Abstract

The P vs. NP problem is one of the most fundamental open questions in computational complexity. This paper presents a Prime Mover Proof, a self-verifying argument that establishes P ≠ NP. The proof asserts that proving P ≠ NP is itself an NP problem, meaning its difficulty serves as direct empirical evidence that NP is distinct from P.

To reinforce this result, we present three supporting mathematical proofs:
1. Set-Theoretic Proof – Establishing the fundamental separation between P and NP.
2. Constructive Proof – Demonstrating that proving P ≠ NP is an NP problem.
3. Reductio ad Absurdum Proof – Showing contradiction if P = NP were assumed.

We introduce a computational framework based on Origin, Approach Space, and Destination Space, providing a structured model for decision problems. Additionally, we clarify how truth tables extend to NP problems, including weighted solution spaces such as knapsack-style problems.

By combining logical elegance with mathematical rigor, this proof offers a compelling case for P ≠ NP that is direct, self-verifying, and independent of reductionist assumptions. We welcome further analysis and discussion from the computational complexity community.

A Prime Mover Proof that $P \neq NP$

Donald Mortvedt

January 31, 2025

1 The Self-Evident Statement

If all P problems are solvable in polynomial time, and P $\neq$ NP is an NP problem, then P $\neq$ NP. (1)

2 Why This Stands Alone

- The statement itself is the proof—it requires no further breakdown.
- If $P = NP$, then proving $P \neq NP$ should be a P problem.
- Since proving $P \neq NP$ is demonstrably an NP problem, it confirms that NP is distinct from P.
- This is the Prime Mover of computational complexity—self-evident, irreducible, and undeniable.

3 Formal Proof That $P \neq NP$

3.1 Definitions

To solve a P or NP problem, we begin with a set **origin** and a set **destination** and attempt to find a satisfactory path from $\mathcal{O}$ to $\mathcal{D}$ using various valid recursive approaches until a solution is found.

- **Origin ($\mathcal{O}$):** The starting point before computation begins.
- **Approach Space ($\mathcal{P}$):** The set of all possible computational approaches connecting $\mathcal{O}$ to $\mathcal{D}$.
- **Destination Space ($\mathcal{D}$):** The conditions defining a valid solution path.
- **Polynomial Time (P):** The set of decision problems solvable in polynomial time on a deterministic Turing machine.
- **Nondeterministic Polynomial Time (NP):** The set of decision problems where a solution can be verified in polynomial time.
- **Truth Table ($\mathcal{T}$):** The set of all possibilities considered to satisfy a decision problem. In problems like knapsack, where solutions involve weights and values, the truth table extends naturally to structured tables of discrete possibilities.

3.2 Key Assertions

1. A bounded truth table can be computed in polynomial time.
2. An unbounded (or relatively infinite) truth table cannot be computed in polynomial time.
3. An NP solution is a P problem, but an NP problem itself is not necessarily P.
4. Since an NP problem cannot be reduced to a finite truth table, P $\neq$ NP.

3.3 The Self-Verifying Proof

1. The P vs. NP problem itself is an NP problem.
2. If P = NP, this proof should be easy to find.
3. Since the mathematical community cannot find this proof in polynomial time, we have empirical confirmation that P $\neq$ NP.

4 Supporting Proofs

While the Prime Mover Proof is sufficient, we present additional structured proofs to reinforce its validity.

4.1 Set-Theoretic Proof

We establish the separation between $\mathcal{P}$ and $\mathcal{NP}$ using formal set relations.

$$\mathcal{P} \subseteq \mathcal{NP}, \quad \exists x \in \mathcal{NP}, x \notin \mathcal{P} \Rightarrow \mathcal{P} \neq \mathcal{NP} \tag{2}$$

4.2 Constructive Proof

We demonstrate that proving $P \neq NP$ is itself an NP problem, reinforcing its difficulty as evidence.

$$\exists x \in \mathcal{NP}, x = \text{“Prove P} \neq \text{NP”} \tag{3}$$

If proving $P \neq NP$ is an NP problem and cannot be solved in polynomial time, then $P \neq NP$.

4.3 Reductio ad Absurdum Proof

We assume $P = NP$ and derive a contradiction based on the truth table structure of NP problems.

$$\forall x \in \mathcal{NP}, \quad |T(x)| > O(n^k) \Rightarrow x \notin \mathcal{P} \tag{4}$$

If an NP problem cannot be reduced to polynomial time, then $P \neq NP$.

5 Conclusion

- The proof of $P \neq NP$ is an NP problem.
- If $P = NP$, proving it should be easy, but it isn't.
- Therefore, $P \neq NP$.
- This proof stands on its own—it needs no further validation.

Prepared for submission to arXiv, ECCC, and formal journal review.