Probabilistically Checkable Proofs

Irit Dinur

The Weizmann Institute of Science

September 14, 2010

- Common wisdom: to check a proof you need to read it
- Why bother? instead-
  - Ask for the proof to be supplied in **PCP** format
  - Check randomly by reading only 3 bits.
    Probability of error, i.e. of accepting a bad proof, is at most $1/2$.
    (For error probability 2^{-k}, read $3k$ bits).

"The PCP Theorem"

There is such a format.

caveat: applies only for Mathematical Proofs

How easy is it to check a proof?

A motivating story

- Common wisdom: to check a proof you need to read it
- Why bother? instead-
  - Ask for the proof to be supplied in **PCP** format
  - Check randomly by reading only 3 bits.
    Probability of error, i.e. of accepting a bad proof, is at most $1/2$.
    (For error probability 2^{-k}, read $3k$ bits).

"The PCP Theorem"

There is such a format.

caveat: applies only for Mathematical Proofs

How easy is it to check a proof?

A motivating story

- Common wisdom: to check a proof you need to read it
- Why bother? instead-
  - Ask for the proof to be supplied in **PCP** format
  - Check randomly by reading only 3 bits.
    Probability of error, i.e. of accepting a bad proof, is at most $1/2$.
    (For error probability 2^{-k}, read $3k$ bits).

"The PCP Theorem"

There is such a format.

caveat: applies only for Mathematical Proofs

How easy is it to check a proof?

A motivating story

- Common wisdom: to check a proof you need to read it
- Why bother? instead-
  - Ask for the proof to be supplied in **PCP** format
  - Check randomly by reading only 3 bits.
    Probability of error, i.e. of accepting a bad proof, is at most $1/2$.
    (For error probability 2^{-k}, read $3k$ bits).

"The PCP Theorem"

There is such a format.

caveat: applies only for Mathematical Proofs

How easy is it to check a proof?

A motivating story

- Common wisdom: to check a proof you need to read it
- Why bother? instead-
  - Ask for the proof to be supplied in **PCP** format
  - Check randomly by reading only 3 bits.
    Probability of error, i.e. of accepting a bad proof, is at most $1/2$.
    (For error probability 2^{-k}, read $3k$ bits).

"The PCP Theorem"

There is such a format.

caveat: applies only for Mathematical Proofs

How easy is it to check a proof?

A motivating story

- Common wisdom: to check a proof you need to read it
- Why bother? instead-
  - Ask for the proof to be supplied in **PCP** format
  - Check randomly by reading only 3 bits.
    Probability of error, i.e. of accepting a bad proof, is at most $1/2$.
    (For error probability 2^{-k}, read $3k$ bits).

“The PCP Theorem”

There is such a format.

caveat: applies only for Mathematical Proofs

Theorems and Proofs, Problems and Solutions

- What is a mathematical proof?
- Anything that can be verified by a *rigorous* procedure, i.e., an algorithm

- More generally,

a theorem = a problem,
a proof = a solution

- The difference between a theorem and its proof, is *how long* it takes to verify it's correctness

Theorems and Proofs, Problems and Solutions

- What is a mathematical proof?
  – Anything that can be verified by a *rigorous* procedure, i.e., an algorithm

- More generally,

  a theorem = a problem,
  a proof = a solution

- The difference between a theorem and its proof, is *how long* it takes to verify it's correctness

Linear Equations

Input: A system of linear equations (over a finite field) :

$$
\begin{aligned}
x_1 + x_2 + x_3 &= 0, \\
x_1 + x_6 - x_2 + x_{90} &= 1 \\
&\vdots
\end{aligned}
$$

Algorithmic goal: Decide if there is a solution to all of the equations

Complexity: easy, by Gaussian elimination

But, "overdetermined" version is hard...

Note:

- Algorithm's efficiency is measured as a function of the input length. Polynomial = good, Exponential = bad.

Computational problems - examples

Linear Equations *LINEQ*

Linear Equations

Input: A system of linear equations (over a finite field) :

$$\begin{aligned} x_1 + x_2 + x_3 &= 0, \\ x_1 + x_6 - x_2 + x_{90} &= 1 \\ &\vdots \end{aligned}$$

Algorithmic goal: Decide if there is a solution to all of the equations

Complexity: easy, by Gaussian elimination
But, "overdetermined" version is hard...

Note:

- Algorithm's efficiency is measured as a function of the input length. Polynomial = good, Exponential = bad.

Computational problems - examples

Linear Equations *LINEQ*

Linear Equations

Input: A system of linear equations (over a finite field) :

$$\begin{aligned} x_1 + x_2 + x_3 &= 0, \\ x_1 + x_6 - x_2 + x_{90} &= 1 \\ &\vdots \end{aligned}$$

Algorithmic goal: Decide if there is a solution to all of the equations

Complexity: easy, by Gaussian elimination
But, "overdetermined" version is hard...

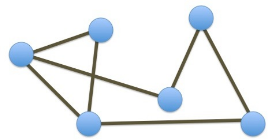
Note:

- Algorithm's efficiency is measured as a function of the input length. Polynomial = good, Exponential = bad.

Computational problems - examples

Graph 3 Colorability ($3COL$)

3-Coloring a graph



Input: A graph $G = (V, E)$

Algorithmic goal: Decide if there is a 3-coloring, i.e., a mapping $c : V \to \{1, 2, 3\}$ such that every edge has differently colored endpoints

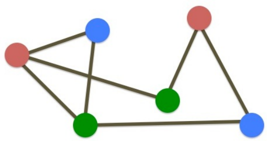
Complexity: hard to solve, but easy to check proof

Proof: A 3-coloring.

Definition (Computational Problem)

We identify a computational problem with the set of all 'yes' inputs.

3-Coloring a graph



Input: A graph $G = (V, E)$

Algorithmic goal: Decide if there is a 3-coloring, i.e., a mapping $c : V \to \{1, 2, 3\}$ such that every edge has differently colored endpoints

Complexity: hard to solve, but easy to check proof

Proof: A 3-coloring.

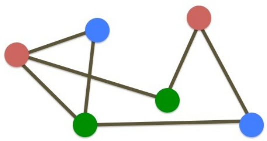
Definition (Computational Problem)

We identify a computational problem with the set of all 'yes' inputs.

Computational problems - examples

Graph 3 Colorability (3*COL*)

3-Coloring a graph



Input: A graph $G = (V, E)$

Algorithmic goal: Decide if there is a 3-coloring, i.e., a mapping $c : V \rightarrow \{1, 2, 3\}$ such that every edge has differently colored endpoints

Complexity: hard to solve, but easy to check proof

Proof: A 3-coloring.

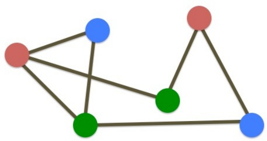
Definition (Computational Problem)

We identify a computational problem with the set of all ‘yes’ inputs.

Computational problems - examples

Graph 3 Colorability ($3COL$)

3-Coloring a graph



Input: A graph $G = (V, E)$

Algorithmic goal: Decide if there is a 3-coloring, i.e., a mapping $c : V \to \{1, 2, 3\}$ such that every edge has differently colored endpoints

Complexity: hard to solve, but easy to check proof

Proof: A 3-coloring.

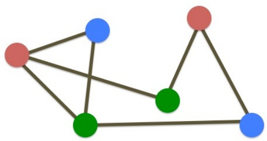
Definition (Computational Problem)

We identify a computational problem with the set of all 'yes' inputs.

Computational problems - examples

Graph 3 Colorability ($3COL$)

3-Coloring a graph



Input: A graph $G = (V, E)$

Algorithmic goal: Decide if there is a 3-coloring, i.e., a mapping $c : V \to \{1, 2, 3\}$ such that every edge has differently colored endpoints

Complexity: hard to solve, but easy to check proof

Proof: A 3-coloring.

Definition (Computational Problem)

We identify a computational problem with the set of all 'yes' inputs.

- P = (polynomial time)
  P is the class of efficiently decidable problems
  e.g. linear equations
- NP = (non-deterministically polynomial time)
  NP is the class of problems with efficiently checkable solutions
  e.g. 3-coloring, max-clique, ...
- $P \neq NP$: $1 Million Question: is discovering a proof as easy as checking it ?
- 3-coloring is "the hardest problem in NP" (aka NP-hard)

Theorem

If 3-coloring is in P then $P = NP$.

To understand NP, enough to study the 3-coloring problem.

P, NP, and all that

- P = (polynomial time)
  P is the class of efficiently decidable problems
  e.g. linear equations
- NP = (non-deterministically polynomial time)
  NP is the class of problems with efficiently checkable solutions
  e.g. 3-coloring, max-clique, ...
- $P \neq NP$: $1 Million Question: is discovering a proof as easy as checking it ?
- 3-coloring is "the hardest problem in NP" (aka NP-hard)

Theorem

If 3-coloring is in P then $P = NP$.

To understand NP, enough to study the 3-coloring problem.

P, NP, and all that

- P = (polynomial time)
  P is the class of efficiently decidable problems
  e.g. linear equations
- NP = (non-deterministically polynomial time)
  NP is the class of problems with efficiently checkable solutions
  e.g. 3-coloring, max-clique, ...
- $P \neq NP$: $1 Million Question: is discovering a proof as easy as checking it ?
- 3-coloring is "the hardest problem in NP" (aka NP-hard)

Theorem

If 3-coloring is in P then $P = NP$.

To understand NP, enough to study the 3-coloring problem.

P, NP, and all that

- P = (polynomial time)
  P is the class of efficiently decidable problems
  e.g. linear equations
- NP = (non-deterministically polynomial time)
  NP is the class of problems with efficiently checkable solutions
  e.g. 3-coloring, max-clique, ...
- $P \neq NP$: $1 Million Question: is discovering a proof as easy as checking it ?
- 3-coloring is "the hardest problem in NP" (aka NP-hard)

Theorem

If 3-coloring is in P then $P = NP$.

To understand NP, enough to study the 3-coloring problem.

Part II - The PCP Theorem

Arora-Safra,

Arora-Lund-Motwani-Sudan-Szegedy

1991

The PCP Theorem

The NP verifier (Definition)

Every problem $A \in NP$ has an efficient verifier V, that reads

- the input string τ and some randomness
- a constant number of bits from the proof string π

and then accepts or rejects, such that

- **Completeness:** If $\tau \in A$ then there is a proof that V accepts with probability 1.
- **Soundness:** If $\tau \notin A$ then for every π V rejects.

- The "error probability" can be reduced to $(\frac{1}{2})^k$ by k repetitions.
- Striking!

The PCP Theorem

The NP verifier (Definition)

Every problem $A \in NP$ has an efficient verifier V, that reads

- the input string τ and some randomness
- a constant number of bits from the proof string π

and then accepts or rejects, such that

- **Completeness:** If $\tau \in A$ then there is a proof that V accepts with probability 1.
- **Soundness:** If $\tau \notin A$ then for every π V rejects.

- The "error probability" can be reduced to $(\frac{1}{2})^k$ by k repetitions.
- Striking!

The PCP Theorem

The PCP verifier (Theorem)

Every problem $A \in NP$ has an efficient verifier V, that reads

- the input string τ and some randomness
- a constant number of bits from the proof string π

and then accepts or rejects, such that

- **Completeness:** If $\tau \in A$ then there is a proof that V accepts with probability 1.
- **Soundness:** If $\tau \notin A$ then for every π $\Pr_r[V^\pi \text{ accepts.}] < \frac{1}{2}$.

- The "error probability" can be reduced to $(\frac{1}{2})^k$ by k repetitions.
- Striking!

The PCP Theorem

The PCP verifier (Theorem)

Every problem $A \in NP$ has an efficient verifier V, that reads

- the input string τ and some randomness
- a constant number of bits from the proof string π

and then accepts or rejects, such that

- **Completeness:** If $\tau \in A$ then there is a proof that V accepts with probability 1.
- **Soundness:** If $\tau \notin A$ then for every π $\Pr_r[V^\pi \text{ accepts.}] < \frac{1}{2}$.

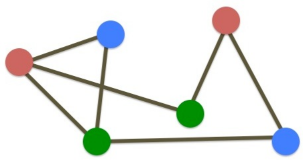
- The "error probability" can be reduced to $(\frac{1}{2})^k$ by k repetitions.
- Striking!

The PCP Theorem for 3-Coloring

The “natural” 3-Coloring verifier reads the coloring

$$c(v_1) = 1, c(v_2) = 2, \ldots$$

and then checks edge-by-edge that endpoints have different colors.



What will the PCP verifier look like?

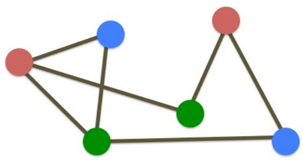
- Naive attempt: Choose a random edge, read the colors of its endpoints, and accept if true
- Fails! – a non 3-colorable graph may have a 3 coloring with as few as only one monochromatic edge.
- Instead: encode the “standard” proof into a “PCP” proof, spreading out the bugs.

The PCP Theorem for 3-Coloring

The "natural" 3-Coloring verifier reads the coloring

$$c(v_1) = 1, c(v_2) = 2, \ldots$$

and then checks edge-by-edge that endpoints have different colors.



What will the PCP verifier look like?

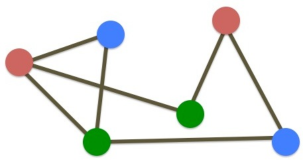
- Naive attempt: Choose a random edge, read the colors of its endpoints, and accept if true
- Fails! – a non 3-colorable graph may have a 3 coloring with as few as only one monochromatic edge.
- Instead: encode the "standard" proof into a "PCP" proof, spreading out the bugs.

The PCP Theorem for 3-Coloring

The "natural" 3-Coloring verifier reads the coloring

$$c(v_1) = 1, c(v_2) = 2, \ldots$$

and then checks edge-by-edge that endpoints have different colors.



What will the PCP verifier look like?

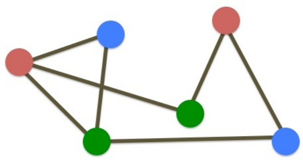
- Naive attempt: Choose a random edge, read the colors of its endpoints, and accept if true
- Fails! – a non 3-colorable graph may have a 3 coloring with as few as only one monochromatic edge.
- Instead: encode the "standard" proof into a "PCP" proof, spreading out the bugs.

The PCP Theorem for 3-Coloring

The “natural” 3-Coloring verifier reads the coloring

$$c(v_1) = 1, c(v_2) = 2, \ldots$$

and then checks edge-by-edge that endpoints have different colors.



What will the PCP verifier look like?

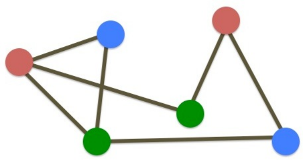
- Naive attempt: Choose a random edge, read the colors of its endpoints, and accept if true
- Fails! – a non 3-colorable graph may have a 3 coloring with as few as only one monochromatic edge.
- Instead: encode the “standard” proof into a “PCP” proof, spreading out the bugs.

The PCP Theorem for 3-Coloring

The "natural" 3-Coloring verifier reads the coloring

$$c(v_1) = 1, c(v_2) = 2, \ldots$$

and then checks edge-by-edge that endpoints have different colors.

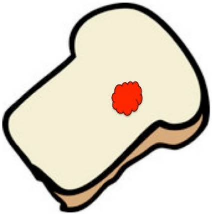


What will the PCP verifier look like?

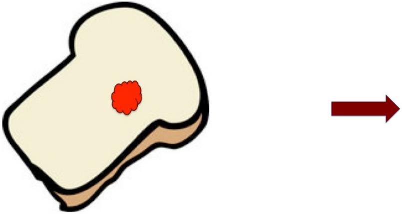
- Naive attempt: Choose a random edge, read the colors of its endpoints, and accept if true
- Fails! – a non 3-colorable graph may have a 3 coloring with as few as only one monochromatic edge.
- Instead: encode the "standard" proof into a "PCP" proof, spreading out the bugs.

The PCP Theorem - blind-folded jam spreading
[B. Chazelle]

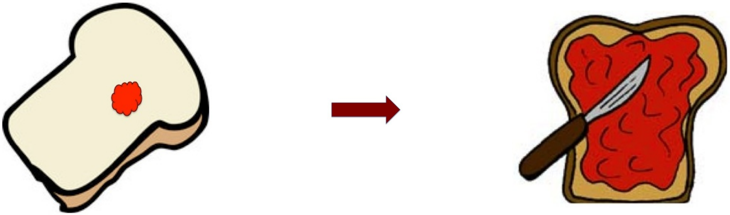
The PCP Theorem - blind-folded jam spreading
[B. Chazelle]



The PCP Theorem - blind-folded jam spreading
[B. Chazelle]



The PCP Theorem - blind-folded jam spreading
[B. Chazelle]



The PCP Theorem & Inapproximability

- What is this good for? refereeing?
- [Feige-Goldwasser-Lovász-Safra-Szegedy, 1990]
  "The PCP theorem stands at the heart of virtually all approximation lower bounds"
- Beautiful connections to robustness questions and inverse theorems in Combinatorics, Algebra, Analysis, ...

- What is this good for? refereeing?
- [Feige-Goldwasser-Lovász-Safra-Szegedy, 1990]
  "The PCP theorem stands at the heart of virtually all approximation lower bounds"
- Beautiful connections to robustness questions and inverse theorems in Combinatorics, Algebra, Analysis, ...

The PCP Theorem & Inapproximability

- What is this good for? refereeing?
- [Feige-Goldwasser-Lovász-Safra-Szegedy, 1990]
  "The PCP theorem stands at the heart of virtually all approximation lower bounds"
- Beautiful connections to robustness questions and inverse theorems in Combinatorics, Algebra, Analysis, ...

The PCP Theorem & Inapproximability

- What is this good for? refereeing?
- [Feige-Goldwasser-Lovász-Safra-Szegedy, 1990]
  "The PCP theorem stands at the heart of virtually all approximation lower bounds"
- Beautiful connections to robustness questions and inverse theorems in Combinatorics, Algebra, Analysis, ...

The PCP Theorem & Inapproximability

- What is this good for? refereeing?
- [Feige-Goldwasser-Lovász-Safra-Szegedy, 1990]
  "The PCP theorem stands at the heart of virtually all approximation lower bounds"
- Beautiful connections to robustness questions and inverse theorems in Combinatorics, Algebra, Analysis, ...

Approximation Problems

- Throughout 70's-80's: many problems discovered to be NP-hard
- Natural to seek approximate solutions. (Almost no known lower bounds)

Optimization

1. Max-LIN: satisfy the largest number of equations.
2. Max-3COL: color the vertices with 3 colors, maximizing number of two-colored edges.

Both these problems are NP-hard (yes, even Max-LIN!)

Approximation

1. satisfy at least $\geq \alpha \cdot OPT$ equations
2. satisfy at least $\geq \alpha \cdot OPT$ edge contraints

Complexity: depends on the problem, and on α

Approximation Problems

- Throughout 70's-80's: many problems discovered to be NP-hard
- Natural to seek approximate solutions. (Almost no known lower bounds)

Optimization

1. Max-LIN: satisfy the largest number of equations.
2. Max-3COL: color the vertices with 3 colors, maximizing number of two-colored edges.

Both these problems are NP-hard (yes, even Max-LIN!)

Approximation

1. satisfy at least $\geq \alpha \cdot OPT$ equations
2. satisfy at least $\geq \alpha \cdot OPT$ edge contraints

Complexity: depends on the problem, and on α

Approximation Problems

- Throughout 70's-80's: many problems discovered to be NP-hard
- Natural to seek approximate solutions. (Almost no known lower bounds)

Optimization

1. Max-LIN: satisfy the largest number of equations.
2. Max-3COL: color the vertices with 3 colors, maximizing number of two-colored edges.

Both these problems are NP-hard (yes, even Max-LIN!)

Approximation

1. satisfy at least $\geq \alpha \cdot OPT$ equations
2. satisfy at least $\geq \alpha \cdot OPT$ edge contraints

Complexity: depends on the problem, and on α

Approximation Problems

- Throughout 70's-80's: many problems discovered to be NP-hard
- Natural to seek approximate solutions. (Almost no known lower bounds)

Optimization

1. Max-LIN: satisfy the largest number of equations.
2. Max-3COL: color the vertices with 3 colors, maximizing number of two-colored edges.

Both these problems are NP-hard (yes, even Max-LIN!)

Approximation

1. satisfy at least $\geq \alpha \cdot OPT$ equations
2. satisfy at least $\geq \alpha \cdot OPT$ edge contraints

Complexity: depends on the problem, and on α

Approximation Problems

- Throughout 70's-80's: many problems discovered to be NP-hard
- Natural to seek approximate solutions. (Almost no known lower bounds)

Optimization

1. Max-LIN: satisfy the largest number of equations.
2. Max-3COL: color the vertices with 3 colors, maximizing number of two-colored edges.

Both these problems are NP-hard (yes, even Max-LIN!)

Approximation

1. satisfy at least $\geq \alpha \cdot OPT$ equations
2. satisfy at least $\geq \alpha \cdot OPT$ edge contraints

Complexity: depends on the problem, and on α

Approximation Problems

- Throughout 70's-80's: many problems discovered to be NP-hard
- Natural to seek approximate solutions. (Almost no known lower bounds)

Optimization

1. Max-LIN: satisfy the largest number of equations.
2. Max-3COL: color the vertices with 3 colors, maximizing number of two-colored edges.

Both these problems are NP-hard (yes, even Max-LIN!)

Approximation

1. satisfy at least $\geq \alpha \cdot OPT$ equations
2. satisfy at least $\geq \alpha \cdot OPT$ edge contraints

Complexity: depends on the problem, and on α

Hardness of Approximation

Claim:

If there is an efficient algorithm that maps a graph G to a graph G' such that:

Yes: If $OPT(G) = 1$, then $OPT(G') = 1$

No: If $OPT(G) < 1$, then $OPT(G') < 0.99$

Then, Max-3COL is NP-hard to 0.99-approximate. □

Hardness of Approximation

Claim:

If there is an efficient algorithm that maps a graph G to a graph G' such that:

Yes: If $OPT(G) = 1$, then $OPT(G') = 1$

No: If $OPT(G) < 1$, then $OPT(G') < 0.99$

Then, Max-3COL is NP-hard to 0.99-approximate. □

This is a "gap amplifying reduction":

Hardness of Approximation

Claim:

If there is an efficient algorithm that maps a graph G to a graph G' such that:

Yes: If $OPT(G) = 1$, then $OPT(G') = 1$

No: If $OPT(G) < 1$, then $OPT(G') < 0.99$

Then, Max-3COL is NP-hard to 0.99-approximate. □

Claim:

Such a reduction implies the PCP theorem.

PCP & Inapproximability

The PCP Theorem (1) – original formulation

There is an efficient verifier for 3-coloring that reads: the input G, randomness r, and then a constant number of bits from the proof, such that

Yes: If G is 3-colorable, then $\Pr_r[V \text{ accepts}] = 1$
No: If G is not 3-colorable, then $\Pr_r[V \text{ accepts}] \leq \frac{1}{2}$.

The PCP Theorem (2) – second formulation

There is an efficient algorithm that maps graphs G to graphs G' such that:

Yes: If $OPT(G) = 1$, then $OPT(G') = 1$
No: If $OPT(G) < 1$, then $OPT(G') \leq 0.99$

To prove (2) $\Longrightarrow$ (1) we present a PCP verifier for 3-coloring:

1. Read the input G, compute G'.
2. "proof" = coloring of G''s vertices. Check on 20 random edges.

(1) $\Longrightarrow$ (2) : exercise.

PCP & Inapproximability

The PCP Theorem (1) – original formulation

There is an efficient verifier for 3-coloring that reads: the input G, randomness r, and then a constant number of bits from the proof, such that

Yes: If G is 3-colorable, then $\Pr_r[V \text{ accepts}] = 1$
No: If G is not 3-colorable, then $\Pr_r[V \text{ accepts}] \leq \frac{1}{2}$.

The PCP Theorem (2) – second formulation

There is an efficient algorithm that maps graphs G to graphs G' such that:

Yes: If $OPT(G) = 1$, then $OPT(G') = 1$
No: If $OPT(G) < 1$, then $OPT(G') \leq 0.99$

To prove (2) $\Longrightarrow$ (1) we present a PCP verifier for 3-coloring:

1. Read the input G, compute G'.
2. "proof" = coloring of G''s vertices. Check on 20 random edges.

(1) $\Longrightarrow$ (2) : exercise.

PCP & Inapproximability

The PCP Theorem (1) – original formulation

There is an efficient verifier for 3-coloring that reads: the input G, randomness r, and then a constant number of bits from the proof, such that

Yes: If G is 3-colorable, then $\Pr_r[V \text{ accepts}] = 1$

No: If G is not 3-colorable, then $\Pr_r[V \text{ accepts}] \leq \frac{1}{2}$.

The PCP Theorem (2) – second formulation

There is an efficient algorithm that maps graphs G to graphs G' such that:

Yes: If $OPT(G) = 1$, then $OPT(G') = 1$

No: If $OPT(G) < 1$, then $OPT(G') \leq 0.99$

To prove (2) $\Longrightarrow$ (1) we present a PCP verifier for 3-coloring:

1. Read the input G, compute G'.
2. "proof" = coloring of G''s vertices. Check on 20 random edges.

(1) $\Longrightarrow$ (2) : exercise.

PCP & Inapproximability

The PCP Theorem (1) – original formulation

There is an efficient verifier for 3-coloring that reads: the input G, randomness r, and then a constant number of bits from the proof, such that

Yes: If G is 3-colorable, then $\Pr_r[V \text{ accepts}] = 1$

No: If G is not 3-colorable, then $\Pr_r[V \text{ accepts}] \leq \frac{1}{2}$.

The PCP Theorem (2) – second formulation

There is an efficient algorithm that maps graphs G to graphs G' such that:

Yes: If $OPT(G) = 1$, then $OPT(G') = 1$

No: If $OPT(G) < 1$, then $OPT(G') \leq 0.99$

To prove (2) $\Longrightarrow$ (1) we present a PCP verifier for 3-coloring:

1. Read the input G, compute G'.
2. "proof" = coloring of G''s vertices. Check on 20 random edges.

(1) $\Longrightarrow$ (2) : exercise.

PCP & Inapproximability

The PCP Theorem (1) – original formulation

There is an efficient verifier for 3-coloring that reads: the input G, randomness r, and then a constant number of bits from the proof, such that

Yes: If G is 3-colorable, then $\Pr_r[V \text{ accepts}] = 1$
No: If G is not 3-colorable, then $\Pr_r[V \text{ accepts}] \leq \frac{1}{2}$.

The PCP Theorem (2) – second formulation

There is an efficient algorithm that maps graphs G to graphs G' such that:

Yes: If $OPT(G) = 1$, then $OPT(G') = 1$
No: If $OPT(G) < 1$, then $OPT(G') \leq 0.99$

To prove (2) $\Longrightarrow$ (1) we present a PCP verifier for 3-coloring:

1. Read the input G, compute G'.
2. "proof" = coloring of G''s vertices. Check on 20 random edges.

(1) $\Longrightarrow$ (2) : exercise.

Tightness of Inapproximability

- For example, Max-3LIN:
  1. NP-hard to 1-approximate, i.e. to solve exactly
  2. Easy to $\frac{1}{2}$-approximate, e.g. by a random assignment
  3. What about α-approximation for $\frac{1}{2} \leq \alpha < 1$?
- Interesting to study boundary between hard and easy ends, possibly pinpoint the point of transition?

Theorem (Håstad '97)

Given a 3LIN instance that is $1 - o(1)$ satisfiable, it is NP-hard to find an assignment satisfying $1/2 + o(1)$ of the clauses.
"Can't beat the random assignment"

- Very active field, connections to *robustness questions* in various math areas

Tightness of Inapproximability

- For example, Max-3LIN:
  1. NP-hard to 1-approximate, i.e. to solve exactly
  2. Easy to $\frac{1}{2}$-approximate, e.g. by a random assignment
  3. What about α-approximation for $\frac{1}{2} \leq \alpha < 1$?
- Interesting to study boundary between hard and easy ends, possibly pinpoint the point of transition?

Theorem (Håstad '97)

Given a 3LIN instance that is $1 - o(1)$ satisfiable, it is NP-hard to find an assignment satisfying $1/2 + o(1)$ of the clauses.

"Can't beat the random assignment"

- Very active field, connections to *robustness questions* in various math areas

Tightness of Inapproximability

- For example, Max-3LIN:
  1. NP-hard to 1-approximate, i.e. to solve exactly
  2. Easy to $\frac{1}{2}$-approximate, e.g. by a random assignment
  3. What about α-approximation for $\frac{1}{2} \leq \alpha < 1$?
- Interesting to study boundary between hard and easy ends, possibly pinpoint the point of transition?

Theorem (Håstad '97)

Given a 3LIN instance that is $1 - o(1)$ satisfiable, it is NP-hard to find an assignment satisfying $1/2 + o(1)$ of the clauses.
"Can't beat the random assignment"

- Very active field, connections to *robustness questions* in various math areas

Tightness of Inapproximability

- For example, Max-3LIN:
  1. NP-hard to 1-approximate, i.e. to solve exactly
  2. Easy to $\frac{1}{2}$-approximate, e.g. by a random assignment
  3. What about α-approximation for $\frac{1}{2} \leq \alpha < 1$?
- Interesting to study boundary between hard and easy ends, possibly pinpoint the point of transition?

Theorem (Håstad '97)

Given a 3LIN instance that is $1 - o(1)$ satisfiable, it is NP-hard to find an assignment satisfying $1/2 + o(1)$ of the clauses.
"Can't beat the random assignment"

- Very active field, connections to *robustness questions* in various math areas

Robust Systems

A system is robust (or *stable*) if every approximate solution is close to a perfect solution.

Example (System of Equations)

$$\begin{aligned} x_1^3 x_2 + 7x_3 &= 2, \\ x_1 x_2 x_3 &= 1 \\ &\vdots \end{aligned}$$

1. approximate solution: satisfies $1-\varepsilon$ of the equations.
2. close to: agrees on $1-\delta$ of the coordinates

- Two different measures: (1) equation-based, (2) solution-based.
- Non-trivial: A small perturbation of a perfect solution is an approximate solution. Here, every approximate solution is a perturbation of a perfect solution.
- Many other examples: clique, 3sat, cuts in graphs

Robust Systems

A system is robust (or *stable*) if every approximate solution is close to a perfect solution.

Example (System of Equations)

$$
\begin{aligned}
x_1^3 x_2 + 7x_3 &= 2, \\
x_1 x_2 x_3 &= 1 \\
&\vdots
\end{aligned}
$$

1. approximate solution: satisfies $1-\varepsilon$ of the equations.
2. close to: agrees on $1-\delta$ of the coordinates

- Two different measures: (1) equation-based, (2) solution-based.
- Non-trivial: A small perturbation of a perfect solution is an approximate solution. Here, every approximate solution is a perturbation of a perfect solution.
- Many other examples: clique, 3sat, cuts in graphs

Robust Systems

A system is robust (or *stable*) if every approximate solution is close to a perfect solution.

Example (System of Equations)

$$
\begin{aligned}
x_1^3 x_2 + 7x_3 &= 2, \\
x_1 x_2 x_3 &= 1 \\
&\vdots
\end{aligned}
$$

1. approximate solution: satisfies $1-\varepsilon$ of the equations.
2. close to: agrees on $1-\delta$ of the coordinates

- Two different measures: (1) equation-based, (2) solution-based.
- Non-trivial: A small perturbation of a perfect solution is an approximate solution. Here, every approximate solution is a perturbation of a perfect solution.
- Many other examples: clique, 3sat, cuts in graphs

Robust Systems

A system is robust (or *stable*) if every approximate solution is close to a perfect solution.

Example (System of Equations)

$$\begin{aligned} x_1^3 x_2 + 7x_3 &= 2, \\ x_1 x_2 x_3 &= 1 \\ &\vdots \end{aligned}$$

1. approximate solution: satisfies $1-\varepsilon$ of the equations.
2. close to: agrees on $1-\delta$ of the coordinates

- Two different measures: (1) equation-based, (2) solution-based.
- Non-trivial: A small perturbation of a perfect solution is an approximate solution. Here, every approximate solution is a perturbation of a perfect solution.
- Many other examples: clique, 3sat, cuts in graphs

Robust Systems

A system is robust (or *stable*) if every approximate solution is close to a perfect solution.

Example (System of Equations)

$$x_1^3 x_2 + 7x_3 = 2,$$
$$x_1 x_2 x_3 = 1$$
$$\vdots$$

1. approximate solution: satisfies $1 - \varepsilon$ of the equations.
2. close to: agrees on $1 - \delta$ of the coordinates

- Two different measures: (1) equation-based, (2) solution-based.
- Non-trivial: A small perturbation of a perfect solution is an approximate solution. Here, every approximate solution is a perturbation of a perfect solution.
- Many other examples: clique, 3sat, cuts in graphs

Robust Systems

A system is robust (or *stable*) if every approximate solution is close to a perfect solution.

Example (System of Equations)

$$\begin{aligned} x_1^3 x_2 + 7x_3 &= 2, \\ x_1 x_2 x_3 &= 1 \\ &\vdots \end{aligned}$$

1. approximate solution: satisfies $1-\varepsilon$ of the equations.
2. close to: agrees on $1-\delta$ of the coordinates

- Two different measures: (1) equation-based, (2) solution-based.
- Non-trivial: A small perturbation of a perfect solution is an approximate solution. Here, every approximate solution is a perturbation of a perfect solution.
- Many other examples: clique, 3sat, cuts in graphs

Robust Systems

A system is robust (or *stable*) if every approximate solution is close to a perfect solution.

Example (System of Equations)

$$
\begin{aligned}
x_1^3 x_2 + 7x_3 &= 2, \\
x_1 x_2 x_3 &= 1 \\
&\vdots
\end{aligned}
$$

1. approximate solution: satisfies $1 - \varepsilon$ of the equations.
2. close to: agrees on $1 - \delta$ of the coordinates

- Two different measures: (1) equation-based, (2) solution-based.
- Non-trivial: A small perturbation of a perfect solution is an approximate solution. Here, every approximate solution is a perturbation of a perfect solution.
- Many other examples: clique, 3sat, cuts in graphs

Robustness Questions and Inverse Theorems

Robustness is a "desirable" property of systems, and natural to study.

1. Additive combinatorics: approximate fields and groups
   If a set is somewhat linear it must be close to a field / group
   [Freiman, Erdös-Szemeredy,. . .]
2. Discrete Fourier analysis & geometry: approximate dictatorships
   If a function $f : \{0,1\}^n \to \mathbb{R}$ is somewhat noise-stable then it must depend on few coordinates [Mossel-O'Donnell-Oleszkiewicz extension of CLT]
3. Extremal set systems: approximate Erdös-Ko-Rado theorems; approximate cliques in certain graphs
   If a clique is somewhat large it must be close to a maximum clique
4. ...
5. The PCP Theorem **If a PCP proof is somewhat correct it must be close to perfectly correct proof**

Robustness Questions and Inverse Theorems

Robustness is a “desirable” property of systems, and natural to study.

1. Additive combinatorics: approximate fields and groups
   If a set is somewhat linear it must be close to a field / group
   [Freiman, Erdös-Szemeredy,. . .]
2. Discrete Fourier analysis & geometry: approximate dictatorships
   If a function $f : \{0,1\}^n \to \mathbb{R}$ is somewhat noise-stable then it must depend on few coordinates [Mossel-O'Donnell-Oleszkiewicz extension of CLT]
3. Extremal set systems: approximate Erdös-Ko-Rado theorems; approximate cliques in certain graphs
   If a clique is somewhat large it must be close to a maximum clique
4. ...
5. The PCP Theorem **If a PCP proof is somewhat correct it must be close to perfectly correct proof**

Robustness Questions and Inverse Theorems

Robustness is a "desirable" property of systems, and natural to study.

1. Additive combinatorics: approximate fields and groups
   If a set is somewhat linear it must be close to a field / group [Freiman, Erdös-Szemeredy,. . .]
2. Discrete Fourier analysis & geometry: approximate dictatorships
   If a function $f : \{0,1\}^n \to \mathbb{R}$ is somewhat noise-stable then it must depend on few coordinates [Mossel-O'Donnell-Oleszkiewicz extension of CLT]
3. Extremal set systems: approximate Erdös-Ko-Rado theorems; approximate cliques in certain graphs
   If a clique is somewhat large it must be close to a maximum clique
4. ...
5. The PCP Theorem **If a PCP proof is somewhat correct it must be close to perfectly correct proof**

Robustness Questions and Inverse Theorems

Robustness is a “desirable” property of systems, and natural to study.

1. Additive combinatorics: approximate fields and groups
   If a set is somewhat linear it must be close to a field / group
   [Freiman, Erdös-Szemeredy,...]
2. Discrete Fourier analysis & geometry: approximate dictatorships
   If a function $f : \{0,1\}^n \to \mathbb{R}$ is somewhat noise-stable then it must depend on few coordinates [Mossel-O'Donnell-Oleszkiewicz extension of CLT]
3. Extremal set systems: approximate Erdös-Ko-Rado theorems; approximate cliques in certain graphs
   If a clique is somewhat large it must be close to a maximum clique
4. ...
5. The PCP Theorem **If a PCP proof is somewhat correct it must be close to perfectly correct proof**

Robustness Questions and Inverse Theorems

Robustness is a “desirable” property of systems, and natural to study.

1. Additive combinatorics: approximate fields and groups
   If a set is somewhat linear it must be close to a field / group [Freiman, Erdös-Szemeredy,. . .]
2. Discrete Fourier analysis & geometry: approximate dictatorships
   If a function $f : \{0,1\}^n \to \mathbb{R}$ is somewhat noise-stable then it must depend on few coordinates [Mossel-O'Donnell-Oleszkiewicz extension of CLT]
3. Extremal set systems: approximate Erdös-Ko-Rado theorems; approximate cliques in certain graphs
   If a clique is somewhat large it must be close to a maximum clique
4. ...
5. The PCP Theorem **If a PCP proof is somewhat correct it must be close to perfectly correct proof**

Part III

(Flavors of)

the Proof of the PCP Theorem

Proving the PCP theorem

- Goal: find an efficient algorithm that maps graphs G to graphs H such that:

  Yes: If $OPT(G) = 1$, then $OPT(H) = 1$
  No: If $OPT(G) < 1$, then $OPT(H) \leq 0.99$

  (let jam$(G) := 1 - OPT(G)$)

- There are two different approaches.
  1. The original "algebraic" proof [Arora-Safra, Arora-Lund-Motwani-Sudan-Szegedy, 1991]
     Technique: H encodes G via low degree polynomials over finite fields
  2. The "combinatorial" proof [Dinur, 2006]
     Technique: gradual gap amplification using graph structure

Proving the PCP theorem

- Goal: find an efficient algorithm that maps graphs G to graphs H such that:

  Yes: If $\text{jam}(G) = 0$, then $\text{jam}(H) = 0$

  No: If $\text{jam}(G) > 0$, then $\text{jam}(H) \geq 0.01$

  (let $\text{jam}(G) := 1 - OPT(G)$)

- There are two different approaches.
  1. The original "algebraic" proof [Arora-Safra, Arora-Lund-Motwani-Sudan-Szegedy, 1991]
     Technique: H encodes G via low degree polynomials over finite fields
  2. The "combinatorial" proof [Dinur, 2006]
     Technique: gradual gap amplification using graph structure

Proving the PCP theorem

- Goal: find an efficient algorithm that maps graphs G to graphs H such that:

  Yes: If $\text{jam}(G) = 0$, then $\text{jam}(H) = 0$
  No: If $\text{jam}(G) > 0$, then $\text{jam}(H) \geq 0.01$

  $$(\text{let } \text{jam}(G) := 1 - OPT(G))$$

- There are two different approaches.
  1. The original "algebraic" proof [Arora-Safra, Arora-Lund-Motwani-Sudan-Szegedy, 1991]
     Technique: H encodes G via low degree polynomials over finite fields
  2. The "combinatorial" proof [Dinur, 2006]
     Technique: gradual gap amplification using graph structure

Proving the PCP theorem

- Goal: find an efficient algorithm that maps graphs G to graphs H such that:

  Yes: If $\text{jam}(G) = 0$, then $\text{jam}(H) = 0$
  No: If $\text{jam}(G) > 0$, then $\text{jam}(H) \geq 0.01$

  $$(\text{let } \text{jam}(G) := 1 - OPT(G))$$

- There are two different approaches.
  1. The original "algebraic" proof [Arora-Safra, Arora-Lund-Motwani-Sudan-Szegedy, 1991]
     Technique: H encodes G via low degree polynomials over finite fields
  2. The "combinatorial" proof [Dinur, 2006]
     Technique: gradual gap amplification using graph structure

Proving the PCP theorem

- Goal: find an efficient algorithm that maps graphs G to graphs H such that:

  Yes: If $\text{jam}(G) = 0$, then $\text{jam}(H) = 0$
  No: If $\text{jam}(G) > 0$, then $\text{jam}(H) \geq 0.01$

  $$(\text{let } \text{jam}(G) := 1 - OPT(G))$$

- There are two different approaches.
  1. The original "algebraic" proof [Arora-Safra, Arora-Lund-Motwani-Sudan-Szegedy, 1991]
     Technique: H encodes G via low degree polynomials over finite fields
  2. The "combinatorial" proof [Dinur, 2006]
     Technique: gradual gap amplification using graph structure

Proving the PCP theorem

- Goal: find an efficient algorithm that maps graphs G to graphs H such that:

  Yes: If $\mathrm{jam}(G) = 0$, then $\mathrm{jam}(H) = 0$
  No: If $\mathrm{jam}(G) > 0$, then $\mathrm{jam}(H) \geq 0.01$

  $$(\text{let } \mathrm{jam}(G) := 1 - OPT(G))$$

- There are two different approaches.
  1. The original "algebraic" proof [Arora-Safra, Arora-Lund-Motwani-Sudan-Szegedy, 1991]
     Technique: H encodes G via low degree polynomials over finite fields
  2. The "combinatorial" proof [Dinur, 2006]
     Technique: gradual gap amplification using graph structure

Proof Idea

The idea is to proceed in many small steps:

$$G \Longrightarrow G_1 \Longrightarrow G_2 \Longrightarrow \cdots \Longrightarrow G_k =: H$$

such that the "jam" value gets amplified, unless it was zero.
(pictorially, the "jam" is spread little by little)

The basic step

We show a mapping $G \Longrightarrow G'$ for which

- Yes: If $\text{jam}(G) = 0$, then $\text{jam}(G') = 0$
- No: If $\text{jam}(G) > 0$, then $\text{jam}(G') \geq 2 \cdot \text{jam}(G)$ (unless large already)

Proof Idea

The idea is to proceed in many small steps:

$$G \Longrightarrow G_1 \Longrightarrow G_2 \Longrightarrow \cdots \Longrightarrow G_k =: H$$

such that the "jam" value gets amplified, unless it was zero.
(pictorially, the "jam" is spread little by little)

The basic step

We show a mapping $G \Longrightarrow G'$ for which

- **Yes:** If $\text{jam}(G) = 0$, then $\text{jam}(G') = 0$
- **No:** If $\text{jam}(G) > 0$, then $\text{jam}(G') \geq 2 \cdot \text{jam}(G)$ (unless large already)

Proof Idea

The idea is to proceed in many small steps:

$$G \Longrightarrow G_1 \Longrightarrow G_2 \Longrightarrow \cdots \Longrightarrow G_k =: H$$

such that the "jam" value gets amplified, unless it was zero.
(pictorially, the "jam" is spread little by little)

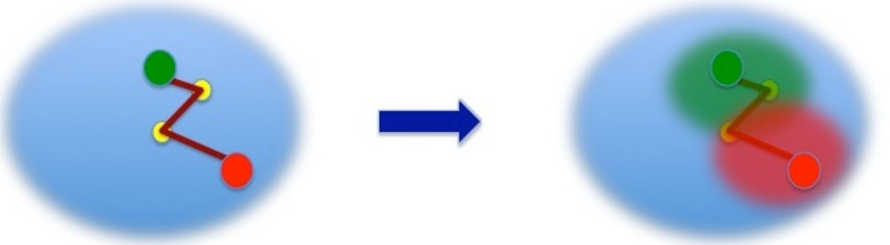
The basic step

We show a mapping $G \Longrightarrow G'$ for which

Yes: If $\text{jam}(G) = 0$, then $\text{jam}(G') = 0$

No: If $\text{jam}(G) > 0$, then $\text{jam}(G') \geq 2 \cdot \text{jam}(G)$ (unless large already)

The basic step $G \Longrightarrow G'$

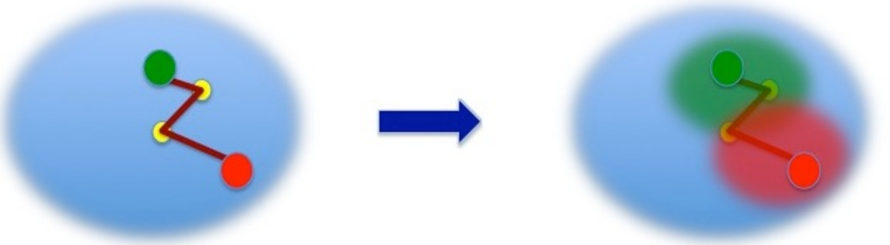


The mapping consists of two sub-steps $G \xrightarrow{1} \hat{G} \xrightarrow{2} G'$:

1. Gather: Each vertex gathers the colors of its neighbors. Encoded by new color of vertex. $\hat{G}$-edges are length-3 paths in G, each tests for inconsistencies.

2. Disperse: This step splits each vertex into several vertices, and replaces the "tests" by regular edges, yielding a 3-coloring instance. Robustly.
   How? by recursion: using a weaker PCP theorem!

The basic step $G \Longrightarrow G'$

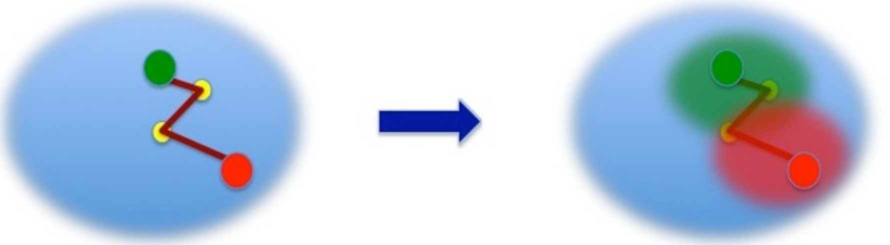


The mapping consists of two sub-steps $G \xrightarrow{1} \hat{G} \xrightarrow{2} G'$:

1. Gather: Each vertex gathers the colors of its neighbors. Encoded by new color of vertex. $\hat{G}$-edges are length-3 paths in G, each tests for inconsistencies.

2. Disperse: This step splits each vertex into several vertices, and replaces the “tests” by regular edges, yielding a 3-coloring instance. Robustly.
   How? by recursion: using a weaker PCP theorem!

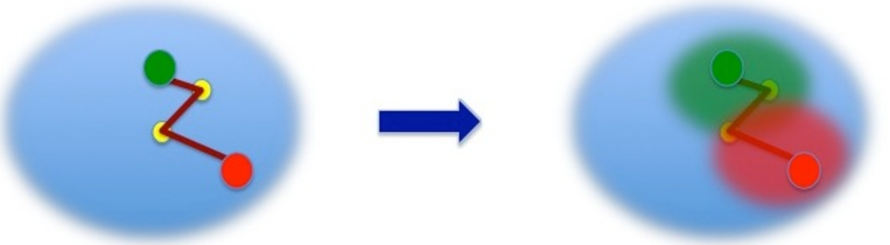
The basic step $G \Longrightarrow G'$



The mapping consists of two sub-steps $G \xrightarrow{1} \hat{G} \xrightarrow{2} G'$:

1. Gather: Each vertex gathers the colors of its neighbors. Encoded by new color of vertex. $\hat{G}$-edges are length-3 paths in G, each tests for inconsistencies.
2. Disperse: This step splits each vertex into several vertices, and replaces the “tests” by regular edges, yielding a 3-coloring instance. Robustly.

   How? by recursion: using a weaker PCP theorem!



The mapping consists of two sub-steps $G \xrightarrow{1} \hat{G} \xrightarrow{2} G'$:

1. Gather: Each vertex gathers the colors of its neighbors. Encoded by new color of vertex. $\hat{G}$-edges are length-3 paths in G, each tests for inconsistencies.

2. Disperse: This step splits each vertex into several vertices, and replaces the "tests" by regular edges, yielding a 3-coloring instance. Robustly.
   How? by recursion: using a weaker PCP theorem!

Wrapping up the proof

- A "gathering" step increases the jam value, but looses the 3-coloring structure.
- A "dispersing" step regains the 3-coloring structure using a "gadget" (i.e. local replacement)
- Each pair of steps spreads the jam value a bit further (each vertex v is aware of vertices at growing distances)
- While each step is "local", the in the final outcome every vertex has been affected by the entire graph.
  ...and the bug, if existed, has been properly spread around.

Wrapping up the proof

- A “gathering” step increases the jam value, but looses the 3-coloring structure.
- A “dispersing” step regains the 3-coloring structure using a “gadget” (i.e. local replacement)
- Each pair of steps spreads the jam value a bit further (each vertex v is aware of vertices at growing distances)
- While each step is “local”, the in the final outcome every vertex has been affected by the entire graph.
  ...and the bug, if existed, has been properly spread around.

□

Summary

- Proofs can come in a robust form, which allows randomized local checking
- Hardness of approximation
- Robustness questions

Summary

- Proofs can come in a robust form, which allows randomized local checking
- Hardness of approximation
- Robustness questions

Summary

- Proofs can come in a robust form, which allows randomized local checking
- Hardness of approximation
- Robustness questions

Thank You!