

GEMS OF TCS

P VS NP

Sasha Golovnev

March 16, 2021

Search Problems

SEARCH PROBLEM

Definition

A **search problem** is defined by an algorithm \mathcal{C} that takes an instance I and a candidate solution S , and runs in time polynomial in the length of I . We say that S is a solution to I iff $\mathcal{C}(S, I) = \text{true}$.

SAT

SAT is a search problem

Example

For SAT, I is a Boolean formula, S is an assignment of Boolean constants to its variables. The corresponding algorithm C checks whether S satisfies all clauses of I .

$$I = (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_3})$$

$$S = x_1 = 1 \quad x_2 = 0 \quad x_3 = 0$$

$C(I, S)$ — checks if S satisfies all clauses of I .

CLASS NP

Definition

A **search problem** is defined by an algorithm \mathcal{C} that takes an instance I and a candidate solution S , and runs in time polynomial in the length of I . We say that S is a solution to I iff $\mathcal{C}(S, I) = \text{true}$.

CLASS NP

Definition

A **search problem** is defined by an algorithm \mathcal{C} that takes an instance I and a candidate solution S , and runs in time polynomial in the length of I . We say that S is a solution to I iff $\mathcal{C}(S, I) = \text{true}$.

Definition

NP is the class of all search problems.

- **NP** stands for “non-deterministic polynomial time”: one can guess a solution, and then verify its correctness in polynomial time

- **NP** stands for “non-deterministic polynomial time”: one can guess a solution, and then verify its correctness in polynomial time
- In other words, the class **NP** contains all problems whose solutions can be efficiently verified

CLASS P

$NP \equiv$ problems whose sols can be verified
in poly time

Definition

P is the class of all search problems that can be
solved in polynomial time.

$P \equiv$ problems that can be solved in poly time

Thm $P \subseteq NP$

Proof:

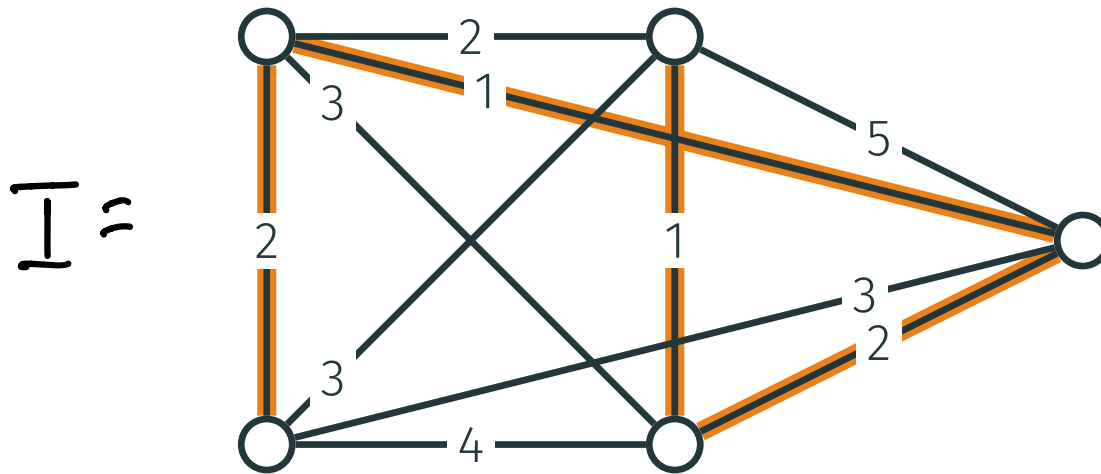
If one can solve a problem in poly time \Rightarrow
then one can verify a solution in poly time \square

TRAVELING SALESMAN PROBLEM

(Lecture 1)

(a cycle)

Given a complete weighted graph, find a path of minimum total weight (length) visiting each node exactly once

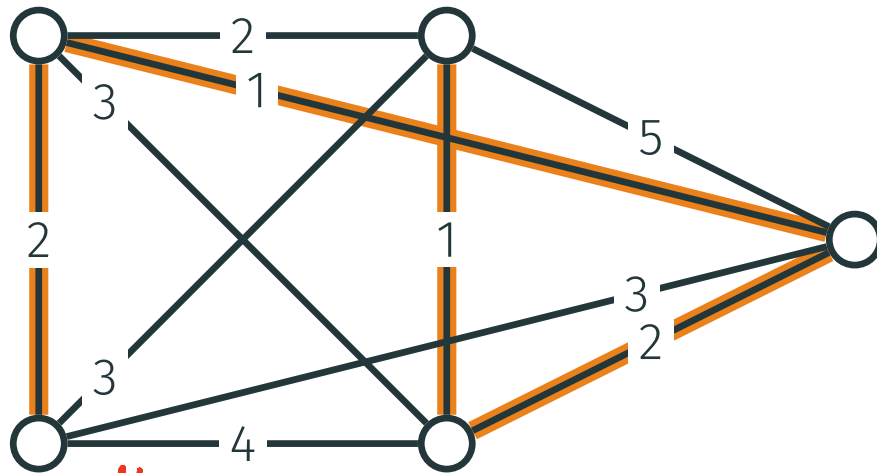


$S =$ some path
poly-time alg C takes I, S

length: 6

TRAVELING SALESMAN PROBLEM

Given a complete weighted graph and a budget b , find a path of total weight (length) $\leq b$ visiting each node exactly once



, b

$I =$

$S =$ some path
 $C(I, S) =$ S is a path that visits every vertex once
total weight of $\leq b$

length: $6 \leq b$

If I can solve TSP with budget b
in poly time \Rightarrow

I can also find minimum-weight
TSP path in poly time.

Binary search.

Proof: $0 \leq \text{every path} \leq M$
 $M = \sum \text{weights of all edges}$

TSP with budget $\frac{M}{2}$

If YES $\Rightarrow \text{min TSP} \in [0; \frac{M}{2}]$

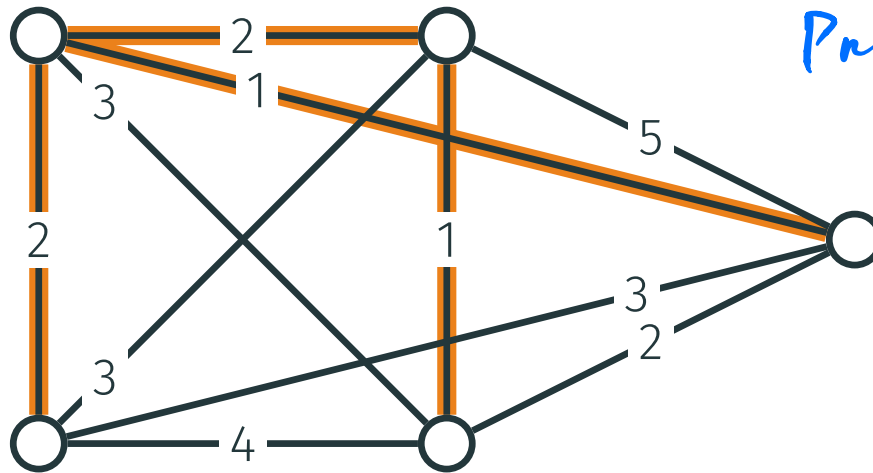
If NO $\Rightarrow \text{min TSP} \in [\frac{M}{2}; M]$.

$$O(\log_2 M) \cdot \text{poly}(n) =$$
$$= \text{poly}$$

□

MINIMUM SPANNING TREE (Lecture 2)

Given a complete weighted graph and a budget b , connect all vertices by $n - 1$ edges of minimum total weight (length)



poly time
Kruskal's
Prim's

length: 6

TSP AND MST

MST

Given n cities, connect them by $(n - 1)$ roads of minimal total length

TSP AND MST

MST $\in P$

Given n cities, connect them by $(n - 1)$ roads of minimal total length

Can be solved efficiently *poly time*

TSP AND MST

MST

Given n cities, connect them by $(n - 1)$ roads of minimal total length

Can be solved efficiently

TSP

Given n cities, connect them **in a path** of minimal total length

TSP AND MST

MST $\in P, NP$

Given n cities, connect them by $(n - 1)$ roads of minimal total length

Can be solved efficiently *poly-time*

TSP $\in NP$

Given n cities, connect them *in a path* of minimal total length

No polynomial algorithm known!

LONGEST PATH

Search problem
 $C(I, S)$
graph, s, t, b path
checks in poly time if S is
a simple path from s to t
of total length $\leq b$.

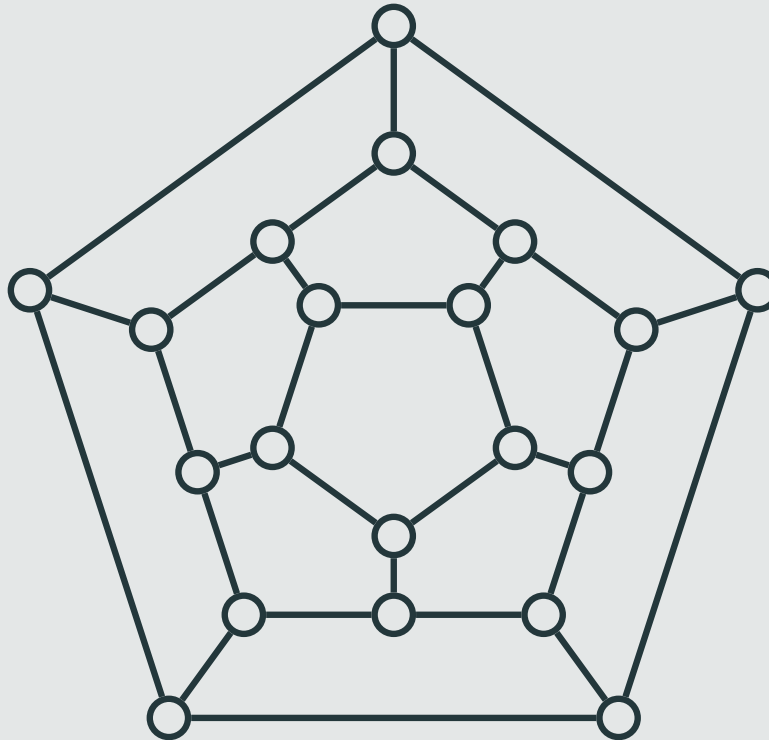
Longest path

Input: A weighted graph, two vertices s , t , and a budget b .

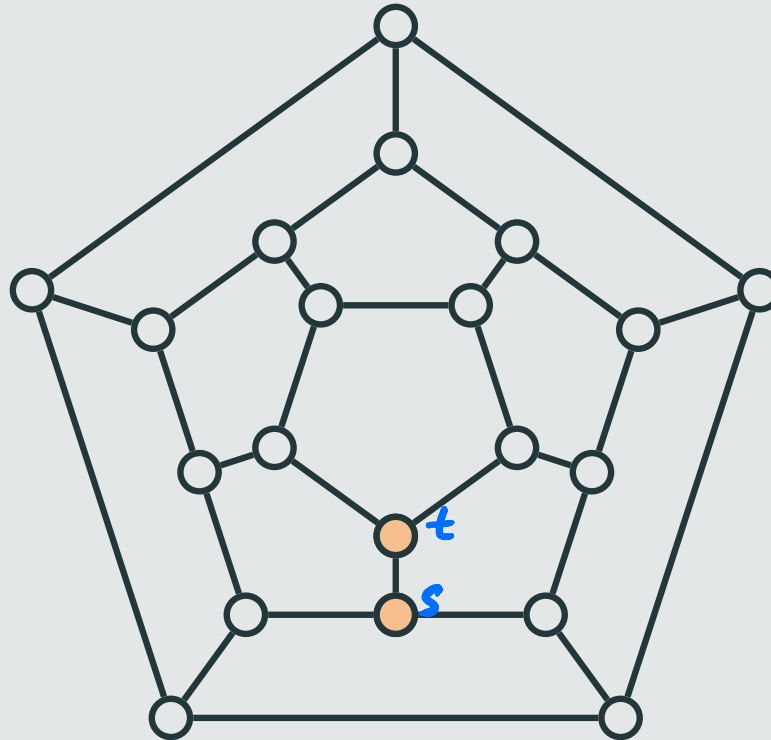
Output: A simple path (containing no repeated vertices) of total length at least b .

Longest path $\in NP$

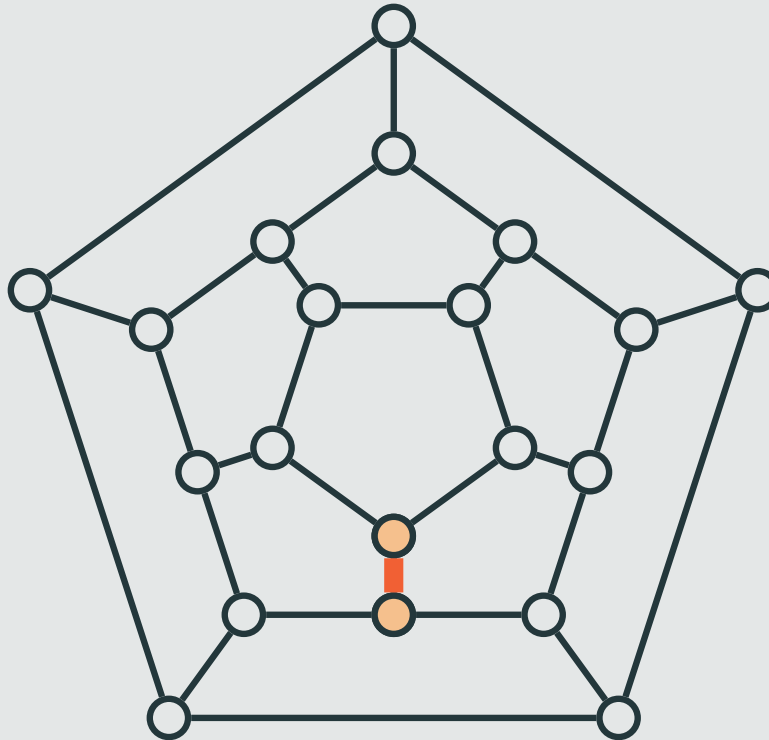
Example



Example

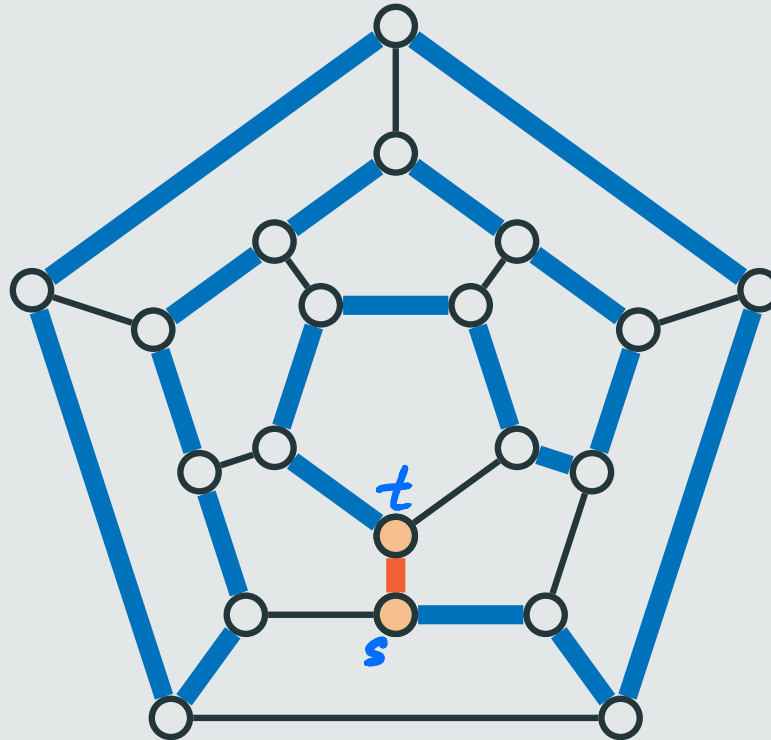


Example



Example

Longest Path Problem



Shortest path

Find a simple path from s to t of total length at

most b

BFS, Dijkstra,
Johnson's
Floyd's

Shortest path $\in P$

Find a simple path from
 s to t of total length at
most b

Can be solved
efficiently *poly-time*

Shortest path

Find a simple path from s to t of total length **at most** b

Can be solved efficiently

Longest path

Find a simple path from s to t of total length **at least** b

Shortest path $\in P, NP$

Find a simple path from s to t of total length at most b

Can be solved efficiently

Longest path $\in NP$

Find a simple path from s to t of total length at least b

No polynomial algorithm known!

INTEGER LINEAR PROGRAMMING PROBLEM

Integer linear programming (Lecture 12)

Input: A set of linear inequalities $Ax \leq b$.

Output: Integer solution.

Example

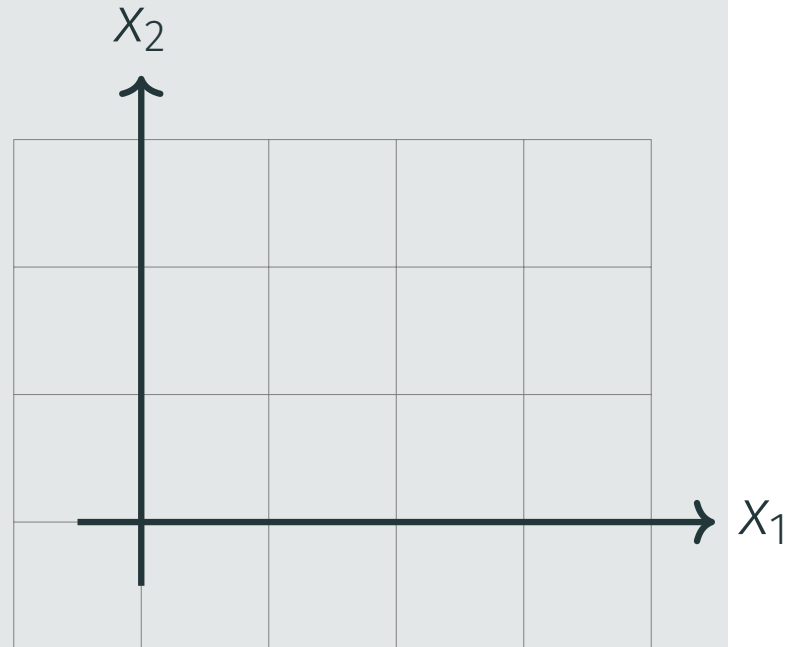
$$\begin{cases} x_1 \geq 0.5 \\ -x_1 + 8x_2 \geq 0 \\ -x_1 - 8x_2 \geq -8 \end{cases}$$

Example

$$x_1 \geq 0.5$$

$$-x_1 + 8x_2 \geq 0$$

$$-x_1 - 8x_2 \geq -8$$

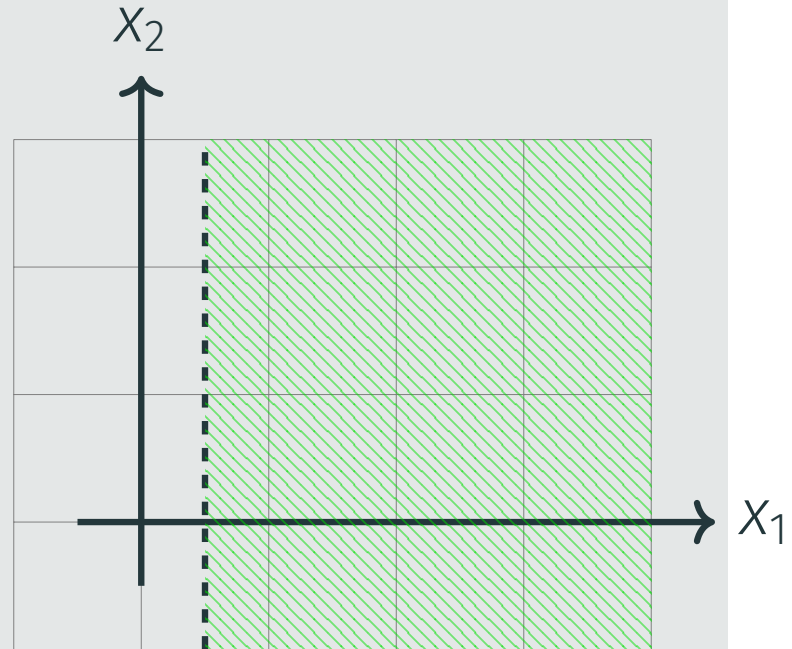


Example

$$x_1 \geq 0.5$$

$$-x_1 + 8x_2 \geq 0$$

$$-x_1 - 8x_2 \geq -8$$

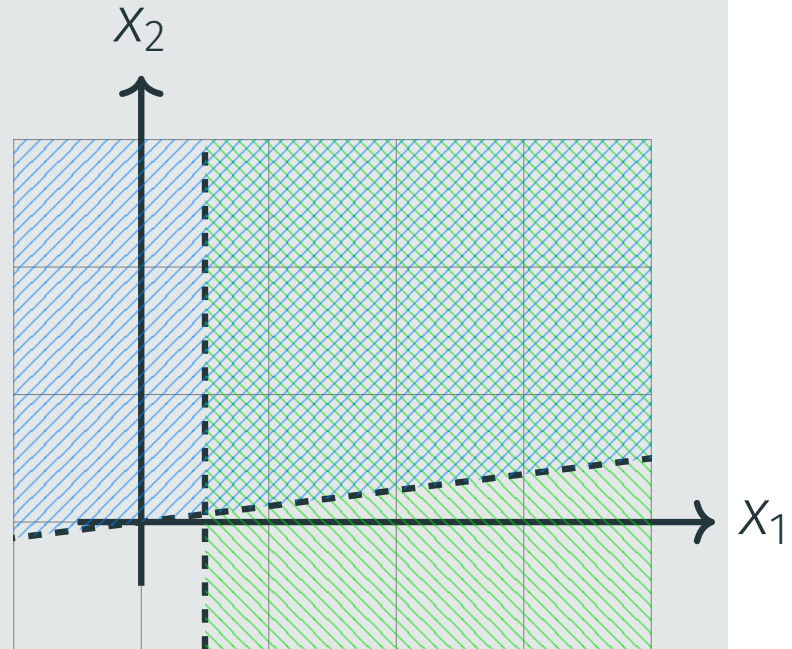


Example

$$x_1 \geq 0.5$$

$$-x_1 + 8x_2 \geq 0$$

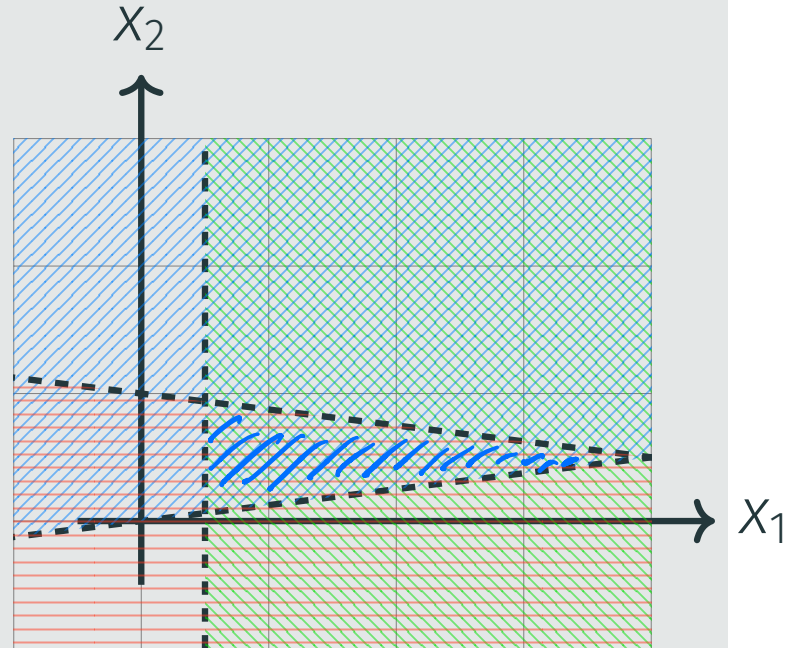
$$-x_1 - 8x_2 \geq -8$$



Example

Lecture 12

$$\begin{aligned}x_1 &\geq 0.5 \\ -x_1 + 8x_2 &\geq 0 \\ -x_1 - 8x_2 &\geq -8\end{aligned}$$



INTEGER LINEAR PROGRAMMING

LP

Find a **real**
solution of a system of
linear inequalities

INTEGER LINEAR PROGRAMMING

LP

Find a **real**
solution of a system of
linear inequalities

Can be solved
efficiently

INTEGER LINEAR PROGRAMMING

LP

Find a **real** solution of a system of linear inequalities

Can be solved efficiently

ILP

Find an **integer** solution of a system of linear inequalities

INTEGER LINEAR PROGRAMMING

Linear Programming (Lecture 9)

$LP \in P, NP$

Find a real solution of a system of linear inequalities

Can be solved efficiently *poly-time*

$ILP \in NP$

Find an integer solution of a system of linear inequalities

No polynomial algorithm known!

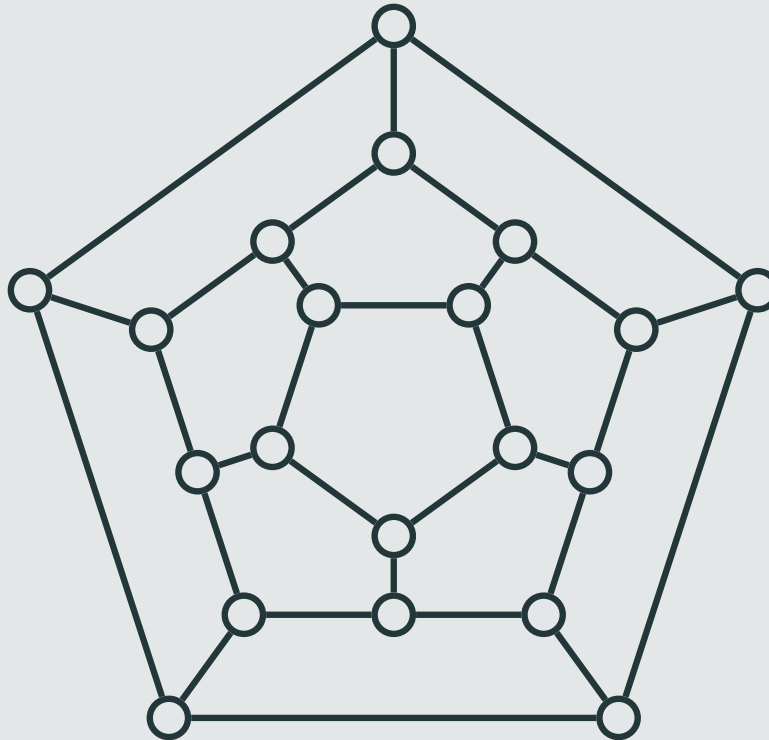
INDEPENDENT SET PROBLEM

Independent set

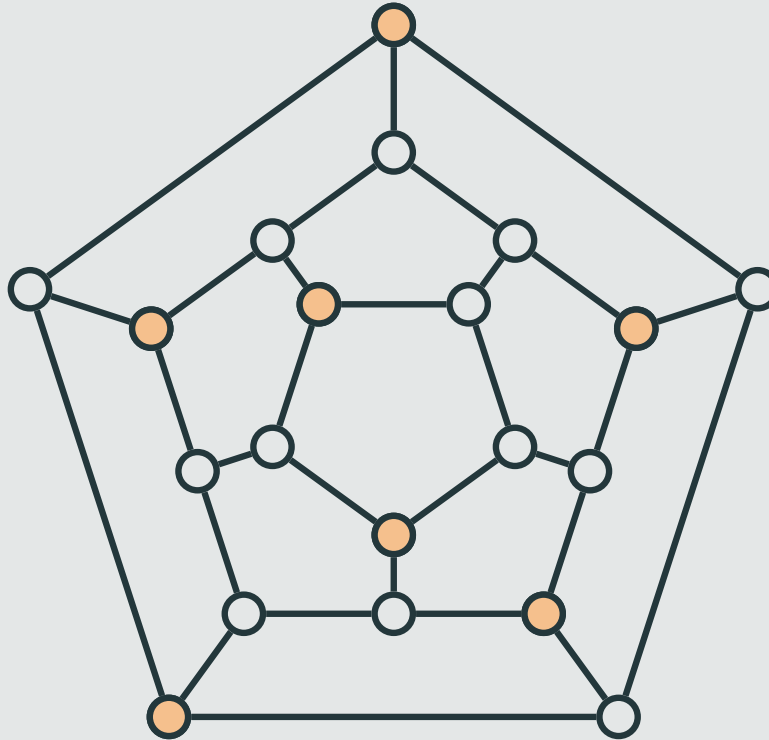
Input: A graph and a budget b .

Output: A subset of vertices of size at least b such that no two of them are adjacent.

Example

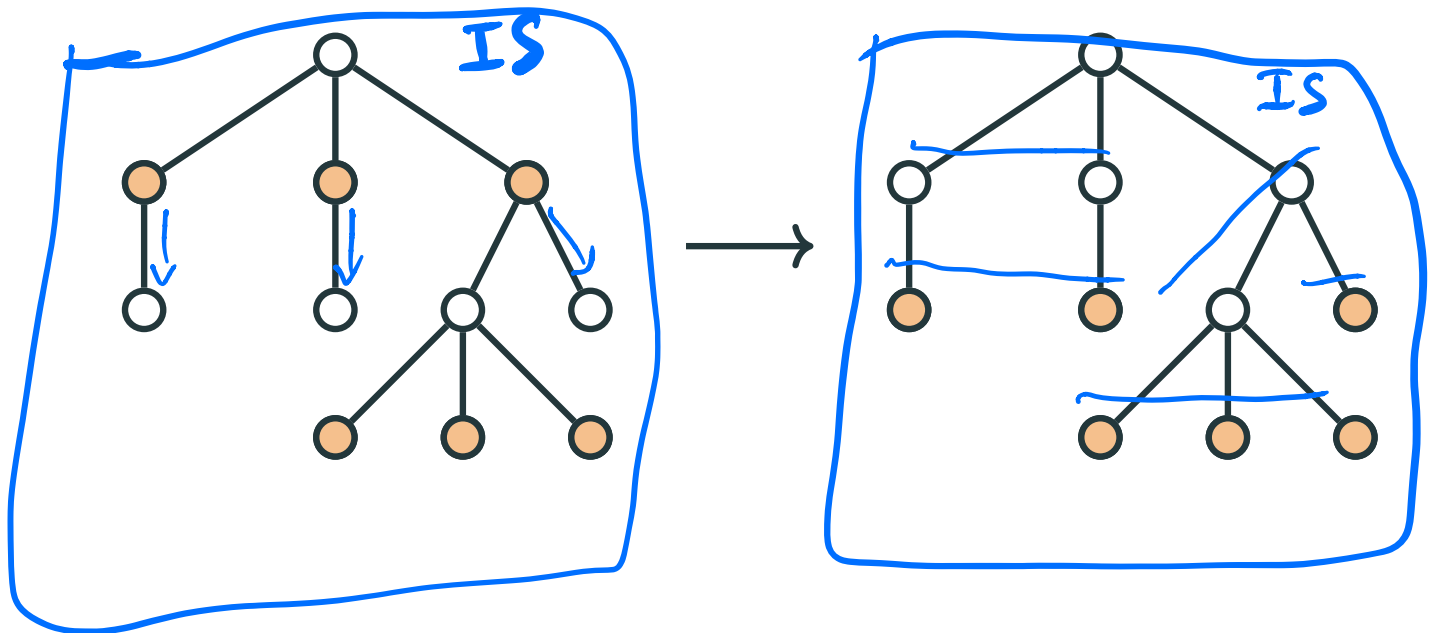


Example of an Independent Set $\in NP$



INDEPENDENT SETS IN A TREE

A maximum independent set in a tree can be found by a simple greedy algorithm: it is safe to take into a solution all the leaves.



Independent set in a tree

Find an independent
set of size at least b in
a given tree

Independent set in a tree

Find an independent
set of size at least b in
a given tree

Can be solved
efficiently

Independent set in a tree

Find an independent
set of size at least b in
a given tree

Can be solved
efficiently

Independent set in a graph

Find an independent
set of size at least b in
a given graph

Independent set in
a tree $\in P$

Find an independent
set of size at least b in
a given tree

Can be solved
efficiently *poly time*

Independent set in
a graph $\in NP$

Find an independent
set of size at least b in
a given graph

No polynomial
algorithm known!

NP

It turns out that all these hard problems are in a sense a single hard problem: a polynomial time algorithm for any of these problems can be used to solve all of them in polynomial time!

Class P

Problems whose
solution can be

found efficiently \equiv poly-time

Class P

Problems whose solution can be found efficiently

- MST
- Shortest path
- LP
- IS on trees

because we know
poly-time alg solve
these problems

Class P

Problems whose solution can be **found** efficiently

- MST
- Shortest path
- LP
- IS on trees

Class NP

Problems whose solution can be **verified** efficiently \equiv *poly-time*

Class P

Problems whose solution can be **found** efficiently

- MST
- Shortest path
- LP
- IS on trees

Class NP

Problems whose solution can be **verified** efficiently

- TSP
- Longest path
- ILP
- IS on graphs

- MST
- Shortest path
- LP
- IS on trees

because
verify
candidate
solution
in poly
time

The main open problem in Computer Science

Know: $P \subseteq NP$

Is P equal to NP ?

$P = NP?$

The main open problem in Computer Science

Is P equal to NP ?

Millennium Prize Problem

Clay Mathematics Institute: \$1M prize for solving the problem

- $P = NP$

- $P \neq NP$

- this question is independent of ZFC?

- If $\mathbf{P=NP}$, then all search problems can be solved in polynomial time.

- If $P=NP$, then all search problems can be solved in polynomial time.
- If $P \neq NP$, then there exist search problems that cannot be solved in polynomial time.

P = "verify a proof of a Thm"

NP = "find a proof of a Thm"


$P=NP \Rightarrow$ proving a theorem is essentially as easy as verifying a proof.

Reductions

(Lecture 7)

INFORMALLY

polys-time alg A
 \Leftarrow polys-time alg for B

We say that a search problem A is  reduced to a search problem B and write $A \rightarrow B$, if a polynomial time algorithm for B can be used (as a black box) to solve A in polynomial time.

REDUCTION: $A \rightarrow B$

instance I of A

REDUCTION: $A \rightarrow B$

instance I of A

Algorithm for A

Algorithm for B

REDUCTION: $A \rightarrow B$

instance I of A



Algorithm for A

Algorithm for B

REDUCTION: $A \rightarrow B$

*assume B can be solved
in poly time*

instance I of A



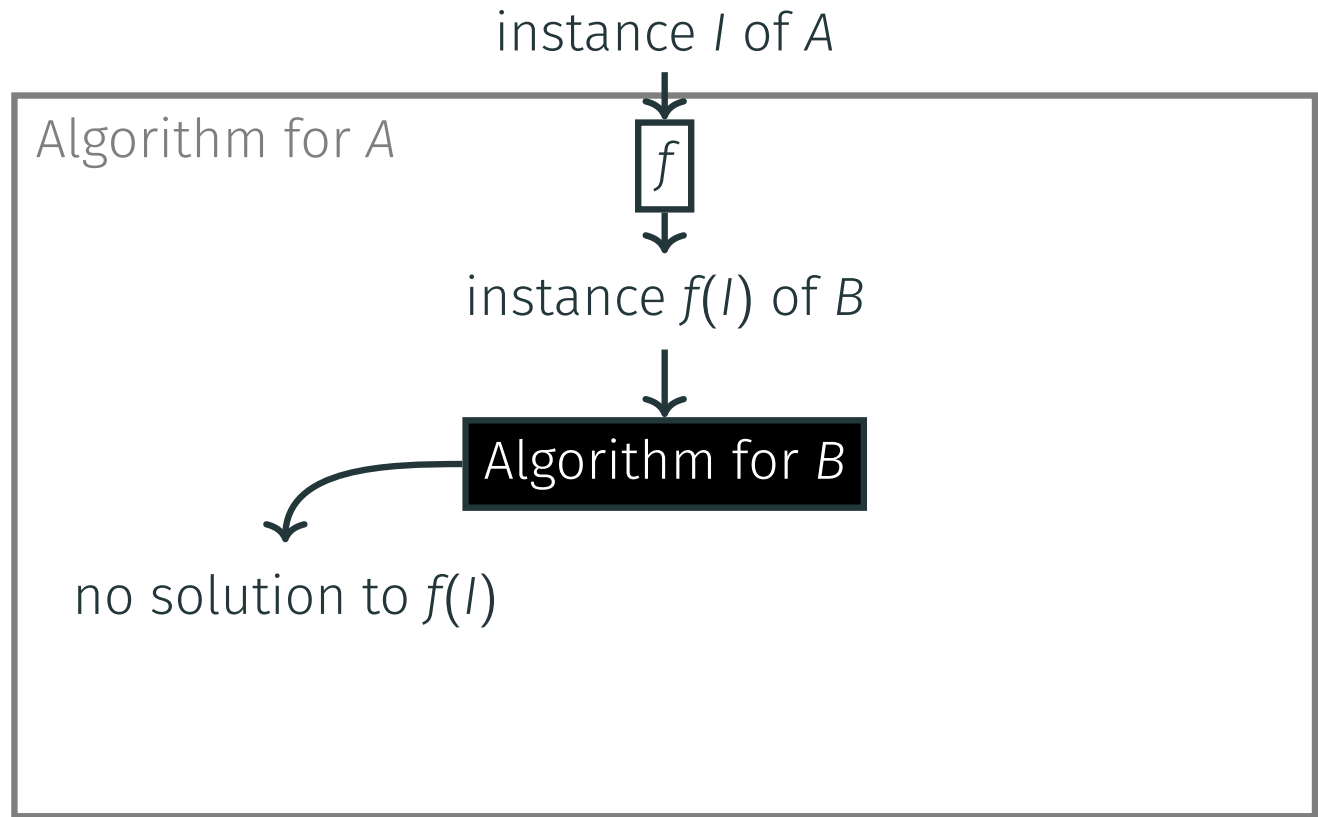
instance $f(I)$ of B



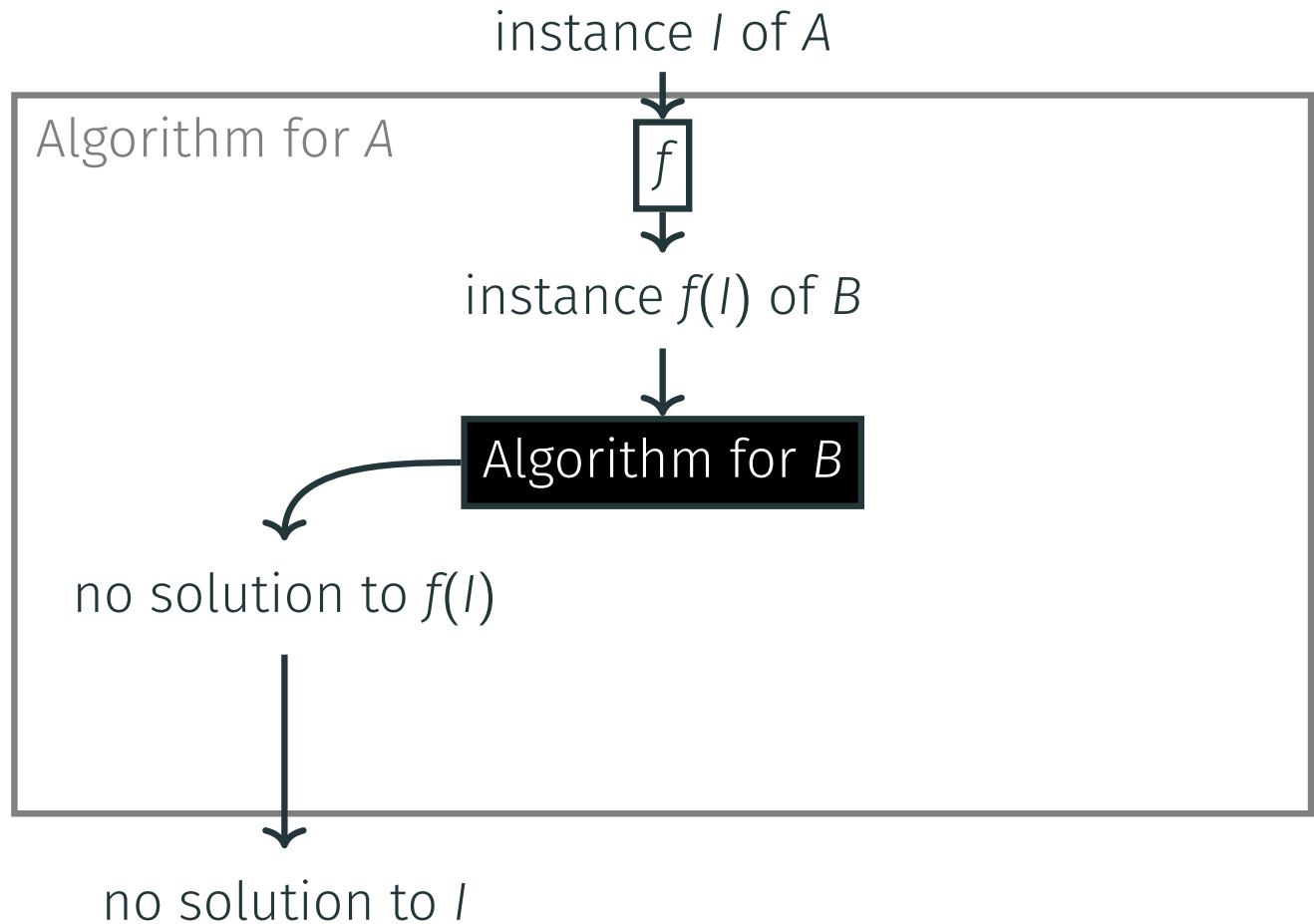
Algorithm for B

Algorithm for A

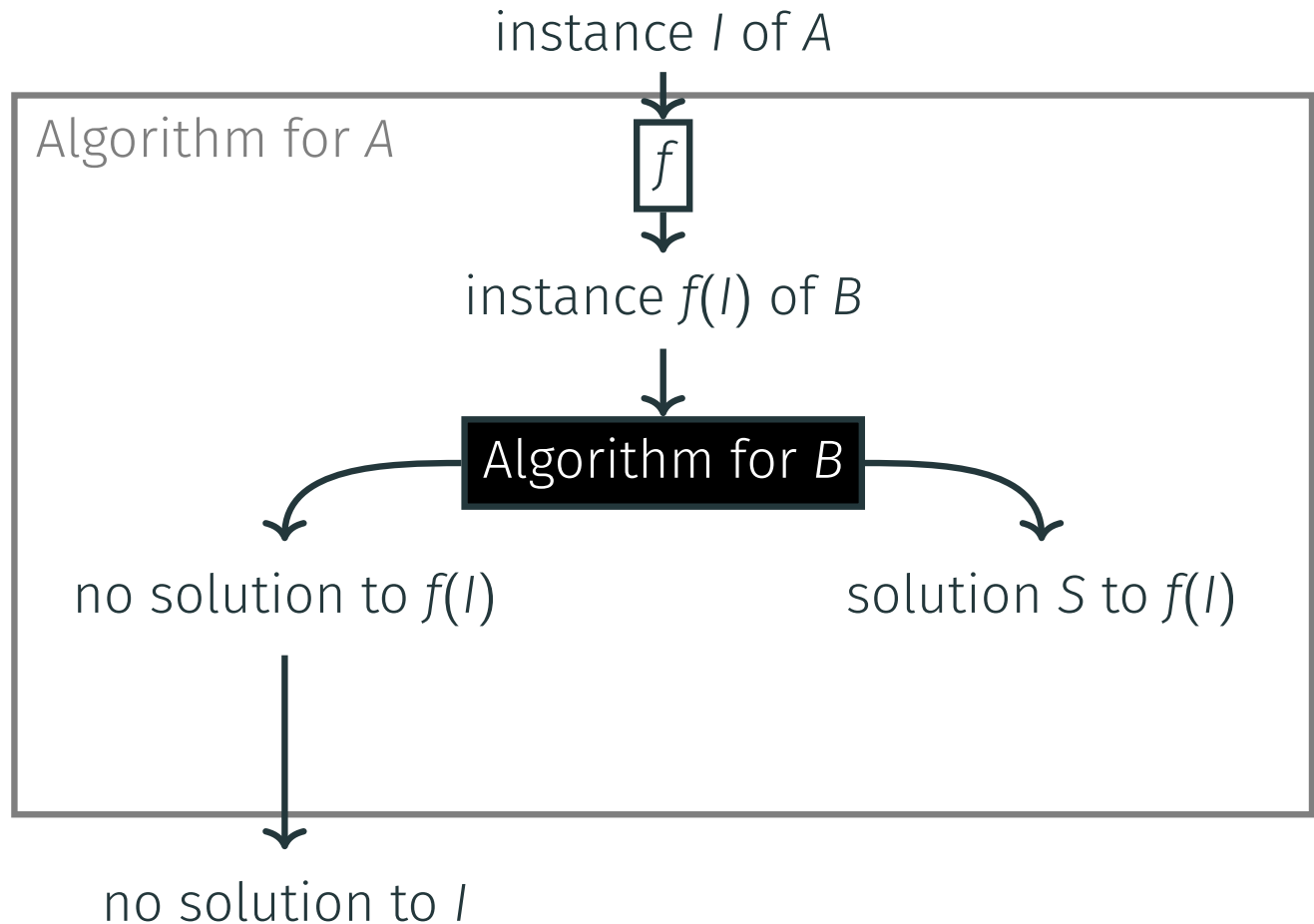
REDUCTION: $A \rightarrow B$



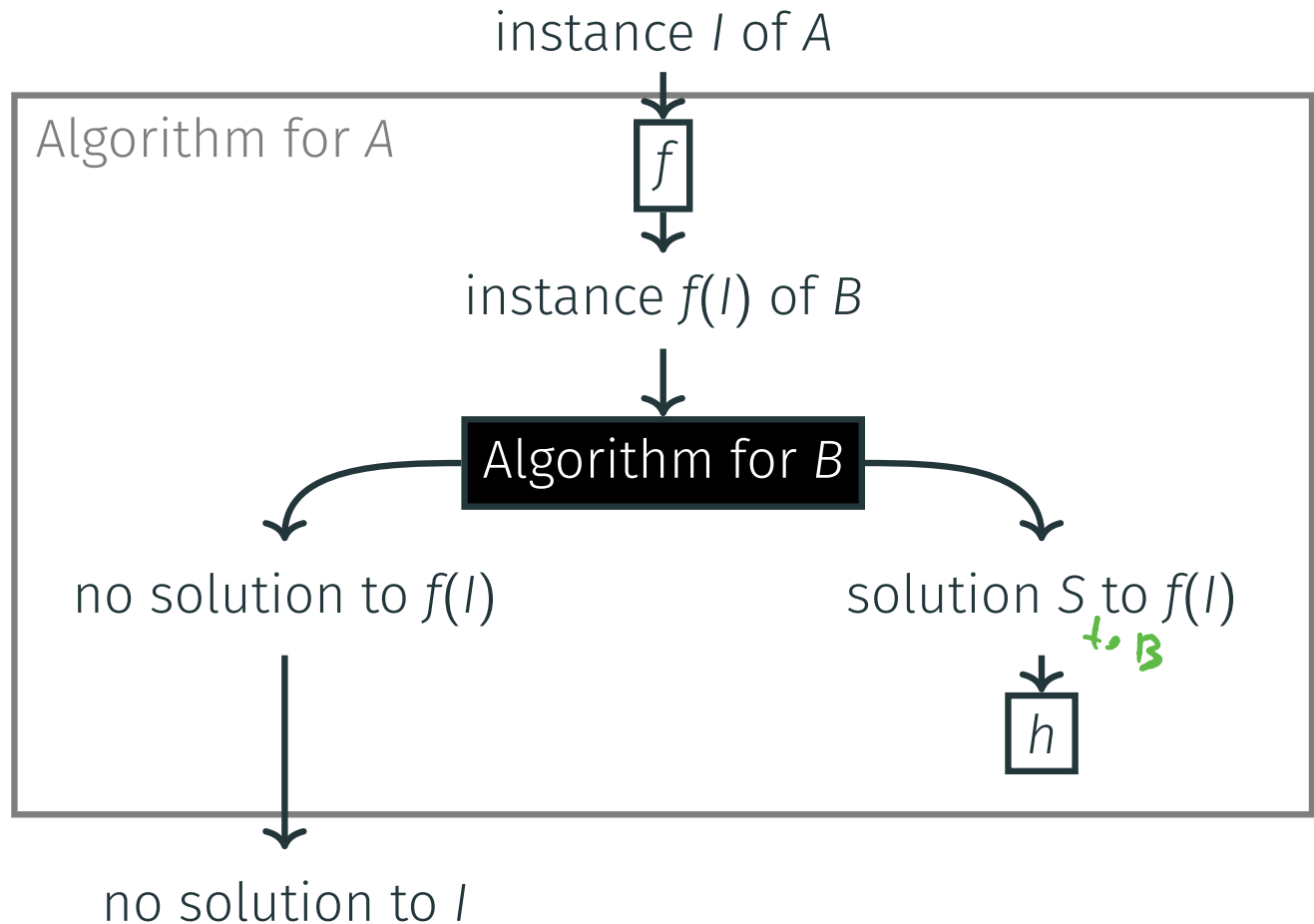
REDUCTION: $A \rightarrow B$



REDUCTION: $A \rightarrow B$



REDUCTION: $A \rightarrow B$

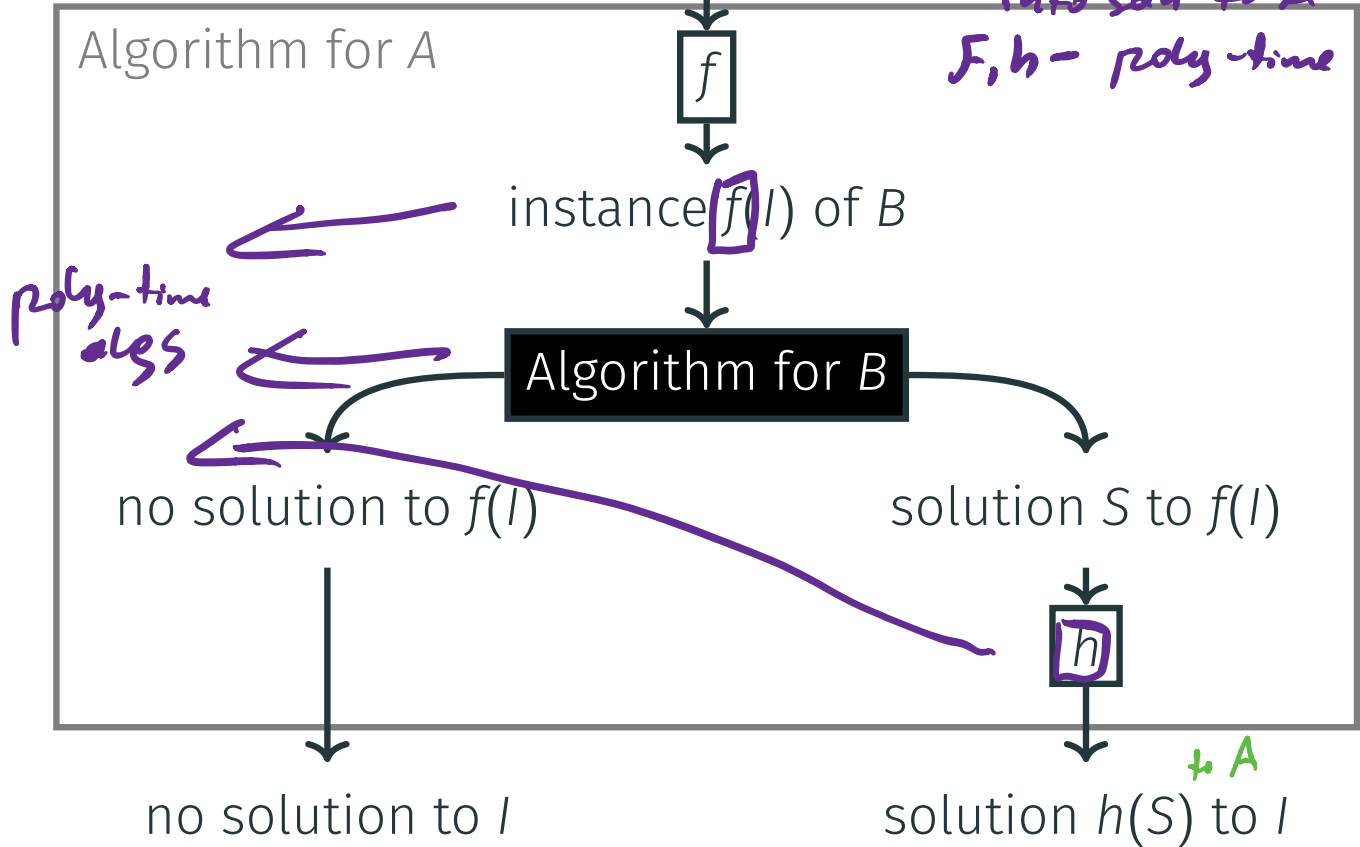


REDUCTION: $A \rightarrow B$

$\hat{=} (f, h)$ f transforms instance of A into instance of B

instance I of A h - soln to B into soln to A

f, h - poly-time algs

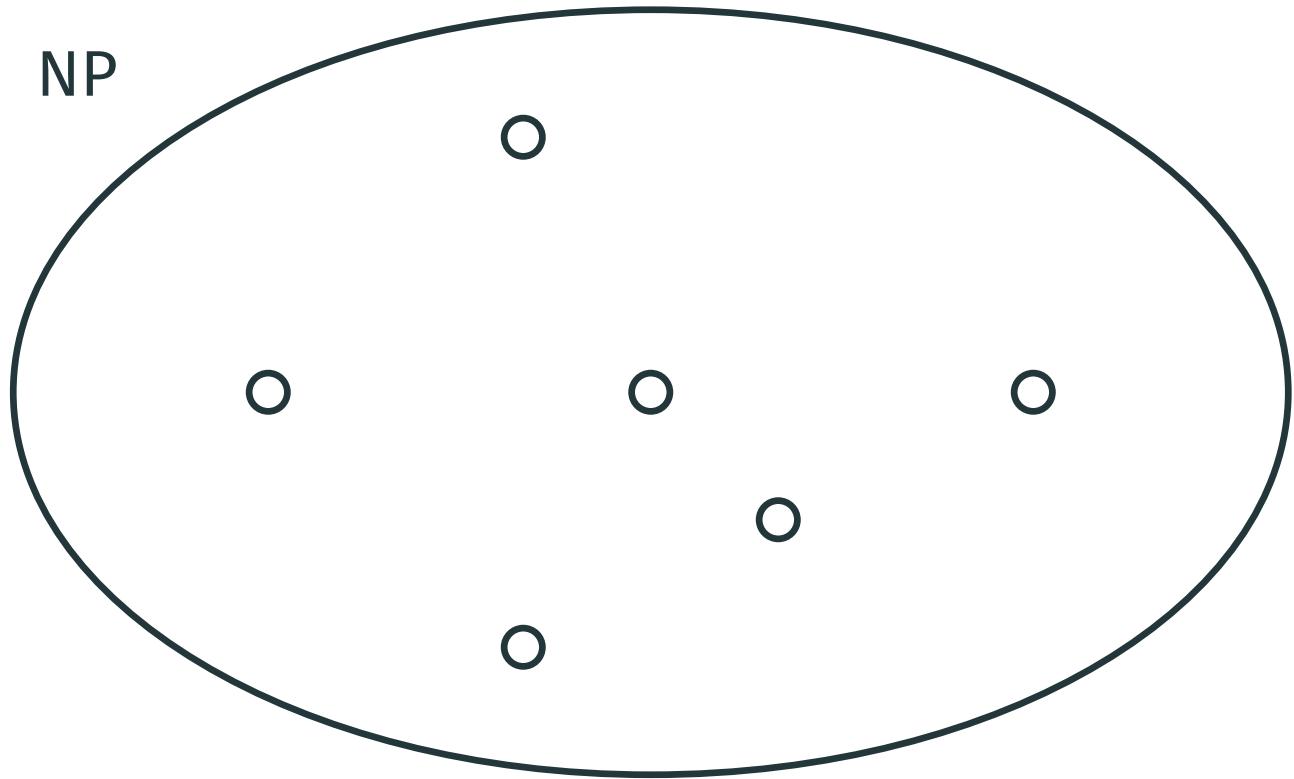


FORMALLY

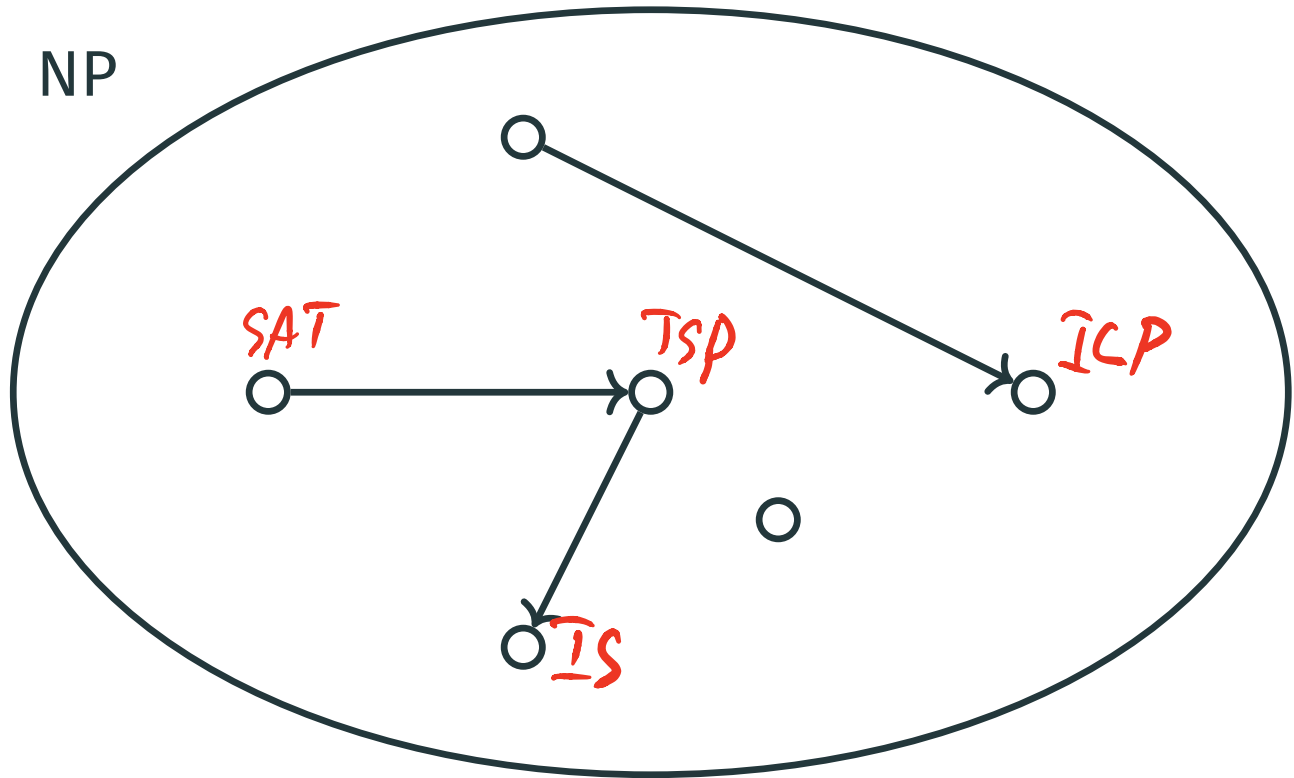
Definition

We say that a search problem A is reduced to a search problem B and write $A \rightarrow B$, if there exists a polynomial time algorithm f that converts any instance I of A into an instance $f(I)$ of B , together with a polynomial time algorithm h that converts any solution S to $f(I)$ back to a solution $h(S)$ to A . If there is no solution to $f(I)$, then there is no solution to I .

GRAPH OF SEARCH PROBLEMS



GRAPH OF SEARCH PROBLEMS



NP-COMPLETE PROBLEMS

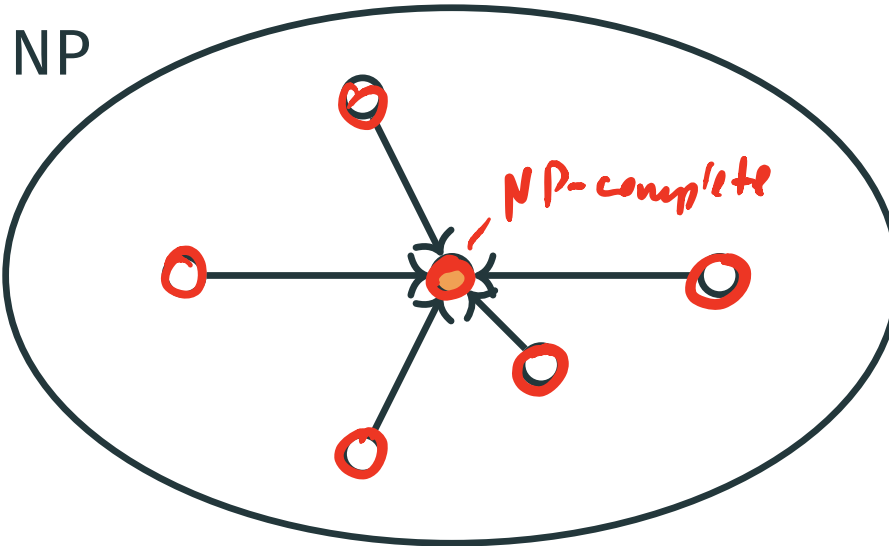
Definition

A search problem is called NP-complete if all other search problems reduce to it.

NP-COMPLETE PROBLEMS

Definition

A search problem is called **NP-complete** if all other search problems reduce to it.



Do they exist?

It's not at all immediate that **NP**-complete problems even exist. We'll see later that all hard problems that we've seen in the previous part are in fact **NP**-complete!

*ILP, IS, TSP, Longest Path are
NP-complete*

If B easy $\Rightarrow A$ easy
 A hard $\Rightarrow B$ hard

Two ways of using $A \rightarrow B$:

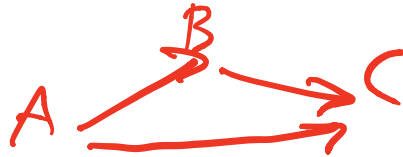
- if B is ^{poly-time} easy (can be solved in polynomial time), then so is A ^{is poly-time}
- if A is hard (^{cannot be solved in poly time} cannot be solved in polynomial time), then so is B ^{cannot be solved in poly-time}

$A \rightarrow B$ A is NP-complete $\Rightarrow B$ is NP-complete

REDUCTIONS COMPOSE

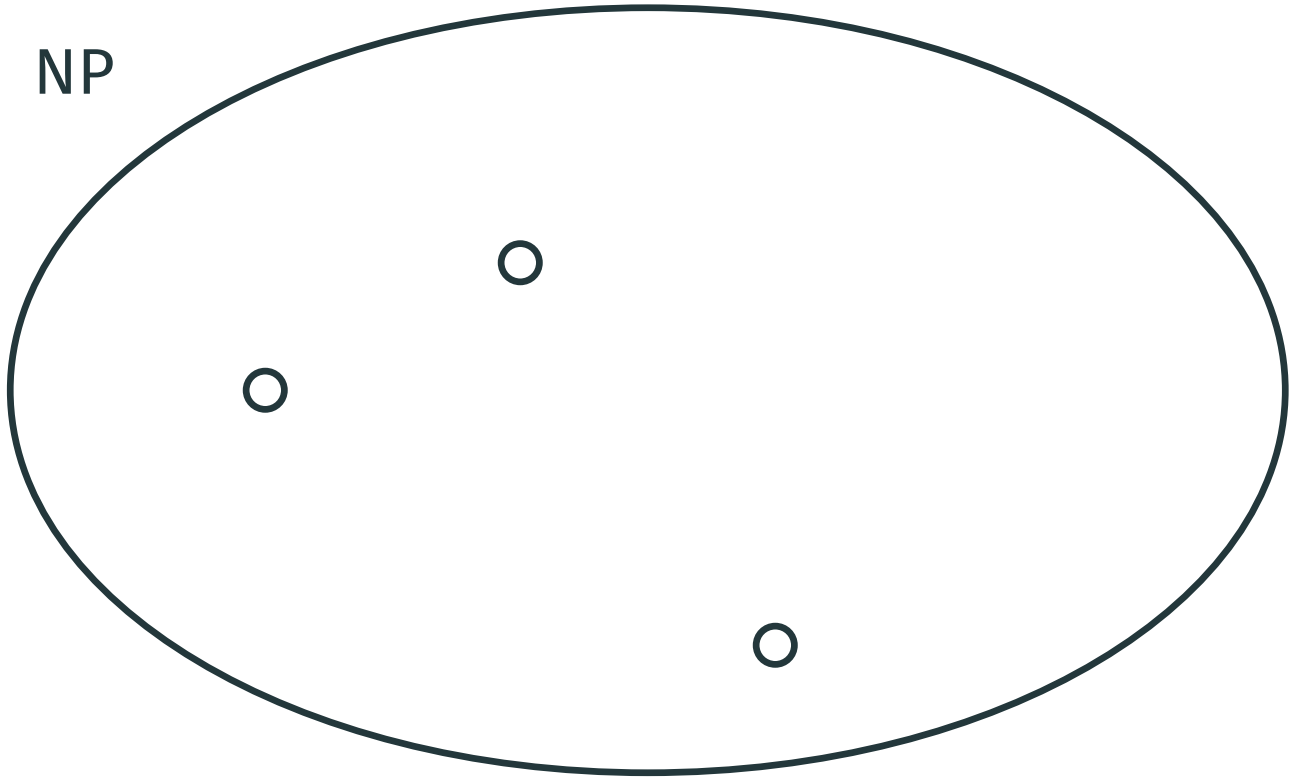
Lemma

If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$.



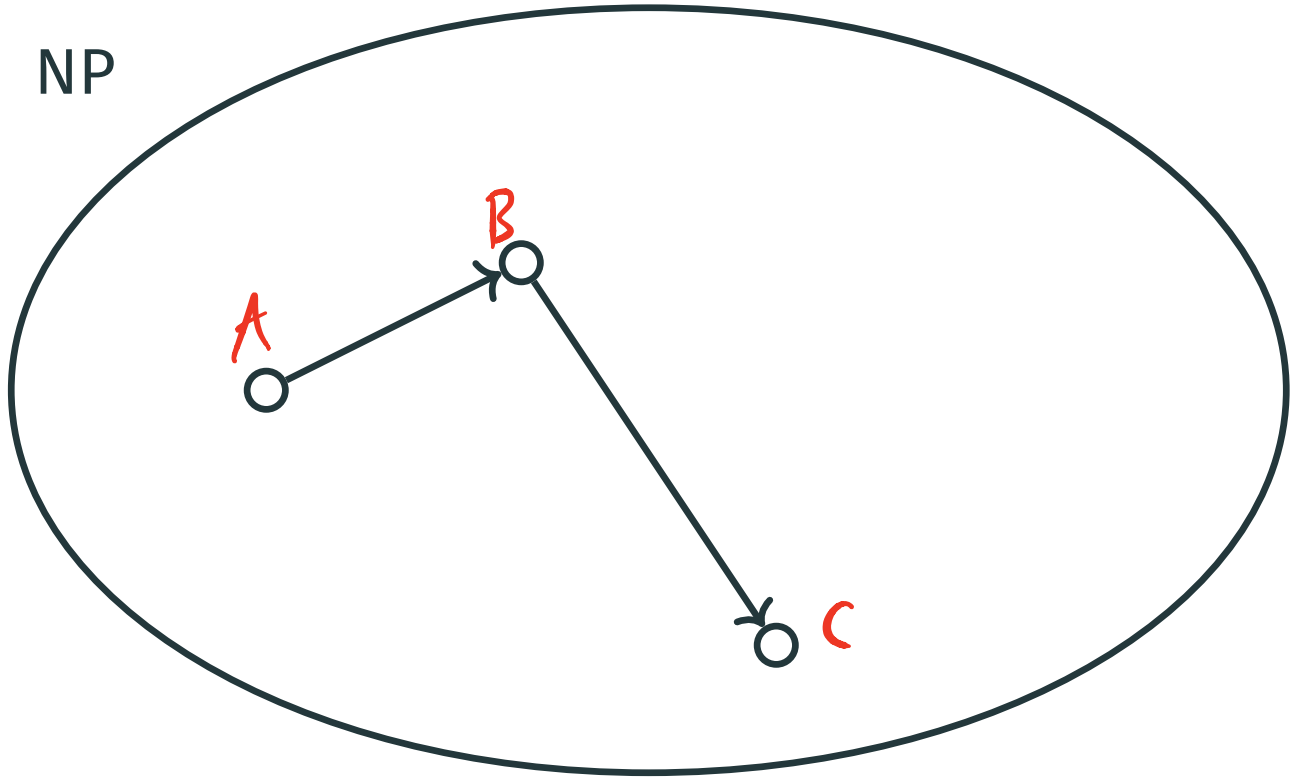
PICTORIALLY

NP



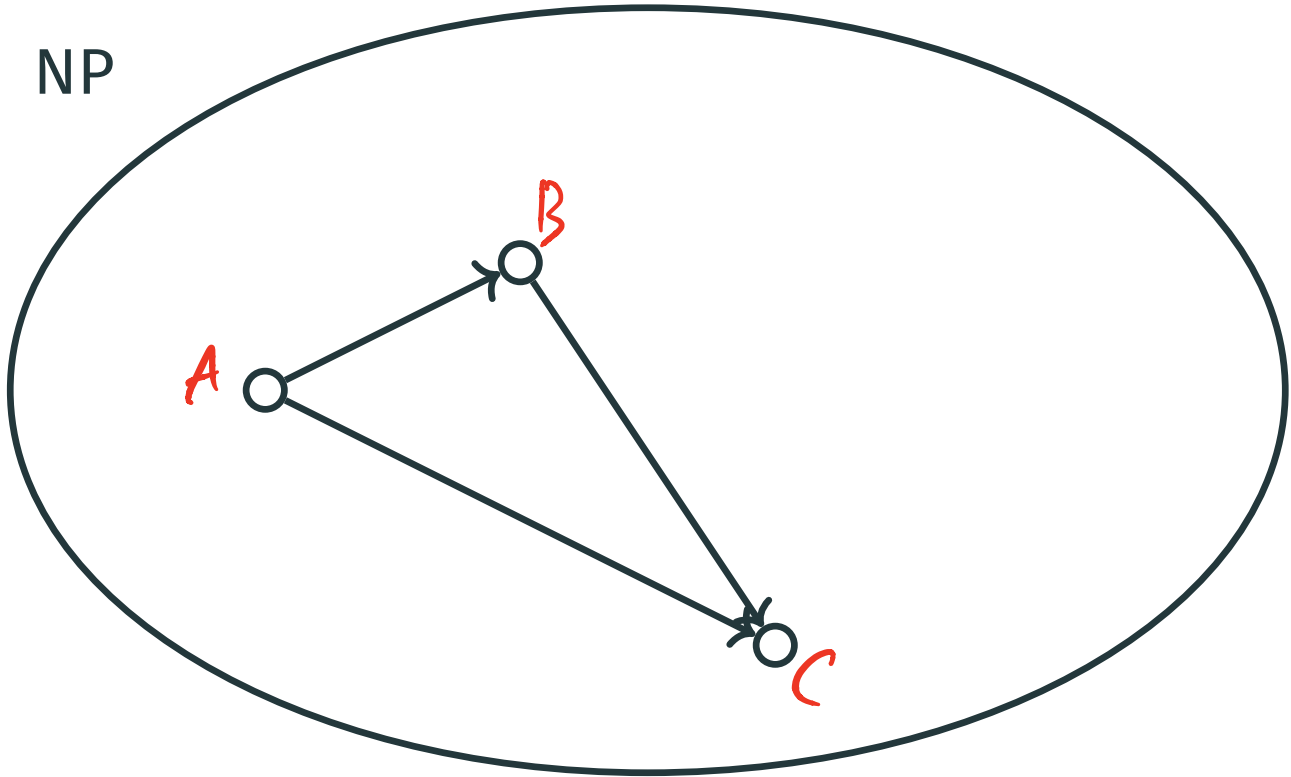
PICTORIALLY

NP



PICTORIALLY

NP



SHOWING NP-COMPLETENESS

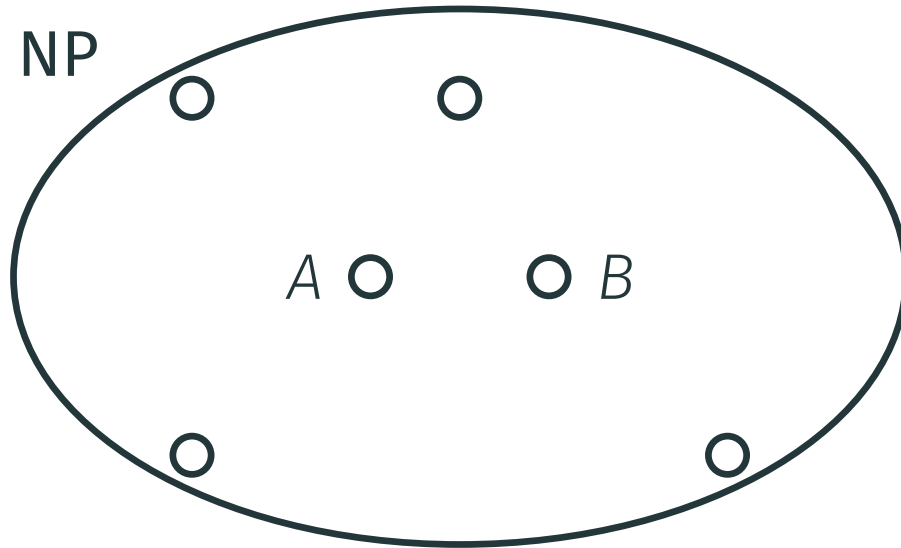
Corollary

If $A \rightarrow B$ and A is **NP**-complete, then so is B .

SHOWING NP-COMPLETENESS

Corollary

If $A \rightarrow B$ and A is **NP**-complete, then so is B .

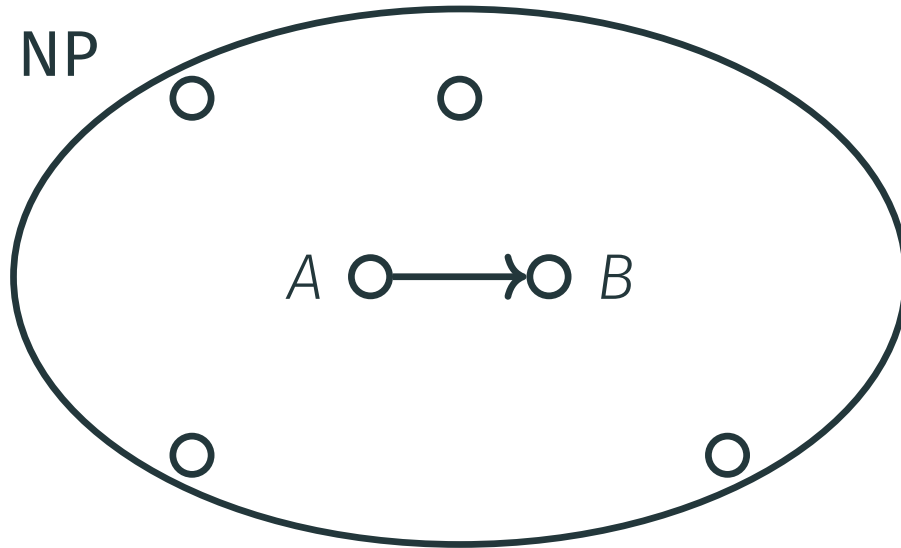


SHOWING NP-COMPLETENESS

Corollary

If $A \rightarrow B$ and A is **NP**-complete, then so is B .

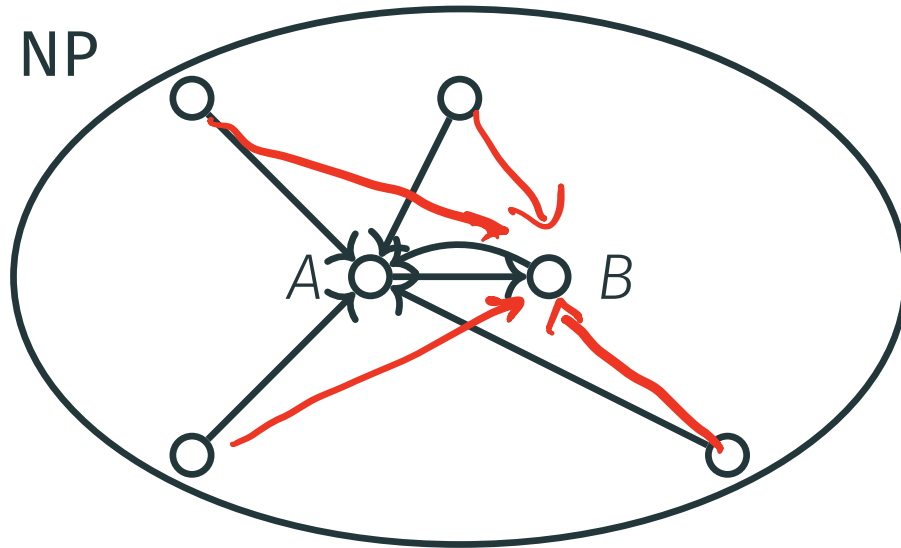
Proof:



SHOWING NP-COMPLETENESS

Corollary

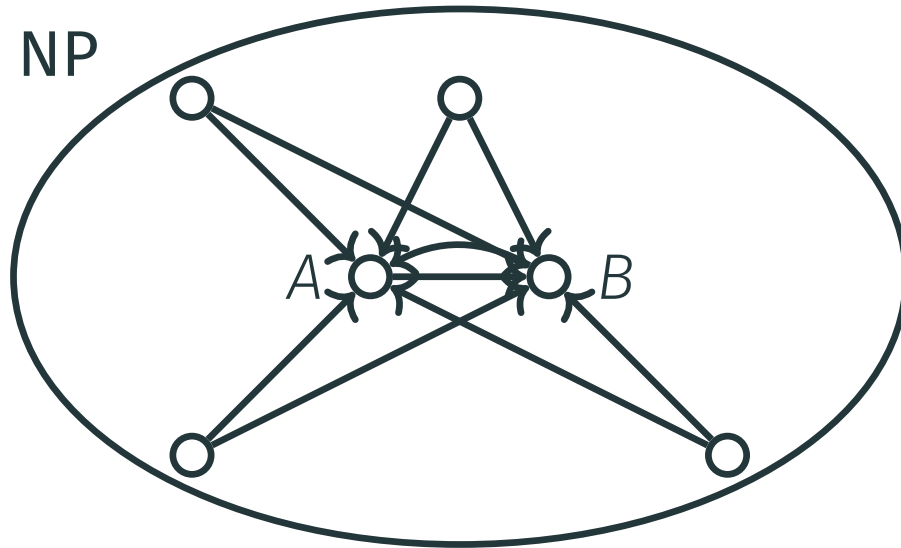
If $A \rightarrow B$ and A is **NP**-complete, then so is B .



SHOWING NP-COMPLETENESS

Corollary

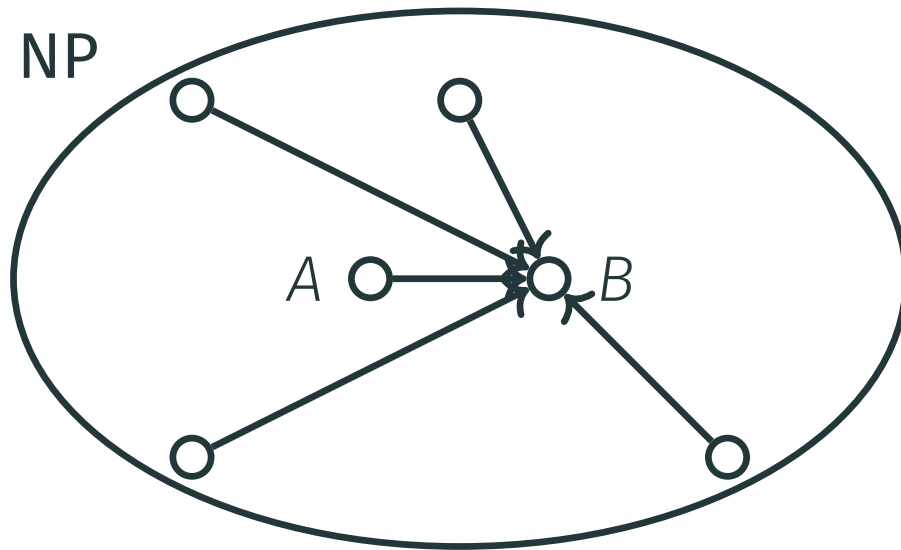
If $A \rightarrow B$ and A is **NP**-complete, then so is B .



SHOWING NP-COMPLETENESS

Corollary

If $A \rightarrow B$ and A is **NP**-complete, then so is B .



NP-Completeness of SAT

Goal

Show that *every* ^{NP} search problem reduces to SAT.

Goal

Show that **every** search problem reduces to SAT.

Instead, we show that any problem reduces to Circuit SAT problem, which, in turn, reduces to SAT.

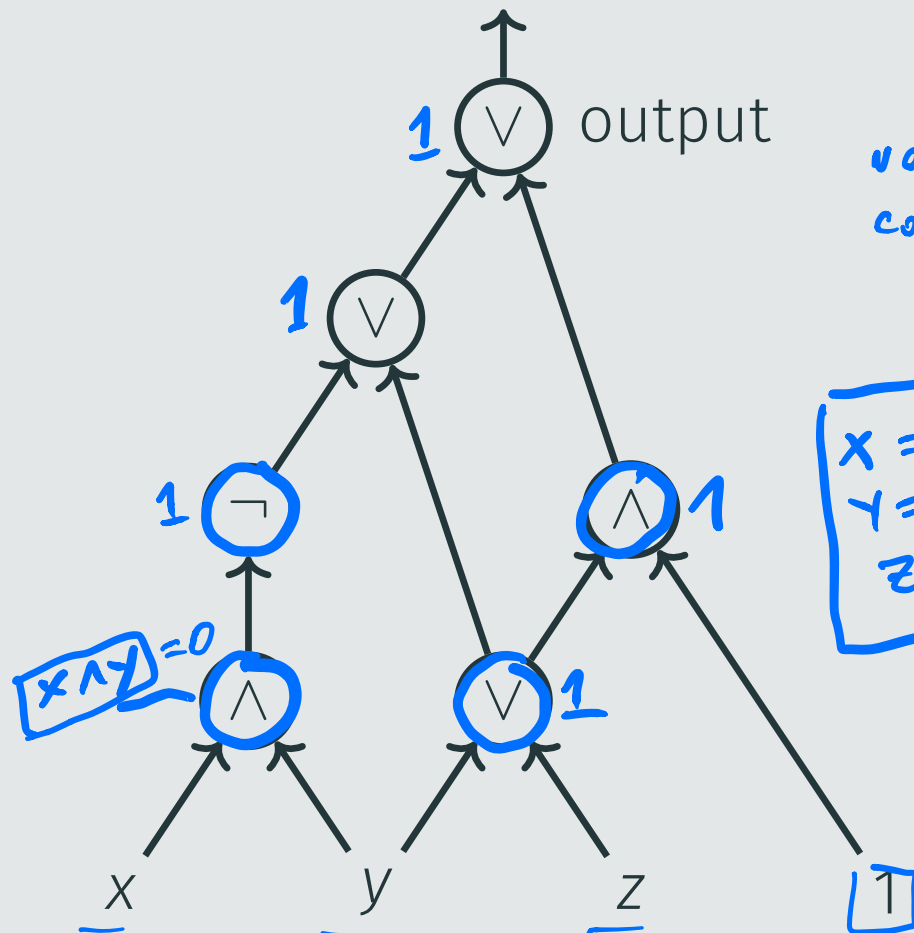
(i) Circuit SAT \rightarrow SAT

(ii) All NP problems \rightarrow Circuit SAT

(ii) \Rightarrow Circuit SAT is NP-complete

(i) \Rightarrow SAT is NP-complete

Circuit

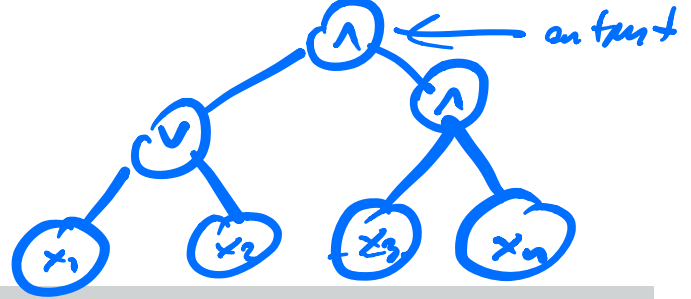


variables $x, y, z \dots$
constants $0, 1$

$$\begin{array}{l} x = 0 \\ y = 1 \\ z = 0 \end{array}$$

Definition

A **circuit** is a directed acyclic graph of in-degree at most 2. Nodes of in-degree 0 are called **inputs** and are marked by Boolean variables and constants. Nodes of in-degree 1 and 2 are called **gates**: gates of in-degree 1 are labeled with **NOT**, gates of in-degree 2 are labeled with **AND** or **OR**. One of the sinks is marked as **output**.



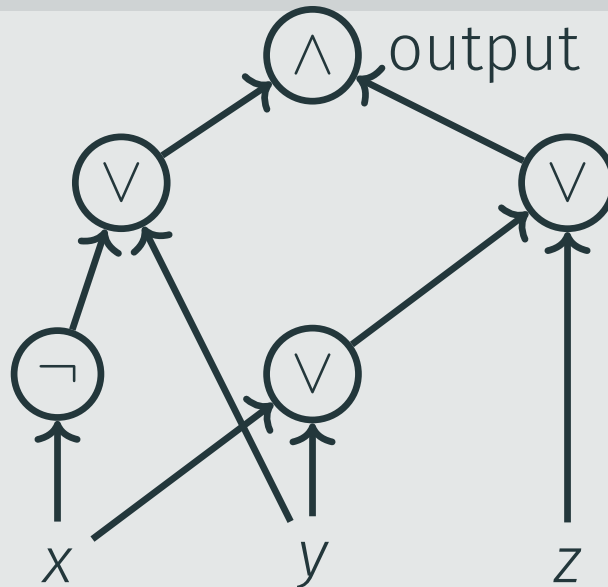
Circuit-SAT

Input: A circuit.

Output: An assignment of Boolean values to the input variables of the circuit that makes the output true.

SAT is a special case of Circuit-SAT as a SAT formula can be represented as a circuit:

Example: $(x \vee y \vee z)(y \vee \bar{x})$



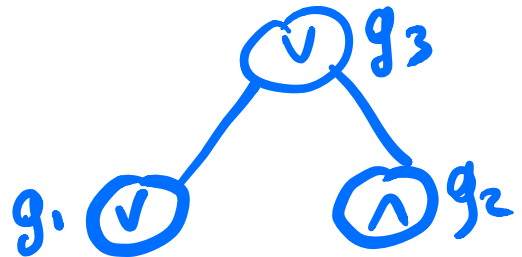
$$\text{CIRCUIT-SAT} \rightarrow \text{SAT}$$

(i) Circuit-SAT \Rightarrow SAT

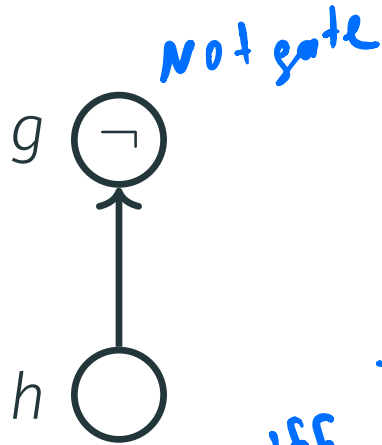
To reduce Circuit-SAT to SAT, we need to design a polynomial time algorithm that for a given circuit outputs a SAT formula which is satisfiable, if and only if the circuit is satisfiable

IDEA

- Introduce a Boolean variable for each gate
- For each gate, write down a few clauses that describe the relationship between this gate and its direct predecessors



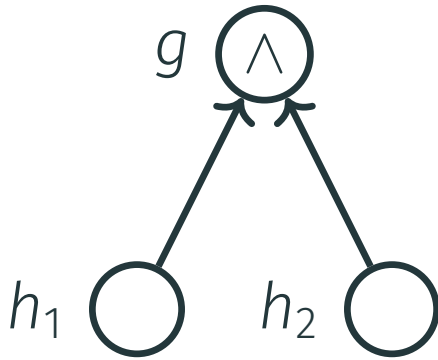
NOT GATES



$(h \vee g)(\bar{h} \vee \bar{g})$ — SAT fla
is SAT

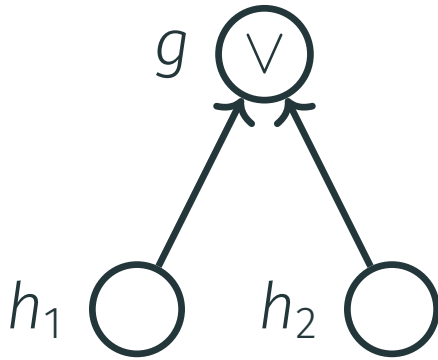
iff $g = \text{neg}(h)$

AND GATES



$$(h_1 \vee \bar{g})(h_2 \vee \bar{g})(\bar{h}_1 \vee \bar{h}_2 \vee g)$$

OR GATES



$$\underline{(\bar{h}_1 \vee g)(\bar{h}_2 \vee g)(h_1 \vee h_2 \vee \bar{g})}$$

OUTPUT GATE

$$g \bigcirc \text{output} \quad (g)$$

- The resulting SAT formula is consistent with the initial circuit: in any satisfying assignment of the formula, the value of g is equal to the value of the gate labeled with g in the circuit

- The resulting SAT formula is consistent with the initial circuit: in any satisfying assignment of the formula, the value of g is equal to the value of the gate labeled with g in the circuit
- Therefore, the SAT formula and the circuit are equisatisfiable

- The resulting SAT formula is consistent with the initial circuit: in any satisfying assignment of the formula, the value of g is equal to the value of the gate labeled with g in the circuit
- Therefore, the SAT formula and the circuit are equisatisfiable
- The reduction takes polynomial time

(:) Circuit SAT \rightarrow SAT

It remains to show : every NP-problem
 \rightarrow Circuit-SAT

Goal

Reduce every search problem to Circuit-SAT.

Goal

Reduce every search problem to Circuit-SAT.

- Let A be a search problem

Goal

Reduce every search problem to Circuit-SAT.

$\in \text{NP}$

- Let A be a search problem
- We know that there exists an algorithm \mathcal{C} that takes an instance I of A and a candidate solution S and checks whether S is a solution for I in time polynomial in $|I|$

Goal

Reduce every search problem to Circuit-SAT.

- Let A be a search problem
- We know that there exists an algorithm \mathcal{C} that takes an instance I of A and a candidate solution S and checks whether S is a solution for I in time polynomial in $|I|$
- In particular, $|S|$ is polynomial in $|I|$

TURN AN ALGORITHM INTO A CIRCUIT

- Note that a computer is in fact a circuit implemented on a chip

TURN AN ALGORITHM INTO A CIRCUIT

- Note that a computer is in fact a circuit implemented on a chip
- Each step of the algorithm $\mathcal{C}(I, S)$ is performed by this computer's circuit



TURN AN ALGORITHM INTO A CIRCUIT

- Note that a computer is in fact a circuit implemented on a chip
- Each step of the algorithm $C(I, S)$ is performed by this computer's circuit
- This gives a circuit of size polynomial in $|I|$ that has input bits for I and S and outputs whether S is a solution for I (a separate circuit for each input length)

solve NP problem defined by C
by just the corresponding Circuit-SAT

REDUCTION

To solve an instance I of the problem A :

- take a circuit corresponding to $\mathcal{C}(I, \cdot)$

REDUCTION

To solve an instance I of the problem A :

- take a circuit corresponding to $\mathcal{C}(I, \cdot)$
- the inputs to this circuit encode candidate solutions

REDUCTION

To solve an instance I of the problem A :

- take a circuit corresponding to $C(I, \cdot)$
- the inputs to this circuit encode candidate solutions
- use a Circuit-SAT algorithm for this circuit to find a solution (if exists)

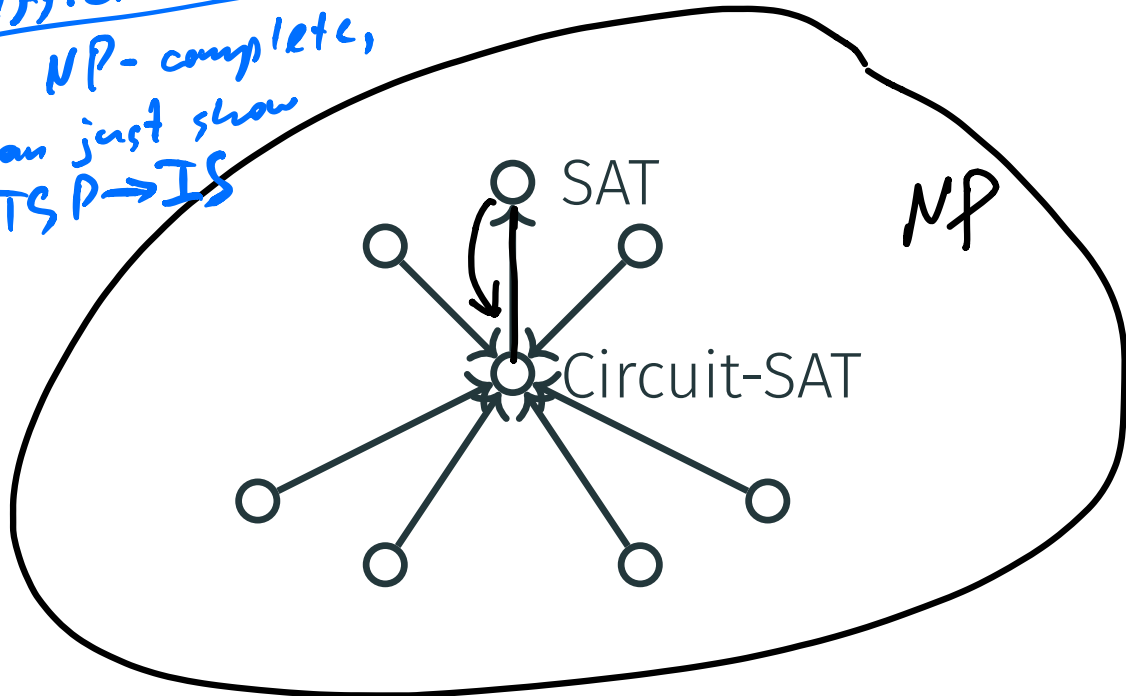
Every NP-problem reduces to Circuit-SAT \Rightarrow
Circuit-SAT is "hardest NP-problem", that is,
NP-complete problem

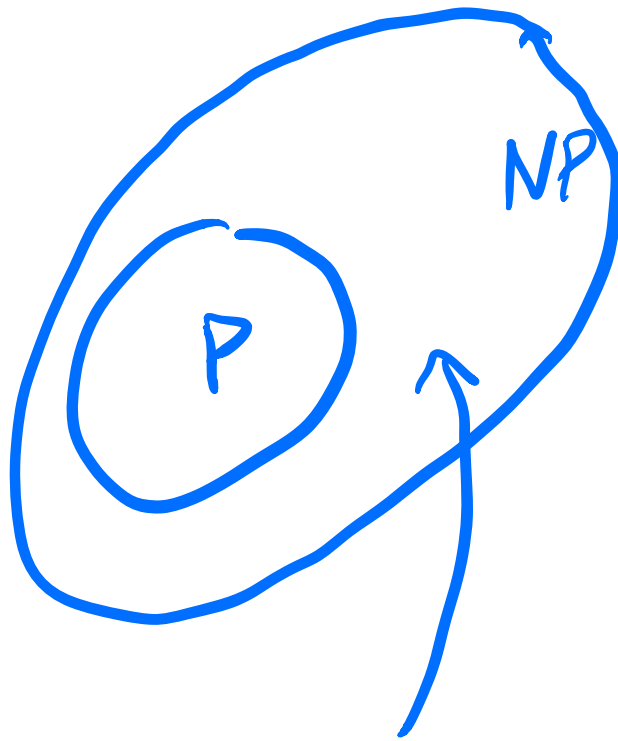
SUMMARY

Circuit-SAT is NP-complete
SAT is NP-complete

You want to prove TSP is NP-complete,
it's sufficient $SAT \rightarrow TSP$

IS is NP-complete,
You can just show
that $TSP \rightarrow IS$





NP-intermediate problem

\exists exist NP-intermediate problems
 $\Leftrightarrow P \neq NP$