CS151
Complexity Theory

Lecture 15

May 22, 2017

New topic(s)

Optimization problems, Approximation Algorithms, and Probabilistically Checkable Proofs

May 22, 2017

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: $(1 + \varepsilon)$ for any $\varepsilon > 0$

Approximation Algorithms

(max) Knapsack: (1 + ε) for any ε > 0

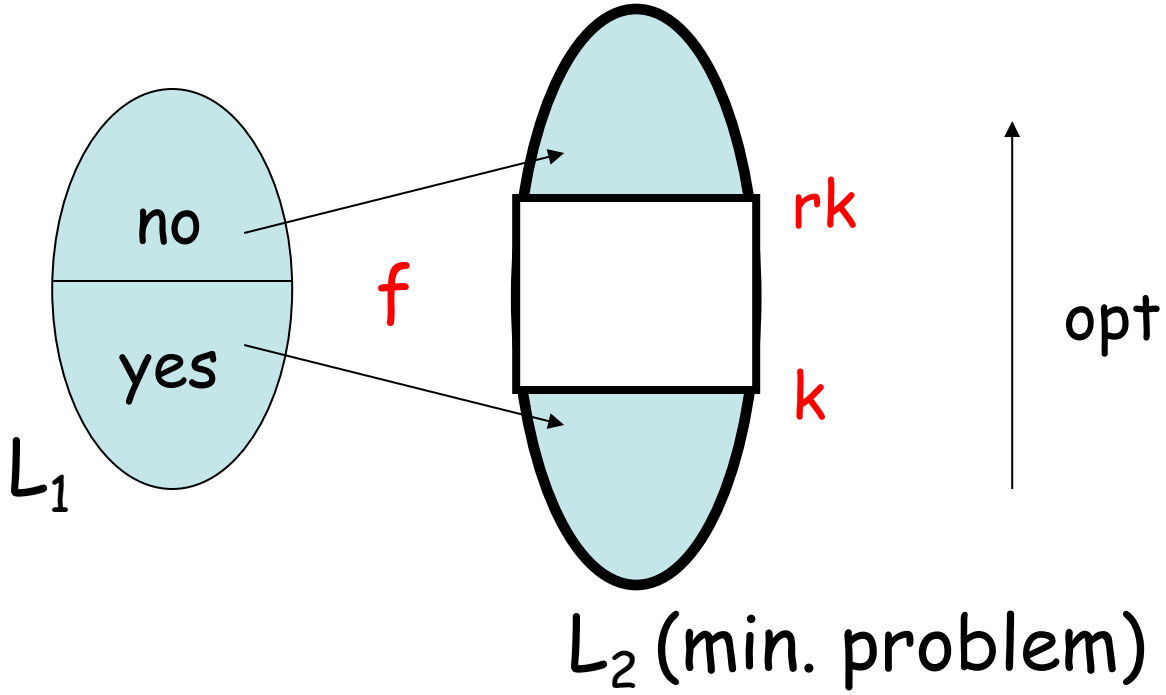
- called Polynomial Time Approximation Scheme (PTAS)
  - algorithm runs in poly time for every fixed ε>0
  - poor dependence on ε allowed
- If all **NP** optimization problems had a PTAS, almost like **P** = **NP** (!)

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

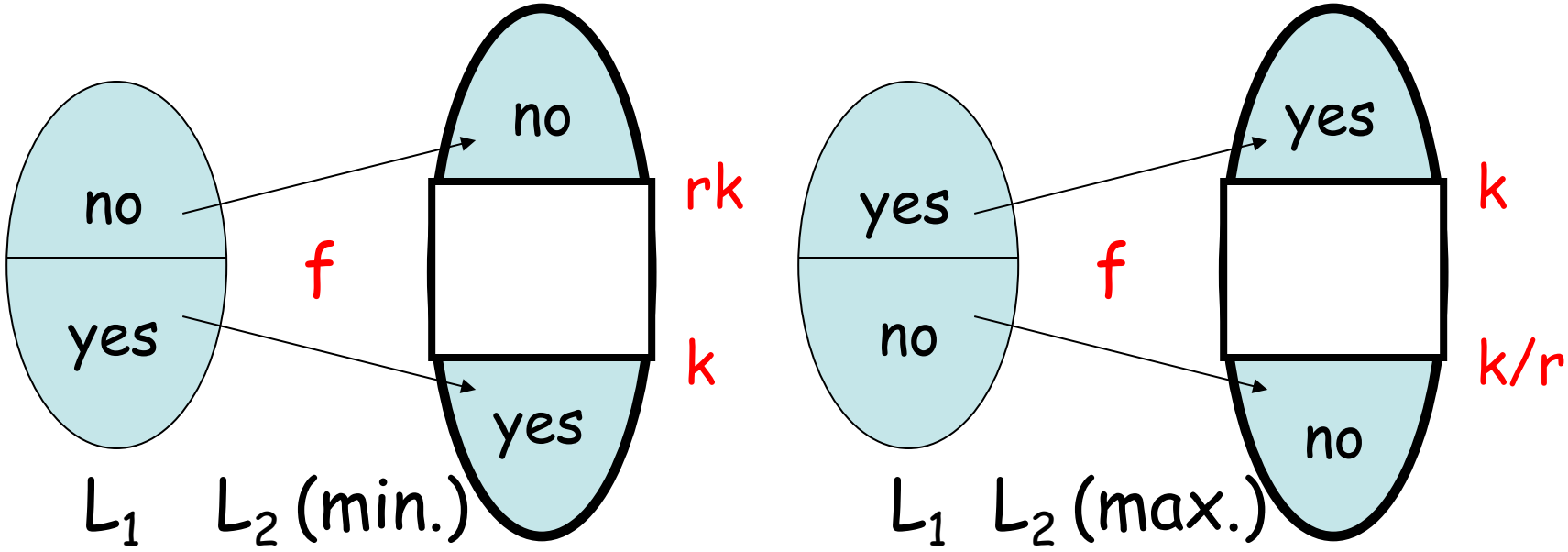
- "gap-producing" reduction from **NP**-complete problem L_1 to L_2



Gap producing reductions

- r-gap-producing reduction:
  - f computable in poly time
  - $x \in L_1 \Rightarrow opt(f(x)) \leq k$
  - $x \notin L_1 \Rightarrow opt(f(x)) > rk$
  - 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

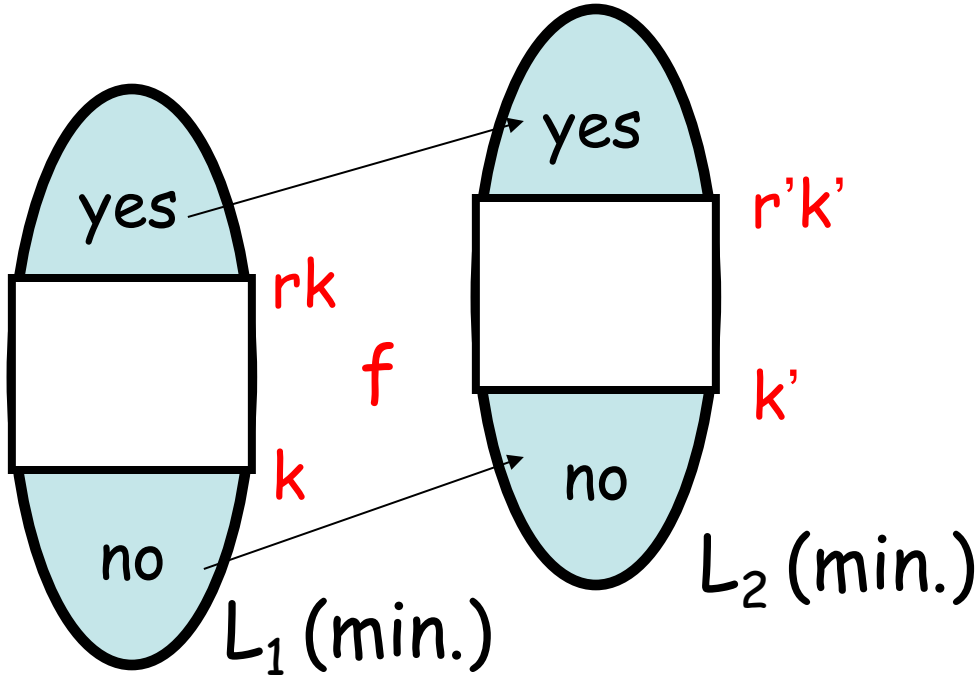


- Main purpose:
  - r-approximation algorithm for L_2 distinguishes between f(yes) and f(no); can use to decide L_1
  - "**NP**-hard to approximate to within r"

Gap preserving reductions

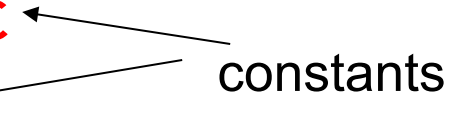
- gap-producing reduction difficult (more later)
- but **gap-preserving** reductions easier

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'



  - "MAX-k-SAT is NP-hard to approx. within $\varepsilon \Rightarrow$ 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 $\Rightarrow$ for any language L $\in$ **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

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

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

Proof systems viewpoint

- MAX-k-SAT with gap ε **NP**-hard $\Rightarrow$ for any language $L \in$ **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] $\leq (1 - \varepsilon)$
  - can repeat $O(1/\varepsilon)$ 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*

    $\Pr[V(x, \text{proof}^*) \text{ accepts}] \leq \frac{1}{2}$

PCP

- Two observations:
  - **PCP[1, poly n] = NP**

    proof?
  - **PCP[log n, 1] $\subset$ 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 $\Rightarrow$ all clauses satisfiable
- "NO" VC instance $\Rightarrow$ every assignment fails to satisfy at least ½ of the $\varphi_i \Rightarrow$ fails to satisfy an $\varepsilon = (½)2^{-k}$ 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

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 not prove

Proof Composition (idea)

NP ⊂ PCP[log n, polylog n] ("outer verifier")

NP ⊂ PCP[n^3, 1] ("inner verifier")

- **composition** of verifiers:
  - reformulate "outer" so that it uses O(log n) random bits to make **1** query to each of **3** provers
  - replies r_1, r_2, r_3 have length polylog n
  - Key: accept/reject decision computable from r_1, r_2, r_3 by small circuit C

Proof Composition (idea)

NP ⊂ PCP[log n, polylog n] ("outer verifier")
NP ⊂ PCP[n^3, 1] ("inner verifier")

- **composition** of verifiers (continued):
  - final proof contains **proof** that $C(r_1, r_2, r_3) = 1$ for inner verifier's use
  - use inner verifier to verify that $C(r_1,r_2,r_3) = 1$
  - O(log n)+polylog n randomness
  - O(1) queries
  - tricky issue: consistency

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 inner verifier

Theorem: **NP $\subset$ PCP[n^2, 1]**

Proof (first steps):

1. Quadratic Equations is NP-hard
2. PCP for QE:

   proof = *all quadratic functions* of a soln. x

   verification = check that a random linear combination of equations is satisfied by x

(if prover keeps promise to supply all quadratic fns of x)

Quadratic Equations

- quadratic equation over F_2:

$$\sum_{i<j} a_{i,j} X_i X_j + \sum_i b_i X_i + c = 0$$

- language QUADRATIC EQUATIONS (QE)

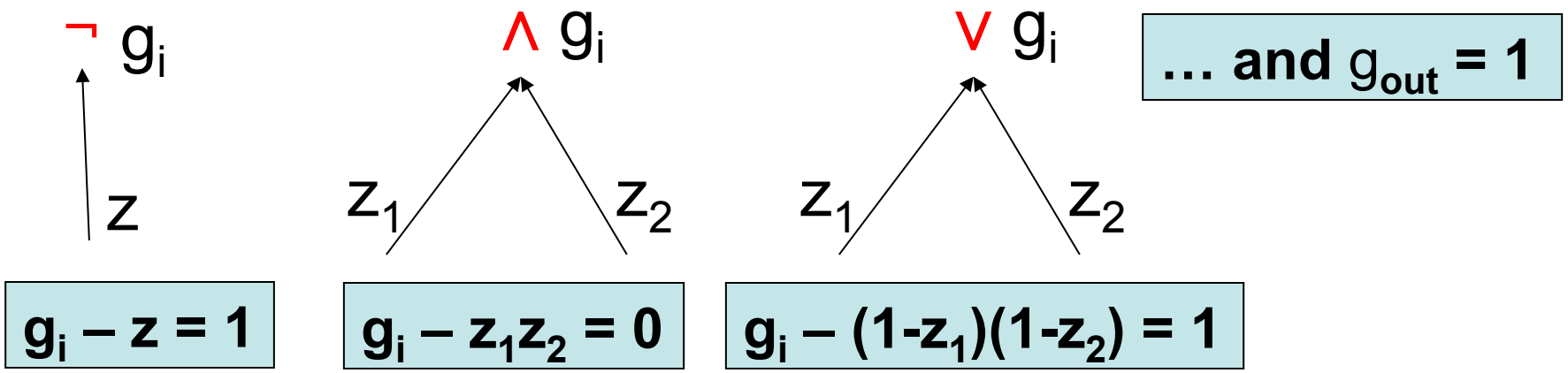
  = { systems of quadratic equations over F_2 that have a solution (assignment to the X variables) }

Quadratic Equations

Lemma: QE is NP-complete.

Proof: clearly in NP; reduce from CIRCUIT SAT

- circuit C an instance of CIRCUIT SAT
- QE variables = variables + gate variables



Quadratic Functions Code

- intended proof:
  - F the field with 2 elements
  - given $x \in F^n$, a solution to instance of QE
  - f_x: $F^n \rightarrow F_2$ all linear functions of x

$$f_x(a) = \sum_i a_i x_i$$

  - g_x: $F^{n \times n} \rightarrow F_2$ **includes** all quadratic fns of x

$$g_x(A) = \sum_{i,j} A[i,j] x_i x_j$$

  - KEY: can evaluate any quadratic function of x with a single evaluation of f_x and g_x

PCP for QE

If prover keeps promise to supply all quadratic fns of x, a solution of QE instance…

- Verifier's action:

–query a *random linear combination* R of the equations of the QE instance

–Completeness: obvious

–Soundness: x fails to satisfy some equation; imagine picking coeff. for this one last

Pr[x satisfies R] = 1/2

PCP for QE

$x \in F^n$ soln $\quad f_x(a) = \sum_i a_i x_i$ $\quad$ Had(x)

$g_x(A) = \sum_{i,j} A[i,j]x_i x_j$ $\quad$ Had($x \otimes x$)

To "enforce promise", verifier needs to perform:

- linearity test: verify f, g are (close to) linear
- self-correction: access the *linear* f', g' that are close to f, g

  [so f' = Had(u) and g' = Had(V)]
- consistency check: verify V = u $\otimes$ u

PCP for QE

$x \in F^n$ soln $\quad f_x(a) = \sum_i a_i x_i$ $\quad$ Had(x)

$g_x(A) = \sum_{i,j} A[i,j]x_i x_j$ $\quad$ Had($x \otimes x$)

- Linearity test: given access to $h:F^m \to F$
  - pick random a,b; check if $h(a) + h(b) = h(a+b)$; repeat O(1) times
  - do this for functions f and g supplied by prover

Theorem [BLR]: h linear$\Rightarrow$prob. success = 1; prob. success $\geq 1 - \delta \Rightarrow \exists$ linear h' s.t.

$$Pr_a [h'(a) = h(a)] \geq 1 - O(\delta)$$

PCP for QE

$x \in F^n$ soln	$f_x(a) = \sum_i a_i x_i$	Had(x)
	$g_x(A) = \sum_{i,j} A[i,j]x_i x_j$	Had($x \otimes x$)

- Self-correction:
  - given access to $h:F^m \to F$ close to linear h'; i.e.,

$$Pr_a [h'(a) = h(a)] \geq 1 - O(\delta)$$

  - to access h'(a), pick random b; compute

$$h(b) + h(a+b)$$

  - with prob. at least $1 - 2 \cdot O(\delta)$, h(b) = h'(b) and h(a+b) = h'(a+b); hence we compute h'(a)

PCP for QE

$x \in F^n$ soln $\quad f_x(a) = \sum_i a_i x_i$ $\quad$ Had(x)

$g_x(A) = \sum_{i,j} A[i,j]x_i x_j$ $\quad$ Had($x \otimes x$)

- Consistency check: given access to linear functions $f' = Had(u)$ and $g' = Had(V)$
  - pick random $a, b \in F^n$; check that

$$f'(a)f'(b) = g'(ab^T)$$

  - completeness: if $V = u \otimes u$

$$f'(a)f'(b) = (\sum_i a_i u_i)(\sum_i b_i u_i) = \sum_{i,j} a_i b_j V[i,j] = g'(ab^T)$$

PCP for QE

$x \in F^n$ soln $\quad f_x(a) = \sum_i a_i x_i$ $\quad$ Had(x)

$g_x(A) = \sum_{i,j} A[i,j] x_i x_j$ $\quad$ Had($x \otimes x$)

- Consistency check: given access to linear functions $f' = Had(u)$ and $g' = Had(V)$
  - soundness: claim that if $V \neq u$

$$\Pr[(\sum_i a_i u_i)(\sum_i b_i u_i) = \sum_{i,j} a$$

$\exists$ i,j s.t. uu^T and V

$\exists$ i s.t. $(uu^T)b$ and Vb differ in entry i; pick a_i last

$$\Pr[(uu^T)b \neq Vb]$$

$$\Pr[a^T(uu^T)b \neq a^TVb] \geq \tfrac{1}{2} \cdot \tfrac{1}{2} = \tfrac{1}{4}$$