

I5-25 I

Great Theoretical Ideas in Computer Science

Proofs: A computational lens

April 23, 2015

Proof. Define f_{ij} as in (5). As f is symmetric, we only need to consider f_{12} .

$$\begin{aligned} \mathbf{E} [f_{12}^2] &= \mathbf{E}_{x_3, \dots, x_n} \left[\frac{1}{4} \cdot (f_{12}^2(00x_3 \dots x_n) + f_{12}^2(01x_3 \dots x_n) + f_{12}^2(10x_3 \dots x_n) + f_{12}^2(11x_3 \dots x_n)) \right] \\ &= \frac{1}{4} \mathbf{E}_{x_3, \dots, x_n} \left[(f(00x_3 \dots x_n) - f(11x_3 \dots x_n))^2 + (f(11x_3 \dots x_n) - f(00x_3 \dots x_n))^2 \right] \\ &\geq \frac{1}{2} \left(\binom{n-2}{r_0-1} \cdot 2^{-(n-2)} \cdot 4 + \binom{n-2}{n-r_1-1} \cdot 2^{-(n-2)} \cdot 4 \right) \\ &= 8 \cdot \left(\frac{(n-r_0+1)(n-r_0)}{n(n-1)} \cdot \binom{n}{r_0-1} + \frac{(n-r_1+1)(n-r_1)}{n(n-1)} \cdot \binom{n}{r_1-1} \right) 2^{-n}. \end{aligned}$$

Inequality (6) follows by applying Lemma 2.2.

In order to establish inequality (7), we show a lower bound on the principal Fourier coefficient of f :

$$\hat{f}(\emptyset) \geq 1 - 2 \left(\sum_{s < r_0} \binom{n}{s} + \sum_{s > n-r_1} \binom{n}{s} \right) 2^{-n},$$

which implies that

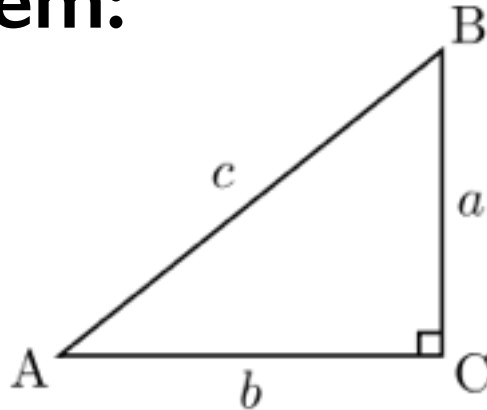
$$\hat{f}(\emptyset)^2 \geq 1 - 4 \cdot \left(\sum_{s < r_0} \binom{n}{s} + \sum_{s < r_1} \binom{n}{s} \right) 2^{-n}.$$

□



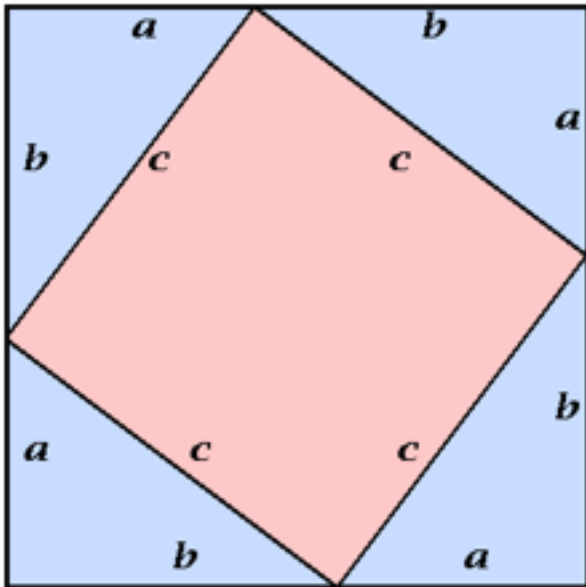
First there was GORM

Pythagoras's Theorem:



$$a^2 + b^2 = c^2$$

Proof:



$$(a + b)^2 = a^2 + 2ab + b^2$$

Looks legit.



Then there was Russell



Russell and others worked on formalizing GORM proofs.

Principia Mathematica Volume 2

86 CARDINAL ARITHMETIC [PART III]

*110·632. $\vdash : \mu \in NC . \supset . \mu +_c 1 = \hat{\xi} \{ (\exists y) . y \in \xi . \xi - t'y \in sm''\mu \}$
Dem.
 $\vdash . *110·631 . *51·211·22 . \supset$
 $\vdash : Hp . \supset . \mu +_c 1 = \hat{\xi} \{ (\exists \gamma, y) . \gamma \in sm''\mu . y \in \xi . \gamma = \xi - t'y \}$
[*13·195] $= \hat{\xi} \{ (\exists y) . y \in \xi . \xi - t'y \in sm''\mu \} : \supset \vdash . Prop$

*110·64. $\vdash . 0 +_c 0 = 0$ [*110·62]
*110·641. $\vdash . 1 +_c 0 = 0 +_c 1 = 1$ [*110·51·61 . *101·2]
*110·642. $\vdash . 2 +_c 0 = 0 +_c 2 = 2$ [*110·51·61 . *101·31]

***110·643. $\vdash . 1 +_c 1 = 2$**
Dem.
 $\vdash . *110·632 . *101·21·28 . \supset$
 $\vdash . 1 +_c 1 = \hat{\xi} \{ (\exists y) . y \in \xi . \xi - t'y \in 1 \}$
[*54·3] $= 2 . \supset \vdash . Prop$

The above proposition is occasionally useful. It is used at least three times, in *113·66 and *120·123·472.

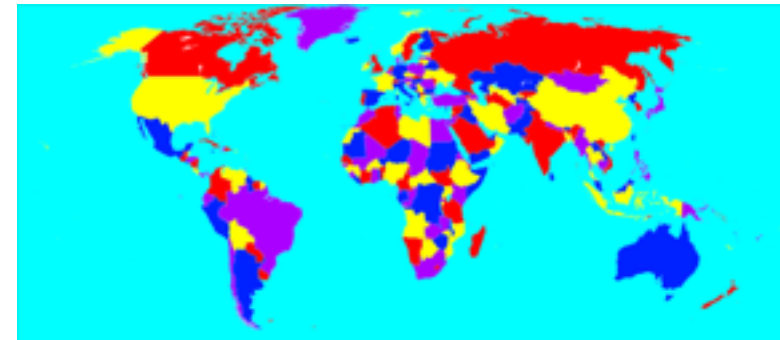
This meant proofs could be found mechanically.
And could be verified mechanically.

Then there were computers

All this played a key role in the birth of computer science.

Computers themselves can find proofs.
(automated theorem provers)

Computers can help us find proofs
(e.g. 4-Color Theorem)



Are these really proofs?

And now...

Thanks to computer science, a “proof” can be:

- randomized
- interactive
- zero-knowledge
- spot-checkable

The original goal of a “proof” was to explain and understand a truth.

Now, sometimes the goal is that it doesn't explain anything!

Review of NP

Definition:

A language A is in NP if

- there is a polynomial time TM V
- a polynomial p

such that for all x :

$$x \in A \iff \exists u \text{ with } |u| \leq p(|x|) \text{ s.t. } V(x, u) = 1$$

“ $x \in A$ iff there is a polynomial length **proof** u that is verifiable by a poly-time algorithm.”

If $x \in A$, there is some **proof** that leads V to **accept**.

If $x \notin A$, every “**proof**” leads V to **reject**.

NP: A game between a Prover and a Verifier

Verifier



poly-time
skeptical

Prover



omniscient
untrustworthy

Given some string x .

Prover wants to convince **Verifier** $x \in A$.

Prover cooks up a proof string u and sends it to **Verifier**.

Verifier, in polynomial time, should be able to tell if the proof is legit.

NP: A game between a Prover and a Verifier

Verifier



poly-time
skeptical

Prover



omniscient
untrustworthy

“Completeness”

If $x \in A$, there must be some proof u that convinces the Verifier.

“Soundness”

If $x \notin A$, no matter what “proof” Prover gives, Verifier should detect the lie.

Limitations of NP

We know many languages are in NP.

SAT, 3SAT, CLIQUE, MAX-CUT, VERTEX-COVER,
SUDOKU, THEOREM-PROVING, 3COL, ...

Anything not known to be in NP ?

Consider 3SAT:

Given a satisfiable formula, there is an easy way
the **Prover** can prove it is satisfiable.

Given an unsatisfiable formula, how can
the **Prover** prove it is unsatisfiable???

i.e. is the complement of 3SAT in NP?

How can we generalize the NP setting?

The NP setting seems too weak for this purpose.

Plus, in real life, people use more general ways of convincing each other of the validity of statements.

- Make the protocol **interactive**.

You can show interaction doesn't really change the model.
i.e., whatever you can do with interaction, you can do with the original setting.

- Make the verifier **probabilistic**.

We don't think randomization by itself adds significant power.

But, magic happens when you combine the two.

Interaction + Randomization

Coke vs Pepsi Challenge



Your friend tells you he can taste the difference between Coke and Pepsi.

How can he convince you of this?

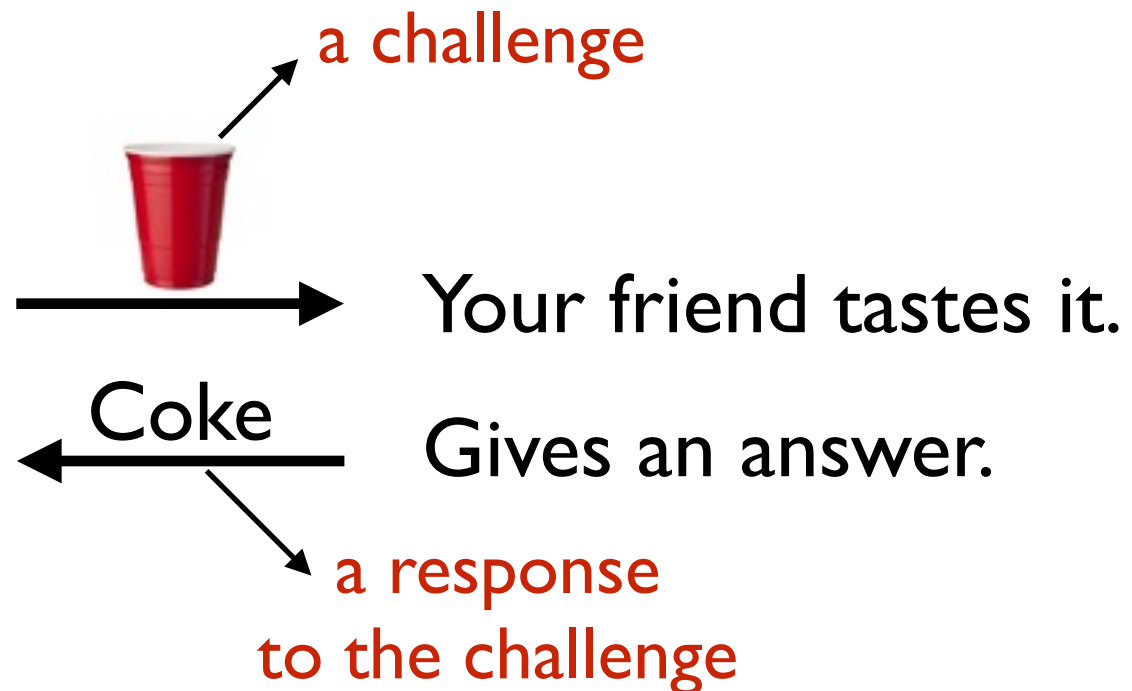
Coke vs Pepsi



Choose Coke or Pepsi
at random.

Send it to your friend.

Repeat

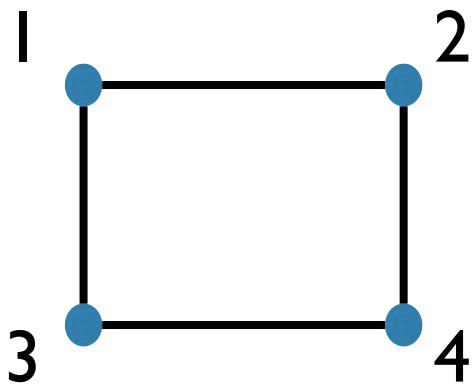


Graph Isomorphism Problem

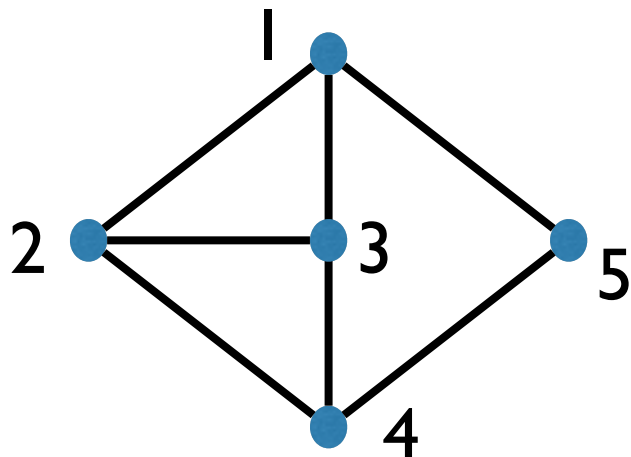
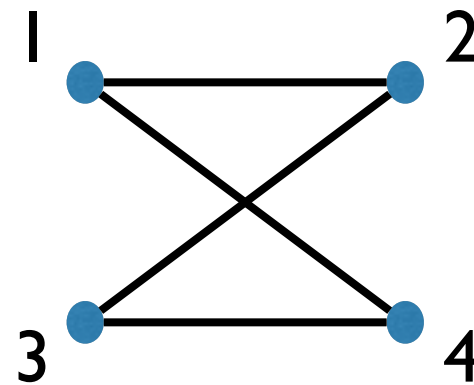
Given two graphs G_1, G_2 , are they isomorphic?

i.e., is there a permutation π of the vertices such that

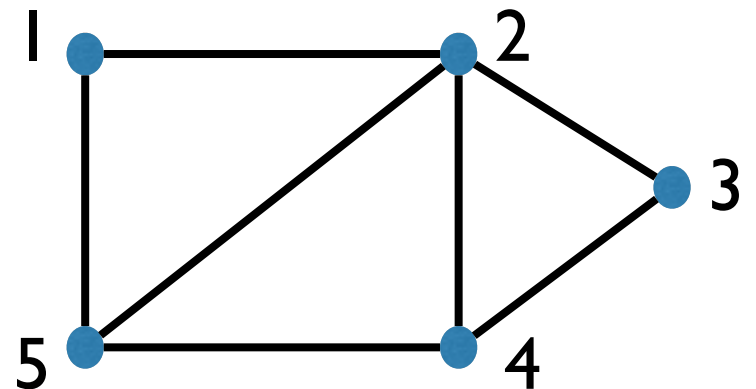
$$\pi(G_1) = G_2$$



=



≠



Graph Isomorphism Problem

Is Graph Isomorphism in NP?

Sure! A good proof is the permutation of the vertices.

Is Graph Non-isomorphism in NP?

No one knows!

But there is a simple randomized interactive proof.

Interactive Proof for Graph Non-isomorphism



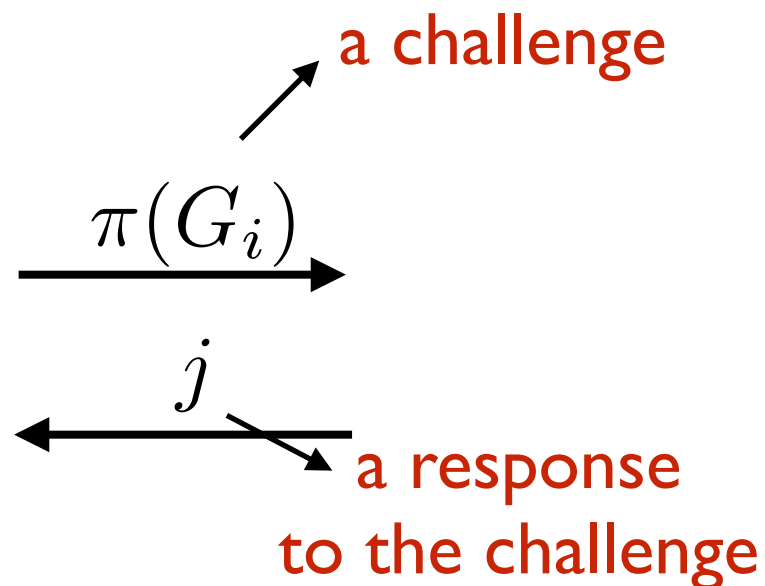
$\langle G_1, G_2 \rangle$



Pick at random $i \in \{1, 2\}$


Choose a permutation π
of vertices at random.

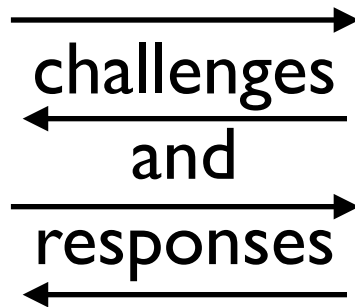
Accept if $i = j$



The complexity class IP

We say that a language A is in IP if:

- there is a probabilistic poly-time **Verifier** 
- there is a computationally unbounded **Prover** 



(poly rounds)

“Completeness”

If $x \in A$, **Verifier** accepts with prob. at least $2/3$.

“Soundness”

If $x \notin A$, **Verifier** accepts with prob. at most $1/3$.

How about complement of 3SAT?

Is $\overline{3SAT}$ in IP?

Yes!

In fact the complement of any language in NP is in IP.

And in fact, many more languages are in IP.

How big is IP?

So how powerful are interactive proofs?

How big is IP?

Theorem:

$$\text{IP} = \text{PSPACE}$$



Adi Shamir

1990

(another application of polynomials)

Chess

An interesting corollary:

Suppose in chess, white can always win in ≤ 300 moves.



How can the wizard prove this to you?

Back to Graph Non-isomorphism



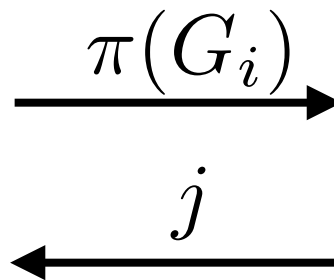
$\langle G_1, G_2 \rangle$



Pick at random $i \in \{1, 2\}$

Choose a permutation π
of vertices at random.

Accept if $i = j$



There is more
to this protocol
than meets the
eye.

Back to Graph Non-isomorphism

Does the verifier gain any insight about why the graphs are not isomorphic?



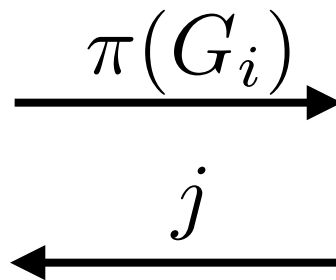
$\langle G_1, G_2 \rangle$



Pick at random $i \in \{1, 2\}$

Choose a permutation π
of vertices at random.

Accept if $i = j$



There is more
to this protocol
than meets the
eye.

Zero-Knowledge Proofs

The **Verifier** is convinced,
but he learns nothing about why the graphs are
not isomorphic!

The **Verifier** could have produced the communication
transcript by himself, with no help from the **Prover**.

A proof with 0 explanatory content!

This can actually be quite useful!

Zero-Knowledge Proofs

I found a truly marvelous proof of Riemann Hypothesis.

I want to convince you that I have a valid proof.

But I don't want you to learn anything about the proof.

Is this possible?

Does every problem in NP have a zero-knowledge IP?

Zero-Knowledge Proofs for NP



Goldreich



Micali



Wigderson

1986

Does every problem in NP have a zero-knowledge IP?

Yup! (under plausible cryptographic assumptions)

And the prover need not be a wizard.

He just needs to know the ordinary proof.

Zero-Knowledge Proofs for NP

Does every problem in NP have a zero-knowledge IP?

Yup! (under plausible cryptographic assumptions)

And the prover need not be a wizard.

He just needs to know the ordinary proof.

It suffices to show this for your favorite NP-complete problem.

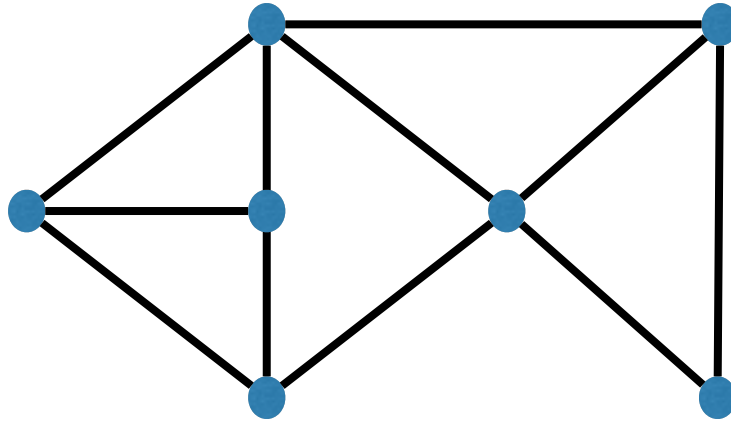
(every problem in NP reduces to an NP-complete prob.)

We'll pick the Hamiltonian cycle problem.

Zero-Knowledge Proofs for NP

Hamiltonian cycle problem

Given an undirected graph:

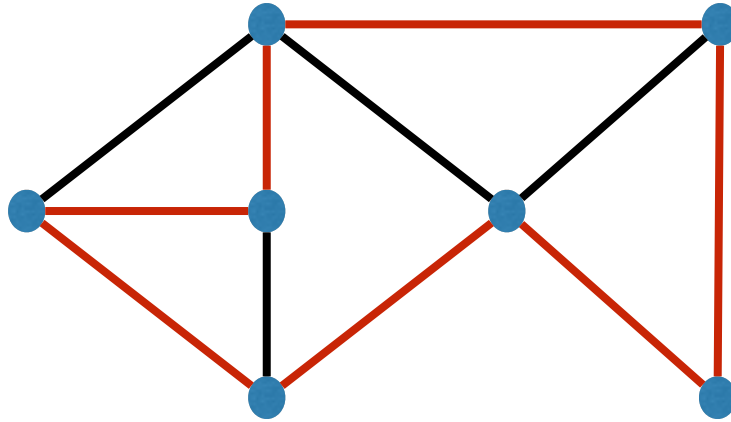


Does it have a cycle that visits every vertex exactly once?

Zero-Knowledge Proofs for NP

Hamiltonian cycle problem

Given an undirected graph:



Does it have a cycle that visits every vertex exactly once?

Zero-Knowledge Proofs for NP

Given undirected graph G

Prover:

Picks randomly a permutation of the vertices π .

Sends $\pi(G)$ in a “locked” way:

- for each pair of vertices, there is a locked bit.
- the bit indicates whether the vertices are connected.

Verifier:

Flips a coin.

If heads, asks **Prover** to show him the Hamiltonian cycle.

If tails, asks **Prover** to unlock everything, and asks for π .

Zero-Knowledge Proofs for NP

Completeness

Soundness

Zero-knowledge

If the “locked” bits work the way they are meant to work, all is good.

- **Verifier** shouldn't be able to unlock them by himself.
- **Prover** shouldn't be able to change bit values.

Can be realized using **bit commitment schemes**.
(assuming Verifier is computationally bounded)

Zero-Knowledge for all?

Does every problem in $IP = PSPACE$ have a zero-knowledge proof?



Ben-Or



Goldreich



Goldwasser



Håstad



Kilian



Micali



Rogaway

1990

"Everything provable is provable in zero-knowledge"

Statistical vs Computational Zero-Knowledge

There is a difference between

- zero-knowledge proof for Graph Non-isomorphism
- zero-knowledge proof for Hamiltonian Cycle

Statistical zero-knowledge:

Verifier wouldn't learn anything even if it was computationally unbounded.

Computational zero-knowledge:

Verifier wouldn't learn anything assuming it cannot unlock the locks in polynomial time.

Statistical vs Computational Zero-Knowledge

SZK = set of all problems with
statistically zero-knowledge proofs

CZK = set of all problems with
computationally zero-knowledge proofs

$IP = PSPACE = CZK$

SZK is believed to be much smaller.

In fact, it is believed that it does not contain
NP-complete problems.

And now...

Thanks to computer science, a “proof” can be:

- randomized
- interactive
- zero-knowledge
- spot-checkable

Spot-Checkable Proofs

There are other surprising facts about proofs!

Suppose I have a proof that $1+1 = 2$.

It is a few hundred pages long.

I give you the proof, and ask you to verify it.

It could be that there is some tiny mistake somewhere in the proof.

Trying to find it is super annoying!

Spot-Checkable Proofs

If only there was a way to just check a few random places of the proof, and be convinced that the proof is correct...

That's a dream too good to be true.

Or is it?

Let's go back to Graph Non-isomorphism.

Can we realize this dream for this problem?

Given two graphs G_0, G_1 , is there a “spot-checkable” proof that they are non-isomorphic?

Spot-Checkable Proofs

Enumerate all possible n -vertex graphs:

$$H_1, H_2, H_3, H_4, H_5, H_6, H_7, \dots, H_N \quad N = 2^{\binom{n}{2}}$$

proof: 0 1 0 0 1 1 0 ... 1

Index i : if $H_i \approx G_0$, put 0.

if $H_i \approx G_1$, put 1.

if neither, put 0 or 1 (doesn't matter).

Verifier:

Pick at random $i \in \{0, 1\}$.

Choose a permutation π of vertices at random.

Figure out the index j corresponding to $\pi(G_i)$.

Check: is the bit at index j equal to i .

Spot-Checkable Proofs

OK, the proof is exponentially long.

Not so useful in that sense.

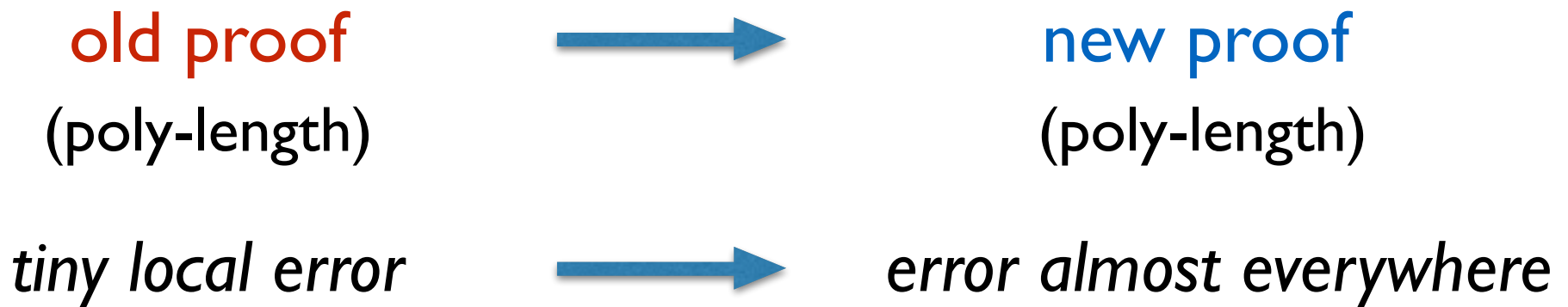
Is there a way to do something similar but with poly-length proof?

Spot-Checkable Proofs

Probabilistically Checkable Proofs (PCP) Theorem:

Every problem in NP admits “spot-checkable” proofs of polynomial length.

The verifier can be convinced with high probability by looking only at a constant number of bits in the proof.



“New shortcut found for long math proofs!”

Spot-Checkable Proofs

Probabilistically Checkable Proofs (PCP) Theorem:

Every problem in NP admits “spot-checkable” proofs of polynomial length.

The verifier can be convinced with high probability by looking only at a constant number of bits in the proof.

1998



Arora



Lund



Motwani



Safra



Sudan



Szegedy

Spot-Checkable Proofs

This theorem is equivalent to:

PCP Theorem (version 2):

There is some constant ϵ such that if there is a polynomial-time ϵ -approximation algorithm for MAX-3SAT then $P=NP$.

(It is NP-hard to approximate MAX-3SAT within an ϵ factor.)

This is called an “*hardness of approximation*” result.

They are hard to prove!

Spot-Checkable Proofs

PCP Theorem is one of the crowning achievements in CS theory!

Proof is a half a semester course.

Blends together:

P/NP

random walks

expander graphs

polynomials / finite fields

error-correcting codes

Fourier analysis

Summary

Computer science gives a whole new perspective on **proofs**:

- can be probabilistic
- can be interactive
- can be zero-knowledge
- can be spot-checkable
- can be quantum mechanical

Summary

old-fashioned proof + deterministic verifier

problems whose solutions can be efficiently verifiable:
NP

randomization + interaction

problems whose solutions can be efficiently verifiable:
PSPACE

PSPACE = Computationally Zero-Knowledge (CZK)

"Everything provable is provable in zero-knowledge"
(some special problems are in SZK)

Summary

PCP Theorem

Old-fashioned proofs can be turned into spot-checkable.
(you only need to check constant number of bits!)

Equivalent to an hardness of approximation result.

Opens the door to many other hardness of approximation results.