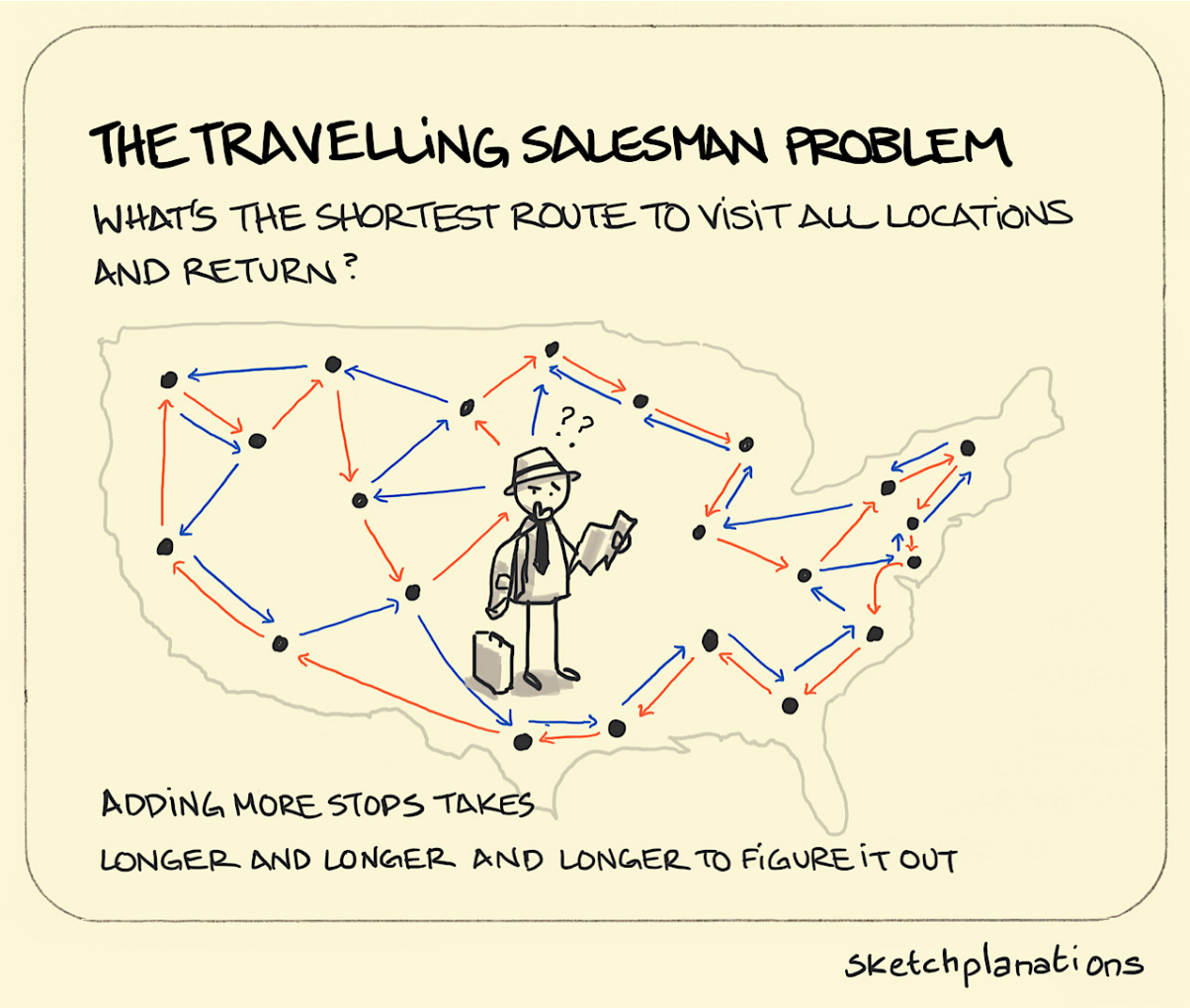
What are the main theoretical barriers to resolving P vs NP?

An Extended Project Qualification Dissertation

Camron Short
Candidate Number: 6439
Centre Number: 19216



Defining the problem

Introduction To Our Case Study

Throughout this paper I will be making references to a theoretical Salesman. The details of what they are selling is irrelevant for our topic so imagine that detail at your own discretion. Our Salesman has one goal, visit every location at least once on a map and return home in the shortest possible time. Finding the correct path varies upon the layout of locations (towns) to visit and how many of these exist.

A graph (map) with trivial result can be seen here:

INSERT TRIVIAL CASE OF TSP

Comparatively to this non-trivial result:

INSERT MEAN CASE OF TSP

Big O-Notation

Our Salesman can go to many different towns in whichever order they wish. The order however will impact the speed at which they can complete the task. This is the rationale behind finding the shortest path. Think back to our trivial case for the salesman from their introduction. Now instead of traversing across the line we loop back repeatedly, using more steps than neccesary. Whilst this will still allow the salesman to complete their rounds they will have missed dinner and be very upset. Similarly, the time taken depends not just on the method, but also on who the Salesman is and if they have a faster van than a different one. 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 salesman traversing across a map to explore algorithmic strategies. Mathematically, the list of towns can be viewed as a **set** - a well-defined collection of distinct objects. 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 include every possible route our salesman 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 back to our salesman: For non-trivial cases of the problem finding the optimal path is $O(n^2 2^n)$. Meaning right now we cannot solve it in polynomial time, but it can be verified in such time. 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 [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 instance could be whilst driving you use a GPS, this tool does not show you how it gets it's result however you will tend to 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 $\mathbf{P}^{\mathbf{A}}$ or $\mathbf{NP}^{\mathbf{A}}$, 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:

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

Natural Proofs

Introduction to Natural Proofs

Simply put, a *Natural Proof* is a method that works on many different functions and is easy to apply. A property $\mathcal{P}$ is said to be **natural** if:

- **Constructivity**: Given the truth table of a Boolean function f, we can determine whether $f \in \mathcal{P}$ in polynomial time.
- **Largeness**: A non-negligible portion of all Boolean functions satisfy $\mathcal{P}$.

This is best described by analogy. Imagine you're an English teacher - a strict one - who sets an essay for homework every lesson. You begin to suspect several students are using AI to write their essays. To catch them, you develop an algorithm that detects AI-generated writing. Maybe it looks for excessive use of em-dashes or unnatural phrasing. This algorithm is simple and can be applied to *every* essay you mark. However, once students realise how it works, they adapt their writing style to evade detection. This captures the flaw with Constructivity: if your method is too effective and too public, it can be countered. Largeness suffers a related issue. If your detection method works on too *many* types of cheating - copied essays, paraphrasing, ChatGPT, etc. - then you're likely catching even legitimate work, or again being too general to be reliable.

We care whether a proof is *natural* because many known lower-bound techniques are. This becomes especially relevant when we're trying to prove the *lower bound* of a function f — that is, the fastest possible way to compute f. If we can do this, we may be able to show that $f \notin \mathbf{P}$.

Circuit Complexity

Before diving deeper into the limitations of Natural Proofs, we must take a step back and understand one of the main battlegrounds for proving $\mathbf{P} \neq \mathbf{NP}$ - **circuit complexity**. Refer to our Salesman. Suppose instead of solving each map manually, they could build a dedicated machine for each map size - a custom calculator of sorts made entirely of logic gates (AND, OR, NOT). This machine would take a list of towns and output the shortest route. In computational terms, this machine is a **Boolean circuit**. Circuits don't run sequentially like an algorithm. They are a set arrengement of gates that perform a binary operation. Each value of n gets a seperate circtuit. We intend to keep these circtuits small. If we can prove that no small circuit can satisfy a problem we have proven that the problem does not lie in P. In doing so proving $\mathbf{P} \neq \mathbf{NP}$. **Formally:** The *circuit complexity* $C(f)$ of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is the size of the smallest Boolean circuit that computes f correctly on all inputs. Referring back to our Salesman, if th4 smallest circuit they can build for the TSP grows faster than any polynomial in n, this proves that we need *super-polynomial* resources. This would strongly suggest $\mathbf{P} \neq \mathbf{NP}$. Despite decades of mathemeticians attempting to prove the lower bounds of problems in **NP**, it has proven difficult. Many of the proof techniques used are considered *Natural*.

"If we can prove that no small circuit solves a problem, we show that even the smartest shortcut-the most efficient mental 'machine'—won't help our Salesman beat the clock."
- Inspired by Fortnow [2009]

Pseudorandom Functions

Pseudorandom Functions (PRFs) are a core idea in cryptography. They are functions that look random, but are actually completely deterministic. To anyone without the secret key, the outputs seem unpredictable - even though they're generated by a fixed rule. This idea is essential for secure communication. If you can't tell the difference between a random number and one generated by a PRF, then you also can't reverse-engineer passwords, decode messages, or guess encryption keys. Here's the link to Natural Proofs: If you had a proof method that could tell whether a function was "hard" (i.e., had no small circuit), and that method was both constructive and large - then it could also be used to tell **whether a function was pseudorandom**. This would be catastrophic. The ability to tell whether something is pseudorandom is exactly what modern cryptography tries to prevent. So, if PRFs exist - and almost every encryption system assumes they do - then Natural Proofs can't work. They're too strong. They don't just solve $P \neq NP$ - they break the internet along the way.

Razborov Rudich Insight

In 1994, Alexander Razborov and Steven Rudich published a result that fundamentally changed how we think about proving lower bounds Razborov and Rudich [1997]. They showed that if secure pseudorandom functions exist - something almost all of cryptography relies on - then Natural Proofs cannot be used to separate **P** from **NP**. This was a shock. Many earlier techniques for proving circuit lower bounds turned out to be *natural* - they were efficient to check and applied to a wide range of functions. But Razborov and Rudich proved that any such natural method could be turned against cryptography: it would allow you to distinguish a pseudorandom function from a truly random one. That's not just proving $\mathbf{P} \neq \mathbf{NP}$. That's breaking encryption. Since we assume pseudorandom functions exist, it must mean that **natural proofs can't work** - not because they're weak, but because they're too powerful. As summarised in Arora and Barak [2009], if a proof method is constructive and large, and still manages to prove a circuit lower bound for functions in **NP**, it must also contradict the existence of pseudorandom functions. This means that the search for new lower-bound techniques must avoid being *natural* in this precise technical sense - a significant restriction.

An Analogy

Imagine you're running airport security. You build a fast, effective scanner that flags any suspicious-looking bag based on known patterns - *density, shape, wiring.* It's fast enough to run on every passenger and catches most known threats. This is your *constructive* and *large* method. But what if attackers knew exactly how your scanner worked? They could design luggage that looks perfectly normal to your algorithm - even if it's dangerous. Now, imagine someone builds a new scanner that works so well it catches even these disguised threats. You get excited: maybe this new system will finally make flying totally safe.

But then someone points out a terrifying implication: *If this scanner works as described, it would also catch military stealth tech. Or diplomatic pouches. Or even encrypted test samples from trusted researchers.* In short - the scanner is *too good*. If it works, it breaks things it wasn't meant to break. That's the heart of the Razborov and Rudich [1997] result. If your method is constructive (efficient) and large (applies broadly), then it's strong enough to detect pseudorandom functions - which modern cryptography says you shouldn't be able to do -Arora and Barak [2009]. So unless we want to throw out the internet, these proof methods can't work.

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-Original`. Relativization.

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

Lance Fortnow. The status of the p versus np problem. *Communications of the ACM*, 52(9):78–86, 2009. URL `https://www.cs.cmu.edu/~15326-f23/CACM-Fortnow.pdf`. Relativization.

Alexander Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, 1997.