Gems of TCS

P vs NP

Sasha Golovnev

October 16, 2023

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 $\mathcal{C}(S, I) = \texttt{true}$.

SAT

Example

For SAT, I is a Boolean formula, S is an assignment of Boolean constants to its variables. The corresponding algorithm $\mathcal{C}$ checks whether 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 $\mathcal{C}(S, I) = \texttt{true}$.

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 $\mathcal{C}(S, I) = \texttt{true}$.

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

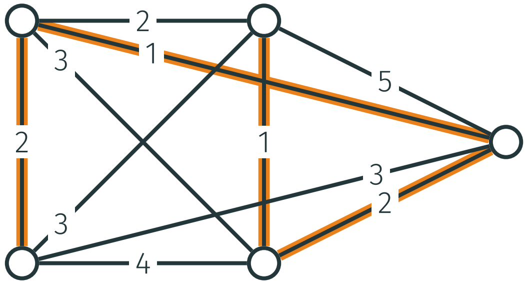
Class P

Definition

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

Traveling Salesman Problem

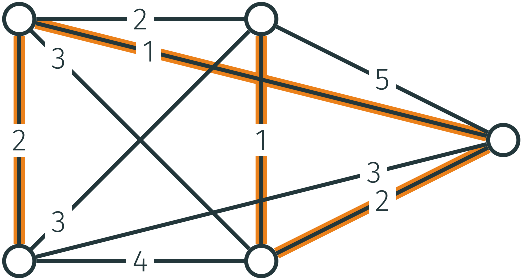
Given a complete weighted graph, find a path of minimum total weight (length) visiting each node exactly once



length: 6

Traveling Salesman Problem

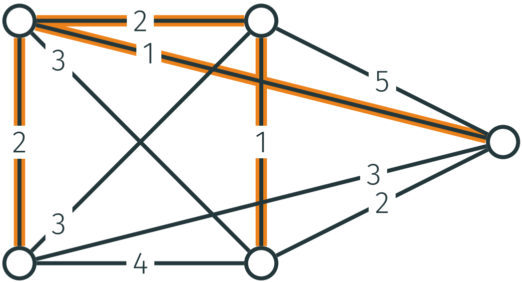
Given a complete weighted graph and a budget b, find a path of total weight (length) $\leq b$ visiting each node exactly once



length: $6 \leq b$

Minimum Spanning Tree

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



length: 6

TSP and MST

MST

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

TSP and MST

MST

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

Can be solved efficiently

TSP and MST

MST

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

TSP

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

Can be solved efficiently

TSP and MST

MST	TSP
Given n cities, connect them by $(n-1)$ roads of minimal total length	Given n cities, connect them in a path of minimal total length
Can be solved efficiently	No polynomial algorithm known!

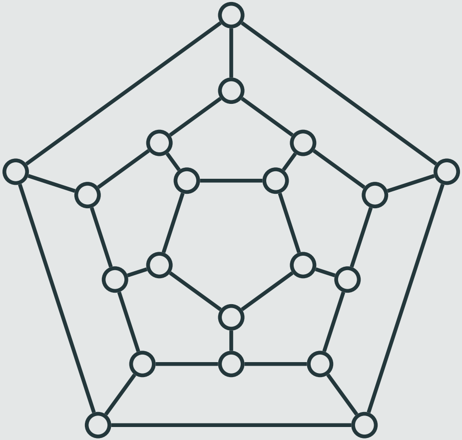
Longest Path

Longest path

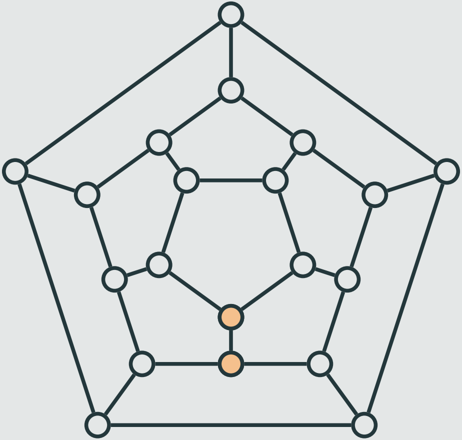
Input: A weighted graph, two vertices s, t, and a budget b.

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

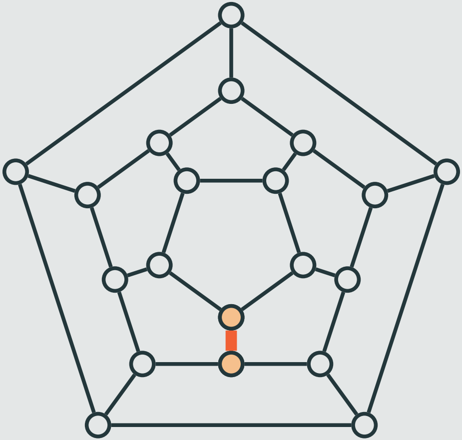
Example



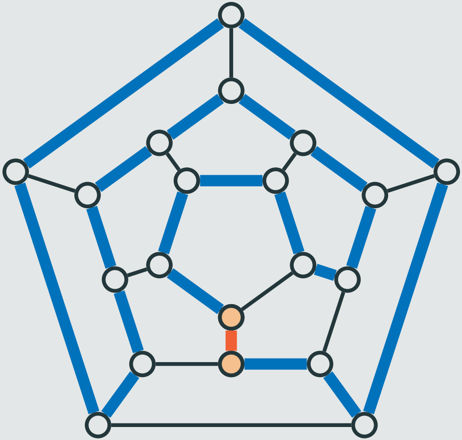
Example



Example



Example



Shortest path

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

Shortest path

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

Can be solved efficiently

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	Longest path
Find a simple path from s to t of total length at most b	Find a simple path from s to t of total length at least b
Can be solved efficiently	No polynomial algorithm known!

Integer Linear Programming Problem

Integer linear programming

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

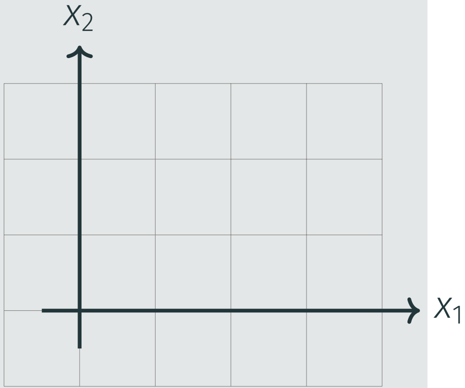
Output: Integer solution.

Example

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

Example

$$x_1 \geq 0.5$$
$$-x_1 + 8x_2 \geq 0$$
$$-x_1 - 8x_2 \geq -8$$

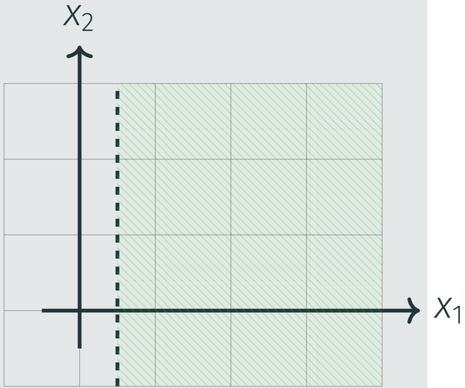


Example

$$x_1 \geq 0.5$$

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

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

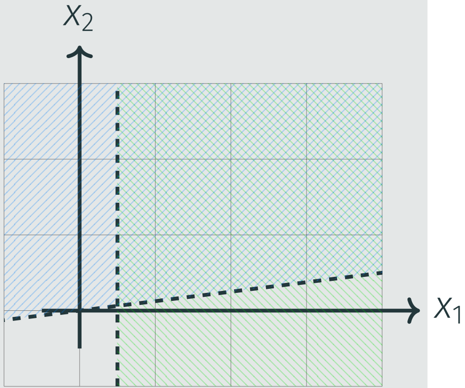


Example

$$x_1 \geq 0.5$$

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

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

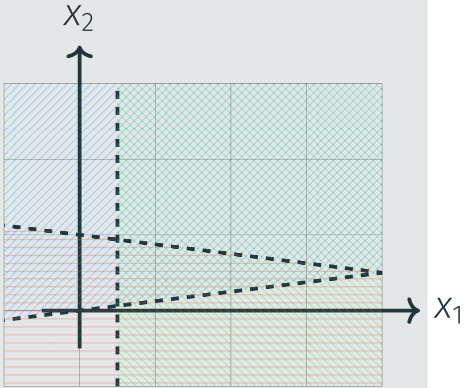


Example

$$x_1 \geq 0.5$$

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

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



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	ILP
Find a real solution of a system of linear inequalities	Find an integer solution of a system of linear inequalities
Can be solved efficiently	

Integer Linear Programming

LP	ILP
Find a real solution of a system of linear inequalities	Find an integer solution of a system of linear inequalities
Can be solved efficiently	No polynomial algorithm known!

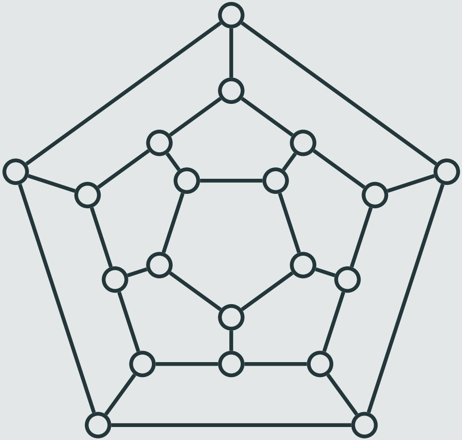
Independent Set Problem

Independent set

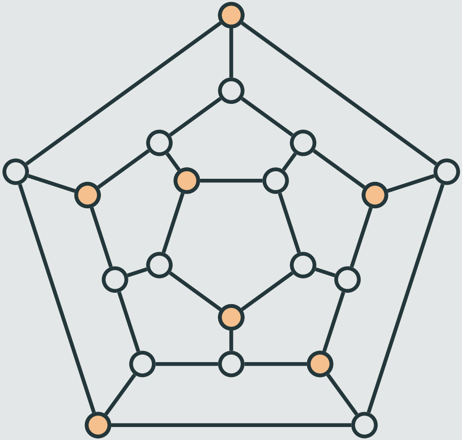
Input: A graph and a budget b.

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

Example

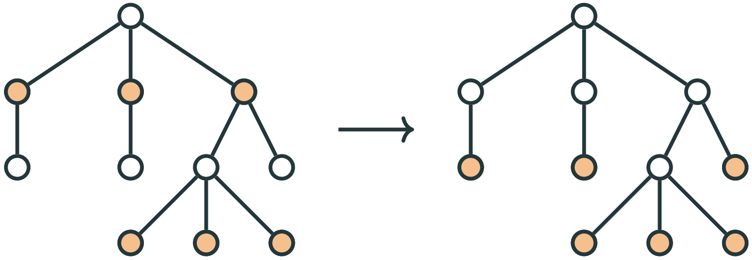


Example



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	Independent set in a graph
Find an independent set of size at least b in a given tree	Find an independent set of size at least b in a given graph
Can be solved efficiently	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

Class P

Problems whose solution can be found efficiently

- MST
- Shortest path
- LP
- IS on trees

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

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

The main open problem in Computer Science

Is P equal to NP?

The main open problem in Computer Science

Is P equal to NP?

Millenium Prize Problem

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

- If 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≠NP, then there exist search problems that cannot be solved in polynomial time.

Reductions

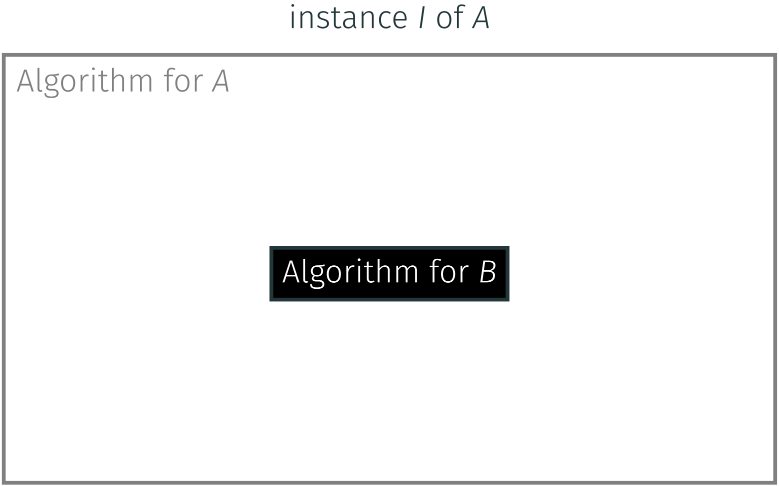
Informally

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

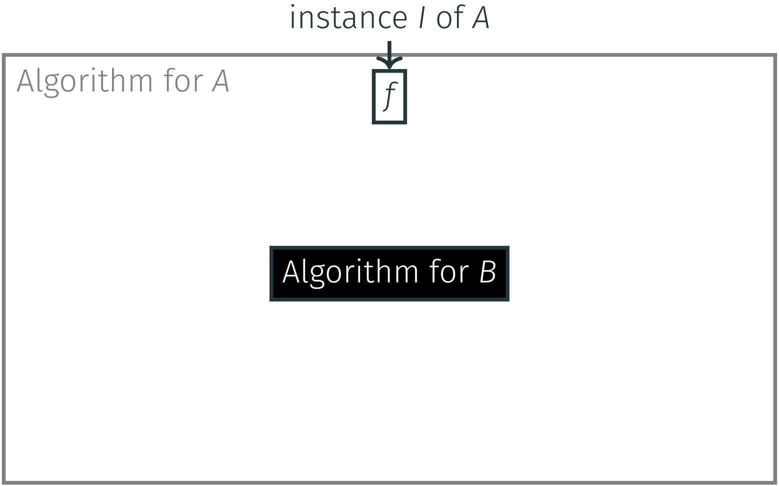
Reduction: $A \to B$

instance I of A

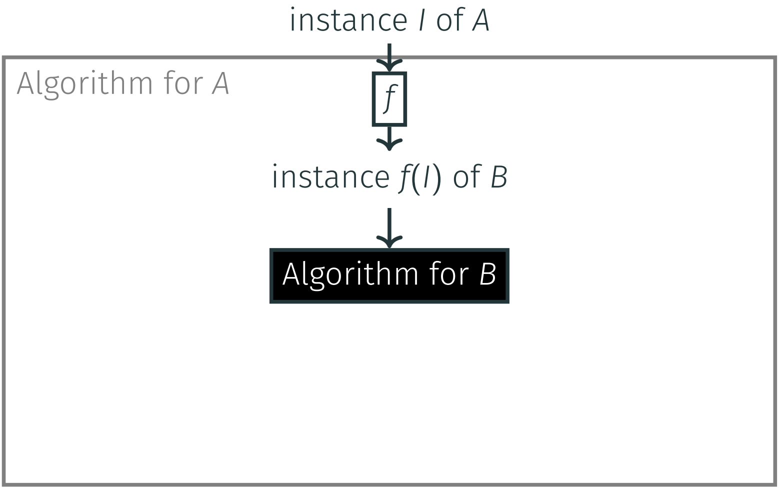
Reduction: $A \to B$



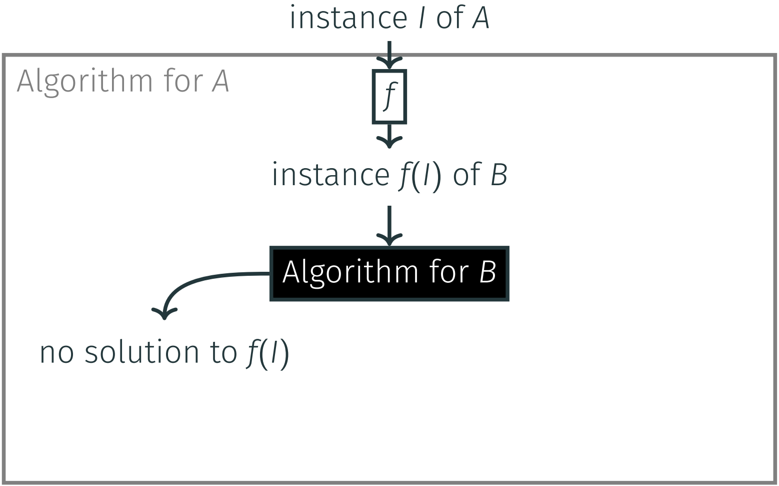
Reduction: $A \to B$



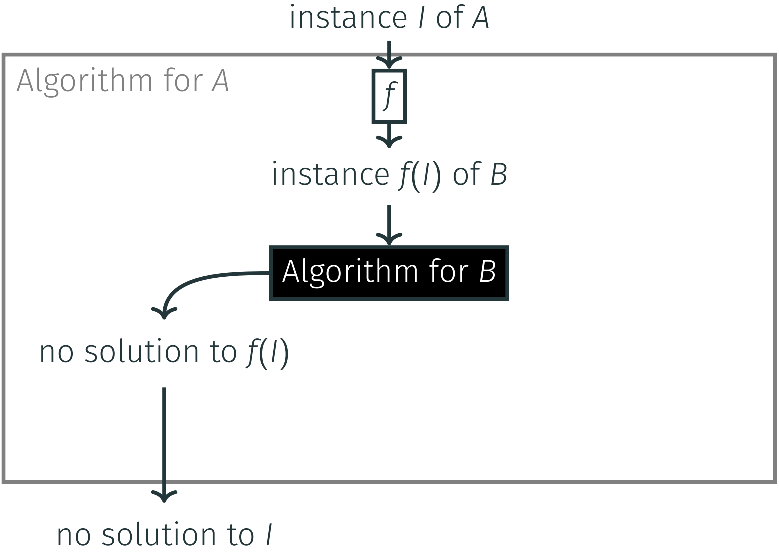
Reduction: $A \to B$



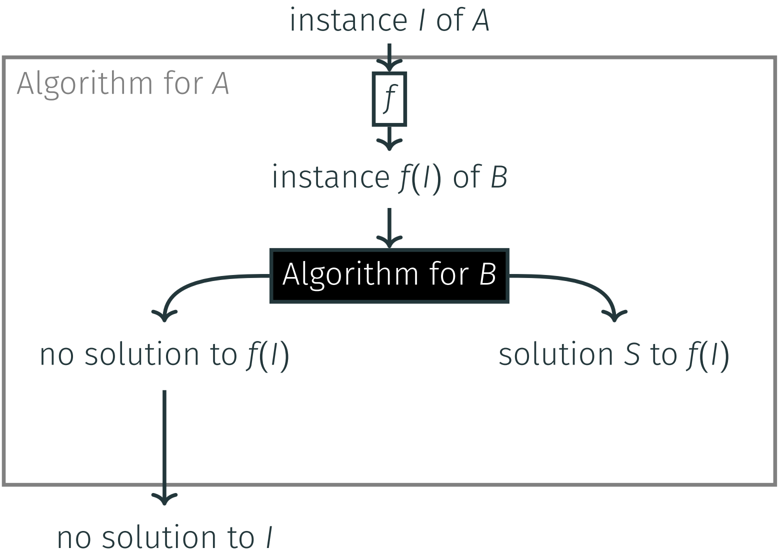
Reduction: $A \to B$



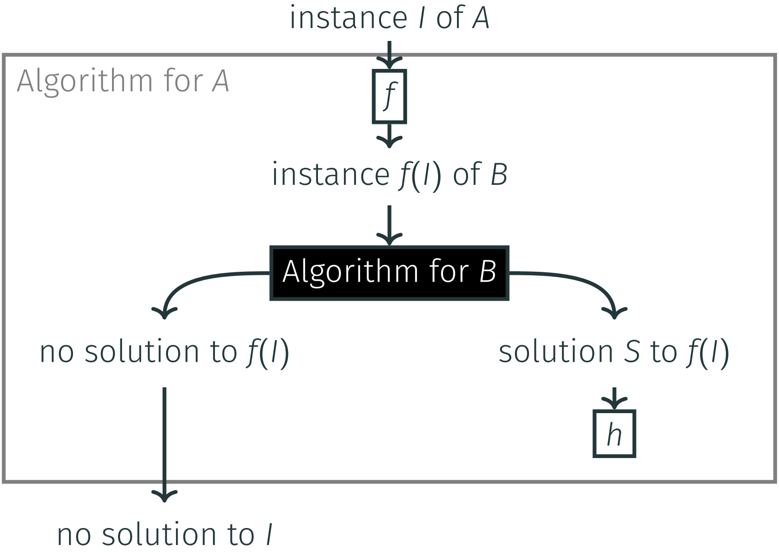
Reduction: $A \to B$



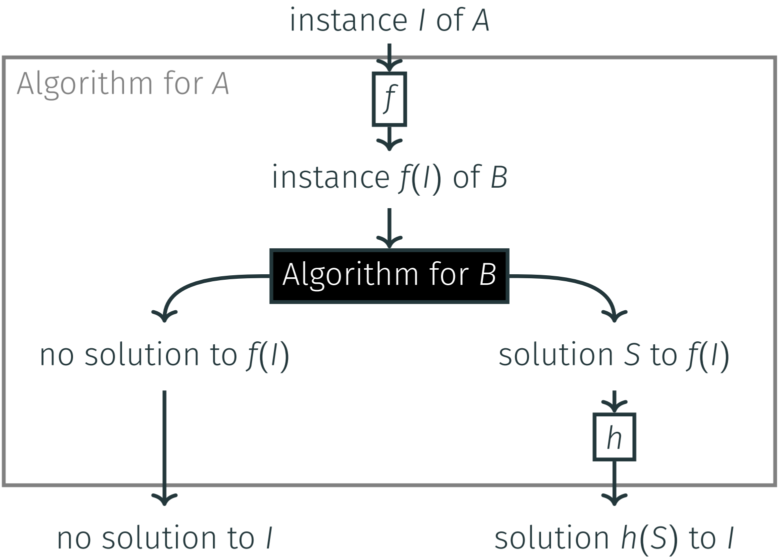
Reduction: $A \to B$



REDUCTION: $A \to B$



Reduction: $A \to B$

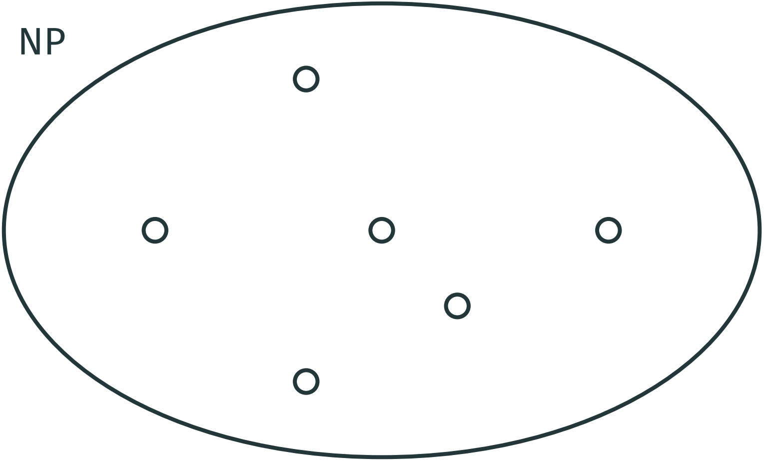


Formally

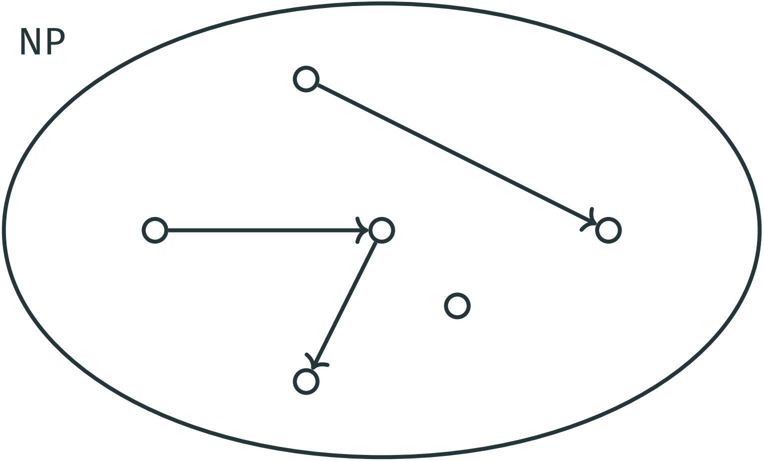
Definition

We say that a search problem A is reduced to a search problem B and write $A \to B$, 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

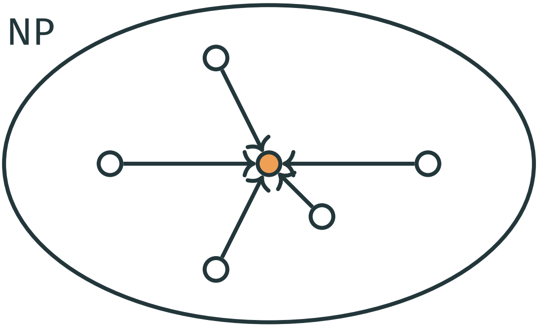
Definition

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!

Two ways of using $A \to B$:

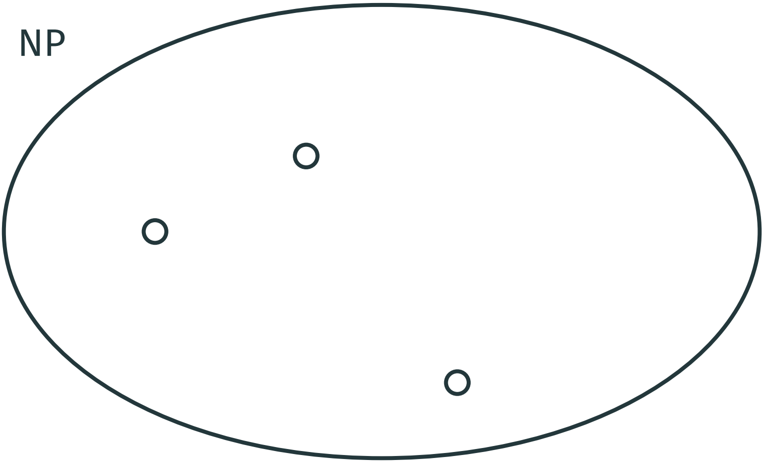
- if B is easy (can be solved in polynomial time), then so is A
- if A is hard (cannot be solved in polynomial time), then so is B

Reductions Compose

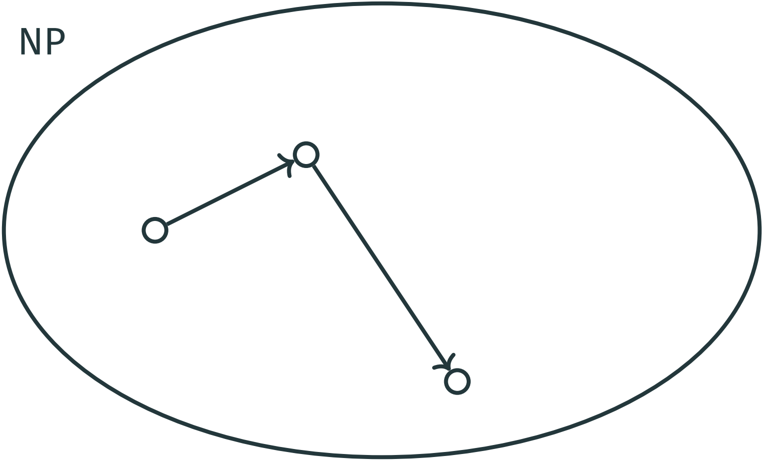
Lemma

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

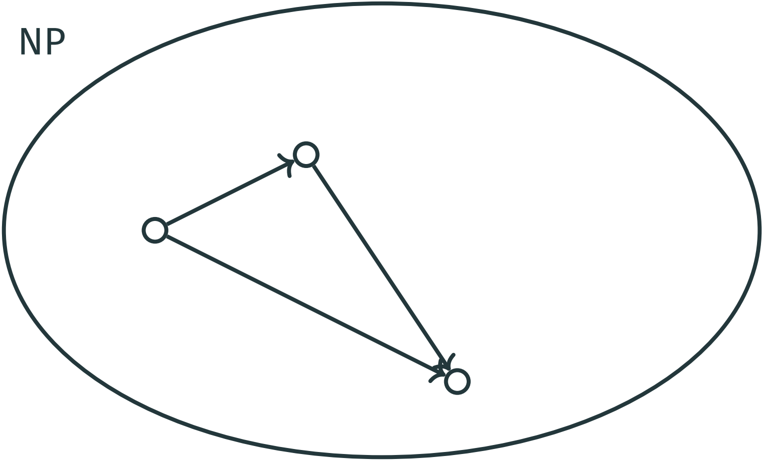
Pictorially



Pictorially



Pictorially



Showing NP-completeness

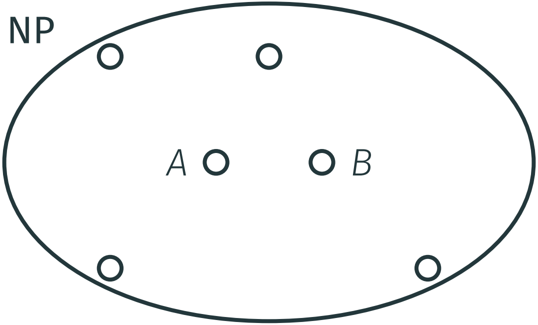
Corollary

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

Showing NP-completeness

Corollary

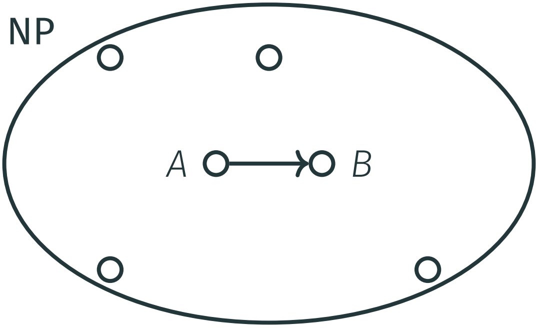
If $A \to B$ and A is NP-complete, then so is B.



Showing NP-completeness

Corollary

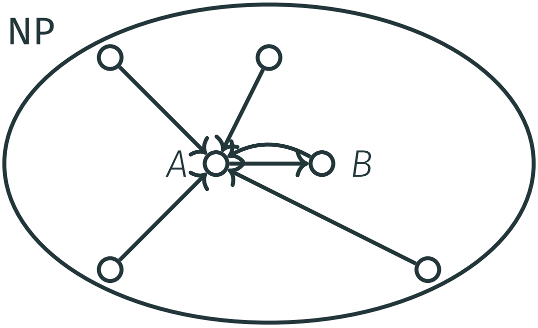
If $A \rightarrow B$ and A is NP-complete, then so is B.



Showing NP-completeness

Corollary

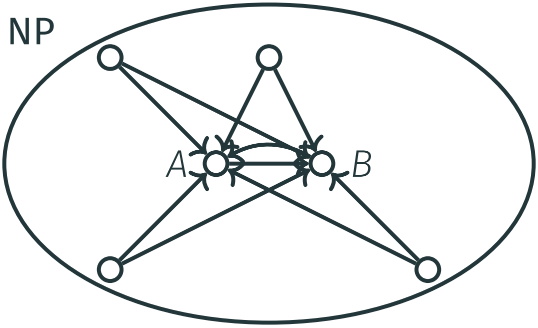
If $A \rightarrow B$ and A is **NP**-complete, then so is B.



Showing NP-completeness

Corollary

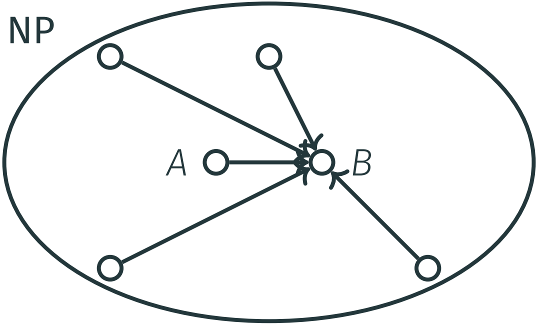
If $A \to B$ and A is NP-complete, then so is B.



Showing NP-completeness

Corollary

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



NP-Completeness of SAT

Goal

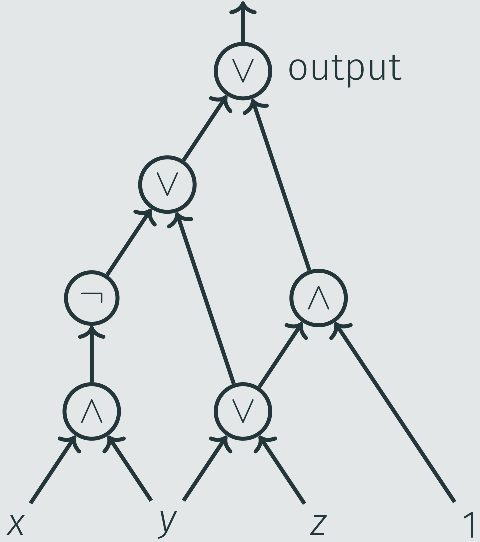
Show that every search problem reduces to SAT.

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.

Circuit



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.

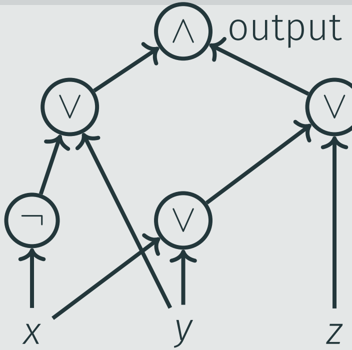
Circuit-SAT

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



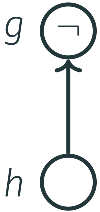
Circuit-SAT $\rightarrow$ SAT

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

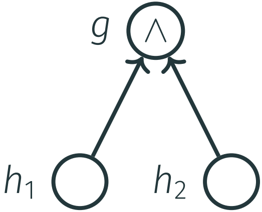
- Introduce a Boolean variable for each gate
- For each gate, write down a few clauses that describe the relationship between this gate and its direct predecessors

NOT Gates



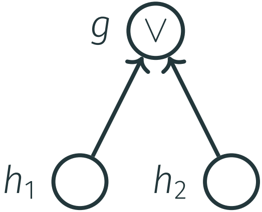
$$(h \vee g)(\overline{h} \vee \overline{g})$$

AND Gates



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

OR Gates



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

Output Gate

$g \bigcirc$ output $\qquad (g)$

- 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

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.

- 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 $\mathcal{C}(I, S)$ 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)

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:

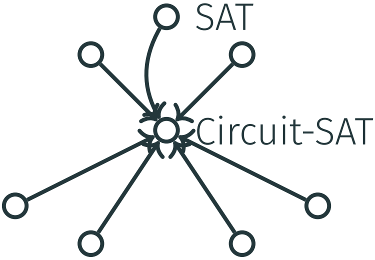
- take a circuit corresponding to $\mathcal{C}(I, \cdot)$
- the inputs to this circuit encode candidate solutions

Reduction

To solve an instance I of the problem A:

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

Summary



Summary