Probabilistically Checkable Proofs

Haris Angelidakis

MPLA

February 16, 2012

Outline

Outline

1. PCP Theorems

2. PCP's and Hardness of Approximation

Question: How easy is to check a proof?

Immediate answer: At least you have to read the whole proof, and try to check every step in it.

Weird question: Can we do better than that? I mean, can we ignore most part of the proof??

Even weirder answer: Yes!

Ok, almost yes....:D

Introduction to PCP's

Question: How easy is to check a proof?

Immediate answer: At least you have to read the whole proof, and try to check every step in it.

Weird question: Can we do better than that? I mean, can we ignore most part of the proof??

Even weirder answer: Yes!

Ok, almost yes....:D

Introduction to PCP's

Question: How easy is to check a proof?

Immediate answer: At least you have to read the whole proof, and try to check every step in it.

Weird question: Can we do better than that? I mean, can we ignore most part of the proof??

Even weirder answer: Yes!

Ok, almost yes....:D

Introduction to PCP's

Question: How easy is to check a proof?

Immediate answer: At least you have to read the whole proof, and try to check every step in it.

Weird question: Can we do better than that? I mean, can we ignore most part of the proof??

Even weirder answer: Yes!

Ok, almost yes....:D

Question: How easy is to check a proof?

Immediate answer: At least you have to read the whole proof, and try to check every step in it.

Weird question: Can we do better than that? I mean, can we ignore most part of the proof??

Even weirder answer: Yes!

Ok, almost yes....:D

The PCP Idea

So, we want to check a proof faster than usual. How is this done?

- We first rewrite the proof in a certain format, the **PCP format**.
- We then check randomly a constant number of its bits:
  - A correct proof always convinces us.
  - A false proof will convince us with probability $\leq 1/2$.

Detail: The rewriting is completely mechanical and does not greatly increase its size. **But**, it requires proofs to be written in a formal axiomatic system (such as ZF Set Theory).

The PCP Idea

So, we want to check a proof faster than usual. How is this done?

- We first rewrite the proof in a certain format, the **PCP format**.
- We then check randomly a constant number of its bits:
  - A correct proof always convinces us.
  - A false proof will convince us with probability $\leq 1/2$.

Detail: The rewriting is completely mechanical and does not greatly increase its size. **But**, it requires proofs to be written in a formal axiomatic system (such as ZF Set Theory).

The PCP Idea

So, we want to check a proof faster than usual. How is this done?

- We first rewrite the proof in a certain format, the **PCP format**.
- We then check randomly a constant number of its bits:
  - A correct proof always convinces us.
  - A false proof will convince us with probability $\leq 1/2$.

Detail: The rewriting is completely mechanical and does not greatly increase its size. **But**, it requires proofs to be written in a formal axiomatic system (such as ZF Set Theory).

The PCP Idea

So, we want to check a proof faster than usual. How is this done?

- We first rewrite the proof in a certain format, the **PCP format**.
- We then check randomly a constant number of its bits:
  - A correct proof always convinces us.
  - A false proof will convince us with probability $\leq 1/2$.

Detail: The rewriting is completely mechanical and does not greatly increase its size. **But**, it requires proofs to be written in a formal axiomatic system (such as ZF Set Theory).

The PCP Idea

So, we want to check a proof faster than usual. How is this done?

- We first rewrite the proof in a certain format, the **PCP format**.
- We then check randomly a constant number of its bits:
  - A correct proof always convinces us.
  - A false proof will convince us with probability $\leq 1/2$.

Detail: The rewriting is completely mechanical and does not greatly increase its size. **But**, it requires proofs to be written in a formal axiomatic system (such as ZF Set Theory).

The PCP Idea

So, we want to check a proof faster than usual. How is this done?

- We first rewrite the proof in a certain format, the **PCP format**.
- We then check randomly a constant number of its bits:
  - A correct proof always convinces us.
  - A false proof will convince us with probability $\leq 1/2$.

Detail: The rewriting is completely mechanical and does not greatly increase its size. **But**, it requires proofs to be written in a formal axiomatic system (such as ZF Set Theory).

The surprising main idea

- In general, a mathematical proof is invalid if it has even a single error somewhere, which can be very difficult to detect.
- What PCP theorems tell us is that there is a mechanical way to rewrite the proof so that the error is almost everywhere!

The surprising main idea

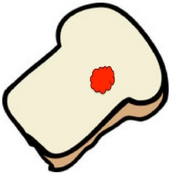
- In general, a mathematical proof is invalid if it has even a single error somewhere, which can be very difficult to detect.
- What PCP theorems tell us is that there is a mechanical way to rewrite the proof so that the error is almost everywhere!

The surprising main idea

- In general, a mathematical proof is invalid if it has even a single error somewhere, which can be very difficult to detect.
- What PCP theorems tell us is that there is a mechanical way to rewrite the proof so that the error is almost everywhere!

A nice analogue is the following:

Initial Proof



The surprising main idea

- In general, a mathematical proof is invalid if it has even a single error somewhere, which can be very difficult to detect.
- What PCP theorems tell us is that there is a mechanical way to rewrite the proof so that the error is almost everywhere!

A nice analogue is the following:

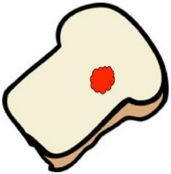
Initial Proof **PCP transformation**

The surprising main idea

- In general, a mathematical proof is invalid if it has even a single error somewhere, which can be very difficult to detect.
- What PCP theorems tell us is that there is a mechanical way to rewrite the proof so that the error is almost everywhere!

A nice analogue is the following:

Initial Proof



PCP transformation

PCP Format

Towards a new definition of NP

Note: From now on, we shall refer to languages $L \subseteq \{0,1\}^*$.

Definition (NP classic definition)

$NP = \cup_{k \in \mathbb{N}} NTIME(n^k)$

Definition (NP "yes"-certificate definition)

A language L is in NP if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a deterministic polynomial-time TM M (called the **verifier** of L) such that for every $x \in \{0,1\}^*$,

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ such that } M(x,u) = 1.$$

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) = 1$, then we call u a **certificate** for x (with respect to the language L and machine M).

Towards a new definition of NP

Note: From now on, we shall refer to languages $L \subseteq \{0,1\}^*$.

Definition (NP classic definition)

$NP = \cup_{k\in\mathbb{N}} NTIME(n^k)$

Definition (NP "yes"-certificate definition)

A language L is in NP if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a deterministic polynomial-time TM M (called the **verifier** of L) such that for every $x \in \{0,1\}^*$,

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ such that } M(x,u) = 1.$$

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) = 1$, then we call u a **certificate** for x (with respect to the language L and machine M).

Towards a new definition of NP

Note: From now on, we shall refer to languages $L \subseteq \{0,1\}^*$.

Definition (NP classic definition)

$NP = \cup_{k \in \mathbb{N}} NTIME(n^k)$

Definition (NP "yes"-certificate definition)

A language L is in NP if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a deterministic polynomial-time TM M (called the **verifier** of L) such that for every $x \in \{0,1\}^*$,

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ such that } M(x,u) = 1.$$

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) = 1$, then we call u a **certificate** for x (with respect to the language L and machine M).

Towards a new definition of *NP*

- Informally, *NP* is the complexity class of problems for which it is easy to check that a solution is correct.
- In contrast, finding solutions to *NP* problems is widely believed to be hard.
- Consider for example the problem 3-SAT. Given a 3-CNF Boolean formula, it is notoriously difficult to come up with a satisfying assignment, whereas given a proposed assignment it is trivial to plug in the values and verify its correctness. Such an assignment is an *NP*-proof for the satisfiability of the formula.

Towards a new definition of *NP*

- Informally, *NP* is the complexity class of problems for which it is easy to check that a solution is correct.
- In contrast, finding solutions to *NP* problems is widely believed to be hard.
- Consider for example the problem 3-SAT. Given a 3-CNF Boolean formula, it is notoriously difficult to come up with a satisfying assignment, whereas given a proposed assignment it is trivial to plug in the values and verify its correctness. Such an assignment is an *NP*-proof for the satisfiability of the formula.

Towards a new definition of *NP*

- Informally, *NP* is the complexity class of problems for which it is easy to check that a solution is correct.
- In contrast, finding solutions to *NP* problems is widely believed to be hard.
- Consider for example the problem 3-SAT. Given a 3-CNF Boolean formula, it is notoriously difficult to come up with a satisfying assignment, whereas given a proposed assignment it is trivial to plug in the values and verify its correctness. Such an assignment is an *NP*-proof for the satisfiability of the formula.

Towards a new definition of *NP*

Some comments

- What is a mathematical proof? Anything that can be verified by a rigorous procedure, i.e., an algorithm.
- A theorem = a problem.
- A proof = a solution.

Some comments

- What is a mathematical proof? Anything that can be verified by a rigorous procedure, i.e., an algorithm.
- A theorem = a problem.
- A proof = a solution.

Some comments

- What is a mathematical proof? Anything that can be verified by a rigorous procedure, i.e., an algorithm.
- A theorem = a problem.
- A proof = a solution.

Some comments

- What is a mathematical proof? Anything that can be verified by a rigorous procedure, i.e., an algorithm.
- A theorem = a problem.
- A proof = a solution.

Towards a new definition of NP

Definition (NP alternative definition)

An alternative way to define NP is as the class of all languages $L \subseteq \{0,1\}^*$ that have **efficient** proof systems: proof systems in which there is a polynomial-time algorithm that verifies correctness of the statement $x \in L$ with assistance of a proof.

- One problem with the usual proof systems (i.e. the "yes"-certificates for NP) is that these proofs are very sensitive to error. A false theorem can be "proven" by a proof that consists of only one erroneous step. Similarly, a 3-SAT formula ϕ can be unsatisfiable, yet have an assignment that satisfies all clauses but one. In these cases, the verifier must check every single proof step / clause in order to make sure that the proof is correct.

Towards a new definition of NP

Definition (NP alternative definition)

An alternative way to define NP is as the class of all languages $L \subseteq \{0,1\}^*$ that have **efficient** proof systems: proof systems in which there is a polynomial-time algorithm that verifies correctness of the statement $x \in L$ with assistance of a proof.

- One problem with the usual proof systems (i.e. the "yes"-certificates for NP) is that these proofs are very sensitive to error. A false theorem can be "proven" by a proof that consists of only one erroneous step. Similarly, a 3-SAT formula ϕ can be unsatisfiable, yet have an assignment that satisfies all clauses but one. In these cases, the verifier must check every single proof step / clause in order to make sure that the proof is correct.

Towards a new definition of NP

Definition (NP alternative definition)

An alternative way to define NP is as the class of all languages $L \subseteq \{0,1\}^*$ that have **efficient** proof systems: proof systems in which there is a polynomial-time algorithm that verifies correctness of the statement $x \in L$ with assistance of a proof.

- One problem with the usual proof systems (i.e. the "yes"-certificates for NP) is that these proofs are very sensitive to error. A false theorem can be "proven" by a proof that consists of only one erroneous step. Similarly, a 3-SAT formula ϕ can be unsatisfiable, yet have an assignment that satisfies all clauses but one. In these cases, the verifier must check every single proof step / clause in order to make sure that the proof is correct.

Towards a new definition of *NP*

- In contrast, the **PCP theorem** gives each set in *NP* an alternative proof system, in which proofs are robust.
- In this system a proof for a false statement is guaranteed to have many errors.
- As a result, a verifier can randomly read only a few bits from the proof and decide, with *high probability* of success, whether the proof is valid or not.

Towards a new definition of *NP*

- In contrast, the **PCP theorem** gives each set in *NP* an alternative proof system, in which proofs are robust.
- In this system a proof for a false statement is guaranteed to have many errors.
- As a result, a verifier can randomly read only a few bits from the proof and decide, with *high probability* of success, whether the proof is valid or not.

Towards a new definition of *NP*

- In contrast, the **PCP theorem** gives each set in *NP* an alternative proof system, in which proofs are robust.
- In this system a proof for a false statement is guaranteed to have many errors.
- As a result, a verifier can randomly read only a few bits from the proof and decide, with *high probability* of success, whether the proof is valid or not.

Towards a new definition of *NP*

Definition (*NP* revisited - The *NP* verifier)

$L \in NP$ iff there exists a poly-time TM V (the verifier) such that:

$$x \in L \Rightarrow \exists \pi \text{ such that } V^{\pi}(x) = 1,$$
$$x \notin L \Rightarrow \forall \pi,\ V^{\pi}(x) = 0.$$

(π is a proof)

Towards a new definition of NP

Definition (The PCP verifier)

Let L be a language and $q, r : \mathbb{N} \to \mathbb{N}$. We say that L has an $(r(n), q(n))$-PCP verifier if there's a polynomial-time probabilistic algorithm V satisfying:

- **Efficiency**: On input $x \in \{0,1\}^n$ and given random access to a string $\pi \in \{0,1\}^*$ of length at most $q(n)2^{r(n)}$ (the *proof*), V uses at most $r(n)$ random coins and makes at most $q(n)$ nonadaptive queries to locations of π. Then it outputs "1" (for "accept") or "0" (for "reject"). We let $V^\pi(x)$ denote the **random** variable representing V's output on input x and with random access to π.
- **Completeness**: $x \in L \Rightarrow \exists \pi \in \{0,1\}^*$ such that $Pr[V^\pi(x) = 1] = 1$. (We call this string π the correct proof for x.)
- **Soundness**: $x \notin L \Rightarrow \forall \pi \in \{0,1\}^*, Pr[V^\pi(x) = 1] \leq 1/2$.

We say that a language L is in $PCP[r(n), q(n)]$ if there are some constants $c, d > 0$ such that L has a $(cr(n), dq(n))$-PCP verifier.

Towards a new definition of NP

Definition (The PCP verifier)

Let L be a language and $q, r : \mathbb{N} \to \mathbb{N}$. We say that L has an $(r(n), q(n))$-PCP verifier if there's a polynomial-time probabilistic algorithm V satisfying:

- **Efficiency**: On input $x \in \{0,1\}^n$ and given random access to a string $\pi \in \{0,1\}^*$ of length at most $q(n)2^{r(n)}$ (the *proof*), V uses at most $r(n)$ random coins and makes at most $q(n)$ nonadaptive queries to locations of π. Then it outputs "1" (for "accept") or "0" (for "reject"). We let $V^{\pi}(x)$ denote the **random** variable representing V's output on input x and with random access to π.
- **Completeness**: $x \in L \Rightarrow \exists \pi \in \{0,1\}^*$ such that $Pr[V^{\pi}(x) = 1] = 1$. (We call this string π the correct proof for x.)
- **Soundness**: $x \notin L \Rightarrow \forall \pi \in \{0,1\}^*, Pr[V^{\pi}(x) = 1] \leq 1/2$.

We say that a language L is in $PCP[r(n), q(n)]$ if there are some constants $c, d > 0$ such that L has a $(cr(n), dq(n))$-PCP verifier.

Towards a new definition of NP

Definition (The PCP verifier)

Let L be a language and $q, r : \mathbb{N} \to \mathbb{N}$. We say that L has an $(r(n), q(n))$-PCP verifier if there's a polynomial-time probabilistic algorithm V satisfying:

- **Efficiency**: On input $x \in \{0,1\}^n$ and given random access to a string $\pi \in \{0,1\}^*$ of length at most $q(n)2^{r(n)}$ (the *proof*), V uses at most $r(n)$ random coins and makes at most $q(n)$ nonadaptive queries to locations of π. Then it outputs "1" (for "accept") or "0" (for "reject"). We let $V^\pi(x)$ denote the **random** variable representing V's output on input x and with random access to π.
- **Completeness**: $x \in L \Rightarrow \exists \pi \in \{0,1\}^*$ such that $Pr[V^\pi(x) = 1] = 1$. (We call this string π the correct proof for x.)
- **Soundness**: $x \notin L \Rightarrow \forall \pi \in \{0,1\}^*, Pr[V^\pi(x) = 1] \leq 1/2$.

We say that a language L is in $PCP[r(n), q(n)]$ if there are some constants $c, d > 0$ such that L has a $(cr(n), dq(n))$-PCP verifier.

Towards a new definition of NP

Definition (The PCP verifier)

Let L be a language and $q, r : \mathbb{N} \to \mathbb{N}$. We say that L has an $(r(n), q(n))$-PCP verifier if there's a polynomial-time probabilistic algorithm V satisfying:

- **Efficiency**: On input $x \in \{0,1\}^n$ and given random access to a string $\pi \in \{0,1\}^*$ of length at most $q(n)2^{r(n)}$ (the *proof*), V uses at most $r(n)$ random coins and makes at most $q(n)$ nonadaptive queries to locations of π. Then it outputs "1" (for "accept") or "0" (for "reject"). We let $V^\pi(x)$ denote the **random** variable representing V's output on input x and with random access to π.
- **Completeness**: $x \in L \Rightarrow \exists \pi \in \{0,1\}^* \text{ such that } Pr[V^\pi(x) = 1] = 1$. (We call this string π the correct proof for x.)
- **Soundness**: $x \notin L \Rightarrow \forall \pi \in \{0,1\}^*, Pr[V^\pi(x) = 1] \leq 1/2$.

We say that a language L is in $PCP[r(n), q(n)]$ if there are some constants $c, d > 0$ such that L has a $(cr(n), dq(n))$-PCP verifier.

Towards a new definition of *NP*

Definition (The PCP verifier)

Let L be a language and $q, r : \mathbb{N} \to \mathbb{N}$. We say that L has an $(r(n), q(n))$-PCP verifier if there's a polynomial-time probabilistic algorithm V satisfying:

- **Efficiency**: On input $x \in \{0,1\}^n$ and given random access to a string $\pi \in \{0,1\}^*$ of length at most $q(n)2^{r(n)}$ (the *proof*), V uses at most $r(n)$ random coins and makes at most $q(n)$ nonadaptive queries to locations of π. Then it outputs "1" (for "accept") or "0" (for "reject"). We let $V^\pi(x)$ denote the **random** variable representing V's output on input x and with random access to π.
- **Completeness**: $x \in L \Rightarrow \exists \pi \in \{0,1\}^* \text{ such that } Pr[V^\pi(x) = 1] = 1$. (We call this string π the correct proof for x.)
- **Soundness**: $x \notin L \Rightarrow \forall \pi \in \{0,1\}^*, Pr[V^\pi(x) = 1] \leq 1/2$.

We say that a language L is in $PCP[r(n), q(n)]$ if there are some constants $c, d > 0$ such that L has a $(cr(n), dq(n))$-PCP verifier.

The PCP Theorem

Theorem (PCP Theorem - Arora, Lund, Motwani, Sudan, Szegedy, Safra)

$NP = PCP[O(\log n), O(1)]$.

The easy direction of the PCP Theorem

Lemma

$PCP[O(\log n), O(1)] \subseteq NP$.

Proof.

On board... □

The easy direction of the PCP Theorem

Lemma

$PCP[O(\log n), O(1)] \subseteq NP$.

Proof.

On board... $\square$

The hard direction of the PCP Theorem

Lemma

$NP \subseteq PCP[O(\log n), O(1)]$.

We will definitely **not** prove this right now, all we can say is that Dinur's approach is based on finding **gap-introducing** reductions.

Gap-introducing reductions and *NP*-completeness (1 / 2)

Theorem

If there is a gap-introducing reduction for some problem L in NP, then $L \in PCP[O(\log n), O(1)]$. *In particular, if L is NP-complete then the PCP theorem holds.*

Proof.

Suppose $L \in NP$, and there is a reduction to a 3CNF formula ϕ_x with m clauses and with the following properties:

$x \in L \Rightarrow \phi_x$ is satisfiable

$x \notin L \Rightarrow$ no assignment satisfies more than $(1 - \epsilon_1)m$ clauses of ϕ_x.

We now describe how to construct a **verifier** V, given a proof w.

Gap-introducing reductions and NP-completeness (1 / 2)

Theorem

If there is a gap-introducing reduction for some problem L in NP, then $L \in PCP[O(\log n), O(1)]$. In particular, if L is NP-complete then the PCP theorem holds.

Proof.

Suppose $L \in NP$, and there is a reduction to a 3CNF formula ϕ_x with m clauses and with the following properties:

$$x \in L \Rightarrow \phi_x \text{ is satisfiable}$$

$$x \notin L \Rightarrow \text{ no assignment satisfies more than } (1 - \epsilon_1)m \text{ clauses of } \phi_x.$$

We now describe how to construct a **verifier** V, given a proof w.

Proof (Continued).

- V picks $O(\frac{1}{\epsilon_1})$ clauses of ϕ_x at random, and checks if w satisfies them all.
- $O(\frac{1}{\epsilon_1}\log m) = O(\log|x|)$ random bits used.
- Number of bits read by the verifier: $O(\frac{1}{\epsilon_1}) = O(1)$.

$x \in L \Rightarrow \phi_x$ is satisfiable

$\Rightarrow \exists w$ such that $V^w(x)$ always accept.

$x \notin L \Rightarrow \forall w$ a fraction ϵ_1 of clauses of ϕ_x are unsatisfied by w

$$\Rightarrow \forall w \; V^w(x) \text{ rejects with probability } \geq \frac{1}{2}$$

(the probability that it doesn't reject is $\leq (1-\epsilon_1)^{1/\epsilon_1} \leq 1/2$)

Gap-introducing reductions and NP-completeness (2 / 2)

Proof (Continued).

- V picks $O(\frac{1}{\epsilon_1})$ clauses of ϕ_x at random, and checks if w satisfies them all.
- $O(\frac{1}{\epsilon_1} \log m) = O(\log |x|)$ random bits used.
- Number of bits read by the verifier: $O(\frac{1}{\epsilon_1}) = O(1)$.

$$x \in L \Rightarrow \phi_x \text{ is satisfiable}$$
$$\Rightarrow \exists w \text{ such that } V^w(x) \text{ always accept.}$$

$$x \notin L \Rightarrow \forall w \text{ a fraction } \epsilon_1 \text{ of clauses of } \phi_x \text{ are unsatisfied by } w$$
$$\Rightarrow \forall w \; V^w(x) \text{ rejects with probability } \geq \frac{1}{2}$$
$$\text{(the probability that it doesn't reject is } \leq (1-\epsilon_1)^{1/\epsilon_1} \leq 1/2)$$

$\square$

Gap-introducing reductions and NP-completeness (2 / 2)

Proof (Continued).

- V picks $O\left(\frac{1}{\epsilon_1}\right)$ clauses of ϕ_x at random, and checks if w satisfies them all.
- $O\left(\frac{1}{\epsilon_1}\log m\right) = O(\log|x|)$ random bits used.
- Number of bits read by the verifier: $O\left(\frac{1}{\epsilon_1}\right) = O(1)$.

$x \in L \Rightarrow \phi_x$ is satisfiable

$\Rightarrow \exists w$ such that $V^w(x)$ always accept.

$x \notin L \Rightarrow \forall w$ a fraction ϵ_1 of clauses of ϕ_x are unsatisfied by w

$$\Rightarrow \forall w \; V^w(x) \text{ rejects with probability } \geq \frac{1}{2}$$

(the probability that it doesn't reject is $\leq (1-\epsilon_1)^{1/\epsilon_1} \leq 1/2$)

□

Proof (Continued).

- V picks $O(\frac{1}{\epsilon_1})$ clauses of ϕ_x at random, and checks if w satisfies them all.
- $O(\frac{1}{\epsilon_1}\log m) = O(\log|x|)$ random bits used.
- Number of bits read by the verifier: $O(\frac{1}{\epsilon_1}) = O(1)$.

$x \in L \Rightarrow \phi_x$ is satisfiable

$\Rightarrow \exists w$ such that $V^w(x)$ always accept.

$x \notin L \Rightarrow \forall w$ a fraction ϵ_1 of clauses of ϕ_x are unsatisfied by w

$$\Rightarrow \forall w \; V^w(x) \text{ rejects with probability } \geq \frac{1}{2}$$

(the probability that it doesn't reject is $\leq (1-\epsilon_1)^{1/\epsilon_1} \leq 1/2$) $\square$

Gap-introducing reductions and NP-completeness (2 / 2)

Proof (Continued).

- V picks $O(\frac{1}{\epsilon_1})$ clauses of ϕ_x at random, and checks if w satisfies them all.
- $O(\frac{1}{\epsilon_1} \log m) = O(\log |x|)$ random bits used.
- Number of bits read by the verifier: $O(\frac{1}{\epsilon_1}) = O(1)$.

$$x \in L \Rightarrow \phi_x \text{ is satisfiable}$$
$$\Rightarrow \exists w \text{ such that } V^w(x) \text{ always accept.}$$

$$x \notin L \Rightarrow \forall w \text{ a fraction } \epsilon_1 \text{ of clauses of } \phi_x \text{ are unsatisfied by } w$$
$$\Rightarrow \forall w\ V^w(x) \text{ rejects with probability } \geq \frac{1}{2}$$
$$(\text{the probability that it doesn't reject is } \leq (1-\epsilon_1)^{1/\epsilon_1} \leq 1/2)$$

$\square$

Outline

How can we get inapproximability results

- In general, standard NP-hardness proofs are not powerful enough to give inapproximability results.
- In order to get such a result, we will need stronger reductions, the gap-introducing reductions we have already mentioned.

How can we get inapproximability results

- In general, standard *NP*-hardness proofs are not powerful enough to give inapproximability results.
- In order to get such a result, we will need stronger reductions, the gap-introducing reductions we have already mentioned.

Approximability of Max3SAT

Theorem

The PCP theorem implies that there is an $\epsilon_1 > 0$ such that there is no polynomial $(1 - \epsilon_1)$-approximation algorithm for Max3SAT, unless $P = NP$.

Proof.

On board... □

Optimal PCP constructions for MaxSAT

Theorem (Håstad)

For every $\epsilon > 0$, $NP = PCP_{1-\epsilon, \frac{1}{2}+\epsilon}[O(\log n), 3]$. Furthermore, the verifier behaves as follows: it uses its randomness to pick three entries i, j, k in the proof w and a bit b, and it accepts iff $w_i \oplus w_j \oplus w_k = b$.

Consequences

- Through a reduction from 3SAT to MaxE3LIN-2, we get that MaxE3LIN-2 cannot be approximated within a factor better than 2, unless $P = NP$.
- Furthermore, Max3SAT cannot be approximated withn a factor better than $8/7$, unless $P = NP$.
- Finally, MaxCUT has an approximability bound of $17/16$.

Optimal PCP constructions for MaxSAT

Theorem (Håstad)

For every $\epsilon > 0$, $NP = PCP_{1-\epsilon,\frac{1}{2}+\epsilon}[O(\log n), 3]$. Furthermore, the verifier behaves as follows: it uses its randomness to pick three entries i, j, k in the proof w and a bit b, and it accepts iff $w_i \oplus w_j \oplus w_k = b$.

Consequences

- Through a reduction from 3SAT to MaxE3LIN-2, we get that MaxE3LIN-2 cannot be approximated within a factor better than 2, unless $P = NP$.
- Furthermore, Max3SAT cannot be approximated withn a factor better than 8/7, unless $P = NP$.
- Finally, MaxCUT has an approximability bound of 17/16.

Optimal PCP constructions for MaxSAT

Theorem (Håstad)

For every $\epsilon > 0$, $NP = PCP_{1-\epsilon, \frac{1}{2}+\epsilon}[O(\log n), 3]$. Furthermore, the verifier behaves as follows: it uses its randomness to pick three entries i, j, k in the proof w and a bit b, and it accepts iff $w_i \oplus w_j \oplus w_k = b$.

Consequences

- Through a reduction from 3SAT to MaxE3LIN-2, we get that MaxE3LIN-2 cannot be approximated within a factor better than 2, unless $P = NP$.
- Furthermore, Max3SAT cannot be approximated withn a factor better than $8/7$, unless $P = NP$.
- Finally, MaxCUT has an approximability bound of $17/16$.

Optimal PCP constructions for MaxSAT

Theorem (Håstad)

For every $\epsilon > 0$, $NP = PCP_{1-\epsilon,\frac{1}{2}+\epsilon}[O(\log n), 3]$. *Furthermore, the verifier behaves as follows: it uses its randomness to pick three entries* i, j, k *in the proof* w *and a bit* b*, and it accepts iff* $w_i \oplus w_j \oplus w_k = b$.

Consequences

- Through a reduction from 3SAT to MaxE3LIN-2, we get that MaxE3LIN-2 cannot be approximated within a factor better than 2, unless $P = NP$.
- Furthermore, Max3SAT cannot be approximated withn a factor better than $8/7$, unless $P = NP$.
- Finally, MaxCUT has an approximability bound of $17/16$.

Improvement on Håstad's Theorem

Theorem (Guruswami, Lewin, Sudan, Trevisan 98)

$NP = PCP_{1,\frac{1}{2}+\epsilon}[O(\log n), 3], \ \forall \epsilon > 0$

Proof of Optimality of the above result

Theorem (Karloff, Zwick 97)

$P = PCP_{1,\frac{1}{2}}[O(\log n), 3]$

Theorem (Guruswami, Lewin, Sudan, Trevisan 98)

$NP = PCP_{1,\frac{1}{2}+\epsilon}[O(\log n), 3], \ \forall \epsilon > 0$

Proof of Optimality of the above result

Theorem (Karloff, Zwick 97)

$P = PCP_{1,\frac{1}{2}}[O(\log n), 3]$

Vertex Cover and Independent Set (1 / 2)

Problem (Vertex Cover)

Given an undirected graph $G = (V, E)$, a vertex cover is a set $C \subseteq V$ such that every edge $(u, v) \in E$ has one endpoint in C. We want to find the Minimum Vertex Cover.

Problem (Independent Set)

Given an undirected graph $G = (V, E)$, an independent set is a set $S \subseteq V$ such that for every $u, v \in S$ we have $(u, v) \notin E$. We want to find the Maximum Independent Set.

- Observe that a set C is a vertex cover iff $V \setminus C$ is an independent set.
- Thus, the two problems are actually the "same".
- However, in terms of approximability, they are very different.

Vertex Cover and Independent Set (1 / 2)

Problem (Vertex Cover)

Given an undirected graph $G = (V, E)$, a vertex cover is a set $C \subseteq V$ such that every edge $(u, v) \in E$ has one endpoint in C. We want to find the Minimum Vertex Cover.

Problem (Independent Set)

Given an undirected graph $G = (V, E)$, an independent set is a set $S \subseteq V$ such that for every $u, v \in S$ we have $(u, v) \notin E$. We want to find the Maximum Independent Set.

- Observe that a set C is a vertex cover iff $V \setminus C$ is an independent set.
- Thus, the two problems are actually the "same".
- However, in terms of approximability, they are very different.

Vertex Cover and Independent Set (1 / 2)

Problem (Vertex Cover)

Given an undirected graph $G = (V, E)$, a vertex cover is a set $C \subseteq V$ such that every edge $(u, v) \in E$ has one endpoint in C. We want to find the Minimum Vertex Cover.

Problem (Independent Set)

Given an undirected graph $G = (V, E)$, an independent set is a set $S \subseteq V$ such that for every $u, v \in S$ we have $(u, v) \notin E$. We want to find the Maximum Independent Set.

- Observe that a set C is a vertex cover iff $V \setminus C$ is an independent set.
- Thus, the two problems are actually the "same".
- However, in terms of approximability, they are very different.

Vertex Cover and Independent Set (1 / 2)

Problem (Vertex Cover)

Given an undirected graph $G = (V, E)$, a vertex cover is a set $C \subseteq V$ such that every edge $(u, v) \in E$ has one endpoint in C. We want to find the Minimum Vertex Cover.

Problem (Independent Set)

Given an undirected graph $G = (V, E)$, an independent set is a set $S \subseteq V$ such that for every $u, v \in S$ we have $(u, v) \notin E$. We want to find the Maximum Independent Set.

- Observe that a set C is a vertex cover iff $V \setminus C$ is an independent set.
- Thus, the two problems are actually the "same".
- However, in terms of approximability, they are very different.

Vertex Cover and Independent Set (1 / 2)

Problem (Vertex Cover)

Given an undirected graph $G = (V, E)$, a vertex cover is a set $C \subseteq V$ such that every edge $(u, v) \in E$ has one endpoint in C. We want to find the Minimum Vertex Cover.

Problem (Independent Set)

Given an undirected graph $G = (V, E)$, an independent set is a set $S \subseteq V$ such that for every $u, v \in S$ we have $(u, v) \notin E$. We want to find the Maximum Independent Set.

- Observe that a set C is a vertex cover iff $V \setminus C$ is an independent set.
- Thus, the two problems are actually the "same".
- However, in terms of approximability, they are very different.

Vertex Cover and Independent Set (2 / 2)

Vertex Cover

- A simple algorithm (just find a maximal matching and take both endpoints) gives a 2-approximation for VC.
- It has been proved (Dinur and Safra) that VC is *NP*-hard to approximate within a factor of 1.3606.
- Assuming the Unique Games Conjecture, we get a tight $2 - o(1)$ inapproximability for VC (Khot and Regev).

Independent Set

- Assuming $ZPP \neq NP$, for every $\epsilon > 0$ there is no $n^{1-\epsilon}$-approximation algorithm for Independent Set.
- If a graph $G = (V, E)$ has maximum degree d, then a maximal independent set contains at least $|V|/(d+1)$ vertices, and so is a $(d+1)$-approximate solution.
- This can be improved to an $O(d \log\log d / \log d)$-approximation.
- It has been proved that no $(d/2^{O(\sqrt{\log d})})$-approximation algorithm exists unless $P = NP$.

Vertex Cover and Independent Set (2 / 2)

Vertex Cover

- A simple algorithm (just find a maximal matching and take both endpoints) gives a 2-approximation for VC.
- It has been proved (Dinur and Safra) that VC is *NP*-hard to approximate within a factor of 1.3606.
- Assuming the Unique Games Conjecture, we get a tight $2 - o(1)$ inapproximability for VC (Khot and Regev).

Independent Set

- Assuming $ZPP \neq NP$, for every $\epsilon > 0$ there is no $n^{1-\epsilon}$-approximation algorithm for Independent Set.
- If a graph $G = (V, E)$ has maximum degree d, then a maximal independent set contains at least $|V|/(d+1)$ vertices, and so is a $(d+1)$-approximate solution.
- This can be improved to an $O(d \log \log d / \log d)$-approximation.
- It has been proved that no $(d/2^{O(\sqrt{\log d})})$-approximation algorithm exists unless $P = NP$.

Vertex Cover and Independent Set (2 / 2)

Vertex Cover

- A simple algorithm (just find a maximal matching and take both endpoints) gives a 2-approximation for VC.
- It has been proved (Dinur and Safra) that VC is *NP*-hard to approximate within a factor of 1.3606.
- Assuming the Unique Games Conjecture, we get a tight $2 - o(1)$ inapproximability for VC (Khot and Regev).

Independent Set

- Assuming $ZPP \neq NP$, for every $\epsilon > 0$ there is no $n^{1-\epsilon}$-approximation algorithm for Independent Set.
- If a graph $G = (V, E)$ has maximum degree d, then a maximal independent set contains at least $|V|/(d+1)$ vertices, and so is a $(d+1)$-approximate solution.
- This can be improved to an $O(d \log\log d / \log d)$-approximation.
- It has been proved that no $(d/2^{O(\sqrt{\log d})})$-approximation algorithm exists unless $P = NP$.

Vertex Cover and Independent Set (2 / 2)

Vertex Cover

- A simple algorithm (just find a maximal matching and take both endpoints) gives a 2-approximation for VC.
- It has been proved (Dinur and Safra) that VC is *NP*-hard to approximate within a factor of 1.3606.
- Assuming the Unique Games Conjecture, we get a tight $2 - o(1)$ inapproximability for VC (Khot and Regev).

Independent Set

- Assuming $ZPP \neq NP$, for every $\epsilon > 0$ there is no $n^{1-\epsilon}$-approximation algorithm for Independent Set.
- If a graph $G = (V, E)$ has maximum degree d, then a maximal independent set contains at least $|V|/(d+1)$ vertices, and so is a $(d+1)$-approximate solution.
- This can be improved to an $O(d \log \log d / \log d)$-approximation.
- It has been proved that no $(d/2^{O(\sqrt{\log d})})$-approximation algorithm exists unless $P = NP$.

Vertex Cover and Independent Set (2 / 2)

Vertex Cover

- A simple algorithm (just find a maximal matching and take both endpoints) gives a 2-approximation for VC.
- It has been proved (Dinur and Safra) that VC is *NP*-hard to approximate within a factor of 1.3606.
- Assuming the Unique Games Conjecture, we get a tight $2 - o(1)$ inapproximability for VC (Khot and Regev).

Independent Set

- Assuming $ZPP \neq NP$, for every $\epsilon > 0$ there is no $n^{1-\epsilon}$-approximation algorithm for Independent Set.
- If a graph $G = (V, E)$ has maximum degree d, then a maximal independent set contains at least $|V|/(d+1)$ vertices, and so is a $(d+1)$-approximate solution.
- This can be improved to an $O(d \log\log d / \log d)$-approximation.
- It has been proved that no $(d/2^{O(\sqrt{\log d})})$-approximation algorithm exists unless $P = NP$.

Vertex Cover and Independent Set (2 / 2)

Vertex Cover

- A simple algorithm (just find a maximal matching and take both endpoints) gives a 2-approximation for VC.
- It has been proved (Dinur and Safra) that VC is *NP*-hard to approximate within a factor of 1.3606.
- Assuming the Unique Games Conjecture, we get a tight $2 - o(1)$ inapproximability for VC (Khot and Regev).

Independent Set

- Assuming $ZPP \neq NP$, for every $\epsilon > 0$ there is no $n^{1-\epsilon}$-approximation algorithm for Independent Set.
- If a graph $G = (V, E)$ has maximum degree d, then a maximal independent set contains at least $|V|/(d+1)$ vertices, and so is a $(d+1)$-approximate solution.
- This can be improved to an $O(d \log\log d / \log d)$-approximation.
- It has been proved that no $(d/2^{O(\sqrt{\log d})})$-approximation algorithm exists unless $P = NP$.

Vertex Cover and Independent Set (2 / 2)

Vertex Cover

- A simple algorithm (just find a maximal matching and take both endpoints) gives a 2-approximation for VC.
- It has been proved (Dinur and Safra) that VC is *NP*-hard to approximate within a factor of 1.3606.
- Assuming the Unique Games Conjecture, we get a tight $2 - o(1)$ inapproximability for VC (Khot and Regev).

Independent Set

- Assuming $ZPP \neq NP$, for every $\epsilon > 0$ there is no $n^{1-\epsilon}$-approximation algorithm for Independent Set.
- If a graph $G = (V, E)$ has maximum degree d, then a maximal independent set contains at least $|V|/(d+1)$ vertices, and so is a $(d+1)$-approximate solution.
- This can be improved to an $O(d \log\log d / \log d)$-approximation.
- It has been proved that no $(d/2^{O(\sqrt{\log d})})$-approximation algorithm exists unless $P = NP$.

Bibliography

1. How NP Got a New Definition: A Survey of Probabilistically Checkable Proofs.
   Sanjeev Arora. *ICM*, 2002.
2. Computational Complexity.
   Sanjeev Arora, Boaz Barak. *Cambridge University Press*, 2009.
3. Probabilistically Checkable Proofs.
   Andreas Galanis. *ECE - NTUA thesis*, 2009.
4. Probabilistically Checkable Proofs and Codes.
   Irit Dinur. *ICM*, 2010.
5. Inapproximability of Combinatorial Optimization Problems.
   Luca Trevisan. *ECCC*, 2010.

THANK YOU!