Computational Complexity

Lecture 1: P, NP and NP-completeness

Ronald de Haan
me@ronalddehaan.eu

University of Amsterdam

April 4, 2024

Overview of practical information

- Lecturer: Ronald de Haan (`me@ronalddehaan.eu`)
- TAs: Wouter Vromen, Hannah Van Santvliet
- Course web page: `https://staff.science.uva.nl/r.dehaan/complexity2024/`
- Canvas page: `https://canvas.uva.nl/courses/42595`
- Discourse: `https://talk.computational-complexity.nl/`
- Book: *Computational Complexity: A Modern Approach* (Arora & Barak, 2009)

Discourse

- We'll use an online discussion board (using the *Discourse* system): `https://talk.computational-complexity.nl/`
  - Questions about the material

    (Feel free to answer each other's questions)
  - Reflecting on the material
  - Summarizing the material together

- Feel free to start discussion topics on any of these

Feedback

- During the course:
  - Please give your ideas for improvement, e.g., anonymously on Discourse
- After the course:
  - Please fill in the course evaluation questionnaire (for the OC and lecturer)

Course activities

- Lectures:
  - Twice 45 minutes, with 15 minute break in between, not recorded
- Exercise sessions:
  - Practice with material, discuss previous homework assignments
- Homework assignments (50% of grade):
  - Three assignments, hand in via Canvas
  - You may work in pairs (but you do not have to)
- Take-home exam (50% of grade):
  - At the end, open book, one week time to complete exam
- Online discussions, question answering

What will we do today?

- Decision problems
- The complexity class P
- Nondeterministic Turing machines
- More complexity classes: EXP, NP, coNP
- Polynomial-time reductions
- NP-hardness and NP-completeness

Quadratic vs. Exponential

- Important difference between
  algorithms that run in time, say, n^2
  vs. algorithms that run in time, say, 2^n

- Illustration (time needed for 10^{10} steps per second):

n	n^2 steps	2^n steps
2	0.00000002 msec	0.00000002 msec
5	0.00000015 msec	0.00000019 msec
10	0.00001 msec	0.0001 msec
20	0.00004 msec	0.10 msec
50	0.00025 msec	31.3 hours
100	0.001 msec	9.4×10^{11} years
1000	0.100 msec	7.9×10^{282} years

- # of atoms in universe $\approx 10^{80}$

Decision problems

- To simplify the theory, we restrict our attention to yes/no questions

Definition (Decision problems)

A *decision problem* is a function $f : \Sigma^* \to \{0, 1\}$ where for each input $x \in \Sigma^*$ the correct output $f(x)$ is either 0 or 1.

Alternatively: a formal language $L \subseteq \Sigma^*$ where $x \in L$ if and only if $f(x) = 1$.

- For decision problems, we typically look at TMs that have two halting states:
  q_{acc} (for *accept*: $f(x) = 1$)
  and q_{rej} (for *reject*: $f(x) = 0$)

The complexity class P

Definition (polynomial-time computability)

A function $f : \Sigma^* \to \Sigma^*$ is *polynomial-time computable* (or *computable in polynomial time*) if there exist a TM $\mathbb{M}$ and a constant $c \in \mathbb{N}$ such that:

- $\mathbb{M}$ computes f
- $\mathbb{M}$ runs in time $O(|x|^c)$

Definition (the complexity class P)

P is the class (set) consisting of all decision problems $L \subseteq \Sigma^*$ that are computable in polynomial time.

(In)tractability

- Tractability: there exists a polynomial-time algorithm that solves the problem
- Intractability: there exists **no** polynomial-time algorithm that solves the problem

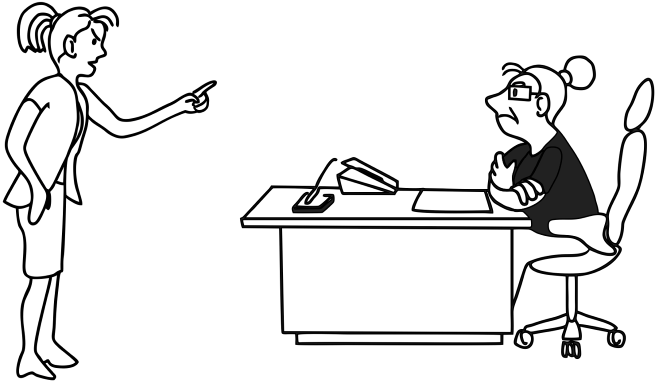
  (or sometimes phrased as: all algorithms that solve the problem take exponential time or more, in the worst case)

- How do we find out which of these two is the case?

Showing intractability: without any theory

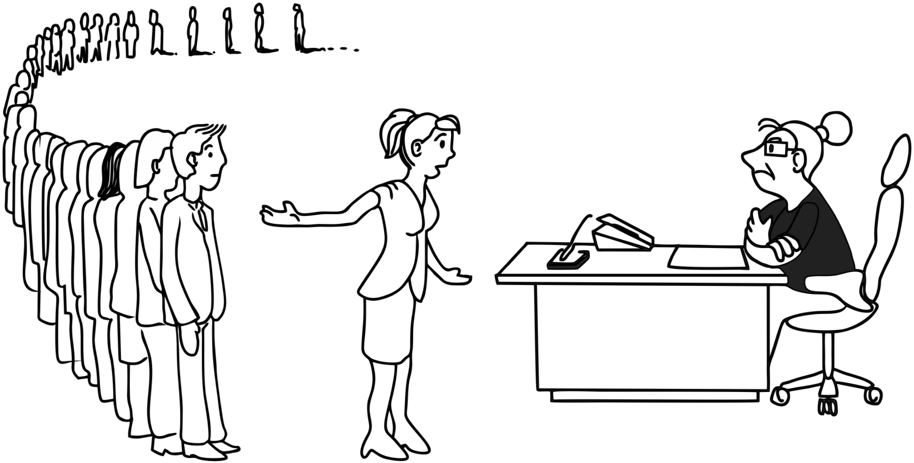
"I can't find an efficient algorithm, I guess I'm just too dumb."

Showing intractability: the ideal case



“I can’t find an efficient algorithm, because no such algorithm is possible!”

Showing intractability: using NP-completeness



“I can’t find an efficient algorithm, but neither can all these famous people.”

Polynomial vs. exponential time

Definition (DTIME)

Let $T : \mathbb{N} \to \mathbb{N}$ be a function. A language $L \subseteq \Sigma^*$ is in $\mathsf{DTIME}(T(n))$ if there exists a Turing machine that decides L and that runs in time $O(T(n))$.

Definition (the complexity classes P and EXP)

$$\mathsf{P} = \bigcup_{c \geq 1} \mathsf{DTIME}(n^c) \qquad \mathsf{EXP} = \bigcup_{c \geq 1} \mathsf{DTIME}(2^{n^c})$$

Definition (the complexity class NP)

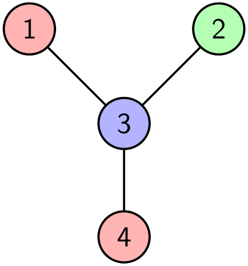
A problem $L \subseteq \Sigma^*$ is in the complexity class NP if there is a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time Turing machine $\mathbb{M}$ (the *verifier*) such that for every $x \in \Sigma^*$:

$$x \in L \quad \text{if and only if} \quad \text{there exists some } u \in \{0,1\}^{p(|x|)} \text{ such that } \mathbb{M}(x,u) = 1.$$

The string $u \in \{0,1\}^{p(|x|)}$ is called a *certificate* for x if $\mathbb{M}(x,u) = 1$.

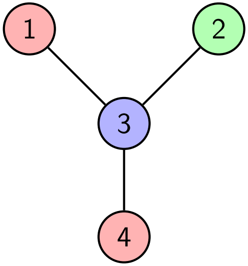
Example problem: 3-coloring

- You are given an undirected graph
- The task is to color each node with one of 3 colors so that the coloring is proper: **no two connected nodes have the same color**
- *Example application:* nodes are regions with their own radio station, colors are radio frequencies, and two nodes are connected if the regions border each other; assign radio frequencies without conflict



Example problem: 3-coloring (ct'd)

- Let's see why the (decision) problem of 3-coloring is in NP.
- Let $G = (V, E)$ be a graph with m nodes.
- Consider as witness a binary string u of length $2m$, where the coloring of each node i is given by the i'th pair of bits—
  say, 01 for red, 10 for green, and 11 for blue.
- Given G and u, we can check in polynomial time if the coloring given by u is *proper*.



$s = 01\ 10\ 11\ 01$

Nondeterministic Turing machines

Definition

A *nondeterministic Turing machines (NTM)* $\mathbb{M}$ is a variant of a (deterministic) Turing machine, where some things are modified.

- Instead of a single transition function δ, there are two transition functions δ_1, δ_2.
- At each step, one of δ_1, δ_2 is chosen nondeterministically to determine the next configuration.
- (As halting states, it has an accept state q_{acc} and a reject state q_{rej}.)

- We write $\mathbb{M}(x) = 1$ if there is some sequence of nondeterministic choices such that $\mathbb{M}$ reaches the state q_{acc} on input x.
- The machine $\mathbb{M}$ runs in time $T(n)$ if for every input x and every sequence of nondeterministic choices, $\mathbb{M}$ halts within $T(|x|)$ steps.

Nondeterministic polynomial time (NP)

Definition (NTIME)

Let $T : \mathbb{N} \to \mathbb{N}$ be a function. A problem $L \subseteq \Sigma^*$ is in $\mathsf{NTIME}(T(n))$ if there exists a nondeterministic Turing machine that decides L and that runs in time $O(T(n))$.

Proposition (characterization of NP)

$$\mathsf{NP} = \bigcup_{c \geq 1} \mathsf{NTIME}(n^c)$$

The complexity class coNP

Definition (the complexity class coNP)

A problem $L \subseteq \Sigma^*$ is in coNP if $\overline{L} \in \mathrm{NP}$, where $\overline{L} = \{ \, x \in \Sigma^* \mid x \notin L \, \}$.

Proposition (verifier characterization of coNP)

A problem $L \subseteq \Sigma^*$ is in coNP if there is a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time Turing machine $\mathbb{M}$ (the *verifier*) such that for every $x \in \Sigma^*$:

$$x \in L \quad \text{if and only if} \quad \textbf{for all } u \in \{0,1\}^{p(|x|)} \text{ it holds that } \mathbb{M}(x,u) = 1.$$

$\mathsf{NP} \subseteq \mathsf{EXP}$

Proposition

$\mathsf{NP} \subseteq \mathsf{EXP}$.

Proof (idea).

- Iterate over all possible witnesses $u \in \{0,1\}^{p(|x|)}$, and check if $\mathbb{M}(x,u) = 1$.
- If for any u this is the case, return 1—otherwise, return 0.
- There are $2^{p(|x|)}$ such strings u, and so this takes time $2^{p(|x|)} \cdot q(|x|)$, for some polynomial q.

$\square$

An overview of complexity classes

(That we've seen so far..)

$$
\begin{tikzcd}
& \mathsf{EXP} & \\
\mathsf{NP} \arrow[ur, "\subseteq"] & & \mathsf{coNP} \arrow[ul, "\subseteq"'] \\
& \mathsf{P} \arrow[ul, "\subseteq"] \arrow[ur, "\subseteq"'] &
\end{tikzcd}
$$

Polynomial-time reductions

Definition (polynomial-time reductions)

A problem $L_1 \subseteq \Sigma^*$ is *polynomial-time reducible* to a problem $L_2 \subseteq \Sigma^*$ if there is a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ (the *reduction*) such that for every $x \in \Sigma^*$ it holds that:

$$x \in L_1 \quad \text{if and only if} \quad f(x) \in L_2.$$

$$\begin{tikzcd}
x \arrow[r] & f(x) \arrow[d, squiggly] \\
x \in L_1? & f(x) \in L_2? \arrow[l]
\end{tikzcd}$$

- We write $L_1 \leq_{\mathrm{p}} L_2$ to indicate that L_1 is polynomial-time reducible to L_2.

NP-hardness and NP-completeness

Definition (NP-hardness)

A problem $L \subseteq \Sigma^*$ is NP-*hard* if every problem in NP is polynomial-time reducible to L.

Definition (NP-completeness)

A problem $L \subseteq \Sigma^*$ is NP-*complete* if $L \in \mathrm{NP}$ and L is NP-hard.

Some properties

Proposition

Polynomial-time reductions are transitive.
That is, if $L_1 \leq_{\mathrm{p}} L_2$ and $L_2 \leq_{\mathrm{p}} L_3$, then $L_1 \leq_{\mathrm{p}} L_3$.

Proposition

Take two problems $L_1, L_2 \subseteq \Sigma^*$. If L_1 is polynomial-time reducible to L_2 and $L_2 \in \mathrm{P}$, then $L_1 \in \mathrm{P}$.

Proposition

Take an NP-complete problem $L \subseteq \Sigma^*$. If $L \in \mathsf{P}$, then $\mathsf{P} = \mathsf{NP}$.
In other words, assuming that $\mathsf{P} \neq \mathsf{NP}$, $L \notin \mathsf{P}$.

Proof.

Since deterministic TMs can be seen also as nondeterministic TMs, we get $\mathsf{P} \subseteq \mathsf{NP}$.

We show that if $L \in \mathsf{P}$, then $\mathsf{NP} \subseteq \mathsf{P}$.

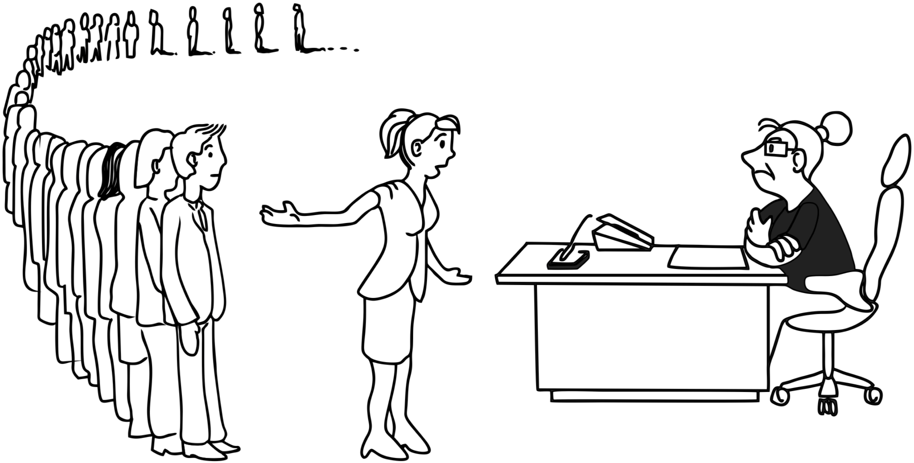
(1) Take an arbitrary problem $M \in \mathsf{NP}$.

(2) Since L is NP-complete, $M \leq_{\mathsf{p}} L$.

(3) Since $L \in \mathsf{P}$, then also $M \in \mathsf{P}$.

Since M was arbitrary, we know that $\mathsf{NP} \subseteq \mathsf{P}$. $\square$

Showing intractability: using NP-completeness



“I can’t find an efficient algorithm, but neither can all these famous people.”

Recap

- Decision problems
- The complexity class P
- Nondeterministic Turing machines
- More complexity classes: EXP, NP, coNP
- Polynomial-time reductions
- NP-hardness and NP-completeness

Next time

- Proving that NP-complete problems exist :-)