Arthur-Merlin Games

• Delimiting # of rounds:
  – \( \text{AM}[k] \) = Arthur-Merlin game with \( k \) rounds, Arthur (verifier) goes first
  – \( \text{MA}[k] \) = Arthur-Merlin game with \( k \) rounds, Merlin (prover) goes first

Theorem: \( \text{AM}[k] \) (\( \text{MA}[k] \)) equals \( \text{AM}[k] \) (\( \text{MA}[k] \)) with perfect completeness.
  – i.e., \( x \in L \) implies accept with probability 1
  – proof on problem set

MA and AM

• Two important classes:
  – \( \text{MA} = \text{MA}[2] \)
  – \( \text{AM} = \text{AM}[2] \)

• definitions without reference to interaction:
  – \( L \in \text{MA} \) iff \( \exists \) poly-time language \( R \)
    \( x \in L \Rightarrow \exists m \Pr[(x, m, r) \in R] = 1 \)
    \( x \notin L \Rightarrow \forall m \Pr[(x, m, r) \in R] \leq \varepsilon \)
  – \( L \in \text{AM} \) iff \( \exists \) poly-time language \( R \)
    \( x \in L \Rightarrow \Pr[(x, m, r) \in R] = 1 \)
    \( x \notin L \Rightarrow \Pr[(x, m, r) \in R] \leq \frac{1}{2} \)

• Relation to other complexity classes:
  – both contain \( \text{NP} \) (can elect to not use randomness)
  – both contained in \( \Pi_2 \)
  – so clear that \( \text{AM} \subseteq \Pi_2 \)
  – know that \( \text{MA} \subseteq \text{AM} \)
Theorem: \( \text{coNP} \subseteq \text{AM} \Rightarrow \text{PH} = \text{AM} \).

- Proof:
  - suffices to show \( \Sigma_2 \subseteq \text{AM} \) (and use \( \text{AM} \subseteq \Pi_2 \))
  - \( L \in \Sigma_2 \iff \exists \text{poly-time language } R \)
    \( x \in L \iff \exists y \forall z (x, y, z) \in R \)
    \( x \notin L \iff \forall y \exists z (x, y, z) \notin R \)
  - Merlin sends \( y \)
  - \( 1 \) AM exchange decides \( \text{coNP} \) query: \( \forall z (x, y, z) \in R \) ?
  - \( 3 \) rounds; in \( \text{AM} \)

Back to Graph Isomorphism

- The payoff:
  - not known if GI is \( \text{NP} \)-complete.
  - previous Theorems:
    - if GI is \( \text{NP} \)-complete then \( \text{PH} = \text{AM} \)
    - unlikely!
  - Proof: GI \( \text{NP} \)-complete \( \Rightarrow \) GNI \( \text{coNP} \)-complete \( \Rightarrow \) \( \text{coNP} \subseteq \text{AM} \Rightarrow \text{PH} = \text{AM} \)

MA and AM

- Two important classes:
  - \( \text{MA} = \text{MA}[2] \)
  - \( \text{AM} = \text{AM}[2] \)
- definitions without reference to interaction:
  - \( L \in \text{MA} \iff \exists \text{poly-time language } R \)
    \( x \in L \iff \exists m \Pr[(x, m, r) \in R] = 1 \)
    \( x \notin L \iff \forall m \Pr[(x, m, r) \in R] \leq \frac{1}{2} \)
  - \( L \in \text{AM} \iff \exists \text{poly-time language } R \)
    \( x \in L \iff \Pr[\exists m (x, m, r) \in R] = 1 \)
    \( x \notin L \iff \Pr[\exists m (x, m, r) \in R] \leq \frac{1}{2} \)
Derandomization revisited

- \( L \in \text{MA} \) iff \( \exists \) poly-time language \( R \)
  \[ x \in L \Rightarrow \exists m \Pr[(x, m, r) \in R] = 1 \]
  \[ x \notin L \Rightarrow \forall m \Pr[(x, m, r) \in R] \leq \frac{1}{2} \]

- Recall PRGs:
  - for all circuits \( C \) of size at most \( s \):
    \[ |\Pr_y[C(y) = 1] - \Pr_z[C(G(z)) = 1]| \leq \varepsilon \]

Using PRGs for MA

- \( L \in \text{MA} \) iff \( \exists \) poly-time language \( R \)
  \[ x \in L \Rightarrow \exists m \Pr[(x, m, r) \in R] = 1 \]
  \[ x \notin L \Rightarrow \forall m \Pr[(x, m, r) \in R] \leq \frac{1}{2} \]

  • produce poly-size circuit \( C \) such that
    \[ C(x, m, r) = 1 \iff (x, m, r) \in R \]
  
  • for each \( x, m \) can hardwire to get \( C_{x,m} \)
    \[ \exists m \Pr_y[C_{x,m}(y) = 1] = 1 \quad \text{("yes")} \]
    \[ \forall m \Pr_y[C_{x,m}(y) = 1] \leq \frac{1}{2} \quad \text{("no")} \]

Using PRGs for MA

- 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 \( (O(|C_{x,m}|) + \text{time for } G)2^t \)
  \[ x \in L \Rightarrow \exists m \Pr_z[C_{x,m}(G(z)) = 1] = 1 \]
  \[ x \notin L \Rightarrow \forall m \Pr_z[C_{x,m}(G(z)) = 1] \leq \frac{1}{2} + \varepsilon \]

- \( L \in \text{NP} \) if PRG with \( t = O(\log n) \), \( \varepsilon < 1/2 \)

**Theorem:** \( E \) requires exponential size circuits \( \Rightarrow \text{MA} = \text{NP} \).

MA and AM

(under a hardness assumption)

- AM
- coAM
- MA
- coMA
- NP
- coNP
- P

What about AM, coAM?

- AM = NP
- coNP = coMA

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

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

- circuit C
  - directed acyclic graph
  - nodes: AND (\text{\&}); OR (\lor); NOT (\neg); variables \(x\)

- 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 \(s(n)\cdot n\) circuits. Contradiction.

**Relativized versions**

**Theorem**: if \(E\) contains \(2^{\Omega(n)}\)-unapproximable fns., there are poly-time PRGs fooling \(\text{poly}(n)\)-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 \(s \cdot n\) circuits. Contradiction.
Using PRGs for AM

\[ L \in \text{AM} \text{ iff } \exists \text{ poly-time language } R \]
\[ x \in L \Rightarrow \Pr_{r} \left[ \exists m (x, m, r) \in R \right] = 1 \]
\[ x \notin L \Rightarrow \Pr_{r} \left[ \exists m (x, m, r) \in R \right] \leq \frac{1}{2} \]

- produce poly-size SAT-oracle circuit \( C \) such that \( C(x, r) = 1 \text{ iff } \exists m (x, m, r) \in R \)
- for each \( x \), can hardwire to get \( C_x \)
\[ \Pr_{r} [C_{x}(y) = 1] = 1 \text{ ("yes") } \]
\[ \Pr_{r} [C_{x}(y) = 1] \leq \frac{1}{2} \text{ ("no") } \]

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 \( \Rightarrow \text{AM} = \text{NP} \).

MA and AM

(under a hardness assumption)

\[ \Pi_2 \]
\[ \Sigma_2 \]
\[ \text{AM} \]
\[ \text{coAM} \]
\[ \text{MA} = \text{NP} \]
\[ \text{coNP} = \text{coMA} \]

New topic(s)

Optimization problems, Approximation Algorithms, and Probabilistically Checkable Proofs
Optimization Problems

- many hard problems (especially \(\text{NP}\)-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 \(r \times \text{opt}\) for minimization
    at least \(\text{opt}/r\) 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?
  - \(\text{NP}\)-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 \(2 \times \text{opt} \geq \text{actual}\)

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: \((1 + \epsilon)\) for any \(\epsilon > 0\)

Approximation Algorithms

- \((\text{max})\) Knapsack: \((1 + \epsilon)\) for any \(\epsilon > 0\)
- called Polynomial Time Approximation Scheme (PTAS)
  - algorithm runs in poly time for every fixed \(\epsilon>0\)
  - poor dependence on \(\epsilon\) allowed
- If all \(\text{NP}\) optimization problems had a PTAS, almost like \(\text{P} = \text{NP}\) (!)
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 $L_2$

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 "$\geq k$" and "$< k/r$"
- Note: target problem is not a language
  - promise problem ($\text{yes} \cup \text{no}$ not all strings)
  - "promise": instances always from ($\text{yes} \cup \text{no}$)

Gap preserving reductions

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

Example gap-preserving reduction:
  - reduce MAX-k-SAT with gap $\epsilon$
  - to MAX-3-SAT with gap $\epsilon$
  - "MAX-k-SAT is NP-hard to approx. within $\epsilon$ \Rightarrow MAX-3-SAT is NP-hard to approx. within $\epsilon$"
- 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: φₙₓ
  – expected proof is satisfying assignment for φₙₓ
  – verifier picks random clause (“local test”) and checks that it is satisfied by the assignment
    \[ x ∈ L \implies \Pr[\text{verifier accepts}] = 1 \]
    \[ x ∉ L \implies \Pr[\text{verifier accepts}] < 1 \]

• 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: φₙₓ
  – expected proof is satisfying assignment for φₙₓ
  – verifier picks random clause (“local test”) and checks that it is satisfied by the assignment
    \[ x ∈ L \implies \Pr[\text{verifier accepts}] = 1 \]
    \[ x ∉ L \implies \Pr[\text{verifier accepts}] ≤ (1 – ε) \]
  – 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

*