CS151
Complexity Theory

Lecture 16

May 24, 2017

The PCP Theorem

- Two major components:
  - **NP ⊂ PCP[log n, polylog n]** ("outer verifier")
    - we will prove this from scratch, assuming low-degree test, and self-correction of low-degree polynomials
  - **NP ⊂ PCP[n^3, 1]** ("inner verifier")
    - we will prove assuming low-degree test

Proof Composition (idea)

- NP $\subset$ PCP[log n, 1] comes from
  - repeated composition

  - PCP[log n, polylog n] with PCP[log n, polylog n] yields PCP[log n, polyloglog n]

  - PCP[log n, polyloglog n] with PCP[n^3, 1] yields PCP[log n, 1]
- many details omitted…

The outer verifier

Theorem: **NP ⊂ PCP[log n, polylog n]**

Proof (first steps):

– define: Polynomial Constraint Satisfaction (PCS) problem

– prove: PCS gap problem is **NP**-hard

NP ⊂ PCP[log n, polylog n]

- MAX-k-SAT
  - given: k-CNF φ
  - output: max. # of simultaneously satisfiable clauses
- generalization: MAX-k-CSP
  - given:
    - variables $x_1, x_2, \ldots, x_n$ taking values from set S
    - k-ary constraints $C_1, C_2, \ldots, C_t$
  - output: max. # of simultaneously satisfiable constraints

NP ⊂ PCP[log n, polylog n]

- algebraic version: MAX-k-PCS
  - given:
    - variables $x_1, x_2, \ldots, x_n$ taking values from **field** F_q
    - $n = q^m$ for some integer m
    - k-ary constraints $C_1, C_2, \ldots, C_t$
  - assignment viewed as $f:(F_q)^m \rightarrow F_q$
  - output: max. # of constraints simultaneously satisfiable by an assignment that has deg. ≤ d

NP ⊂ PCP[log n, polylog n]

- MAX-k-PCS **gap problem**:
  - given:
    - variables $x_1, x_2, \ldots, x_n$ taking values from **field** F_q
    - $n = q^m$ for some integer m
    - k-ary constraints $C_1, C_2, \ldots, C_t$
  - assignment viewed as $f:(F_q)^m \rightarrow F_q$
  - YES: some degree d assignment satisfies all constraints
  - NO: no degree d assignment satisfies more than $(1-\varepsilon)$ fraction of constraints

NP ⊂ PCP[log n, polylog n]

Lemma: for every constant $1 > \varepsilon > 0$, the MAX-k-PCS gap problem with

t = poly(n) k-ary constraints with k = polylog(n)

field size q = polylog(n)

$n = q^m$ variables with $m = O(\log n / \log\log n)$

degree of assignments d = polylog(n)

gap ε

is **NP**-hard.

NP ⊂ PCP[log n, polylog n]

t = poly(n) k-ary constraints with k = polylog(n)

field size q = polylog(n)

n = q^m variables with m = O(log n / loglog n)

degree of assignments d = polylog(n)

- check: headed in right direction
  - O(log n) random bits to pick a constraint
  - query assignment in O(polylog(n)) locations to determine if it is satisfied
  - completeness 1; soundness 1-ε

(if prover keeps promise to supply degree d polynomial)

NP ⊂ PCP[log n, polylog n]

- Proof of Lemma
  - reduce from 3-SAT
  - 3-CNF $\varphi(x_1, x_2, \ldots, x_n)$
  - can encode as $\psi:[n] \times [n] \times [n] \times \{0,1\}^3 \rightarrow \{0,1\}$
  - $\psi(i_1, i_2, i_3, b_1, b_2, b_3) = 1$ iff φ contains clause

$$(x_{i_1}{}^{b_1} \vee x_{i_2}{}^{b_2} \vee x_{i_3}{}^{b_3})$$

  - e.g. $(x_3 \vee \neg x_5 \vee x_2) \Rightarrow \psi(3,5,2,1,0,1) = 1$

NP ⊂ PCP[log n, polylog n]

– pick $H \subset F_q$ with $\{0,1\} \subset H$, $|H| = \text{polylog } n$

– pick $m = O(\log n/\log\log n)$ so $|H|^m = n$

– identify $[n]$ with H^m

– $\psi: H^m \times H^m \times H^m \times H^3 \rightarrow \{0,1\}$ encodes φ

– assignment $a: H^m \rightarrow \{0,1\}$

– Key: a satisfies φ iff $\forall i_1,i_2,i_3,b_1,b_2,b_3$

$$\psi(i_1,i_2,i_3,b_1,b_2,b_3) = 0 \text{ or}$$

$$a(i_1)=b_1 \text{ or } a(i_2)=b_2 \text{ or } a(i_3)=b_3$$

NP ⊂ PCP[log n, polylog n]

$\psi: H^m \times H^m \times H^m \times H^3 \rightarrow \{0,1\}$ encodes φ

a satisfies φ iff $\forall i_1,i_2,i_3,b_1,b_2,b_3$

$\psi(i_1,i_2,i_3,b_1,b_2,b_3) = 0$ or $a(i_1)=b_1$ or $a(i_2)=b_2$ or $a(i_3)=b_3$

– **extend** ψ to a function $\psi': (F_q)^{3m+3} \rightarrow F_q$ with degree at most $|H|$ in each variable

– can **extend** any assignment $a: H^m \rightarrow \{0,1\}$ to a': $(F_q)^m \rightarrow F_q$ with degree $|H|$ in each variable

NP $\subset$ PCP[log n, polylog n]

ψ':$(F_q)^{3m+3} \rightarrow F_q$ encodes φ

a':$(F_q)^m \rightarrow F_q$ s.a. iff $\forall(i_1,i_2,i_3,b_1,b_2,b_3) \in H^{3m+3}$

$\psi'(i_1,i_2,i_3,b_1,b_2,b_3) = 0$ or $a'(i_1)=b_1$ or $a'(i_2)=b_2$ or $a'(i_3)=b_3$

– define: $p_{a'}$:$(F_q)^{3m+3} \rightarrow F_q$ from a' as follows

$p_{a'}(i_1,i_2,i_3,b_1,b_2,b_3) =$

$\psi'(i_1,i_2,i_3,b_1,b_2,b_3)(a'(i_1) - b_1)(a'(i_2) - b_2)(a'(i_3) - b_3)$

– a' s.a. iff $\forall (i_1,i_2,i_3,b_1,b_2,b_3) \in H^{3m+3}$

$p_{a'}(i_1,i_2,i_3,b_1,b_2,b_3) = 0$

NP ⊂ PCP[log n, polylog n]

ψ':$(F_q)^{3m+3} \rightarrow F_q$ encodes φ

a':$(F_q)^m \rightarrow F_q$ s.a. iff $\forall(i_1,i_2,i_3,b_1,b_2,b_3) \in H^{3m+3}$

$p_{a'}(i_1,i_2,i_3,b_1,b_2,b_3) = 0$

– note: $\deg(p_{a'}) \leq 2(3m+3)|H|$

– start using Z as shorthand for $(i_1,i_2,i_3,b_1,b_2,b_3)$

– another way to write "a' s.a." is:

- exists p_0:$(F_q)^{3m+3} \rightarrow F_q$ of degree $\leq 2(3m+3)|H|$
- $p_0(Z) = p_{a'}(Z)$ $\quad \forall Z \in (F_q)^{3m+3}$
- $p_0(Z) = 0$ $\quad \forall Z \in H^{3m+3}$

NP ⊂ PCP[log n, polylog n]

– Focus on "$p_0(Z) = 0\ \forall Z \in H^{3m+3}$"

– given: $p_0:(F_q)^{3m+3} \rightarrow F_q$

– define: $p_1(x_1, x_2, x_3, \ldots, x_{3m+3}) =$

$$\Sigma_{h_j \in H} p_0(h_j, x_2, x_3, \ldots, x_{3m+3}) x_1^j$$

– **Claim**:

$$p_0(Z)=0\ \forall Z \in H^{3m+3} \Leftrightarrow p_1(Z)=0\ \forall Z \in F_q x H^{3m+3-1}$$

– Proof ($\Rightarrow$) for each $x_2, x_3, \ldots, x_{3m+3} \in H^{3m+3-1}$, resulting univariate poly in x_1 has all 0 coeffs.

NP ⊂ PCP[log n, polylog n]

– Focus on "$p_0(Z) = 0\ \forall Z \in H^{3m+3}$"

– given: $p_0:(F_q)^{3m+3} \rightarrow F_q$

– define: $p_1(x_1, x_2, x_3, \ldots, x_{3m+3}) =$

$$\Sigma_{h_j \in H} p_0(h_j, x_2, x_3, \ldots, x_{3m+3}) x_1^{\,j}$$

$\deg(p_1) \leq \deg(p_0) + |H|$

– **Claim**:

$$p_0(Z)=0\ \forall Z \in H^{3m+3} \Leftrightarrow p_1(Z)=0\ \forall Z \in F_q x H^{3m+3-1}$$

– Proof ($\Leftarrow$) for each $x_2, x_3, \ldots, x_{3m+3} \in H^{3m+3-1}$, univariate poly in x_1 is $\equiv 0 \Rightarrow$ has all 0 coeffs.

NP ⊂ PCP[log n, polylog n]

– given: $p_1:(F_q)^{3m+3} \rightarrow F_q$

$\deg(p_2) \leq \deg(p_1) + |H|$

– define: $p_2(x_1, x_2, x_3, \ldots, x_{3m+3}) =$

$$\Sigma_{h_j \in H} p_1(x_1, h_j, x_3, x_4, \ldots, x_{3m+3}) x_2^j$$

– Claim:

$$p_1(Z)=0 \ \forall Z \in F_q \times H^{3m+3-1}$$

$$\Leftrightarrow$$

$$p_2(Z)=0 \ \forall Z \in (F_q)^2 \times H^{3m+3-2}$$

– Proof: same.

NP ⊂ PCP[log n, polylog n]

– given: $p_{i-1}:(F_q)^{3m+3} \rightarrow F_q$

– define: $p_i(x_1, x_2, x_3, \ldots, x_{3m+3}) =$

$\Sigma_{h_j \in H} p_{i-1}(x_1, x_2, \ldots, x_{i-1}, h_j, x_{i+1}, x_{i+2}, \ldots, x_{3m+3}) x_i^j$

$\deg(p_i) \leq \deg(p_{i-1}) + |H|$

– Claim:

$$p_{i-1}(Z)=0 \ \forall Z \in (F_q)^{i-1} \times H^{3m+3-(i-1)}$$

$$\Leftrightarrow$$

$$p_i(Z)=0 \ \forall Z \in (F_q)^{i} \times H^{3m+3-i}$$

– Proof: same.

NP ⊂ PCP[log n, polylog n]

– define degree $3m+3+2$ poly. $\delta_i : F_q \rightarrow F_q$ so that

- $\delta_i(v) = 1$ if $v = i$
- $\delta_i(v) = 0$ if $0 \leq v \leq 3m+3+1$ and $v \neq i$

– define $Q : F_q \times (F_q)^{3m+3} \rightarrow F_q$ by:

$$Q(v, Z) = \Sigma_{i=0\ldots 3m+3} \delta_i(v) p_i(Z) + \delta_{3m+3+1}(v) a'(Z)$$

– note: degree of Q is at most

$$3(3m+3)|H| + 3m + 3 + 2 < 10m|H|$$

NP ⊂ PCP[log n, polylog n]

- Recall: MAX-k-PCS **gap problem**:
  - given:
    - variables $x_1, x_2, \ldots, x_n$ taking values from **field** F_q
    - $n = q^m$ for some integer m
    - k-ary constraints $C_1, C_2, \ldots, C_t$
  - assignment viewed as $f:(F_q)^m \rightarrow F_q$
  - YES: some degree d assignment satisfies all constraints
  - NO: no degree d assignment satisfies more than $(1-\varepsilon)$ fraction of constraints

NP ⊂ PCP[log n, polylog n]

– Instance of MAX-k-PCS **gap problem**:

- set $d = 10m|H|$
- given assignment $Q:F_q \times (F_q)^{3m+3} \rightarrow F_q$
- expect it to be formed in the way we have described from an assignment $a:H^m \rightarrow \{0,1\}$ to φ
- note

  to access $a'(Z)$, evaluate $Q(3m+3+1, Z)$

  $p_{a'}(Z)$ formed from a' and ψ' (formed from φ)

  to access $p_i(Z)$, evaluate $Q(i, Z)$

NP ⊂ PCP[log n, polylog n]

– Instance of MAX-k-PCS **gap problem**:

- set $d = 10m|H|$
- given assignment $Q:F_q \times (F_q)^{3m+3} \rightarrow F_q$
- expect it to be formed in the way we have described from an assignment $a:H^m \rightarrow \{0,1\}$ to φ
- **constraints**: $\forall Z \in (F_q)^{3m+3}$

$(C_{0,Z})$: $p_0(Z) = p_{a'}(Z)$

$0<i\leq 3m+2$ $(C_{i,Z})$: $p_i(z_1, z_2, \ldots, z_i, z_{i+1}, \ldots, z_{3m+3}) = \Sigma_{h_j \in H} p_{i-1}(z_1, z_2, \ldots, z_{i-1}, h_j, z_{i+1}, \ldots, z_k) z_i^j$

$(C_{3m+3,Z})$: $p_{3m+3}(Z) = 0$

NP ⊂ PCP[log n, polylog n]

- given $Q:F_q \times (F_q)^{3m+3} \rightarrow F_q$ of degree $d = 10m|H|$
- **constraints**: $\forall Z \in (F_q)^{3m+3}$

$(C_0,_Z)$: $p_0(Z) = p_{a'}(Z)$

$(C_i,_Z)$: $p_i(z_1, z_2, \ldots, z_i, z_{i+1}, \ldots, z_{3m+3}) =$

$\Sigma_{h_j \in H}\, p_{i-1}(z_1, z_2, \ldots, z_{i-1}, h_j, z_{i+1}, \ldots, z_k) z_i^j$

$(C_{3m+3},_Z)$: $p_{3m+3}(Z) = 0$

Key: all low-degree polys

– **Schwartz-Zippel**: if any one of these sets of constraints is violated *at all* then at least a **(1 – 12m|H|/q)** fraction in the set are violated

NP ⊂ PCP[log n, polylog n]

- Proof of Lemma (summary):
  - reducing 3-SAT to MAX-k-PCS **gap problem**
  - $\varphi(x_1, x_2, \ldots, x_n)$ instance of 3-SAT
  - set $m = O(\log n / \log\log n)$
  - $H \subset F_q$ such that $|H|^m = n$ $\quad (|H| = \text{polylog } n,\ q \approx |H|^3)$
  - generate $|F_q|^{3m+3} = \text{poly}(n)$ constraints:

$$C_Z = \Lambda_{i=0\ldots 3m+3+1}\, C_{i,\,Z}$$

  - each refers to assignment poly Q and φ (via $p_{a'}$)
  - all polys degree $d = O(m|H|) = \text{polylog } n$
  - either all are satisfied or at most $d/q = o(1) << \varepsilon$

NP ⊂ PCP[log n, polylog n]

- O(log n) random bits to pick a constraint
- query assignment in O(polylog(n)) locations to determine if constraint is satisfied
  - completeness 1
  - soundness (1-ε) if prover keeps promise to supply degree d polynomial
- prover can cheat by not supplying proof in expected form

NP ⊂ PCP[log n, polylog n]

- Low-degree testing:
  - want: randomized procedure that is given d, oracle access to $f:(F_q)^m \rightarrow F_q$
    - runs in poly(m, d) time
    - always accepts if deg(f) ≤ d
    - rejects with high probability if deg(f) > d
  - too much to ask. Why?

NP ⊂ PCP[log n, polylog n]

Definition: functions f, g are δ-close if

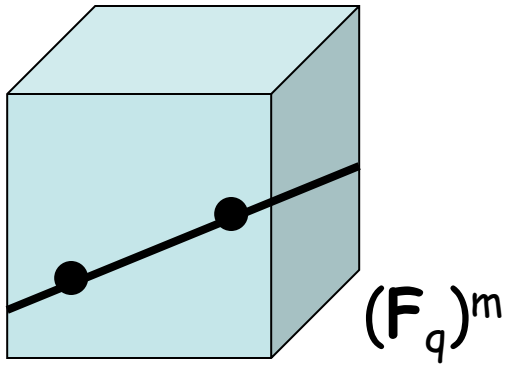
$$\Pr_x[f(x) \neq g(x)] \leq \delta$$

Lemma: ∃ δ > 0 and a randomized procedure that is given d, oracle access to $f:(F_q)^m \rightarrow F_q$

- runs in poly(m, d) time
- uses $O(m \log |F_q|)$ random bits
- always accepts if deg(f) ≤ d
- rejects with high probability if f is not δ-close to any g with deg(g) ≤ d

NP ⊂ PCP[log n, polylog n]

- idea of proof:
  - restrict to random line L
  - check if it is low degree



  - always accepts if deg(f) ≤ d
  - other direction much more complex

NP ⊂ PCP[log n, polylog n]

– can only force prover to supply function f that is close to a low-degree polynomial

– how to bridge the gap?

– recall low-degree polynomials form an error correcting code (Reed-Muller)

– view “close” function as corrupted codeword

NP ⊂ PCP[log n, polylog n]

- Self-correction:
  - want: randomized procedure that is given x, oracle access to $f:(F_q)^m \rightarrow (F_q)$ that is δ-close to a (unique) degree d polynomial g
    - runs in poly(m, d) time
    - uses $O(m \log |F_q|)$ random bits
    - with high probability outputs g(x)

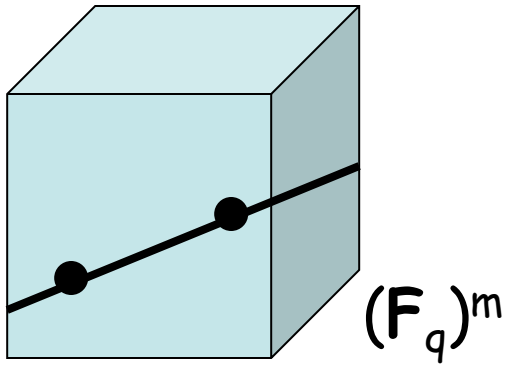
NP ⊂ PCP[log n, polylog n]

Lemma: ∃ a randomized procedure that is given x, oracle access to $f:(F_q)^m \rightarrow (F_q)$ that is δ-close to a (unique) degree d polynomial g

- runs in poly(m, d) time
- uses $O(m \log |F_q|)$ random bits
- outputs g(x) with high probability

NP ⊂ PCP[log n, polylog n]

- idea of proof:
  - restrict to random line L passing through x
  - query points along line
  - apply error correction



NP ⊂ PCP[log n, polylog n]

- Putting it all together:
  - given L ∈ NP and an instance x, verifier computes reduction to MAX-k-PCS **gap problem**
  - prover supplies proof in form

    $$f:(F_q)^m \rightarrow (F_q)$$

    (plus some other info used for low-degree testing)
  - verifier runs low-degree test
    - rejects if f not close to some low degree function g
  - verifier picks random constraint C_i; checks if sat. by g
    - uses self-correction to get values of g from f
  - accept if C_i satisfied; otherwise reject

New topic:
relativization
and
natural proofs

Approaches to open problems

- Almost all major open problems we have seen entail proving **lower bounds**
  - **P ≠ NP**
  - **L ≠ P**
  - **P ≠ PSPACE**
  - **NC** proper
  - **BPP ≠ EXP**
  - **PH** proper
  - **EXP ⊄ P/poly**
  - **P = BPP ***
  - **NP = AM ***

• we know circuit lower bounds imply derandomization

• more difficult (and recent): derandomization implies circuit lower bounds!

Approaches to open problems

- two natural approaches
  - simulation + diagonalization (uniform)
  - circuit lower bounds (non-uniform)
- no success for either approach as applied to date

Why?

Approaches to open problems

in a precise, formal sense
these approaches are
too powerful !

- if they could be used to resolve major open problems, a side effect would be:
  - proving something that is false, or
  - proving something that is believed to be false

Relativization

- Many proofs and techniques we have seen **relativize**:
  - they hold after replacing all TMs with oracle TMs that have access to an oracle A
  - e.g. $\mathbf{L^A \subset P^A}$ for all oracles A
  - e.g. $\mathbf{P^A \neq EXP^A}$ for all oracles A

Relativization

- Idea: design an oracle A relative to which some statement is *false*
  - implies there can be no **relativizing proof** of that statement
  - e.g. design A for which $\mathbf{P^A = NP^A}$
- Better: also design an oracle B relative to which statement is *true*
  - e.g. also design B for which $\mathbf{P^B \neq NP^B}$
  - implies no **relativizing proof** can resolve truth of the statement either way !

Relativization

- Oracles are known that falsify almost every major conjecture concerning complexity classes
  - for these conjectures, non-relativizing proofs are required
  - almost all known proofs in Complexity relativize (sometimes after some reformulation)
  - notable exceptions:
    - The PCP Theorem
    - IP = PSPACE
    - most circuit lower bounds (more on these later)

Oracles for **P** vs. **NP**

- Goal:
  - oracle A for which $\mathbf{P^A = NP^A}$
  - oracle B for which $\mathbf{P^B \neq NP^B}$

- conclusion: resolving

  P vs. NP

  requires a non-relativizing proof

Oracles for **P** vs. **NP**

- for $\mathbf{P^A = NP^A}$ need A to be powerful
  - warning: intend to make **P** more powerful, but also make **NP** more powerful.
  - e.g. A = SAT doesn't work
  - however A = QSAT works:

$$\mathbf{PSPACE} \subset \mathbf{P^{QSAT}} \subset \mathbf{NP^{QSAT}} \subset \mathbf{NPSPACE}$$

and we know $\mathbf{NPSPACE} \subset \mathbf{PSPACE}$