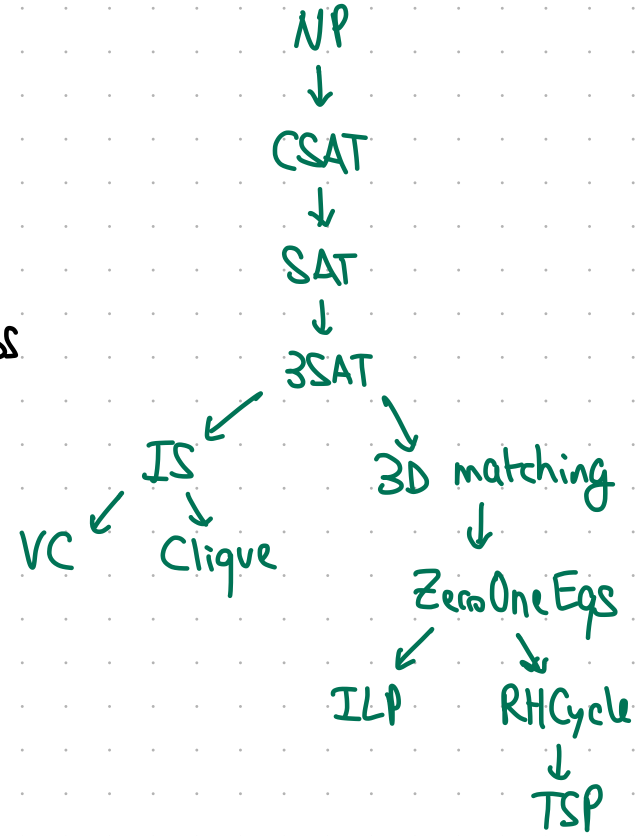
LECTURE #21

CS 170

Spring 2021

Last time(s):

- reductions
- NP-hardness (and NP-completeness)
- reductions to establish NP-hardness & several natural problems:



Today:

How to cope with NP-hardness?

You are interested to solve a computational task A

Try to show that $A \in P$. (directly, or reduce to ShortestPaths, MaxFlow, LP, ...)

If you succeed then good.

Otherwise, try to show that A is NP-hard. (reduce from 3SAT, ...)

You are likely to succeed. (few problems are not believed to be in P nor NP-hard)

What to do if A is NP-hard?

A. find a special case of A that is in P (NP-hardness uses abstruse instances)

B. intelligent exponential search (mitigate the exponential)
via techniques such as backtracking, branch and bound, ...

C. use an approximation algorithm
efficient and incorrect but not by much

D. use heuristics: no guarantees on running time or approximation,
but informed by intuition of problem and inputs of interest

Approximation Algorithms for Optimization Problems

input: instance $x \in \mathcal{I}$, which induces a solution space S_x and value function $\mathrm{val}_x(\cdot)$

output: $s^* \in S_x$ s.t. $\mathrm{val}_x(s^*) = \mathrm{opt}(x)$ $\left(\max_{s \in S_x} \mathrm{val}_x(s), \text{ or } \min_{s \in S_x} \mathrm{val}_x(s) \right)$

Ex: maximum independent set, smallest-weight tour, ...

The approximation ratio of an algorithm A is

- for maximization problems: $\alpha(A) := \max_{x \in \mathcal{I}} \dfrac{\mathrm{opt}(x)}{\mathrm{val}_x(A(x))} \in [1, \infty)$

- for minimization problems: $\alpha(A) := \max_{x \in \mathcal{I}} \dfrac{\mathrm{val}_x(A(x))}{\mathrm{opt}(x)} \in [1, \infty)$

New goal: $\boxed{\text{design efficient algorithms for NP-complete problems with as small approximation ratio as possible}}$

Vertex Cover

$S \subseteq V$ is a vertex cover if $\forall e \in E\ \exists v \in S$ that is an endpoint of e

input: undirected graph $G=(V,E)$
output: vertex cover $S \subseteq V$
goal: minimize $|S|$

VC is a special case of SetCover (given $S_1, \ldots, S_m \subseteq U$, find smallest $I \subseteq [m]$ s.t. $\bigcup_{i \in I} S_i = U$):
set $U := E$ and $S_i :=$ "edges incident to vertex i".

VC is NP-hard: VC reduces to the NP-hard problem IS (S is a vertex cover iff $V \setminus S$ is an independent set)

theorem: VC has an approximation algorithm with approx ratio $= 2$

Idea: exploit a connection to matchings

def: $M \subseteq E$ is a *matching* if edges in M don't share vertices

claim: $S \subseteq V$ vertex cover, $M \subseteq E$ matching $\Rightarrow |M| \le |S|$ $\left(\text{hence } \max_M |M| \le \min_S |S|\right)$

proof: $\forall e \in M \; \exists v \in S$ that touches e (and no other edge) $\blacksquare$

def: For $M \subseteq E$ define $V(M) :=$ all endpoints of edges in M.

claim: $M \subseteq E$ maximal matching (cannot add more edges) $\Rightarrow V(M)$ vertex cover of size $2|M|$

proof: Since M is a matching, we know that $V(M)$ has $2|M|$ vertices. Moreover if $V(M)$ is not a vertex cover then $\exists e \in E$ not touched by $V(M)$, and so can add e to M. $\blacksquare$

This leads to a simple algorithm:

arbitrarily add edges to $\tilde{M}$ until no longer a matching

$$\boxed{\begin{aligned} A(G) := \; & 1.\ \text{Find a maximal matching } \tilde{M} \text{ in } G. \\ & 2.\ \text{Output } S := V(\tilde{M}). \end{aligned}}$$

- A outputs a vertex cover & is efficient
- A has approx ratio 2: $\dfrac{val_G(A(G))}{opt(G)} = \dfrac{|V(\tilde{M})|}{\min_S |S|} = \dfrac{2|\tilde{M}|}{\min_S |S|} \le \dfrac{2|\tilde{M}|}{\max_M |M|} \le \dfrac{2|\tilde{M}|}{|\tilde{M}|} = 2.$

Hardness of Approximation

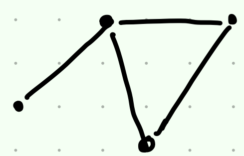
Not every NP-hard problem has approximation ratio 2.

claim: if TSP has approx ratio 2 then P = NP

[we believe that $P \neq NP$
$\Rightarrow$ we believe that TSP cannot be approximated to within factor 2]

proof: We show how to solve HamCycle (which is NP-complete) in polynomial time.

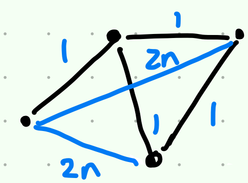
instance of HamCycle



$\longmapsto$

length 1 on existing edges
length $2n$ on all others

instance of TSP



If $G \in$ HamCycle then G' has tour of length n.

If $G \notin$ HamCycle then every tour must use at least one new edge and so must have length at least $(n-1) \cdot 1 + 1 \cdot 2n = 3n-1$.

An algorithm for TSP with approx ratio 2 can tell the difference. $\blacksquare$

The same argument also rules out any approx ratio $\alpha(n)$ that is poly-time computable! (E.g. $\alpha(n) = 2^n$.)

The study of inapproximability involves beautiful tools. See $\nearrow$

Inapproximability of Combinatorial Optimization Problems

Luca Trevisan*

February 21, 2010

Heuristics

Say that we want to find maximum of $f: \mathbb{R} \to \mathbb{R}$.

Naive idea: try inputs to f at random ← this will not get us far

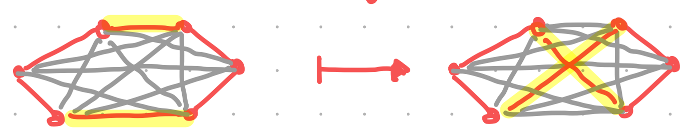
Better idea: follow the "up" direction (until you reach a maximum or get tired)

this is a fundamental idea from optimization known as GRADIENT ASCENT

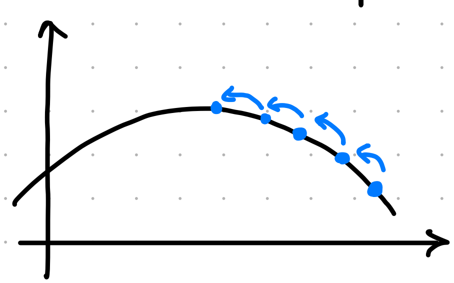
```
z := random starting point
repeat M times
    z' := random point near z
    if val(z') > val(z) : z := z'
```

- M (# iterations) is chosen heuristically
- "near" means from a neighborhood of z, and choosing this definition matters a lot

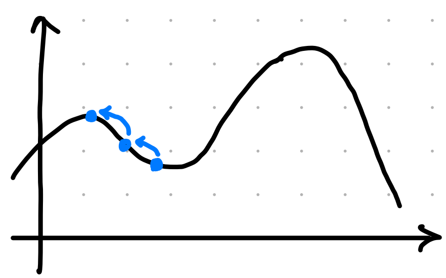
Eg for TSP: pick two edges at random & cross them



The behavior depends on how f looks:



finds maximum



may find max after retrying

gradient ascent works badly

Simulated Annealing

Idea: move to worse options with some probability

Fix a temperature schedule: probabilities $p_1 > p_2 > \cdots > p_N$ with an exponential decay.

The algorithm is:

```
z := random starting point
for i = 1, 2, 3, ..., N:
    repeat M times:
        z' := random point near z
        if val(z') > val(z): z := z'
        else w.p. p_i^(val(z) - val(z')): z := z'
```

- initially (small i), the algorithm is quite random because p_i is large
- as i increases, the algorithm looks more and more like gradient ascent (and spends more time where values are larger)

A reasonable first attempt to solve an NP-complete problem.