PCP & Hardness of Approximation

Vasilis Margonis

Advanced Topics in Algorithms & Complexity

$\propto \wedge \mu \forall$

May 11, 2017

Overview

Overview

Introduction

- Suppose a mathematician circulates a proof of an important result, say Riemann Hypothesis, fitting 10 thousand pages.
- To verify it would take us several years, going through all of those pages.
- **Weird question**: Can we do better than that? (e.g. ignore most part of the proof)
- **Even weirder answer**: Yes, according to the PCP theorem.

Introduction

- Suppose a mathematician circulates a proof of an important result, say Riemann Hypothesis, fitting 10 thousand pages.
- To verify it would take us several years, going through all of those pages.
- **Weird question**: Can we do better than that? (e.g. ignore most part of the proof)
- **Even weirder answer**: Yes, according to the PCP theorem.

Introduction

- Suppose a mathematician circulates a proof of an important result, say Riemann Hypothesis, fitting 10 thousand pages.
- To verify it would take us several years, going through all of those pages.
- **Weird question**: Can we do better than that? (e.g. ignore most part of the proof)
- **Even weirder answer**: Yes, according to the PCP theorem.

Introduction

- Suppose a mathematician circulates a proof of an important result, say Riemann Hypothesis, fitting 10 thousand pages.
- To verify it would take us several years, going through all of those pages.
- **Weird question**: Can we do better than that? (e.g. ignore most part of the proof)
- **Even weirder answer**: Yes, according to the PCP theorem.

The idea behind PCP

So, the mathematician can rewrite his proof in a certain format. **the PCP format**, so we can verify it by probabilistically selecting a constant number of bits to examine it. Furthermore, this verification has the following properties:

1. A correct proof will always convince us.
2. A false proof will convince us with only negligible probability (2^{-100} if we examine 300 bits). In fact, a stronger assertion is true: if the Riemann hypothesis is false, then we are guaranteed to reject any string of letters placed before us with high probability.

Note: This proof rewriting is completely mechanical (a computer could do it) and does not greatly increase its size.

The idea behind PCP

So, the mathematician can rewrite his proof in a certain format. **the PCP format**, so we can verify it by probabilistically selecting a constant number of bits to examine it. Furthermore, this verification has the following properties:

1. A correct proof will always convince us.
2. A false proof will convince us with only negligible probability (2^{-100} if we examine 300 bits). In fact, a stronger assertion is true: if the Riemann hypothesis is false, then we are guaranteed to reject any string of letters placed before us with high probability.

Note: This proof rewriting is completely mechanical (a computer could do it) and does not greatly increase its size.

The idea behind PCP

So, the mathematician can rewrite his proof in a certain format. **the PCP format**, so we can verify it by probabilistically selecting a constant number of bits to examine it. Furthermore, this verification has the following properties:

1. A correct proof will always convince us.
2. A false proof will convince us with only negligible probability (2^{-100} if we examine 300 bits). In fact, a stronger assertion is true: if the Riemann hypothesis is false, then we are guaranteed to reject any string of letters placed before us with high probability.

Note: This proof rewriting is completely mechanical (a computer could do it) and does not greatly increase its size.

The idea behind PCP

So, the mathematician can rewrite his proof in a certain format. **the PCP format**, so we can verify it by probabilistically selecting a constant number of bits to examine it. Furthermore, this verification has the following properties:

1. A correct proof will always convince us.
2. A false proof will convince us with only negligible probability (2^{-100} if we examine 300 bits). In fact, a stronger assertion is true: if the Riemann hypothesis is false, then we are guaranteed to reject any string of letters placed before us with high probability.

Note: This proof rewriting is completely mechanical (a computer could do it) and does not greatly increase its size.

The idea behind PCP

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

The idea behind PCP

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

The idea behind PCP

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

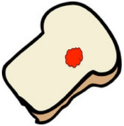
A nice analogue is the following:

The idea behind PCP

- In general, a mathematical proof is invalid if it has even a single error somewhere, which can be very difficult to detect.
- What the PCP theorem tells 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 idea behind PCP

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

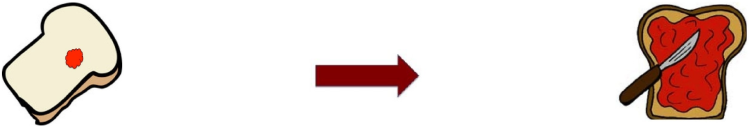
A nice analogue is the following:

Initial Proof **PCP transformation**

- In general, a mathematical proof is invalid if it has even a single error somewhere, which can be very difficult to detect.
- What the PCP theorem tells 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**



Overview

Standard definitions of NP

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

Definition (Classic definition)

$NP = \bigcup_{c \in \mathbb{N}} NTIME(n^c)$

Definition (YES-certificate definition)

A language L is in NP if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM V (called **verifier**) such that, given an input x, verifies certificates (proofs), denoted π:

$$x \in L \Rightarrow \exists \pi \in \{0,1\}^{p(|x|)} : V^{\pi}(x) = 1$$

$$x \notin L \Rightarrow \forall \pi \in \{0,1\}^{p(|x|)} : V^{\pi}(x) = 0$$

$V^{\pi}(x)$ has access to an input string x and a proof string π. If $x \in L$ and $\pi \in \{0,1\}^{p(|x|)}$ satisfy $V^{\pi}(x) = 1$, then we call π a **correct proof** for x.

Standard definitions of NP

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

Definition (Classic definition)

$$NP = \bigcup_{c \in \mathbb{N}} NTIME(n^c)$$

Definition (YES-certificate definition)

A language L is in NP if there exists a polynomial $p : \mathbb{N} \rightarrow \mathbb{N}$ and a polynomial-time TM V (called **verifier**) such that, given an input x, verifies certificates (proofs), denoted π:

$$x \in L \Rightarrow \exists \pi \in \{0,1\}^{p(|x|)} : V^{\pi}(x) = 1$$

$$x \notin L \Rightarrow \forall \pi \in \{0,1\}^{p(|x|)} : V^{\pi}(x) = 0$$

$V^{\pi}(x)$ has access to an input string x and a proof string π. If $x \in L$ and $\pi \in \{0,1\}^{p(|x|)}$ satisfy $V^{\pi}(x) = 1$, then we call π a **correct proof** for x.

Standard definitions of NP

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

Definition (Classic definition)

$NP = \bigcup_{c \in \mathbb{N}} NTIME(n^c)$

Definition (YES-certificate definition)

A language L is in NP if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM V (called **verifier**) such that, given an input x, verifies certificates (proofs), denoted π:

$$x \in L \Rightarrow \exists \pi \in \{0,1\}^{p(|x|)} : V^\pi(x) = 1$$
$$x \notin L \Rightarrow \forall \pi \in \{0,1\}^{p(|x|)} : V^\pi(x) = 0$$

$V^\pi(x)$ has access to an input string x and a proof string π. If $x \in L$ and $\pi \in \{0,1\}^{p(|x|)}$ satisfy $V^\pi(x) = 1$, then we call π a **correct proof** for x.

Towards a new characterization of NP

Definition (PCP verifier)

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

- **Efficiency**: On input a string $x \in \{0,1\}^n$ and given random access to a string $\pi \in \{0,1\}^*$ (the proof), V uses at most $r(n)$ random coins and makes at most $q(n)$ non-adaptive queries to locations of π. Then it outputs "1" (accept) or "0" (reject). We denote by $V^\pi(x)$ 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 π a 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 $L \in \mathrm{PCP}[r(n), q(n)]$, if there are some constants $c, d > 0$ such that L has a $[c \cdot r(n), d \cdot q(n)]$-PCP verifier.

Towards a new characterization of NP

Definition (PCP verifier)

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

- **Efficiency**: On input a string $x \in \{0,1\}^n$ and given random access to a string $\pi \in \{0,1\}^*$ (the proof), V uses at most $r(n)$ random coins and makes at most $q(n)$ non-adaptive queries to locations of π. Then it outputs "1" (accept) or "0" (reject). We denote by $V^\pi(x)$ 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 π a 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 $L \in \text{PCP}[r(n), q(n)]$, if there are some constants $c, d > 0$ such that L has a $[c \cdot r(n), d \cdot q(n)]$-PCP verifier.

Towards a new characterization of NP

Definition (PCP verifier)

Let L be a language and $r, q : \mathbb{N} \rightarrow \mathbb{N}$. We say that L has an $[r(n), q(n)]$-PCP verifier if there is a polynomial-time TM V satisfying:

- **Efficiency**: On input a string $x \in \{0,1\}^n$ and given random access to a string $\pi \in \{0,1\}^*$ (the proof), V uses at most $r(n)$ random coins and makes at most $q(n)$ non-adaptive queries to locations of π. Then it outputs "1" (accept) or "0" (reject). We denote by $V^\pi(x)$ 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 π a 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 $L \in \mathrm{PCP}[r(n), q(n)]$, if there are some constants $c, d > 0$ such that L has a $[c \cdot r(n), d \cdot q(n)]$-PCP verifier.

Towards a new characterization of NP

Definition (PCP verifier)

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

- **Efficiency**: On input a string $x \in \{0,1\}^n$ and given random access to a string $\pi \in \{0,1\}^*$ (the proof), V uses at most $r(n)$ random coins and makes at most $q(n)$ non-adaptive queries to locations of π. Then it outputs "1" (accept) or "0" (reject). We denote by $V^{\pi}(x)$ 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 π a 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 $L \in \text{PCP}[r(n), q(n)]$, if there are some constants $c, d > 0$ such that L has a $[c \cdot r(n), d \cdot q(n)]$-PCP verifier.

Towards a new characterization of NP

Definition (PCP verifier)

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

- **Efficiency**: On input a string $x \in \{0,1\}^n$ and given random access to a string $\pi \in \{0,1\}^*$ (the proof), V uses at most $r(n)$ random coins and makes at most $q(n)$ non-adaptive queries to locations of π. Then it outputs "1" (accept) or "0" (reject). We denote by $V^\pi(x)$ 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 π a 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 $L \in \text{PCP}[r(n), q(n)]$, if there are some constants $c, d > 0$ such that L has a $[c \cdot r(n), d \cdot q(n)]$-PCP verifier.

Towards a new characterization of NP

Notes:

1. Proofs checkable by an $[r, q]$-PCP verifier are of length at most $q2^r$. The verifier looks at only q places of the proof for any particular choice of its random coins, and there are only 2^r such choices.
2. The constant $1/2$ in the soundness condition is arbitrary, in the sense that we can execute the verifier multiple times to make the constant as small as we want.
   - For instance, if we run k times a PCP verifier with soundness of $1/2$ that uses r coins and makes q queries, it can be seen as a PCP verifier with soundness of $(1/2)^k$ that uses $(k \cdot r)$ coins and makes $(k \cdot q)$ queries.

Towards a new characterization of NP

Notes:

1. Proofs checkable by an $[r, q]$-PCP verifier are of length at most $q2^r$. The verifier looks at only q places of the proof for any particular choice of its random coins, and there are only 2^r such choices.
2. The constant $1/2$ in the soundness condition is arbitrary, in the sense that we can execute the verifier multiple times to make the constant as small as we want.
   - For instance, if we run k times a PCP verifier with soundness of $1/2$ that uses r coins and makes q queries, it can be seen as a PCP verifier with soundness of $(1/2)^k$ that uses $(k \cdot r)$ coins and makes $(k \cdot q)$ queries.

Towards a new characterization of NP

Notes:

1. Proofs checkable by an $[r, q]$-PCP verifier are of length at most $q2^r$. The verifier looks at only q places of the proof for any particular choice of its random coins, and there are only 2^r such choices.
2. The constant $1/2$ in the soundness condition is arbitrary, in the sense that we can execute the verifier multiple times to make the constant as small as we want.
   - For instance, if we run k times a PCP verifier with soundness of $1/2$ that uses r coins and makes q queries, it can be seen as a PCP verifier with soundness of $(1/2)^k$ that uses $(k \cdot r)$ coins and makes $(k \cdot q)$ queries.

The PCP theorem

Theorem (2.1 - PCP theorem - Arora, Safra, Lund, Motwani, Sudan, Szegedy)

$NP = PCP[O(logn), O(1)]$

Proof of the PCP theorem - easy direction

Lemma

$PCP[O(logn), O(1)] \subseteq NP$

Proof.

An $[r(n), q(n)]$-PCP verifier can check proofs of length at most $2^{r(n)}q(n)$. Hence, a nondeterministic machine could "guess" the proof in $2^{r(n)}q(n)$ time, and verify it deterministically by running the verifier for all $2^{r(n)}$ possible outcomes of its random coin tosses. If the verifier accepts for all these possible coin tosses then the nondeterministic machine accepts.

It follows that $PCP[r(n), q(n)] \subseteq NTIME(2^{r(n)}q(n))$.

As a special case, $PCP[O(logn), O(1)] \subseteq NTIME(2^{O(logn)} \cdot O(1)) = NP$. □

Proof of the PCP theorem - easy direction

Lemma

$PCP[O(logn), O(1)] \subseteq NP$

Proof.

An $[r(n), q(n)]$-PCP verifier can check proofs of length at most $2^{r(n)}q(n)$. Hence, a nondeterministic machine could "guess" the proof in $2^{r(n)}q(n)$ time, and verify it deterministically by running the verifier for all $2^{r(n)}$ possible outcomes of its random coin tosses. If the verifier accepts for all these possible coin tosses then the nondeterministic machine accepts.

It follows that $PCP[r(n), q(n)] \subseteq NTIME(2^{r(n)}q(n))$.

As a special case, $PCP[O(logn), O(1)] \subseteq NTIME(2^{O(logn)} \cdot O(1)) = NP$.

□

Proof of the PCP theorem - easy direction

Lemma

$PCP[O(logn), O(1)] \subseteq NP$

Proof.

An $[r(n), q(n)]$-PCP verifier can check proofs of length at most $2^{r(n)}q(n)$. Hence, a nondeterministic machine could "guess" the proof in $2^{r(n)}q(n)$ time, and verify it deterministically by running the verifier for all $2^{r(n)}$ possible outcomes of its random coin tosses. If the verifier accepts for all these possible coin tosses then the nondeterministic machine accepts.

It follows that $PCP[r(n), q(n)] \subseteq NTIME(2^{r(n)}q(n))$.

As a special case, $PCP[O(logn), O(1)] \subseteq NTIME(2^{O(logn)} \cdot O(1)) = NP$.

Proof of the PCP theorem - easy direction

Lemma

$PCP[O(logn), O(1)] \subseteq NP$

Proof.

An $[r(n), q(n)]$-PCP verifier can check proofs of length at most $2^{r(n)}q(n)$. Hence, a nondeterministic machine could "guess" the proof in $2^{r(n)}q(n)$ time, and verify it deterministically by running the verifier for all $2^{r(n)}$ possible outcomes of its random coin tosses. If the verifier accepts for all these possible coin tosses then the nondeterministic machine accepts.

It follows that $PCP[r(n), q(n)] \subseteq NTIME(2^{r(n)}q(n))$.

As a special case, $PCP[O(logn), O(1)] \subseteq NTIME(2^{O(logn)} \cdot O(1)) = NP$. □

Proof of the PCP theorem - hard direction

Lemma

$NP \subseteq PCP[O(logn), O(1)]$

We will definitely **not** prove this right now.

Overview

Motivation: Approximate solutions to NP-hard problems

- Since the discovery of *NP*-completeness in 1972, researchers tried to efficiently compute near-optimal solutions to *NP*-hard optimization problems.
- They failed to design such approximation algorithms for most problems. Then they tried to show that computing approximate solutions is also hard, but apart from a few isolated successes this effort also stalled.
- Researchers slowly began to realize that the Cook-Levin-Karp style reductions do not suffice to prove any limits on approximation algorithms.
- The PCP Theorem, not only gave a new definition of *NP*, but also provided a new starting point for reductions (the **gap**-producing reductions).

Motivation: Approximate solutions to NP-hard problems

- Since the discovery of *NP*-completeness in 1972, researchers tried to efficiently compute near-optimal solutions to *NP*-hard optimization problems.
- They failed to design such approximation algorithms for most problems. Then they tried to show that computing approximate solutions is also hard, but apart from a few isolated successes this effort also stalled.
- Researchers slowly began to realize that the Cook-Levin-Karp style reductions do not suffice to prove any limits on approximation algorithms.
- The PCP Theorem, not only gave a new definition of *NP*, but also provided a new starting point for reductions (the **gap**-producing reductions).

Motivation: Approximate solutions to NP-hard problems

- Since the discovery of *NP*-completeness in 1972, researchers tried to efficiently compute near-optimal solutions to *NP*-hard optimization problems.
- They failed to design such approximation algorithms for most problems. Then they tried to show that computing approximate solutions is also hard, but apart from a few isolated successes this effort also stalled.
- Researchers slowly began to realize that the Cook-Levin-Karp style reductions do not suffice to prove any limits on approximation algorithms.
- The PCP Theorem, not only gave a new definition of *NP*, but also provided a new starting point for reductions (the **gap**-producing reductions).

Motivation: Approximate solutions to NP-hard problems

- Since the discovery of *NP*-completeness in 1972, researchers tried to efficiently compute near-optimal solutions to *NP*-hard optimization problems.
- They failed to design such approximation algorithms for most problems. Then they tried to show that computing approximate solutions is also hard, but apart from a few isolated successes this effort also stalled.
- Researchers slowly began to realize that the Cook-Levin-Karp style reductions do not suffice to prove any limits on approximation algorithms.
- The PCP Theorem, not only gave a new definition of *NP*, but also provided a new starting point for reductions (the **gap**-producing reductions).

The hardness of approximation view

The PCP theorem states that computing near-optimal solutions for some *NP*-hard problems is no easier than computing exact solutions.

For concreteness, we focus on *MAX-3SAT*. We begin by defining what an ρ-approximation algorithm for *MAX-3SAT* is.

Definition (Approximation of *MAX-3SAT*)

For every $3CNF$ formula ϕ, the **value** of ϕ (denoted $val(\phi)$), is the maximum fraction of clauses that can satisfied by any assignment to ϕ's variables. In particular, ϕ is satisfiable iff $val(\phi) = 1$.

Let $\rho < 1$. An algorithm A is an ρ-approximation algorithm for *MAX-3SAT* if for every $3CNF$ formula ϕ with m clauses, $A(\phi)$ outputs an assignment satisfying at least $(\rho \cdot val(\phi) \cdot m)$ clauses of ϕ.

The hardness of approximation view

The PCP theorem states that computing near-optimal solutions for some *NP*-hard problems is no easier than computing exact solutions.

For concreteness, we focus on *MAX*-3*SAT*. We begin by defining what an ρ-approximation algorithm for *MAX*-3*SAT* is.

Definition (Approximation of *MAX*-3*SAT*)

For every 3*CNF* formula ϕ, the **value** of ϕ (denoted $val(\phi)$), is the maximum fraction of clauses that can satisfied by any assignment to ϕ's variables. In particular, ϕ is satisfiable iff $val(\phi) = 1$.

Let $\rho < 1$. An algorithm A is an ρ-approximation algorithm for *MAX*-3*SAT* if for every 3*CNF* formula ϕ with m clauses, $A(\phi)$ outputs an assignment satisfying at least $(\rho \cdot val(\phi) \cdot m)$ clauses of ϕ.

The hardness of approximation view

The PCP theorem states that computing near-optimal solutions for some *NP*-hard problems is no easier than computing exact solutions.

For concreteness, we focus on *MAX*-3*SAT*. We begin by defining what an ρ-approximation algorithm for *MAX*-3*SAT* is.

Definition (Approximation of *MAX*-3*SAT*)

For every 3*CNF* formula ϕ, the **value** of ϕ (denoted $val(\phi)$), is the maximum fraction of clauses that can satisfied by any assignment to ϕ's variables. In particular, ϕ is satisfiable iff $val(\phi) = 1$.

Let $\rho < 1$. An algorithm A is an ρ-approximation algorithm for *MAX*-3*SAT* if for every 3*CNF* formula ϕ with m clauses, $A(\phi)$ outputs an assignment satisfying at least $(\rho \cdot val(\phi) \cdot m)$ clauses of ϕ.

The hardness of approximation view

- Until 1992, we did not know whether or not MAX-$3SAT$ has a polynomial-time ρ-approximation algorithm for **every** $\rho < 1$.
- It turns out that the PCP Theorem means that the answer is NO (unless $P = NP$). The reason is that it can be equivalently stated as follows:

Theorem (3.1 - PCP theorem: Hardness of approximation view)

There exists $\rho < 1$ such that $\forall L \in NP$ there is a polynomial-time function f mapping strings to $3CNF$ formulas such that:

$$x \in L \Rightarrow val(f(x)) = 1 \quad (1)$$

$$x \notin L \Rightarrow val(f(x)) < \rho \quad (2)$$

The hardness of approximation view

- Until 1992, we did not know whether or not *MAX-3SAT* has a polynomial-time ρ-approximation algorithm for **every** $\rho < 1$.
- It turns out that the PCP Theorem means that the answer is NO (unless $P = NP$). The reason is that it can be equivalently stated as follows:

Theorem (3.1 - PCP theorem: Hardness of approximation view)

There exists $\rho < 1$ such that $\forall L \in NP$ there is a polynomial-time function f mapping strings to 3CNF formulas such that:

$$x \in L \Rightarrow val(f(x)) = 1 \quad (1)$$

$$x \notin L \Rightarrow val(f(x)) < \rho \quad (2)$$

The hardness of approximation view

- Until 1992, we did not know whether or not MAX-$3SAT$ has a polynomial-time ρ-approximation algorithm for **every** $\rho < 1$.
- It turns out that the PCP Theorem means that the answer is NO (unless $P = NP$). The reason is that it can be equivalently stated as follows:

Theorem (3.1 - PCP theorem: Hardness of approximation view)

There exists $\rho < 1$ such that $\forall L \in NP$ there is a polynomial-time function f mapping strings to $3CNF$ formulas such that:

$$x \in L \Rightarrow val(f(x)) = 1 \quad (1)$$

$$x \notin L \Rightarrow val(f(x)) < \rho \quad (2)$$

The hardness of approximation view

Hence, theorem 3.1 immediately implies the following corollary.

Corollary

There exists some constant $\rho < 1$ such that there is no polynomial-time ρ-approximation algorithm for MAX-$3SAT$, unless $P = NP$.

- Indeed, we can convert a ρ-approximation algorithm A for MAX-$3SAT$ into an algorithm deciding L.
- We apply the reduction f on x and then run the approximation algorithm to the resultant $3CNF$ formula $f(x)$.
- (1) and (2) together imply that $x \in L$ iff $A(f(x))$ returns an assignment that satisfies <u>at least</u> a ρ fraction of $f(x)$'s clauses.

The hardness of approximation view

Hence, theorem 3.1 immediately implies the following corollary.

Corollary

There exists some constant $\rho < 1$ such that there is no polynomial-time ρ-approximation algorithm for MAX-3SAT, unless $P = NP$.

- Indeed, we can convert a ρ-approximation algorithm A for MAX-3SAT into an algorithm deciding L.
- We apply the reduction f on x and then run the approximation algorithm to the resultant $3CNF$ formula $f(x)$.
- (1) and (2) together imply that $x \in L$ iff $A(f(x))$ returns an assignment that satisfies at least a ρ fraction of $f(x)$'s clauses.

The hardness of approximation view

Hence, theorem 3.1 immediately implies the following corollary.

Corollary

There exists some constant $\rho < 1$ such that there is no polynomial-time ρ-approximation algorithm for MAX-$3SAT$, unless $P = NP$.

- Indeed, we can convert a ρ-approximation algorithm A for MAX-$3SAT$ into an algorithm deciding L.
- We apply the reduction f on x and then run the approximation algorithm to the resultant $3CNF$ formula $f(x)$.
- (1) and (2) together imply that $x \in L$ iff $A(f(x))$ returns an assignment that satisfies at least a ρ fraction of $f(x)$'s clauses.

The hardness of approximation view

Hence, theorem 3.1 immediately implies the following corollary.

Corollary

There exists some constant $\rho < 1$ such that there is no polynomial-time ρ-approximation algorithm for MAX-3SAT, unless $P = NP$.

- Indeed, we can convert a ρ-approximation algorithm A for MAX-$3SAT$ into an algorithm deciding L.
- We apply the reduction f on x and then run the approximation algorithm to the resultant $3CNF$ formula $f(x)$.
- (1) and (2) together imply that $x \in L$ iff $A(f(x))$ returns an assignment that satisfies at least a ρ fraction of $f(x)$'s clauses.

Equivalence of the two views

- To show the equivalence of the "proof view" and the "hardness of approximation view" of the PCP theorem, we first introduce the notion of **Constrained Satisfaction Problems** (CSP).
- We will then prove the equivalence of the two views by showing that they are both equivalent to the *NP*-hardness of a certain **gap** version of *CSP*.

Equivalence of the two views

- To show the equivalence of the "proof view" and the "hardness of approximation view" of the PCP theorem, we first introduce the notion of **Constrained Satisfaction Problems** (CSP).
- We will then prove the equivalence of the two views by showing that they are both equivalent to the *NP*-hardness of a certain **gap** version of *CSP*.

Constrained Satisfaction Problems

Definition (CSP)

Let $q \in \mathbb{N}$, a *qCSP* instance $\phi = \{\phi_1, \ldots, \phi_m\}$ is a collection of functions (called **constraints**). where $\phi_i : \{0,1\}^n \to \{0,1\}$, such that each function ϕ_i depends on at most q of its input locations.

We say that $u \in \{0,1\}^n$ satisfies constraint ϕ_i, if $\phi_i(u) = 1$. The fraction of the constraints satisfied by u is $\left(\frac{\sum_{i=1}^{m} \phi_i(u)}{m}\right)$, and we let $val(\phi)$ denote the maximum is this value over all $u \in \{0,1\}^n$. We say that ϕ is satisfiable if $val(\phi) = 1$ and we call q the **arity** of ϕ.

Notes:

- We define the size of a *qCSP* instance ϕ to be m, the number of constraints.
- Because variables not used by any constraints are redundant, we always assume $n \leq qm$.

Constrained Satisfaction Problems

Definition (CSP)

Let $q \in \mathbb{N}$, a *qCSP* instance $\phi = \{\phi_1, \ldots, \phi_m\}$ is a collection of functions (called **constraints**). where $\phi_i : \{0,1\}^n \rightarrow \{0,1\}$, such that each function ϕ_i depends on at most q of its input locations.

We say that $u \in \{0,1\}^n$ satisfies constraint ϕ_i, if $\phi_i(u) = 1$. The fraction of the constraints satisfied by u is $\left(\frac{\sum_{i=1}^{m} \phi_i(u)}{m}\right)$, and we let $val(\phi)$ denote the maximum is this value over all $u \in \{0,1\}^n$. We say that ϕ is satisfiable if $val(\phi) = 1$ and we call q the **arity** of ϕ.

Notes:

- We define the size of a *qCSP* instance ϕ to be m, the number of constraints.
- Because variables not used by any constraints are redundant, we always assume $n \leq qm$.

The gap version of *CSP*

Definition (ρ-*GAPqCSP*)

Let $q \in \mathbb{N}$, $\rho < 1$. We define ρ-*GAPqCSP* to be the problem of determining for a given *qCSP* instance ϕ whether:

- $val(\phi) = 1$ (ϕ is a YES-instance of ρ-*GAPqCSP*)
- $val(\phi) < \rho$ (ϕ is a NO-instance of ρ-*GAPqCSP*)

We say that ρ-*GAPqCSP* is *NP*-hard if $\forall L \in NP$ there is a polynomial-time function f mapping strings to *qCSP* instances satisfying:

- **Completeness**: $x \in L \Rightarrow val(f(x)) = 1$
- **Soundness**: $x \notin L \Rightarrow val(f(x)) < \rho$

Theorem (3.2 - *NP*-hardness of ρ-*GAPqCSP*)

There exists $q \in \mathbb{N}$, $\rho < 1$ such that ρ-GAPqCSP is NP-hard.

The gap version of CSP

Definition (ρ-$GAPqCSP$)

Let $q \in \mathbb{N}$, $\rho < 1$. We define ρ-$GAPqCSP$ to be the problem of determining for a given $qCSP$ instance ϕ whether:

- $val(\phi) = 1$ (ϕ is a YES-instance of ρ-$GAPqCSP$)
- $val(\phi) < \rho$ (ϕ is a NO-instance of ρ-$GAPqCSP$)

We say that ρ-$GAPqCSP$ is NP-hard if $\forall L \in NP$ there is a polynomial-time function f mapping strings to $qCSP$ instances satisfying:

- **Completeness**: $x \in L \Rightarrow val(f(x)) = 1$
- **Soundness**: $x \notin L \Rightarrow val(f(x)) < \rho$

Theorem (3.2 - NP-hardness of ρ-$GAPqCSP$)

There exists $q \in \mathbb{N}$, $\rho < 1$ such that ρ-$GAPqCSP$ is NP-hard.

The gap version of *CSP*

Definition (ρ-*GAPqCSP*)

Let $q \in \mathbb{N}$, $\rho < 1$. We define ρ-*GAPqCSP* to be the problem of determining for a given *qCSP* instance ϕ whether:

- $val(\phi) = 1$ (ϕ is a YES-instance of ρ-*GAPqCSP*)
- $val(\phi) < \rho$ (ϕ is a NO-instance of ρ-*GAPqCSP*)

We say that ρ-*GAPqCSP* is *NP*-hard if $\forall L \in NP$ there is a polynomial-time function f mapping strings to *qCSP* instances satisfying:

- **Completeness**: $x \in L \Rightarrow val(f(x)) = 1$
- **Soundness**: $x \notin L \Rightarrow val(f(x)) < \rho$

Theorem (3.2 - *NP*-hardness of ρ-*GAPqCSP*)

There exists $q \in \mathbb{N}$, $\rho < 1$ such that ρ-GAPqCSP is NP-hard.

Theorem 2.1 $\equiv$ Theorem 3.2 (1/2)

We will show that theorems 2.1, 3.1 and 3.2 are all equivalent to one another. We begin by proving that Theorem 2.1 $\equiv$ Theorem 3.2.

($\Rightarrow$).

Assume that $NP \subseteq PCP[O(logn), O(1)]$. We will show that 1/2-$GAPqCSP$ is NP-hard for some q, through a reduction from some $L \in NP$. Under our assumption, L has a $[clogn, q]$-PCP verifier. Let x be the input of the verifier and $r \in \{0,1\}^{clogn}$ an outcome of a random coin toss. Define $V_{x,r}(\pi) = 1$ if $V^{\pi}(x) = 1$ for the coin toss r. Note that $V_{x,r}(\pi)$ depends on at most q bits of the proof π. Hence, $\phi = \{V_{x,r}\}_{r \in \{0,1\}^{clogn}}$ is a polynomial-sized instance of $qCSP$. Furthermore, since V runs in polynomial-time, the transformation from x to ϕ can also be carried out in polynomial-time. By the completeness and soundness of the PCP-verifier, if $x \in L$ then $val(\phi) = 1$, while if $x \notin L$ then $val(\phi) < 1/2$. □

Theorem 2.1 $\equiv$ Theorem 3.2 (2/2)

($\Leftarrow$).

Suppose that ρ-$GAPqCSP$ is NP-hard for some constants q and $\rho < 1$. Then this easily translate into a PCP-verifier with logarithmic randomness, q queries and ρ soundness for any language L:

Given an input x, the verifier will run the reduction $f(x)$ to obtain a $qCSP$ instance $\phi = \{\phi_1, \ldots, \phi_m\}$. It will expect the proof π to be an assignment to the variables of ϕ, which it will verify by choosing a random $i \in [m]$ and checking that ϕ_i is satisfied (by making queries). Clearly, if $x \in L$ then the verifier will accept with probability 1, while if $x \notin L$ it will accept with probability at most ρ.

The soundness can be boosted to $1/2$ at the expense of a constant factor in the randomness and number of queries. □

Review of the equivalence

Theorem 2.1	**Theorem 3.2**
PCP verifier (V)	CSP instance (ϕ)
Proof (π)	Assignment to variables (u)
Length of proof	Number of variables (n)
Number of queries (q)	Arity of constraints (q)
Number of random bits (r)	Logarithm of number of constraints ($logm$)
Soundness parameter	Maximum of $val(\phi)$ for a NO instrance
$NP \subseteq PCP[O(logn), O(1)]$	ρ-$GAPqCSP$ is NP-hard

Theorem 3.1 $\equiv$ Theorem 3.2 (1/3)

Now we will prove that theorem 3.1 is equivalent to theorem 3.2.

($\Rightarrow$).

Since $3CNF$ formulas are a special case $3CSP$ instances, theorem 3.1 implies theorem 3.2. □

($\Leftarrow$).

Let $\varepsilon > 0$ and $q \in \mathbb{N}$ be such that by theorem 3.2, $(1 - \varepsilon)$-$GAPqCSP$ is NP-hard. Let ϕ be a $qCSP$ instance over n variables with m constraints. Each constraint ϕ_i of ϕ can be expressed as an AND of at most 2^q clauses, where each clause is the OR of at most q variables (or their negations). Let ϕ' denote the collection of at most $m2^q$ clauses corresponding to all the constraints of ϕ.

- If ϕ is a YES-instance of $(1 - \varepsilon)$-$GAPqCSP$, then there exists an assignment satisfying all the clauses of ϕ'.

Theorem 3.1 ≡ Theorem 3.2 (1/3)

Now we will prove that theorem 3.1 is equivalent to theorem 3.2.

(⇒).

Since $3CNF$ formulas are a special case $3CSP$ instances, theorem 3.1 implies theorem 3.2. □

(⇐).

Let $\varepsilon > 0$ and $q \in \mathbb{N}$ be such that by theorem 3.2, $(1-\varepsilon)$-$GAPqCSP$ is NP-hard. Let ϕ be a $qCSP$ instance over n variables with m constraints. Each constraint ϕ_i of ϕ can be expressed as an AND of at most 2^q clauses, where each clause is the OR of at most q variables (or their negations). Let ϕ' denote the collection of at most $m2^q$ clauses corresponding to all the constraints of ϕ.

- If ϕ is a YES-instance of $(1-\varepsilon)$-$GAPqCSP$, then there exists an assignment satisfying all the clauses of ϕ'.

Theorem 3.1 $\equiv$ Theorem 3.2 (1/3)

Now we will prove that theorem 3.1 is equivalent to theorem 3.2.

($\Rightarrow$).

Since $3CNF$ formulas are a special case $3CSP$ instances, theorem 3.1 implies theorem 3.2. □

($\Leftarrow$).

Let $\varepsilon > 0$ and $q \in \mathbb{N}$ be such that by theorem 3.2, $(1-\varepsilon)$-$GAPqCSP$ is NP-hard. Let ϕ be a $qCSP$ instance over n variables with m constraints. Each constraint ϕ_i of ϕ can be expressed as an AND of at most 2^q clauses, where each clause is the OR of at most q variables (or their negations). Let ϕ' denote the collection of at most $m2^q$ clauses corresponding to all the constraints of ϕ.

- If ϕ is a YES-instance of $(1-\varepsilon)$-$GAPqCSP$, then there exists an assignment satisfying all the clauses of ϕ'.

Theorem 3.1 $\equiv$ Theorem 3.2 (2/3)

($\Leftarrow$), Cont'd.

- If ϕ is a NO-instance of $(1-\varepsilon)$-*GAPqCSP*, then every assignment violates at least an ε fraction of the constraints of ϕ, and hence at least an $\frac{\varepsilon}{2^q}$ fraction of the constraints of ϕ'.

We can use the Cook-Levin technique to transform any clause C on q variables $u_1, \ldots, u_q$ to a set $C_1, \ldots, C_q$ of clauses over the variables $u1, \ldots, u_q$ and additional auxiliary variables $y_1, \ldots, y_q$ such that:

1. Each clause C_i is the OR of at most three variables or their negations.
2. if $u_1, \ldots, u_q$ satisfy C then there is an assignment to $y_1, \ldots, y_q$ such that $u_1, \ldots, u_q, y_1, \ldots, y_q$ simultaneously satisfy $C_1, \ldots, C_q$.
3. if $u_1, \ldots, u_q$ does not satisfy C then for every assignment to $y_1, \ldots, y_q$, there is some clause C_i that is not satisfied by $u_1, \ldots, u_q, y_1, \ldots, y_q$.

Theorem 3.1 $\equiv$ Theorem 3.2 (3/3)

($\Leftarrow$), Cont'd.

Let ϕ'' denote the collection of at most $qm2^q$ clauses over the $n + qm2^q$ variables obtained in this way from ϕ'. Note that ϕ'' is a $3SAT$ formula. Our reduction will map ϕ to ϕ''.

- **Completeness** holds since if ϕ were satisfiable, then so would be ϕ', and hence ϕ''.
- **Soundness** holds since if every assignment violates at least an ε fraction of the constraints of ϕ, then every assignment violates at least an $\frac{\varepsilon}{2^q}$ fraction of the constraints of ϕ', and so every assignment violates at least an $\frac{\varepsilon}{q2^q}$ fraction of the constraints of ϕ''.

□

Consequences

MAX-3SAT is a central problem in the study of hardness of approximation. Once we have proved its inapproximability, other inapproximability results easily follow. For example:

- There is some $\rho < 1$ such that if there no polynomial-time ρ-approximation algorithm for *VERTEX-COVER*, unless $P = NP$.
- For every $\rho < 1$ if there no polynomial-time ρ-approximation algorithm for *INDSET*, unless $P = NP$.

Note that the inapproximability result for *INDSET* is much stronger than the result for *VERTEX-COVER*.

Consequences

MAX-3SAT is a central problem in the study of hardness of approximation. Once we have proved its inapproximability, other inapproximability results easily follow. For example:

- There is some $\rho < 1$ such that if there no polynomial-time ρ-approximation algorithm for *VERTEX-COVER*, unless $P = NP$.
- For every $\rho < 1$ if there no polynomial-time ρ-approximation algorithm for *INDSET*, unless $P = NP$.

Note that the inapproximability result for *INDSET* is much stronger than the result for *VERTEX-COVER*.

Consequences

MAX-3SAT is a central problem in the study of hardness of approximation. Once we have proved its inapproximability, other inapproximability results easily follow. For example:

- There is some $\rho < 1$ such that if there no polynomial-time ρ-approximation algorithm for *VERTEX-COVER*, unless $P = NP$.
- For every $\rho < 1$ if there no polynomial-time ρ-approximation algorithm for *INDSET*, unless $P = NP$.

Note that the inapproximability result for *INDSET* is much stronger than the result for *VERTEX-COVER*.

Consequences

MAX-3SAT is a central problem in the study of hardness of approximation. Once we have proved its inapproximability, other inapproximability results easily follow. For example:

- There is some $\rho < 1$ such that if there no polynomial-time ρ-approximation algorithm for *VERTEX-COVER*, unless $P = NP$.
- For every $\rho < 1$ if there no polynomial-time ρ-approximation algorithm for *INDSET*, unless $P = NP$.

Note that the inapproximability result for *INDSET* is much stronger than the result for *VERTEX-COVER*.

Overview

Asking questions

- We proved that there exists some $\rho < 1$ such that there is no polynomial-time ρ-approximation algorithm for MAX-$3SAT$, unless $P = NP$.
- But can we calculate that ρ ?
- There is a simple polynomial-time greedy algorithm that achieves an approximation ratio of $1/2$.
- Karloff and Zwick used semidefinite programming to design a polynomial-time $(7/8 - \varepsilon)$-approximation algorithm for every $\varepsilon > 0$.
- Can we do better than $7/8$?
- Håstad proved that the answer is NO (unless $P = NP$, of course).

Asking questions

- We proved that there exists some $\rho < 1$ such that there is no polynomial-time ρ-approximation algorithm for MAX-$3SAT$, unless $P = NP$.
- But can we calculate that ρ ?
- There is a simple polynomial-time greedy algorithm that achieves an approximation ratio of $1/2$.
- Karloff and Zwick used semidefinite programming to design a polynomial-time $(7/8 - \varepsilon)$-approximation algorithm for every $\varepsilon > 0$.
- Can we do better than $7/8$?
- Håstad proved that the answer is NO (unless $P = NP$, of course).

Asking questions

- We proved that there exists some $\rho < 1$ such that there is no polynomial-time ρ-approximation algorithm for MAX-$3SAT$, unless $P = NP$.
- But can we calculate that ρ ?
- There is a simple polynomial-time greedy algorithm that achieves an approximation ratio of $1/2$.
- Karloff and Zwick used semidefinite programming to design a polynomial-time $(7/8 - \varepsilon)$-approximation algorithm for every $\varepsilon > 0$.
- Can we do better than $7/8$?
- Håstad proved that the answer is NO (unless $P = NP$, of course).

Asking questions

- We proved that there exists some $\rho < 1$ such that there is no polynomial-time ρ-approximation algorithm for $MAX\text{-}3SAT$, unless $P = NP$.
- But can we calculate that ρ ?
- There is a simple polynomial-time greedy algorithm that achieves an approximation ratio of $1/2$.
- Karloff and Zwick used semidefinite programming to design a polynomial-time $(7/8 - \varepsilon)$-approximation algorithm for every $\varepsilon > 0$.
- Can we do better than $7/8$?
- Håstad proved that the answer is NO (unless $P = NP$, of course).

Asking questions

- We proved that there exists some $\rho < 1$ such that there is no polynomial-time ρ-approximation algorithm for MAX-$3SAT$, unless $P = NP$.
- But can we calculate that ρ ?
- There is a simple polynomial-time greedy algorithm that achieves an approximation ratio of $1/2$.
- Karloff and Zwick used semidefinite programming to design a polynomial-time $(7/8 - \varepsilon)$-approximation algorithm for every $\varepsilon > 0$.
- Can we do better than $7/8$?
- Håstad proved that the answer is NO (unless $P = NP$, of course).

Asking questions

- We proved that there exists some $\rho < 1$ such that there is no polynomial-time ρ-approximation algorithm for *MAX-3SAT*, unless $P = NP$.
- But can we calculate that ρ ?
- There is a simple polynomial-time greedy algorithm that achieves an approximation ratio of $1/2$.
- Karloff and Zwick used semidefinite programming to design a polynomial-time $(7/8 - \varepsilon)$-approximation algorithm for every $\varepsilon > 0$.
- Can we do better than $7/8$?
- Håstad proved that the answer is NO (unless $P = NP$, of course).

Only 3 bits ?

The optimal inapproximability result for *MAX-3SAT* is based on the following PCP construction:

Theorem (Håstad, 1997)

$NP = PCP_{1-\varepsilon, \frac{1}{2}+\varepsilon}[O(logn), 3]$, $\forall \varepsilon > 0$

Moreover, the tests used by V are linear: Given a proof $\pi \in \{0,1\}^m$, *V chooses a triple* $(i,j,k) \in [m]^3$ *and a bit* $b \in \{0,1\}$ *according to some distribution and accepts iff* $\pi_i \oplus \pi_j \oplus \pi_k = b$.

3-bit PCP and *MAX-E3LIN*

- Håstad's 3-bit PCP is intimately connected to the hardness of approximating a problem called *MAX-E3LIN*.
- *MAX-E3LIN* is a subcase of $3CSP$ in which the constraints specify the parity of triples of variables.
- We are interested in determining the largest subset of equations that are simultaneously satisfiable.

Corollary

Håstad's Theorem implies that $(1/2 + \nu)$-approximation to MAX-E3LIN is NP-hard for every $\nu > 0$.

- This is a threshold result since *MAX-E3LIN* has a simple 1/2-approximation algorithm.

3-bit PCP and *MAX-E3LIN*

- Håstad's 3-bit PCP is intimately connected to the hardness of approximating a problem called *MAX-E3LIN*.
- *MAX-E3LIN* is a subcase of $3CSP$ in which the constraints specify the parity of triples of variables.
- We are interested in determining the largest subset of equations that are simultaneously satisfiable.

Corollary

Håstad's Theorem implies that $(1/2+\nu)$-approximation to MAX-E3LIN is NP-hard for every $\nu > 0$.

- This is a threshold result since *MAX-E3LIN* has a simple 1/2-approximation algorithm.

3-bit PCP and *MAX-E3LIN*

- Håstad's 3-bit PCP is intimately connected to the hardness of approximating a problem called *MAX-E3LIN*.
- *MAX-E3LIN* is a subcase of 3*CSP* in which the constraints specify the parity of triples of variables.
- We are interested in determining the largest subset of equations that are simultaneously satisfiable.

Corollary

Håstad's Theorem implies that $(1/2+\nu)$*-approximation to MAX-E3LIN is NP-hard for every* $\nu > 0$.

- This is a threshold result since *MAX-E3LIN* has a simple 1/2-approximation algorithm.

3-bit PCP and *MAX-E3LIN*

- Håstad's 3-bit PCP is intimately connected to the hardness of approximating a problem called *MAX-E3LIN*.
- *MAX-E3LIN* is a subcase of $3CSP$ in which the constraints specify the parity of triples of variables.
- We are interested in determining the largest subset of equations that are simultaneously satisfiable.

Corollary

Håstad's Theorem implies that $(1/2+\nu)$*-approximation to MAX-E3LIN is NP-hard for every* $\nu > 0$*.*

- This is a threshold result since *MAX-E3LIN* has a simple $1/2$-approximation algorithm.

3-bit PCP and *MAX-E3LIN*

- Håstad's 3-bit PCP is intimately connected to the hardness of approximating a problem called *MAX-E3LIN*.
- *MAX-E3LIN* is a subcase of $3CSP$ in which the constraints specify the parity of triples of variables.
- We are interested in determining the largest subset of equations that are simultaneously satisfiable.

Corollary

Håstad's Theorem implies that $(1/2+\nu)$-approximation to MAX-E3LIN is NP-hard for every $\nu > 0$.

- This is a threshold result since *MAX-E3LIN* has a simple $1/2$-approximation algorithm.

Hardness of approximating *MAX-3SAT* (1/2)

Corollary

For every $\varepsilon > 0$, $(7/8 + \varepsilon)$-approximation to MAX-3SAT is NP-hard.

Proof.

- We reduce *MAX-E3LIN* to *MAX-3SAT*.
- Take an instance of *MAX-E3LIN*, where we are interested in determining whether $(1 - \nu)$ fraction of the equations can be satisfied or at most $(1/2 + \nu)$ are.
- Represent each linear constraint by four 3*CNF* clauses in the obvious way. For example, the linear constraint $x \oplus y \oplus z = 0$ is equivalent to the clauses $(\overline{x} \vee y \vee z)$, $(x \vee \overline{y} \vee z)$, $(x \vee y \vee \overline{z})$, $(\overline{x} \vee \overline{y} \vee \overline{z})$.
- If x, y, z satisfy the linear constraint, then they satisfy all four clauses. Otherwise, they satisfy three clauses.

Hardness of approximating *MAX-3SAT* (2/2)

Proof (Cont'd).

Conclusion:

- In one case at least $(1 - \frac{\nu}{4})$ fraction of clauses are simultaneously satisfiable.
- In the other case at most $1 - (\frac{1}{2} - \nu) \times \frac{\nu}{4} = \frac{7}{8} + \frac{\nu}{4}$ fraction of clauses are simultaneously satisfiable.
- Since distinguishing between the two cases is *NP*-hard, we conclude that it is *NP*-hard to compute a ρ-approximation to *MAX-3SAT* where $\rho = 7/8 + \nu/4$.
- As ν decreases, ρ can be arbitrarily close to $7/8$, and hence $(7/8 + \varepsilon)$-approximation is *NP*-hard for every $\varepsilon > 0$.

$\square$

Overview

Vertex Cover & Independent Set

Vertex Cover:

- A simple algorithm (just find a maximal matching and take both endpoints) gives a 2-approximation for *VC*.
- *VC* is *NP*-hard to approximate within a factor of 1.3606. [Dinur & Safra, 2005]
- If UGC is true, *VC* cannot be approximated within any constant factor better than 2. [Khot & Regev, 2008]

Independent Set:

- There is a completely trivial $(1/n)$-approximation algorithm to the problem: return any vertex of the graph.
- For every $\varepsilon > 0$ there is no $(1/n^{1-\varepsilon})$-approximation algorithm for *IS*. [Zuckerman, 2007]
- No $(2^{O(\sqrt{\log d})}/d)$-approximation algorithm exists, where d is the graph's maximum degree. [Trevisan, 2001]

Vertex Cover & Independent Set

Vertex Cover:

- A simple algorithm (just find a maximal matching and take both endpoints) gives a 2-approximation for *VC*.
- *VC* is *NP*-hard to approximate within a factor of 1.3606. [Dinur & Safra, 2005]
- If UGC is true, *VC* cannot be approximated within any constant factor better than 2. [Khot & Regev, 2008]

Independent Set:

- There is a completely trivial $(1/n)$-approximation algorithm to the problem: return any vertex of the graph.
- For every $\varepsilon > 0$ there is no $(1/n^{1-\varepsilon})$-approximation algorithm for *IS*. [Zuckerman, 2007]
- No $(2^{O(\sqrt{\log d})}/d)$-approximation algorithm exists, where d is the graph's maximum degree. [Trevisan, 2001]

Vertex Cover & Independent Set

Vertex Cover:

- A simple algorithm (just find a maximal matching and take both endpoints) gives a 2-approximation for *VC*.
- *VC* is *NP*-hard to approximate within a factor of 1.3606. [Dinur & Safra, 2005]
- If UGC is true, *VC* cannot be approximated within any constant factor better than 2. [Khot & Regev, 2008]

Independent Set:

- There is a completely trivial $(1/n)$-approximation algorithm to the problem: return any vertex of the graph.
- For every $\varepsilon > 0$ there is no $(1/n^{1-\varepsilon})$-approximation algorithm for *IS*. [Zuckerman, 2007]
- No $(2^{O(\sqrt{\log d})}/d)$-approximation algorithm exists, where d is the graph's maximum degree. [Trevisan, 2001]

Vertex Cover & Independent Set

Vertex Cover:

- A simple algorithm (just find a maximal matching and take both endpoints) gives a 2-approximation for *VC*.
- *VC* is *NP*-hard to approximate within a factor of 1.3606. [Dinur & Safra, 2005]
- If UGC is true, *VC* cannot be approximated within any constant factor better than 2. [Khot & Regev, 2008]

Independent Set:

- There is a completely trivial $(1/n)$-approximation algorithm to the problem: return any vertex of the graph.
- For every $\varepsilon > 0$ there is no $(1/n^{1-\varepsilon})$-approximation algorithm for *IS*. [Zuckerman, 2007]
- No $(2^{O(\sqrt{\log d})}/d)$-approximation algorithm exists, where d is the graph's maximum degree. [Trevisan, 2001]

Vertex Cover & Independent Set

Vertex Cover:

- A simple algorithm (just find a maximal matching and take both endpoints) gives a 2-approximation for *VC*.
- *VC* is *NP*-hard to approximate within a factor of 1.3606. [Dinur & Safra, 2005]
- If UGC is true, *VC* cannot be approximated within any constant factor better than 2. [Khot & Regev, 2008]

Independent Set:

- There is a completely trivial $(1/n)$-approximation algorithm to the problem: return any vertex of the graph.
- For every $\varepsilon > 0$ there is no $(1/n^{1-\varepsilon})$-approximation algorithm for *IS*. [Zuckerman, 2007]
- No $(2^{O(\sqrt{\log d})}/d)$-approximation algorithm exists, where d is the graph's maximum degree. [Trevisan, 2001]

Vertex Cover & Independent Set

Vertex Cover:

- A simple algorithm (just find a maximal matching and take both endpoints) gives a 2-approximation for *VC*.
- *VC* is *NP*-hard to approximate within a factor of 1.3606. [Dinur & Safra, 2005]
- If UGC is true, *VC* cannot be approximated within any constant factor better than 2. [Khot & Regev, 2008]

Independent Set:

- There is a completely trivial $(1/n)$-approximation algorithm to the problem: return any vertex of the graph.
- For every $\varepsilon > 0$ there is no $(1/n^{1-\varepsilon})$-approximation algorithm for *IS*. [Zuckerman, 2007]
- No $(2^{O(\sqrt{\log d})}/d)$-approximation algorithm exists, where d is the graph's maximum degree. [Trevisan, 2001]

Max-Cut & Metric TSP

Max-Cut:

- It has been proven that *MAX-CUT* is *NP*-hard to approximate with an approximation ratio better than $16/17 \approx 0.941$. [Håstad, 2001]
- Using semidefinite programming, there is an approximation algorithm with a ratio of $\alpha \approx 0.878$. [Goemans & Williamson, 1995]
- If UGC is true, this is the best possible approximation ratio for *MAX-CUT*. [Khot et al., 2007]

Metric TSP:

- The best known approximation ratio is 3/2 [Christofides, 1976].
- There is an 8/7-approximation algorithm if the distances are restricted to 1 and 2 (but still are a metric). [Berman & Karpinski, 2006]
- There is no polynomial time algorithm for Metric TSP with performance ratio better that 123/122 (and 75/74 for asymmetric distances). [Karpinski, Lampis & Schmied, 2013]

Max-Cut & Metric TSP

Max-Cut:

- It has been proven that *MAX-CUT* is *NP*-hard to approximate with an approximation ratio better than $16/17 \approx 0.941$. [Håstad, 2001]
- Using semidefinite programming, there is an approximation algorithm with a ratio of $\alpha \approx 0.878$. [Goemans & Williamson, 1995]
- If UGC is true, this is the best possible approximation ratio for *MAX-CUT*. [Khot et al., 2007]

Metric TSP:

- The best known approximation ratio is 3/2 [Christofides, 1976].
- There is an 8/7-approximation algorithm if the distances are restricted to 1 and 2 (but still are a metric). [Berman & Karpinski, 2006]
- There is no polynomial time algorithm for Metric TSP with performance ratio better that 123/122 (and 75/74 for asymmetric distances). [Karpinski, Lampis & Schmied, 2013]

Max-Cut & Metric TSP

Max-Cut:

- It has been proven that *MAX-CUT* is *NP*-hard to approximate with an approximation ratio better than $16/17 \approx 0.941$. [Håstad, 2001]
- Using semidefinite programming, there is an approximation algorithm with a ratio of $\alpha \approx 0.878$. [Goemans & Williamson, 1995]
- If UGC is true, this is the best possible approximation ratio for *MAX-CUT*. [Khot et al., 2007]

Metric TSP:

- The best known approximation ratio is $3/2$ [Christofides, 1976].
- There is an $8/7$-approximation algorithm if the distances are restricted to 1 and 2 (but still are a metric). [Berman & Karpinski, 2006]
- There is no polynomial time algorithm for Metric TSP with performance ratio better that $123/122$ (and $75/74$ for asymmetric distances). [Karpinski, Lampis & Schmied, 2013]

Max-Cut & Metric TSP

Max-Cut:

- It has been proven that *MAX-CUT* is *NP*-hard to approximate with an approximation ratio better than $16/17 \approx 0.941$. [Håstad, 2001]
- Using semidefinite programming, there is an approximation algorithm with a ratio of $\alpha \approx 0.878$. [Goemans & Williamson, 1995]
- If UGC is true, this is the best possible approximation ratio for *MAX-CUT*. [Khot et al., 2007]

Metric TSP:

- The best known approximation ratio is $3/2$ [Christofides, 1976].
- There is an $8/7$-approximation algorithm if the distances are restricted to 1 and 2 (but still are a metric). [Berman & Karpinski, 2006]
- There is no polynomial time algorithm for Metric TSP with performance ratio better that $123/122$ (and $75/74$ for asymmetric distances). [Karpinski, Lampis & Schmied, 2013]

Max-Cut & Metric TSP

Max-Cut:

- It has been proven that *MAX-CUT* is *NP*-hard to approximate with an approximation ratio better than $16/17 \approx 0.941$. [Håstad, 2001]
- Using semidefinite programming, there is an approximation algorithm with a ratio of $\alpha \approx 0.878$. [Goemans & Williamson, 1995]
- If UGC is true, this is the best possible approximation ratio for *MAX-CUT*. [Khot et al., 2007]

Metric TSP:

- The best known approximation ratio is $3/2$ [Christofides, 1976].
- There is an $8/7$-approximation algorithm if the distances are restricted to 1 and 2 (but still are a metric). [Berman & Karpinski, 2006]
- There is no polynomial time algorithm for Metric TSP with performance ratio better that $123/122$ (and $75/74$ for asymmetric distances). [Karpinski, Lampis & Schmied, 2013]

Max-Cut & Metric TSP

Max-Cut:

- It has been proven that *MAX-CUT* is *NP*-hard to approximate with an approximation ratio better than $16/17 \approx 0.941$. [Håstad, 2001]
- Using semidefinite programming, there is an approximation algorithm with a ratio of $\alpha \approx 0.878$. [Goemans & Williamson, 1995]
- If UGC is true, this is the best possible approximation ratio for *MAX-CUT*. [Khot et al., 2007]

Metric TSP:

- The best known approximation ratio is $3/2$ [Christofides, 1976].
- There is an $8/7$-approximation algorithm if the distances are restricted to 1 and 2 (but still are a metric). [Berman & Karpinski, 2006]
- There is no polynomial time algorithm for Metric TSP with performance ratio better that $123/122$ (and $75/74$ for asymmetric distances). [Karpinski, Lampis & Schmied, 2013]

Colorability

- For every $\varepsilon > 0$, there is no $n^{1-\varepsilon}$-approximation algorithm. [Zuckerman, 2007]

An interesting special case of the problem is to devise algorithms that color a 3-colorable graph with a minimum number of colors.

- There is a polynomial time algorithm that colors every 3-colorable graph with at most $\tilde{O}(n^{3/14 \approx 0.214})$ colors. [Karger & Blum, 1997]
- There is no polynomial time algorithm that colors every 3-colorable graph using at most 4 colors. [Khanna, Linial & Safra, 1993]

This is one of the largest gaps between known approximation algorithms and known inapproximability results.

Colorability

- For every $\varepsilon > 0$, there is no $n^{1-\varepsilon}$-approximation algorithm. [Zuckerman, 2007]

An interesting special case of the problem is to devise algorithms that color a 3-colorable graph with a minimum number of colors.

- There is a polynomial time algorithm that colors every 3-colorable graph with at most $\tilde{O}(n^{3/14 \approx 0.214})$ colors. [Karger & Blum, 1997]
- There is no polynomial time algorithm that colors every 3-colorable graph using at most 4 colors. [Khanna, Linial & Safra, 1993]

This is one of the largest gaps between known approximation algorithms and known inapproximability results.

Colorability

- For every $\varepsilon > 0$, there is no $n^{1-\varepsilon}$-approximation algorithm. [Zuckerman, 2007]

An interesting special case of the problem is to devise algorithms that color a 3-colorable graph with a minimum number of colors.

- There is a polynomial time algorithm that colors every 3-colorable graph with at most $\tilde{O}(n^{3/14 \approx 0.214})$ colors. [Karger & Blum, 1997]
- There is no polynomial time algorithm that colors every 3-colorable graph using at most 4 colors. [Khanna, Linial & Safra, 1993]

This is one of the largest gaps between known approximation algorithms and known inapproximability results.

Colorability

- For every $\varepsilon > 0$, there is no $n^{1-\varepsilon}$-approximation algorithm. [Zuckerman, 2007]

An interesting special case of the problem is to devise algorithms that color a 3-colorable graph with a minimum number of colors.

- There is a polynomial time algorithm that colors every 3-colorable graph with at most $\tilde{O}(n^{3/14 \approx 0.214})$ colors. [Karger & Blum, 1997]
- There is no polynomial time algorithm that colors every 3-colorable graph using at most 4 colors. [Khanna, Linial & Safra, 1993]

This is one of the largest gaps between known approximation algorithms and known inapproximability results.

Colorability

- For every $\varepsilon > 0$, there is no $n^{1-\varepsilon}$-approximation algorithm. [Zuckerman, 2007]

An interesting special case of the problem is to devise algorithms that color a 3-colorable graph with a minimum number of colors.

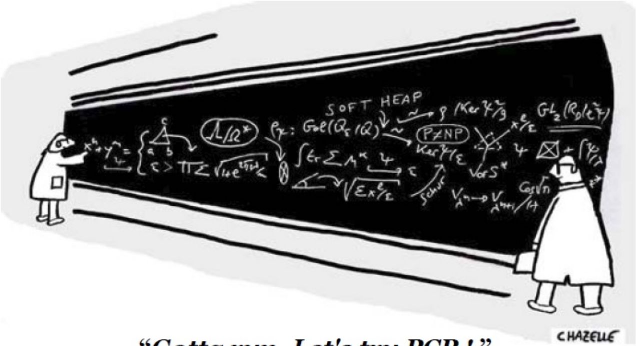
- There is a polynomial time algorithm that colors every 3-colorable graph with at most $\tilde{O}(n^{3/14 \approx 0.214})$ colors. [Karger & Blum, 1997]
- There is no polynomial time algorithm that colors every 3-colorable graph using at most 4 colors. [Khanna, Linial & Safra, 1993]

This is one of the largest gaps between known approximation algorithms and known inapproximability results.

Bibliography

1. How NP Got a New Definition: A Survey of Probabilistically Checkable Proofs.
   Sanjeev Arora. ICM, 2002.
2. Computational Complexity: A Modern Approach.
   Sanjeev Arora, Boaz Barak. Cambridge University Press, 2009.
3. Inapproximability of Combinatorial Optimization Problems.
   Luca Trevisan. ECCC, 2004.
4. Probabilistic Checking of Proofs: A New Characterization of NP
   Sanjeev Arora, Shmuel Safra. ACM, 1998.
5. Approximation Algorithms.
   Vijay V. Vazirani. Springer, 2003.

Thank You!



"Gotta run. Let's try PCP!"