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 $R$ such that:
  - $x \in L \Rightarrow \exists m \Pr_r[(x, m, r) \in R] = 1$
  - $x \notin L \Rightarrow \forall m \Pr_r[(x, m, r) \in R] \leq \frac{1}{2}$
- $L \in \text{AM}$ iff $\exists$ poly-time language $R$ such that:
  - $x \in L \Rightarrow \Pr_r[\exists m (x, m, r) \in R] = 1$
  - $x \notin L \Rightarrow \Pr_r[\exists m (x, m, r) \in R] \leq \frac{1}{2}$

Using PRGs for MA
- $L \in \text{MA}$ iff $\exists$ poly-time language $R$ such that:
  - $x \in L \Rightarrow \exists m \Pr_r[(x, m, r) \in R] = 1$
  - $x \notin L \Rightarrow \forall m \Pr_r[(x, m, r) \in R] \leq \frac{1}{2}$
- 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 $C_{x,m}$:
  - $\exists m \Pr_r[C_{x,m}(y) = 1] = 1$ ("yes")
  - $\forall m \Pr_r[C_{x,m}(y) = 1] \leq \frac{1}{2}$ ("no")

Derandomization revisited
- $L \in \text{MA}$ iff $\exists$ poly-time language $R$ such that:
  - $x \in L \Rightarrow \exists m \Pr_r[(x, m, r) \in R] = 1$
  - $x \notin L \Rightarrow \forall m \Pr_r[(x, m, r) \in R] \leq \frac{1}{2}$
- Recall PRGs:
  - $|\Pr_y[C(y) = 1] - \Pr_z[C(G(z)) = 1]| \leq \epsilon$
- For each circuit $C$ of size at most $s$:
  - $|\Pr_y[C(x, m, r) = 1] - \Pr_z[C(x, m, r) = 1]| \leq \epsilon$

Using PRGs for MA
- Can compute $\Pr_r[C_{x,m}(G(z)) = 1]$ exactly:
  - Evaluate $C_{x,m}(G(z))$ on every seed $z \in \{0,1\}^t$
  - Running time $O(|C_{x,m}| + \text{time for } G)2^t$
- $L \in \text{NP}$ if PRG with $t = O(\log n)$, $\epsilon < 1/2$

Theorem: E requires exponential size circuits $\Rightarrow \text{MA} = \text{NP}$. 
May 18, 2021 CS151 Lecture 15

**MA and AM**

(under a hardness assumption)

\[ \Pi_2 \quad \Sigma_2 \quad \Pi_2 \quad \Sigma_2 \]

\[ \text{AM} \quad \text{coAM} \quad \text{MA} \quad \text{coMA} \]

\[ \text{NP} \quad \text{coNP} \quad \text{P} \]

May 18, 2021 CS151 Lecture 15

**MA and AM**

(under a hardness assumption)

\[ \Pi_2 \quad \Sigma_2 \quad \Pi_2 \quad \Sigma_2 \]

\[ \text{AM} \quad \text{coAM} \quad \text{MA} \quad \text{coMA} \]

\[ \text{NP} \quad \text{coNP} = \text{coMA} \]

What about AM, coAM?

May 18, 2021 CS151 Lecture 15

**Derandomization revisited**

**Theorem** (IW, STV): If \( E \) contains functions that require size \( 2^{\Omega(n)} \) circuits, then \( E \) contains functions that are \( 2^{\Omega(n)} \)-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 \( t = O(\log n) \), and error \( \epsilon < 1/4 \).

May 18, 2021 CS151 Lecture 15

**Oracle circuits**

- **circuit** \( C \)
  - directed acyclic graph
  - nodes: AND (\( \land \)); OR (\( \lor \)); NOT (\( \neg \)); variables \( x_i \)

- **A-oracle circuit** \( C \)
  - also allow “A-oracle gates”

May 18, 2021 CS151 Lecture 15

**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 \( s(n)^{\Omega(n)} \)-size circuits. Contradiction.

May 18, 2021 CS151 Lecture 15

**Relativized versions**

- \( f : \{0,1\}^{\log k} \rightarrow \{0,1\} \)
  - small A-oracle circuit \( C \) approximating \( f \)

- \( \text{Decoding procedure} \)
  - \( i \in \{0,1\}^{\log k} \)
  - small A-oracle circuit computing \( f \) exactly

May 18, 2021 CS151 Lecture 15
Relativized versions

**Theorem**: if \( E \) contains \( 2^{\Omega(n)} \)-unapproximable functions, 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 \( A \)-oracle circuits, then use those circuits to compute hard function by size \( s \cdot n^k \) size circuits. Contradiction.

Using PRGs for AM

**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 \( \epsilon < 1/2 \).

**Theorem**: \( E \) requires exponential size SAT-oracle circuits \( \Rightarrow AM = NP \).
MA and AM
(under a hardness assumption)

\[
\begin{array}{c}
\Sigma_2 \\
\Pi_2 \\
AM = NP \\
\text{coNP} = \text{coMA} \\
\end{array}
\]

MA = NP = coMA

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

New topic(s)

Optimization problems, Approximation Algorithms, and Probabilistically Checkable Proofs

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 \( r \cdot \text{opt} \) for minimization
  at least \( \text{opt}/r \) for maximization

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 \(2 \ast \text{opt} \geq \text{actual}\)

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: \(\frac{n}{\log^2 n}\)
  – (max) Knapsack: \((1 + \epsilon)\) for any \(\epsilon > 0\)

Approximation Algorithms

• (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 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 \(L_2\)

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 (yes \& no not all strings)
  – “promise”: instances always from (yes \& no)
Gap producing reductions

- Main purpose:
  - $r$-approximation algorithm for $L_2$ distinguishes between $f(\text{yes})$ and $f(\text{no})$; 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

Example gap-preserving reduction:
- reduce MAX-$k$-SAT with gap $\epsilon$ to MAX-$3$-SAT with gap $\epsilon' = \frac{1}{3}$
- "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

Proof systems viewpoint

- $k$-SAT NP-hard $\Rightarrow$ for any language $L \in$ NP proof system of form:
  - given $x$, compute reduction to $k$-SAT: $\phi_x$
  - expected proof is satisfying assignment for $\phi_x$
  - verifier picks random clause ("local test") and checks that it is satisfied by the assignment
    - $x \in L \Rightarrow \Pr[\text{verifier accepts}] = 1$
    - $x \notin L \Rightarrow \Pr[\text{verifier accepts}] < 1$

Proof systems viewpoint

- MAX-$k$-SAT with gap $\epsilon$ NP-hard $\Rightarrow$ for any language $L \in$ NP proof system of form:
  - given $x$, compute reduction to MAX-$k$-SAT: $\phi_x$
  - expected proof is satisfying assignment for $\phi_x$
  - verifier picks random clause ("local test") and checks that it is satisfied by the assignment
    - $x \in L \Rightarrow \Pr[\text{verifier accepts}] = 1$
    - $x \notin L \Rightarrow \Pr[\text{verifier accepts}] \leq (1 - \epsilon)$
  - can repeat $O(1/\epsilon)$ times for error $< \frac{1}{2}$.
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 \( \varepsilon \)" 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 \( \varepsilon \) 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) \(x \notin L \Rightarrow \forall \text{ proof*} \)
    \[ \Pr[V(x, \text{proof*}) \text{ accepts}] \leq \frac{1}{2} \]

The PCP Theorem (AS, ALMSS):

PCP[log n, 1] = NP.

Corollary: MAX-k-SAT is NP-hard to approximate to within some constant \( \varepsilon \).

- 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 \( \varphi \) for verifier's test on each
  - note: k-CNF since function on only k bits
- "YES" VC instance \( \Rightarrow \) all clauses satisfiable
- "NO" VC instance \( \Rightarrow \) every assignment fails to satisfy at least \( \frac{1}{2} \) of the \( \varphi \) \( \Rightarrow \) fails to satisfy an \( \varepsilon = (\frac{1}{2})2^{k} \) 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