Lec26-NP-hardness

KT 8.1, 8.2, 8.3, 8.4

Friday, November 15, 2024 7:26 PM

Polynomial time: e.g. $O(m^2 n)$

Often we want poly-time because exp algs $O(2^n)$ are too slow.

Mistake: NP-hard $\neq$ non-polynomial time

not the correct definition

Decision vs. Optimization problems

Optimization: output answer that minimizes / maximizes an objective function

Ex. All of linear programming.

Shortest path

Minimum spanning tree.

Max - Flow

Decision: Yes/No answer to whether there exists a solution that is good enough

Ex $\exists$ s-t path shorter than 5

$\exists$ MST with weight ≤ 100

Ex. Circulation with demands

Optimization $\rightarrow$ Decision

Problem (MST) Given a graph $G = (V, E)$, with weights w_e on each edge in E, find a spanning tree of smallest total weight.

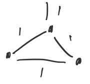
Problem (MST-Decision) Given a number C, and a graph $G = (V, E)$, with weights w_e on each edge in E, is there a spanning tree of total weight less than C

Glossary

Problem: a decision problem in general terms

Instance: A particular set of input into a problem:

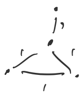
e.g. $C = 5$ $G =$



Yes-Instance: An instance with answer "YES"

No-Instance: An instance with answer "NO"

$C = 2$ $G =$



deciding: determining if an Instance has answer "YES" or "NO"

Define: The class P = set of problems.

A problem $Q \in P$ if

$\exists$ algorithm A_Q s.t. $\forall$ instances $I \in Q$,

$A_Q(I)$ runs in fewer than poly$(|I|)$ steps, and s.t.

if I is a YES-instance, $A_Q(I) = YES$

if I is a NO-instance, $A_Q(I) = NO$.

on a deterministic Turing machine

P is the set of problems that can be decided in polynomial number of steps. i.e "efficient" solutions.

Define: The class $\boxed{N}P$ = set of problems.

A problem $Q \in NP$ if a non-deterministic Turing machine can return YES on a YES instance in poly-time.

Easier Definition: A problem $Q \in NP$ if evidence of a YES-instance can be verified in poly-time. (on normal Turing machine)

certificate

Ex. (Traveling Salesman- Decision)

Given distances d_{ij} between n cities, and a number C, where is there a way to visit each city exactly once the total length of the route is $\leq C$.

If someone gives you a route, it is easy to check if it provides a solution that makes this a YES-instance.

Ex. (Independent Set -Decision)

Given graph G, $\exists S \subseteq V$, $|S| \geq k$ s.t. no two nodes in S are connected by an edge?

Finding set is hard. (later)

Checking a set S^* I give you is easy.

S^* acts as a certificate that $\langle G, k \rangle$ is a Yes-instance

Define: NP = set of problems Q for which

$\exists$ algorithm C_Q s.t.

$\forall$ instances $I \in Q$,

if I is yes-instance, $C_Q(I, S) = YES$ for some S

if I is no-instance, $C_Q(I, S) = NO$ $\forall S$.

and $\underbrace{C_Q(I, S)}_{\text{verifier}}$ runs in poly$(|I|)$ time.

(efficient certifier)

certificate

(nondeterministic polynomial)

NP - set of problems with efficient certifier

P - set of problems with efficient certifier that ignores the certificate

(polynomial)

Thm $P \subseteq NP$.

proof. Suppose $X \in P$. Then $\exists$ poly-time alg A that decides X.

To show $X \in NP$, need to design efficient certifier $C_X(I, S)$.

Just take $C_X(I, S) = A(I)$. $\square$

every problem with poly-time alg is in NP.

Question: $P \overset{?}{=} NP$

Know $P \subseteq NP$.

Does there $\exists$ a problem in NP that is NOT in P?

Is checking a solution actually easier than finding a solution? Intuitively maybe.

No one knows answer, but most people believe it to be true.

Reductions as tool for hardness

We want to say some problems are hard.
Easier 1st step: Problem X is at lest as hard as Problem Y.

Proof technique: we reduce Problem Y to Problem X:

Suppose we have black box that solves instances of X.
How can you solves instances of Y using poly steps
+ poly calls to black box.

If Y can be reduced to X, $Y \leq_p X$

Y is poly-time reducible to X

$\Rightarrow$ X is at least as hard as Y because if you can solve X, you can solve Y.

Note: reduce $\underline{\underline{\text{to}}}$ problem we want to show is harder

Examples:

Max Bipartite Matching $\leq_p$ Max Network Flow

Image Segmentation $\leq_p$ Min-Cut

Max Network Flow $\leq_p$ Linear Programming

Circulation w/ Demands & Lower Bounds $\leq_p$ Circulation w/ Demands $\leq_p$ Max Flow

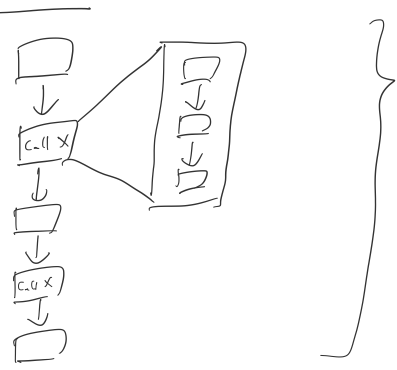
If ILP can be reduced to TSP, $ILP \leq_p TSP$

$\Rightarrow$ if we can solve TSP, we can solve ILP.

Suppose $Y \leq_p X$ and $\exists$ poly-time alg for X.
Then $\exists$ poly-time alg for Y.

Why? Polynomials compose

Alg for Y



all steps are poly-time

Thm: If $Y \leq_p X$, and Y $\underline{\text{CANNOT}}$ be solved in poly-time, then X cannot be solved in poly-time.

proof. Suppose we have alg. A that solves X in poly-time.
Then we reduce Y to X, giving a solution to Y in poly time a contradiction.

So if we can find a hard problem Y, can show a bunch of other problems are hard by reducing Y to those other problems

---

Define: A problem X is NP-hard if $\forall Y \in NP$, $Y \leq_p X$.

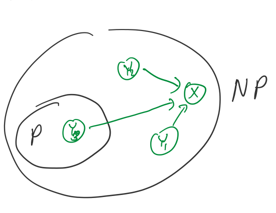
In other words, these are problems that are at least as hard as every problem in NP.

Lemma: If X is NP-hard, and $X \leq_p Z$, then Z is NP-hard.

proof. If we can reduce any $Y \in NP$ to X, and we can reduce X to Z, then clearly, we can reduce Y to Z.

Define: X is NP-complete. if
- $X \in NP$
- X is NP-hard.

X can be used to solve every other problem in NP.



Thm. If X is NP-complete, then X is solveable in poly time iff $P = NP$.

proof. If $P = NP$, then $X \in NP = P$, so X is solveable in poly-time

Suppose $X \in P$. Then $\forall Y \in NP$,
$Y \leq_p X$ so we can solve Y by reducing it
in polytime to X, which we can also solve in poly-time,
So we can solve Y in poly-time.
$\Rightarrow Y \in P$, so $P = NP$.

Thm If Y is NP-complete, $X \in NP$, and $Y \leq_p X$, then
X is NP-complete.

proof. Let $Z \in NP$. Since Y is NP-complete, $Z \leq_p Y \leq_p X$ ▨