Computational Complexity 2: The NP (Non-Deterministic Polynomial) Class

Yufei Tao

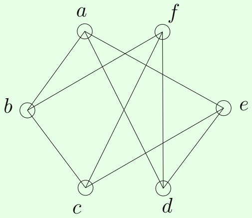
Department of Computer Science and Engineering
Chinese University of Hong Kong

In the last lecture, we have already defined the class P of decision problems. Today we will define the class NP which is a **superset** of P. Intuitively, NP includes a set of decision problems where it is **easy to accept or reject** a proposed solution.

Before formalizing this concept, let us look at a concrete example.

The Clique Decision Problem: Let $G = (V, E)$ be an undirected graph. Given an integer k, decide whether we can find a set S of at least k vertices in V that are mutually connected (i.e., there is an edge between any two vertices in S).

Example: Consider



The answer is "yes" for $k \leq 3$, but "no" for $k \geq 4$.

No polynomial time algorithm is known for the clique decision problem! This means no algorithms are known to solve this problem in time polynomial to $|V|, |E|$ and k. In other words, human beings currently **do not** know if the problem is in $\mathtt{P}$.

However, if someone proposes a candidate solution S (with size k) to us, we can easily decide whether the vertices in S are mutually connected in $O(k^2 \cdot |E|)$, which is polynomial to $|V|, |E|$, and k. If so, S is called a **certificate**, because it serves as evidence that we should return 1.

Therefore, the problem is in $\mathtt{NP}$.

We now start to formalize the problem class $\mathtt{NP}$.

Fix a decision problem (L, Π).
Define L_1 as the set of input sequences $\sigma \in L$ such that $\Pi(L) = 1$.
Define L_0 as the set of input sequences $\sigma \in L$ such that $\Pi(L) = 0$.

Now consider an input sequence $\sigma \in L$.
Let N be the bit-length of σ.

Let ϕ be any input sequence whose bit-length is **a polynomial of** N. Denote by $\sigma : \phi$ the input sequence obtained by concatenating σ and ϕ. We will refer to $\sigma : \phi$ a **polynomial extension** of σ.

The input sequence $\sigma : \phi$ then forms the "real input" for an algorithm to process.

We say that an algorithm $\mathcal{A}$ can **verify in polynomial time** a decision problem (L, Π) if both of the following hold:

- For any input sequence $\sigma \in L_1$ with bit-length N, $\mathcal{A}$ returns 1 on **at least one** polynomial extension $\sigma : \phi$ of σ in time polynomial to N.
    - In this case, ϕ is called a **certificate**.
- For any input sequence $\sigma \in L_0$ with bit-length N, $\mathcal{A}$ returns 0 on **every** polynomial extension $\sigma : \phi$ of σ in time polynomial to N.

> NP is the set of decision problems that can be verified by an algorithm in polynomial time.

Example: Clique Decision Problem

Consider the clique decision problem with graph $G = (V, E)$ and integer k as the input. Denote by σ the input sequence that encodes G and k. We will assume that the bit-length N of σ satisfies $N = \Omega(|E|)$ and $N = \Omega(|V|)$.

We now give an algorithm $\mathcal{A}$ that can verify the problem in polynomial time. For any polynomial extension $\sigma : \phi$, we require ϕ to be a sequence of k distinct integers in $[1, |V|]$, corresponding to k vertices in V. If ϕ violates this condition, $\mathcal{A}$ returns 0 immediately. Otherwise, $\mathcal{A}$ checks whether the k vertices are mutually connected, and returns 1 or 0 accordingly.

If the answer to the problem is "yes", there is a set S of k mutually connected vertices. Clearly our algorithm can verify that S is indeed a certificate in $O(k^2 \cdot |E|) = O(|V|^2|E|) = O(N^3)$ time.

If the answer to the problem is "no", our algorithm always returns 0.

$\mathtt{NP}$ stands for **non-deterministic polynomial**.

Intuitively, you can think of a problem (L, Π) in $\mathtt{NP}$ as being "solvable" in polynomial **parallel** time as follows. Create a **very large** (often **exponential** in the bit-length) number of parallel threads, each of which works on a different polynomial extension.

As long as one thread returns 1, we know that the problem should have an output of 1. If no thread returns 1 after some polynomial time, we return 0.

> Most of the decision problems that have ever been studied in computer science are in NP.

Next we will give more examples of NP problems.

Lemma: Every decision problem in $\texttt{P}$ is in $\texttt{NP}$.

The proof is simple and left to you as an exercise.

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:

$$(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee x_3 \vee x_4) \wedge (\bar{x}_1 \vee \bar{x}_4)$$

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".

3-SAT

No polynomial-time algorithms are known for the 3-SAT problem. This means that no algorithm can solve the problem in time polynomial to the number n of variables and to the number m of clauses.

Hence, human beings do not know whether the problem is in $\mathtt{P}$.

The problem is clearly in $\mathtt{NP}$ (**think:** why?).

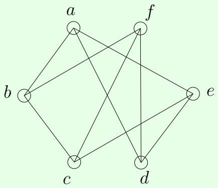
Vertex Cover

Consider an undirected graph $G = (V, E)$.

Consider a subset $S \subseteq V$.
S is a **vertex cover** if every edge $\{u, v\} \in E$ is adjacent to at least one vertex in S, i.e., $u \in S$, $v \in S$, or both.

The vertex cover decision problem: Given an integer k, decide whether there is a vertex cover with at most k vertices.

Example:



The answer is "no" for $k \leq 3$, but "yes" for $k \geq 4$.

Vertex Cover

No polynomial-time algorithms are known for the vertex cover decision problem. This means that no algorithm can solve the problem in time polynomial to $|V|$, $|E|$, and k.

Hence, human beings do not know whether the problem is in $\mathtt{P}$.

The problem is clearly in $\mathtt{NP}$ (**think:** why?).

Set Cover

Consider any set U, called the **universe**.
We are given n subsets of U: $S_1, S_2, ..., S_n$.

The set cover decision problem: Given an integer k, decide whether we can find k subsets from $\{S_1, S_2, ..., S_n\}$ such that the union of the k subsets is U.

Example: Consider $U = \{1, 2, 3, 4, 5, 6, 7, 8\}$
$S_1 = \{1, 2, 3, 4\}$
$S_2 = \{2, 5, 7\}$
$S_3 = \{6, 7\}$
$S_4 = \{1, 8\}$
$S_5 = \{1, 2, 3, 8\}$
The answer is "no" for $k \leq 3$ but "yes" for $k \geq 4$.

Set Cover

No polynomial-time algorithms are known for the set cover decision problem. This means that no algorithm can solve the problem in time polynomial to $m = \sum_{i=1}^{n} |S_i|$.

Hence, human beings do not know whether the problem is in $\mathtt{P}$.

The problem is clearly in $\mathtt{NP}$ (**think:** why?).

Rule of thumb: Ask yourself — can someone give you a certificate of a polynomial size (i.e., polynomial to all the problem parameters) that allows you to decide that the output should be 1 in polynomial time? If so, the problem is (almost for sure) in NP.

Let us end today's lecture by throwing out a difficult question:

$\texttt{P} = \texttt{NP}$?

This is one of the biggest open problems in computer science today.