Hardness vs. Randomness

Derandomization and the Impagliazzo–Wigderson theorem

Lecture 12

Outline

Do we need randomness?

Circuits

Pseudorandom generators

The Nisan–Wigderson generator

Hardness amplification

Putting it together

Outline

Do we need randomness?

Circuits

Pseudorandom generators

The Nisan–Wigderson generator

Hardness amplification

Putting it together

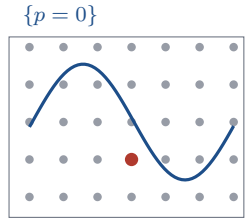
Where this course started

Polynomial identity testing. Given an arithmetic circuit for $p \in \mathbb{F}[x_1, \ldots, x_n]$ of degree d: is $p \equiv 0$?

Schwartz–Zippel

$$p \not\equiv 0 \implies \mathbb{P}_{a \sim S^n}\left[p(a) = 0\right] \leq \frac{d}{|S|}$$

Evaluate at one random point: two lines of code, one-sided error, confidence as high as we like.



the zero set is thin: a random point (•) misses it

And yet: no deterministic polynomial-time algorithm for PIT is known.

Randomness all over this course

- **L1** identity testing: a random evaluation point
- **L3** power method: a random starting vector
- **L5** st-connectivity: a random walk (and then: derandomized)
- **L6–L7** random matrices, random projections
- **L9** Max-Cut: a random hyperplane
- **L11** codes: a random (linear) code meets the Gilbert–Varshamov bound

Is randomness a genuine computational *resource* — or only a convenience?

The class BPP

Definition

$L \in$ BPP if there is a polynomial-time algorithm $A(x, r)$ with

$$x \in L \;\Rightarrow\; \mathop{\mathbb{P}}_{r}[A(x,r) = 1] \geq \tfrac{2}{3}, \qquad x \notin L \;\Rightarrow\; \mathop{\mathbb{P}}_{r}[A(x,r) = 1] \leq \tfrac{1}{3}.$$

What we know

- P $\subseteq$ BPP $\subseteq$ EXP
- primality: randomized 1977, deterministic 2002
- st-connectivity: Lecture 5

What we believe

- P $=$ BPP
- every randomized algorithm can be simulated deterministically with polynomial overhead

One-problem-at-a-time derandomization is hard work. Today: a *generic* theorem.

Today's theorem

Theorem (Impagliazzo–Wigderson, 1997)

Suppose there is a language L decidable in time $2^{O(n)}$ that requires Boolean circuits of size $2^{\delta n}$, for some $\delta > 0$. Then

$$\mathsf{P} = \mathsf{BPP}.$$

The shape of the statement

a *lower bound* we cannot prove

$\Longrightarrow$

an *algorithm* we would like to have

- one hard problem is enough — to derandomize *everything*
- hardness is converted into pseudorandomness
- the machinery: designs, codes, list decoding

Outline

Do we need randomness?

Circuits

Pseudorandom generators

The Nisan–Wigderson generator

Hardness amplification

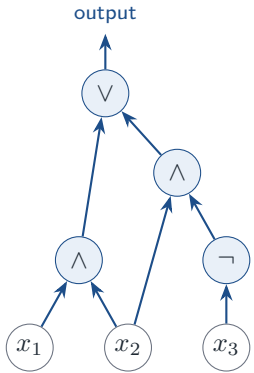
Putting it together

Boolean circuits

Definition

A Boolean circuit on n inputs is a DAG with

- n input nodes, labelled $x_1, \ldots, x_n$,
- internal nodes labelled $\wedge$, $\vee$, $\neg$,
- one output node.



- **size** $=$ number of nodes
- every circuit computes a unique $f : \{0,1\}^n \to \{0,1\}$
- every f is computed by *some* circuit — the question is how big

Circuit families and circuit complexity

A circuit has a *fixed* number of inputs, so one circuit per input length:

Definition

A problem is solved by a circuit family of size $T : \mathbb{N} \to \mathbb{N}$ if for every n there is a circuit of size $\leq T(n)$ deciding it on inputs of length n.

Theorem (essentially Cook–Levin)

A problem solvable in time $T(n)$ is solvable by a circuit family of size $O(T(n) \log T(n))$.

Unroll the computation: one small circuit per step, wired in sequence.

Consequence: circuit lower bounds are *stronger* than running-time lower bounds — a circuit family may even be uncomputable.

Should we believe hard problems exist?

Yes, by counting (Shannon)

- 2^{2^n} functions on n bits
- only $\approx S^{2S}$ circuits of size S
- so almost every f needs size $\geq 2^n/(10n)$

the probabilistic method, at its simplest

But we need an explicit one

- the hard function must be computable in time $2^{O(n)}$
- time hierarchy: such problems need $2^{\Omega(n)}$ *time*
- best known *circuit* lower bound for an explicit function: $\approx 5n$

So the hypothesis is a conjecture — plausible, wide open, and far beyond current techniques.

The hypothesis, restated

Assumption (hardness)

There is $f = \{f_n\}$, $f_n : \{0,1\}^n \to \{0,1\}$, computable in time $2^{O(n)}$, and $\delta > 0$ such that no circuit of size $2^{\delta n}$ computes f_n.

- "computable in time $2^{O(n)}$": the truth table of f_n can be written down in time $2^{O(n)}$ — which is poly in its length 2^n
- "requires size $2^{\delta n}$": δ may be tiny; the two exponents need not match
- this is worst-case hardness: one bad input per circuit is enough

The plan: turn this single hard function into pseudorandom bits.

Outline

Do we need randomness?

Circuits

Pseudorandom generators

The Nisan–Wigderson generator

Hardness amplification

Putting it together

Indistinguishability

Definition

Distributions $\mathcal{D}_1, \mathcal{D}_2$ over $\{0,1\}^N$ are ε-**indistinguishable** by a class $\mathcal{F}$ of functions $\{0,1\}^N \to \{0,1\}$ if

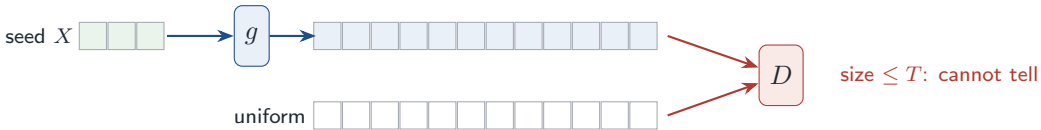
$$\left| \mathbb{E}_{x \sim \mathcal{D}_1} f(x) \; - \; \mathbb{E}_{x \sim \mathcal{D}_2} f(x) \right| \; < \; \varepsilon \qquad \text{for every } f \in \mathcal{F}.$$

- $\mathcal{F} =$ *all* functions: this is total variation distance — far too strong
- $\mathcal{F} =$ circuits of size T: computational indistinguishability
- the distributions may be miles apart — they only have to look alike to a bounded observer

Pseudorandom generators

Definition

$g : \{0,1\}^s \to \{0,1\}^N$ is an (ε, T)-**PRG** if $g(X)$, for X uniform on $\{0,1\}^s$, is ε-indistinguishable from $\mathrm{Unif}(\{0,1\}^N)$ by all circuits of size $\leq T$.



s truly random bits in, $N \gg s$ pseudorandom bits out

A short seed is all we need

Lemma

If for every N there is a $(1/10, N)$-PRG $g : \{0,1\}^{O(\log N)} \to \{0,1\}^N$ computable in time $\mathrm{poly}(N)$, then $\mathsf{P} = \mathsf{BPP}$.

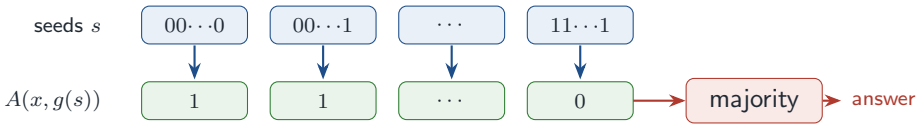
Proof, step 1: freeze the input. Let $A(x, r)$ decide $L \in \mathsf{BPP}$ in time T, using $N \leq T$ random bits.

- hard-wire x: the map $r \mapsto A(x, r)$ is computed by a circuit C_x of size $\tilde{O}(T)$
- $x \in L$: $\mathbb{E}_r\, C_x(r) \geq \frac{2}{3}$; $\quad x \notin L$: $\mathbb{E}_r\, C_x(r) \leq \frac{1}{3}$
- g fools C_x: $\left|\mathbb{E}_s\, C_x(g(s)) - \mathbb{E}_r\, C_x(r)\right| < \frac{1}{10}$

  so $\mathbb{E}_s\, C_x(g(s)) > \frac{2}{3} - \frac{1}{10} > \frac{1}{2}$ or $< \frac{1}{3} + \frac{1}{10} < \frac{1}{2}$, according to the answer

A short seed is all we need (2)

Proof, step 2: enumerate the seeds.



- $2^{O(\log N)} = \mathrm{poly}(N)$ seeds; each $g(s)$ costs $\mathrm{poly}(N)$; each run of A costs T
- the majority is correct, since a strict majority of seeds gives the right answer
- total: deterministic polynomial time ■

Everything now rests on building g.

Such generators exist — but that is not enough

Lemma

For every N there exists a $(1/10, N)$-PRG $g : \{0,1\}^{O(\log N)} \to \{0,1\}^N$.

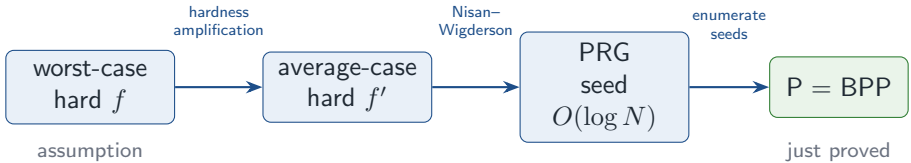
Proof. Probabilistic method: a random function on $O(\log N)$ seeds fools all $2^{O(N \log N)}$ circuits of size N, by Chernoff and a union bound. □

The catch

Existence is easy. We need g to be computable in time $\mathrm{poly}(N)$.

Note there is no circularity: g runs in time $\mathrm{poly}(N)$ and only has to fool circuits of size N — it is *allowed to be slower than its adversary*. (Cryptographic PRGs must be faster; that is why they need stronger assumptions.)

The plan



- **next:** Nisan–Wigderson — one average-case hard function, many pseudorandom bits
- **then:** hardness amplification — worst-case hardness is already enough

Outline

Do we need randomness?

Circuits

Pseudorandom generators

The Nisan–Wigderson generator

Hardness amplification

Putting it together

One hard function, many output bits

Let $f : \{0,1\}^n \to \{0,1\}$ be hard. From a seed $x \in \{0,1\}^m$ output

$$g(x) \;=\; f(x|_{S_1})\, f(x|_{S_2}) \cdots f(x|_{S_N}), \qquad S_i \subseteq [m],\ |S_i| = n.$$

Two failures

- S_i **disjoint**: perfect bits, but $m = nN$ — no stretch
- $S_i = S_j$: m is small, but two output bits are equal

The compromise

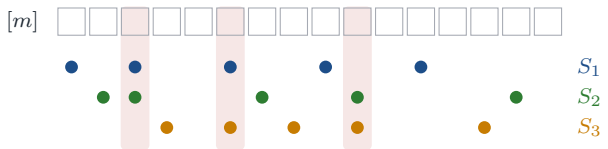
many sets, each of size n, inside a universe of size $m = O(n)$,

pairwise overlaps small

Intuition: a small overlap means that, once one block is fixed, the other output bits carry little information about it.

Definition

A family $\mathcal{S}$ of subsets of $[m]$ is an (m, n, a)-**design** if $|S| = n$ for all $S \in \mathcal{S}$, and $|S_1 \cap S_2| \leq a$ for all $S_1 \neq S_2$.



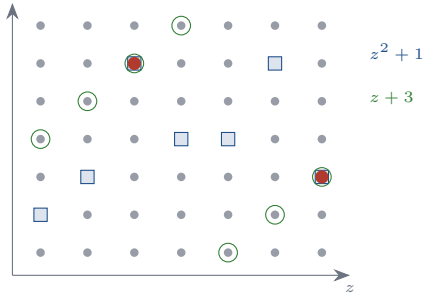
each pair overlaps in at most a places

Target: an $(cn,\ n,\ \delta n)$-design with $2^{\Omega(n)}$ sets, for constants c and δ.

Designs from low-degree polynomials

Let $\mathbb{F}_q$ be a field and take the universe $[m] = \mathbb{F}_q \times \mathbb{F}_q$. For each polynomial p with $\deg p < d$ put

$$S_p = \{(z, p(z)) \ : \ z \in \mathbb{F}_q\}.$$



$q = 7$, $d = 3$: the two graphs meet in 2 points

- $|S_p| = q$
- $|S_p \cap S_{p'}| \leq d - 1$ — two distinct polynomials of degree $< d$ agree in at most $d - 1$ points
- number of sets: q^d

The intersection bound is Schwartz–Zippel in one variable — Lecture 1, and again in Lecture 11.

Which regime do we need?

Polynomial design

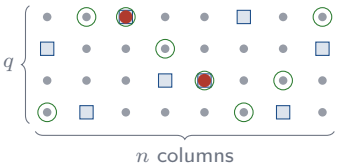
set size q, universe q^2.

Seed $m = n^2$ instead of $O(n)$: seed length $O(\log^2 N)$, hence derandomization in *quasi-polynomial* time.

What P = BPP needs

a universe of linear size:

$m = O(n),\ |S_i| = n,\ a = \delta n,\ 2^{\Omega(n)}$ sets.



one cell per column = a word in $[q]^n$;
$|S \cap S'|$ = agreement of the two words.

A design is a code: intersections $\leq \delta n \iff$ relative distance $\geq 1 - \delta$.

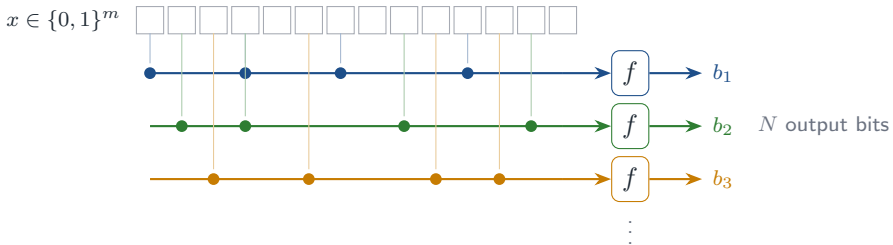
The design we need exists — by Lecture 11

Lemma

For every $\delta > 0$ there are $c = O(1/\delta)$ and $\beta > 0$ such that a $(cn,\, n,\, \delta n)$-design with $2^{\beta n}$ sets exists, and can be constructed in time $2^{O(n)}$.

- take $q = O(1/\delta)$ and a q-ary code of length n, relative distance $\geq 1 - \delta$
- such a code has rate $\beta > 0$ — the Gilbert–Varshamov argument: a random word is close to a fixed one with probability $q^{-\Omega(n)}$, then a union bound
- greedy construction: add words one at a time, scanning the $q^n = 2^{O(n)}$ candidates
- $2^{O(n)} = \mathrm{poly}(N)$ — affordable, since $N = 2^{\Omega(n)}$

The generator



$$g_f(x) = f(x|_{S_1}) \cdots f(x|_{S_N}), \quad \text{seed } m = O(n) = O(\log N), \quad \text{time } 2^{O(n)} = \text{poly}(N)$$

Theorem (Nisan–Wigderson)

Let $f : \{0,1\}^n \to \{0,1\}$ be computable in time $2^{O(n)}$ and suppose that every circuit C of size $2^{\delta n}$ satisfies

$$\mathop{\mathbb{P}}_{x \sim \{0,1\}^n} \left[C(x) = f(x) \right] \;\leq\; \tfrac{1}{2} + \gamma, \qquad \gamma = 2^{-\delta n/3}.$$

Then, for an $(cn, n, \frac{\delta n}{2})$-design with $N = 2^{\delta n/4}$ sets, g_f is a $(1/10, N)$-PRG computable in time $\mathrm{poly}(N)$.

With the lemma from the previous section: such an f gives P = BPP.

The required advantage γ is *exponentially* small — the proof loses a factor N. With $\gamma = 1/\mathrm{poly}(n)$ the same argument still gives a seed of length $O(\log^2 N)$, i.e. quasi-polynomial derandomization.

Proof, step 1: hybrids

Suppose a circuit D of size N distinguishes $g_f(X)$ from uniform with advantage $\varepsilon = \frac{1}{10}$.



Telescoping: some i has $\left|\mathbb{E}D(H_i) - \mathbb{E}D(H_{i-1})\right| \geq \frac{\varepsilon}{N}$.

H_i: the first i bits come from f, the rest are uniform.

Proof, step 2: distinguishing ⇒ predicting

Yao's next-bit trick

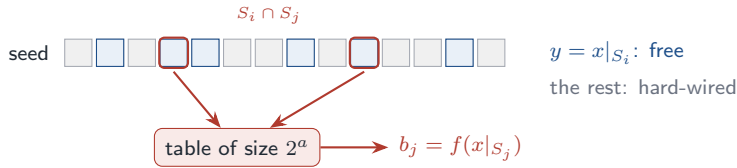
A circuit that distinguishes H_{i-1} from H_i with advantage η yields a circuit P with

$$\mathop{\mathbb{P}}_{x}\Big[\, P\big(b_1,\ldots,b_{i-1}\big) = b_i \,\Big] \;\geq\; \tfrac{1}{2}+\eta, \qquad b_j = f(x|_{S_j}).$$

- P: guess b_i by a coin r, run D on $(b_1,\ldots,b_{i-1},r,r_{i+1},\ldots,r_N)$; output r if D says 1, else $1-r$
- the two hybrids differ exactly in whether position i is f or a coin
- the trailing coins can be fixed to their best values — P is a *circuit*

So: a predictor for $f(x|_{S_i})$ of size $N + O(N)$, advantage ε/N.

Proof, step 3: freeze everything outside S_i



- fix the bits outside S_i to their best values: the advantage survives
- then b_j depends on y only through $S_i \cap S_j$ — at most a bits — so a lookup table of size 2^a computes it
- the predictor becomes a circuit computing $f(y)$ from y, of size $N + N \cdot 2^a + O(N)$

Proof, step 4: count

With $N = 2^{\delta n/4}$ and $a = \dfrac{\delta n}{2}$:

Size

$$N + N \cdot 2^a + O(N) \;\le\; 2^{3\delta n/4} \;<\; 2^{\delta n}$$

Advantage

$$\frac{\varepsilon}{N} = \frac{2^{-\delta n/4}}{10} \;>\; \gamma = 2^{-\delta n/3}$$

A circuit of size $< 2^{\delta n}$ agreeing with f on more than $\frac{1}{2} + \gamma$ of the inputs — contradicting the hypothesis. ■

Conclusion: an average-case hard f gives the PRG, hence P = BPP.

It remains to build f — from a merely *worst-case* hard function.

Outline

Do we need randomness?

Circuits

Pseudorandom generators

The Nisan–Wigderson generator

Hardness amplification

Putting it together

Worst case is enough

Theorem (Impagliazzo–Wigderson, amplification)

If f is computable in time $2^{O(n)}$ and requires circuits of size $2^{\delta n}$, then there is f' on $n' = O(n)$ bits, computable in time $2^{O(n')}$, such that no circuit of size $2^{\delta' n'}$ agrees with f' on more than a $\frac{1}{2} + 2^{-\delta' n'}$ fraction of inputs.



Three steps, each one a coding-theoretic argument.

Warm-up: the permanent

Definition

$$\mathrm{perm}(A) = \sum_{\pi \in S_n} \prod_{i=1}^{n} A_{i,\pi(i)}$$

- a polynomial of degree n in the n^2 entries
- believed to be very hard (#P-complete over the integers)

The question

Suppose an algorithm computes perm correctly on $9/10$ of all matrices.

Can we compute it on *every* matrix?

Theorem

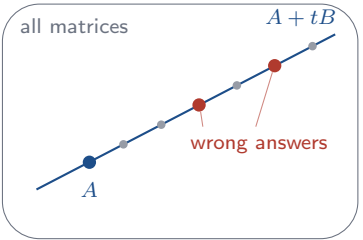
If $|\mathbb{F}| > 3n$ and some algorithm running in time $T(n)$ computes perm correctly on a uniformly random input with probability $9/10$, then perm can be computed on *every* input by a randomized algorithm running in time $O(n\,T(n)) + \mathrm{poly}(n)$.

"Average-case easy" $\Rightarrow$ "worst-case easy". We will use the contrapositive.

Take a random line

To compute $\mathrm{perm}(A)$: pick a uniformly random matrix B and set

$$h(t) \;:=\; \mathrm{perm}(A + tB).$$



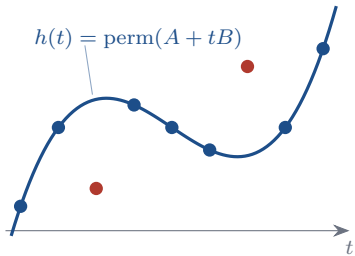
- h is a univariate polynomial of degree $\leq n$
- $h(0) = \mathrm{perm}(A)$ — the value we want
- for every fixed $t \neq 0$, $A + tB$ is uniformly distributed

So the oracle is right on each queried point with probability $9/10$ — but the queries are far from independent.

Decode the line

Query the oracle at $3n$ distinct nonzero points $t_1, \ldots, t_{3n}$:

- $\mathbb{E}[\#\text{wrong answers}] \le \frac{3n}{10}$, so by Markov fewer than n are wrong with probability ≥ 0.7
- $3n$ evaluations of a degree-n polynomial tolerate $\frac{3n-n}{2} = n$ errors
- **Welch–Berlekamp** (Lecture 11) recovers h, and we output $h(0)$
- repeat with fresh lines, take a majority



last week's picture, in a new role

This is random self-reducibility: low-degree polynomials are their own error-correcting code.

Step 1: encode the truth table

Let $f : \{0,1\}^n \to \{0,1\}$ be worst-case hard. Pick $\mathbb{F}$ with $|\mathbb{F}| = \mathrm{poly}(n)$, a subset $H \subseteq \mathbb{F}$, and identify $\{0,1\}^n$ with H^ℓ, $\ell = n/\log|H|$.

Low-degree extension (Reed–Muller encoding)

the unique $\hat{f} : \mathbb{F}^\ell \to \mathbb{F}$ of degree $< |H|$ in each variable with $\hat{f}|_{H^\ell} = f$

- computable in time $2^{O(n)}$; inputs are still $O(n)$ bits long
- total degree $d = \ell(|H| - 1) \ll |\mathbb{F}|$ — so the line trick applies verbatim
- if a circuit of size S computes $\hat{f}$ on 99% of $\mathbb{F}^\ell$, then a circuit of size $S \cdot \mathrm{poly}(n)$ computes $\hat{f}$, hence f, everywhere

Contrapositive: f worst-case hard $\implies$ $\hat{f}$ is hard on 1% of all inputs.

Step 2: past the unique-decoding radius

From "hard on 1% of inputs" to "unpredictable with advantage $1/\mathrm{poly}$": encode each value by the **Hadamard code**, $f''(x,r) = \langle \hat{f}(x), r \rangle$.

Goldreich–Levin (local list decoding of the Hadamard code)

Given oracle access to g with $\mathbb{P}_x[g(x) = \langle a, x \rangle] \geq \frac{1}{2} + \varepsilon$, one can output, in time $\mathrm{poly}(n, 1/\varepsilon)$, a list of $O(1/\varepsilon^2)$ candidates containing a.

- agreement $\frac{1}{2} + \varepsilon$ is beyond unique decoding — a list is unavoidable (Lecture 11)
- local: we must output a small *circuit*, so we may only query the corrupted word — never write it down
- which list element? $O(\log 1/\varepsilon)$ bits of non-uniform advice

Proof: a student presentation later in the term.

Step 3: all the way down to $\frac{1}{2} + 2^{-\Omega(n)}$

Yao's XOR lemma

If no circuit of size S computes f on more than $1 - \rho$ of the inputs, then no circuit of size S' computes $f^{\oplus k}(x_1, \ldots, x_k) = f(x_1) \oplus \cdots \oplus f(x_k)$ on more than $\frac{1}{2} + e^{-\Omega(\rho k)}$ of the inputs.

- with $k = \Theta(n)$ the advantage is $2^{-\Omega(n)}$ — exactly what NW wants
- but the input is now kn bits long, and hardness $2^{\delta' kn}$ against inputs of length kn is no gain
- **Impagliazzo–Wigderson:** derandomize the direct product — take $x_1, \ldots, x_k$ to be a random walk of length k on an expander (Lecture 5), costing only $n + O(k)$ bits

The proof of the XOR lemma itself goes through Impagliazzo's hardcore lemma: a function hard on a ρ fraction has a "hardcore set" on which it is essentially unpredictable.

The amplification chain



- every step keeps the input length $O(n)$ and the evaluation time $2^{O(n)}$
- every step is a *decoding* algorithm: hardness is preserved because errors can be corrected
- all reductions are non-uniform — they map circuits to circuits

Outline

Do we need randomness?

Circuits

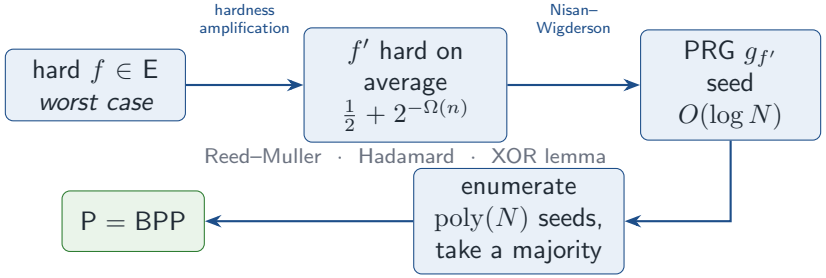
Pseudorandom generators

The Nisan–Wigderson generator

Hardness amplification

Putting it together

The whole argument



Theorem (Impagliazzo–Wigderson)

If some language in E requires circuits of size $2^{\delta n}$ for a constant $\delta > 0$, then P = BPP: every polynomial-time randomized algorithm can be simulated deterministically with polynomial overhead.

And hardness is also necessary

The converse direction holds as well: derandomization would prove circuit lower bounds.

Theorem (Kabanets–Impagliazzo, 2004)

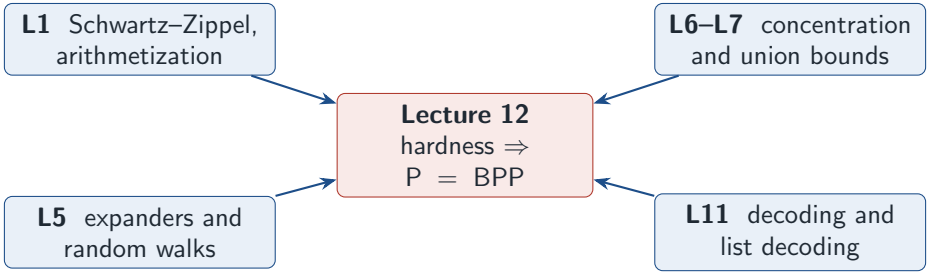
If polynomial identity testing has a deterministic polynomial-time algorithm, then

- $\mathrm{NEXP} \not\subseteq \mathrm{P/poly}$, **or**
- the permanent has no arithmetic circuits of polynomial size.

Derandomizing *the very first problem of this course* requires proving a lower bound that nobody knows how to prove.

Hardness and pseudorandomness are two views of one phenomenon: a lower bound buys an algorithm, and the algorithm buys back a lower bound.

What today's proof borrowed



- low-degree polynomials: the design, and the Reed–Muller encoding of a truth table
- expander walks: the derandomized direct product
- union bound + Chernoff: existence of PRGs, and of the design
- Welch–Berlekamp and list decoding: self-correction and Goldreich–Levin

Closing

To read

- Arora–Barak, *Computational Complexity*, ch. 19–20
- Vadhan, *Pseudorandomness*
- Nisan–Wigderson '94; Impagliazzo–Wigderson '97
- Sudan–Trevisan–Vadhan '01; Kabanets–Impagliazzo '04

Coming up

- student presentation: the Goldreich–Levin algorithm
- open: a deterministic algorithm for PIT
- open: any $2^{\Omega(n)}$ circuit lower bound

The course in one sentence:

randomness, structure and hardness are the same subject, seen from three sides.