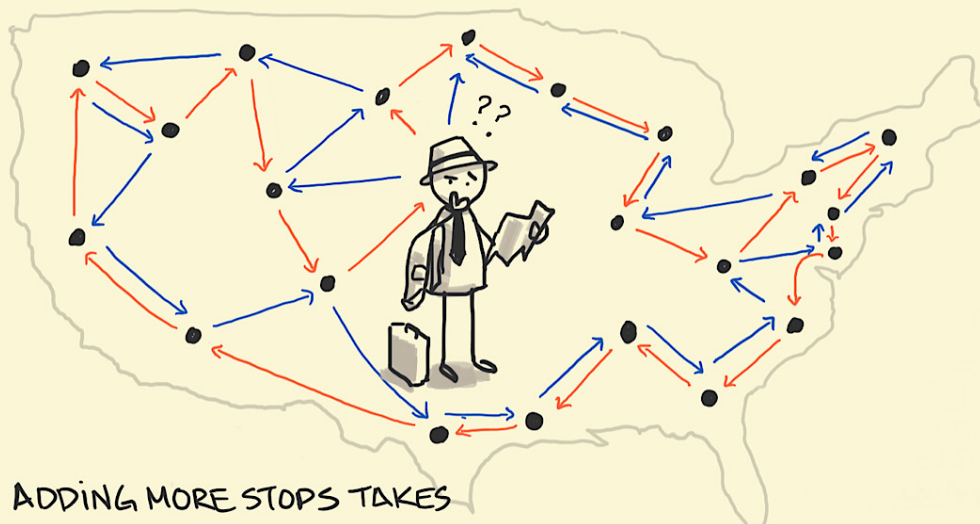# What are the main theoretical barriers to resolving P vs NP?

An Extended Project Qualification Dissertation

Camron Short

Candidate Number: 6439

Centre Number: 19216

Chapter 1

Defining the Problem

1.1 Introduction To Our Case Study

Throughout this paper I will be making references to a theoretical salesman. The details their sales are irrelevant for our topic. 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:
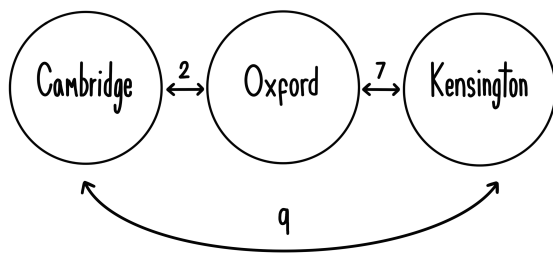
Comparatively to this non-trivial result:
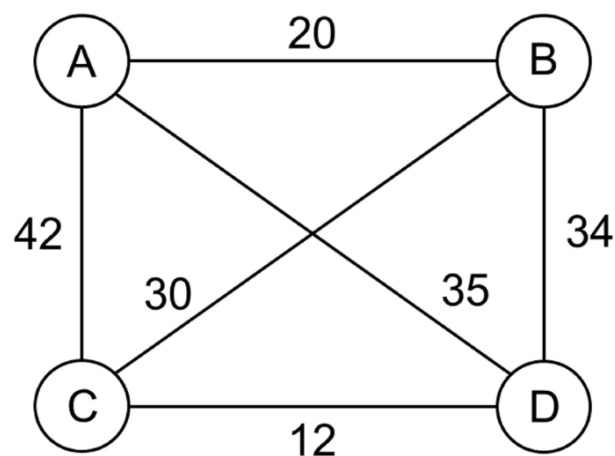
1.2 Big O-Notation

Our salesman can go to 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 or if they have a faster van. These 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 significant increase in 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**, generally considered efficient and tractable.

1.3 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. Sets are fundamental within computer science and mathematics. 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)

1.4 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 $O(n^2)$.

1.5   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 $P \subseteq NP$ as for any problem we can solve efficiently, we must also verify efficiently. Whether or not these sets are actually equal is the essence of our central question.

1.6   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 **P = NP** — implying that every hard-looking problem has a hidden efficient solution — while others believe **P ≠ 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.

Chapter 2

Relativization

2.0.1 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"*, and how an algorithm's reasoning encounters a limit when this additional power is introduced.

2.1 Oracles

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 $\mathbf{P^A}$ or $\mathbf{NP^A}$, meaning class $\mathbf{P}$ (or $\mathbf{NP}$) relative to oracle A.

2.2 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 $\mathbf{P = NP}$, as shown by Baker, Gill, and Solovay Baker et al. [1975].
They constructed two oracles, A and B, such that:

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

- $\mathbf{P^B \neq NP^B}$ (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.

2.3 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.

Chapter 3

Natural Proofs

3.1 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 $f \notin \mathbf{P}$.

3.2 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 arrangement of gates that perform a binary

operation. Each value of n gets a separate circuit. We intend to keep these circuits 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 $f : \{0,1\}^n \rightarrow \{0,1\}$ is the size of the smallest Boolean circuit that computes f correctly on all inputs. Referring back to our Salesman, if the 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 $\mathbf{NP}$, 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]

3.3   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.

3.4   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 $\mathbf{P}$ from $\mathbf{NP}$. 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 $\mathbf{NP}$, 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.

3.5   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.

3.6   Consequences of a Natural Barrier

The Razborov-Rudich insight does not just crumble our hope of a proof. It fundamentally changes where we have to look for a proof for $\mathbf{P} \neq \mathbf{NP}$. Their proof showed us that any proof method that satisfies the criteria to be *Natural* would allow us to distinguish between pseudorandom and random functions. Something assured to be computationally impossible under standard cryptographic assumptions. In short: if your proof techniques can work broadly and efficiently it is too broad to break the foundations of Cryptography. So unless you are willing to fundamentally shatter the foundations of the modern internet your technique cannot be sufficient.
This creates a barrier with serious consequences:

- It disqualifies many previously successful proof techniques, especially those that proved circuit lower bounds for simpler problems.

- It suggests that if $\mathbf{P} \neq \mathbf{NP}$ is true, then any successful proof must avoid being natural - it must violate either Constructivity, Largeness, or both.

- It highlights a surprising link between cryptography and complexity theory: breakthroughs in one domain may endanger assumptions in the other.

Return to our salesman: Imagine you try to expose their inefficiency by building a test, a kind of security scanner for bad routes. But every time you make it strong enough to work, it starts scanning and breaking real-world encrypted data. In trying to catch the salesman, you've also accidentally hacked a bank. The scanner is too powerful to be safely used.

"Natural proofs aren't too weak to prove $\mathbf{P} \neq \mathbf{NP}$ - they're too powerful. They solve more than they should, and in doing so, they violate assumptions we rely on elsewhere."
- Adapted from Razborov and Rudich [1997]

This result doesn't resolve the central question — but it tells us something just as important: If $\mathbf{P} \neq \mathbf{NP}$, then the path to proving it must avoid a large class of techniques.

Chapter 4

Prerequesites to Algebrization

Please note to the reader that this section may repeat some existing topics however the topics below are vital to our final barrier that it will prove beneficial to remind ourselves of them in depth.

4.1 Boolean Circuits... Again

To recap:
Previously, we met our salesman and imagined that they could build a machine to solve each map. This machine is not a computer in our sense, it does not run lines sequentially. Instead, it is a physical circuit composed of 'simple' logic gates like AND, OR, and NOT. These gates are arranged in a fixed pattern to solve one specific problem. This model is known as a **Boolean circuit**. Each circuit is designed for a single input size n. Therefore, a circuit to solve 5 towns is different to 6. This allows us to show that: No matter how ingenious we are, any circuit that solves a problem like TSP must be enormous, then we have shown the problem to be fundamentally difficult. We call this proving a *Lower Bound* Allowing us to answer the question of "even with custom hardware, how complex is the problem?" Boolean circuits have given us concrete methods to measure difficulty. But in order to develop this idea further researchers began looking for more flexible and powerful tools. Leading us to Oracles.

4.2 Relativization: algorithms with Oracles

Our Salesman is struggling to find the best route. Instead of working it out themselves, they have been given access to a magical helper that can answer a question at any given time. This helper is called an *Oracle* An Oracle does not explain its logic. Simply responds with a boolean value (True or False) instantly.
"Is this route shorter than that one?" — False
This Oracle changes the rules of our problem entirely. The salesman has just became significantly more powerful. They can now build strategies that would be too difficult to compute themselves with the aid of their new magical helper. Within Computer science, we represent complexity classes like: $\mathbf{P^A}$ or $\mathbf{NP^A}$ — Meaning the class $\mathbf{P}$ or $\mathbf{NP}$ when given access to the Oracle A. We then say that a proof relativizes if it works regardless of an Oracles presence.
There is a twist to this however. In 1975, Baker, Gill, and Solovay showed that some

oracles make $\mathbf{P} = \mathbf{NP}$, and others make $\mathbf{P} \neq \mathbf{NP}$ [Baker et al., 1975]. That means any proof technique that still works no matter what Oracle you add — any technique that *relativizes* — can't possibly settle the question. It gives contradictory results depending on the Oracle. As noted by Arora and Barak [2009], relativization "applies to many known techniques" and explains why "some previously promising approaches are unlikely to resolve $\mathbf{P} \neq \mathbf{NP}$".

> "Relativizing techniques are too blunt — they treat all oracles equally, even when the truth depends on which Oracle you choose."
> - Paraphrased from Baker et al. [1975]

So giving the Salesman a magic assistant makes them more powerful, it also reveals that many of our traditional methods of proof hit a ceiling, including those using Circuits. If they *relatavize* they cannot prove $\mathbf{P} \neq \mathbf{NP}$. To proceed further, we need tools that *don't relativize.* This is where algebra begins to enter the spotlight,

4.3 Arithmetization

Our Salesman so far has used logic gates and Oracles, boolean values and rigid machinery. What if they could see the problem in higher resolution? Instead of responding with a shallow "yes" or "no", what if they responded with a graph to explore or a map to see. This is the key concept behind **arithmetization**. Whithin complexity theory, arithmetization means transforming a boolean function, into a polynomial over a non infinite range. The goal is simple: turn rigid boolean logic into flexible algebra. Because algebra is powerful. With polynomials, we can interpolate values, check identities, and apply mathematical techniques that just don't work on pure logic. It gives us tools that are smoother and more continuous, even when applied to discrete problems. In the early 1990s this approach was considered revolutionary. researchers proved that $\mathbf{IP} = \mathbf{PSPACE}$ - a result that shattered the barrier of relativization by using artithmetization [Arora and Barak, 2009]. Showing that some techniques, by stepping outside convention, could reach further than thought possible. **Metaphorically:** Imagine the GPS of our Salesman no longer is limited to just giving directions. It can now overlay a detailed landscape as smooth curves. Now you don't just know if a better path exists, they can estimate just how much better the new path is.

> "Arithmetization allows an algorithm to reason about a function not just at specific points, but everywhere in between — like turning a pixelated map into a continuous surface."
> - Adapted from [Arora and Barak, 2009, Ch. 20.4]

This method does not parallel relativization, it is significantly ahead. Yet, when we tried to combine this with Oracle access we discovered a new barrier.

4.4 Oracles with Algebra

We know Oracles provide instant boolean results. We also know arithmetization allows algorithms to work with algebra instead of boolean logic. What happens if we combine these? This is the general idea behind **algebrization**. Suppose our Salesman now has a magic map — an oracle — that only answers binary questions. But also shows them

the shape of terrain. It reveals curves and gradients, costs and optimisations all as algebraic information. This new Oracle not only answers boolean problems it gives algebraic reasoning. In technical language: the oracle has been extended to allow access to a low-degree polynomial that agrees with the function from boolean input. The algorithm can now access both the function and algebraic extension $\tilde{A}$.

This looked like the breakthrough we needed. After all, arithmetization had led to

the remarkable result $\textbf{IP} = \textbf{PSPACE}$, even though that result doesn't relativize [Arora and Barak, 2009]. Maybe combining these ideas could finally allow us to show $\textbf{P} \neq \textbf{NP}$. But this hope was short-lived. In 2008, Arora, Barak, and Wigderson showed that even *algebrizing* techniques hit the same wall [Arora et al., 2009]. They constructed special oracles where:

- $\textbf{P}^{\textbf{A}} = \textbf{NP}^{\textbf{A}}$, and

- $\textbf{P}^{\textbf{B}} \neq \textbf{NP}^{\textbf{B}}$

Meaning that any proof method that works when you give it an Oracle *and* its algebraic extension - a method that *algebrizes* - cannot resolve $\textbf{P} \neq \textbf{NP}$.

Metaphorically: It's like giving the Salesman a magical map that doesn't just point

the way — it paints the whole landscape in equations. But even with this power, they still can't find a shortcut. No matter how detailed the map becomes, it doesn't contain the secret.

> "Algebrization unifies the strengths of relativization and arithmetization — and shows their combined limits."
> - Adapted from Arora et al. [2009]

For some, this result was sobering. It showed us that even the most powerful techniques known may not be enough, even united. If we are to solve $\textbf{P} \neq \textbf{NP}$, we must discover a fundamentally new kind of idea. And that is where we turn next: to the barrier itself.

Chapter 5

Algebrization

5.1 What is Algebrization?

Previously we explored giving our Salesman access to an *Oracle* and saw how it allowed them to bypass certain computations. We also saw how replacing rigid boolean with flexible logic we could achieve powerful results. When we combine these we get *algebrization.* This was mentioned previously in our Prerequesite knowledge.

> "An algebrizing technique is one that works even when the algorithm can query both the function and its low-degree polynomial extension."
> - Arora et al. [2009]

A technique is formally said to *algebrize* if it continues to work when both the Oracle A and its low-degree extension $\tilde{A}$ are available. These extensions come from viewing Boolean functions no longer as black boxes but polynomials.
Our Salesman would previously ask the GPS if a shortcut existed. Now they can know if a road is open and get an update on the terrain. This makes no mystery to be revealed with the answer and mathmatical logic behind it.

5.2 Why It Mattered

Until now, our Salesman had relied on his magic assistant (Oracle) or map (algebra). Now imagine the power of our Salesman with both. They would be able to conquer his problem trivially. This intuition did seem strong. One aspect of it revolutionised with a proof near our desired overarching problem. And the other is an intrinsic property. As Arora and Barak [2009] note, algebrization "combines the strengths of both relativization and arithmetization", it was a natural progression from our current techniques. This is where the optimism runs dry unfortunately.

5.3 The Barrier Itself

Arora et al. [2009] introduced the concept formally and then dismantled it. They showed us that techniques that combine the strengths of Oracles and algebraic resoning are not strong enough to resolve **P** versus **NP**. They constructed two oracles, A and B, and their corresponding algebraic extensions $\tilde{A}$ and $\tilde{B}$, such that:

- $\mathbf{P^A} = \mathbf{NP^A}$ — relative to Oracle A, the two classes collapse

- $\mathbf{P^B} \neq \mathbf{NP^B}$ — relative to Oracle B, the separation holds

When algorithms were allowed to query the low-degree polynomial extensions $\tilde{A}$ and $\tilde{B}$, the results differed. This proved that any technique that works under algebraic Oracle access must fail to decide $\mathbf{P} = \mathbf{NP}$.

> "The relativizing and arithmetizing barriers can be unified under the notion of algebrization — and this unified barrier still fails to separate complexity classes like $\mathbf{P}$ and $\mathbf{NP}$."
> - Arora et al. [2009]

Our Salesman is now equipped with everything — a GPS that answers instantly and a map of highest resolution. Despite the armoury presented they still cannot prove the existance of an efficient way to visit all the towns. The tools are as advanced as possible, but the truth remains obscured. The significance of this result not only lied in its ability to reveal a new barrier but because it showed that even *hybrid methods* were still not enough. It unified previous barriers and then extended their failure. For a brief moment, algebrization felt like the next great leap. But it turned out to be the next great limitation.

5.4   What algebrization Tells Us

Our Salesman has tried everything. Simple algorithms. Oracles for instantaneous answers. Powerful mathmatical maps have been used. And even the final 2 in parallel. Yet we are still no closer to proving if a solution exists. This is the moral of our final barrier. Even combinations of our most powerful techniques cannot resolve $\mathbf{P}$ vs $\mathbf{NP}$ [Arora et al., 2009, Arora and Barak, 2009].

It unifies two of the greatest successes of theoretical computer science then just as

swiftly, sobers us with a swift defeat in the form of a new barrier. Displaying the true difficulty of $\mathbf{P}$ vs $\mathbf{NP}$

> "Any technique that algebrizes will fail to separate $\mathbf{P}$ from $\mathbf{NP}$. This suggests the need for fundamentally new ideas."
> - [Arora and Barak, 2009, Ch. 20.4]

5.5   What Does This Mean?

It means that if $\mathbf{P} \neq \mathbf{NP}$ — and many believe it is (myself included) — the proof must lie somewhere we have not yet looked. Somewhere beyond the barriers of this report. The road our Salesman needs does exist, but it is hidden behind walls our current tools cannot penetrate. This is not a failure for complexity theory. It's a challenge, a challenge to be met by future computer scientist and mathematicians. A new barrier does not close doors to discovery instead will leave an open window into the understanding of our world and maths itself. This is not discouragement but a direction and foundation to guide the next generation.

"Algebrization subsumes the known barriers of relativization and natural proofs, and yet it too falls short of separating **P** from **NP**. We must look elsewhere."

-Arora et al. [2009]

Chapter 6

Conclusion

6.1 Why the Road Remains Untraveled

The heart of my project was the question "Why is the **P** vs **NP** problem still unresolved?"
Our goal to understand how it resisted solution for over 50 years by some of history's
greatest minds. Through the spectation of a travelling Salesment - inspired by an NP
problem (TSP) — we explored the best strategies computer science has developed to
date. And through this metaphor, we saw why none of them are yet enough. Beginning
with **Relativization** to **Natural Proofs** and their spiritual successor **Algebrization**.
We came to the unfortunate Conclusion our techniques are not powerful enough. These
barriers not only act as a technical limitation. But a statement obout knowledge itself.
About the rules we as mathematicians must follow in order to prove something so pro-
found. For our unfortunate Salesman these barriers presented themselved as physical
barriers and technological limits. Thus is the beauty of theoretical computer science and
maths. Each "failure" refines our understanding and each limitation marks progress. In-
creasing our knowledge of things that cannot work will increase our knowledge of what
might.

> "Barriers are not endpoints — they are waypoints. They tell us how far we've
> come, and where new paths must begin."
> -Inspired by themes in Arora and Barak [2009]

The question of **P** versus **NP** remains one of the deepest in both Maths and Computer
Science. And because of the work of those who identified these barriers we know some-
thing equally important. **We know which paths to not follow and hopefully this
can lead us to the correct path.**

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.

Sanjeev Arora, Boaz Barak, and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Transactions on Computation Theory (TOCT)*, 1(1):1–54, 2009. Originally published as ECCC TR08-112.