אוניברסיטת
אריאל
בשומרון

Computational Complexity

Lecturer: Edward A. Hirsch

https://edwardahirsch.github.io/edwardahirsch

edward.a.hirsch@gmail.com

December 31, 2025

Lecture plan (including next lecture(s))

- Randomized Computation *continued*.
  - Where is **BPP** relative to non-randomized classes?
    BPP $\subseteq$ **P/poly**, **BPP** $\subseteq \Sigma_2^p$.
  - Probabilistic reductions.

- Interactive protocols.
  - Private and public coin. Definitions of **MA**, **AM**, **IP**.
  - Examples: Graph (non)isomorphism, Set size estimation.
  - Where are **MA** and **AM**?
  - **IP** = **PSPACE**.

- The PCP Theorem and hardness of approximation.
  - Statement of the theorem.
  - Approximation algorithms.
  - Inapproximability as a corollary to the PCP Theorem.
  - Proof of the theorem.

Reminder: A PPTM, what is it?

If we have a $p(n)$-time machine (poly-time NTM, PPTM), we can assume that its witnesses and random strings have this exact length $p(n)$ (unless we are interested in shortening them).

A probabilistic Turing machine can be viewed as

- a DTM that has a dedicated readonly random tape, *// This is the default view.*
  it allows to use the probability theory: nice proofs if you remember it!
- a witness-type NTM where witness is the random tape,
  useful if we count the witnesses in an easy combinatorial way,
- a NTM making decisions – just they are now randomized, not nondeterministic,
  visualized as a computation tree.

A PPTM is a particular case of a poly-time NTM.

So what's the difference?

- Every poly-time NTM defines *some* language in **NP**.
- Not every PPTM defines a language in, say, **RP**.

```
Stupid PPTM (input x):
-- a:=random({0,1});  //toss a random coin
-- return a;
```

This one defines no language in **RP** or **BPP**.

Reminder: Two-sided bounded error

a probabilistic Turing machine: with a random string instead of a witness

A two-sided bounded error algorithm A for language L:

- $\forall x \in L \qquad \Pr\{A(x) = 1\} \geqslant \frac{3}{4}.$ // *False negative with probability* $\leqslant 1/4$
- $\forall x \notin L \qquad \Pr\{A(x) = 1\} \leqslant \frac{1}{4}.$ // *False positive with probability* $\leqslant 1/4$

In total:

$$\Pr\{A(x) \neq L(x)\} \leqslant \frac{1}{4}.$$

Class **BPTime**$[t(n)]$: languages with such $O(t(n))$-time algorithms.

$$\mathbf{BPP} = \bigcup_{k \in \mathbb{N}} \mathbf{BPTime}[n^k]$$

By repeating it a polynomial number of times, we can decrease the probability of error from $\frac{1}{4}$ and even from $\frac{1}{2} - \frac{1}{\mathrm{poly}(n)}$ to $\frac{1}{2^{\mathrm{poly}(n)}}$. Time up to poly! For specific n^k you need to count!

In particular, $\ell(n)$ iterations decrease it from $\frac{1}{4}$ to $\frac{1}{2^{\Omega(\ell(n))}}$.

Non-uniform advice: another view of **P/poly**

Definition 1. $L \in \mathbf{P/poly}$ if there is a family $\{C_n\}_{n\in\mathbb{N}}$ of poly-size circuits such that for every input x,

$$x \in L \iff C_{|x|}(x) = 1.$$

For each input length n there is a circuit C_n.

Definition 2. $L \in \mathbf{P/poly}$ if there is a sequence $\{a_n\}_{n\in\mathbb{N}}$ of poly-size strings and a poly-time DTM M such that for every input x,

$$x \in L \iff M(\langle x, a_{|x|}\rangle) = 1.$$

For each input length n there is an advice string a_n.

Equivalence:

$\boxed{\Rightarrow}$ $a_n = C_n$ and $M(x, C) = C(x)$.

Non-uniform advice: another view of **P/poly**

Definition 1. $L \in \mathbf{P/poly}$ if there is a family $\{C_n\}_{n\in\mathbb{N}}$ of poly-size circuits such that for every input x,

$$x \in L \iff C_{|x|}(x) = 1.$$

For each input length n there is a circuit C_n.

Definition 2. $L \in \mathbf{P/poly}$ if there is a sequence $\{a_n\}_{n\in\mathbb{N}}$ of poly-size strings and a poly-time DTM M such that for every input x,

$$x \in L \iff M(\langle x, a_{|x|}\rangle) = 1.$$

For each input length n there is an advice string a_n.

Equivalence:

$\boxed{\Rightarrow}$ $a_n = C_n$ and $M(x, C) = C(x)$.

$\boxed{\Leftarrow}$ C_n simulates M on $\langle x, a_n\rangle$ for (known) poly number of steps, see **NP**-completeness of `Circuit-SAT`. a_n is hardwired into C_n.

Non-uniform advice: another view of **P/poly**

Definition 1. $L \in \mathbf{P/poly}$ if there is a family $\{C_n\}_{n\in\mathbb{N}}$ of poly-size circuits such that for every input x,

$$x \in L \iff C_{|x|}(x) = 1.$$

For each input length n there is a circuit C_n.

Definition 2. $L \in \mathbf{P/poly}$ if there is a sequence $\{a_n\}_{n\in\mathbb{N}}$ of poly-size strings and a poly-time DTM M such that for every input x,

$$x \in L \iff M(\langle x, a_{|x|}\rangle) = 1.$$

For each input length n there is an advice string a_n.

Equivalence:

$\boxed{\Rightarrow}$ $a_n = C_n$ and $M(x, C) = C(x)$.

$\boxed{\Leftarrow}$ C_n simulates M on $\langle x, a_n\rangle$ for (known) poly number of steps, see **NP**-completeness of `Circuit-SAT`. a_n is hardwired into C_n.

NB!!! String a_n is a non-uniform advice and not a witness.

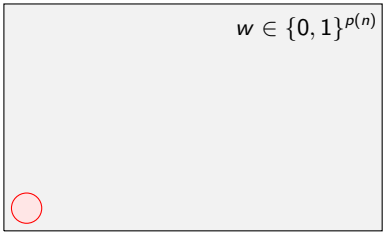
- We do not need to verify it, we believe it and just use it.
- We get the same string a_n for every input of the same length n.

BPP $\subset$ P/poly

Let A be a **BPP** algorithm. We write $A(x, w)$ where x is the input and w is the random string.

- A "good" random string w for x leads to no error: $A(x, w) = L(x)$.

  We can assume the share of such "good" random strings $\geqslant 1 - \frac{1}{4^n}$ ("bad" strings: at most $\frac{1}{4^n}$).



BPP $\subset$ P/poly

Let A be a **BPP** algorithm. We write $A(x, w)$ where x is the input and w is the random string.

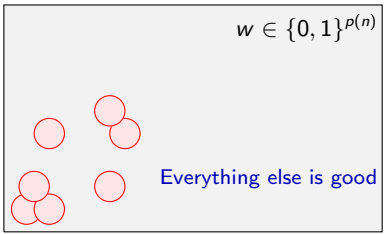
- A "good" random string w for x leads to no error: $A(x, w) = L(x)$.

  We can assume the share of such "good" random strings $\geqslant 1 - \frac{1}{4^n}$ ("bad" strings: at most $\frac{1}{4^n}$).

- There exists a random string that is good for every input:

$$\text{"share of bad strings for one input"} \times \text{"number of possible inputs"} \leqslant \frac{1}{4^n} \times 2^n < 1.$$

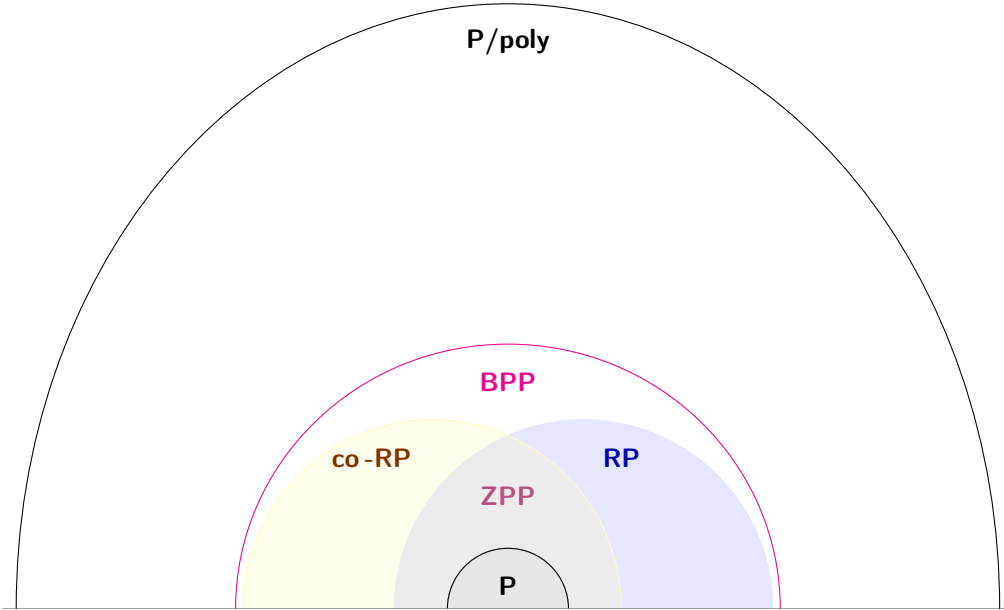
- Let a_n be the random string that is good for all $x \in \{0,1\}^n$.



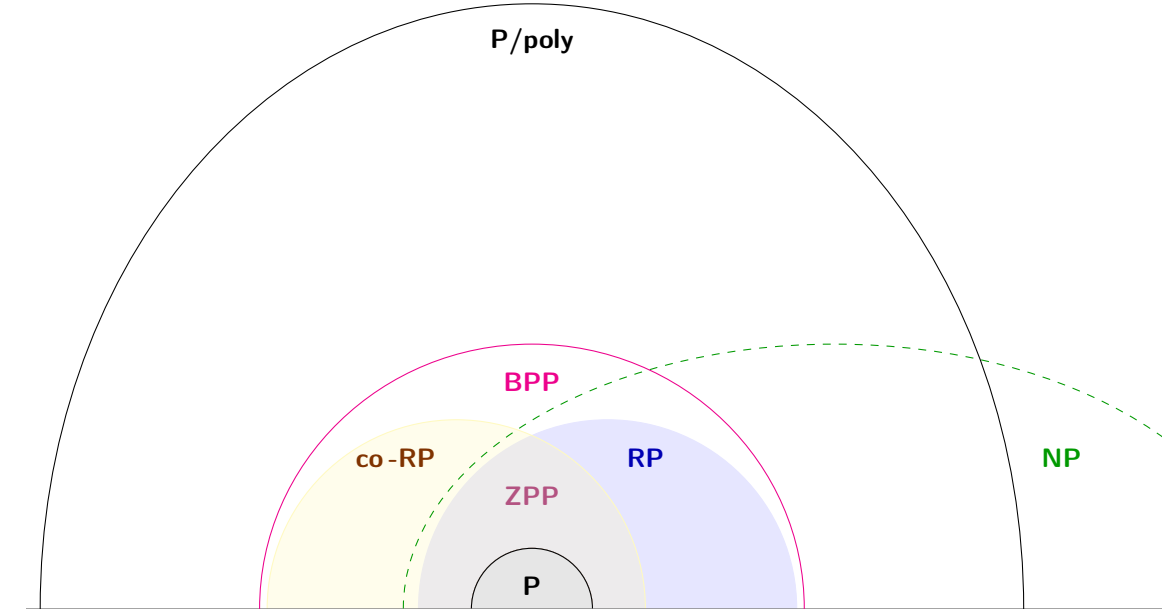
Theorem

BPP $\subset$ P/poly

Randomized algorithms inside **P/poly**



Randomized algorithms inside **P/poly**



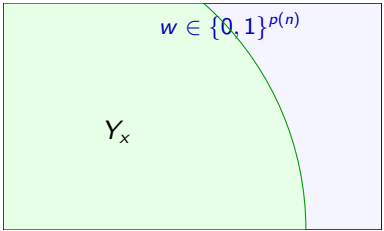
$\mathbf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$

Theorem

$\mathbf{BPP} \subseteq \Sigma_2^p$.

Consider a probabilistic algorithm $A(x, w)$, where w is its random string (of length $p(n)$).

- $Y_x := \{w \in \{0,1\}^{p(n)} \mid A(x, w) = 1\}$: random strings leading to the acceptance.
  This is not the set of "good" random strings! They are good only for the answer "yes".
- Assume the error probability $\frac{1}{2^n}$. Then if $x \in L$, then $|Y_x| \geqslant \left(1 - \frac{1}{2^n}\right) \cdot 2^{p(n)}$.



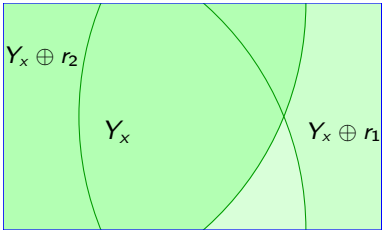
$\mathbf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$

Theorem

$\mathbf{BPP} \subseteq \Sigma_2^p$.

Consider a probabilistic algorithm $A(x, w)$, where w is its random string (of length $p(n)$).

- $Y_x := \{w \in \{0,1\}^{p(n)} \mid A(x, w) = 1\}$: random strings leading to the acceptance.
  This is not the set of "good" random strings! They are good only for the answer "yes".
- Assume the error probability $\frac{1}{2^n}$. Then if $x \in L$, then $|Y_x| \geqslant \left(1 - \frac{1}{2^n}\right) \cdot 2^{p(n)}$.
- For $x \in L$, cover the whole set $U = \{0,1\}^{p(n)}$ using "xor-copies" of Y_x



$$Y \oplus r = \{w \oplus r : w \in Y\},$$

where $(w \oplus r)_i = w_i \oplus r_i$, bitwise xor

$\mathbf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$

Theorem

$\mathbf{BPP} \subseteq \Sigma_2^p$.

Consider a probabilistic algorithm $A(x, w)$, where w is its random string (of length $p(n)$).

- $Y_x := \{w \in \{0,1\}^{p(n)} \mid A(x,w) = 1\}$: random strings leading to the acceptance.
  This is not the set of "good" random strings! They are good only for the answer "yes".
- Assume the error probability $\frac{1}{2^n}$. Then if $x \in L$, then $|Y_x| \geqslant \left(1 - \frac{1}{2^n}\right) \cdot 2^{p(n)}$.
- For $x \in L$, cover the whole set $U = \{0,1\}^{p(n)}$ using d "xor-copies" of Y_x (let us compute this d):

$$\exists \{t_i\}_{i=1}^d \quad \forall r \in U \quad \bigvee_{i=1}^d (r \in Y_x \oplus t_i),$$

- $d = p(n)$ copies suffice for it: Consider a random collection $T = \{t_1, \ldots, t_d\}$ just for proving $\exists$:

$$\Pr_T\{\neg(\forall r \in U \quad \bigvee_{i=1}^d (r \in Y_x \oplus t_i))\} \overset{\text{de Morgan}}{=\!=} \Pr_T\{\exists r \in U \quad \bigwedge_{i=1}^d (r \notin Y_x \oplus t_i)\} \overset{\text{union bound}}{\leq}$$

$$\sum_{r \in U} \Pr_T\{\bigwedge_{i=1}^d (r \notin Y_x \oplus t_i)\} \overset{t_i\text{'s independent}}{=\!=} \sum_{r \in U} \prod_{i=1}^d \Pr_T\{r \notin Y_x \oplus t_i\} \overset{|Y_x \oplus t_i| = |Y_x| \text{ is big!}}{\leq} \frac{1}{2^{nd}} 2^{p(n)} \overset{\text{choose } d}{<} 1.$$

$\mathbf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$

Theorem

$\mathbf{BPP} \subseteq \Sigma_2^p$.

Consider a probabilistic algorithm $A(x, w)$, where w is its random string (of length $p(n)$).

- $Y_x := \{w \in \{0,1\}^{p(n)} \mid A(x, w) = 1\}$: random strings leading to the acceptance.
  This is not the set of "good" random strings! They are good only for the answer "yes".
- Assume the error probability $\frac{1}{2^n}$. Then if $x \in L$, then $|Y_x| \geqslant \left(1 - \frac{1}{2^n}\right) \cdot 2^{p(n)}$.
- For $x \in L$, cover the whole set $U = \{0,1\}^{p(n)}$ using d "xor-copies" of Y_x (let us compute this d):

$$\exists \{t_i\}_{i=1}^{d} \quad \forall r \in U \quad \bigvee_{i=1}^{d} (r \in Y_x \oplus t_i),$$

- This is a two quantifier $\exists\forall$ condition, like in Σ_2^p!

  To check $r \in Y_x \oplus t_i$ in polynomial time: check $r \oplus t_i \in Y_x$, i.e., run $A(x, r \oplus t_i)$.

$\mathbf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$

Theorem

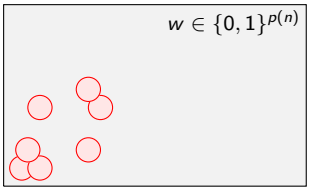
$\mathbf{BPP} \subseteq \Sigma_2^p$.

Consider a probabilistic algorithm $A(x, w)$, where w is its random string (of length $p(n)$).

- $Y_x := \{w \in \{0,1\}^{p(n)} \mid A(x, w) = 1\}$: random strings leading to the acceptance.
  This is not the set of "good" random strings! They are good only for the answer "yes".
- Assume the error probability $\frac{1}{2^n}$. Then if $x \in L$, then $|Y_x| \geqslant \left(1 - \frac{1}{2^n}\right) \cdot 2^{p(n)}$.
- For $x \in L$, cover the whole set $U = \{0,1\}^{p(n)}$ using d "xor-copies" of Y_x (let us compute this d):

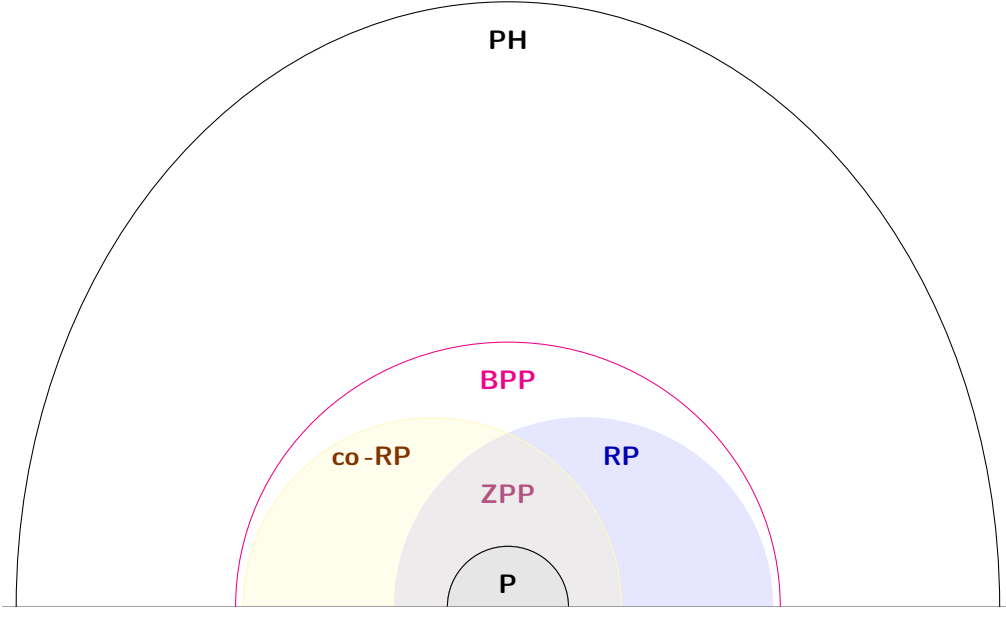
$$\exists \{t_i\}_{i=1}^{d} \quad \forall r \in U \quad \bigvee_{i=1}^{d} (r \in Y_x \oplus t_i),$$

  For $x \notin L$, one cannot cover it just because of the cardinality!

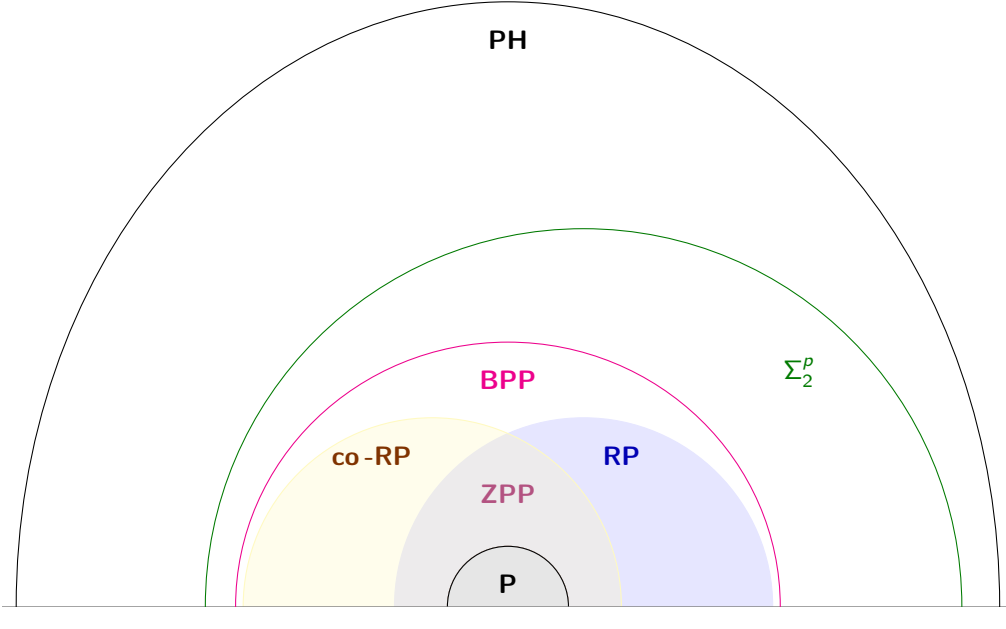
- 

$$\frac{1}{2^n} \times d \quad < \quad 1$$

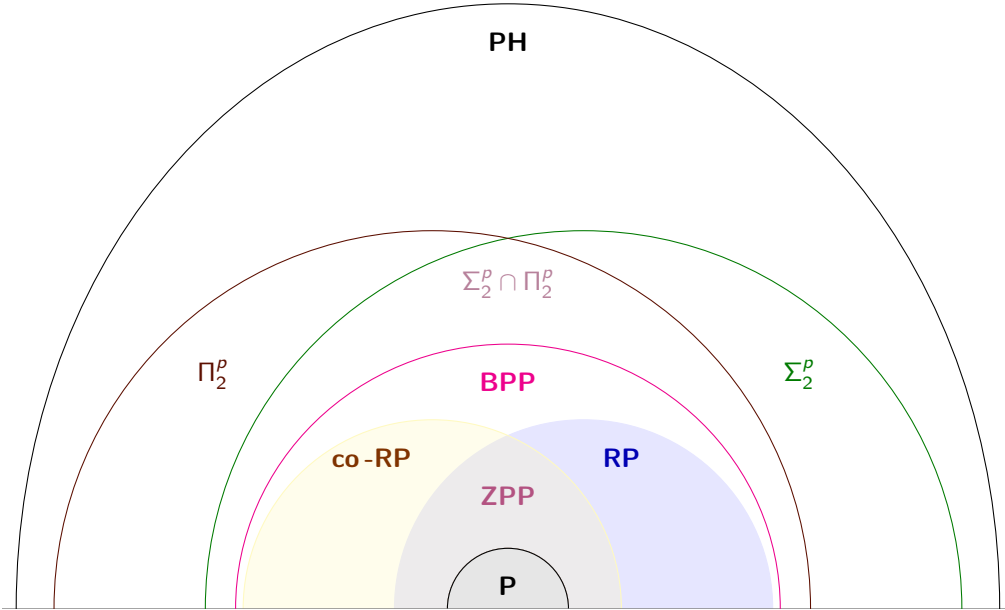
Randomized algorithms inside **PH**



Randomized algorithms inside **PH**



Randomized algorithms inside **PH**



Solution isolation by randomized reduction

Plan:

- "Unique-SAT": Find a satisfying assignment if it is promised there is at most one.
- Is it easier to find a satisfying assignment for F if it is unique?
- Formally and actually — yes.
  For example, for every variable there must be a clause satisfied by this variable only: $(\overline{x_1} \vee x_2 \vee \overline{x_3})$.
  It can improve (exponential) running time — see ***Boolean*** `SAT` ***course***.
- It turns out that w.r.t. polynomial time there is no gain.

Solution isolation by randomized reduction

Plan:

- "Unique-SAT": Find a satisfying assignment if it is promised there is at most one.
- Is it easier to find a satisfying assignment for F if it is unique?
- Formally and actually — yes.
  For example, for every variable there must be a clause satisfied by this variable only: $(\overline{x_1} \vee x_2 \vee \overline{x_3})$.
  It can improve (exponential) running time — see ***Boolean*** SAT ***course***.
- It turns out that w.r.t. polynomial time there is no gain.

Strategy:

- If there are many sat. assignments, blindly pick one of them using hashing.
- Change $F(x)$ into $F(x) \wedge h(x) = z$ for randomly chosen h and z.
- If we can modify F this way with high probability,
  then we can solve SAT using an oracle for "Unique-SAT".

Theorem

If one can solve Unique-SAT in polynomial time, then **NP** = **RP**.

A family of pairwise-independent hash-functions

Definition

$H_{n,k}$ is a family of pairwise-independent hash-functions if

- it contains functions $h : \{0,1\}^n \to \{0,1\}^k$,
- for every $x \neq x'$, for every y, y', $\Pr_{h \leftarrow H_{n,k}}\{h(x) = y \wedge h(x') = y'\} = 2^{-2k}$.

(In particular, $\Pr_h\{h(x) = y\} = 2^{-k}$.)

A family of pairwise-independent hash-functions

Definition

$H_{n,k}$ is a family of pairwise-independent hash-functions if

- it contains functions $h : \{0,1\}^n \to \{0,1\}^k$,
- for every $x \neq x'$, for every y, y', $\Pr_{h \leftarrow H_{n,k}}\{h(x) = y \wedge h(x') = y'\} = 2^{-2k}$.

(In particular, $\Pr_h\{h(x) = y\} = 2^{-k}$.)

Lemma (construction of such a family)

The field with 2^n elements. If you don't know, one think about another field.

Identify $\{0,1\}^n$ with $GF(2^n)$, then one can take $H_{n,n}$ composed of functions

$$h_{a,b}(x) = ax + b \qquad (a, b \in GF(2^n)).$$

Proof: if $h_{a,b}(x) = y$ and $h_{a,b}(x') = y'$, then a, b is unambiguously defined by

$$\begin{aligned} a \cdot x + b &= y \\ a \cdot x' + b &= y'. \end{aligned}$$

Thus for every $x \mapsto y$ and $x' \mapsto y'$ there is just one function out of $\frac{1}{2^{2n}}$.

A family of pairwise-independent hash-functions

Definition

$H_{n,k}$ is a family of pairwise-independent hash-functions if

- it contains functions $h : \{0,1\}^n \to \{0,1\}^k$,
- for every $x \neq x'$, for every y, y', $\Pr_{h \leftarrow H_{n,k}}\{h(x) = y \wedge h(x') = y'\} = 2^{-2k}$.

(In particular, $\Pr_h\{h(x) = y\} = 2^{-k}$.)

Lemma (construction of such a family)

The field with 2^n elements. If you don't know, one think about another field.

Identify $\{0,1\}^n$ with $GF(2^n)$, then one can take $H_{n,n}$ composed of functions

$$h_{a,b}(x) = ax + b \qquad (a, b \in GF(2^n)).$$

Proof: if $h_{a,b}(x) = y$ and $h_{a,b}(x') = y'$, then a, b is unambiguously defined by

$$\begin{aligned} a \cdot x + b &= y \\ a \cdot x' + b &= y'. \end{aligned}$$

Thus for every $x \mapsto y$ and $x' \mapsto y'$ there is just one function out of $\frac{1}{2^{2n}}$.

For $k < n$ truncate outputs of $H_{n,n}$ to k bits. (Exercise!) For $k > n$ extend inputs of $H_{k,k}$ to k bits.

The Valiant–Vazirani Lemma

Lemma (Valiant, Vazirani)

There is a polynomial-time computable (with randomness) function f mapping every F to G such that

- $F \notin \mathtt{SAT} \Rightarrow G \notin \mathtt{SAT}$,
- $F \in \mathtt{SAT} \Rightarrow \Pr\{\exists! y \ G(y) = 1\} \geqslant \frac{1}{8n}$.

there exists a unique

$G(x) = (F(x) \wedge h(x) = 0)$, where h is randomly selected from $H_{n,k}$.

Actually, $G(x, x')$: One needs to express $h(x)$ in CNF using auxilary variables:
see Tseitin's reduction $\mathtt{Circuit\text{-}SAT} \to^p \mathtt{3\text{-}SAT}$, it does not change the number of sat. assignments.

The Valiant–Vazirani Lemma

Lemma (Valiant, Vazirani)

There is a polynomial-time computable (with randomness) function f mapping every F to G such that

- $F \notin \texttt{SAT} \Rightarrow G \notin \texttt{SAT}$,
- $F \in \texttt{SAT} \Rightarrow \Pr\{\exists! y\ G(y) = 1\} \geqslant \frac{1}{8n}$.

there exists a unique

$G(x) = (F(x) \wedge h(x) = 0)$, where h is randomly selected from $H_{n,k}$.

Actually, $G(x, x')$: One needs to express $h(x)$ in CNF using auxilary variables:
see Tseitin's reduction $\texttt{Circuit-SAT} \to^p \texttt{3-SAT}$, it does not change the number of sat. assignments.

- How to select k?
- Let S be the set of all satisfying assignments to F.
- Choose k such that $2^{k-2} < |S| < 2^{k-1}$.
  (Just guess: there are only n possibilities.)
- $\Pr\limits_{h \in H_{n,k}} \{|S \cap \{x\colon h(x) = 0\}| = 1\} \geqslant \dfrac{1}{8}$:

The Valiant–Vazirani Lemma

Lemma (Valiant, Vazirani)

There is a polynomial-time computable (with randomness) function f mapping every F to G such that

- $F \notin \mathtt{SAT} \Rightarrow G \notin \mathtt{SAT}$,
- $F \in \mathtt{SAT} \Rightarrow \Pr\{\exists!y\ G(y) = 1\} \geqslant \frac{1}{8n}$.

there exists a unique

$G(x) = (F(x) \wedge h(x) = 0)$, where h is randomly selected from $H_{n,k}$.

Actually, $G(x, x')$: One needs to express $h(x)$ in CNF using auxilary variables:
see Tseitin's reduction $\mathtt{Circuit\text{-}SAT} \to^p \mathtt{3\text{-}SAT}$, it does not change the number of sat. assignments.

- How to select k?
- Let S be the set of all satisfying assignments to F.
- Choose k such that $2^{k-2} < |S| < 2^{k-1}$.
  (Just guess: there are only n possibilities.)
- $\Pr\limits_{h \in H_{n,k}} \{|S \cap \{x\colon h(x) = 0\}| = 1\} \geqslant \dfrac{1}{8}$:

Let $N = |\{x \in S :\ h(x) = 0\}|$. (Ideally, $N = 1$, but it's not yet.)

$\Pr\{N \geqslant 1\} \geqslant \sum\limits_{x \in S} \Pr\{h(x) = 0\} - \sum\limits_{x, x' \in S} \Pr\{h(x) = 0,\ h(x') = 0\} = |S| 2^{-k} - \binom{|S|}{2} 2^{-2k}$.

$\Pr\{N \geqslant 2\} =$ the number of pairs $\times$ probability to keep both assignments $\leqslant \binom{|S|}{2} 2^{-2k}$.

The Valiant–Vazirani Lemma

Lemma (Valiant, Vazirani)

There is a polynomial-time computable (with randomness) function f mapping every F to G such that

- $F \notin \mathtt{SAT} \Rightarrow G \notin \mathtt{SAT}$,
- $F \in \mathtt{SAT} \Rightarrow \Pr\{\exists! y\ G(y) = 1\} \geqslant \frac{1}{8n}$.

there exists a unique

$G(x) = (F(x) \wedge h(x) = 0)$, where h is randomly selected from $H_{n,k}$.

Actually, $G(x, x')$: One needs to express $h(x)$ in CNF using auxilary variables:
see Tseitin's reduction $\mathtt{Circuit\text{-}SAT} \to^p \mathtt{3\text{-}SAT}$, it does not change the number of sat. assignments.

- How to select k?
- Let S be the set of all satisfying assignments to F.
- Choose k such that $2^{k-2} < |S| < 2^{k-1}$.
  (Just guess: there are only n possibilities.)
- $\Pr\limits_{h \in H_{n,k}}\{|S \cap \{x\colon h(x) = 0\}| = 1\} \geqslant \frac{1}{8}$:

$\Pr\{N = 1\} =$
$\Pr\{N \geqslant 1\} - \Pr\{N \geqslant 2\} \geqslant$
$|S|2^{-k} - |S|^2 2^{-2k} =$
$|S|2^{-k}(1 - |S|2^{-k}) \geqslant \frac{1}{8}$.

Let $N = |\{x \in S\colon\ h(x) = 0\}|$. (Ideally, $N = 1$, but it's not yet.)

$\Pr\{N \geqslant 1\} \geqslant \sum\limits_{x \in S} \Pr\{h(x) = 0\} - \sum\limits_{x, x' \in S} \Pr\{h(x) = 0,\ h(x') = 0\} = |S|2^{-k} - \binom{|S|}{2} 2^{-2k}$.

$\Pr\{N \geqslant 2\} =$ the number of pairs $\times$ probability to keep both assignments $\leqslant \binom{|S|}{2} 2^{-2k}$.

The Valiant–Vazirani Lemma

Lemma (Valiant, Vazirani)

There is a polynomial-time computable (with randomness) function f mapping every F to G such that

- $F \notin \mathtt{SAT} \Rightarrow G \notin \mathtt{SAT}$,
- $F \in \mathtt{SAT} \Rightarrow \Pr\{\exists! y \ G(y) = 1\} \geqslant \frac{1}{8n}$.

there exists a unique

$G(x) = (F(x) \wedge h(x) = 0)$, where h is randomly selected from $H_{n,k}$.

Actually, $G(x, x')$: One needs to express $h(x)$ in CNF using auxilary variables:
see Tseitin's reduction $\mathtt{Circuit\text{-}SAT} \to^p \mathtt{3\text{-}SAT}$, it does not change the number of sat. assignments.

Corollary

If SAT can be solved in polynomial time for uniquely satisfiable formulas, then **RP** = **NP**.

Repeat $8n$ times (trying to apply the imaginary Unique-SAT algorithm to every G obtained).

Pr. of error: $\left(1 - \frac{1}{8n}\right)^{8n} < \frac{1}{\mathrm{e}}$.

Interactive Protocols

Interactive proofs

Consider the problem $x \in^? L$, so x is the input. Prover claims $x \in L$.

Interactive proofs

Consider the problem $x \in^{?} L$, so x is the input. Prover claims $x \in L$.

Prover

a magician

Verifier

an ordinary person
(polynomial-time machine)

message m_1

message m_2

message m_3

message m_4

message m_5

V accepts or rejects this „proof"

Interactive proofs

Consider the problem $x \in^? L$, so x is the input. Prover claims $x \in L$.

$L = \texttt{SAT}$: a nondeterministic machine as a 1-round deterministic "interactive" proof

Prover

a magician

Verifier

an ordinary person
(polynomial-time machine)

message m_1

satisfying assignment

V accepts or rejects this "proof"

Interactive proofs

Consider the problem $x \in^? L$, so x is the input. Prover claims $x \in L$.

Without randomness many-round proofs have the same power (**NP**):

Prover

a magician

Verifier

an ordinary person
(polynomial - time machine)

message m_1 →

← message m_2 $\quad m_2 = f(x, m_1)$

message m_3 →

← message m_4 $\quad m_4 = f(x, m_1, m_2, m_3, m_4)$

message m_5 →

V accepts or rejects this „proof"

Interactive proofs

Consider the problem $x \in^? L$, so x is the input. Prover claims $x \in L$.

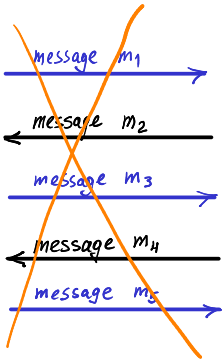
Without randomness many-round proofs have the same power (**NP**):

Prover

a magician

Verifier

an ordinary person
(polynomial - time machine)



$m_2 = f(x, m_1)$

$m_4 = f(x, m_1, m_2, m_3, m_4)$

Prover can send
$(m_1, m_2, \ldots, m_{last})$
at once

V accepts or rejects this „proof"

Interactive proofs. Private coin: **IP**

Consider the problem $x \in^? L$, so x is the input. Prover claims $x \in L$.

Prover

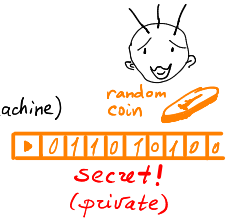
a magician

Verifier

an ordinary person

(polynomial-time machine)

randomized



message m_1 →

← message m_2

message m_3 →

← message m_4

message m_5 →

V accepts or rejects this "proof"

Interactive proofs. Private coin: **IP**

Denote (random variable!) $\mathrm{Out}\langle P, V\rangle(x)$ = the answer V gives after interacting with P on x

Definition

$L \in \mathbf{IP}[k] \iff$ there is a k-round protocol with randomized poly-time algorithm V s.t. $\forall x$

(Completeness) $x \in L \Rightarrow \exists P$ s.t. $\Pr\{\mathrm{Out}\langle P, V\rangle(x) = 1\} \geqslant \frac{3}{4}$ (some P convinces V).

(Soundness) $x \notin L \Rightarrow \forall P'$ only $\Pr\{\mathrm{Out}\langle P', V\rangle(x) = 1\} \leqslant \frac{1}{4}$ (nobody convinces V).

$$\mathbf{IP} = \bigcup_{c \in \mathbb{N}} \mathbf{IP}[n^c]$$

Interactive proofs. Private coin: **IP**

Denote (random variable!) $\mathrm{Out}\langle P, V\rangle(x)$ = the answer V gives after interacting with P on x

Definition

$L \in \mathbf{IP}[k] \iff$ there is a k-round protocol with randomized poly-time algorithm V s.t. $\forall x$

(Completeness) $x \in L \Rightarrow \exists P$ s.t. $\Pr\{\mathrm{Out}\langle P, V\rangle(x) = 1\} \geqslant \frac{3}{4}$ (some P convinces V).

(Soundness) $x \notin L \Rightarrow \forall P'$ only $\Pr\{\mathrm{Out}\langle P', V\rangle(x) = 1\} \leqslant \frac{1}{4}$ (nobody convinces V).

$$\mathbf{IP} = \bigcup_{c \in \mathbb{N}} \mathbf{IP}[n^c]$$

Example: **IP**-protocol for `Graph Nonisomorphism`.

Input: Graphs $G_0 = (U, E_1)$, $G_1 = (U, E_2)$.

Verifier

π := (secret) random permutation of U.
b := (secret) random bit.
V sends $\boldsymbol{\pi(G_b)}$.

(*** Now if $G_0 \approx G_1$, then $\pi(G_0)$ is distributed as $\pi(G_1)$ ***)

Prover

P sends $\boldsymbol{b'}$ (it has no chance to guess it with prob. $> \frac{1}{2}$ if $G_0 \approx G_1$).

Verifier

Accepts iff $\boldsymbol{b = b'}$.

Probabilities: 1 for $\not\approx$, $\leqslant \frac{1}{2}$ for $\approx$ (can decrease by repeating)

Relativization

Theorem (Baker–Gill–Solovay)

There is an oracle A such that $\mathbf{P}^A = \mathbf{NP}^A$.
There is an oracle B such that $\mathbf{P}^B \neq \mathbf{NP}^B$.

Thus it is impossible to resolve $\mathbf{P} =^? \mathbf{NP}$ using a "relativizable" proof.
(A proof that remains valid in every world where someone has access to some oracle.)

Relativization

Theorem (Baker–Gill–Solovay)

There is an oracle A such that $\mathbf{P}^A = \mathbf{NP}^A$.
There is an oracle B such that $\mathbf{P}^B \neq \mathbf{NP}^B$.

Thus it is impossible to resolve $\mathbf{P} =^? \mathbf{NP}$ using a "relativizable" proof.
(A proof that remains valid in every world where someone has access to some oracle.)

Oracle A: One can take $A = \mathtt{QBF}$, because $\mathbf{P}^{\mathtt{QBF}} = \mathbf{NP}^{\mathtt{QBF}} = \mathbf{PSPACE}$.
Or consider $A' = \mathtt{EXPCOMP} := \{\langle M, x, n\rangle : \text{ DTM } M \text{ accepts } x \text{ in } \leqslant n \text{ steps}\}$. (Exercise)

Relativization

Theorem (Baker–Gill–Solovay)

There is an oracle A such that $\mathbf{P}^A = \mathbf{NP}^A$.
There is an oracle B such that $\mathbf{P}^B \neq \mathbf{NP}^B$.

Thus it is impossible to resolve $\mathbf{P} =^? \mathbf{NP}$ using a "relativizable" proof.
(A proof that remains valid in every world where someone has access to some oracle.)

Oracle A: One can take $A = \mathtt{QBF}$, because $\mathbf{P}^{\mathtt{QBF}} = \mathbf{NP}^{\mathtt{QBF}} = \mathbf{PSPACE}$.
Or consider $A' = \mathtt{EXPCOMP} := \{\langle M, x, n\rangle : \text{ DTM } M \text{ accepts } x \text{ in } \leqslant n \text{ steps}\}$. (Exercise)

Oracle B: Will prove if there is time

Exercises

What if...

1. If $\mathbf{NP} \subseteq \mathbf{BPP}$, then ?

Exercises

What if...

1. If $\mathbf{NP} \subseteq \mathbf{BPP}$, then

Then $\mathbf{NP}^{\mathbf{NP}} \subseteq \mathbf{NP}^{\mathbf{BPP}}$.

Exercises

What if...

1. If $\mathbf{NP} \subseteq \mathbf{BPP}$, then

Then $\mathbf{NP}^{\mathbf{NP}} \subseteq \mathbf{NP}^{\mathbf{BPP}}$.

Stop here! One cannot simply replace the lower class, no one said $\mathbf{NP}^{O} \subseteq \mathbf{BPP}^{O}$ for every oracle.

Exercises

What if...

1. If $\mathbf{NP} \subseteq \mathbf{BPP}$, then

Then $\mathbf{NP}^{\mathbf{NP}} \subseteq \mathbf{NP}^{\mathbf{BPP}}$.

Stop here! One cannot simply replace the lower class, no one said $\mathbf{NP}^O \subseteq \mathbf{BPP}^O$ for every oracle.

Still one can prove that under this assumption $\mathbf{PH} = \mathbf{BPP}$.
Hint. Recall the proof of $\mathbf{BPP}^{\mathbf{BPP}} = \mathbf{BPP}$: works also for computing a function (not a language).

Complete this proof!

Exercises

What if...

1. If $\mathbf{NP} \subseteq \mathbf{BPP}$, then

Then $\mathbf{NP}^{\mathbf{NP}} \subseteq \mathbf{NP}^{\mathbf{BPP}}$.

Stop here! One cannot simply replace the lower class, no one said $\mathbf{NP}^O \subseteq \mathbf{BPP}^O$ for every oracle.

Still one can prove that under this assumption $\mathbf{PH} = \mathbf{BPP}$.
Hint. Recall the proof of $\mathbf{BPP}^{\mathbf{BPP}} = \mathbf{BPP}$: works also for computing a function (not a language).

Complete this proof!

2. If $\mathbf{BPP} \subseteq \mathbf{NP}$, then

Exercises

What if...

1. If $\mathbf{NP} \subseteq \mathbf{BPP}$, then

Then $\mathbf{NP}^{\mathbf{NP}} \subseteq \mathbf{NP}^{\mathbf{BPP}}$.

Stop here! One cannot simply replace the lower class, no one said $\mathbf{NP}^O \subseteq \mathbf{BPP}^O$ for every oracle.

Still one can prove that under this assumption $\mathbf{PH} = \mathbf{BPP}$.
Hint. Recall the proof of $\mathbf{BPP}^{\mathbf{BPP}} = \mathbf{BPP}$: works also for computing a function (not a language).

Complete this proof!

2. If $\mathbf{BPP} \subseteq \mathbf{NP}$, then

Then by symmetry $\mathbf{BPP} \subseteq \mathbf{NP} \cap \mathbf{co\text{-}NP}$. That's it. Some people believe even in $\mathbf{P} = \mathbf{BPP}$.

Moral: If you are trying to replace one class by another one, start it from the oracle.