P vs. NP

Jason Zhang

VJZ

December 25, 2025

Outline

Motivation

Named one of the most important problems of the 21st century, P vs. NP is a the question of whether two computational classes are equivalent.
The resolution to this problem will be useful regardless if P = NP.

Turing Machines

We assume the audience has a basic understanding of what a Turing Machine is. Even if not, any modern computational system (e.g., Python) suffices due to the *(Extended) Church-Turing Thesis*.

Definition (Deciders).

We say that a machine M *decides* the language $L \subseteq \{0,1\}^*$ if for all $x \in \{0,1\}^*$,

$$x \in L \iff M \text{ accepts } x.$$

Time Complexity

Let us review O time complexity, as the computational classes we are interested in deal with worst-case O time complexity.

Definition (O).

We say that a function $f(n)$ is $O(g(n))$ if and only if there exists constants c, n_0 such that for all $n \geq n_0$, $c \cdot g(n) \geq |f(n)|$.

Equivalently,

Definition.

$f(n)$ is $O(g(n))$ iff

$$\lim_{n\to\infty} \sup \frac{|f(n)|}{g(n)} < \infty.$$

Polynomial Time

We say a language L is in P if and only if there exists a polynomial p such that M accepts/rejects x in $O(p(|x|))$ time.

Non-Deterministic Polynomial Time

We have two equivalent definitions:

1. A non-deterministic Turing Machine accepts/rejects in polynomial time.
2. A machine can verify a witness string w in polynomial time.

$P \subseteq NP$

This result ($P \subseteq NP$) is quite trivial; We can make the verifier M to completely ignore the witness string w provided and then try to come up with its own.

Oracles

An *oracle Turing Machine* is one which has access to an oracle for a certain language L. At any time, the Turing Machine can query the oracle for a string x and the oracle responds with whether $x \in L$ instantly.
We write M^L for a Turing Machine with access to an oracle that decides L. We will write P^L to denote the set of languages that can be solved by M^L in polynomial time (similar definition for NP^L).

NP-Hard and Complete

Definition (NP-Hard).

We say that a language L is NP-hard iff $\mathrm{NP} \subseteq \mathrm{P}^L$.

A language L is NP-complete iff it is NP-hard and NP.

Finding Witnesses

Theorem.

If P = NP, then there exists a polynomial time algorithm to find the witness of any NP problem.

Finding Witnesses (Proof)

Proof.

Suppose $M(x, w)$ is the decider where x is the string and w is the witness that verifies x. Let $w = w_1 w_2 \cdots w_{p(n)}$. We will construct our decider M' as follows:

1. Does there exist w such that $M(x, w)$ accepts and $w_1 = 0$?
2. If yes, then ask if there exists w where $w_1 = 0$ and $w_2 = 0$.
3. If no, then ask if there exists w where $w_1 = 1$ and $w_2 = 0$.
4. Repeat.

Each question asked is an NP decision problem, and since P = NP, this procedure is P. You can also think of this as binary searching for the witness.

Introduction

There are three *barriers* to resolving $P \stackrel{?}{=} NP$. Researchers have spent time proving why showing either $P = NP$ or $P \neq NP$ is difficult.

1. Relativization Barrier
2. Natural Proofs Barrier
3. Algebrization Barrier

We will only cover the Relativization Barrier in depth.

PSPACE

Definition.

PSPACE is the set of languages L decidable by a Turing Machine that uses a polynomial amount of space. There is no time restriction.

It shouldn't be hard to see that P $\subseteq$ NP $\subseteq$ PSPACE (consider that in t steps, an algorithm can only use at most t cells).

It is currently open whether P $\stackrel{?}{=}$ PSPACE.

Barrier

Theorem (Baker-Gill-Solovay).

There exists oracles A and B such that $\mathrm{P}^A = \mathrm{NP}^A$ and $\mathrm{P}^B \neq \mathrm{NP}^B$.

The reason why this is a "barrier" is because that this theorem implies that any correct proof about P vs. NP must not apply to any oraclized version P^A vs. NP^A. This eliminates various classical proof techniques such as diagonalization.

Proof Sketch

Let us find the A oracle first.

Claim.

Let A be the oracle given by some PSPACE-complete language. Then, $\mathrm{P}^A = \mathrm{NP}^A = \mathrm{PSPACE}$.

Proof.

Definitionally, $\mathrm{PSPACE} \subseteq \mathrm{P}^A$. But also, $A \in \mathrm{PSPACE}$ and thus $\mathrm{P}^A \subseteq \mathrm{PSPACE}$ so $\mathrm{P}^A = \mathrm{PSPACE}$.
Trivially, $\mathrm{PSPACE} \subseteq \mathrm{NP}^A$. Suppose we DFS on the NTM's computation tree. This takes polynomial space, and answering A is also polynomial space. So, $\mathrm{NP}^A \subseteq \mathrm{PSPACE}$ and $\mathrm{NP}^A = \mathrm{PSPACE} = \mathrm{P}^A$.

Proof Sketch (Part 2)

Now, we will try to find B. Define

$$L_B = \{1^n : \exists x \in B(|x| = n)\}.$$

for any B.

Claim 1.

For any B, $L_B \in \mathrm{NP}^B$.

Claim 2.

There exists a B where $L_B \notin \mathrm{P}^B$.

These two facts, combined, show that there exists a B where $\mathrm{P}^B \neq \mathrm{NP}^B$.

Proof of Claim 1

For any input string 1^n, we can "guess" any string x of length n and query B in $O(1)$. The "guessing" is actually using the NTM, so this occurs in NP^B.

Proof of Claim 2

Let us first consider what such a proof would look like intuitively. Essentially, we need to find a B where any DTM must query at least 2^n times, which is not possible in P^B.

Proof Sketch.

The idea is to use diagonalization. Notice that we can enumerate all DTMs with oracles in polynomial time in the fashion $M_1, M_2, \ldots$. We start with $B_0 = \varnothing$ and then iteratively add strings to make M_i^B fail to decide B_i. Then, we take $B = \bigcup_i B_i$ which is sufficient. The exact details of the construction are beyond the scope for this lecture, but it involves essentially looking at each previous B_{i-1} and M_i^B and adding any string that wasn't previously queried (an adversarial argument of sorts).

This completes the proof of the theorem.

Consequences

If I wrote a proof that $\mathrm{P} = \mathrm{NP}$, but it also proved that $\mathrm{P}^A = \mathrm{NP}^A$ for all oracles A, then I automatically know it is wrong.

Natural Proofs

A very broad class of proofs that are ineffective against the P vs. NP problem. Informally, natural proofs are lower bound proofs that give rise to certain algorithms operating on boolean truth tables. They often rely on combinatorial techniques and pseudo-random functions. Unfortunately, the natural proof is self-defeating as it would yield an efficient algorithm for a problem we were trying to prove to be hard.

The Crux of the Problem

Aaronson phrased it best:

Quote (Aaronson).

"… we're trying to prove that certain functions are hard, but the problem of deciding whether a function is hard is itself hard, according to the very sorts of conjectures that we're trying to prove."

Algebrization Barrier

This requires some knowledge from abstract algebra. Informally, this barrier can be seen as a generalization of Relativization. The details are far too technical for this lecture, but it goes something like this:

1. For an oracle O, let $\tilde{O}$ denote its extension for some finite field $\mathbb{F}$ such that $\tilde{O}$ is a collection of polynomials $\tilde{O}_{n,\mathbb{F}} : \mathbb{F}^n \to \mathbb{F}$ over $n \in \mathbb{N}$ and all finite fields satisfying certain conditions.
2. If C and D are complexity classes, then $\mathrm{C} \subseteq \mathrm{D}$ algebrizes if for all oracles O and their extensions $\tilde{O}$, $\mathrm{C}^{O} \subseteq \mathrm{D}^{\tilde{O}}$.
3. $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ does not algebrize.

The Modern Approach

In the modern era, three main approaches are being seen in order to prove $P \stackrel{?}{=} NP$:

1. Ironic Complexity Theory
2. Arithmetical Complexity Theory
3. Geometric Complexity Theory (most promising)

Each one of these have made massive strides in a variety of different fields. Thus, even if $P \neq NP$ as most experts believe, the process of solving this problem is still valuable to the general mathematical and computer science community.

References

The contents of this lecture were adapted from the following sources:

1. P $\stackrel{?}{=}$ NP by Scott Aaronson
2. Lecture 8: Relativizations. Baker-Gill-Solovay Theorem by Jin-Yi Cai. Recorded down by: Matthew Lee, Yingchao Liu, Uchechukwu Okpara
3. Barriers in Complexity Theory by Arthur Vale