The PCP theorem

Klaus Meer

Brandenburg University of Technology, Cottbus, Germany

Santaló's Summer School, Part 2, July, 2012

(work supported by DFG, GZ:ME 1424/7-1)

Outline

1. Introduction

Suppose you work at a university and have to grade a Master's thesis; you know the student is bad and you do not want to read the entire text in order to prove it.

However, you have to write a report which should not state there are faults if everything is correct.

1. Introduction

Suppose you work at a university and have to grade a Master's thesis; you know the student is bad and you do not want to read the entire text in order to prove it.

However, you have to write a report which should not state there are faults if everything is correct.

Question: Can you read only a very small fraction (less than one page) of the thesis and nevertheless be almost sure to detect a fault?

1. Introduction

Suppose you work at a university and have to grade a Master's thesis; you know the student is bad and you do not want to read the entire text in order to prove it.

However, you have to write a report which should not state there are faults if everything is correct.

Question: Can you read only a very small fraction (less than one page) of the thesis and nevertheless be almost sure to detect a fault?

Yes, if thesis is written according to the PCP-theorem

1. Introduction

Suppose you work at a university and have to grade a Master's thesis; you know the student is bad and you do not want to read the entire text in order to prove it.

However, you have to write a report which should not state there are faults if everything is correct.

Question: Can you read only a very small fraction (less than one page) of the thesis and nevertheless be almost sure to detect a fault?

Yes, if thesis is written according to the PCP-theorem

(and for some students also without it ...)

Probabilistically Checkable Proofs give a new surprising characterization of class NP

One of the most important results in Theoretical Computer Science in last 20 years

important as well for questions about approximation algorithms

Arora & Safra 1992 / 1998

Arora & Lund & Motwani & Sudan & Szegedy 1992 / 1998

Dinur 2005

Example (NP-verification for NP-complete problem 3-SAT)

Given $\phi(x_1,\ldots,x_n) = C_1 \wedge \ldots \wedge C_m$ formula in Conjunctive Normal Form, each C_i with at most 3 literals, is there a satisfying assignment $y \in \{0,1\}^n$ for ϕ? $(C_i = x_1 \vee \bar{x}_2 \vee x_4)$

Example (NP-verification for NP-complete problem 3-SAT)

Given $\phi(x_1,\ldots,x_n) = C_1 \wedge \ldots \wedge C_m$ formula in Conjunctive Normal Form, each C_i with at most 3 literals, is there a satisfying assignment $y \in \{0,1\}^n$ for ϕ? $(C_i = x_1 \vee \bar{x}_2 \vee x_4)$

NP-verification algorithm requires

- polynomial running time in $size(\phi)$ on input (ϕ, y)
- for each satisfiable ϕ there is a guess y^* such that algorithm accepts (ϕ, y^*)
- for all unsatisfiable ϕ and all guesses y algorithm rejects

Example (NP-verification for NP-complete problem 3-SAT)

Given $\phi(x_1, \ldots, x_n) = C_1 \wedge \ldots \wedge C_m$ formula in Conjunctive Normal Form, each C_i with at most 3 literals, is there a satisfying assignment $y \in \{0,1\}^n$ for ϕ? $(C_i = x_1 \vee \bar{x}_2 \vee x_4)$

NP-verification algorithm requires

- polynomial running time in $size(\phi)$ on input (ϕ, y)
- for each satisfiable ϕ there is a guess y^* such that algorithm accepts (ϕ, y^*)
- for all unsatisfiable ϕ and all guesses y algorithm rejects

Easy: Guess assignment y^*, check by plugging into ϕ

Central for above verification: the algorithm has to inspect all components of the potential satisfying assignment.

Can we design other verification algorithms that have to inspect less many parts of a potential proof, may be paying something for it?

Central for above verification: the algorithm has to inspect all components of the potential satisfying assignment.

Can we design other verification algorithms that have to inspect less many parts of a potential proof, may be paying something for it?

Surprising result: Less many above turns out to be constantly many only; we pay by including randomization, i.e., false proofs might be accepted with very small probability.

Proofs must code assignments completely differently

Example

Suppose as part of a verification proof you want to check whether two vectors $a, b \in \{0,1\}^n$ are the same.

It is up to you how information about a, b is coded in your proof (this scenario at the moment sounds a bit strange, but reoccurs later on)

Example

Suppose as part of a verification proof you want to check whether two vectors $a, b \in \{0,1\}^n$ are the same.

It is up to you how information about a, b is coded in your proof (this scenario at the moment sounds a bit strange, but reoccurs later on)

1. Easy way: Write down a, b and compare componentwise; each component has to be read

Example

Suppose as part of a verification proof you want to check whether two vectors $a, b \in \{0,1\}^n$ are the same.

It is up to you how information about a, b is coded in your proof (this scenario at the moment sounds a bit strange, but reoccurs later on)

1. Easy way: Write down a, b and compare componentwise; each component has to be read

2. A bit more tricky: Expect proof to contain all results $a^t \cdot r$ and $b^t \cdot r$; pick randomly an $r \in \{0,1\}^n$ and read only the two corresponding results in your proof.

Example (cntd.)

With probability $\frac{1}{2}$ test detects if $a \neq b$.

This probability can be made arbitrarily small by constantly many repetitions, i.e., still reading constantly many components only.

Example (cntd.)

With probability $\frac{1}{2}$ test detects if $a \neq b$.

This probability can be made arbitrarily small by constantly many repetitions, i.e., still reading constantly many components only.

Disadvantage:

Example (cntd.)

With probability $\frac{1}{2}$ test detects if $a \neq b$.

This probability can be made arbitrarily small by constantly many repetitions, i.e., still reading constantly many components only.

Disadvantage: Table of values the proof expects is exponentially large

2. Verifiers and PCP classes

Let $r, q : \mathbb{N} \mapsto \mathbb{N}$ be integer (resource) functions

2. Verifiers and PCP classes

Let $r, q : \mathbb{N} \mapsto \mathbb{N}$ be integer (resource) functions

An $(r(n), q(n))$-verifier V is a polynomial time randomized Turing machine that works in three phases on an input x of size n and a potential proof y for showing that x has a desired property:

2. Verifiers and PCP classes

Let $r, q : \mathbb{N} \mapsto \mathbb{N}$ be integer (resource) functions

An $(r(n), q(n))$-verifier V is a polynomial time randomized Turing machine that works in three phases on an input x of size n and a potential proof y for showing that x has a desired property:

Phase 1	generate $r(n)$ random bits
Phase 2	use x and the random bits to determine $q(n)$ many positions in y which the verifier wants to inspect
Phase 3	use x, the $r(n)$ random bits and the $q(n)$ chosen components from y to compute $V(x, y, r) \in \{0, 1\}$

$r(n)$ measures amount of randomness used

$q(n)$ measures number of proof components to be seen

Note: Only Phase 1 is randomized, rest is deterministic

$r(n)$ measures amount of randomness used

$q(n)$ measures number of proof components to be seen

Note: Only Phase 1 is randomized, rest is deterministic

Example

$(0, poly(n))$-verifiers work like

$r(n)$ measures amount of randomness used

$q(n)$ measures number of proof components to be seen

Note: Only Phase 1 is randomized, rest is deterministic

Example

$(0, poly(n))$-verifiers work like NP-verification algorithms.

$r(n)$ measures amount of randomness used

$q(n)$ measures number of proof components to be seen

Note: Only Phase 1 is randomized, rest is deterministic

Example

$(0, poly(n))$-verifiers work like NP-verification algorithms.

Question: Which language does a verifier accept?

Definition

a) Let V be an (r, q)-verifier. V accepts a language L if the following holds:

Definition

a) Let V be an (r, q)-verifier. V accepts a language L if the following holds:

i) For all $x \in L$ there is a y such that $Pr_r(V(x, y, r) = 1) \ = \ 1$; there is a proof which the verifier accepts with probability 1.

ii) For all $x \notin L$ and for all y it is $Pr_r(V(x, y, r) = 1) \ \leq \frac{1}{4}$; the verifier accepts a false proof with probability at most $\frac{1}{4}$.

Definition

a) Let V be an (r, q)-verifier. V accepts a language L if the following holds:

i) For all $x \in L$ there is a y such that $Pr_r(V(x, y, r) = 1) \;=\; 1$; there is a proof which the verifier accepts with probability 1.

ii) For all $x \notin L$ and for all y it is $Pr_r(V(x, y, r) = 1) \;\leq \frac{1}{4}$; the verifier accepts a false proof with probability at most $\frac{1}{4}$.

b) Let $\mathcal{F}, \mathcal{G}$ be function classes; a language L belongs to class $PCP(\mathcal{F}, \mathcal{G})$ if there is a (r, q)-verifier accepting L where $r \in \mathcal{F}, q \in \mathcal{G}$.

Remark

The bound $\frac{1}{4}$ is arbitrarily chosen.

The definition says nothing about wrong proofs if $x \in L$;

Remark

The bound $\frac{1}{4}$ is arbitrarily chosen.

The definition says nothing about wrong proofs if $x \in L$; this makes sense since such a proof could be false in a single position only. Below the goal is to rewrite proofs such that errors in false proofs in case $x \notin L$ will spread all over the proof.

Example

1. PCP(0,poly) $=$ NP

Example

1. PCP(0,poly) $=$ NP
2. PCP(poly,0) $=$

Example

1. PCP(0,poly) $=$ NP
2. PCP(poly,0) $=$

Example

1. PCP(0,poly) $=$ NP
2. PCP(poly,0) $=$ co-RP

Example

1. PCP(0,poly) $=$ NP
2. PCP(poly,0) $=$ co-RP
3. PCP($\log n, 1$) $=$

Example

1. PCP(0,poly) $=$ NP
2. PCP(poly,0) $=$ co-RP
3. PCP($\log n, 1$) $=$

Example

1. PCP(0,poly) $=$ NP
2. PCP(poly,0) $=$ co-RP
3. PCP($\log n, 1$) $=$ P

Example

1. PCP(0,poly) $=$ NP
2. PCP(poly,0) $=$ co-RP
3. PCP$(\log n, 1) =$ P ; for "$\subseteq$" simulate verifier on all random choices; for each component y_i it wants to see check which value of y_i would cause V to accept the input; this determines a potential proof (if there is any)

Example

1. PCP(0,poly) $=$ NP
2. PCP(poly,0) $=$ co-RP
3. PCP($\log n, 1$) $=$ P ; for "$\subseteq$" simulate verifier on all random choices; for each component y_i it wants to see check which value of y_i would cause V to accept the input; this determines a potential proof (if there is any)
4. PCP($\log n, 2$) $=$

Example

1. PCP(0,poly) $=$ NP
2. PCP(poly,0) $=$ co-RP
3. PCP($\log n, 1$) $=$ P ; for "$\subseteq$" simulate verifier on all random choices; for each component y_i it wants to see check which value of y_i would cause V to accept the input; this determines a potential proof (if there is any)
4. PCP($\log n, 2$) $=$

Example

1. PCP(0,poly) $=$ NP
2. PCP(poly,0) $=$ co-RP
3. PCP($\log n, 1$) $=$ P ; for "$\subseteq$" simulate verifier on all random choices; for each component y_i it wants to see check which value of y_i would cause V to accept the input; this determines a potential proof (if there is any)
4. PCP($\log n, 2$) $=$ P

Example

1. PCP(0,poly) = NP
2. PCP(poly,0) = co-RP
3. $PCP(\log n, 1) = P$; for "$\subseteq$" simulate verifier on all random choices; for each component y_i it wants to see check which value of y_i would cause V to accept the input; this determines a potential proof (if there is any)
4. $PCP(\log n, 2) = P$; similarly: simulating V on all random choices leads to a 2-SAT formula determining a potential proof; reading 3 components likely leaves P.

3. The PCP theorem

Theorem (PCP theorem)

$$PCP(O(\log n), O(1)) \;=\; NP$$

3. The PCP theorem

Theorem (PCP theorem)

$$PCP(O(\log n), O(1)) \; = \; NP$$

Proof.

Easy part $\subseteq$: Let V be $(O(\log n), q)$-verifier for L, x an instance.

An NP verification for L works as follows:

3. The PCP theorem

Theorem (PCP theorem)

$PCP(O(\log n), O(1)) \; = \; NP$

Proof.

Easy part $\subseteq$: Let V be $(O(\log n), q)$-verifier for L, x an instance.

An NP verification for L works as follows: Guess proof y and simulate V deterministically for all (polynomially many) random strings r. Accept iff all these results $V(x, y, r) = 1$.

3. The PCP theorem

Theorem (PCP theorem)

$PCP(O(\log n), O(1)) \; = \; NP$

Proof.

Easy part $\subseteq$: Let V be $(O(\log n), q)$-verifier for L, x an instance.

An NP verification for L works as follows: Guess proof y and simulate V deterministically for all (polynomially many) random strings r. Accept iff all these results $V(x, y, r) = 1$.

If $x \in L$ and y is the correct proof for V it is as well correct for above NP verification; if $x \notin L$ then for each y and $\frac{3}{4}$ of the strings r V rejects and so does the NP verification. Thus $L \in$ NP.

The inclusion $NP \subseteq PCP(O(\log n), O(1))$ is the hard part to prove

The inclusion $NP \subseteq PCP(O(\log n), O(1))$ is the hard part to prove

Note: $PCP(O(\log n), O(1))$ closed under polynomial time reductions; thus sufficient to show that a fixed NP-complete problem has an $(O(\log n), O(1))$-verifier; consider 3-SAT

The inclusion $NP \subseteq PCP(O(\log n), O(1))$ is the hard part to prove

Note: $PCP(O(\log n), O(1))$ closed under polynomial time reductions; thus sufficient to show that a fixed NP-complete problem has an $(O(\log n), O(1))$-verifier; consider 3-SAT

Steps towards the proof by Arora et al.:

Step 1: Show existence of long transparent proofs for 3-SAT:

Theorem

$$\text{3-SAT} \in PCP(O(n^2), O(1))$$

Step 1: Show existence of long transparent proofs for 3-SAT:

Theorem

$$3\text{-}SAT \ \in \ PCP(O(n^2), O(1))$$

A satisfying assignment a of a given formula ϕ is coded as follows:

i) arithmetization of formula together with randomization leads to polynomial P_r of degree 2 such that

- if $a \in \{0,1\}^n$ satisfies ϕ, then $P_r(a) = 0$
- if a is not satisfying, then $P_r(a) = 0$ only with small probability w.r.t. r

ii) P_r can be decomposed as

$$P_r(a) = f_0(r) + A_a(f_1(r)) + B_a(f_2(r))$$

where $A_a : \{0,1\}^n \mapsto \{0,1\}$, $B_a : \{0,1\}^{n^2} \mapsto \{0,1\}$ are linear functions canonically attached to a and

ii) P_r can be decomposed as

$$P_r(a) = f_0(r) + A_a(f_1(r)) + B_a(f_2(r))$$

where $A_a : \{0,1\}^n \mapsto \{0,1\}$, $B_a : \{0,1\}^{n^2} \mapsto \{0,1\}$ are linear functions canonically attached to a and the $f_i(r)$ can be computed efficiently from r.

ii) P_r can be decomposed as

$$P_r(a) = f_0(r) + A_a(f_1(r)) + B_a(f_2(r))$$

where $A_a : \{0,1\}^n \mapsto \{0,1\}$, $B_a : \{0,1\}^{n^2} \mapsto \{0,1\}$ are linear functions canonically attached to a and the $f_i(r)$ can be computed efficiently from r.

Thus evaluating $P_r(a)$ for given r requires only to look up one function value of A_a and B_a

The proof the verifier expects thus contains the function values of A_a and B_a; it has exponential size.

Introduced new difficulties to be circumvented:

1. Does a table of function values correspond to an almost linear function:

(self-)testing linear functions

Introduced new difficulties to be circumvented:

1. Does a table of function values correspond to an almost linear function:

   (self-)testing linear functions

2. If yes, how can we compute the correct values if table contains small errors:

   (self-)correcting linear functions

Introduced new difficulties to be circumvented:

1. Does a table of function values correspond to an almost linear function:

   (self-)testing linear functions

2. If yes, how can we compute the correct values if table contains small errors:

   (self-)correcting linear functions

3. Even if the tables for A_a and B_a represent linear functions are they coming from the same a:

   consistency

Step 2: Existence of short almost transparent proofs for 3-SAT:

Theorem

$$\textit{3-SAT} \in PCP(O(\log n), O(\textit{polylog}(n)))$$

Step 2: Existence of short almost transparent proofs for 3-SAT:

Theorem

$$\textit{3-SAT} \in PCP(O(\log n), O(polylog(n)))$$

For the proof a satisfying assignment is coded via a low-degree polynomial

Develop similar - though much harder - techniques for self-testing and self-correcting such polynomials

Step 3: Instead of constructing better and better verifiers the final step uses a composition of the verifiers obtained in Steps 1 and 2 to obtain the one proving the PCP theorem.

Step 3: Instead of constructing better and better verifiers the final step uses a composition of the verifiers obtained in Steps 1 and 2 to obtain the one proving the PCP theorem.

4. PCPs and Approximation

Area of approximation algorithms tries to classify NP-hard optimization problems according to how well optimal solutions can be approximated; recall non-approximability of minimal multi-homogeneous Bézout number from previous lecture!

4. PCPs and Approximation

Area of approximation algorithms tries to classify NP-hard optimization problems according to how well optimal solutions can be approximated; recall non-approximability of minimal multi-homogeneous Bézout number from previous lecture!

Example (MAX-3-SAT)

Input: m clauses $C_1, C_2, \ldots, C_m$ each with at most 3 literals over variables $x_1, \ldots, x_n$

Goal: Compute the maximal number of clauses that can be satisfied in common

4. PCPs and Approximation

Area of approximation algorithms tries to classify NP-hard optimization problems according to how well optimal solutions can be approximated; recall non-approximability of minimal multi-homogeneous Bézout number from previous lecture!

Example (MAX-3-SAT)

Input: m clauses $C_1, C_2, \ldots, C_m$ each with at most 3 literals over variables $x_1, \ldots, x_n$

Goal: Compute the maximal number of clauses that can be satisfied in common

Clear: computing exact maximum is NP-hard

Example (cntd.)

Can we efficiently approximate the maximum up to a given constant factor including a corresponding assignment?

Example (cntd.)

Can we efficiently approximate the maximum up to a given constant factor including a corresponding assignment?

Easy for factor 2:

Example (cntd.)

Can we efficiently approximate the maximum up to a given constant factor including a corresponding assignment?

Easy for factor 2: For each x_i count in how many clauses x_i and in how many $\bar{x}_i$ occur; choose x_i^*'s value according to majority; then x^* satisfies at least $\frac{m}{2}$ many clauses, i.e.,

Example (cntd.)

Can we efficiently approximate the maximum up to a given constant factor including a corresponding assignment?

Easy for factor 2: For each x_i count in how many clauses x_i and in how many $\bar{x}_i$ occur; choose x_i^*'s value according to majority; then x^* satisfies at least $\frac{m}{2}$ many clauses, i.e.,

$$\frac{\text{optimal value}}{\text{Algorithm's result}} \leq 2$$

Example (cntd.)

Question: Can we reduce the error arbitrarily, i.e., is there an algorithm which given ϕ and $\epsilon > 0$ computes an assignment with relative error $\leq 1 + \epsilon$?

Running time polynomial in *size*(ϕ), arbitrary in ϵ^{-1}; defines complexity class PTAS

Example (cntd.)

Question: Can we reduce the error arbitrarily, i.e., is there an algorithm which given ϕ and $\epsilon > 0$ computes an assignment with relative error $\leq 1 + \epsilon$?

Running time polynomial in *size*(ϕ), arbitrary in ϵ^{-1};

defines complexity class PTAS

The question whether MAX-3-SAT $\in$ PTAS could only be answered after the PCP-theorem was proven

GAP-technique: method to show that certain approximation problems do not belong to PTAS unless $P \neq NP$

GAP-technique: method to show that certain approximation problems do not belong to PTAS unless P $\neq$ NP

GAP creating reduction: reduce efficiently instance ϕ for 3-SAT decision problem to instance $\psi(\phi)$ for MAX-3-SAT such that

- if ϕ is satisfiable, then all clauses $\psi(\phi)$ are satisfiable in common
- if ϕ is not satisfiable, then at most $1 - c$ of the clauses of $\psi(\phi)$ are commonly satisfiable, where $0 < c < 1$ is a constant independent of ϕ.

Relation to non-existence of PTAS algorithms:

Lemma

Suppose a GAP creating reduction from 3-SAT to MAX-3-SAT exists, then MAX-3-SAT $\notin$ PTAS unless $P = NP$.

Relation to non-existence of PTAS algorithms:

Lemma

Suppose a GAP creating reduction from 3-SAT to MAX-3-SAT exists, then MAX-3-SAT $\notin$ PTAS unless $P = NP$.

Proof.

Suppose $\mathcal{A}$ is a PTAS algorithm for MAX-3-SAT; given instance ϕ for 3-SAT we can decide satisfiability efficiently as follows: compute the reduction and apply $\mathcal{A}$ to the resulting MAX-3-SAT instance $\psi(\phi)$ and ϵ sufficiently small such that $(1-c)\cdot(1+\epsilon) < 1$ Now approximating $\psi(\phi)$ within relative error $\leq 1+\epsilon$ results in deciding satisfiability of ϕ.

However so far unclear whether such a gap creating reduction exists

However so far unclear whether such a gap creating reduction exists

Theorem (Arora, Motwani, Safra, Sudan, Szegedy '92)

The PCP theorem is equivalent to the existence of a gap creating reduction from 3-SAT to MAX-3-SAT.

However so far unclear whether such a gap creating reduction exists

Theorem (Arora, Motwani, Safra, Sudan, Szegedy '92)

The PCP theorem is equivalent to the existence of a gap creating reduction from 3-SAT to MAX-3-SAT.

Proof.

"if-part": Suppose the reduction ψ exists; construct an $(O(\log n), O(1))$-verifier for 3-SAT:

However so far unclear whether such a gap creating reduction exists

Theorem (Arora, Motwani, Safra, Sudan, Szegedy '92)

The PCP theorem is equivalent to the existence of a gap creating reduction from 3-SAT to MAX-3-SAT.

Proof.

"if-part": Suppose the reduction ψ exists; construct an $(O(\log n), O(1))$-verifier for 3-SAT: Given 3-SAT formula ϕ the verifier V first computes 3-SAT formula $\psi(\phi)$;

However so far unclear whether such a gap creating reduction exists

Theorem (Arora, Motwani, Safra, Sudan, Szegedy '92)

The PCP theorem is equivalent to the existence of a gap creating reduction from 3-SAT to MAX-3-SAT.

Proof.

"if-part": Suppose the reduction ψ exists; construct an $(O(\log n), O(1))$-verifier for 3-SAT: Given 3-SAT formula ϕ the verifier V first computes 3-SAT formula $\psi(\phi)$; V expects as proof a satisfying assignment for $\psi(\phi)$;

However so far unclear whether such a gap creating reduction exists

Theorem (Arora, Motwani, Safra, Sudan, Szegedy '92)

The PCP theorem is equivalent to the existence of a gap creating reduction from 3-SAT to MAX-3-SAT.

Proof.

"if-part": Suppose the reduction ψ exists; construct an $(O(\log n), O(1))$-verifier for 3-SAT: Given 3-SAT formula ϕ the verifier V first computes 3-SAT formula $\psi(\phi)$; V expects as proof a satisfying assignment for $\psi(\phi)$; it randomly guesses one of the clauses in $\psi(\phi)$; there are polynomially many in $size(\phi)$, thus $O(\log size(\phi))$ random bits are needed;

Proof (cntd.)

V then verifies by reading 3 bits from the given proof whether it satisfies the chosen clause;

Proof (cntd.)

V then verifies by reading 3 bits from the given proof whether it satisfies the chosen clause;

if ϕ is satisfiable so is $\psi(\phi)$ and V accepts a satisfying assignment as proof with probability 1;

Proof (cntd.)

V then verifies by reading 3 bits from the given proof whether it satisfies the chosen clause;

if ϕ is satisfiable so is $\psi(\phi)$ and V accepts a satisfying assignment as proof with probability 1;

if ϕ is not satisfiable, then V chooses only with probability $< 1 - c$ a clause that is satisfied by the given assignment; repeating the procedure constantly many times this probability can be reduced to $< \frac{1}{4}$.

Proof.

"only-if part": Let V be a $(c \cdot \log n, q)$-verifier for 3-SAT, q a constant, ϕ a 3-SAT instance. We show how to obtain a gap creating reduction to MAX-3-SAT

Proof.

"only-if part": Let V be a $(c \cdot \log n, q)$-verifier for 3-SAT, q a constant, ϕ a 3-SAT instance. We show how to obtain a gap creating reduction to MAX-3-SAT

Step 0: The verifier has at most n^c different runs and wants to inspect at most $N := q \cdot n^c$ components of a proof. Let $y_1, \ldots, y_N$ be Boolean variables. An assignment to them corresponds to a proof.

Proof.

"only-if part": Let V be a $(c \cdot \log n, q)$-verifier for 3-SAT, q a constant, ϕ a 3-SAT instance. We show how to obtain a gap creating reduction to MAX-3-SAT

Step 0: The verifier has at most n^c different runs and wants to inspect at most $N := q \cdot n^c$ components of a proof. Let $y_1, \ldots, y_N$ be Boolean variables. An assignment to them corresponds to a proof.

Step 1: For random string ρ define $A_\rho \subseteq \{0,1\}^q$ as those w such that V rejects (ϕ, y, ρ) if the components read from y constitute w.

Proof.

all A_ρ are efficiently computable and of constant cardinality!

Let $C_1 \wedge \ldots \wedge C_{|A_\rho|}$ be a set of clauses in q variables such that for each $w \notin A_\rho$ precisely one C_i is false. Note that number s of clauses is a constant depending on q only.

Proof.

all A_ρ are efficiently computable and of constant cardinality!

Let $C_1 \wedge \ldots \wedge C_{|A_\rho|}$ be a set of clauses in q variables such that for each $w \notin A_\rho$ precisely one C_i is false. Note that number s of clauses is a constant depending on q only.

Step 2: Next consider for each random string ρ that V generates the variant $C_1^\rho \wedge \ldots \wedge C_{|A_\rho|}^\rho$ of clauses, where the variables are replaced by those q many components of $y_1, \ldots, y_N$ the verifier wants to see.

Proof.

Conjunction of all those clauses we obtain by trying all ρ gives a SAT-formula with $s \cdot n^c$ many clauses. If a proof $y \in \{0,1\}^{n^c}$ is satisfying for the verifier it satisfies all above clauses;

Proof.

Conjunction of all those clauses we obtain by trying all ρ gives a SAT-formula with $s \cdot n^c$ many clauses. If a proof $y \in \{0,1\}^{n^c}$ is satisfying for the verifier it satisfies all above clauses; if not, for at least $\frac{3}{4}$ many choices of ρ at least one among the s clauses in $C_1^\rho \wedge \ldots \wedge C_s^\rho$ is not satisfied. This results in a constant fraction of at least $\frac{3}{4s}$ many unsatisfied clauses; thus a gap is achieved

Proof.

Conjunction of all those clauses we obtain by trying all ρ gives a SAT-formula with $s \cdot n^c$ many clauses. If a proof $y \in \{0,1\}^{n^c}$ is satisfying for the verifier it satisfies all above clauses; if not, for at least $\frac{3}{4}$ many choices of ρ at least one among the s clauses in $C_1^\rho \wedge \ldots \wedge C_s^\rho$ is not satisfied. This results in a constant fraction of at least $\frac{3}{4s}$ many unsatisfied clauses; thus a gap is achieved

Above proof easily extends to obtain a 3-SAT formula by usual reduction.

5. Dinur's proof: Short remark

In 2005 Dinur gave an alternative proof of the PCP theorem

Main idea: construct a gap creating reduction directly for another problem called Constraint Satisfiability Problem; problem defined (and needed) over arbitrary finite alphabets.

5. Dinur's proof: Short remark

In 2005 Dinur gave an alternative proof of the PCP theorem

Main idea: construct a gap creating reduction directly for another problem called Constraint Satisfiability Problem; problem defined (and needed) over arbitrary finite alphabets.

Starting from a small gap depending on $\frac{1}{\text{input size}}$ a tricky gap amplification construction is invoked $O(\log n)$ times to increase the gap to be constant.

Amplification increases size of underlying finite alphabets; second step performs alphabet reduction by using long transparent proofs.

Today: focus on ideas for first proof of PCP theorem

Next talk: PCPs for real number model, in particular

- long transparent proofs for $\mathrm{NP}_{\mathbb{R}}$
- ideas for proving real $\mathrm{PCP}_{\mathbb{R}}$ theorem along Dinur's lines

References

1. Arora, Lund, Motwani, Sudan, Szegedy: Proof verification and intractability of approximation problems. JACM 45: 501–555, 1998
2. Arora, Safra: Probabilistically checkable proofs: a new characterization of NP. JACM 45: 70–122, 1998
3. Ausiello et al: Approximation Algorihtms. Springer, 1998
4. Hougardy, Prömel, Steger: PCPs and their consequences for approximation algorithms. DAM 136, 175–223, 1994.
5. Radhakrishnan & Sudan, Bulletin AMS 44 (1), 19-61, 2007
6. Webpage of M. Sudan contains a lot of survey papers

Definition of class RP: there is a randomized polynomial time machine M which accepts each $x \in L$ with probability at least $\frac{2}{3}$ and rejects all $x \notin L$.