Computational Complexity

Lecture 2: Reductions, NP and NP-completeness

Ronald de Haan
me@ronalddehaan.eu
University of Amsterdam

April 5, 2023

Recap

What we saw last time..

- (Deterministic) Turing machines
- Decision problems
- Polynomial time and the class P

What will we do today?

- The universal Turing machine
- Nondeterministic Turing machines
- More complexity classes: EXP, NP, coNP
- Polynomial-time reductions
- NP-hardness and NP-completeness

Representing Turing machines as (binary) strings

- We can encode Turing machines into binary strings, such that:

  1. each string $s \in \{0,1\}^*$ represents some Turing machine $\mathbb{M}$

  2. each Turing machine $\mathbb{M}$ is represented by infinitely many strings $s \in \{0,1\}^*$

  3. given a TM $\mathbb{M}$, we can efficiently compute a string s that represents $\mathbb{M}$

- Idea:

  - Write out the tuple (Γ, Q, δ), together with starting and halting states, in an appropriate alphabet, and then encode into binary

  - Allow padding (cf. comments in programming languages)

Efficient universal Turing machine

Proposition

There exists a TM $\mathbb{U}$ such that for every $x, s \in \{0,1\}^*$ it holds that $\mathbb{U}(x,s) = \mathbb{M}_s(x)$, where $\mathbb{M}_s$ is the TM represented by the string s.

Moreover, if $\mathbb{M}_s$ halts on x in time T, then $\mathbb{U}(x,s)$ halts in time $C \cdot T \log T$, where C depends only on s (and not on x).

- $\mathbb{U}$ is an efficient universal Turing machine: it can simulate other TMs in an efficient way.

(In)tractability

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

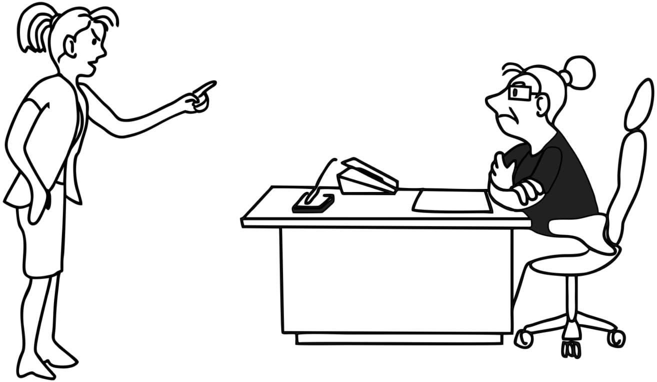
  (or sometimes: all algorithms that solve the problem take exponential time, in the worst case)

- How do we find out which of these two is the case for—for example—the problem of 3-coloring?

Showing intractability: without any theory

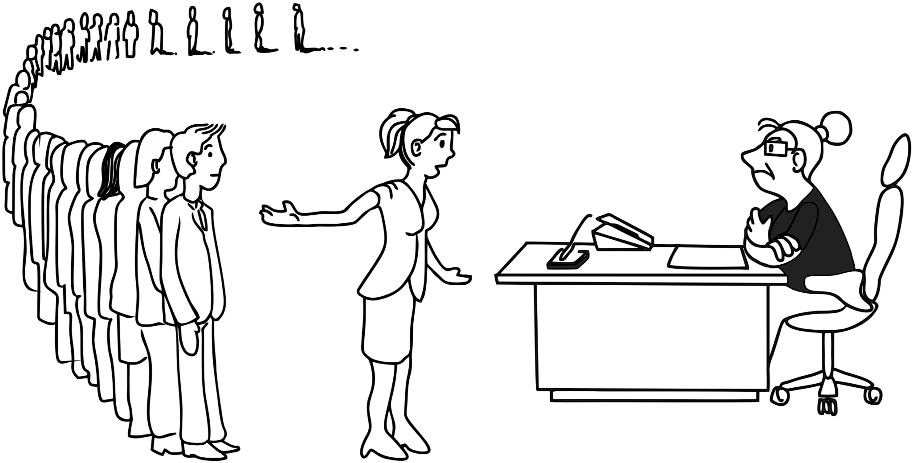
"I can't find an efficient algorithm, I guess I'm just too dumb."

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 $\mathrm{DTIME}(T(n))$ if there exists a Turing machine that decides L and that runs in time $O(T(n))$.

Definition (the complexity classes P and EXP)

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

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 $\mathbb{M}(x,u) = 1$.

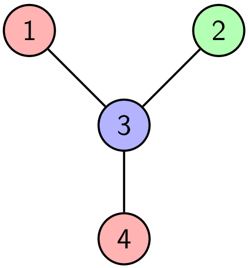
Example: 3-coloring

- 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)

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

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 $x \in \Sigma^*$:

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

NP $\subseteq$ EXP

Proposition

NP $\subseteq$ 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:

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

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

- 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 $L \in$ 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 $L_1 \in \mathrm{P}$.

Some properties (ct'd)

Proposition

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

Proof.

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

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

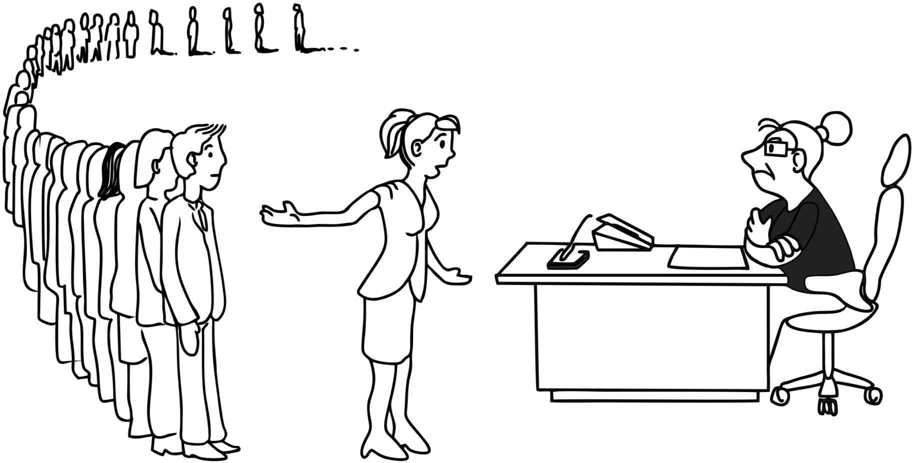
(1) Take an arbitrary problem $M \in \mathrm{NP}$.

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

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

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

Showing intractability: using NP-completeness



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

Recap

- The universal Turing machine
- 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 :-)