CS151 Complexity Theory

Lecture 17

May 27, 2004

Outline

- elements of the proof of the PCP Theorem
- counting problems
  - **#P** and its relation to other classes
  - complexity of computing the permanent
- proofs about proofs:
  - relativization
  - natural proofs
- course summary

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 $|H|^m = n$ $\quad$ ($|H|$ = polylog n, $q \approx |H|^3$)
  - generate $|F_q|^{3m+3}$ = poly(n) constraints:

$$C_Z = \wedge_{i=0\ldots3m+3+1} C_{i,\,Z}$$

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

NP ⊂ PCP[log n, polylog n]

- log n random bits to pick a constraint
- query assignment in 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 $\deg(f) \le d$
    - rejects with high probability if $\deg(f) > d$
  - too much to ask. Why?

NP ⊂ PCP[log n, polylog n]

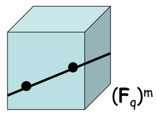
Definition: functions f, g are δ-close if

$$\Pr_x[f(x) \ne g(x)] \le \delta$$

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 $\deg(f) \le d$
- rejects with high probability if f is not δ-close to any g with $\deg(g) \le d$

NP $\subset$ 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 $\subset$ 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 $\subset$ 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)

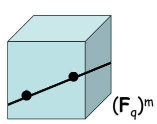
NP $\subset$ PCP[log n, polylog n]

<u>Lemma</u>: ∃ 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 $\subset$ PCP[log n, polylog n]

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



NP $\subset$ 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

Counting problems

- So far, we have ignored function problems
  – given x, compute f(x)
- justification: usually easily reducible to related decision problem
- important class of function problems that don't seem to have this property:

  counting problems

  – e.g. given 3-CNF φ how many satisfying assignments are there?

Counting problems

- **#P** is the class of function problems expressible as:

  input x $\quad f(x) = |\{y : (x, y) \in R\}|$

  where $R \in$ **P**.

- compare to **NP** (decision problem)

  input x $\quad f(x) = \exists y : (x, y) \in R$?

  where $R \in$ **P**.

Counting problems

- examples
  – #SAT: given 3-CNF φ how many satisfying assignments are there?

  – #CLIQUE: given (G, k) how many cliques of size at least k are there?

Reductions

- Reduction from function problem f_1 to function problem f_2
  – two efficiently computable functions Q, A

$$\begin{tikzcd} \underset{\text{(prob. 1)}}{x} \arrow[r, "Q"] \arrow[d, "f_1"] & \underset{\text{(prob. 2)}}{y} \arrow[d, "f_2"] \\ f_1(x) & f_2(y) \arrow[l, "A"'] \end{tikzcd}$$

Reductions

- problem f is **#P**-complete if
  – f is in **#P**
  – every problem in **#P** reduces to f

$$\begin{tikzcd} \underset{\text{(prob. 1)}}{x} \arrow[r, "Q"] \arrow[d, "f_1"] & \underset{\text{(prob. 2)}}{y} \arrow[d, "f_2"] \\ f_1(x) & f_2(y) \arrow[l, "A"'] \end{tikzcd}$$

- "parsimonious reduction": A is identity
  – many standard **NP**-completeness reductions are parsimonious
  – therefore: if #SAT is **#P**-complete we get lots of **#P**-complete problems

#SAT

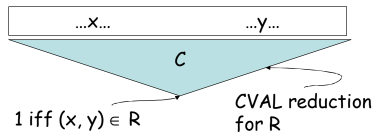
#SAT: given 3-CNF φ how many satisfying assignments are there?

<u>Theorem</u>: #SAT is **#P**-complete.

- Proof:
  – clearly in **#P:** $(\varphi, A) \in R \Leftrightarrow A$ satisfies φ
  – take any $f \in$ **#P** defined by $R \in$ **P**

#SAT

f(x) =
|{y : (x, y) ∈ R}|



– add new variables z, produce φ such that

$\exists z\ \varphi(x, y, z) = 1 \Leftrightarrow C(x, y) = 1$

– for (x, y) such that C(x, y) = 1 this z is *unique*

– hardwire x

– # satisfying assignments = |{y : (x, y) ∈ R}|

Relationship to other classes

- To compare to classes of decision problems, usually consider

  $\mathbf{P}^{\#\mathbf{P}}$

  which is a decision class…
- easy: **NP**, **coNP** ⊂ $\mathbf{P}^{\#\mathbf{P}}$
- easy: $\mathbf{P}^{\#\mathbf{P}}$ ⊂ **PSPACE**

Toda's Theorem: **PH** ⊂ $\mathbf{P}^{\#\mathbf{P}}$**.**

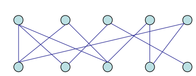
Relationship to other classes

Question: is **#P** hard because it entails *finding* **NP** witnesses?

…or is *counting* difficult by itself?

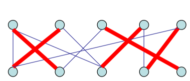
Bipartite Matchings

- Definition:
  - G = (U, V, E) bipartite graph with |U| = |V|
  - a **perfect matching** in G is a subset M ⊂ E that touches every node, and no two edges in M share an endpoint



Bipartite Matchings

- Definition:
  - G = (U, V, E) bipartite graph with |U| = |V|
  - a **perfect matching** in G is a subset M ⊂ E that touches every node, and no two edges in M share an endpoint



Bipartite Matchings

- #MATCHING: given a bipartite graph G = (U, V, E) how many perfect matchings does it have?

Theorem: #MATCHING is **#P**-complete.

- But… can *find* a perfect matching in polynomial time!
  - counting itself must be difficult

The permanent

- The permanent of a matrix A is defined as:

  $$\text{per}(A) = \Sigma_\pi \Pi_i A_{i,\ \pi(i)}$$

- # of perfect matchings in a bipartite graph G is exactly permanent of G's adjacency matrix A_G
  - a perfect matching defines a permutation that contributes 1 to the sum

May 27, 2004 CS151 Lecture 17 25

The permanent

- thus permanent is **#P-**complete
  - permanent also has many nice properties that make it a favorite of complexity theory
- contrast permanent (very hard)

  $$\text{per}(A) = \Sigma_\pi \Pi_i A_{i,\ \pi(i)}$$

  to determinant (very easy):

  $$\det(A) = \Sigma_\pi \text{sgn}(\pi) \Pi_i A_{i,\ \pi(i)}$$

May 27, 2004 CS151 Lecture 17 26

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
  - **P/poly ≠ EXP**
  - **P = BPP** *
  - **NP = AM** *

• we know circuit lower bounds imply derandomization

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

May 27, 2004 CS151 Lecture 17 27

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?

May 27, 2004 CS151 Lecture 17 28

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

May 27, 2004 CS151 Lecture 17 29

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. $\mathbf{L}^A \subset \mathbf{P}^A$ for all oracles A
  - e.g. $\mathbf{P}^A \neq \mathbf{EXP}^A$ for all oracles A

May 27, 2004 CS151 Lecture 17 30

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 $\mathbf{P}^A = \mathbf{NP}^A$
- Better: also design an oracle B relative to which statement is *true*
  - e.g. also design B for which $\mathbf{P}^B \neq \mathbf{NP}^B$
  - 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 $\mathbf{P}^A = \mathbf{NP}^A$
  - oracle B for which $\mathbf{P}^B \neq \mathbf{NP}^B$
- conclusion: resolving

  P vs. NP

  requires a non-relativizing proof

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

- for $\mathbf{P}^A = \mathbf{NP}^A$ 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:

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

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

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

<u>**Theorem**</u>: there exists an oracle B for which $\mathbf{P}^B \neq \mathbf{NP}^B$.

- Proof:
  - define

    $L = \{1^k : \exists\, x \in B \text{ s.t. } |x| = k\}$
  - we will show $L \in \mathbf{NP}^B - \mathbf{P}^B$.
  - easy: $L \in \mathbf{NP}^B$ (no matter what B is)

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

- design B by diagonalizing against all

  "$\mathbf{P}^B$ machines"
- $M_1, M_2, M_3, \ldots$ is an enumeration of deterministic OTMs
- each machine appears infinitely often $\Rightarrow$ all poly-time machines appear even if we force machine M_i to accept after $n^{\log n}$ steps
- B_i will be those strings of length $\leq i$ in B
- we build B_i after simulating machine M_i

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

$L = \{1^k : \exists\, x \in B \text{ s.t. } |x| = k\}$

- Proof (continued):
  - maintain "exceptions" X that must not go in B
  - initially X = { }, B_0 = { }

  Stage i:
  - simulate $M_i(1^i)$ for $i^{\log i}$ steps
  - when M_i makes an oracle query q:
    - if $|q| < i$, answer using B_{i-1}
    - if $|q| \geq i$, answer "no"; add q to X
  - if simulated M_i accepts 1^i then $B_i = B_{i-1}$
  - if simulated M_i rejects 1^i, $B_i = B_{i-1} \cup \{x \in \{0,1\}^i : x \notin X\}$

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

$L = \{1^k : \exists\, x \in B \text{ s.t. } |x| = k\}$

- Proof (continued):
  - if M_i accepts, we ensure no strings of length i in B
  - therefore $1^i \notin L$, and M_i does not decide L
  - if M_i rejects, we ensure *some* string of length i in B
  - Why?

    $$B_i = B_{i-1} \cup \{x \in \{0,1\}^i : x \notin X\}$$

    and |X| is at most $\Sigma_{j \leq i}\, j^{\log j} << 2^i$
  - therefore $1^i \in L$, and M_i does not decide L
  - Conclude: $L \notin \mathbf{P}^{\mathbf{B}}$

Circuit lower bounds

- Relativizing techniques are out…
- but most circuit lower bound techniques do not relativize
- exponential circuit lower bounds known for weak models:
  - e.g. constant-depth poly-size circuits
- But, utter failure (so far) for more general models. Why?

Natural Proofs

- Razborov and Rudich defined the following "natural" format for circuit lower bounds:
  - identify property **P** of functions $f:\{0,1\}^* \rightarrow \{0,1\}$
  - **P** $= \cup_n$ $\mathbf{P}_n$ is a natural property if:
    - (useful) $\forall n$ $f_n \in \mathbf{P}_n$ implies f does not have poly-size circuits
    - (constructive) can decide "$f_n \in \mathbf{P}_n$?" in poly time given the *truth table* of f_n
    - (large) at least $(½)^{O(n)}$ fraction of all 2^{2^n} functions on n bits are in $\mathbf{P}_n$
  - show some function family g = $\{g_n\}$ is in $\mathbf{P}_n$

Natural Proofs

- *all known circuit lower bounds* are natural for a suitably parameterized version of the definition

Theorem (RR): if there is a 2^{n^δ}-OWF, then there is no natural property **P**.
  - factoring believed to be 2^{n^δ}-OWF
  - general version also rules out natural properties useful for proving many other separations, under similar cryptographic assumptions

Natural Proofs

- Proof sketch:
  - main tool: **pseudo-random functions**
  - ensemble $F_k = \{p_y:\{0,1\}^{n(k)} \rightarrow \{0,1\}\}_{y \in \{0,1\}^k}$
  - $F = \cup_k F_k$ is t(k)-pseudo-random if
    - given y, x, can compute $p_y(x)$ in poly(|y|, |x|) time
    - for every prob. TM M running in time t(k):

      $$|\Pr_y[M^{p_y}(1^k) = 1] - \Pr_{f_n}[M^{f_n}(1^k) = 1]| \leq 1/t$$
  - can construct from (BMY-style) PRGs
  - 2^{n^δ}-OWF implies 2^{cn}-pseudo-random functions $\forall$ c

Natural Proofs

(useful) $\forall n\ f_n \in \underline{\mathbf{P}}_n \Rightarrow f$ does not have poly-size circuits

(constructive) "$f_n \in \underline{\mathbf{P}}_n$?" in poly time given *truth table* of f_n

(large) at least $(½)^{O(n)}$ fraction of all 2^{2^n} fns. on n-bits in $\underline{\mathbf{P}}_n$

- Proof sketch (continued):
  - pseudo-random function p_y has poly-size circuits, and so $p_y \notin \underline{\mathbf{P}}_n$ (useful)
  - Define OTM M so that $M(1^k)$ reads $2^{n(k)}$ -size truth table of oracle and accepts if it is in $\underline{\mathbf{P}}_n$ (constructive)

    $\Pr_y[M^{p_y}(1^k)=1]=0 \quad \Pr_{f_n}[M^{f_n}(1^k) = 1] \geq (½)^{O(n)}$ (large)
  - contradiction.

Natural Proofs

- To prove circuit lower bounds, we must either:
  - Violate largeness: seize upon an incredibly specific feature of hard functions (one not possessed by a random function !)
  - Violate constructivity: identify a feature of hard functions that cannot be computed efficiently from the **truth table**
- no "non-natural property" known for all but the very weakest models…

Course summary

• Time and space	**L, P, PSPACE, EXP**
• Non-determinism	**NL, NP, coNP, NEXP**
• Non-uniformity	**NC, P/poly**
• Randomness	**RL, ZPP, RP, coRP, BPP**
• Alternation	**PH, PSPACE**
• Interaction	**IP, MA, AM, PCP[log n, 1]**
• Counting	**#P**

The big picture

- All classes on previous slide are probably distinct, except:
  - **P, ZPP, RP, coRP, BPP** (probably all equal)
  - **L, RL, NL** (probably all equal)
  - **NP, MA, AM** (probably all equal)
  - **IP** = **PSPACE**
  - **PCP[log n, 1] = NP**
- Only real separations we know separate classes delimiting same resource:
  - e.g. **L ≠ PSPACE, NP ≠ NEXP**

The big picture

Remember:

possible explanation for failure to prove conjectured separations…

…is that they are false

The big picture

- Important techniques/ideas:
  - simulation and diagonalization
  - reductions and completeness
  - self-reducibility
  - encoding information using low-degree polynomials
  - randomness
  - others…

The big picture

- I hope you take away:
  - an ability to extract the essential features of a problem that make it hard/easy…
  - knowledge and tools to connect computational problems you encounter with larger questions in complexity
  - background needed to understand current research in this area

The last slide…

- background to contribute to current research in this area
  - many open problems
  - young field
  - try your hand…

Thank you!