of [**NW94**] can also be used to construct efficient randomness extractors. This application has been discovered only recently [**Tre99**], and it is the base of most of the best current

[1]Computer Science Division, U.C. Berkeley.
E-mail address: `luca@eecs.berkeley.edu`.

extractor constructions, such as the ones in [**RRV99**] and [**ISW00**] (but not the one in [**RSW00**]).

Conventions

In the following lectures we will refer to "algorithms" and to their "running time" on particular inputs, without explicitly fixing a model of computation. This is done intentionally, to point out the model-independence of most of the results. The reader can instantiate "algorithm" and "running time" using any model of his or her choice that is polynomial-time equivalent to, say, single-tape deterministic Turing machines.

We will also need, at some point, to use a "non-uniform" measure of complexity. Here, for concreteness, we will use Boolean circuits with gates having fan-in 2 and arbitrary fan-out (see, e.g., [**Pap94**] for a description of this model). We will need the following facts about such circuits.

- If a circuit has s gates, then it can be described using $O(s \log s)$ bits. In particular, there are $2^{O(s \log s)}$ circuits of size $\leq s$.
- Every Boolean function on n inputs can be computed by a circuit of size $O(2^n)$. (In fact, a stronger result is known, but this will be enough for our purposes.)

Further Reading

A general perspective on pseudorandomness, derandomization, and randomness extraction is given by Goldreich in [**Gol99**, Chapter 3].

The original paper by Nisan and Wigderson [**NW94**] is still one of the best places to read about their construction; more detailed presentations will be found in the upcoming journal versions of [**Tre99, RRV99**].

Two alternative proofs of the major result of [**IW97**] appear in [**STV99**], of which a journal version is also upcoming. The proofs in [**STV99**] are somewhat simpler than the original one in [**IW97**].

A survey paper by Nisan [**Nis96**] (see also [**NTS98**]) gives an excellent introduction to the problem of random extraction, to the applications of randomness extractors, and to some techniques that are used to construct them. Probably [**Tre99**] and [**RRV99**] are the best places to read about the connection between pseudorandom generation and randomness extraction (Nisan's survey was written before the connection was discovered).

An interesting development not covered in these notes is the use of the Nisan-Wigderson generator to de-randomize bounded-round interactive proofs. This direction has been explored in [**AK97, KvM99**], with surprising results.

LECTURE 1
Deterministic Simulation of Randomized Algorithms

1. Probabilistic Algorithms versus Deterministic Algorithms

A probabilistic algorithm $A(\cdot,\cdot)$ is an algorithm that takes two inputs x and r, where x is an instance of some problem that we want to solve, and r is the output of a *random source*. A random source is an idealized device that outputs a sequence of bits that are uniformly and independently distributed. For example the random source could be a device that tosses coins, observes the outcome, and outputs it. A probabilistic algorithm A is good if it is efficient and if, say, for every x,

$$\mathop{\mathbf{Pr}}_r[A(x,r) = \text{ right answer for } x\ ] \geq \frac{3}{4}$$

We will typically restrict to the case where A solves a decision problem (e.g. it tests whether a given number is prime). In this case we say that a language L is in BPP if there is a polynomial time algorithm $A(\cdot,\cdot)$ (polynomial in the length of the first input) such that for every x

$$\mathop{\mathbf{Pr}}_r[A(x,r) = \chi_L(x)] \geq \frac{3}{4}$$

or, said, another way,

$$x \in L \Rightarrow \mathop{\mathbf{Pr}}_r[A(x,r) = 1] \geq \frac{3}{4}$$

and

$$x \notin L \Rightarrow \mathop{\mathbf{Pr}}_r[A(x,r) = 1] \leq \frac{1}{4}\ .$$

The choice of the constant 3/4 is clearly quite arbitrary. For any constant $1/2 < p < 1$, if we had defined BPP by requiring the probabilistic algorithm to be correct with probability st least p, we would have given an equivalent definition. In fact, for any polynomial p, it would have been equivalent to define BPP by asking the algorithm to be correct with probability at least $1/2 + 1/p(n)$, where n is the size of the input, and it would have also been equivalent if we had asked the algorithm to be correct with probability at least $1 - 1/2^{p(n)}$. That is, for any two polynomials p and q, if for a decision problem L we have a probabilistic polynomial time A that solves L on every input of length n with probability at least $1/2 + 1/p(n)$, then there is another probabilistic algorithm A', still running in polynomial time, that solves L on every input of length n with probability at least $1 - 2^{-q(n)}$.

For quite a few interesting problems, the only known polynomial time algorithms are probabilistic. A well-known example is the problem of testing whether a given integer is a prime number or not (note that in this case the size of the input is the number of digits of the integer). Another example is the problem of extracting "square roots" modulo a prime, i.e. to find solutions, if they exist, to equations of the form $x^2 = a \pmod{p}$ where p and a are given, and p is prime. More generally, there are probabilistic polynomial time algorithms to find roots of polynomials modulo a prime. There is no known deterministic polynomial time algorithm for any of the above problems.

It is not clear whether the existence of such probabilistic algorithms suggests that probabilistic algorithms are inherently more powerful than deterministic ones, or that we have not been able yet to find the best possible deterministic algorithms for these problems. In general, it is quite an interesting question to determine what is the relative power of probabilistic and deterministic computations. This question is the main motivations for the results described in this Part.

1.1. A trivial deterministic simulation

Let A be a probabilistic algorithm that solves a decision problem L. On input x of length n, say that A uses a random string r of length $m = m(n)$ and runs in time $T = T(n)$ (note that $m \leq T$).

It is easy to come up with a deterministic algorithm that solves L in time $2^{m(n)}T(n)$. On input x, compute $A(x, r)$ for every r. The correct answer is the one that comes up the majority of the times, so, in order to solve our problem, we just have to keep track, during the computation of $A(x, r)$ for every r, of the number of strings r for which $A(x, r) = 1$ and the number of strings r for which $A(x, r) = 0$.

Notice that the running time of the simulation depends exponentially on the number of random bits used by A, but only polynomially on the running time of A. In particular, if A uses a logarithmic number of random bits, then the simulation is polynomial. However, typically, a probabilistic algorithm uses a linear, or more, number of random bits, and so this trivial simulation is exponential. As we will see in the next section, it is not easy to obtain more efficient simulations.

1.2. Exponential gaps between randomized and deterministic procedures

For some computational problems (e.g. approximating the size of a convex body) there are probabilistic algorithms that work even if the object on which they operate is exponentially big and given as a black box; in some cases one can prove that deterministic algorithms cannot solve the same problem in the same setting, unless they use exponential time. Let us see a particularly clean (but more artificial) example of this situation.

Suppose that there is some function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ that is given as an oracle; we want to devise an algorithm that on input x finds an approximation (say, to within an additive factor 1/10) to the value $\mathbf{Pr}_y[f(x,y) = 1]$. A probabilistic algorithm would pick $O(1)$ points $y_1, \ldots, y_t$ at random, evaluate $f(x, y_i)$, and then output the fraction of i such that $f(x, y_i) = 1$. This will be an approximation to within 1/10 with good probability. However a deterministic subexponential algorithm, given x, can only look at a negligible fraction of the values $f(x, y)$.

Suppose that f is zero everywhere. Now consider the function $g(x, y)$ that is equal to f on all the points that our algorithm queries, and is 1 elsewhere (note that, by this definition, the queries of the algorithm on input x will be the same for f and g). If the algorithm takes sub-exponential time, g is almost everywhere one, yet the algorithm will give the same answer as when accessing f, which is everywhere zero. If our algorithm makes less than 2^{n-1} oracle queries, it cannot solve the problem with the required accuracy.

2. De-randomization Under Complexity Assumptions

It is still not known how to improve, in the general case, the deterministic simulation of Section 1.1, and the observation of Section 1.2 shows one of the difficulties in achieving an improvement. If we want to come up with a general way of transforming probabilistic procedures into deterministic sub-exponential procedures, the transformation cannot be described and analyzed by modeling in a "black box" way the probabilistic procedure.[1] If we want to deterministically and sub-exponentially simulate BPP algorithms, we have to exploit the fact that a BPP algorithm $A(\cdot, \cdot)$ is not an arbitrary function, but an efficiently computable one, and this is difficult because we still have a very poor understanding of the nature of efficient computations.

The results described in these notes show that it is indeed possible to deterministically simulate probabilistic algorithms in sub-exponential (or even polynomial) time, provided that certain complexity-theoretic assumptions are true. It is quite usual in complexity theory that, using reductions, one can show that the answer to some open question is implied by (or even equivalent to) the answer to some other question, however the nature of the results of these notes is somewhat unusual. Typically a reduction from a computational problem A to a problem B shows that if B has an efficient algorithm then A has also an efficient algorithm, and, by counterpositive, if A is intractable then B is also intractable. In general, using reductions one shows that algorithmic assumptions imply algorithmic consequences, and intractability assumptions imply intractability consequences. In these notes we will see instead that the existence of efficient derandomized algorithms is implied by the *intractability* of some other problem, so that a hardness condition implies an algorithm consequence.

In the next section we will introduce some notation about computational problems and complexity measures, and then we will state some results about conditional de-randomization.

2.1. Formal Definitions of Complexity Measures and Complexity Classes

For a decision problem L and an integer n we denote by L_n the restriction of L to inputs of length n. It will be convenient to think of L_n as a Boolean function $L_n : \{0,1\}^n \to \{0,1\}$ (with the convention that $x \in L_n$ if and only if $L_n(x) = 1$).

[1] More precisely, it is impossible to have a sub-exponential time deterministic "universal derandomization procedure" that given x and oracle access to an arbitrary function $A(\cdot, \cdot)$ outputs 1 when $\mathbf{Pr}_r[A(x,r) = 1] \geq 3/4$ and outputs 0 when $\mathbf{Pr}_r[A(x,r) = 1] \leq 1/4$. In fact, more generally, it is impossible to give sub-exponential time algorithms for all BPP problems by using "relativizing" techniques. It is beyond the scope of these notes to explain what this means, and why it is more general. "Relativizations" are discussed in [**Pap94**], where it is possible to find pointers to the relevant literature.

For a function $f : \{0,1\}^n \to \{0,1\}$, consider the size of the smallest circuit that solves f; denote this number by $CC(f)$. By definition, we have that if C is a circuit with n inputs of size less than $CC(f)$ then there exists an $x \in \{0,1\}^n$ such that $C(x) \neq f(x)$.

Consider now, for every n, what is the minimum s such that there is a circuit C of size s such that $\mathbf{Pr}_{x\in\{0,1\}^n}[C(x) = f(x)] \geq 1/2 + 1/s$; denote this number by $H(f)$.

Recall that $DTIME(T(n))$ is the class of decision problems that can be solved by deterministic algorithms running in time at most $T(n)$ on inputs of length n. We have the classes $\mathsf{E} = DTIME(2^{O(n)})$ and $\mathsf{EXP} = DTIME(2^{n^{O(1)}})$.

2.2. Hardness versus Randomness

From our previous arguments, we have $\mathsf{BPP} \subseteq \mathsf{EXP}$. Since there are settings where probabilistic procedures require exponential time to be simulated, one would conjecture that $\mathsf{BPP} \not\subseteq 2^{n^{o(1)}}$; on the other hand, BPP seems to still represent a class of feasible computations, and it would be *very* surprising if $\mathsf{BPP} = \mathsf{EXP}$. As we will see in a moment, something is wrong with the above intuition. Either $\mathsf{BPP} = \mathsf{EXP}$, which sounds really impossible, or else it must be the case that BPP has sub-exponential time deterministic algorithms (that will work well only on average, but that would be quite remarkable enough).

Theorem 1 ([**IW98**])**.** *Suppose* $\mathsf{BPP} \neq \mathsf{EXP}$*; then for every* BPP *language* L *and every* $\varepsilon > 0$ *there is a deterministic algorithm* A *that works in time* 2^{n^ε} *and, for infinitely many* n*, solves* L *on a fraction* $1 - 1/n$ *of the inputs of length* n*.*

This gives a non-trivial simulation of BPP under an uncontroversial assumption. We can also get an optimal simulation of BPP under an assumption that is much stronger, but quite believable.

Theorem 2 ([**IW97**])**.** *Suppose there is a problem* L *in* E *and a fixed* $\delta > 0$ *such that, for all sufficiently large* n*,* $CC(L_n) \geq 2^{\delta n}$*; then* $\mathsf{P} = \mathsf{BPP}$*.*

We will call the statement "there is a problem L in E and a fixed $\delta > 0$ such that, for all sufficiently large n, $CC(L_n) \geq 2^{\delta n}$" the "IW assumption." Note that if the IW assumption is true, then it is true in the case where

$$L = \{(M, x, 1^k) : \text{ machine } M \text{ halts within } 2^k \text{ steps on input } x\ \}$$

Notice also that L cannot be solved by algorithms running in time $2^{o(n)}$, and so it would be a little bit surprising if it could be solvable by circuits of size $2^{o(n)}$, because it would mean that, for general exponential time computations, non-uniformity buys more than a polynomial speed-up. In fact it would be very surprising if circuits of size $2^{.99n}$ existed for L.

The two theorems that we just stated are the extremes of a continuum of results showing that by making assumptions on the hardness of problems in E and EXP it is possible to devise efficient deterministic algorithms for all BPP problems. The stronger the assumption, the more efficient the simulation.

Notice that the assumption in Theorem 2 is stronger than the assumption in Theorem 1 in two ways, and that, similarly, the conclusion of Theorem 2 is stronger than the conclusion in Theorem 1 in two ways. On the one hand, the assumption in Theorem 2 refers to circuit size, that is, to a non-uniform measure of complexity, whereas the assumption in Theorem 1 uses a uniform measure of complexity (running time of probabilistic algorithms). This difference accounts for the fact that the conclusion of Theorem 2 gives an algorithm that works for all inputs, while the conclusion of Theorem 1 gives an algorithm that works only for most inputs. The other difference is that Theorem 2 assumes exponential hardness, while Theorem 2 assumes only super-polynomial hardness. This is reflected in the running time of the consequent deterministic simulations (respectively, polynomial and sub-exponential).

When one makes the non-uniform assumption that there is a problem in E that requires circuits of size $s(n)$, then the consequence is a deterministic simulation of BPP in time roughly $2^{s^{-1}(n^{O(1)})}$ [**ISW99**]. So if one assumes that E requires super-polynomial circuits, BPP can be simulated in time $2^{n^{o(1)}}$, if one assumes that E requires circuits of size $2^{\Omega(n)}$ then the simulation runs in time $n^{O(1)}$, if one assumes that E requires circuits of size $n^{\log n}$ then the simulation runs in time $2^{2^{O(\sqrt{\log n})}}$, and so on. The result of [**IW98**] does not scale up so well when one is willing to make stronger uniform assumptions. In particular, the following is an open question.

Conjecture 3. *Suppose* $\mathsf{E} \not\subseteq \bigcap_{\delta>0} BPTIME(2^{\delta n})$*; then for every* BPP *language* L *there is a deterministic polynomial time algorithm* A *that, for infinitely many* n*, solves* L *on a fraction* $1 - 1/n$ *of the inputs of length* n*.*

LECTURE 2
The Nisan-Wigderson Generator

In this lecture we will review the notion of a pseudorandom generator, and we will see that in order to prove Theorem 2 it is enough to prove that the IW assumption implies the existence of pseudorandom generators with certain parameters. We will then state two results that, combined, give such an implication. One of these results is the Nisan-Wigderson pseudorandom generator construction. We will start developing some intuition about this construction, and we will analyze it in the next lecture.

1. Pseudorandom Generators

We say that a function $G : \{0,1\}^t \to \{0,1\}^m$ is a (s,ε)-pseudorandom generator if for every circuit D of size $\leq s$ we have

$$|\mathop{\mathbf{Pr}}_r[D(r)=1] - \mathop{\mathbf{Pr}}_z[D(G(z))=1]| \leq \varepsilon$$

Suppose that we have a probabilistic algorithm A such that for inputs x of length n the computation $A(x,\cdot)$ can be performed by a circuit of size $s(n)$; suppose that for every x we have $\mathop{\mathbf{Pr}}_r[A(x,r) = \text{ right answer }] \geq 3/4$, and suppose that we have a $(s,1/8)$ pseudorandom generator $G : \{0,1\}^{t(n)} \to \{0,1\}^{m(n)}$. Then we can define a new probabilistic algorithm A' such that $A'(x,z) = A(x,G(z))$. It is easy to observe that for every x we have

$$\mathop{\mathbf{Pr}}_z[A'(x,z) = \text{ right answer }] \geq 5/8$$

and that, using the trivial derandomization we can get a deterministic algorithm A'' that always works correctly and whose running time is 2^t times the sum of the running time of A plus the running time of G.

If t is logarithmic in m and s, and if G is computable in poly(m,s) time, then the whole simulation runs in deterministic polynomial time. Notice also that if we have a (s,ε)-pseudorandom generator $G : \{0,1\}^t \to \{0,1\}^m$, then for every $m' \leq m$ we also have, for a stronger reason, a (s,ε) pseudorandom generator $G' : \{0,1\}^t \to \{0,1\}^{m'}$ (G' just computes G and omits the last $m - m'$ bits of the output). So there will be no loss in generality if we consider only generators for the special case where, say, $s = 2m$. (This is not really necessary, but it will help reduce the number of parameters in the statements of theorems.) We have the following easy theorem.

Theorem 4. *Suppose there is a family of generators $G_m : \{0,1\}^{O(\log m)} \to \{0,1\}^m$ that are computable in* $\text{poly}(m)$ *time and that are* $(2m, 1/8)$*-pseudorandom; then* $\mathsf{P} = \mathsf{BPP}$.

Of course this is only a sufficient condition. There could be other approaches to proving (conditionally) $\mathsf{P} = \mathsf{BPP}$, without passing through the construction of such strong generators. Unfortunately we hardly know of any other approach, and anyway the (arguably) most interesting results are proved using pseudorandom generators.[1]

2. The two main theorems

2.1. The Nisan-Wigderson Theorem

Theorem 5 (Special case of [**NW94**]). *Suppose there is $L \in \mathsf{E}$ and $\delta > 0$ such that, for all sufficiently large n, $H(L_n) \geq 2^{\delta n}$; then there is a family of generators $G_m : \{0,1\}^{O(\log m)} \to \{0,1\}^m$ that are computable in* $\text{poly}(m)$ *time and that are* $(2m, 1/8)$*-pseudorandom (in particular,* $\mathsf{P} = \mathsf{BPP}$*).*

Notice the strength of the assumption. For almost every input length n, our problem has to be so hard that even circuits of size $2^{\delta n}$ have to be unable to solve the problem correctly on more than a fraction $1/2 + 2^{-\delta n}$ of the inputs. A circuit of size 1 can certainly solve the problem on a fraction at least $1/2$ of the inputs (either by always outputting 0 or by always outputting 1). Furthermore, a circuit of size 2^n always exist that solves the problem on every input. A circuit of size $2^{\delta n}$ can contain, for example, the right solution to our problem for every input whose first $(1 - \delta)n$ bits are 0; the circuit can give the right answer on these $2^{\delta n}$ inputs, and answer always 0 or always 1 (whichever is better) on the other inputs. This way the circuit is good on about a fraction $1/2 + 2^{-(1-\delta)n}$ of the inputs. So, in particular, for every problem, there is a circuit of size $2^{n/2}$ that solves the problem on a fraction $1/2 + 2^{-n/2}$ of the inputs. It is somewhat more tricky to show that there is in fact even a circuit of size $2^{(1/3+o(1))n}$ that solves the problem on a fraction $1/2 + 2^{-(1/3+o(1))n}$ of the inputs, and this is about best possible for general problems [**ACR97**].

2.2. Worst-case to Average-case Reduction

Theorem 6 ([**BFNW93, Imp95, IW97**]). *Suppose there is $L \in \mathsf{E}$ and $\delta > 0$ such that, for all sufficiently large n, $CC(L_n) \geq 2^{\delta n}$; Then there is $L' \in \mathsf{E}$ and $\delta' > 0$ such that, for all sufficiently large n, $H(L'_n) \geq 2^{\delta' n}$.*

[1]Some exceptions are discussed below. Andreev et al. [**ACR98**] show that in order to deterministically simulate probabilistic algorithms it is enough to construct *hitting set generators*, a seemingly weaker primitive than a pseudorandom generator. The complicated proof of [**ACR98**] was simplified in subsequent work [**ACRT99, BF99, GW99**]. Andreev et al. [**ACR99**] also show how to construct hitting set generators, but only under very strong complexity assumptions. Miltersen and Vinodchandran [**MV99**] give a very elegant construction of hitting set generators, but it also requires a stronger hardness assumption than in [**IW97**]. On the other hand, [**MV99**] also gets a stronger conclusion, and, in particular, it is not known how to prove the main result of [**MV99**] (about the "derandomization" of two-rounds interactive proofs) using pseudorandom generators.

This is quite encouraging: the (believable) IW assumption implies the (a priori less believable) NW assumption. Notice how Theorem 2 follows from Theorems 5 and 6.

3. Error-Correcting Codes and Worst-Case to Average-Case Reductions

The purpose of this section is to give an overview of the proof of Theorem 6. The proof will not be the one of [**BFNW93, Imp95, IW97**], but rather the one of [**STV99**], and it will rely on the notion of an *error-correcting code*.

For two strings $x, y \in \{0,1\}^n$, their Hamming distance is the number of places where they differ, i.e., the number of indices i such that $x_i \neq y_j$. In the following we will consider the normalized Hamming distance (that we will just abbreviate with "distance"), defined as $d(x,y) = \mathbf{Pr}_{i \in \{1,\dots,n\}}[x_i \neq y_i]$, that is, the Hamming distance divided by n.

Consider a mapping $C : \{0,1\}^n \to \{0,1\}^{\bar{n}}$; such a mapping is called an error-correcting code with minimum distance γ if for any distinct $x, x' \in \{0,1\}^n$ we have $d(C(x), C(x')) \geq \gamma$. The term "error-correcting" comes from the following observation: suppose that we transmit $C(x)$ over a noisy channel, and that what is received at the other end of the channel is a string y such that $d(C(x), y) < \gamma/2$; then, at least in principle, it is still possible to reconstruct x from y, since, by triangle inequality, x will be the only possible string such that $d(C(x), y) < \gamma/2$.

Interestingly, for any $\gamma < 1/2$ there are polynomial-time computable codes with minimum distance γ, such that the decoding problem is also solvable in polynomial time (in fact there are codes that are both encodable and decodable in *linear* time). Perhaps even more surprisingly, if we are interested in decoding only a small part of the message (in the extreme, only one bit), then there are codes with *sub-linear* probabilistic decoding procedures.

Theorem 7. *For any fixed $\gamma < 1/4$ and for any sufficiently large n there is a code $C : \{0,1\}^n \to \{0,1\}^{\bar{n}}$ computable in $\mathrm{poly}(n)$ time (in particular, $\bar{n} = \mathrm{poly}(n)$) and a $\mathrm{poly}\log n$ time probabilistic algorithm A, such that for every $x \in \{0,1\}^n$, for any $y \in \{0,1\}^{\bar{n}}$ such that $d(C(x), y) \leq \gamma$, for any $i \in \{1,\dots,n\}$, we have*

$$\mathbf{Pr}[A(i,y) = x_i] \geq 1 - 1/4n$$

where the probability is taken over the internal random choices of the algorithm.

We leave as an exercise to prove the following consequence.

Theorem 8. *Suppose there is a problem L in E and a fixed $\delta > 0$ such that, for all sufficiently large n, $CC(L_n) \geq 2^{\delta n}$; Then there is a problem L' in E and a $\delta' > 0$ such that for all circuits C of size $\leq 2^{\delta' n}$ we have*

$$\mathbf{Pr}[C(x) = L_n(x)] \leq .76$$

Unfortunately the theorem cannot be extended to the case $\gamma > 1/4$. In order for the decoding problem to even be well-defined, we would need codes with minimum distance $> 2\gamma > 1/2$, but such codes do not exist (except for finitely many n). Suppose, for example, that we would like to deal with a channel that only guarantees that the received string is at distance at most $1/3$ from the transmitted codeword. Suppose that we are using a code of minimum distance $.49$. When we receive a string y, we know for sure that the only possibly decodings come from the set

$\{x : d(C(x), y) < 1/3\}$. We cannot argue anymore that the set contains only one element, however it would be useful to argue that it contains few elements. (Algorithmically, it would be nice to be able to reconstruct such a set efficiently given y.) The following theorem states that if the code has high minimum distance, then there is an upper bound on the number of codewords in such sets.

Theorem 9. *Let* $\mathbf{C} : \{0,1\}^n \to \{0,1\}^{\bar{n}}$ *be a code of minimum distance* $1/2 - \delta^2$. *Then for every* $y \in \{0,1\}^{\bar{n}}$*, there are at most* $1/\delta^2$ *elements* $x \in \{0,1\}^n$ *such that* $d(\mathbf{C}(x), y) \le 1/2 - \delta$.

So if we transmit $C(x)$ and we receive a string y that agrees with y on only a fraction $1/2 + \delta$ of the places, it is possible (at least in principle) to create a list of only $1/\delta^2$ possible decodings for y, and one of them is guaranteed to be x. Such a computational task is called "list-decoding." There are error-correcting codes with polynomial time encoding algorithms and polynomial time list decoding algorithms.

It is also possible to come up with codes having sublinear time list decoding algorithms, but even the statement of such a result is somewhat complicated. From the existence of such codes one can derive 6.

4. The Nisan-Wigderson Construction

The Nisan-Wigderson generator is based on the existence of a decision problem L in E such that for almost every input length l we have $H(L_l) \ge 2^{\delta l}$, yet there is a uniform algorithm that solves L_l in $2^{O(l)}$ time. Our goal is to use these assumptions on L_l to build a generator whose input seed is of length $O(l)$, whose output is of length $2^{\Theta(l)}$ and indistinguishable from uniform by adversaries of size $2^{\Theta(l)}$, and the generator should be computable in time $2^{O(l)}$.

As we will see in a moment, it is not too hard to construct a generator that maps l bits into $l+1$ bits, and whose running time and pseudorandomness are as required. Recall that in Part 1 we saw how to turn a pseudorandom generator that stretches its input by one bit into a pseudorandom generator with a much longer output. Unfortunately, the same approach will not work in our case.[2] We will then present the Nisan-Wigderson construction, and defer its analysis to the next lecture.

4.1. Impredictability versus Pseudorandomness

Let $f : \{0,1\}^l \to \{0,1\}$ be a function such that $H(f) \ge s$, and consider the pseudorandom generator $G : \{0,1\}^l \to \{0,1\}^{l+1}$ defined as $G(x) = x \cdot f(x)$, where '$\cdot$' is used to denote concatenation. We want to argue that G is a $(s-3, 1/s)$-pseudorandom generator.

The argument works by contradiction, and consists in the proof of the following result.

Lemma 10. *Let* $f : \{0,1\}^l \to \{0,1\}$. *Suppose that there is a circuit* D *of size* s *such that*

$$|\mathop{\mathbf{Pr}}_{x}[D(x \cdot f(x)) = 1] - \mathop{\mathbf{Pr}}_{x,b}[D(x \cdot b) = 1]| > \varepsilon$$

[2]The main difference with respect to the setting of Part 1 is that we allow the running time of the generator to be larger than the circuit size of the adversary. We will elaborate on the difference in Section 4.2.

then there is a circuit A of size $s+3$ such that

$$\mathop{\mathbf{Pr}}_x[A(x) = f(x)] > \frac{1}{2} + \varepsilon$$

Proof. First of all, we observe that there is a circuit D' of size at most $s+1$ such that

$$\mathop{\mathbf{Pr}}_z[D'(x \cdot f(x)) = 1] - \mathop{\mathbf{Pr}}_{x,b}[D'(x \cdot b) = 1] > \varepsilon \tag{1}$$

This is because Expression (1) is satisfied either by taking $D = D'$ or by taking $D = \neg D'$. A way to interpret Expression (1) is to observe that when the first l bits of the input of $D'()$ are a random string x, D' is more likely to accept if the last bit is $f(x)$ than if the last bit is random (and, for a stronger reason, if the last bit is $1 - f(x)$). This observation suggests the following strategy in order to use D' to predict f: given an input x, for which we want to compute $f(x)$, we guess a value b, and we compute $D'(x, b)$. If $D'(x, b) = 1$, we take it as evidence that b was a good guess for $f(x)$, and we output b. If $D'(x, b) = 0$, we take it as evidence that b was the wrong guess, and we output $1 - b$. Let A_b be the procedure that we just described. We claim that

$$\mathop{\mathbf{Pr}}_{x,b}[A_b(x) = f(x)] > \frac{1}{2} + \varepsilon \tag{2}$$

The claim is proved by the following derivation

$$\begin{aligned}
& \mathop{\mathbf{Pr}}_{x,b}[A_b(x) = f(x)] \\
= \; & \mathop{\mathbf{Pr}}_{x,b}[A_b(x) = f(x) | b = f(x)] \mathop{\mathbf{Pr}}_{x,b}[b = f(x)] \\
& + \mathop{\mathbf{Pr}}_{x,b}[A_b(x) = f(x) | b \neq f(x)] \mathop{\mathbf{Pr}}_{x,b}[b \neq f(x)] \\
= \; & \frac{1}{2} \mathop{\mathbf{Pr}}_{x,b}[A_b(x) = f(x) | b = f(x)] + \frac{1}{2} \mathop{\mathbf{Pr}}_{x,b}[A_b(x) = f(x) | b \neq f(x)] \\
= \; & \frac{1}{2} \mathop{\mathbf{Pr}}_{x,b}[D'(x,b) = 1 | b = f(x)] + \frac{1}{2} \mathop{\mathbf{Pr}}_{x,b}[D'(x,b) = 0 | b \neq f(x)] \\
= \; & \frac{1}{2} + \frac{1}{2} \mathop{\mathbf{Pr}}_{x,b}[D'(x,b) = 1 | b = f(x)] - \frac{1}{2} \mathop{\mathbf{Pr}}_{x,b}[D'(x,b) = 1 | b \neq f(x)] \\
= \; & \frac{1}{2} + \mathop{\mathbf{Pr}}_{x,b}[D'(x,b) = 1 | b = f(x)] \\
& - \frac{1}{2} \left(\mathop{\mathbf{Pr}}_{x,b}[D'(x,b) = 1 | b = f(x)] + \mathop{\mathbf{Pr}}_{x,b}[D'(x,b) = 1 | b \neq f(x)] \right) \\
= \; & \frac{1}{2} + \mathop{\mathbf{Pr}}_{x}[D'(x, f(x)) = 1] - \mathop{\mathbf{Pr}}_{x,b}[D'(x,b) = 1] \\
> \; & \frac{1}{2} + \varepsilon
\end{aligned}$$

From Expression (2) we can observe that there must be a $b_0 \in \{0, 1\}$ such that

$$\mathop{\mathbf{Pr}}_x[A_{b_0}(x) = f(x)] > \frac{1}{2} + \varepsilon$$

And A_{b_0} is computed by a circuit of size at most $s+3$ because $A_{b_0}(x) = b_0 \oplus (\neg D'(x, b_0))$, which can be implemented with two more gates given a circuit for D'. □

4.2. The Difference with the Blum-Micali-Yao Setting

We just saw how to construct a pseudorandom generator that maps l bits into $l+1$ bits, given a function $f : \{0,1\}^l \to \{0,1\}$ of high hardness. The construction is reminiscent of the construction of a similar generator given a one-way permutation and a hard predicate. In fact, if f is hard, it can be seen as the "hard-core predicate" of the identity function, and, from this perspective, the construction of the previous section is the same as the construction seen in Part 1. Of course the identity function is not a one-way permutation, and a function f of high hardness is not necessarily a hard-core predicate. The difference is that a hard-core predicate B for a permutation π is such that $B(x)$ is hard to compute given $\pi(x)$, *but* it is easy to compute given x. In our current scenario, f is just hard. This difference is reflected in the efficiency of the generator construction. In the case of one-way permutations and hard predicates, it is possible to have a (s, ε) generator that is computable in time significantly smaller than s. In our case, the running time of the generator has to be bigger than s.

This difference is very important. In Part 1 we saw how to get a pseudorandom generator with large output length given a pseudorandom generator whose output is only one bit longer than the input. The same construction fails in our setting.

In fact, starting from a generator that maps x in $x \cdot f(x)$, the "bootstrap" construction of Part 1 would create a generator that maps x in $f(x) \cdot f(x) \cdot f(x) \cdots$, that is certainly distinguishable from uniform.

The Nisan-Wigderson construction and its analysis are therefore quite different from what we have seen in Part 1, however, at a higher level of abstraction, there are similarities. Starting from a permutation π and a hard-core predicate B, the generator described in Part 1, on input x, would output $B(x) \cdot B(\pi(x)) \cdot B(\pi(\pi(x))) \cdots$, that is, it would evaluate the hard predicate on points obtained by repeatedly applying π to x. In the Nisan-Wigderson generator, the output is also the evaluation of the hard function f on points generated using the input seed. The difference is in the generation of the points. In the Nisan-Wigderson generator, the process by which the seed is converted into a series of evaluation points for f uses 'combinatorial designs," that we describe next.

4.3. Combinatorial Designs

Consider a family $(S_1, \dots, S_m)$ of subsets of an universe U. We say the family is a (l, α)-design if, for every i, $|S_i| = l$, and, for every $i \neq j$, $|S_i \cap S_j| \leq \alpha$.

Theorem 11. *For every integer l, fraction $\gamma > 0$, there is an $(l, \log m)$ design $(S_1, \dots, S_m)$ over the universe $[t]$, where $t = O(l/\gamma)$ and $m = 2^{\gamma l}$; such a design can be constructed in $O(2^t t m^2)$ time.*

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 $z_{|S} = (0,0,1,1)$.

4.4. The Nisan-Wigderson Generator

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

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

LECTURE 3
Analysis of the Nisan-Wigderson Generator

This lecture is devoted to the proof of the following result.

Lemma 12. *Let $f : \{0,1\}^l \to \{0,1\}$ be a Boolean function and $\mathbf{S} = (S_1, \ldots, S_m)$ be a $(l, \log m)$ design over $[t]$. Suppose $D : \{0,1\}^m \to \{0,1\}$ is such that*

$$|\underset{r}{\mathbf{Pr}}[D(r) = 1] - \underset{z}{\mathbf{Pr}}[D(NW_{f,\mathbf{S}}(z)) = 1]| > \varepsilon \ .$$

Then there exists a circuit C of size $O(m^2)$ such that

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

Proof. The main idea is that if D distinguishes $NW_{f,\mathbf{S}}(\cdot)$ from the uniform distribution, then we can find a bit of the output of the generator where this distinction is noticeable. On such a bit, D is distinguishing $f(x)$ from a random bit, and such a distinguisher can be turned into a predictor for f. In order to find the "right bit", we will use the *hybrid argument.* At this level of abstraction, the analysis is the same as the analysis of the Blum-Micali-Yao generator, however, as the analysis unfolds, we will see major differences.

Let us start with the hybrid argument. We define $m+1$ distributions $H_0, \ldots, H_m$; H_i is defined as follows: sample a string $v = NW_{f,\mathbf{S}}(z)$ for a random z, and then sample a string $r \in \{0,1\}^m$ according to the uniform distribution, then concatenate the first i bits of v with the last $m - i$ bits of r. By definition, H_m is distributed as $NW_{f,\mathbf{S}}(y)$ and H_0 is the uniform distribution over $\{0,1\}^m$.

Using the hypothesis of the Lemma, we know that there is a bit $b_0 \in \{0,1\}$ such that

$$\underset{y}{\mathbf{Pr}}[D'(NW_{f,\mathbf{S}}(y)) = 1] - \underset{r}{\mathbf{Pr}}[D'(r)] > \varepsilon$$

where $D'(x) = b_0 \oplus D(x)$.

We then observe that

$$\begin{aligned}
\varepsilon &\leq \underset{z}{\mathbf{Pr}}[D'(NW_{f,\mathbf{S}}(z)) = 1] - \underset{r}{\mathbf{Pr}}[D'(r)] \\
&= \mathbf{Pr}[D'(H_m) = 1] - \mathbf{Pr}[D'(H_0) = 1] \\
&= \sum_{i=1}^{m} (\mathbf{Pr}[D'(H_i) = 1] - \mathbf{Pr}[D'(H_{i-1}) = 1])
\end{aligned}$$

In particular, there exists an index i such that

$$\mathbf{Pr}[D'(H_i) = 1] - \mathbf{Pr}[D'(H_{i-1}) = 1] \geq \varepsilon/m \tag{3}$$

Now, recall that

$$H_{i-1} = f(z_{|S_1}) \cdots f(z_{|S_{i-1}}) r_i r_{i+1} \cdot r_m$$

and

$$H_i = f(z_{|S_1}) \cdots f(y_{|S_{i-1}}) f(y_{|S_i}) r_{i+1} \cdot r_m \ .$$

We can assume without loss of generality (up to a renaming of the indices) that $S_i = \{1, \ldots, l\}$. Then we can see $z \in \{0,1\}^t$ as a pair (x, y) where $x = z_{|S_i} \in \{0,1\}^l$ and $y = z_{|[t]-S_i} \in \{0,1\}^{t-l}$. For every $j < i$ and $z = (x, y)$, let us define $f_j(x, y) = f(z_{|S_j})$: note that $f_j(x, y)$ depends on $|S_i \cap S_j| \leq \log m$ bits of x and on $l - |S_i \cap S_j| \geq l - \log m$ bits of y. With this notation we have

$$\mathop{\mathbf{Pr}}_{x,y,r_{i+1},\ldots,r_m}[D'(f_1(x,y), \ldots, f_{i-1}(x,y), f(x), \ldots, r_m) = 1]$$

$$- \mathop{\mathbf{Pr}}_{x,y,r_{i+1},\ldots,r_m} D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1] > \varepsilon/m$$

That is, when D' is given a string that contains $f_j(x, y)$ for $j < i$ in the first $i - 1$ entries, and random bits in the last $m - i$ entries, then D' is more likely to accept the string if it contains $f(x)$ in the i-th entry than if it contains a random bit in the i-th entry. This is good enough to (almost) get a predictor for f. Consider the following algorithm:

```
Algorithm $A$
Input: $x \in \{0,1\}^l$
Pick random $r_i, \ldots, r_m \in \{0,1\}$
Pick random $y \in \{0,1\}^{t-l}$
Compute $f_1(x,y), \ldots, f_{i-1}(x,y)$
If $D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1$ output $r_i$
Else output $1 - r_i$
```

Let us forget for a moment about the fact that the step of computing $f_1(x, y), \ldots, f_{i-1}(x, y)$ looks very hard, and let us check that A is good predictor. Let us denote by

$A(x, y, r_1, \ldots, r_m)$ the output of A on input x and random choices $y, r_1, \ldots, r_m$.

$$\begin{aligned}
& \mathop{\mathbf{Pr}}_{x,y,r}[A(x,y,r) = f(x)] \\
= \; & \mathop{\mathbf{Pr}}_{x,y,r}[A(x,y,r) = f(x) | r_i = f(x)] \mathop{\mathbf{Pr}}_{x,r_i}[r_i = f(x)] \\
& + \mathop{\mathbf{Pr}}_{x,y,r}[A(x,y,r) = f(x) | r_i \neq f(x)] \mathop{\mathbf{Pr}}_{x,r_i}[r_i \neq f(x)] \\
= \; & \frac{1}{2} \mathop{\mathbf{Pr}}_{x,y,r}[D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1 | f(x) = r_i] \\
& + \frac{1}{2} \mathop{\mathbf{Pr}}_{x,y,r}[D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 0 | f(x) \neq r_i] \\
= \; & \frac{1}{2} + \frac{1}{2} \Big(\mathop{\mathbf{Pr}}_{x,y,r}[D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1 | f(x) = b] \\
& - \mathop{\mathbf{Pr}}_{x,y,r}[D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1 | f(x) \neq b] \Big) \\
= \; & \frac{1}{2} + \mathop{\mathbf{Pr}}_{x,y,r}[D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1 | f(x) = b] \\
& - \frac{1}{2} \Big(\mathop{\mathbf{Pr}}_{x,y,r}[D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1 | f(x) = b] \\
& + \mathop{\mathbf{Pr}}_{x,y,r}[D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1 | f(x) \neq b] \Big) \\
= \; & \frac{1}{2} + \mathbf{Pr}[D'(H_i) = 1] - \mathbf{Pr}[D'(H_{i-1}) = 1] \\
\geq \; & \frac{1}{2} + \frac{\varepsilon}{m}
\end{aligned}$$

So A is good, and it is worthwhile to see whether we can get an efficient implementation. We said we have

$$\mathop{\mathbf{Pr}}_{x,y,r_i,\ldots,r_m}[A(x,y,r) = f(x)] \geq \frac{1}{2} + \frac{\varepsilon}{m}$$

so there surely exist fixed values $c_i, \ldots, c_m$ to give to $r_i, \ldots, r_m$, and a fixed value w to give to y such that

$$\mathop{\mathbf{Pr}}_{x,r}[A(x, w, c_i, c_{i+1}, \ldots, c_m) = f(x)] \geq \frac{1}{2} + \frac{\varepsilon}{m}$$

At this point we are essentially done. Since w is fixed, now, in order to implement A, we only have to compute $f_j(x,w)$ given x. However, for each j, $f_j(x,w)$ is a function that depends only on $\leq \log m$ bits of x, and so is computable by a circuits of size $O(m)$. Even composing $i-1 < m$ such circuit, we still have that the sequence $f_1(x,w), \ldots, f_{i-1}(x,w), c_i, c_{i+1}, \ldots, c_m$ can be computed, given x, by a circuit C of size $O(m^2)$. Finally, we notice that at this point $A(x,w,c)$ is doing the following: output the xor between c_i and the complement of $D'(C(x))$. Since c_i is fixed, either $A(x,w,c)$ always equals $D(C(x))$, or one is the complement of the other. In either case the Lemma follows. □

At this point it should not be too hard to derive Theorem 5.

LECTURE 4
Randomness Extractors

See [**Nis96, Gol99**] to find references and proper credits about the material in this section.

1. Use of Weak Random Sources

Suppose that we have a probabilistic algorithm $A(\cdot, \cdot)$ that on inputs of length n runs in time $T(n)$ and uses $m(n)$ random bits. Instead of a perfect source of randomness, we assume that we have a source that produces an output containing some "impredictability," but that can still be very far from uniform. A very general way of modeling this source is to assume that on input 1^N it outputs a string in $\{0,1\}^N$, and that the output string has "some randomness" (a notion that we will formalize and quantify in a moment). Typically, a good way to quantify the amount of randomness, or impredictability, in a distribution, is to compute its (Shannon) entropy. For a random variable X whose range is $\{0,1\}^N$, its entropy is defined as $\mathbf{H}(X) = \sum_{a \in \{0,1\}^N} \mathbf{Pr}[X = a] \log(1/\mathbf{Pr}[X = a])$,

Shannon entropy is a very good measure in cases where one is allowed to take multiple samples from the distribution, but in our setting this is not the best measure. Consider for example a distribution X such that $X = (0, 0, \cdots, 0)$ with probability $1 - 1/\sqrt{N}$, and it is uniform with probability $1/\sqrt{N}$. Then its Shannon entropy is about $\sqrt{N}$, which is quite high, yet it is almost always a useless string of zeroes. It is a good intuition to think that the amount of randomness contained in the outcome a of a random variable X is $\log 1/\mathbf{Pr}[X = a]$. If X has Shannon entropy k, then *on average*, when we sample from X we get a value of "randomness" k, however it can be the case that with very high probability we get almost zero randomness, and with low probability we get high randomness. We would rather have a measure of randomness that guarantees to have almost always, or, even better, always, high randomness. This motivates the definition of *min-entropy*: a random variable X has min-entropy at least k if for every a in the range of X we have $\mathbf{Pr}[X = a] \leq 1/2^k$. That is, the min-entropy of X is $\min_a\{\log 1/\mathbf{Pr}[X = a]\}$.

Definition 13. *A random variable with range $\{0,1\}^N$ having min-entropy at least k will be called a (N, k)-source.*

Given one access to a (N, k) source, we would like to be able to simulate any probabilistic algorithm that uses m random bits, where m is close to k. If the simulation is "black box" and takes time T, one can argue that $m \leq k + O(\log T)$. We will not define formally what a black-box simulation is, but we will develop simulations that are black box, so it will come as no surprise that our simulations will work only for m smaller than k, in fact only for m smaller than $k^{1/3}$. (This is partly due to oversimplifications in the analysis; one could get $k^{.99}$ with almost the same proof.)

2. Extractors

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

Formally, we have the following definition.

Definition 14. *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.*

Definition 15. *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 we have that $Ext(X, U_l)$ is ε-close to uniform, where U_l is 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

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

3. Applications

See [**Nis96**] for an extensive survey. Here we present only one application. Another notable application is the construction of expanders.

Suppose that $A(\cdot, \cdot)$ is a probabilistic algorithm that on an input of length n uses $m(n)$ random bits, and suppose that for every x we have $\mathop{\mathbf{Pr}}_r[A(x, r) = \text{ right answer }] \geq 3/4$. 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 weakly random $a \in \{0,1\}^N$, A' computes $A(x, Ext(a, z))$ for every $z \in \{0,1\}^t$, and then it outputs the answer that appears the majority of such 2^t times. We want to argue that A' is correct with high probability if a is sampled from a weak random source of entropy slightly higher than k. Let us fix the input x. Consider the set B of strings $a \in \{0,1\}^N$ for which the algorithm A' makes a mistake:

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

Consider the random variable X that is uniformly distributed over B (clearly, X has min-entropy $\log B$). Then we have

$$\mathop{\mathbf{Pr}}_{a \in X, z \in \{0,1\}^t}[A(x, Ext(a, z)) = \text{ right answer }] < 1/2$$

and so

$$|\mathop{\mathbf{Pr}}_{a\in X, z\in\{0,1\}^t}[A(x, Ext(a,z)) = \text{ right answer }] - \mathop{\mathbf{Pr}}_{r}[A(x,r) = \text{ right answer }]| > 1/4$$

and then it follows form the property of Ext that X must have min-entropy less than k, that is $|B| \leq 2^k$.

Let now X be a $(N, k+2)$-source, and let us execute algorithm A' using X. Then

$$\mathop{\mathbf{Pr}}_{a\in X, z\in\{0,1\}^t}[A(x, Ext(a,z)) = \text{ right answer }] = 1 - \mathop{\mathbf{Pr}}_{a\in X}[a \in B] \geq 3/4$$

More generally

Theorem 16. *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.*

4. An Extractor from Nisan-Wigderson

This is a simplified presentation of results in [**Tre99**] (see also [**RRV99, ISW00**]).

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

For a string $x \in \{0,1\}^{\bar{N}}$, let $< x >: \{0,1\}^{\log \bar{N}} \to \{0,1\}$ be the function whose truth table is x. Let $l = \log \bar{N}$, and let $\mathbf{S} = (S_1, \ldots, S_m)$ be a $(l, \log m)$ design over $[t]$. Then consider the procedure $ExtNW : \{0,1\}^N \times \{0,1\}^t \to \{0,1\}^m$ defined as

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

That is, $ExtNW$ first encodes its first input using an error-correcting code, then views it as a function, and finally applies the Nisan-Wigderson construction to such a function, using the second input as a seed.

Lemma 17. *For sufficiently large m and for $\varepsilon > 2^{-m^2}$, $ExtNW_{\mathbf{C},\mathbf{S}}$ is a $(m^3, 2\varepsilon)$-extractor.*

Proof. Fix a random variable X of min-entropy m^3 and a function $D : \{0,1\}^m \to \{0,1\}$; we will argue that

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

Let us call a value a bad if it happens that

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

and let us call B the set of bad a. When a is bad, it follows that there is a circuit C of size $O(m^2)$ such that either $D(C())$ or its complement agrees with a on a fraction $1/2 + \varepsilon/m$ of its entries. Therefore, a is totally specified by D, C, and $2\log(m/\varepsilon)$ additional bits (once we have D and C, we know that the encoding of a sits in a given sphere of radius $1/2 - \varepsilon/m$, together with at most other $(m/\varepsilon)^2$ codewords). Therefore, for a fixed D, the size of B is upper bounded by the number of circuits of size $O(m^2)$, that is $2^{O(m^2 \log m)}$, times $(m/\varepsilon)^2$, times 2. The total is $2^{O(m^2 \log m)}$.

The probability that an element a taken from X belongs to B is therefore at most $2^{-m^3} \cdot 2^{O(m^2 \log m)} < \varepsilon$ for sufficiently large m. We then have

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

□

Theorem 18. *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 $t = O(\log N)$.*

BIBLIOGRAPHY

[ACR97] A.E. Andreev, A.E.F. Clementi, and J.D.P. Rolim. Optimal bounds for the approximation of boolean functions and some applications. *Theoretical Computer Science*, 180:243–268, 1997.

[ACR98] A.E. Andreev, A.E.F. Clementi, and J.D.P. Rolim. A new general derandomization method. *Journal of the ACM*, 45(1):179–213, 1998.

[ACR99] A.E. Andreev, A.E.F. Clementi, and J.D.P. Rolim. Worst-case hardness suffices for derandomization: A new method for hardness vs randomness trade-offs. *Theoretical Computer Science*, 221:3–18, 1999.

[ACRT99] A.E. Andreev, A.E.F. Clementi, J.D.P. Rolim, and L. Trevisan. Weak random sources, hitting sets, and BPP simulations. *SIAM Journal on Computing*, 28(6):2103–2116, 1999. Preliminary version in *Proc of FOCS'97*.

[AK97] V. Arvind and J. Köbler. On resource-bounded measure and pseudorandomness. In *Proceedings of the 17th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 235–249. LNCS 1346, Springer-Verlag, 1997.

[BF99] H. Buhrman and L. Fortnow. One-sided versus two-sided error in probabilistic computation. In *STACS'99*, pages 100–109, 1999.

[BFNW93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.

[Gol99] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Springer-Verlag, 1999.

[GW99] O. Goldreich and A. Wigderson. Improved derandomization of BPP using a hitting set generator. In *RANDOM'99*, pages 131–137, 1999.

[Imp95] R. Impagliazzo. Hard-core distributions for somewhat hard problems. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 538–545, 1995.

[ISW99] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Near-optimal conversion of hardness into pseudo-randomness. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, 1999.

[ISW00] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Extractors and pseudorandom generators with optimal seed length. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 1–10, 2000.

[IW97] R. Impagliazzo and A. Wigderson. $P = BPP$ unless E has subexponential circuits. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 220–229, 1997.

[IW98] R. Impagliazzo and A. Wigderson. Randomness versus time: Derandomization under a uniform assumption. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, pages 734–743, 1998.

[KvM99] A. Klivans and D. van Milkebeek. Graph non-isomorphism has subexponential size proofs unless the polynomial hierarchy collapses. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, pages 659–667, 1999.

[MV99] P.B. Miltersen and N.V. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 71–80, 1999.

[Nis96] N. Nisan. Extracting randomness: How and why. In *Proceedings of the 11th IEEE Conference on Computational Complexity*, pages 44–58, 1996.

[NTS98] N. Nisan and A. Ta-Shma. Extrating randomness : A survey and new constructions. *Journal of Computer and System Sciences*, 1998. To appear. Preliminary versions in [**Nis96, TS96**].

[NW94] N. Nisan and A. Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994. Preliminary version in *Proc. of FOCS'88.*

[NZ96] N. Nisan and D. Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996. Preliminary version in *Proc. of STOC'93.*

[Pap94] C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[RRV99] R. Raz, O. Reingold, and S. Vadhan. Extracting all the randomness and reducing the error in Trevisan's extractors. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, pages 149–158, 1999.

[RSW00] O. Reingold, R. Shaltiel, and A. Wigderson. Extracting randomness by repeated condensing. In *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science*, 2000.

[STV99] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, pages 537–546, 1999.

[Tre99] L. Trevisan. Construction of extractors using pseudo-random generators. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, pages 141–148, 1999.

[TS96] A. Ta-Shma. On extracting randomness from weak random sources. In *Proceedings of the 28th ACM Symposium on Theory of Computing*, pages 276–285, 1996.

[Yao82] A.C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23th IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.

[Zuc96] D. Zuckerman. Simulating BPP using a general weak random source. *Algorithmica*, 16(4/5):367–391, 1996.