What are the main theoretical barriers to resolving P vs NP?

An Extended Project Qualification Dissertation

Camron Short

Candidate Number: 6439
Centre Number: 19216

Defining the problem

Big O-Notation

When shopping, there are multiple ways to collect all the items on your list. You might start at one end of the store and work through it methodically, which is often the most efficient - but only if your list is already sorted by store layout. Without planning, you might find yourself doubling back or criss-crossing the store multiple times. These differences in method lead to drastically different total effort. Similarly, the time taken depends not just on the method, but also on who is doing the shopping and how many items are on the list. These human and environmental factors make it impossible to measure an algorithm's speed purely in seconds. To address this, computer scientists use **Big O notation**, a mathematical way to describe how the number of steps an algorithm takes grows with the size of the input. For example, an algorithm with complexity $O(n)$ will perform at most a number of operations proportional to the length of the input list (n). An algorithm with $O(n^2)$ complexity might check each item against every other, leading to significantly more steps. These expressions describe the algorithm's **asymptotic behaviour** - that is, how it scales as inputs grow very large. When this growth follows a polynomial pattern like $O(n)$ or $O(n^2)$, we say it runs in **polynomial time**, which is generally considered efficient and tractable.

Sets and Set Notation

Earlier, we used a shopping list to explore algorithmic strategies. Mathematically, that list can be viewed as a **set** - a well-defined collection of distinct objects, such as items to buy. In computer science and mathematics, Sets are fundamental. A set can contain numbers, items, states, algorithms, or even other sets. For example, the set of paths through a store might include every possible route you could take. The set of solutions to a problem includes all algorithms that solve it. We describe relationships between sets and their elements using **Set notation**:

- $\in$: "is an element of" (e.g., $3 \in \{1,2,3\}$ means 3 is in the set)
- $\notin$: "is not an element of" (e.g., $4 \notin \{1,2,3\}$)
- $\subseteq$: "is a subset of" (e.g., $\{1,2\} \subseteq \{1,2,3\}$)
- $\cup$: union (e.g., $A \cup B$ contains all elements in A or B)
- $\cap$: intersection (e.g., $A \cap B$ contains only elements in both A and B)
- $\setminus$: set difference (e.g., $A \setminus B$ contains elements in A but not in B)

P

The first set in our investigation is called **P**. This set contains all problems that can be solved by an algorithm in **polynomial time** - meaning the number of steps required grows at most like n, n^2, n^3, etc., where n is the size of the input. A good example is the weekly shop. Planning the optimal route through the store may take time, but it is feasible to compute in polynomial time - for example, with an algorithm of complexity $O(n^2)$.

NP

The (potentially larger) set **NP** stands for **nondeterministic polynomial time**. This set contains all problems where a proposed solution can be *verified* in polynomial time, even if finding that solution might be difficult. Consider Sudoku: finding a solution from scratch may be hard, but checking whether a filled grid obeys the rules is quick. Similarly, navigating a store inefficiently still allows you to verify whether you've bought all items - even if the path wasn't optimal. We know that $P \subseteq NP$: any problem we can solve efficiently, we can also verify efficiently. Whether or not these sets are actually equal is the essence of our central question.

Our Question

This brings us to our guiding problem: **What is the relationship between P and NP?** Mathematically, we know that P is a subset of NP, but we do not know whether $P = NP$. That is, we don't know whether every efficiently verifiable problem can also be efficiently solved. Some mathematicians believe $\mathbf{P} = \mathbf{NP}$ - implying that every hard-looking problem has a hidden efficient solution - while others believe $\mathbf{P} \neq \mathbf{NP}$, suggesting some problems are inherently hard to solve even if easy to check. This project does not aim to resolve the question. Instead, we will explore *why* it has remained unanswered for decades - and examine the theoretical barriers that prevent us from settling it.

Relativization

Introduction to Relativization

One of the most important—and possibly oldest—barriers to resolving *P vs NP* is known as *relativization.* This idea was first formalized by three mathematicians in 1975, showing that a large group of mathematical proof methods face a fundamental limitation. This limitation lies in their inability to separate the complexity classes discussed earlier when introducing *Big O Notation.* At its core, relativization involves measuring how these complexity classes behave when given access to an *Oracle* (explored next), and how an algorithm's reasoning encounters a limit when this additional power is introduced.

Oracles

As defined by Arora and Barak Arora and Barak [2009]:
Much like the legendary Oracle of Delphi, an *Oracle* in complexity theory can provide an instant solution to a given subproblem. It is an abstract tool—considered a 'black box'—that never reveals how it reaches its answer, only that it does. We refer to any *Turing Machine* with access to such a tool as an *Oracle Turing Machine*, which may query the oracle at any point during computation. A relatable example might be solving a Sudoku puzzle using a hint: the app does not explain why the hint is correct, but you trust it. In that moment, using this information to guide your solution makes you behave almost like an *Oracle Turing Machine.* We denote this formally as $\mathbf{P^A}$ or $\mathbf{NP^A}$, meaning class $\mathbf{P}$ (or $\mathbf{NP}$) relative to oracle A.

How does it apply to a resolution?

A proof technique is said to *relativize* if the logic of the proof still holds when both complexity classes are given access to the same oracle. However, relativization as a method cannot resolve questions like $\mathbf{P} = \mathbf{NP}$, as shown by Baker, Gill, and Solovay Baker et al. [1975].
They constructed two oracles, A and B, such that:

- $\mathbf{P^A} = \mathbf{NP^A}$ (i.e., relative to oracle A, $\mathbf{P}$ and $\mathbf{NP}$ are equal)
- $\mathbf{P^B} \neq \mathbf{NP^B}$ (i.e., relative to oracle B, they are distinct)

This demonstrates that any proof method which relativizes will yield inconsistent results depending on the oracle. Consequently, such techniques cannot be used to resolve *P vs NP*, because they fail in certain relativized scenarios. This eliminates a large class of approaches and shows that any successful proof must go beyond relativization.

Reflection

As Aaronson Aaronson [2005] argues, relativization reflects a deeper philosophical boundary. Just as some truths lie beyond formal proof, some complexity class separations may lie beyond the reach of techniques that relativize.

Bibliography

Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. Relativization.

Theodore Baker, John Gill, and Robert Solovay. Relativizations of the p=?np question. *SIAM Journal on Computing*, 4(4):431–442, 1975. URL `https://www.scribd.com/document/688253900/Relativizations-of-the-P-NP-Question-Orig` Relativization.

Scott Aaronson. Why philosophers should care about computational complexity, 2005. URL `https://www.scottaaronson.com/papers/philos.pdf`. Relativization.