# Computational Complexity; slides 5, HT 2023
# Randomisation and complexity

Paul W. Goldberg (Dept. of CS, Oxford)

HT 2023

# Randomised Algorithms

Randomised algorithms have access to a stream of random bits.

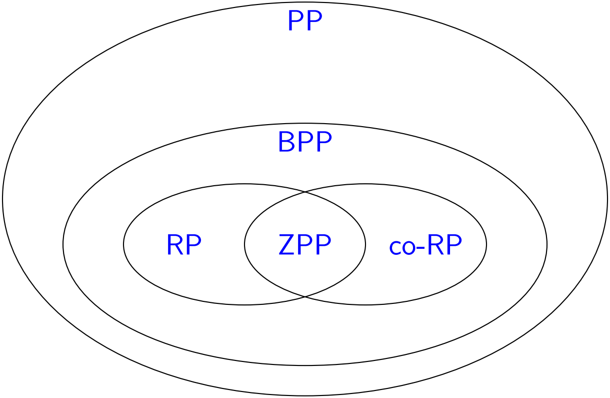
The running time and even the outcome may depend on random choices.

We may allow randomised algorithms to

- produce the wrong result, but only with small probability.
- take more than polynomially many steps, but "not too often"

  $\rightsquigarrow$ expected running time is polynomial.

# Some randomised classes




ZPP: "Las Vegas algorithms"; contains P. Poly *expected* time
RP: one-sided error; no-instance$\mapsto$"no", yes-instance$\mapsto$"yes" with probability$\geq p$ (for some constant $p > 0$)
PP: "majority-P", contains NP, within PSPACE
BPP: allow error either way (constant probability $< \frac{1}{2}$)

# Usage of randomised algorithms

In practice, not so much for language recognition, more for simulation, crypto, stats/ML, or sampling for probability from probability distributions of interest

search for approximate average via sampling

Find median element of list $\{a_1, \ldots, a_n\}$: To find $k$-th highest element, randomly select "pivot" element and find $k'$-th highest element of sublist (for suitable $k'$)

Miller-Rabin test for primality, subsequently superseded by 2002 AKS primality test (deterministic)

- given prime number as input, says "prime"
- Given composite number as input, with prob. $1/4$ says "prime" (correct with prob. $3/4$).

One-sided error; co-RP. Run it $k$ times, say "composite" if we ever get that result, else "prime". Error prob is only $(1/4)^k$.

# Language recognition problem where randomisation seems to help

Polynomial identity testing:

E.g. $(x^2 + y)(x^2 - y) \equiv x^4 - y^2$
where $\equiv$ means equality holds for $x$, $y \in \mathbb{N}$.

In general, if we have many variables, no known deterministic and efficient algorithm, but notice you can try plugging in random $x$, $y$ and checking for equality: if we find answer is "no" we are done; moreover it turns out that for all no-instances you have good chance of verifying that.

works for arithmetic circuits; consider question $p(x_1, \ldots, x_n) \equiv 0$ for circuit with $n$ inputs, 1 output, gates are $+, -, \times$.

# Randomised Complexity Classes

RP⊆NP: accepting computation of an RP machine is a certificate of yes-instance.

It's unknown whether BPP⊆NP, but we argue that BPP represents problems that are in a sense solvable in practice (we expect NP-complete problems to lie outside BPP).

PP (Gill, 1977):
Languages recognised by a probabilistic TM for which yes-instances are accepted with prob. $> \frac{1}{2}$; no-instance with prob. $\leq \frac{1}{2}$.

- PP contains BPP (almost follows directly from the definitions)
- It also contains NP: we can make a PP algorithm that solves SAT. (consider $X \vee \varphi$ where $\varphi$ is a SAT-instance)
- PP is a subset of PSPACE.

# Probability amplification

BPP: problems that can be solved by a randomised algorithm

- with polynomial worst-case running time
- which has an error probability of $\varepsilon < \frac{1}{2}$.

For RP, easy to see how we can improve error probability of algorithm (and evaluate the improvement):
RP: one-sided error; no-instance$\mapsto$"no", yes-instance$\mapsto$"yes" with probability$\geq p$ (for some constant $p > 0$)

For problem $X$ with RP algorithm having (say) $p = 10^{-6}$, run the algorithm $10^6$ times, finally output "yes" iff we see at least one "yes" output. Error probability goes down to $< \frac{1}{2}$!

co-RP algorithm: similar trick, output "no" iff we see at least one "no"

# Probability Amplification

***Corollary*** for RP algorithms:

Suppose $\mathcal{A}$ solves problem $X$ in polynomial time $p(n)$ and the probability that a yes-instance gives answer "yes" is only $1/p'(n)$ ($p'$ a polynomial), and no-instances always give answer "no". Then $X \in$RP.

***Corollary*** for RP algorithms:

Suppose $\mathcal{A}$ solves problem $X$ in polynomial time $p(n)$ and the probability that a yes-instance gives answer "yes" is only $1/p'(n)$ ($p'$ a polynomial), and no-instances always give answer "no". Then $X \in$RP.

***Warm-up for BPP:*** BPP algorithm with error prob $\frac{1}{2} - \delta$: suppose we run it 3 times and take majority vote.

$\Pr[error] = (\frac{1}{2} - \delta)^3 + 3(\frac{1}{2} - \delta)^2(\frac{1}{2} + \delta)$
$= (\frac{1}{2} - \delta)^2(\frac{1}{2} - \delta + \frac{3}{2} + 3\delta) = (\frac{1}{4} - \delta + \delta^2)(2 + 2\delta) = \frac{1}{2} - \frac{3}{2}\delta + 2\delta^3$

***Theorem.*** If a problem can be solved by a BPP algorithm $\mathcal{A}$

- with polynomial worst-case running time
- which has an error probability of $0 < \varepsilon < \frac{1}{2}$.

then it can also be solved by a poly-time randomised algorithm with error probability $2^{-p(n)}$ for any fixed polynomial $p(n)$.

# Probability Amplification

***Proof.***

Algorithm $\mathcal{B}$: On input $w$ of length $n$,

1. Calculate number $k$ (to be determined; details to follow)
2. Run $2k$ independent simulations of $\mathcal{A}$ on input $w$
3. accept if more calls to the algorithm accept than reject.

# Probability Amplification

$S := a_1, \ldots, a_{2k}$: sequence of results obtained by running $\mathcal{A}$ $2k$ times.

Suppose $c$ of these are correct and $i = 2k - c$ are incorrect.

$S$ is a bad sequence if $c \leq i$ so that $\mathcal{B}$ gives the wrong answer.

The probability $p_S$ for any individual bad sequence $S$ to occur is

$$p_S \leq \varepsilon^i (1-\varepsilon)^c \quad \leq \quad \varepsilon^k (1-\varepsilon)^k$$

# Probability Amplification

$S := a_1, \ldots, a_{2k}$: sequence of results obtained by running $\mathcal{A}$ $2k$ times.

Suppose $c$ of these are correct and $i = 2k - c$ are incorrect.

$S$ is a bad sequence if $c \leq i$ so that $\mathcal{B}$ gives the wrong answer.

The probability $p_S$ for any individual bad sequence $S$ to occur is

$$p_S \leq \varepsilon^i (1-\varepsilon)^c \quad \leq \quad \varepsilon^k (1-\varepsilon)^k$$

Hence: $\Pr[\mathcal{B}$ gives wrong result on input $w$ $] =$

$$\sum_{S \text{ bad}} p_S \quad \leq \quad 2^{2k} \cdot \varepsilon^k (1-\varepsilon)^k \quad = \quad (4\varepsilon(1-\varepsilon))^k$$

As $\varepsilon < \frac{1}{2}$ we get $4\varepsilon(1-\varepsilon) < 1$. Hence, to obtain probability $2^{-p(n)}$ we let

$$\alpha = -\log_2(4\varepsilon(1-\varepsilon)) \text{ and choose } k \geq p(n)/\alpha. \qquad \square$$

# General note

So, every problem that can be solved with error probability $\varepsilon < \frac{1}{2}$ can be solved with error probability $< 2^{-p(n)}$.

...practically useful?

# General note

So, every problem that can be solved with error probability $\varepsilon < \frac{1}{2}$ can be solved with error probability $< 2^{-p(n)}$.

...practically useful?

Arguably yes:

- the probability that an algorithm with error probability of $2^{-100}$ has bad luck with the coin tosses is much smaller than the chance that any algorithm fails due to
  - hardware failures,
  - random bit mutations in the memory
  - ...

# Hoeffding's inequality

Consider a (biased) coin that comes up heads with probability $p$. So, if we toss it $n$ times, should get $p.n$ heads on average. Letting random variable $H(n)$ be number of heads seen after $n$ coin tosses, it turns out that

$$\Pr[H(n) \leq (p - \varepsilon)n] \leq \exp(-2\varepsilon^2 n)$$

and similarly,

$$\Pr[H(n) \geq (p + \varepsilon)n] \leq \exp(-2\varepsilon^2 n)$$

Probability that we're off by a constant factor, is inverse-exponential in $n$. Often useful in analysing randomised algorithms!

# Relationships to other complexity classes

Recall we noted that RP$\subseteq$NP.
(convert a randomised algorithm to a non-deterministic one by replacing coin flips with non-deterministic guesses.)

Doesn't work for BPP.

We do have BPP$\subseteq \Sigma_2^P \cap \Pi_2^P$ (Sipser-Gács-Lautemann theorem)
Consequently, if P=NP, it would follow that P=BPP since if P=NP, the polynomial hierarchy collapses to P.

We also know: BPP$\subseteq$P/poly (Adleman's theorem).
"Any BPP language has polynomial-size circuits."

**Next:** A randomised algorithms for reducing a (satisfiable) SAT instance to one having a unique solution

Then, a quick look at probabilistically checkable proofs

# Reducing SAT to USAT with the aid of randomness

We give another example of a task where randomisation seems to be useful.
Also, interesting technique; illustration of probabilistic reasoning.

USAT: given a formula $\varphi$ with at most 1 satisfying assignment, determine whether it is satisfiable. (U stands for "unique")

So, USAT is no harder than SAT, and in a sense it's also no easier.

Afterwards: a quick look at interactive proofs, another setting where randomisation is important

# Reducing SAT to USAT with the aid of randomness

We reduce SAT to USAT.

Motivation: known algorithms for SAT take time $poly(n)2^n$. The "strong exponential time hypothesis" asserts that you *need* time proportional to $2^n$.[1]
But: note Grover's algorithm, a quantum algorithm solving USAT in time $poly(n)2^{n/2}$. Reducing SAT to USAT means that on a quantum machine, SAT is also solved in time $poly(n)2^{n/2}$!

[1](non-strong) ETH: for 3SAT, $2^{kn}$ needed for some $k > 0$

# Reducing SAT to USAT with the aid of randomness

We reduce SAT to USAT.

Motivation: known algorithms for SAT take time $poly(n)2^n$. The "strong exponential time hypothesis" asserts that you *need* time proportional to $2^n$.[1]
But: note Grover's algorithm, a quantum algorithm solving USAT in time $poly(n)2^{n/2}$. Reducing SAT to USAT means that on a quantum machine, SAT is also solved in time $poly(n)2^{n/2}$!

**Challenge:** Given $\varphi$, construct $\psi$ such that $\psi$ has a unique satisfying assignment iff $\varphi$ is satisfiable.

[1] (non-strong) ETH: for 3SAT, $2^{kn}$ needed for some $k > 0$

We reduce SAT to USAT.

Motivation: known algorithms for SAT take time $poly(n)2^n$. The "strong exponential time hypothesis" asserts that you *need* time proportional to $2^n$.[1]
But: note Grover's algorithm, a quantum algorithm solving USAT in time $poly(n)2^{n/2}$. Reducing SAT to USAT means that on a quantum machine, SAT is also solved in time $poly(n)2^{n/2}$!

**Challenge:** Given $\varphi$, construct $\psi$ such that $\psi$ has a unique satisfying assignment iff $\varphi$ is satisfiable.

**Idea:** $\psi := \varphi \wedge \rho$, where $\rho$ is some other formula over the same variables.

[1](non-strong) ETH: for 3SAT, $2^{kn}$ needed for some $k > 0$

# Reducing SAT to USAT

**Challenge:** Given $\varphi$, construct $\psi$ such that $\psi$ has a unique satisfying assignment iff $\varphi$ is satisfiable.
**Idea:** $\psi := \varphi \wedge \rho$, where $\rho$ is some other formula over the same variables.
**Extension of the idea:** $\psi_1 := \varphi \wedge \rho_1$, ... ,$\psi_k := \varphi \wedge \rho_k$; look for satisfying assignment of any of these...

**Problem:** Think of $\varphi$ as having been chosen by an opponent. Given a choice of $\rho_1, \ldots, \rho_k$, he can pick a $\varphi$ that fails for your choice. This is where randomness helps!

# Reducing SAT to USAT

**Challenge:** Given $\varphi$, construct $\psi$ such that $\psi$ has a unique satisfying assignment iff $\varphi$ is satisfiable.
**Idea:** $\psi := \varphi \wedge \rho$, where $\rho$ is some other formula over the same variables.
**Extension of the idea:** $\psi_1 := \varphi \wedge \rho_1$, ... ,$\psi_k := \varphi \wedge \rho_k$; look for satisfying assignment of any of these...

**Problem:** Think of $\varphi$ as having been chosen by an opponent. Given a choice of $\rho_1, \ldots, \rho_k$, he can pick a $\varphi$ that fails for your choice. This is where randomness helps!

(random) parity functions: let $x_1, \ldots, x_n$ be the variables of $\varphi$. Let $\pi := \oplus_{x \in R}(x) \oplus b$ where each $x_i$ is added to $R$ with prob. $\frac{1}{2}$, and $b$ is chosen to be $\text{TRUE}/\text{FALSE}$ with equal probability $\frac{1}{2}$.

Think of $R$ as standing for "relevant attributes"

# Reducing SAT to USAT

Q: Why are random parity functions great?
A: Consider $\varphi$ with set $S$ of satisfying assignments. For random p.f. $\pi$, the expected number of satisfying assignments of $\varphi \wedge \pi$ is $\frac{1}{2}|S|$.

To see this, note that any satisfying assignment of $\varphi$ gets eliminated with probability $\frac{1}{2}$.

# Reducing SAT to USAT

Q: Why are random parity functions great?
A: Consider $\varphi$ with set $S$ of satisfying assignments. For random p.f. $\pi$, the expected number of satisfying assignments of $\varphi \wedge \pi$ is $\frac{1}{2}|S|$.

To see this, note that any satisfying assignment of $\varphi$ gets eliminated with probability $\frac{1}{2}$.

**Corollary:** letting $\rho_k := \pi_1 \wedge \ldots \wedge \pi_k$ for independently randomly-chosen $\pi_i$, the expected number of satisfying assignments to $\varphi \wedge \rho_k$ is $|S|/2^k$.

# Reducing SAT to USAT

Q: Why are random parity functions great?
A: Consider $\varphi$ with set $S$ of satisfying assignments. For random p.f. $\pi$, the expected number of satisfying assignments of $\varphi \wedge \pi$ is $\frac{1}{2}|S|$.

To see this, note that any satisfying assignment of $\varphi$ gets eliminated with probability $\frac{1}{2}$.

**Corollary:** letting $\rho_k := \pi_1 \wedge \ldots \wedge \pi_k$ for independently randomly-chosen $\pi_i$, the expected number of satisfying assignments to $\varphi \wedge \rho_k$ is $|S|/2^k$.

This suggests the following approach:

- Generate $\rho_k$ as above, for each $k = 1, 2, \ldots, n+1$.
- Search for a satisfying assignment to $\varphi \wedge \rho_k$.

Need to argue that for $k \approx \log_2 |S|$, we have reasonable chance of producing a formula with a *unique* s.a.

# Pairwise independence of random p.f's:

Given $x \neq x' \in S$, and a random parity function $\pi$, we have:

$\Pr[x \text{ satisfies } \pi] = \frac{1}{2}$ $\qquad$ $\Pr[x' \text{ satisfies } \pi] = \frac{1}{2}$

In addition:

$\Pr[x \text{ satisfies } \pi | x' \text{ satisfies } \pi] = \frac{1}{2}$

**Proof:**

For any $x$, $\pi(x) = v.x$ (or, $\neg v.x$) where $v$ is characteristic vector of relevant attributes $R$ of $\pi$.

($v.x$ denotes sum (XOR) of entries of $x$ where corresponding entry of $v$ is 1)

Let $i$ be a bit position where $x'_i = 1$ and $x_i = 0$. $i$ gets added to $R$ with probability $\frac{1}{2}$, so value of $\pi(x')$ gets flipped with probability $\frac{1}{2}$.

similarly for conjunctions of random parity functions

For some $k$, we have $2^{k-2} \leq |S| \leq 2^{k-1}$.

***Lemma:*** $\Pr[\text{there is unique } x \in S \text{ satisfying } \varphi \wedge \rho_k] \geq \frac{1}{8}$ (probability is w.r.t. random choice of $\rho_k$).

***Proof:*** Let $p = 2^{-k}$ be the probability that $x \in S$ satisfies $\rho_k$.
Let $N$ be the random variable consisting of the number of s.a.'s of $\varphi \wedge \rho_k$.
$E[N] = |S|p \in [\frac{1}{4}, \frac{1}{2}]$.

$$\Pr[N \geq 1] \geq \sum_{x \in S} \Pr[x \models \rho_k] - \sum_{x < x' \in S} \Pr[x \models \rho_k \wedge x' \models \rho_k] = |S|p - \binom{|S|}{2} p^2$$

By pairwise independence and union bound, we have $\Pr[N \geq 2] \leq \binom{|S|}{2} p^2$. So

$$\Pr[N = 1] = \Pr[N \geq 1] - \Pr[N \geq 2] \geq |S|p - 2\binom{|S|}{2} p^2 \geq |S|p - |S|^2 p^2 \geq \frac{1}{8}.$$

(where the last inequality uses $\frac{1}{4} \leq |S|p \leq \frac{1}{2}$.)

# Interactive proofs

- an important application of randomisation in context of computational complexity

NP problems as "one-round interrogation":

skeptic: show me a solution
prover: $\langle$solution$\rangle$

skeptic can easily *check* prover's solution.
prover is "all-powerful".

A problem $\mathcal{X}$ is in NP if there's a poly-time TM (the skeptic), and a function (the prover) that can convince the skeptic...

Can an extension of above protocol "capture" other complexity classes?

# Interactive proofs

- General idea: multi-round interaction

c.f. mathematician with new theorem, tries to convince colleagues...

***Idea for definition:*** A problem belongs to IP if there's a communication protocol with a function $\mathcal{P}$ (the prover) and a poly-time computable function $\mathcal{V}$ (the verifier) such that:

- for problem-instance $\mathcal{I}$ of size $n$, allow poly($n$) rounds of interaction (sequence of questions/challenges). Let's limit messages to polynomial length.
- $\mathcal{P}$ and $\mathcal{V}$'s messages may depend on previous interaction
- $\mathcal{V}$ ends up accepting iff $\mathcal{I}$ is a yes-instance...

# Interactive proofs

- General idea: multi-round interaction

c.f. mathematician with new theorem, tries to convince colleagues...

***Idea for definition:*** A problem belongs to IP if there's a communication protocol with a function $\mathcal{P}$ (the prover) and a poly-time computable function $\mathcal{V}$ (the verifier) such that:

- for problem-instance $\mathcal{I}$ of size $n$, allow poly($n$) rounds of interaction (sequence of questions/challenges). Let's limit messages to polynomial length.
- $\mathcal{P}$ and $\mathcal{V}$'s messages may depend on previous interaction
- $\mathcal{V}$ ends up accepting iff $\mathcal{I}$ is a yes-instance...

But: consider *deterministic* verifier. Prover can supply all answers "upfront": no need to interact.

# The Complexity Class IP

***Definition.*** A decision problem $\mathcal{L}$ belongs to the complexity class IP if there is

- a communication protocol $\mathcal{C}$ and
- a randomised polynomial-time bounded algorithm $\mathcal{V}$ (the verifier)

with the property that

1. there is a function $\mathcal{P}$ (the prover) such that if $w \in \mathcal{L}$

$$\Pr[\ \mathcal{P} \text{ persuades } \mathcal{V} \text{ to accept } w] \geq \frac{2}{3}$$

2. for all "prover" functions $\mathcal{P}'$, if $w \notin \mathcal{L}$

$$\Pr[\ \mathcal{P}' \text{ persuades } \mathcal{V} \text{ to accept } w] \leq \frac{1}{3}$$

$\mathcal{L}$ belongs to IP$[k]$ if at most $k$ communication rounds are necessary.

# Graph-Non-Isomorphism in IP

***Recall.*** An isomorphism between two graphs $H$ and $G$ is a function $f : V(H) \to V(G)$ such that

1. $f$ is a bijection between $V(H)$ and $V(G)$ and
2. for all $u, v \in V(H)$: $\{u, v\} \in E(H) \iff \{f(v), f(u)\} \in E(G)$.

Graph isomorphism has no known poly-time algorithm

Graph isomorphism is easily seen to be in NP but unlikely to be NP-complete, has subexponential algorithm

It's also known that if GI is NP-complete, then $\Sigma_2^P = \Pi_2^P$, thus PH collapses

# Graph-Non-Isomorphism in IP

(c.f. coke vs pepsi taste test)

Input. Graphs $G_1$ and $G_2$.

Communication.

1. $\mathcal{V}$ randomly chooses $i \in \{1, 2\}$, randomly permutes vertices of $G_i$ to obtain new graph $H$ isomorphic to $G_i$.
2. $\mathcal{V}$ sends $H$ to $\mathcal{P}$
3. $\mathcal{P}$ identifies the graph $G_j$ to which $H$ is isomorphic, and sends $j$ back.
4. $\mathcal{V}$ accepts if $i = j$.

Repeat (in parallel or sequentially) until $\mathcal{V}$ "reasonably convinced".

# Graph-Non-Isomorphism in IP

(c.f. coke vs pepsi taste test)

Input. Graphs $G_1$ and $G_2$.

Communication.

1. $\mathcal{V}$ randomly chooses $i \in \{1, 2\}$, randomly permutes vertices of $G_i$ to obtain new graph $H$ isomorphic to $G_i$.
2. $\mathcal{V}$ sends $H$ to $\mathcal{P}$
3. $\mathcal{P}$ identifies the graph $G_j$ to which $H$ is isomorphic, and sends $j$ back.
4. $\mathcal{V}$ accepts if $i = j$.

Repeat (in parallel or sequentially) until $\mathcal{V}$ "reasonably convinced".

***Theorem.*** IP = PSPACE

(See Sipser, Theorem 10.29)
Arora/Barak: IP=PSPACE (Chapter 8.3)

# Zero-Knowledge Proofs

***Applications.***

1. Secure authentication. convince someone you know some password etc without revealing it
2. Auctions.
   - Several companies place bids for items/frequencies/mining rights ...
   - They place their bids simultaneously.
   - After the bidding process, each company wants to be convinced that the winner really bid more than itself.
   - The winner doesn't want to reveal their bid.

Next: graph isomorphism. Standard IP has prover reveal the isomorphism: let's disallow that!

# A Zero-Knowledge Proof for Graph Isomorphism

**Given:** Two graphs $G_1, G_2$

***Prover's secret:*** An isomorphism $\pi$ between $G_1, G_2$

Prover wants to prove to Verifier that $G_1 \cong G_2$ without revealing $\pi$.

# A Zero-Knowledge Proof for Graph Isomorphism

**Given:** Two graphs $G_1, G_2$

***Prover's secret:*** An isomorphism $\pi$ between $G_1, G_2$

Prover wants to prove to Verifier that $G_1 \cong G_2$ without revealing $\pi$.

***Communication protocol.***

1. $\mathcal{P}$ randomly selects $i \in \{1, 2\}$ and computes a random permutation of $|V(G_i)|$ generating a graph $H \cong G_i$
2. $\mathcal{P}$ sends $H$ to $\mathcal{V}$ and keeps the isomorphism $f : H \cong G_i$.
3. $\mathcal{V}$ randomly selects $j \in \{1, 2\}$ and sends $j$ back to $\mathcal{P}$.
4. $\mathcal{P}$ computes an isomorphism $\pi_j$ (either $f$ or $\pi \circ f$) between $G_j$ and $H$, and sends it to $\mathcal{V}$.
5. $\mathcal{V}$ accepts if $H = \pi_j(G_j)$

# A Zero-Knowledge Proof for Graph Isomorphism

**Given:** Two graphs $G_1, G_2$

***Prover's secret:*** An isomorphism $\pi$ between $G_1, G_2$

Prover wants to prove to Verifier that $G_1 \cong G_2$ without revealing $\pi$.

***Communication protocol.***

1. $\mathcal{P}$ randomly selects $i \in \{1, 2\}$ and computes a random permutation of $|V(G_i)|$ generating a graph $H \cong G_i$
2. $\mathcal{P}$ sends $H$ to $\mathcal{V}$ and keeps the isomorphism $f : H \cong G_i$.
3. $\mathcal{V}$ randomly selects $j \in \{1, 2\}$ and sends $j$ back to $\mathcal{P}$.
4. $\mathcal{P}$ computes an isomorphism $\pi_j$ (either $f$ or $\pi \circ f$) between $G_j$ and $H$, and sends it to $\mathcal{V}$.
5. $\mathcal{V}$ accepts if $H = \pi_j(G_j)$

- If $G_1 \cong G_2$ then $\mathcal{P}$ can always convince $\mathcal{V}$.
- Otherwise, $\mathcal{P}$ fails with probability $\frac{1}{2}$, which again can be amplified.
- The computation can be done efficiently.