Computational Complexity 3: NP-Completeness and NP-Hardness

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong

Rationales behind NP-Hardness

We have defined earlier the classes P and NP:

- P is the set of decision problems that can be **solved** efficiently;
- NP is the set of decision problems that can be **verified** efficiently.

Recall, however, that a main objective of the NP-hardness theory is to argue that some decision problems **could** not be solvable in polynomial time. The keyword "could" — as you can feel — implies that we **are not absolutely sure**.

What a conundrum! How can we "argue for" the unlikely presence of polynomial-time algorithms when we are "unsure"?

Rationales behind NP-Hardness (cont.)

Let us resolve the conundrum.

We will define a set of NP problems — called the **NP-complete class** — that have been studied by human beings for **several decades**, but no polynomial time solutions are known (examples: 3-SAT, clique decision, vertex cover decision, set cover decision, etc.)! And yet, if we manage to find a polynomial time algorithm for **any** of those problems, then suddenly **all** those problems can be solved in polynomial time!

So there are two possibilities:

1. There exists a mysterious algorithm that has escaped the scrutiny of **every one on earth** that has ever worked on those problems.
2. Or maybe such an algorithm does not exist at all, meaning that **none** of those problems can be solved in polynomial time.

> Many people believe it is the second possibility that is true.

The **NP-Complete class** — denoted as NPC — is the set of decision problems π such that

- π is in NP;
- if π can be **solved** in polynomial time, then **every** problem in NP can be **solved** in polynomial time.

Every problem in NPC is said to be **NP-complete**.

In other words, NPC includes the most difficult decision problems in NP.

If a decision problem π satisfies the second bullet (but not necessarily the first), π is said to be **NP-hard**.

Reduction

Let π_1 and π_2 be two decision problems.
Next, we will learn a technique to prove a claim the following type:

> If we can solve π_1 in polynomial time, then we can solve π_2 in polynomial time.

More specifically, suppose that you are given an algorithm $\mathcal{A}_1$ solving π_1 in polynomial time, how can you use $\mathcal{A}_1$ to solve π_2 in polynomial time?

Reduction

Answer: Convert problem $\pi_2 = (L_2, \Pi_2)$ to problem $\pi_1 = (L_1, \Pi_1)$, and then solve the latter using $\mathcal{A}_1$.

Let σ_2 be an input sequence of π_2.
Generate an input sequence σ_1 for π_1 in polynomial time such that $\Pi_2(\sigma_2)$ can be inferred from $\Pi_1(\sigma_1)$.
Use $\mathcal{A}_1$ to obtain $\Pi_1(\sigma_1)$ in polynomial time.

Overall, we thus have obtained an algorithm solving π_2 in polynomial time.

In general, if a problem π_2 can be solved using an algorithm $\mathcal{A}_1$ for another problem π_1, we say that π_2 can be **reduced to** π_1, and refer to the conversion as a **reduction**.
If the whole reduction takes polynomial time, we denote the fact using $\pi_2 \leq_P \pi_1$.

Recall:

The **NP-Complete class** — denoted as NPC — is the set of decision problems π such that

- π is in NP;
- if π can be **solved** in polynomial time, then **every** problem in NP can be **solved** in polynomial time.

If π is an NPC problem, it means that **every** other problem π' in NP can be reduced to π!

What a difficult claim to prove!
You must consider every possible π', but NP has an infinite number of problems!

Interestingly, once the first NPC problem has been found, it is much easier to prove the second.

Theorem: Let π^* be a decision problem in NPC. If π^* can be reduced to another decision problem π in polynomial time, then π must be NP-hard.
Hence, if π is also in NP, π is NP-complete.

Proof: Let π' be any problem in NP. It holds that

$$\pi' \le_P \pi^* \le_P \pi$$

which means that π' can be reduced to π. $\square$

The "first" `NPC` problem is excessively technical for our course. To apply the theorem on the previous page, however, it suffices to use **any** `NPC` problem π^*. One `NPC` problem commonly used to prove NP-hardness is 3-SAT.

Recall:

3-SAT

Variable: a boolean unknown x that can be assigned 0 or 1.
Literal: a variable x or its negation $\bar{x}$.
Clause: the OR of up to 3 literals.
Formula: the AND of clauses

The 3-SAT problem: Is there an assignment to the variables under which the formula evaluates to 1?

Example:

$$(x_1 \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_2} \vee x_3 \vee x_4) \wedge (\bar{x_1} \vee \bar{x_4})$$

The answer is "yes". A certificate: $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 0$.

$$(x_1) \wedge (\bar{x_1} \vee x_2) \wedge (\bar{x_2})$$

The answer is "no".

Theorem: 3-SAT is NP-complete.

The proof is beyond the scope of the course and omitted.

We will use 3-SAT to prove the NP-completeness of the clique decision problem in the next lecture.