Algorithms Design.
Georgia Institute of Technology.

Introduction to the class NP.

What it means for a problem to be hard?

- A typical problem asks for a solution out of an *exponentially* large set of candidates.
- We (human kind) don't have the time to check each candidate until we find a solution.
- Sometimes this is the best we can do!

Search problems

What is a problem?

Given an instance $\mathcal{I}$ we need to find a solution S for it, or report if such solution does not exists.

Search problems

Search problems

Given an instance $\mathcal{I}$, and a **candidate solution** S' we can confirm in polynomial time* that S' is indeed a solution.

Search problems

Search problems

Given an instance $\mathcal{I}$, and a **candidate solution** S' we can confirm in polynomial time* that S' is indeed a solution.

(*) Polynomial in the size of the input $|\mathcal{I}|$.

The classes P and NP

NP= set of all search problems.

P= subset of all search problems that can be solved in polynomial time.

$$P \subseteq NP$$

Examples.

Problem: ($K-$coloring) Given an integer $K > 0$ and a graph $G = (V, E)$, return a coloring of V with at most K colors such that every edge gets different colors on its end vertices.

Examples.

Problem: ($K-$coloring) Given an integer $K > 0$ and a graph $G = (V, E)$, return a coloring of V with at most K colors such that every edge gets different colors on its end vertices.

$K-$coloring is in NP: given an instance (i.e.: K and a graph) and a candidate solution (an assignment of at most K colors to the vertices of G!) loop through the edges and compare the colors of the end vertices. $O(m)$

Examples.

Problem: (SAT) Given a boolean formula in *conjunctive normal form** find an assignment of the variables that evaluates to true or return NO if such assignment does not exist.

Examples.

conjunctive normal form

$$f(x_1, x_2, \ldots, x_n) \to \{0, 1\}$$

Each x_i is a boolean variable: $x_i \in \{0, 1\}$.
f is the intersection (AND, denoted by $\wedge$) of m clauses, each been a disjunction (OR, denoted by $\vee$) of *literals*. Each literal is equal to some x_i or its negation $\bar{x}_i$.

$$f = (x_1 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee x_5) \wedge (\bar{x}_1) \wedge (\bar{x}_2 \vee \bar{x}_3).$$

$x_1 = 0,\ x_4 = 1,\ x_2 = 1,\ x_3 = 0,\ x_5 = 1.$

Examples.

Problem: (SAT) Given a boolean formula in *conjunctive normal form** find an assignment of the variables that evaluates to true or return NO if such assignment does not exist.

<u>SAT is in NP:</u> Given an instance (the boolean function f) and a candidate solution S (an assignment of the variables) we can evaluate each clauses in time $O(n)$ and conclude S is a solution if all return true. Since there are m clauses this takes $O(mn)$.

Examples.

Problem: (MST) Given a weighted, undirected graph $G = (V, E)$, find a minimum spanning tree, or return NO if such tree does not exist.

Examples.

Problem: (MST) Given a weighted, undirected graph $G = (V, E)$, find a minimum spanning tree, or return NO if such tree does not exist.

<u>MST is in NP:</u> Given an instance (weighted and undirected graph $G = (V, E)$) and a candidate solution S (a subgraph of G) we must check it is a MST.

Examples.

MST is in NP: Given an instance (weighted and undirected graph $G = (V, E)$) and a candidate solution S (a subgraph of G) we must check it is a MST:

1. S is a tree.
2. S is spanning.
3. S is minimum.

Examples.

MST is in NP: Given an instance (weighted and undirected graph $G = (V, E)$) and a candidate solution S (a subgraph of G) we must check it is a MST:

1. S is a tree. Run DFS on S and check for back edges!* $O(n + m)$.
2. S is spanning.
3. S is minimum.

(*) This tell us that S is cycle-free, not a tree!

Examples.

MST is in NP: Given an instance (weighted and undirected graph $G = (V, E)$) and a candidate solution S (a subgraph of G) we must check it is a MST:

1. S is a tree. Run DFS on S and check for back edges! $O(n+m)$.
2. S is spanning. Run Explore on S and check every vertex has been visited. $O(n+m)$.
3. S is minimum.

Connectivity and cycle-free imply we have a spanning tree.

Examples.

MST is in NP: Given an instance (weighted and undirected graph $G = (V, E)$) and a candidate solution S (a subgraph of G) we must check it is a MST:

1. S is a tree. Run DFS on S and check for back edges! $O(n + m)$.
2. S is spanning. Run DFS on S and check every vertex has been visited. $O(n + m)$.
3. S is minimum. Run Kruskal's algorithm on G to get a MST T. Check if $\omega(T) = \omega(S)$. $O(m \log(n))$.

Examples.

Problem: (Knapsack) Given a list of n objects along with their weights and values, and a capacity B outputs the value of the maximum profit you can make.

object	1	2	...	n
weight	w_1	w_2	...	w_n
value	v_1	v_2	...	v_n

Want a subset $S \subseteq [n]$ such that:

$$\sum_{i \in S} v_i \text{ is maximal while } \sum_{i \in S} w_i \leq B.$$

Examples.

Knapsack is in NP: Given an instance (objects, weights, values, capacity) and a candidate solution S (a subset of the objects) we must check it maximizes the profit.

Examples.

Knapsack is in NP:

- Best solution we know runs in exponential time! (cannot find a solution like MST).
- There are exponentially many subsets of $[n]$.

Examples.

Knapsack is not in NP:

- Best solution we know runs in exponential time! (cannot find a solution like MST).
- There are exponentially many subsets of $[n]$.

The class NP.

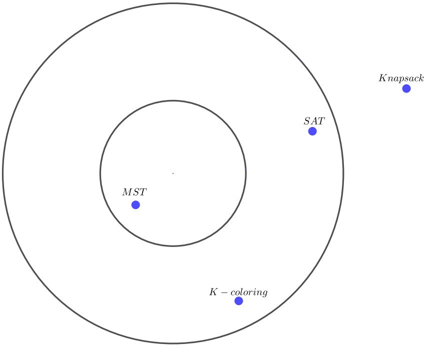


Figure: The class NP and the problems from the examples.

The hardest problems in NP.

Informal idea: a problem A is hard if solving it in polynomial time implies we can solve **all** problems in NP also in polynomial time.

Reductions.

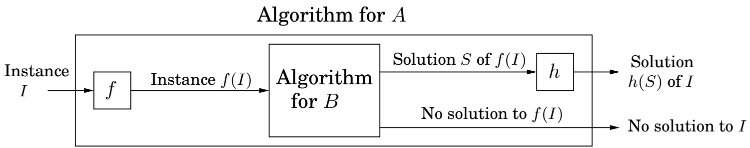
Definition

Given two problems A and B. We said A reduces to B if there are two polynomial time algorithms f and h such that f maps an instance $\mathcal{I}$ of A to an instance $f(\mathcal{I})$ of B and h maps a solution S of $f(\mathcal{I})$ back to a solution $h(S)$ of $\mathcal{I}$.

We write $A \to B$.

Reductions.

- If $A \rightarrow B$ an algorithm to solve B can be transform into an algorithm to solve A.
- The following holds: $\mathcal{I}$ has a solution if and only if $f(\mathcal{I})$ has a solution.



The hardest problems in NP.

<u>Informal idea</u>: a problem A is hard if solving it in polynomial time implies we can solve **all** problems in NP also in polynomial time.

Definition

A problem B is said to be `NP-hard` if for any $A \in \mathrm{NP}$ we have $A \to B$. If $B \in \mathrm{NP}$ is `NP-hard` we said it is `NP-complete`.

The hardest problems in NP.

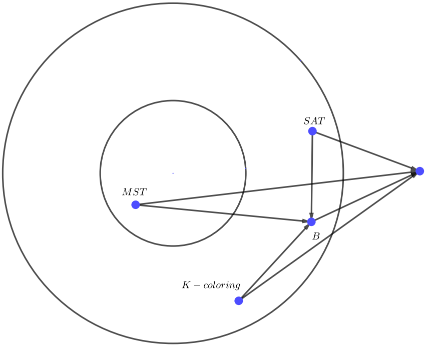
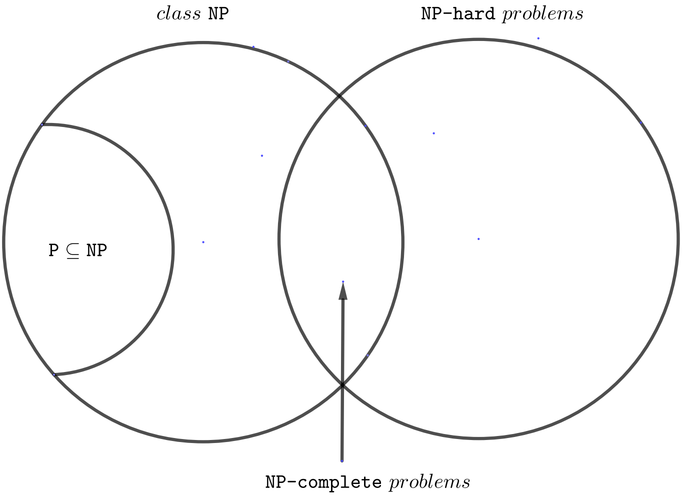


Figure: Two `NP-hard` problems. Since $B \in$ NP it is `NP-complete`.

Complexity classes.



The big question.

Is P=NP?

We know all problems in the class NP reduce to any NP-hard problem.

Solving **one** NP-hard problem in polynomial time implies P=NP.

How to determine if a problem is hard?

Lemma

Let A be $\texttt{NP-hard}$ and $A \to B$. Then B is also $\texttt{NP-hard}$.

Proof: Note that for any three problems $\{X, Y, Z\}$, if $X \to Y$ and $Y \to Z$ then $X \to Z$.
So, for any problem $C \in \texttt{NP}$:

$$C \to A \to B.$$

SAT is NP-complete.

Cook-Levin Theorem (1971)

SAT is NP-complete.

In 1972, Richard E. Karp published a paper listing many *new* NP-hard problems.