CS151
Complexity Theory

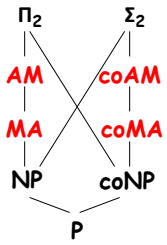
Lecture 15

May 18, 2021

MA and AM

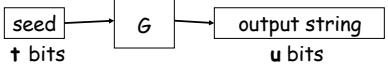
- Two important classes:
  - **MA** = **MA[2]**
  - **AM** = **AM[2]**
- definitions without reference to interaction:
  - L ∈ **MA** iff ∃ poly-time language R
    $x \in L \Rightarrow \exists\, m\ Pr_r[(x, m, r) \in R] = 1$
    $x \notin L \Rightarrow \forall\, m\ Pr_r[(x, m, r) \in R] \leq ½$
  - L ∈ **AM** iff ∃ poly-time language R
    $x \in L \Rightarrow Pr_r[\exists\, m\ (x, m, r) \in R] = 1$
    $x \notin L \Rightarrow Pr_r[\exists\, m\ (x, m, r) \in R] \leq ½$

MA and AM



Derandomization revisited

- L ∈ **MA** iff ∃ poly-time language R
  $x \in L \Rightarrow \exists\, m\ Pr_r[(x, m, r) \in R] = 1$
  $x \notin L \Rightarrow \forall\, m\ Pr_r[(x, m, r) \in R] \leq ½$
- Recall PRGs:



  – for all circuits C of size at most **s**:
  $|Pr_y[C(y) = 1] - Pr_z[C(G(z)) = 1]| \leq \boldsymbol{\varepsilon}$

Using PRGs for MA

- L ∈ **MA** iff ∃ poly-time language R
  $x \in L \Rightarrow \exists\, m\ Pr_r[(x, m, r) \in R] = 1$
  $x \notin L \Rightarrow \forall\, m\ Pr_r[(x, m, r) \in R] \leq ½$
- produce poly-size circuit C such that
  $C(x, m, r) = 1 \Leftrightarrow (x,m,r) \in R$
- for each x, m can hardwire to get $C_{x,m}$
  $\exists\, m\ Pr_y[C_{x,m}(y) = 1] = 1$ ("yes")
  $\forall\, m\ Pr_y[C_{x,m}(y) = 1] \leq 1/2$ ("no")

Using PRGs for MA

poly(n)

- can compute $Pr_z[C_{x,m}(G(z)) = 1]$ exactly
  – evaluate $C_{x,m}(G(z))$ on every seed $z \in \{0,1\}^t$
  – running time $(O(|C_{x,m}|)+(\text{time for } G))\mathbf{2^t}$

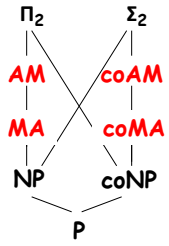
poly-time

  $x \in L \Rightarrow \exists\, m\ [Pr_z[C_{x,m}(G(z)) = 1] = 1]$
  $x \notin L \Rightarrow \forall\, m\ [Pr_z[C_{x,m}(G(z)) = 1] \leq ½ + \epsilon]$
  – L ∈ NP if PRG with t = O(log n), $\epsilon < 1/2$

Theorem: **E** requires exponential size circuits ⇒ **MA** = **NP**.

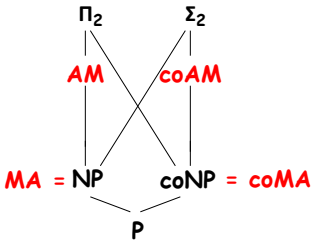
MA and AM

(under a hardness assumption)



MA and AM

(under a hardness assumption)



What about AM, coAM?

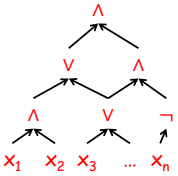
Derandomization revisited

Theorem (IW, STV): If **E** contains functions that require size $2^{\Omega(n)}$ circuits, then **E** contains functions that are $2^{\Omega(n)}$–un-approximable by circuits.

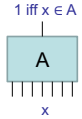
Theorem (NW): if **E** contains $2^{\Omega(n)}$-unapp-roximable functions there are poly-time PRGs fooling poly(n)-size circuits, with seed length t = O(log n), and error $\epsilon < 1/4$.

Oracle circuits

- **circuit** C
  - directed acyclic graph
  - nodes: AND (∧); OR (∨); NOT (¬); variables x_i



- **A-oracle circuit C**
  - also allow "A-oracle gates"

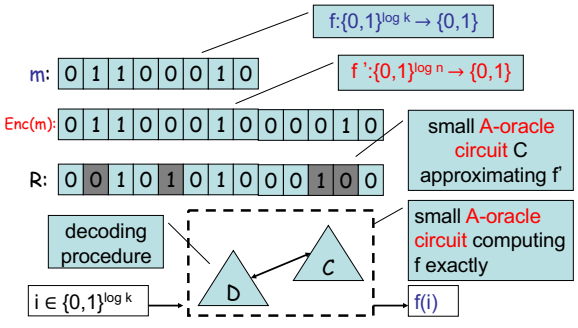


Relativized versions

Theorem: If **E** contains functions that require size $2^{\Omega(n)}$ A-oracle circuits, then **E** contains functions that are $2^{\Omega(n)}$-unapproximable by A-oracle circuits.

- Recall proof:
  - encode truth table to get hard function
  - if approximable by s(n)-size circuits, then use those circuits to compute original function by size $s(n)^{O(1)}$-size circuits. Contradiction.

Relativized versions



Relativized versions

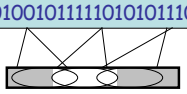
Theorem: if **E** contains $2^{\Omega(n)}$-unapproximable fns., there are poly-time PRGs fooling poly(n)-size A-oracle circuits, with seed length t = O(log n), and error $\epsilon < 1/4$.

- Recall proof:
  - PRG from hard function on O(log n) bits
  - if doesn't fool s-size circuits, then use those circuits to compute hard function by size $s \cdot n^{\delta}$-size circuits. Contradiction.

Relativized versions

$G_n(y)=f_{\log n}(y_{|S_1}) \circ f_{\log n}(y_{|S_2}) \circ \ldots \circ f_{\log n}(y_{|S_m})$

$f_{\log n}$: 01010010111110101011001010



- doesn't fool A-oracle circuit of size s:

$|Pr_x[C(x) = 1] - Pr_y[C(\ G_n(y)\ ) = 1]| > \varepsilon$

- implies A-oracle circuit P of size s' = s + O(m):

$Pr_y[P(G_n(y)_1 \cdots_{i-1}) = G_n(y)_i] > \frac{1}{2} + \varepsilon/m$

Relativized versions

$G_n(y)=f_{\log n}(y_{|S_1}) \circ f_{\log n}(y_{|S_2}) \circ \ldots \circ f_{\log n}(y_{|S_m})$

$f_{\log n}$: 01010010111110101011001010

output $f_{\log n}(y')$

A-oracle circuit approximating f

P

y' →

hardwired tables

Using PRGs for AM

L ∈ **AM** iff ∃ poly-time language R

$x \in L \Rightarrow Pr_r[\exists m\ (x, m, r) \in R] = 1$

$x \notin L \Rightarrow Pr_r[\exists m\ (x, m, r) \in R] \leq \frac{1}{2}$

- produce poly-size SAT-oracle circuit C such that (1 SAT query, accepts iff answer is "yes")

$C(x, r) = 1 \Leftrightarrow \exists m\ (x,m,r) \in R$

- for each x, can hardwire to get C_x

$Pr_y[C_x(y) = 1] = 1$ ("yes")

$Pr_y[C_x(y) = 1] \leq \frac{1}{2}$ ("no")

Using PRGs for AM

$x \in L \Rightarrow [Pr_z[C_x(G(z)) = 1] = 1]$

$x \notin L \Rightarrow [Pr_z[C_x(G(z)) = 1] \leq \frac{1}{2} + \epsilon]$

- C_x makes a *single* SAT query, accepts iff answer is "yes"
- if G is a PRG with t = O(log n), $\epsilon < \frac{1}{4}$, can check in NP:
  - does $C_x(G(z)) = 1$ for all z?

Relativized versions

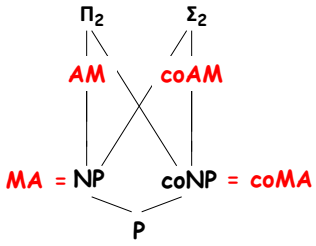
Theorem: If **E** contains functions that require size $2^{\Omega(n)}$ A-oracle circuits, then **E** contains functions that are $2^{\Omega(n)}$-unapproximable by A-oracle circuits.

Theorem: if **E** contains $2^{\Omega(n)}$-unapproximable functions there are PRGs fooling poly(n)-size A-oracle circuits, with seed length t = O(log n), and error $\epsilon < \frac{1}{2}$.

Theorem: **E** requires exponential size SAT-oracle circuits ⇒ **AM** = **NP**.

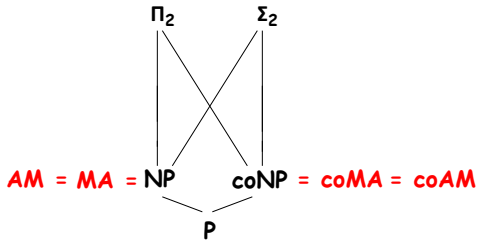
MA and AM

(under a hardness assumption)



MA and AM

(under a hardness assumption)



New topic(s)

Optimization problems,
Approximation Algorithms,
and
Probabilistically Checkable Proofs

Optimization Problems

- many hard problems (especially **NP**-hard) are optimization problems
  - e.g. find *shortest* TSP tour
  - e.g. find *smallest* vertex cover
  - e.g. find *largest* clique

  - may be minimization or maximization problem
  - "opt" = value of optimal solution

Approximation Algorithms

- often happy with approximately optimal solution
  - warning: lots of heuristics
  - we want approximation algorithm with guaranteed approximation ratio of r
  - meaning: on every input x, output is guaranteed to have value

    at most r*opt for minimization

    at least opt/r for maximization

Approximation Algorithms

- Example approximation algorithm:
  - Recall:

  Vertex Cover (VC): given a graph G, what is the *smallest* subset of vertices that touch every edge?

  - **NP**-complete

Approximation Algorithms

- Approximation algorithm for VC:
  - pick an edge (x, y), add vertices x and y to VC
  - discard edges incident to x or y; repeat.
- Claim: approximation ratio is 2.
- Proof:
  - an optimal VC must include at least one endpoint of each edge considered
  - therefore 2*opt ≥ actual

Approximation Algorithms

- diverse array of ratios achievable
- some examples:
  - (min) Vertex Cover: 2
  - MAX-3-SAT (find assignment satisfying largest # clauses): 8/7
  - (min) Set Cover: ln n
  - (max) Clique: $n/\log^2 n$
  - (max) Knapsack: (1 + ε) for any ε > 0

Approximation Algorithms

(max) Knapsack: (1 + ε) for any ε > 0

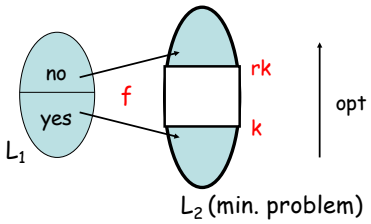
- called Polynomial Time Approximation Scheme (PTAS)
  - algorithm runs in poly time for every fixed ε>0
  - poor dependence on ε allowed
- If all **NP** optimization problems had a PTAS, almost like **P** = **NP** (!)

Approximation Algorithms

- A job for complexity: How to explain failure to do better than ratios on previous slide?
  - just like: how to explain failure to find poly-time algorithm for SAT...
  - first guess: probably NP-hard
  - what is needed to show this?
- "gap-producing" reduction from **NP**-complete problem L_1 to L_2

Approximation Algorithms

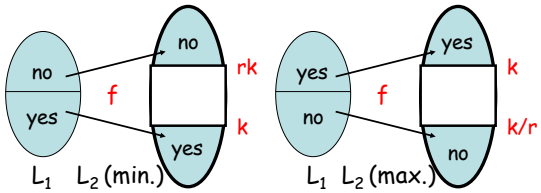
- "gap-producing" reduction from **NP**-complete problem L_1 to L_2



Gap producing reductions

- r-gap-producing reduction:
  - f computable in poly time
  - $x \in L_1 \Rightarrow \text{opt}(f(x)) \leq k$
  - $x \notin L_1 \Rightarrow \text{opt}(f(x)) > rk$
  - for max. problems use "≥ k" and "< k/r"
- Note: target problem is not a language
  - **promise problem** (yes ∪ no *not* all strings)
  - "promise": instances always from (yes ∪ no)

Gap producing reductions

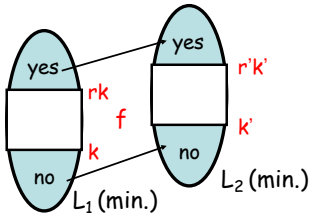


- Main purpose:
  - r-approximation algorithm for L_2 distinguishes between f(yes) and f(no); can use to decide L_1
  - "**NP**-hard to approximate to within r"

Gap preserving reductions

- gap-producing reduction difficult (more later)
- but **gap-preserving** reductions easier

Warning: many reductions **not** gap-preserving



Gap preserving reductions

- Example **gap-preserving** reduction:
  - reduce MAX-k-SAT with gap ε
  - to MAX-3-SAT with gap ε' (constants)
  - "MAX-k-SAT is NP-hard to approx. within ε ⇒ MAX-3-SAT is NP-hard to approx. within ε' "
- **MAXSNP** (PY) – a class of problems reducible to each other in this way
  - PTAS for **MAXSNP**-complete problem iff PTAS for all problems in **MAXSNP**

MAX-k-SAT

- Missing link: first gap-producing reduction
  - history's guide
    it should have something to do with SAT
- Definition: MAX-k-SAT with gap ε
  - instance: k-CNF φ
  - YES: some assignment satisfies **all** clauses
  - NO: no assignment satisfies more than (1 – ε) fraction of clauses

Proof systems viewpoint

- k-SAT **NP**-hard ⇒ for any language L∈**NP** proof system of form:
  - given x, compute reduction to k-SAT: ϕ_x
  - expected proof is satisfying assignment for ϕ_x
  - verifier picks random clause ("local test") and checks that it is satisfied by the assignment

    $x \in L \Rightarrow$ Pr[verifier accepts] = 1

    $x \notin L \Rightarrow$ Pr[verifier accepts] < 1

Proof systems viewpoint

- MAX-k-SAT with gap ε **NP**-hard ⇒ for any language L ∈ **NP** proof system of form:
  - given x, compute reduction to MAX-k-SAT: ϕ_x
  - expected proof is satisfying assignment for ϕ_x
  - verifier picks random clause ("local test") and checks that it is satisfied by the assignment

    $x \in L \Rightarrow$ Pr[verifier accepts] = 1

    $x \notin L \Rightarrow$ Pr[verifier accepts] ≤ (1 – ε)
  - can repeat O(1/ε) times for error < ½

Proof systems viewpoint

- can think of reduction showing k-SAT NP-hard as designing a proof system for **NP** in which:
  - verifier only performs local tests
- can think of reduction showing "MAX-k-SAT with gap ε" NP-hard as designing a proof system for **NP** in which:
  - verifier only performs local tests
  - invalidity of proof* evident all over: "holographic proof" and an ϵ fraction of tests notice such invalidity

PCP

- Probabilistically Checkable Proof (PCP) permits novel way of verifying proof:
  - pick random local test
  - query proof in specified k locations
  - accept iff passes test
- fancy name for a NP-hardness reduction

PCP

- **PCP[r(n),q(n)]**: set of languages L with p.p.t. verifier V that has (r, q)-restricted access to a string "proof"
  - V tosses O(r(n)) coins
  - V accesses proof in O(q(n)) locations
  - (completeness) $x \in L \Rightarrow \exists$ proof such that $\Pr[V(x, \text{proof}) \text{ accepts}] = 1$
  - (soundness) $x \notin L \Rightarrow \forall$ proof* $\Pr[V(x, \text{proof}^*) \text{ accepts}] \leq ½$

PCP

- Two observations:
  - **PCP[1, poly n] = NP**
    proof?
  - **PCP[log n, 1] ⊆ NP**
    proof?

The PCP Theorem (AS, ALMSS):
PCP[log n, 1] = NP.

PCP

Corollary: MAX-k-SAT is **NP**-hard to approximate to within some constant ϵ.

- using PCP[log n, 1] protocol for, say, VC
- enumerate all $2^{O(\log n)} = \text{poly}(n)$ sets of queries
- construct a k-CNF φ_i for verifier's test on each
  - note: k-CNF since function on only k bits
- "YES" VC instance ⇒ all clauses satisfiable
- "NO" VC instance ⇒ every assignment fails to satisfy at least ½ of the φ_i ⇒ fails to satisfy an $\epsilon = (½)2^{-k}$ fraction of clauses.

The PCP Theorem

- Elements of proof:
  - arithmetization of 3-SAT
    - we will do this
  - low-degree test
    - we will state but not prove this
  - self-correction of low-degree polynomials
    - we will state but not prove this
  - proof composition
    - we will describe the idea