Probabilistically Checkable Proofs + Hardness of Approximation

Note Title 5/21/2013

Many hard problems (especially NP-hard problems) are optimization problems. (minimization or maximization)

e.g. smallest tour, smallest VC, largest I.S.

"opt" = value of the optimal solution.

Is approximating opt good enough?

- there are lots of heuristics.
- we want an approximation algorithm that guarantees some approximation ratio r.

on every input x

$$\frac{opt}{r} \le \text{answer} \le opt \quad (\text{for maximization})$$

$$opt \le \text{answer} \le r \cdot opt \quad (\text{for minimization})$$

Vertex Cover: Input: undirected graph $G = (V, E)$.
What is the smallest subset of the vertices that touches every edge?

Vertex Cover (VC) is NP-complete.

Here's an approximation algorithm:

pick an edge (x, y), add $x + y$ to VC
discard all edges incident to x or y.
Continue until no edges remain.

approx ratio for this simple algorithm is 2:

For each edge considered, we add 2 vertices.
Edges considered do not share any endpoints
opt $\ge$ 1 vertex per edge considered.
$opt \ge (\text{our VC})/2$.

There is a diverse array of ratios achievable.

Vertex Cover: 2.

MAX-3-SAT: 8/7 (find assignment that satisfies the most clauses).

Set Cover: $\log n$.

Knapsack: $(1+\epsilon)$ for any $\epsilon > 0$.

Clique: $n/\log^2 n$.

Best Case:

Poly-time approximation scheme (PTAS)

for every $\epsilon > 0$

$(1+\epsilon)$ ratio can be achieved

the approx algorithm runs in poly(n) where n = input size

but may depend exponentially on $1/\epsilon$.

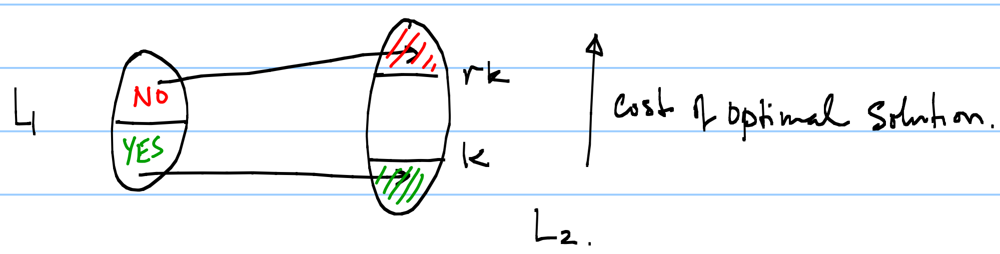
How do we explain the failure to improve some of these?

(How to explain the failure to find a poly-time alg for SAT?)

Is it NP-hard to improve an approximation ratio?

In order to prove a statement like this, we need a "gap-producing" reduction from L_1 to L_2:

Minimization:



r-gap producing reduction: f: poly-time computable.

$x \in L_1 \Longrightarrow \text{Opt}(f(x)) \leq k.$

$x \notin L_1 \Longrightarrow \text{opt}(f(x)) > rk.$

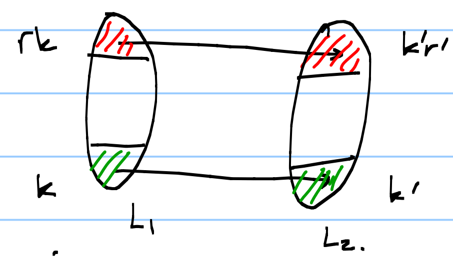
The target problem is not a language. $\to$ opt $\geq rk$ $\to$ opt $\leq k$.

It is a promise problem. (Yes $\cup$ No instances are not all strings)

The algorithm is promised that the input comes from the set of Yes or No instances.

<u>Purpose</u>: r-approximation alg for L_2 distinguishes between f(yes) + f(no). This can be used to decide L_1.

Note that if it is NP-hard to compute with the promise, then it's also hard to compute without it.



Gap producing reductions are difficult.

Gap preserving reductions are easier.

many typical reductions are <u>not</u> gap preserving.

For example the standard reduction of k-SAT to 3-SAT is gap preserving. Max-k-SAT w/ gap ϵ reduces to Max-3-SAT w/ gap ϵ'.

If it's hard to approximate Max-k-SAT to within ϵ, then it's hard to approximate Max-3-SAT to within ϵ'.

MAXSNP (Papadimitriou + Yannakakis) a set of problems reducible to each other in this way.

A PTAS for a MAX-SNP-complete problem gives a PTAS for all the problems in MAX-SNP.

Missing piece: first gap-producing reduction.

Consider Max-k-SAT w/ promise gap ϵ.

input: instance ϕ k-CNF.

YES: Some assignment satisfies all the clauses.

NO: no assignment satisfies more than a $(1-\epsilon)$ of the clauses.

Let's look at a Proof System view of this problem.

Suppose there is a reduction from an NP-hard problem to Max-k-SAT w/ promise gap ϵ.

Then the following protocol will solve the NP-hard problem:

given x compute reduction to k-SAT ϕ_x

The verifier expects that the proof is a satisfying assignment to ϕ_x.

Verifier picks random clause ("local test")

and checks that it is satisfied by the assignment.

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

$$x \notin L \Rightarrow \Pr[V \text{ accepts}] < 1-\epsilon.$$

This can be repeated $O(1/\epsilon)$ times for error $< 1/2$.

Note: Prover commits to the whole proof.

Verifier only looks at part of the proof.

$\Rightarrow$ Verifier does only local test. An ϵ fraction of these tests will notice the invalidity.

$\hookrightarrow$ looking for $\epsilon = 1/\text{poly}(n)$

<u>PCP</u> Probabilistically Checkable Proof

Novel way of verifying a proof.

- pick a random local test (chosen w/ random bits)
- query proof in specified k locations

$\to$ then accept if the proof passes the test.

PCP [$r(n)$, $q(n)$] set of languages w/ p.p.t. verifier V that has (r,q) restricted access to the proof.

V tosses $O(r(n))$ coins.

V access the proof in $O(q(n))$ locations (bits).

(Completeness) $x \in L \Rightarrow \exists$ proof s.t.

$\Pr[V(x, \text{proof}) \text{ accepts}] = 1.$

(Soundness) $x \notin L \Rightarrow \forall$ proof*

$\Pr[V(x, \text{proof}) \text{ accepts}] \leq 1/2.$

Observations PCP $[1, \text{poly}(n)] = NP$ verifier sees the entire poly-size proof.

PCP $[\log n, 1] \subseteq NP$ → run through all $\log n$ possibilities.

witness y is the set of all bits examined for each random choice.

Can calculate prob of acceptance explicitly.

PCP Theorem

PCP $[\log n, 1] = NP$

Any problem in NP has a $[\log n, 1]$ prob. checkable proof.

How does this relate to hardness of approximations?

Corollary: Max-k-SAT is hard to approximate to within some constant ϵ.

Pf of Cor: Use PCP $[\log n, 1]$ protocol for some NP-hard problem.

variables in ϕ: $x_1 \cdots x_m$ bits of the proof.

$m \leq 2^{O(\log n)} \cdot O(1)$

⇒ Enumerate all $2^{O(\log n)}$ = poly (n) sets of queries.

⇒ Construct a k-CNF ϕ_i for verifier's test on each. ⇒ k-CNF function on only k bits.

query $i \in \{0,1\}^{O(\log n)} \longrightarrow \phi_i$ acc/rej as a fcn of the k-bit response.

there are $\leq 2^k$ clauses in ϕ_i. They are all satisfied iff Verifier accepts.

"Yes" instance of VC $\Rightarrow$ all clauses satisfied.

"No" instance of VC $\Rightarrow$ every assignment fails to satisfy $\leq 1/2$ of the ϕ_i.

ϕ_i not satisfied $\Rightarrow$ ≥ 1 unsatisfied clause.

$\#$ unsat clauses $\geq \underbrace{1/2 (2^{-k})}_{\epsilon.}$ fraction