Derandomizing BPP and Pseudorandomness - A Survey

Lecturer: Avi Wigderson [*] Scribe: Irit Dinur [†]

June 20, 2002

1 Introduction

"Do natural hard problems really exist?"
"Can randomness really help computation?"

These are two great questions in computer science today.

In this survey we will reveal surprising connections between these two questions, surveying progress made in the past 20 years in this field, and leading up to the current state-of-the-art.

Question 1 - Are there natural hard problems? There are many natural problems for which no efficient (i.e. polynomial-time) algorithm is known. A few examples are (i) factoring integers into their prime factors, (ii) deciding whether a Boolean formula is satisfiable, (iii) computing the permanent of a given matrix. We classify problems such as these into *complexity classes*, according to the amount of resources (e.g. running time, randomness, memory space) they require (for a list of complexity classes and their definitions, see Appendix A). For example, the complexity class EXP is the class of all problems that are decidable in exponential time. We consider a problem to be 'hard' if it does not to belong to an 'easy' complexity class, for example if it is outside the class P (consisting of all problems decidable in polynomial time). In formal terms, the question of whether or not hard problems exist can be written as an inequality between complexity classes:

$$\begin{aligned} \mathsf{NP} &\overset{?}{\not\subseteq} \mathsf{P} \\ \mathsf{NP} &\overset{?}{\not\subseteq} \mathsf{DTIME}(2^{\epsilon n}) \\ \mathsf{EXP} &\overset{?}{\not\subseteq} \mathsf{P/poly} \end{aligned}$$

This question remains notoriously open, with no significant progress in sight.

[*]Institute for Advanced Study, Princeton, NJ. E-Mail: avi@ias.edu.
[†]Institute for Advanced Study, Princeton, NJ. E-Mail: iritd@ias.edu.

Question 2 - Does randomness help computation? Randomness is a resource that is widely used in algorithms both to simplify them and to speed up computation. Indeed, there are a number of natural problems for which the best-known probabilistic algorithm is significantly better than the best-known deterministic one.

For example, there is a polynomial-time probabilistic algorithm to test whether a given number is prime or not, but there is no such deterministic algorithm.

Another example is the symbolic determinant: Given a $n \times n$ matrix A whose entries are either integers or variables from $x_1, \ldots, x_n$, determine whether $\det(A) \equiv 0$. An extremely simple probabilistic algorithm solving this problem is to plug in *random* values for each variable; and then compute the determinant. The result would be non-zero with high probability unless $\det A \equiv 0$. There is no known sub-exponential-time deterministic algorithm for this problem.

It is trivial that any probabilistic polynomial-time algorithm can be simulated in exponential time by trying all possible coin toss combinations and computing the majority, hence $\mathsf{BPP} \subseteq \mathsf{EXP}$. The question is whether any non-trivial derandomization is possible, i.e.,

$$\begin{aligned} \mathsf{BPP} &\overset{?}{\not\subseteq} \mathsf{P} \\ \mathsf{BPP} &\overset{?}{\not\subseteq} \mathsf{SUBEXP} = \mathsf{DTIME}(2^{n^\epsilon}) \end{aligned}$$

This question, similarly to the previous one, has remained widely unanswered for quite some time.

However, as it turns out, there is a beautiful connection between questions 1 and 2, that prevents a simultaneous 'yes' answer to both. Formally, we know the following concrete relations,

Theorem 1.1 ([Yao82, IW97, IW98]) *The following relations are true,*

- *If there exist one-way-functions then* $\mathsf{BPP} \subseteq \mathsf{SUBEXP}$.
- *If* $\mathsf{EXP} \not\subset \mathsf{P/poly}$ *then* $\mathsf{BPP} \subseteq \mathsf{SUBEXP}$.
- *If* $\mathsf{EXP} \not\subseteq \mathsf{BPSUBEXP}$ *then*[1] $\mathsf{BPP} \subseteq \mathsf{Avg{-}P}$.

The link between these two seemingly unrelated questions is provided by a general derandomization scheme, discussed next.

2 General Derandomization

We recall that the class BPP is the class of all languages that are decidable in bounded probabilistic polynomial time. Formally, BPP consists of all languages L for which there is a polynomial-time algorithm A that receives in addition to its input[2] x, a random string r, and such that for

[1]The class Avg$-$P is, roughly speaking, the class of all languages for which there is a polynomial-time algorithm solving with high probability on all polynomially sampleable distributions.

[2]Throughout this paper we think of the input length, n, as a parameter that is asymptotically tending to infinity. Wlog we also take the length of the random string to be n as well. We think of a function over $\{0,1\}^n$ as belonging to an *ensemble* of functions with increasing input lengths.

every input it gives the correct output with high probability, namely,

$$\begin{array}{llll} \textit{If} & x \in L & \textit{then} & \Pr_{r \in_R \{0,1\}^n}[A(x,r) = \textit{yes}] \geq 2/3\,. \\ \textit{If} & x \notin L & \textit{then} & \Pr_{r \in_R \{0,1\}^n}[A(x,r) = \textit{yes}] \leq 1/3\,. \end{array}$$

A trivial derandomization can be obtained by running A on all possible random strings and deciding according to majority. The running time of this deterministic procedure is exponential, hence it implies $\mathsf{BPP} \subseteq \mathsf{EXP}$. A non-trivial derandomization of BPP is obtained by running A on some smaller carefully selected subset of the random strings so that the majority result still reflects the true majority.

Let us denote the uniform distribution on $\{0,1\}^n$ by $\mathcal{U}_n$. Instead of supplying the algorithm A with a random string $r \in \mathcal{U}_n$, we can replace r by a pseudo-random string $G(s) \in \{0,1\}^n$ deterministically generated from a shorter random seed $s \in \mathcal{U}_m$. If, for any polynomial-time algorithm A, A's behavior remains virtually unchanged for all inputs, then the function G is called a pseudo-random generator:

Definition 2.1 (Pseudo Random Generator - PRG) *A function* $G : \{0,1\}^{m(n)} \to \{0,1\}^n$ *is a* pseudo-random generator *if*

$$\textit{For any circuit } C,\ |C| < n^2, \qquad |\Pr[C(\mathcal{U}_n) = 1] - \Pr[C(G(\mathcal{U}_m)) = 1]| < n^{-2} \tag{1}$$

The distribution $\mathcal{D}_n = G(\mathcal{U}_m)$ *is said to be* pseudo-random[3].

The choice of n^2 and n^{-2} in the above definition is arbitrary and can be made any polynomial via padding.

The distribution $G(\mathcal{U}_m)$ is 'computationally close to uniform'. It is interesting to contrast this with the information theoretic definition of being statistically close to uniform, defined in terms of the statistical distance between two distributions,

Definition 2.2 (Statistical Distance) *The statistical distance of two distributions* $\mathcal{D}_n, \mathcal{D}'_n$ *is*

$$\max_C \left|\Pr[C(\mathcal{D}_n) = 1] - \Pr[C(\mathcal{D}'_n) = 1]\right|$$

where the maximum is taken over all *possible circuits* C.

Thus, a distribution $\mathcal{D}_n$ is statistically ϵ-close to uniform if

$$\text{For any circuit } C, \qquad |\Pr[C(\mathcal{U}_n) = 1] - \Pr[C(\mathcal{D}_n) = 1]| \leq \epsilon \tag{2}$$

Note the similarity between (1) and (2), the only difference being the limit on the size of the 'observer' circuit in the computational case. Thus, pseudo-randomness may be viewed as a new "computational" variation of the statistical distance, where the distance between distributions is measured not by any observer (i.e. any circuit), but rather by observers with limited computational power.

Here is how a pseudo-random generator can be used to derandomize probabilistic algorithms,

[3]The definitions employed herein are non-uniform. One can replace every mention of a size-bounded circuit by a time-bounded probabilistic Turing machine to obtain the analogous uniform definition.

Theorem 2.3 (Basic Derandomization) *Let $G : \{0,1\}^{m(n)} \to \{0,1\}^n$ be a pseudo-random generator. Then,*

$$\mathsf{BPP} \subseteq \mathsf{DTIME}(time(G) \cdot 2^m)\,.$$

Proof: Given a language $L \in \mathsf{BPP}$, an input $x \overset{?}{\in} L$, and a probabilistic algorithm $A(x,r)$ deciding L, run this algorithm on x using $r = G(s)$ as the random string, for all possible seeds $s \in \{0,1\}^m$ and take the majority result. If there exists some input x on which the majority of $A(x, G(s))$ differs from the majority of $A(x,r)$, then A with input x (which can be converted into a small circuit) can serve as a distinguisher between $G(\mathcal{U}_m)$ and $\mathcal{U}_n$ contradicting G being a PRG. ∎

The central question in derandomization is: "What are the minimal values of m and $time(G)$ for which a pseudo-random generator G exists?".

In what follows we present a historical survey of derandomization results up until the current state of the art. In Section 3 we present the first pseudo-random generators, originating from cryptography. In Section 4 we present the Nisan-Wigderson generator and the hardness vs. randomness paradigm. In Section 5 we present newer views of the NW generator. We conclude in Section 6 with a brief comparison between computational and information-theoretic views of randomness.

3 Sequential Pseudo-random generators

Predating the beginning of a rigorous theory of pseudo-randomness, certain computational applications required random bits, and for this purpose sequential pseudo-random-generators were used. A sequential generator takes a function $g : \{0,1\}^m \to \{0,1\}^m$, and produces a sequence of m-bit numbers $X_1, \ldots, X_n$ (computed from a seed $X_0 \in \{0,1\}^m$ by iterating g) where $X_{i+1} = g(X_i)$. Various functions g have been suggested.

The Linear Congruential Generator. This generator was the most widely used generator, in real-life applications. It is defined by selecting an m-bit prime p, values $a, b \in \mathbb{Z}_p$, and setting $g_{\text{LC}}(X) \overset{def}{=} aX + b \ (mod\, p)$. Eventually this generator was "broken" by good predictor algorithms, that give correct predictions even when a, b and p are unknown.

The von-Neumann Generator. The function g of this generator is defined by squaring an m-bit integer X and then taking the middle m bits of the $2m$ bits of the outcome,

$$g_{\text{VN}}(X) \overset{def}{=} \text{ bits } \tfrac{3m}{2} \ldots \tfrac{m}{2} \text{ of } X^2$$

Von Neumann conjectured that this generator, due to its non-linearity, would be hard to break. This is not backed by any rigorous analysis (it hasn't been disproved either).

Shamir's 'Reverse' Discrete Logarithm Generator. Shamir [Sha83] suggested the use of the discrete logarithm function as g. Given a prime p and a generator a of $\mathbb{Z}_p^*$, the discrete logarithm $DL_{p,a} : \{0,1\}^m \rightarrow \{0,1\}^m$ is defined by

$$\forall X \in \mathbb{Z}_p^*, \qquad DL_{p,a}(X) \overset{def}{=} Y \qquad s.t. \quad a^Y = X \ (mod\, p)$$

There is no known polynomial-time algorithm for DL, but the generator G can be computed in reverse, since $DL_{p,a}^{-1}(Y) \equiv a^Y \ mod\, p$ is quite easily computable. Indeed, the DL function is an example of a one-way-function – a function which is easy to compute but hard to invert – which is a fundamental cryptographic primitive. Shamir's suggestion was to exploit this property of DL and to compute the pseudo-random sequence in reverse order, selecting $X_{n+1} \in_R \mathbb{Z}_p$ uniformly at random and setting $X_n = g^{-1}(X_{n+1})$ and inductively $X_i = g^{-1}(X_{i+1})$. Since the DL function is a permutation, the distribution over sequences obtained in this manner is identical to that in which we'd selected $X_0 \in_R \mathbb{Z}_p$ at random and computed the sequence in the forward direction. Shamir also suggested a criterion for evaluating a (sequential generator based on a) given function g: A sequence is called *unpredictable* if when reading the output sequence from left to right, given any initial i values, it is hard to compute the $i+1$st value.

Definition 3.1 (Unpredictable Sequence of Numbers) *A sequence of m-bit numbers $(X_1, \ldots, X_n)$ is called* (s, ϵ)-unpredictable *if it is drawn from a distribution $\mathcal{D}_n$ such that for every circuit C, $|C| < s$,*

$$\forall i < n, \qquad \Pr_{(X_1,..,X_n) \in \mathcal{D}_n} [C(X_1, \ldots, X_i) = X_{i+1}] < \epsilon \ .$$

We say that a sequence is unpredictable if it is (n^c, n^{-c})-unpredictable for any constant $c > 0$.

Theorem 3.2 ([Sha83]) *If there is no polynomial-time circuit for computing Discrete-Log, then $\mathcal{D}_n = \{ (X_1, \ldots, X_n) \mid \forall i, \ X_{i+1} = DL(X_i)\}$ is unpredictable.*

Proof Sketch: The proof proceeds by contradiction. Suppose $\mathcal{D}_n$ was predictable, i.e. there is some i and some polynomial-size circuit C such that $\Pr_{X_0 \in \{0,1\}^m} [C(X_1, \ldots, X_i) = X_{i+1}] \geq \epsilon$. We can compute the discrete logarithm for a random $X \in \mathbb{Z}_p$ by setting $X_i = X$ and for all $j < i$ $X_{j-1} = a^{X_j}$ $(mod\ p)$ and feeding this sequence to C. Since $\mathcal{D}_n$ is predictable, $C(X_1, \ldots, X_i) = X_{i+1} = DL(X_i)$ with probability $\geq \epsilon$, which means we are able to compute the discrete logarithm on ϵ fraction of the inputs X.

In order to reach a contradiction we need to present a polynomial-size circuit computing the discrete logarithm on *every* input. For this, we observe that the discrete logarithm has the additional special property of being *random-self-reducible*: an oracle for random values of the function can be used to compute the function on an arbitrary input. Indeed, given any algorithm A (e.g. the above) that computes the discrete logarithm on some ϵ fraction of the inputs, we compute $DL(Z)$ for an arbitrary value of $Z \in \mathbb{Z}_p^*$, by selecting a random $r \in \mathbb{Z}_p$ and outputting $A(a^r \cdot Z) - r \ mod\ p$. Since $a^r \cdot Z \ mod\ p$ is distributed uniformly in $\mathbb{Z}_p^*$, this process will output the value $DL(Z) \ mod\ p$ with probability $\geq \epsilon$. It is easy to check whether the outcome is correct or not, so repeating this process $O(1/\epsilon)$ times will find the correct value for $DL(Z)$ with probability $> 2/3$.

■

Note, of course, that the unpredictability of Shamir's sequence is entirely dependent on the order of the numbers. The sequence is completely predictable from right to left.

The Blum-Micali-Yao Generator. The unpredictable sequence of Shamir's generator, is not pseudo-random: the first i numbers leak some information about the $i+1$-st number. For example, if X_i is a quadratic residue (and this can be decided in polynomial time), we know that X_{i+1} is even.

Our next step would be to find an unpredictable sequence of *bits* rather than of m-bit numbers.

Definition 3.3 (Unpredictable Sequence of Bits) *A sequence of n bits $(x_1, \ldots, x_n)$ is called (s, ϵ)-*unpredictable *if it is drawn from a distribution $\mathcal{D}_n$ such that for every circuit C, $|C| < s$,*

$$\forall i < n, \qquad \Pr_{x \in \mathcal{D}_n} [C(x_1, \ldots, x_i) = x_{i+1}] < \frac{1}{2} + \epsilon$$

We also call the distribution $\mathcal{D}_n$ unpredictable.

Blum and Micali [BM84] proved that the most significant bit (msb) of the discrete logarithm function (i.e. the bit indicating whether the value of the discrete logarithm is greater than $p/2$) is as hard to compute as is the whole function. Formally, they proved the following reduction,

Lemma 3.4 ([BM84]) *Given access to a circuit C for which*

$$\Pr_{X \in \mathbb{Z}_p^*} [C(X) = msb(DL(X))] > \frac{1}{2} + \epsilon$$

one can compute the discrete logarithm with high probability in time $poly(|X|, \frac{1}{\epsilon})$.

A Boolean predicate such as the *msb* predicate is called a *hard-core* predicate of a given function because it captures all of the hardness of the function.

From this fundamental lemma it follows that,

Theorem 3.5 ([BM84]) *Let $X_0 \in \mathbb{Z}_p$ and define*

$$\forall i = 1, \ldots, n \qquad X_i \overset{def}{=} DL_{p,a}(X_{i-1}) \text{ and } b_i \overset{def}{=} msb(X_i)\,.$$

The sequence $(b_1, b_2, \ldots, b_n)$ is unpredictable.

Proof Sketch: The proof follows by reduction to the previous theorem (Theorem 3.2). Suppose C is a predictor algorithm for the bits $b_1, \ldots, b_n$, predicting b_{i+1} from $b_1, \ldots, b_i$ for some i. We apply Lemma 3.4 and use C to predict X_{i+1} from $X_1, \ldots, X_i$, in contradiction to the unpredictability of $X_1, \ldots, X_n$. ■

Interestingly, in contrast to the number sequence $X_1, \ldots, X_n$, the msb bit-sequence $b_1, \ldots, b_n$ cannot be predicted even in the reverse order. Moreover, Yao [Yao82] proved that *every* unpredictable bit-sequence is in fact pseudo-random (see Definition 2.1),

Theorem 3.6 ([Yao82]) *Every unpredictable bit sequence is pseudo-random.*

Proof Sketch: Let $\mathcal{D}_n$ be an unpredictable distribution. Suppose there is a circuit C that distinguishes between a random $\bar{u} = (u_1, \ldots, u_n) \in \mathcal{U}_n$ and $\bar{b} = (b_1, \ldots, b_n) \in \mathcal{D}_n$ with probability $\epsilon > 0$. Define the hybrid distributions $\mathcal{H}_i = \left\{ (b_1, \ldots, b_i, u_{i+1}, \ldots, u_n) \,|\, \bar{b} \in_R \mathcal{D}_n, \bar{u} \in_R \mathcal{U}_n \right\}$ for $i = 0, \ldots, n$ (noting that $\mathcal{H}_0 = \mathcal{U}_n$ and $\mathcal{H}_n = \mathcal{D}_n$), and denote

$$P_i \stackrel{def}{=} \left| \Pr_{x \in \mathcal{H}_{i+1}} [C(x) = 1] - \Pr_{x \in \mathcal{H}_i} [C(x) = 1] \right| .$$

Since $\sum_{i=0}^{n-1} P_i > \epsilon$, there must be some $i \in [n]$ for which $P_i > \epsilon/n$, i.e. C distinguishes $\mathcal{H}_i$ from $\mathcal{H}_{i+1}$ with an advantage of $\geq \epsilon/n$. From this we can construct (uniformly) a predictor for b_{i+1} from the bits $b_1, .., b_i$, whose success probability is $\frac{1}{2} + \epsilon/n$. ■

A corollary of this (plugging in Theorem 2.3) is

Corollary 3.7 *The following are true:*

- *If Discrete-Log* $\notin$ P/poly, *then* BPP $\subseteq$ SUBEXP.
- *If Discrete-Log* $\notin$ BPP, *then*[4] BPP $\subseteq$ AvgSUBEXP.
- *If Discrete-Log* $\notin$ *SUBEXP/Poly, then* BPP $\subseteq \tilde{\mathsf{P}} \stackrel{def}{=} \mathsf{DTIME}(2^{(\log n)^{O(1)}})$.

Yao also showed that a pseudo-random generator can be constructed from any one-way permutation (i.e. a one-to-one function that is easy to compute but hard to invert). This was later extended,

Theorem 3.8 ([HILL98]) *One-way functions exist if and only if pseudo-random generators exist.*

An important theorem that was used for proving this result, is the Goldreich-Levin 'hard-core-bit' theorem, which states that a non-negligible advantage in the prediction of xors of subsets of bits in the output of a one-way function, can be converted into an algorithm for inverting the whole function,

Theorem 3.9 ([GL89]) *Let* $f : \{0,1\}^n \to \{0,1\}^n$, *and define* $b : \{0,1\}^{2n} \to \{0,1\}$ *by*

$$b(x,r) \stackrel{def}{=} \langle r, f(x) \rangle \stackrel{def}{=} \sum_{i=1}^{n} r_i \cdot f(x)_i \;\; mod\, 2$$

Any polynomial-time algorithm for predicting b *with success probability* $\frac{1}{2} + \epsilon$ *can be converted to a polynomial-time algorithm that inverts* f *with success probability* ϵ.

This theorem has several interpretations, and is useful in other contexts. For example, it can be viewed as a local list decoding of the Hadamard code. This is the first such result, in an area that is currently very active. It can also be viewed as an algorithm which learns all the high Fourier coefficients of a Boolean function (given oracle access to the function), an algorithm that is useful in various computational learning algorithms.

[4]The class AvgSUBEXP is, roughly speaking, the class of all languages for which there is a sub-exponential time algorithm solving with high probability on all polynomially sampleable distributions.

Given these first derandomization results, it was natural to seek unconditional derandomization results for complexity classes where lower-bounds do exist unconditionally. Two such models are small-space computations and constant-depth circuits.

Umesh Vazirani observed that the $2n+1$ bit-sequence $x_1, \ldots, x_n, y_1, \ldots, y_n, (x \cdot y \ mod\ 2)$ is unpredictable for any (read-once) automaton whose space is bounded by $o(n)$. This is shown by relying on the lower bound of $n+1$ for the number of communication bits required to compute $x \cdot y \ mod\ 2$. The state of the machine after reading $x_1, \ldots, x_n$ can be viewed as the communication sent from player x to player y when trying to compute $x \cdot y \ mod\ 2$, which is known to require $n+1$ communication bits. Following that, Nisan [Nis92] used recursive hashing to construct a generator that stretches $(\log n)^2$ bits to n bits, that appear random to any logspace machine.

Impagliazzo, Nisan and Wigderson [INW94] extended Nisan's communication complexity lower-bounds to derandomize the computation of any network of processors whose inter-communication is bounded.

Another model for which we have unconditional lower-bounds is bounded-depth circuits. Ajtai and Wigderson [AW89] showed the existence of a bit-generator for AC_0 (the class of bounded-depth circuits with unbounded fan-in AND and OR gates), stretching n^ϵ bits to n bits. Nisan [Nis91] constructed a pseudo-random generator that stretches $(\log n)^c$ bits into n bits that appear random to any constant depth circuit. This paper served as the seed for the next development.

4 Hardness vs. Randomness

In the previous section we saw the first seeds of the general paradigm of trading hardness for randomness. A significant expansion of this paradigm, leaving behind the world of cryptographic functions, was introduced by Nisan and Wigderson with the NW-generator.

4.1 The NW Generator

The Nisan Wigderson pseudo-random generator [NW94] is a completely different approach to derandomization, and is essentially the basis of almost all new derandomization results. The NW construction is non-sequential, and can be used with a whole range of hardness assumptions whose extreme will yield the ultimate derandomization, $\mathsf{P} = \mathsf{BPP}$.

In the NW setting, the function whose hardness is exploited for derandomization can be *any* function in EXP, rather than being an inverse of a function in P. Breaking loose from the cryptographic setting, the generator is allowed to run in time super-polynomial (even exponential) in the seed length, which is still ok for derandomization purposes.

The generator is constructed as follows. Let $f : \{0,1\}^m \to \{0,1\}$ be a function that we assume is unpredictable, i.e.,

Definition 4.1 (Unpredictable function) *A function $f : \{0,1\}^m \to \{0,1\}$ is* unpredictable *if for any polynomial-size circuit C, and for any $c > 0$,*

$$\left| \Pr_{x \in \mathcal{U}_m} [C(x) = f(x)] - \frac{1}{2} \right| < n^{-c}.$$

Let $l > m$ and let $S_1, \dots, S_n \subset [l]$, $|S_i| = m$. For a string $x \in \{0,1\}^l$ denote by x_{S_i} the projection of x on the coordinates of S_i, and define $NW_f : \{0,1\}^l \to \{0,1\}^n$,

$$\forall x \in \{0,1\}^l, \qquad NW_f(x) \overset{def}{=} f(x_{S_1})f(x_{S_2})\dots f(x_{S_n})$$

This function is a pseudo-random generator if the subsets S_i are 'almost disjoint', or more formally, a combinatorial *design*,

Lemma 4.2 (Design [NW94]) *For every integers $n, m > 0$ such that $\log n \le m \le n$, there exists $l = O(m^2)$ and subsets $S_1, \dots, S_n \subset [l]$, $|S_i| = m$, such that for all $i \neq j$, $|S_i \cap S_j| \le \log n$. Moreover, the subsets S_i can be found deterministically by a Turing machine running in space $O(\log n)$.*

Assuming this lemma, NW_f is a pseudo-random generator,

Theorem 4.3 ([NW94]) *Let $f \in \mathsf{EXP}$ be unpredictable, then NW_f is a pseudo-random generator.*

Proof Sketch: Let us assume that NW_f is not a pseudo-random generator, and deduce that f is predictable. Define the hybrid distributions $\mathcal{H}_i$ consisting of the first i bits of $NW_f(\mathcal{U}_m)$ and the remaining $n-i$ bits of $\mathcal{U}_n$. If $\mathcal{H}_n = NW_f(\mathcal{U}_m)$ is not pseudo-random then there is an algorithm that distinguishes $\mathcal{H}_i$ from $\mathcal{H}_{i+1}$ for some i, which can be transformed into a predictor, predicting the $i+1$st bit of $NW_f(\mathcal{U}_m)$ from the first i bits with some non-negligible advantage. To contradict the unpredictability of f we transform this into a predictor predicting the ith bit of $NW_f(x)$ – denoted y_i – from the bits $x_1, \dots, x_l$. Since $y_i = f(x_{S_i})$ actually depends only on the m bits in S_i, let us fix the rest of the bits in x so that the predictor does as well as in the average case. Whenever the predictor relies on some value of y_j with $j < i$, we can plug into our circuit a truth-table computation of y_j. Since y_j depends on only $|S_i \cap S_j| \le \log m$ bits of x_{S_i}, it can be encoded by a circuit of size m. The size of the new predictor-circuit thus increases by at most $m \cdot n$, and we have reached a contradiction.

Note that even if the predictor of y_i from $y_1, \dots, y_{i-1}$ is uniform, the predictor from $x_1, \dots, x_l$ is not so, as it relies on oracle calls to the hard function f. ∎

4.2 Hardness Amplification

The hard function f taken by the NW construction above was assumed to be unpredictable, i.e. such that no circuit can predict it much better than a random coin toss. Naturally, the weaker the assumptions we make, the stronger (and more believable) the derandomization results. Hardness amplification techniques allow us to construct extremely hard functions from mildly hard ones, therefore allowing derandomization under weaker hardness assumptions.

We assume existence of functions that are hard in the worst case and from them construct functions that are unpredictable, which can then be transformed via the NW generator into a pseudo-random generator.

Definition 4.4 *Let $f : \{0,1\}^n \to \{0,1\}$. Let $s = poly(n)$ and let $\epsilon = 1/poly(n)$. f is said to be*

- Hard in the worst case: $\forall$ *circuit* $C, |C| < s, \quad \Pr_x[C(x) = f(x)] < 1$.

- Mildly hard: $\forall$ *circuit* $C, |C| < s, \quad \Pr_x[C(x) = f(x)] < 1 - \epsilon.$
- Unpredictable: $\forall$ *circuit* $C, |C| < s, \quad \Pr_x[C(x) = f(x)] < \frac{1}{2} + \epsilon.$

The first hardness amplification was Yao's XOR-lemma [Yao82] taking a function that is mildly hard and converting it into an unpredictable function, i.e. one that can be efficiently computed on no more than slightly above half of the inputs.

Theorem 4.5 (Yao's XOR-Lemma [Yao82]) *If f is mildly hard, then there is a function $f' \in \mathsf{P}^f$ that is unpredictable.*

The function f' is taken to be the XOR of polynomially many copies of f on different inputs. Therefore, f' is easy to compute whenever f is. For several proofs of this theorem see [GNW95, IW97].

A second hardness amplification step is to transform a worst-case hard function into a mildly hard function.

Theorem 4.6 ([Lip91, BF90, BFNW91]) *If $f \in \mathsf{EXP}$ is hard in the worst case, then there is a function $f_e \in \mathsf{EXP}^f$ that is mildly hard.*

Proof Sketch: Let $f : \{0,1\}^n \to \{0,1\}$ and let $f_e : GF(q)^n \to GF(q)$ be the unique multi-linear polynomial over $GF(q)$ such that for every $x \in \{0,1\}^n$, $f(x) = f_e(x)$, i.e.,

$$\forall (x_1, \ldots, x_n) \in GF(q)^n, \quad f_e(x_1, \ldots, x_n) \stackrel{def}{=} \sum_{(a_1,\ldots,a_n)\in\{0,1\}^n} f(a_1, \ldots, a_n) \cdot \prod_{i=1}^{n} (x_i - (1 - a_i))$$

As seen by the above formula, f_e can be computed from f by simple interpolation, hence $f_e \in \mathsf{EXP}^f$. (This computation can actually be done in PSPACE. Interestingly however, even when $f \in \mathsf{P}$, the extension function f_e is not known to be computable in any class below PSPACE. Luckily, the NW generator requires only that f_e be in EXP.)

A circuit that computes f_e on $1 - \epsilon$ of its domain, can be used to compute f on *every* $x \in \{0,1\}^n$ by choosing a random line passing through x, and interpolating $f(x)$ from values of f_e on random points z on that line. The majority of the lines passing through x will give the correct value for $f(x)$.

Note that the function f_e is non-Boolean, so in order to turn it into our mildly hard function, we need to concatenate it with an appropriate binary error-correcting code. ■

Combination of the two hardness amplification steps yields,

Theorem 4.7 ([BFNW91, NW94]) *If* $\mathsf{EXP} \not\subseteq \mathsf{P/poly}$ *then* $\mathsf{BPP} \subset \mathsf{SUBEXP}$.

Theorem 4.8 ([IW97]) *If* $\mathsf{E} \not\subseteq size(2^{o(n)})$ *then* $\mathsf{BPP} = \mathsf{P}$.

The last result needed to avoid the input size blow-up of the Yao XOR-Lemma, and invented a 'derandomized' version of it.

A few years later, Sudan, Trevisan and Vadhan [STV01], used list-decoding to show that the XOR-lemma is not needed for hardness amplification. They proved that the function f_e is not only mildly hard, but in fact unpredictable. They showed that a list-decoding algorithm for Reed-Muller codes (of which a multi-linear polynomial is a special case) can be used to transform an algorithm that predicts f_e on $\frac{1}{2}+\epsilon$ of its inputs, to one that predicts f on an arbitrary value.

Theorem 4.9 ([STV01]) *If $f \in \mathsf{EXP}$ is hard in the worst case, then there is a function $f_e \in \mathsf{EXP}^f$ that is unpredictable.*

5 New Views of the NW Generator

The proof of the NW generator is actually a "black-box" reduction, taking a distinguisher D that "breaks" the pseudo-random generator NW_f, and creates from it a predictor algorithm $A^{f,D}$ for f. Hence, a hardness assumption that prevents $A^{f,D}$ from belonging to a certain class, implies the corresponding derandomization result, which applies to a wider range of complexity classes. This observation was made by Klivans and Melkebeek [KvM99], who then used it to derandomize AM, placing for example graph non-isomorphism in NP, under a certain hardness assumption:

Theorem 5.1 ([KvM99]) *Denote by $C^{SAT}(f)$ the circuit complexity of f, relative to a SAT oracle. If there exists $f \in NE \cap coNE$ with $C^{SAT}(f) \in 2^{\Omega(n)}$, then $\mathsf{AM} = \mathsf{NP}$.*

Trevisan's Extractor. Another direction was taken by Trevisan [Tre99]. He considered the NW reduction as an algorithm, taking as input both the seed *and* a truth-table of a function f. Rather than assume that f is hard, Trevisan considered a random f and showed that if the distribution from which f is drawn has sufficient min-entropy, then the distribution on the outputs of the NW-generator is *statistically close* to uniform. This is a much stronger requirement than just being pseudo-random. Such an object, transforming a "weakly random" distribution to one that is statistically close (recall Definition 2.2) to uniform is called an extractor, (see, e.g. [Nis96]). The proof follows the NW-reduction: Assume any (in this case, random) function f, and a distinguisher D that distinguishes between the uniform distribution and the output distribution $NW_f(x)$. The NW reduction constructs a polynomial-sized circuit A_f^D (that is allowed to use D-gates) predicting f with non-negligible advantage. The (small!) number of possible such circuits translates (by taking the logarithm) to an immediate bound on the min-entropy of the distribution from which f is drawn.

The Shaltiel-Umans PRG. A completely different construction of pseudo-random generators was suggested by Shaltiel and Umans [SU01, Uma02] who take an extractor construction [TZS01, SU01] and transform it into a PRG.

In all known derandomization results, one takes a 'hard' function f, and uses the seed to determine some t inputs $x_1, .., x_t$ on which f will be evaluated, and then the pseudo-random sequence is $f(x_1), \ldots, f(x_t)$. Here, the seed is viewed as a point $x \in F^d$, where F is a finite field with generator g, and the 'hard' function f is evaluated on $x, g \cdot x, g^2 \cdot x, \ldots$. The pseudo-randomness proof relies on the fact that $x \to gx$ can be represented (if F^d is viewed as a vector space) as a linear transformation.

Uniform derandomization. Beginning with the NW generator, derandomization results relied on hardness assumptions that, while considerably weaker than the previous cryptographic assumptions regarding the existence of a one-way function, are all non-uniform. Impagliazzo and Wigderson [IW98] extended these results to work for a uniform assumption, i.e. derandomization based on the assumption that $\mathsf{EXP} \neq \mathsf{BPP}$ rather than $\mathsf{EXP} \not\subseteq \mathsf{P/poly}$.

Intuitively, derandomization requires a non-uniform assumption because if there was a function that is uniformly hard but non-uniformly easy, then a rare BPP instance may encode the information that makes this function easy, and this instance could not be 'derandomized' by this function. However, we can still hope to achieve a derandomization for which it is hard to find such 'bad' instances. Roughly speaking, we define the class $\mathsf{AvgSUBEXP}$ to be the class of all languages for which there is a sub-exponential time algorithm solving with high probability on all polynomially sampleable distributions. Now, the following "gap-theorem" regarding BPP is known:

Theorem 5.2 ([IW98]) *If* $\mathsf{BPP} \neq \mathsf{EXP}$ *then* $\mathsf{BPP} \subset \mathsf{AvgSUBEXP}$*.*

Proof Sketch: Previous derandomization proofs were inherently non-uniform in that they take a distinguisher allegedly "breaking" the PRG and construct from it a circuit for the hard function f. The process relies on values of f at certain (sometimes random) points.

Suppose we already had a circuit $C_{f,n-1}$ for computing f on $n-1$ sized inputs. If f were downward-self-reducible, then oracle calls for f on inputs of length n can be answered relying on $C_{f,n-1}$. Thus, the NW reduction can be applied to produce a circuit $C_{f,n}$ (note that this reduction queries f only on inputs of length n) constructed by the help of the smaller $(n-1)$ circuit. It is crucial that the circuit $C_{f,n}$ does not depend on $C_{f,n-1}$ at all. Thus, recursively, we can construct a circuit for f *from scratch.*

For the proof to work we must use the fact that the NW generator only fails if $\mathsf{EXP} \subseteq \mathsf{P/poly}$ which implies [KL82] that $\mathsf{EXP} = \mathsf{P}^{\#\mathsf{P}}$, in which case EXP has downward-self-reducible functions that are complete. ■

Corollary 5.3 ([IW98]) $\mathsf{EXP} \cap \mathsf{P/poly} = \mathsf{BPP}$ *if and only if* $\mathsf{EXP} = \mathsf{BPP}$*.*

Is there hope for derandomization without proving lower-bounds? Known derandomizations of BPP rely on EXP containing 'hard' problems for circuits. Without any such assumptions we cannot even rule out $\mathsf{NEXP} = \mathsf{BPP}$. In fact, it is well-known that the NW pseudo-random generator derandomizes BPP if *and only if* EXP contains 'hard' functions. Since proving circuit lower-bounds (i.e. showing existence of hard functions in EXP) is very hard, it is interesting whether there might be a different approach to derandomization that would not require proving lower-bounds. Let us imagine that $\mathsf{P} = \mathsf{BPP}$, Impagliazzo, Kabanets, and Wigderson [IKW01] outlined three conceivable scenarios for proving this,

1. A non-constructive proof showing that for every BPP algorithm there *exists* a P algorithm accepting the same language.

2. A constructive proof describing a deterministic polynomial-time algorithm for estimating the acceptance probability of a given algorithm, relying on an encoding of a Boolean circuit computing the function.

3. A constructive proof describing a deterministic polynomial-time algorithm for estimating the acceptance probability of a given function, relying only on *oracle access* to the function.

Known derandomization results all fall into the (stronger) third scenario. As already mentioned above, such proofs are known to be equivalent to proving circuit lower bounds. Impagliazzo, Kabanets, and Wigderson [IKW01] proved that a derandomization result following the second scenario will also necessarily prove circuit lower bounds. This can be reformulated as proving that $\mathsf{promise\text{-}BPP} \subset \mathsf{SUBEXP}$. They proved that even a non-deterministic derandomization of MA (which will imply $\mathsf{promise\text{-}BPP} \subseteq \mathsf{SUBEXP}$) cannot be proven without simultaneously proving a circuit lower bound for some problem in NEXP:

Theorem 5.4 ([IKW01]) *If* $\mathsf{NEXP} \subset \mathsf{P/poly}$ *then* $\mathsf{NEXP} = \mathsf{MA}$.

Interestingly, this theorem assumes a non-uniform assumption, and deduces a uniform result. This can be conceptually compared to the unknown implication $\mathsf{NP} \subset \mathsf{P/poly} \overset{?}{\Longrightarrow} \mathsf{NP} = \mathsf{P}$.

6 Computational vs. Information Theoretic Randomness

We have discussed complexity theoretic aspects of randomness, and the fundamental relations between derandomization and proving complexity lower bounds. Let us conclude this survey by a short comparison between some computational 'views' of randomness and their statistical counterparts.

Computational	Information theoretic
Unpredictable function	Coin toss
Pseudorandom distribution	Statistically close to uniform
Worst-case hardness $\to$ Average case hardness	(Local) error-correcting codes
Pseudo-random generators	Extractors

The first two items in this table display the essential difference between information theoretic randomness and computational randomness is in the limitations we put on the observer. When the distinguisher is computationally bounded, an *unpredictable function* appears like a coin toss, and a *pseudo-random distribution* looks like the uniform one. Moreover, this is precisely the definition of these two objects!

The last two items relate some interesting connections or equivalences between these two outlooks on randomness. The third item says that if we look at a function as an error-correcting code, and at its inputs as indexing the 'bits' of that code, then an average case – worst case reduction for the function amounts to encoding it in a locally decodable error-correcting code, see [KT00] for more on this topic. The last item in the table is a place where the study of computational randomness has lead to some fruitful insights about extracting true randomness from a 'weak' random source, and vice versa. In one direction Trevisan [Tre99] showed that the NW generator, if interpreted appropriately, already extracts much of the randomness in a weakly random source. In the other direction, Umans and Shaltiel [SU01, Uma02] were able to transform a strong extractor construction back into a PRG construction.

A A List of Complexity Classes

Definition A.1 *The following definitions are standard,*

- $\mathsf{DTIME}(t(n))$ *is the class of all languages L that are decidable by a deterministic Turing machine with running time bounded by $t(n)$.*
- $\mathsf{NDTIME}(t(n))$ *is the class of all languages L that are decidable by a non-deterministic Turing machine with running time bounded by $t(n)$.*
- $\mathsf{BPTIME}(t(n))$ *is the class of all languages L that are decidable by a probabilistic Turing machine M with running time bounded by $t(n)$, such that if $x \in L$ then $\Pr[M \text{ accepts } x] > \frac{2}{3}$ and if $x \notin L$ then $\Pr[M \text{ accepts } x] < \frac{1}{3}$.*

We name complexity classes as follows,

- $\mathsf{P} = \mathsf{DTIME}(n^{O(1)})$
- $\mathsf{BPP} = \mathsf{BPTIME}(n^{O(1)})$
- $\mathsf{NP} = \mathsf{NDTIME}(n^{O(1)})$
- $\mathsf{SUBEXP} = \bigcap_{\epsilon>0} \mathsf{DTIME}(2^{n^\epsilon})$
- $\mathsf{E} = \mathsf{DTIME}(2^{O(n)})$
- $\mathsf{EXP} = \mathsf{DTIME}(2^{n^{O(1)}})$
- $\mathsf{NEXP} = \mathsf{NDTIME}(2^{n^{O(1)}})$

Definition A.2 ($\mathsf{P/poly}$) *The class $\mathsf{P/poly}$ consists of all languages L such that for every n, there exists an "advice string" $S(n)$ of length $|S(n)| \leq n^{O(1)}$ such that $L' = \{(x, S(|x|)) \mid x \in L\} \in \mathsf{P}$.*

Definition A.3 (MA, AM, **[BM88]**) *A language L is in the class MA if there exists a polynomial-time decidable predicate $R(x, y, z)$ and a constant $c > 0$ such that for every $x \in \{0,1\}^n$,*

$$\begin{aligned} x \in L \quad &\Rightarrow \quad \exists y \in \{0,1\}^{n^c} : \Pr_{z \in \{0,1\}^{n^c}}[R(x,y,z) = 1] \geq \frac{2}{3} \\ x \notin L \quad &\Rightarrow \quad \exists y \in \{0,1\}^{n^c} : \Pr_{z \in \{0,1\}^{n^c}}[R(x,y,z) = 1] \leq \frac{1}{3} \end{aligned}$$

A language L is in the class AM if there exists a polynomial-time decidable predicate $R(x, y, z)$ and a constant $c > 0$ such that for every $x \in \{0,1\}^n$,

$$\begin{aligned} x \in L \quad &\Rightarrow \quad \Pr_{z \in \{0,1\}^{n^c}}\left[\exists y \in \{0,1\}^{n^c} : R(x,y,z) = 1\right] \geq \frac{2}{3} \\ x \notin L \quad &\Rightarrow \quad \Pr_{z \in \{0,1\}^{n^c}}\left[\exists y \in \{0,1\}^{n^c} : R(x,y,z) = 1\right] \leq \frac{1}{3} \end{aligned}$$

References

[AW89] Miklós Ajtai and Avi Wigderson. Deterministic simulation of probabilistic constant depth circuits. *ADVCR: Advances in Computing Research*, 5, 1989.

[BF90] Donald Beaver and Joan Feigenbaum. Hiding instances in multioracle queries. In *7th Annual Symposium on Theoretical Aspects of Computer Science*, volume 415 of *lncs*, pages 37–48, Rouen, France, 22–24 February 1990. Springer.

[BFNW91] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. *BPP* has subexponential time simulations unless *exptime* has publishable proofs. In *Proceedings of the 6th Annual Conference on Structure in Complexity Theory (SCTC '91)*, pages 213–219, Chicago, IL, USA, June 1991. IEEE Computer Society Press.

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13:850–864, 1984.

[BM88] L. Babai and S. Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, pages 254–276, 1988.

[GL89] O. Goldreich and L.A. Levin. A hard-core predicate for all one-way functions. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 25–32, 1989.

[GNW95] Oded Goldreich, Noam Nisan, and Avi Wigderson. On yao's XOR-lemma. In *Electronic Colloquium on Computational Complexity, technical reports*, 1995.

[HILL98] Johan Hastad, Russel Impagliazzo, Leonid Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SICOMP: SIAM Journal on Computing*, 28, 1998.

[IKW01] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. In *Proceedings of the Sixteenth IEEE Conference on Computational Complexity*, pages 1–11, 2001.

[INW94] Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In ACM, editor, *Proceedings of the twenty-sixth annual ACM Symposium on the Theory of Computing: Montreal, Quebec, Canada, May 23–25, 1994*, pages 356–364, New York, NY, USA, 1994. ACM Press.

[IW97] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing (STOC '97)*, pages 220–229, New York, May 1997. Association for Computing Machinery.

[IW98] Russell Impagliazzo and Avi Wigderson. Randomness vs. time: de-randomization under a uniform assumption. In IEEE, editor, *39th Annual Symposium on Foundations of Computer Science: proceedings: November 8–11, 1998, Palo Alto, Califor-*

nia, pages 734–743, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1998. IEEE Computer Society Press.

[KL82] R. M. Karp and R. J. Lipton. Turing machines that take advice. In *LANDA: Logic and Algorithmic: An International Symposium Held in Honour of Ernst Specker*. L'Enseignement Mathematique, Universite de Geneve, 1982.

[KT00] Katz and Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2000.

[KvM99] Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing (STOC'99)*, pages 659–667, New York, May 1999. Association for Computing Machinery.

[Lip91] R. J. Lipton. New directions in testing. In *Distributed Computing and Cryptography*, volume 2 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 191–202. American Mathematics Society, 1991.

[Nis91] Noam Nisan. Pseudorandom bits for constant depth circuits. *COMBINAT: Combinatorica*, 11, 1991.

[Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4), 1992.

[Nis96] Noam Nisan. Extracting randomness: How and why: A survey. In *Proceedings of the 11th Annual IEEE Conference on Computational Complexity (CCC-96)*, pages 44–58, Los Alamitos, May 24–27 1996. IEEE Computer Society.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, October 1994.

[Sha83] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Transactions on Computer Systems*, 1(1):38–44, February 1983.

[STV01] Madhu Sudan, Luca Trevisan, and Sahlil Vadhan. Pseudorandom generators without the XOR lemma. *JCSS: Journal of Computer and System Sciences*, 62, 2001.

[SU01] Ronen Shaltiel and Christopher Umans. Simple extractors for all Min-Entropies and a new Pseudo-Random generator. In Bob Werner, editor, *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS-01)*, pages 648–657, Los Alamitos, CA, October 14–17 2001. IEEE Computer Society.

[Tre99] Luca Trevisan. Construction of extractors using pseudo-random generators. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing (STOC'99)*, pages 141–148, New York, May 1999. Association for Computing Machinery.

[TZS01] Amnon Ta-Shma, David Zuckerman, and Shmuel Safra. Extractors from Reed-Muller codes. In Bob Werner, editor, *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS-01)*, pages 638–647, Los Alamitos, CA, October 14–17 2001. IEEE Computer Society.

[Uma02] C. Umans. Pseudo-random generators for all hardnesses. In *Proc. 34th ACM Symp. on Theory of Computing*, 2002. To appear.

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd Symposium on Foundations of Computer Science (FOCS)*, pages 80–91. IEEE Computer Society Press, 1982.