Algorithms and Complexity

2. Complexity

Marc Hellmuth

University of Stockholm

Reminder: O-, Θ- and Ω-Notation

For positive functions f and g, we define

- $g(n) \in O(f(n)) :\Leftrightarrow \exists c > 0, n_0 > 0 : \forall n > n_0 : g(n) \leq cf(n)$
- $g(n) \in \Omega(f(n)) :\Leftrightarrow \exists c > 0, n_0 > 0 : \forall n > n_0 : g(n) \geq cf(n)$
- $g(n) \in \Theta(f(n)) :\Leftrightarrow g(n) \in O(f(n))$ and $g(n) \in \Omega(f(n))$.

The notation $g(n) = O(f(n))$ is also very commonly used.

WHITEBOARD: merge sort vs insertion sort $\Longrightarrow$ runtime matters!

Complexity

Question: Does there exist algorithms for every problem to solve it in polynomial time?

Complexity

Question: Does there exist algorithms for every problem to solve it in polynomial time?

Answer

A1 NO, there are problems that cannot be solved by any algorithm [e.g. Halting Problem]

Complexity

Question: Does there exist algorithms for every problem to solve it in polynomial time?

Answer

- A1 NO, there are problems that cannot be solved by any algorithm [e.g. Halting Problem]
- A2 Many problems can be solved by alorithms but not in polynomial-time (under reasonable assumptions)

Complexity

Question: Does there exist algorithms for every problem to solve it in polynomial time?

Answer

A1 NO, there are problems that cannot be solved by any algorithm [e.g. Halting Problem]

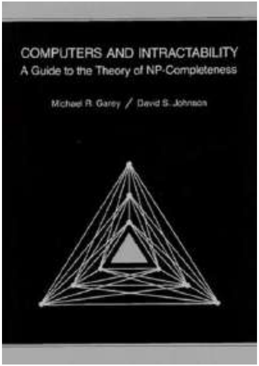
A2 Many problems can be solved by alorithms but not in polynomial-time (under reasonable assumptions)

A1 WHITEBOARD: unsolvable (undecidable) problem

- the halting problem
- almost all decision problems cannot be solved by algorithms.

A2 Theory of NP-completeness

Theory of NP-completeness: Literature



Computers and Intractability: A Guide to the Theory of NP-Completeness
by Michael R. Garey, David S. Johnson
Published January 15th 1979 by W. H. Freeman

In a nutshell:

Aim: How difficult or easy is a problem that we want to solve?

In a nutshell:

Aim: How difficult or easy is a problem that we want to solve?

Recap:

- There are problems that can be solved in polynomial time
  (EASY, tractable problems)

In a nutshell:

Aim: How difficult or easy is a problem that we want to solve?

Recap:

- There are problems that can be solved in polynomial time
  (EASY, tractable problems)

- There are problems where no algorithms exist to solve them in finite time (UNSOLVABLE)
  e.g. Halting-problem [determine whether a given algorithm terminates]

In a nutshell:

Aim: How difficult or easy is a problem that we want to solve?

Recap:

- There are problems that can be solved in polynomial time
  (EASY, tractable problems)

- There are problems where no algorithms exist to solve them in finite time (UNSOLVABLE)
  e.g. Halting-problem [determine whether a given algorithm terminates]

Questions:

- What about problems that can be solved in finite time, say there is an exponential-time algorithm with runtime $\mathcal{O}(1.5^n)$, but no polynomial-time algorithm has been found so-far?

In a nutshell:

Aim: How difficult or easy is a problem that we want to solve?

Recap:

- There are problems that can be solved in polynomial time
  (EASY, tractable problems)

- There are problems where no algorithms exist to solve them
  in finite time (UNSOLVABLE)
  e.g. Halting-problem [determine whether a given algorithm terminates]

Questions:

- What about problems that can be solved in finite time, say there is an exponential-time algorithm with runtime $\mathcal{O}(1.5^n)$, but no polynomial-time algorithm has been found so-far?

  $\Rightarrow$ Should we search further for polynomial-time algorithms?

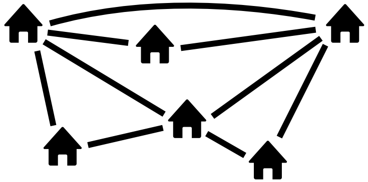
  Or could we stop searching, since there is some evidence that no polynomial-time algorithm may exist?

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.

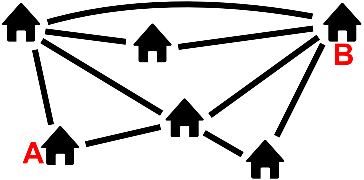
Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



Easy vs. Difficult is not obvious!

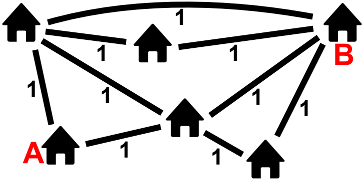
Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



Find a shortest path between A and B under the assumption that all edges e have weight

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an "easy" problem to a "difficult" one.

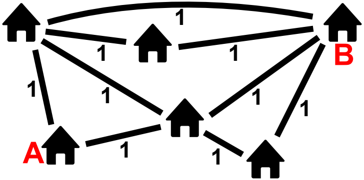


Find a shortest path between A and B under the assumption that all edges e have weight

$$w(e) = \mathbf{1}$$

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



Find a shortest path between A and B under the assumption that all edges e have weight

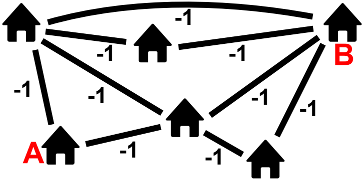
$w(e) = \mathbf{1}$

(solvable in polynomial time)

EASY!

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an "easy" problem to a "difficult" one.



Find a shortest path between A and B under the assumption that all edges e have weight

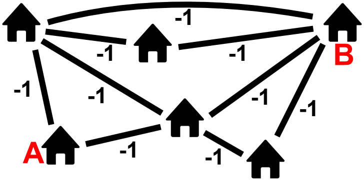
$w(e) = \mathbf{1}$ (solvable in polynomial time)

$w(e) = \mathbf{-1}$

EASY!

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



Find a shortest path between A and B under the assumption that all edges e have weight

$w(e) =$ **1**

(solvable in polynomial time)

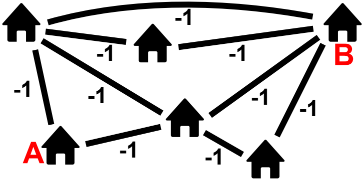
EASY!

$w(e) =$ **-1**

Up to now, only exponential-time algorithms are known!

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an "easy" problem to a "difficult" one.



Find a shortest path between A and B under the assumption that all edges e have weight

$w(e) =$ **1**

(solvable in polynomial time)

EASY!

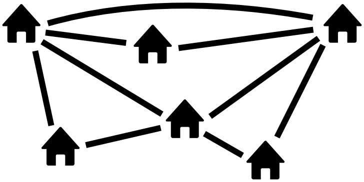
$w(e) =$ **-1**

Up to now, only exponential-time algorithms are known!

DIFFICULT?

Easy vs. Difficult is not obvious!

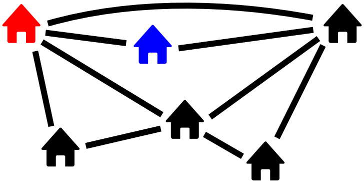
Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



Find a coloring of the houses such that neighbors have different colors

Easy vs. Difficult is not obvious!

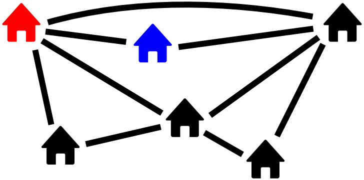
Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



Find a coloring of the houses such that neighbors have different colors

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.

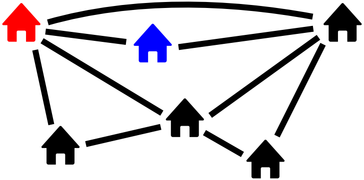


Find a coloring of the houses such that neighbors have different colors

with **2 colors**

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



Find a coloring of the houses such that neighbors have different colors

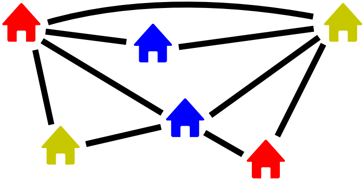
with **2 colors**

solvable in polynomial time

EASY!

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



Find a coloring of the houses such that neighbors have different colors

with **2 colors**

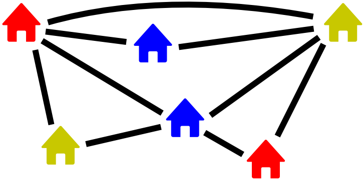
solvable in polynomial time

EASY!

with **3 colors**

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



Find a coloring of the houses such that neighbors have different colors

with **2 colors**

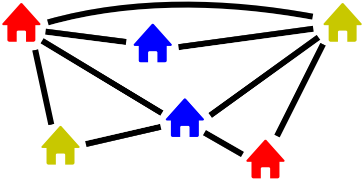
solvable in polynomial time

EASY!

with **3 colors**

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



Find a coloring of the houses such that neighbors have different colors

with **2 colors**

solvable in polynomial time

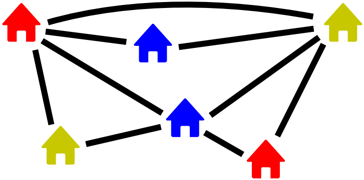
EASY!

with **3 colors**

Up to now, only exponential-time algorithms are known!

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



Find a coloring of the houses such that neighbors have different colors

with **2 colors**

solvable in polynomial time

EASY!

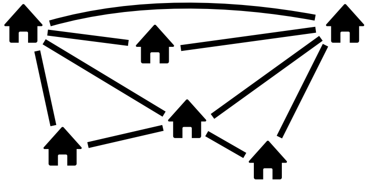
with **3 colors**

Up to now, only exponential-time algorithms are known!

DIFFICULT?

Easy vs. Difficult is not obvious!

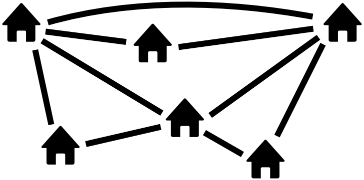
Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



It doesn’t seem to be a simple task to distinguish between **difficult** and **easy**!

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an "easy" problem to a "difficult" one.

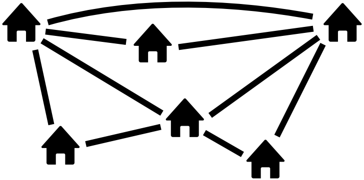


It doesn't seem to be a simple task to distinguish between **difficult** and **easy**!

(Q1) What does **difficult** formally mean?

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



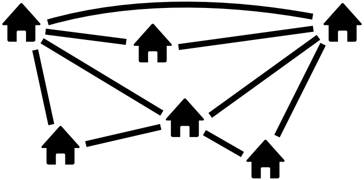
It doesn’t seem to be a simple task to distinguish between **difficult** and **easy**!

(Q1) What does **difficult** formally mean?

(Q2) Are there problems for which no polynomial-time algorithm exists *under reasonable assumptions*? **(= intractable problems)**

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an "easy" problem to a "difficult" one.



It doesn't seem to be a simple task to distinguish between **difficult** and **easy**!

(Q1) What does **difficult** formally mean?

(Q2) Are there problems for which no polynomial-time algorithm exists *under reasonable assumptions*? **(= intractable problems)**

What are these assumptions?

Easy vs. Difficult is not obvious!

- Main Ingredients
  - Optimization problems vs. decision problems
  - Classes P and NP
  - Reduction and NP-hardness
  - NP-completeness

We start with a brief overview of the main ingredients to answer these questions.

Optimization Problems vs. Decision Problems

Optimization Problems vs. Decision Problems

- An **optimization problem** has as solution one that satisfies pre-described optimality constraints

Optimization Problems vs. Decision Problems

- An **optimization problem** has as solution one that satisfies pre-described optimality constraints

  Shortest Path: Find between two houses A and B a path of min-length.

Optimization Problems vs. Decision Problems

- An **optimization problem** has as solution one that satisfies pre-described optimality constraints

  Shortest Path: Find between two houses A and B a path of min-length.

- A **decision problem** has as solution only an answer YES or NO

Optimization Problems vs. Decision Problems

- An **optimization problem** has as solution one that satisfies pre-described optimality constraints

  Shortest Path: Find between two houses A and B a path of min-length.

- A **decision problem** has as solution only an answer YES or NO

  Shortest Path: Does there exist a path between two houses A and B of length $\leq K$ for some $K \in \mathbb{N}$?

Optimization Problems vs. Decision Problems

- An **optimization problem** has as solution one that satisfies pre-described optimality constraints

  Shortest Path: Find between two houses A and B a path of min-length.

- A **decision problem** has as solution only an answer YES or NO

  Shortest Path: Does there exist a path between two houses A and B of length $\leq K$ for some $K \in \mathbb{N}$?

Memory Hook: Opt. Problem "easy" $\Rightarrow$ Dec. Problem "easy"

Optimization Problems vs. Decision Problems

- An **optimization problem** has as solution one that satisfies pre-described optimality constraints

  Shortest Path: Find between two houses A and B a path of min-length.

- A **decision problem** has as solution only an answer YES or NO

  Shortest Path: Does there exist a path between two houses A and B of length $\leq K$ for some $K \in \mathbb{N}$?

Memory Hook: Opt. Problem “easy” $\Rightarrow$ Dec. Problem “easy”
Dec. Problem “difficult” $\Rightarrow$ Opt. Problem “difficult”

Optimization Problems vs. Decision Problems

- An **optimization problem** has as solution one that satisfies pre-described optimality constraints

  Shortest Path: Find between two houses A and B a path of min-length.

- A **decision problem** has as solution only an answer YES or NO

  Shortest Path: Does there exist a path between two houses A and B
  of length $\leq K$ for some $K \in \mathbb{N}$?

Memory Hook: Opt. Problem "easy" $\Rightarrow$ Dec. Problem "easy"
Dec. Problem "difficult" $\Rightarrow$ Opt. Problem "difficult"

To classify "difficulty", decisions problems are used:

- To prove that an optimization problem is difficult, it suffices to show that the corresponding decision problem is difficult.

Optimization Problems vs. Decision Problems

- An **optimization problem** has as solution one that satisfies pre-described optimality constraints

  Shortest Path: Find between two houses A and B a path of min-length.

- A **decision problem** has as solution only an answer YES or NO

  Shortest Path: Does there exist a path between two houses A and B of length $\leq K$ for some $K \in \mathbb{N}$?

Memory Hook: Opt. Problem "easy" $\Rightarrow$ Dec. Problem "easy"
Dec. Problem "difficult" $\Rightarrow$ Opt. Problem "difficult"

To classify "difficulty", decisions problems are used:

- To prove that an optimization problem is difficult, it suffices to show that the corresponding decision problem is difficult.
- Decision problems have a very natural, formal counterpart called "language", as we have seen in the section about "TM and languages that are accepted by TM"

Outline

- Main Ingredients
  - Optimization problems vs. decision problems ✓
  - Classes P and NP
  - Reduction and NP-hardness
  - NP-completeness

TM and Algorithms

A TM M accepts $w \in \Sigma^* \iff M$ halts in state q_{accept} for w as input.

$L(M) := \{w \in \Sigma^* \mid M \text{ accepts } w\}$.

Note, if $w \in L(M) \setminus \Sigma^*$, then M with input w may never halt or rejects

TM and Algorithms

A TM M accepts $w \in \Sigma^* \iff M$ halts in state q_{accept} for w as input.

$\mathsf{L}(M) := \{w \in \Sigma^* \mid M \text{ accepts } w\}$.

Note, if $w \in \mathsf{L}(M) \setminus \Sigma^*$, then M with input w may never halt or rejects

A language $L \subseteq \Sigma^*$ is TM-recognizable $\iff$ there is TM M such that $L = \mathsf{L}(M)$

TM and Algorithms

A TM M accepts $w \in \Sigma^* \iff M$ halts in state q_{accept} for w as input.

$\mathsf{L}(M) := \{w \in \Sigma^* \mid M \text{ accepts } w\}$.

Note, if $w \in \mathsf{L}(M) \setminus \Sigma^*$, then M with input w may never halt or rejects

A language $L \subseteq \Sigma^*$ is TM-recognizable $\iff$ there is TM M such that $L = \mathsf{L}(M)$

A decision problem D can be viewed as a language encoding all yes-instances:
$L_D = \{\text{all instances } w \in D \text{ with YES-answer}\}$
Hence, any $w \in L_D \setminus \Sigma^*$ does not encode an instance of D or the answer is NO.

TM and Algorithms

A TM M accepts $w \in \Sigma^* \iff M$ halts in state q_{accept} for w as input.

$\mathsf{L}(M) := \{w \in \Sigma^* \mid M \text{ accepts } w\}$.

Note, if $w \in \mathsf{L}(M) \setminus \Sigma^*$, then M with input w may never halt or rejects

A language $L \subseteq \Sigma^*$ is TM-recognizable $\iff$ there is TM M such that $L = \mathsf{L}(M)$

A decision problem D can be viewed as a language encoding all yes-instances:
$L_D = \{\text{all instances } w \in D \text{ with YES-answer}\}$
Hence, any $w \in L_D \setminus \Sigma^*$ does not encode an instance of D or the answer is NO.

A TM M (equ. an algorithm) solves a decision problem $D \iff L_D = \mathsf{L}(M)$ and it halts for all $w \in \Sigma^*$. If this TM runs in polynomial-time, then D is said to be polynomial-time solvable.

TM and Algorithms

A TM M accepts $w \in \Sigma^* \iff M$ halts in state q_{accept} for w as input.

$\mathsf{L}(M) := \{w \in \Sigma^* \mid M \text{ accepts } w\}$.

Note, if $w \in \mathsf{L}(M) \setminus \Sigma^*$, then M with input w may never halt or rejects

A language $L \subseteq \Sigma^*$ is TM-recognizable $\iff$ there is TM M such that $L = \mathsf{L}(M)$

A decision problem D can be viewed as a language encoding all yes-instances:
$L_D = \{\text{all instances } w \in D \text{ with YES-answer}\}$
Hence, any $w \in L_D \setminus \Sigma^*$ does not encode an instance of D or the answer is NO.

A TM M (equ. an algorithm) solves a decision problem $D \iff L_D = \mathsf{L}(M)$ and it halts for all $w \in \Sigma^*$. If this TM runs in polynomial-time, then D is said to be polynomial-time solvable.

The TMs considered so-far are deterministic, i.e., each step of computation is uniquely determined by the transition function δ and the same input will always yield the same computational steps.

P vs. NP

$\mathrm{P} = \{$dec. probl. that can be solved by **deterministic** alg. in polynomial time$\}$

P vs. NP

$\mathrm{P} = \{$dec. probl. that can be solved by **deterministic** alg. in polynomial time$\}$
$= \{$dec. probl. **solvable** in polynomial time$\}$

P vs. NP

$$\mathrm{P} = \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\}$$

$$\mathrm{NP} = \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\}$$

P vs. NP

$$\mathrm{P} = \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\}$$

$$\mathrm{NP} = \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\}$$

What is a non-deterministic algorithm ?

P vs. NP

$$\begin{aligned} \mathrm{P} &= \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\} \\ &= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\} \end{aligned}$$

$$\mathrm{NP} = \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\}$$

What is a non-deterministic algorithm ?

A non-deterministic algorithm consists of two separate stages:

P vs. NP

$$\mathrm{P} = \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\}$$

$$\mathrm{NP} = \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\}$$

What is a non-deterministic algorithm ?

A non-deterministic algorithm consists of two separate stages:

guessing stage [*Guess a solution once*]

non-deterministic part: If there is a YES-solution, then one of these solutions is returned!

P vs. NP

$$\begin{aligned} \mathrm{P} &= \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\} \\ &= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\} \end{aligned}$$

$$\mathrm{NP} = \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\}$$

What is a non-deterministic algorithm ?

A non-deterministic algorithm consists of two separate stages:

guessing stage [*Guess a solution once*]

non-deterministic part: If there is a YES-solution, then one of these solutions is returned!

checking stage [*Verify if the solution is a YES-answer*]

computed in a normal deterministic manner.

P vs. NP

$$\mathrm{P} = \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\}$$

$$\mathrm{NP} = \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{verifiable} \text{ in polynomial time}\}$$

What is a non-deterministic algorithm ?

A non-deterministic algorithm consists of two separate stages:

guessing stage [*Guess a solution once*]

non-deterministic part: If there is a YES-solution, then one of these solutions is returned!

checking stage [*Verify if the solution is a YES-answer*]

computed in a normal deterministic manner.

P vs. NP

$$\text{P} = \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\}$$

$$\text{NP} = \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{verifiable} \text{ in polynomial time}\}$$

What is a non-deterministic algorithm ?

A non-deterministic algorithm consists of two separate stages:

guessing stage [*Guess a solution once*]

non-deterministic part: If there is a YES-solution, then one of these solutions is returned!

checking stage [*Verify if the solution is a YES-answer*]

computed in a normal deterministic manner.

NP means "**N**on-deterministic **P**olynomial" (**NOT:** *non-polynomial*!)

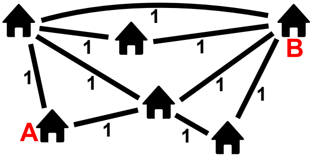
Keep in mind that algorithms are essentially defined in terms of TM.

P vs. NP

$$\begin{aligned} \mathrm{P} &= \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\} \\ &= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\} \end{aligned}$$

$$\begin{aligned} \mathrm{NP} &= \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\} \\ &= \{\text{dec. probl. } \textbf{verifiable} \text{ in polynomial time}\} \end{aligned}$$

Example: Shortest Path



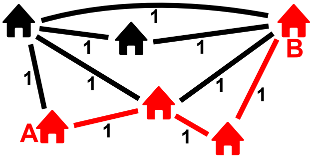
Question: Does there exist a path of length ≤ 4 from A to B?

P vs. NP

$\mathrm{P} = \{$dec. probl. that can be solved by **deterministic** alg. in polynomial time$\}$
$= \{$dec. probl. **solvable** in polynomial time$\}$

$\mathrm{NP} = \{$dec. probl. that can be solved by **non-deterministic** alg. in polynomial time$\}$
$= \{$dec. probl. **verifiable** in polynomial time$\}$

Example: Shortest Path



Question: Does there exist a path of length ≤ 4 from A to B?

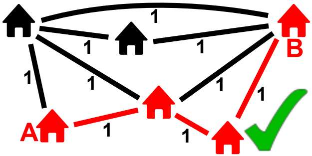
guessing stage [*Guess a solution once*]

P vs. NP

$$\mathrm{P} = \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\}$$

$$\mathrm{NP} = \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{verifiable} \text{ in polynomial time}\}$$

Example: Shortest Path



Question: Does there exist a path of length ≤ 4 from A to B?

guessing stage [*Guess a solution once*]

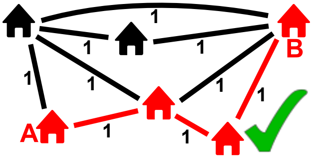
checking stage [*Verify if the solution is a YES-answer*]

P vs. NP

$$\mathrm{P} = \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\}$$

$$\mathrm{NP} = \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{verifiable} \text{ in polynomial time}\}$$

Example: Shortest Path $\in$ NP



Question: Does there exist a path of length ≤ 4 from A to B?

guessing stage [*Guess a solution once*]

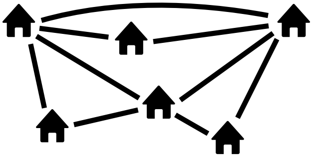
checking stage [*Verify if the solution is a YES-answer*]

P vs. NP

$$\begin{aligned} \mathrm{P} &= \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\} \\ &= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\} \end{aligned}$$

$$\begin{aligned} \mathrm{NP} &= \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\} \\ &= \{\text{dec. probl. } \textbf{verifiable} \text{ in polynomial time}\} \end{aligned}$$

Example: 3-coloring



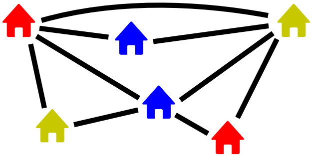
Question: Does there exist a 3-coloring?

P vs. NP

$$\mathrm{P} = \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\}$$

$$\mathrm{NP} = \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{verifiable} \text{ in polynomial time}\}$$

Example: 3-coloring



Question: Does there exist a 3-coloring?

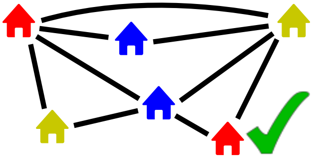
guessing stage [*Guess a solution once*]

P vs. NP

$$\mathrm{P} = \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\}$$

$$\mathrm{NP} = \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{verifiable} \text{ in polynomial time}\}$$

Example: 3-coloring



Question: Does there exist a 3-coloring?

guessing stage [*Guess a solution once*]

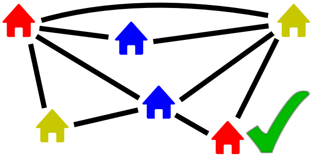
checking stage [*Verify if the solution is a YES-answer*]

P vs. NP

$\text{P} = \{$dec. probl. that can be solved by **deterministic** alg. in polynomial time$\}$
$\quad = \{$dec. probl. **solvable** in polynomial time$\}$

$\text{NP} = \{$dec. probl. that can be solved by **non-deterministic** alg. in polynomial time$\}$
$\quad = \{$dec. probl. **verifiable** in polynomial time$\}$

Example: 3-coloring $\in$ NP



Question: Does there exist a 3-coloring?

guessing stage [*Guess a solution once*]

checking stage [*Verify if the solution is a YES-answer*]

P vs. NP

$$\begin{aligned}\text{P} &= \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\} \\ &= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\}\end{aligned}$$

$$\begin{aligned}\text{NP} &= \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\} \\ &= \{\text{dec. probl. } \textbf{verifiable} \text{ in polynomial time}\}\end{aligned}$$

Observation: $P \subseteq NP$

P vs. NP

$$\begin{aligned} \text{P} &= \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\} \\ &= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\} \end{aligned}$$

$$\begin{aligned} \text{NP} &= \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\} \\ &= \{\text{dec. probl. } \textbf{verifiable} \text{ in polynomial time}\} \end{aligned}$$

Observation: $P \subseteq NP$

P-NP-Problem: $P \subset NP$ or $P = NP$?

P vs. NP

$$\text{P} = \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\}$$

$$\text{NP} = \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{verifiable} \text{ in polynomial time}\}$$

Observation: $P \subseteq NP$

P-NP-Problem: $P \subset NP$ or $P = NP$?

P vs. NP

$$\begin{aligned} \text{P} &= \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\} \\ &= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\} \end{aligned}$$

$$\begin{aligned} \text{NP} &= \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\} \\ &= \{\text{dec. probl. } \textbf{verifiable} \text{ in polynomial time}\} \end{aligned}$$

Observation: $P \subseteq NP$

P-NP-Problem: $P \subset NP$ or $P = NP$?

If $P = NP$, then verifying a solution is as easy as finding a solution
(quite unlikely, but still unsolved!)

P vs. NP

$$\mathrm{P} = \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\}$$

$$\mathrm{NP} = \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{verifiable} \text{ in polynomial time}\}$$

Observation: $P \subseteq NP$

P-NP-Problem: $P \subset NP$ or $P = NP$?

If $P = NP$, then verifying a solution is as easy as finding a solution

(quite unlikely, but still unsolved!)

(Q2) Are there problems for which no polynomial-time algorithm exists *under reasonable assumptions*? **(= intractable problems)**

What are these assumptions?

P vs. NP

$\text{P} = \{$dec. probl. that can be solved by **deterministic** alg. in polynomial time$\}$
$= \{$dec. probl. **solvable** in polynomial time$\}$

$\text{NP} = \{$dec. probl. that can be solved by **non-deterministic** alg. in polynomial time$\}$
$= \{$dec. probl. **verifiable** in polynomial time$\}$

Observation: $P \subseteq NP$

P-NP-Problem: $P \subset NP$ or $P = NP$?

If $P = NP$, then verifying a solution is as easy as finding a solution
(quite unlikely, but still unsolved!)

(Q2) Are there problems for which no polynomial-time algorithm exists *under reasonable assumptions*? **(= intractable problems)**

What are these assumptions? **Answer: $\mathbf{P} \subset \mathbf{NP}$**

P vs. NP

$$\text{P} = \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\}$$

$$\text{NP} = \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{verifiable} \text{ in polynomial time}\}$$

Observation: $P \subseteq NP$

P-NP-Problem: $P \subset NP$ or $P = NP$?

If $P = NP$, then verifying a solution is as easy as finding a solution *(quite unlikely, but still unsolved!)*

(Q2) Are there problems for which no polynomial-time algorithm exists *under reasonable assumptions*? **(= intractable problems)**

What are these assumptions? **Answer: $\mathbf{P} \subset \mathbf{NP}$**

$\Rightarrow$ There are problems in $NP \setminus P \neq \emptyset$
These problems can be verified but not solved in polynomial time!

P vs. NP

$$\mathrm{P} = \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\}$$

$$\mathrm{NP} = \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\}$$
$$= \{\text{dec. probl. } \textbf{verifiable} \text{ in polynomial time}\}$$

Observation: $P \subseteq NP$

P-NP-Problem: $P \subset NP$ or $P = NP$?

If $P = NP$, then verifying a solution is as easy as finding a solution
(quite unlikely, but still unsolved!)

(Q2) Are there problems for which no polynomial-time algorithm exists
under reasonable assumptions? **(= intractable problems)**

What are these assumptions? **Answer: $\mathbf{P} \subset \mathbf{NP}$**

$\Rightarrow$ There are problems in $NP \setminus P \neq \emptyset$
These problems can be verified but not solved in polynomial time!

The formal theory of P and NP is defined in term of languages and Turing machines

Outline

- Main Ingredients
  - Optimization problems vs. decision problems ✓
  - Classes P and NP ($P \subsetneq NP$)✓
  - Reduction and NP-hardness
  - NP-completeness

Reduction . . .

. . . is about transforming one problem into another problem

Reduction . . .

. . . is about transforming one problem into another problem

ARRR, we leave you at this lonely island if you cannot answer the following question correctly within 2 minutes:

Are there less then 12222 coins in the chest?

Reduction . . .

. . . is about transforming one problem into another problem

ARRR, we leave you at this lonely island if you cannot answer the following question correctly within 2 minutes:

Are there less then 12222 coins in the chest?

Reduction . . .

. . . is about transforming one problem into another problem

ARRR, we leave you at this lonely island if you cannot answer the following question correctly within 2 minutes:

Are there less then 12222 coins in the chest?

Reduction . . .

. . . is about transforming one problem into another problem

ARRR, we leave you at this lonely island if you cannot answer the following question correctly within 2 minutes:

Are there less then 12222 coins in the chest?

YOU: 1 coin = 8g

Reduction . . .

. . . is about transforming one problem into another problem

ARRR, we leave you at this lonely island if you cannot answer the following question correctly within 2 minutes:

Are there less then 12222 coins in the chest?

YOU: 1 coin = 8g
all coins = 97760 g
= 12220 * 8g

Reduction . . .

. . . is about transforming one problem into another problem

ARRR, we leave you at this lonely island if you cannot answer the following question correctly within 2 minutes:

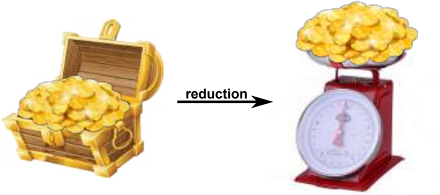
Are there less then 12222 coins in the chest?

YOU: 1 coin = 8g
all coins = 97760 g
= 12220 * 8g

Answer: YESSSS!!

Reduction . . .

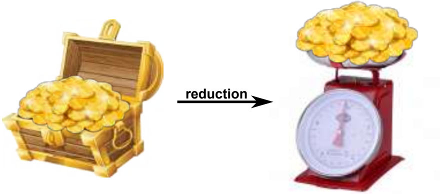
. . . is about transforming one problem into another problem



We reduced a counting problem to a weighting problem!

Reduction . . .

. . . is about transforming one problem into another problem



We reduced a counting problem to a weighting problem!

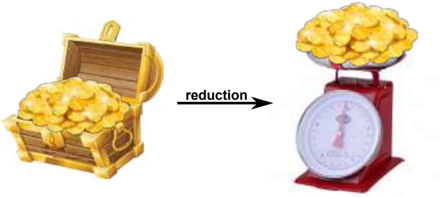
D, D' decision problems

D' (number coins)
IN: Set of coins & integer K
Q: Number of coins $\leq K$?

D (weight of items)
IN: Item i & integer L
Q: Weight(i)$\leq L$?

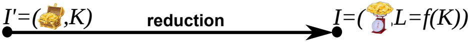
Reduction . . .

. . . is about transforming one problem into another problem



We reduced a counting problem to a weighting problem!

D, D' decision problems

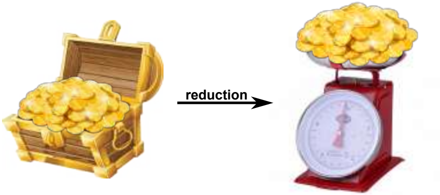


D' (number coins)
IN: Set of coins & integer K **Q:** Number of coins $\leq K$?

D (weight of items)
IN: Item i & integer L **Q:** Weight(i)$\leq L$?

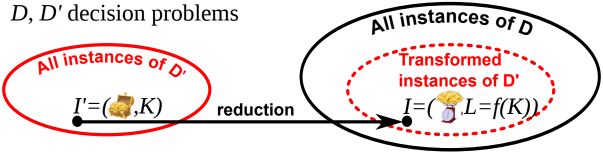
Reduction . . .

. . . is about transforming one problem into another problem



We reduced a counting problem to a weighting problem!

D, D' decision problems

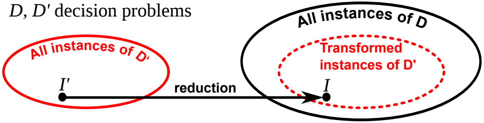


D' (number coins)
IN: Set of coins & integer K **Q:** Number of coins $\leq K$?

D (weight of items)
IN: Item i & integer L **Q:** Weight(i)$\leq L$?

Reduction

D, D' decision problems

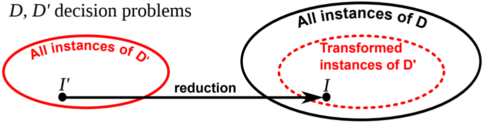


$\text{P} = \{$dec. probl. *solvable* in polynomial time$\}$

$\text{NP} = \{$dec. probl. *verifiable* in polynomial time$\}$

Reduction

D, D' decision problems



$\mathrm{P} \ = \{$dec. probl. *solvable* in polynomial time$\}$

$\mathrm{NP} = \{$dec. probl. *verifiable* in polynomial time$\}$

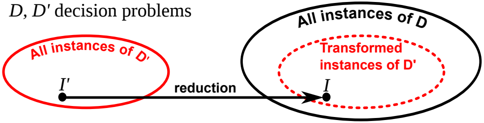
instance = *specified input*.

A **reduction** from D' to D is a procedure that transforms every instance I' of D' to an instance I of D such that

- the transformation can be done in *polynomial time* (= "easy") and
- I has YES-answer if and only if I' has YES-answer.

Reduction

D, D' decision problems



P $= \{$dec. probl. *solvable* in polynomial time$\}$

NP $= \{$dec. probl. *verifiable* in polynomial time$\}$

instance = *specified input*.

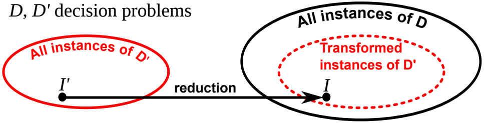
A **reduction** from D' to D is a procedure that transforms every instance I' of D' to an instance I of D such that

- the transformation can be done in *polynomial time* (= "easy") and
- I has YES-answer if and only if I' has YES-answer.

$\Rightarrow$ Every algorithm that solves D can be used to solve D'.

Reduction and NP-hard

D, D' decision problems



$\mathrm{P} \;\; = \{$dec. probl. *solvable* in polynomial time$\}$

$\mathrm{NP} = \{$dec. probl. *verifiable* in polynomial time$\}$

instance = *specified input*.

A **reduction** from D' to D is a procedure that transforms every instance I' of D' to an instance I of D such that

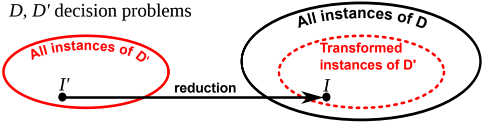
- the transformation can be done in *polynomial time* (= "easy") and
- I has YES-answer if and only if I' has YES-answer.

$\Rightarrow$ Every algorithm that solves D can be used to solve D'.

A dec. problem D is **NP-hard** if *every* problem $D' \in \mathrm{NP}$ can be reduced to D.

Reduction and NP-hard

D, D' decision problems



P $= \{$dec. probl. *solvable* in polynomial time$\}$

NP $= \{$dec. probl. *verifiable* in polynomial time$\}$

instance = *specified input*.

A **reduction** from D' to D is a procedure that transforms every instance I' of D' to an instance I of D such that

- the transformation can be done in *polynomial time* (= "easy") and
- I has YES-answer if and only if I' has YES-answer.

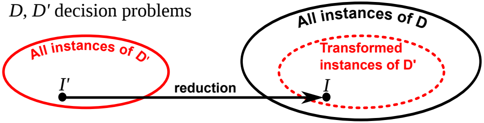
$\Rightarrow$ Every algorithm that solves D can be used to solve D'.

A dec. problem D is **NP-hard** if *every* problem $D' \in \mathrm{NP}$ can be reduced to D.

If D is NP-hard, then

Reduction and NP-hard

D, D' decision problems



$\mathrm{P} \;= \{$dec. probl. *solvable* in polynomial time$\}$
$\mathrm{NP} = \{$dec. probl. *verifiable* in polynomial time$\}$

instance = *specified input*.

A **reduction** from D' to D is a procedure that transforms every instance I' of D' to an instance I of D such that

- the transformation can be done in *polynomial time* (= "easy") and
- I has YES-answer if and only if I' has YES-answer.

$\Rightarrow$ Every algorithm that solves D can be used to solve D'.

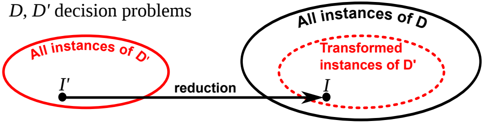
A dec. problem D is **NP-hard** if *every* problem $D' \in \mathrm{NP}$ can be reduced to D.

If D is NP-hard, then

- every problem $D' \in \mathrm{NP}$ can be considered as a "special case" of D.

Reduction and NP-hard

D, D' decision problems



$\text{P} \;= \{$dec. probl. *solvable* in polynomial time$\}$

$\text{NP} = \{$dec. probl. *verifiable* in polynomial time$\}$

instance = *specified input*.

A **reduction** from D' to D is a procedure that transforms every instance I' of D' to an instance I of D such that

- the transformation can be done in *polynomial time* (= "easy") and
- I has YES-answer if and only if I' has YES-answer.

$\Rightarrow$ Every algorithm that solves D can be used to solve D'.

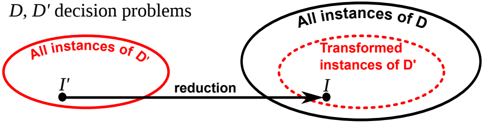
A dec. problem D is **NP-hard** if *every* problem $D' \in \text{NP}$ can be reduced to D.

If D is NP-hard, then

- every problem $D' \in \text{NP}$ can be considered as a "special case" of D.
- D is at least as difficult to solve as any other problem in NP.

Reduction and NP-hard

D, D' decision problems



$\mathrm{P} \;\; = \{$dec. probl. *solvable* in polynomial time$\}$
$\mathrm{NP} = \{$dec. probl. *verifiable* in polynomial time$\}$

instance = *specified input*.

A **reduction** from D' to D is a procedure that transforms every instance I' of D' to an instance I of D such that

- the transformation can be done in *polynomial time* (= "easy") and
- I has YES-answer if and only if I' has YES-answer.

$\Rightarrow$ Every algorithm that solves D can be used to solve D'.

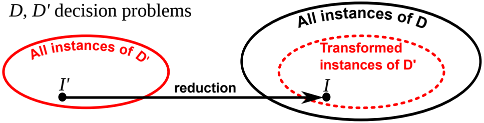
A dec. problem D is **NP-hard** if *every* problem $D' \in \mathrm{NP}$ can be reduced to D.

If D is NP-hard, then

- every problem $D' \in \mathrm{NP}$ can be considered as a "special case" of D.
- D is at least as difficult to solve as any other problem in NP.
- D is at least as difficult to solve as those problems in NP for which no polynomial-time algorithm exists ($P \subset NP$)

Reduction and NP-hard

D, D' decision problems



$\mathrm{P} \;\, = \{$dec. probl. *solvable* in polynomial time$\}$
$\mathrm{NP} = \{$dec. probl. *verifiable* in polynomial time$\}$

instance = *specified input*.

A **reduction** from D' to D is a procedure that transforms every instance I' of D' to an instance I of D such that

- the transformation can be done in *polynomial time* (= “easy”) and
- I has YES-answer if and only if I' has YES-answer.

$\Rightarrow$ Every algorithm that solves D can be used to solve D'.

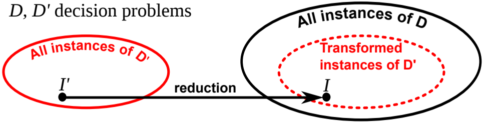
A dec. problem D is **NP-hard** if *every* problem $D' \in \mathrm{NP}$ can be reduced to D.

If D is NP-hard, then

- every problem $D' \in \mathrm{NP}$ can be considered as a “special case” of D.
- D is at least as difficult to solve as any other problem in NP.
- D is at least as difficult to solve as those problems in NP for which no polynomial-time algorithm exists ($P \subset NP$)
- it is reasonable to assume that for D there are no polynomial-time algorithm.

Reduction and NP-hard

D, D' decision problems



$\mathrm{P} \;= \{$dec. probl. *solvable* in polynomial time$\}$
$\mathrm{NP} = \{$dec. probl. *verifiable* in polynomial time$\}$

instance = *specified input*.

A **reduction** from D' to D is a procedure that transforms every instance I' of D' to an instance I of D such that

- the transformation can be done in *polynomial time* (= "easy") and
- I has YES-answer if and only if I' has YES-answer.

$\Rightarrow$ Every algorithm that solves D can be used to solve D'.

A dec. problem D is **NP-hard** if *every* problem $D' \in \mathrm{NP}$ can be reduced to D.

If D is NP-hard, then

- every problem $D' \in \mathrm{NP}$ can be considered as a "special case" of D.
- D is at least as difficult to solve as any other problem in NP.
- D is at least as difficult to solve as those problems in NP for which no polynomial-time algorithm exists ($P \subset NP$)
- it is reasonable to assume that for D there are no polynomial-time algorithm.

(Q1) What does **difficult** formally mean? **Answer:** NP-hard

Outline

- Main Ingredients
  - Optimization problems vs. decision problems ✓
  - Classes P and NP ($P \subsetneq NP$)✓
  - Reduction and NP-hardness ✓
  - NP-completeness

NP-complete

(Q1) What does **difficult** formally mean? **Answer:** NP-hard

(Q2) Are there problems for which no polynomial-time algorithm exists *under reasonable assumptions*? **(= intractable problems)**

What are these assumptions? **Answer:** $\mathbf{P} \subset \mathbf{NP}$

NP-complete

(Q1) What does **difficult** formally mean? **Answer:** NP-hard

(Q2) Are there problems for which no polynomial-time algorithm exists *under reasonable assumptions*? **(= intractable problems)**

Answer: Yes, NP-complete problems

What are these assumptions? **Answer:** $\mathbf{P} \subset \mathbf{NP}$

A decision problem D is **NP-complete** if

- $D \in \mathrm{NP}$
- D is NP-hard: *every* problem in NP can be reduced to D.

NP-complete

(Q1) What does **difficult** formally mean? **Answer:** NP-hard

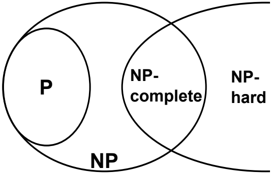
(Q2) Are there problems for which no polynomial-time algorithm exists *under reasonable assumptions*? **(= intractable problems)**

Answer: Yes, NP-complete problems

What are these assumptions? **Answer:** $\mathbf{P} \subset \mathbf{NP}$

A decision problem D is **NP-complete** if

- $D \in \mathrm{NP}$
- D is NP-hard: *every* problem in NP can be reduced to D.



NP-complete problems are the most difficult problems in NP.

NP-complete

Note, Languages and Decision problem can be seen as being "equivalent", i.e., in the following definition, we can use these terms interchangeably.

A decision problem D is **NP-complete** if

- $D \in \mathrm{NP}$
- D is NP-hard: *every* problem in NP can be reduced to D.

  In symbols,

$$\forall D' \in \mathrm{NP} \colon D' \preceq_p D$$

Q: How to show that EVERY problem in NP can be reduced to D?

A: There was a first problem "SAT" that was shown to be NP-complete:
Cook-Levin-Thm 1971 (without proof here)

$\Rightarrow$ Every problem $D' \in NP$ can be reduced to SAT.
$\Rightarrow$ If we can show for some problem D that SAT can be reduced to D
then every problem $D' \in NP$ can be reduced to D.
($\preceq_p$ is transitive)

$$\forall D' \in NP \colon D' \preceq_p \text{SAT and SAT} \preceq_p D \implies \forall D' \in NP \colon D' \preceq_p D$$

Outline

- Main Ingredients
  - Optimization problems vs. decision problems ✓
  - Classes P and NP ($P \subsetneq NP$)✓
  - Reduction and NP-hardness ✓
  - NP-completeness ✓

Outline

- Main Ingredients
  - Optimization problems vs. decision problems ✓
  - Classes P and NP ($P \subsetneq NP$)✓
  - Reduction and NP-hardness ✓
  - NP-completeness ✓

. . . and now examples !

WHITEBOARD:

SAT $\preceq_p$ 3-SAT $\preceq_p$ CLIQUE $\preceq_p$ VERTEX-COVER

3-SAT $\preceq_p$ VERTEX-COLORING

3-SAT $\preceq_p$ HAMILTONIAN PATH/CYCLE $\preceq_p$ TSP