The Relativization Barrier

Marco L Carmosino

September 30, 2023

1 Introduction

Using diagonalization, we can prove results like the deterministic time hierarchy theorem: separation of $\mathsf{DTIME}[f(n)]$ from $\mathsf{DTIME}[g(n)]$ when $f(n)$ is sufficiently larger than $g(n)$. Notice that the computational resource is **the same** — deterministic time — on **both sides** of this separation. Major open problems ask instead for separations between *different* resources: space, time, nondeterminism, parallelism, randomness, non-uniformity, and many more. For example, can we separate $\mathsf{NTIME}[f(n)]$ from $\mathsf{DTIME}[g(n)]$ for non-trivial f and g? Does "plain diagonalization" suffice to answer such questions? Often, **no**. Here we will see why, using the first meta-mathematical barrier in computational complexity theory [BGS75].

2 Definitions & Tools

We'll begin by stating the most important[1] open question in theoretical computer science. First, we give machine-based definitions of two fundamental complexity classes.

Definition 1 (Deterministic & Nondeterministic Polynomial Time)**.**

$$\mathsf{P} = \bigcup_{c \in \mathbb{N}} \mathsf{DTIME}[n^c]$$
$$\mathsf{NP} = \bigcup_{c \in \mathbb{N}} \mathsf{NTIME}[n^c]$$

Let's adopt Cobham's Thesis: "feasible" languages can be decided in a fixed polynomial[2] number of steps on a deterministic Turing Machine. This identifies NP as the class of languages where membership is feasibly *checkable*. That is, for every language $\mathcal{L} \in \mathsf{NP}$ given x and a certificate y we can check in deterministic polynomial time if y proves that $x \in \mathcal{L}$. Formally,

Definition 2 (Verifier Definition of NP 2.3 of [AB09])**.** A language $\mathcal{L} \in \{0,1\}^*$ is in NP if there exists a polynomial $p \in \mathsf{poly}(n)$ and a polynomial-time deterministic TM M (the *verifier* for $\mathcal{L}$) such that

$$\forall x \in \{0,1\}^* \quad \mathcal{L}(x) \iff \exists y \in \{0,1\}^{p(|x|)}\ M(x,y)$$

We now ask: does every problem with efficiently *checkable* solutions have efficiently *discoverable* solutions?

Question 1.
Is P *equal to* NP*?*

[1] arguably

[2] Though the disctinctions between linear and quadratic runtimes are the subject of rich investigations into "fine-grained" complexity, we encounter sufficient difficulties attempting to separate P from NP for the purposes of these notes.

Let's motivate the class NP by defining natural problems related to logic, following [AB09, Section 2.7.2]. Fix a formal axiomatic system and logical language $\mathcal{A}$. For reasonable $\mathcal{A}$, these two languages are feasible to decide:

$$\mathsf{Parse} = \{x \in \{0,1\}^* \mid x \text{ encodes a well-formed formula } \varphi \text{ of } \mathcal{A}\}$$
$$\mathsf{Proves} = \{\langle x, y\rangle \mid \mathsf{Parse}(x) \wedge y \text{ encodes an } \mathcal{A}\text{-proof of } \varphi\}$$

That is, Parse and Proves are in P. Using the verifier definition, the "Bounded-Length Provability" language

$$\mathsf{BProvable} = \{\langle x, 1^n\rangle \mid \mathsf{Parse}(x) \wedge \varphi \text{ has a formal proof of } \leq n \text{ symbols in formal system } \mathcal{A}\}$$

is in NP. Therefore, asking if $\mathsf{P} = \mathsf{NP}$ is like asking if we can automate the areas of mathematics where proofs have "feasible" length. There are many detailed treatments of motivation and history for the P vs NP problem (AB, Lipton's Blog, Avi's Knowledge Creativity essay). It is a longstanding and central open problem in theoretical computer science; many people have tried to resolve it. But this is an empirical observation, not a theorem — we want *mathematical justification* for the difficulty of resolving P vs NP. This note describes the statements that can be proved "in a straightforward way" using diagonalization, and then we show that $\mathsf{P} \neq \mathsf{NP}$ is not one of these statements.

3 Relativizing Statements About Complexity Classes

Diagonalization arguments can give us more than intended. To see how, we'll define *Oracle Turing Machines*. These TMs are given access to a black box ("Oracle") containing some language $\mathcal{O} \subseteq \{0,1\}^*$, which they can query to obtain the answer to "is q in $\mathcal{O}$" in a single step. Thus, $M^{\mathcal{O}}$ is given the ability to decide $\mathcal{O}$ for "free", paying only for the resources needed to write down queries. Later we will restrict how machines are allowed to access an oracle, but in the most basic model a machine may issue a number of queries limited only by its time bound. We extend this notion to add a fixed oracle to an entire complexity class, possibly expanding the set of languages in the class.

Definition 3 (Relativized P and NP)**.** For any language $\mathcal{O} \subseteq \{0,1\}^*$ the class $P^{\mathcal{O}}$ is all languages that can be decided by a deterministic polynomial time oracle machine with access to $\mathcal{O}$. The class $\mathsf{NP}^{\mathcal{O}}$ is all langauges that can be decided by nondeterminstic polynomial time oracle macine with access to $\mathcal{O}$.

We can relativize any complexity class: take the machine definition of the class and allow it access to some $\mathcal{O}$ via queries. For any class $\mathcal{C}$, $\forall \mathcal{O}\ \mathcal{C} \subseteq \mathcal{C}^{\mathcal{O}}$ — adding an oracle will never "shrink" a complexity class, it can only become more powerful (decide more languages) or stay the same (if the oracle is useless). Some theorems remain true relative to *every* oracle, no matter how complicated or strange. For example,

Theorem 1 (Relativizing Simplified Deterministic Time Hierarchy Theorem)**.**

$$\forall \mathcal{O} \subseteq \{0,1\}^*\ \forall b \in \mathbb{N}\ \ \mathsf{DTIME}^{\mathcal{O}}[n^b] \subsetneq \mathsf{DTIME}^{\mathcal{O}}[n^{5b}]$$

Again, the complexity measure is the same on both sides of the separation: we have added *the same* oracle to deterministic time. Recalling the proof explains why: we only need the Universal TM to be able to efficiently simulate $O(n^b)$ time in $O(n^{5b})$ time. If it has the same oracle as the target machine, it can simply "pass through" queries and obtain the same results with constant overhead. Generalizing from this example, we have

Definition 4 (Relativizing Statements)**.** Let $\varphi(\mathcal{C}, \mathcal{D})$ be a statement about complexity classes $\mathcal{C}$ and $\mathcal{D}$. A true statement φ is *Relativizing* if $\forall \mathcal{O} \subseteq \{0,1\}^*\ \varphi(\mathcal{C}^{\mathcal{O}}, \mathcal{D}^{\mathcal{O}})$ — when we equip every class in the statement with the *same* oracle, it remains true. We write $\varphi^{\mathcal{O}}$ as shorthand for equipping every complexity class or machine mentioned by φ with oracle $\mathcal{O}$.

To prove that φ is relativizing, we inspect the proof of φ and generalize it to $\forall \mathcal{O}\ \varphi(\mathcal{C}^{\mathcal{O}}, \mathcal{D}^{\mathcal{O}})$. This is how some proof techniques give us more than expected; we get φ "relative to" every oracle, not just φ. Many heuristics have been developed for extracting relativizing theorems from existing proofs. When proofs of φ use only

1. Encoding TMs via bitstrings
2. An Efficient UTM

these arguments can often be adapted to prove that φ is relativizing. Intuitively, they treat computation as a "black box" so the addition of the same oracle to all classes involved does not change whether a simulation is efficient or not. Unfortunately, this is not formal: a statement φ is relativizing when a human can extend the proof of φ to show $\forall \mathcal{O}\ \varphi^{\mathcal{O}}$. Later, we will give a logical characterization of relativizing proofs. For now, we will see how even this informal notion can help explain why resolving P vs NP and many other open questions seems so difficult. For this, we need

Definition 5 (Non-Relativizing Statement)**.** Let $\varphi(\mathcal{C}, \mathcal{D})$ be a statement about complexity classes $\mathcal{C}$ and $\mathcal{D}$. A statement φ is *Non-Relativizing* if $\exists A\ \varphi^A \wedge \exists B\ \neg\varphi^B$.

To prove that φ is non-relativizing, we must exhibit two oracles — one where φ is true and one where it is false. We need not know if φ is true or false to show that it is non-relativizing, in contrast to a relativizing statement. This makes it plausible to discuss non-relativizing conjectures. It turns out that many open questions ask about non-relativizing statements, and this is the substance of the relativization barrier. The informal argument goes:

1. Many theorems φ are relativizing statements.
2. Therefore, many proofs consist only of relativizing "ingredients" — they extend to imply $\forall \mathcal{O}\ \varphi^{\mathcal{O}}$.
3. Many conjectures ψ in complexity theory concern non-relativizing statements ψ.
4. A "relativizing proof" of ψ would extend to imply $\forall \mathcal{O}\ \psi^{\mathcal{O}}$.
5. Therefore, no "relativizing proof" of ψ can exist, because it would extend to imply a contradiction: $\exists B\ \neg\psi^B$ by the definition of a non-relativizing statement.

Our world does not contain enough scare quotes to sufficiently decorate the above "argument." The phrase *extend to* does a lot of work; it corresponds to human inspection of a proof. Even interpreting the barrier can be controversial, because we have been somewhat cavalier about the definition of a relativizing statement — what if different ways of *adding an oracle to a complexity class* result in different classifications of the same φ, as relativizing or non-relativizing? Nevertheless, the relativization barrier has been a rich source of inspiration and research directions. And in the case of P vs NP, it seems that there is only one reasonable way to add an oracle to both classes (using the machine definitions above). So, we conclude by showing that $\mathsf{P} \neq \mathsf{NP}$ is a non-relativizing statement, and take this as one mathematical justification that the conjecture is difficult to prove. We exhibit appropriate oracles below.

4 $\mathsf{P} \neq \mathsf{NP}$ is Behind the Relativization Barrier

Theorem 2 (3.7 of [AB09], originally [BGS75])**.**
There exist oracles A, B such that $\mathsf{P}^A = \mathsf{NP}^A$ and $\mathsf{P}^B \neq \mathsf{NP}^B$. Thus, $\mathsf{P} \neq \mathsf{NP}$ is a non-relativizing statement.

Proof.

Claim 1. $\exists B\ \mathsf{P}^B = \mathsf{NP}^B$

To equate complexity classes relative to an oracle, we supply an oracle so powerful that it subsumes *both* base classes. Let's allow P and NP to ask about an excessive number of steps of deterministic computation,

$$\mathsf{EXPCOM} = \{\langle M, x, 1^n\rangle\ :\ M(x) = 1 \text{ within } 2^n \text{ steps}\}.$$

Plugging in the right machine description and padding, we immediately have $\mathsf{P}^{\mathsf{EXPCOM}} = \mathsf{EXP}$. Then,

$$\mathsf{EXP} \subseteq \mathsf{P}^{\mathsf{EXPCOM}} \subseteq \mathsf{NP}^{\mathsf{EXPCOM}} \subseteq \mathsf{EXP}.$$

For the last inclusion: NP can only issue poly-many oracle queries of poly-length on exponentially-many branches of computation to EXPCOM. Thus, brute-force simulation of NP in EXP has enough time to simulate each oracle query with an efficient UTM.

Claim 2. $\exists B\ \mathsf{P}^B \neq \mathsf{NP}^B$

Let $\mathcal{L}$ be any language. Define the unary content-indicator language based on $\mathcal{L}$ as

$$U_{\mathcal{L}} = \{1^n \mid \text{some } n\text{-bit } x \text{ in in } \mathcal{L}\}$$

That is, $1^n \in U_{\mathcal{L}} \iff \mathcal{L}_n$ is non-empty. With nondeterminism, every content-indicator language is easy. Indeed, we only need enough time to write down a query: $\forall \mathcal{L}\ U_{\mathcal{L}} \in \mathsf{NTIME}[O(n)]^{\mathcal{L}}$. On input 1^n do the following:

1. Guess $x \in \{0,1\}^n$
2. Query the oracle with x
3. Accept if $x \in \mathcal{L}$

There is at least one accepting path iff there is at least one n-bit x in $\mathcal{L}$, correctly deciding $U_{\mathcal{L}}$.

On the other hand, we will construct B such that $U_B \notin P^B$. Fix an enumeration of TMs $M_1, M_2, \ldots, M_i, \ldots$ where each TM is represented infinitely many times; the 'code # padding' enumeration from our simplified proof of the deterministic time hierarchy theorem would suffice, for example. The (perhaps ironic) idea is to diagonalize against deterministic oracle TMs: find out which stringa they attempt to query, and define the oracle B such that queried strings give no information about U_B.

Algorithm 1 Oracle Construction

Ensure: $\forall x\ B(x) \in \{?, 0, 1\}$

1: ▷ *All strings are either undetermined, outside B, or inside B* ◁
2: $i \leftarrow 0$ ▷ *stage counter*
3: $\forall x\ B(x) \leftarrow\ ?$
4: ▷ *Oracle, initially **undetermined** for every string* ◁
5: **for all** $i \in \mathbb{N}$ **do**
6:   ▷ *Each iteration, fix n and B such that M_i^B errs on $U_B(1^n)$ in $2^n/10$ time* ◁
7:   $n \leftarrow \max\{\ |x|\ :\ B(x) \in \{0,1\}\ \} + 1$
8:   ▷ *n is larger than length of every string determined so far* ◁
9:   Simulate $M_i^B(1^n)$ for $2^n/10$ steps,
    Monitoring and responding to each oracle query q:
10:   **if** $B(q) \in \{0,1\}$ **then** ▷ *q is determined, answer consistently*
11:     Answer with $B(q)$
12:   **else if** $B(q) =\ ?$ **then** ▷ *q is not determined, just say NO & **determine** q*
13:     Answer M_i with 0
14:     $B(q) \leftarrow 0$
15:   ▷ *Ensure M_i is wrong on 1^n using ? strings — more n-bit strings than steps of M, so it works* ◁
16:   **if** $M_i^B(1^n)$ Accepted **then**
17:     $B(x) \leftarrow 0\ \forall x \in \{0,1\}^n$
18:     ▷ $\implies U_B(1^n) = 0 \implies U_B(1^n) \neq M_i^B(1^n)$ ◁
19:   **else**
20:     $x \leftarrow$ lex-first x such that $B(x) =?$
21:     $B(x) \leftarrow 1$
22:     ▷ $\implies U_B(1^n) = 1 \implies U_B(1^n) \neq M_i^B(1^n)$ ◁

Let M be an arbitrary $p(n) \in \mathsf{poly}$-time oracle TM. M appears infinitely often in our enumeration of TMs and line 7 selects monotonically increasing n. Fix i witnessing this, such that M_i codes M and $p(n) < 2^n/10$. This means the simulation of M will terminate. Further, line 7 ensures that every string in $\{0,1\}^n$ is not determined at the beginning of simulation. Consider the state after simulation of M concludes (line 15): at most $2^n/10$ strings are determined, by the runtime bound on M. So at least $2^n - 2^n/10$ strings remain ? at line 15 of stage i. Case analysis on the acceptance of M concludes the proof. If M accepts, then we determine the remainer of strings such that B_n is empty, and so M is incorrect. If M rejects, we select one of the remaining ? strings to set to 1, and so M is incorrect.

□

5 Discussion Questions

1. Can one prove φ is relativizing meaning $\forall \mathcal{O} \varphi^{\mathcal{O}} \oplus \forall \mathcal{O} \neg \varphi^{\mathcal{O}}$ without proving φ ?
2. Is $\mathsf{NP} \neq \text{io}\mathsf{P}$ a non-relativizing statement?

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009. ISBN: 978-0-521-42426-4. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

[BGS75] Theodore P. Baker, John Gill, and Robert Solovay. "Relativizations of the P =? NP Question". In: *SIAM J. Comput.* 4.4 (1975), pp. 431–442. DOI: 10.1137/0204037. URL: https://doi.org/10.1137/0204037.