EasyChair Preprint

№ 13469

Note for the P Versus NP Problem (II)

Frank Vega

July 1, 2024

Article

Note for the P versus NP Problem (II)

Frank Vega [1]

[1] Information Physics Institute, Miami, Florida, United States; vega.frank@gmail.com

Abstract: One of the biggest unsolved mysteries in computer science is the P versus NP problem. It asks a simple question: can every problem whose solution can be quickly verified be solved just as quickly (Here, "quickly" means in polynomial time)? While the question itself was hinted at in a 1955 letter from John Nash, a formalization of the problem is credited to Stephen Cook and Leonid Levin. Despite decades of effort, no one has been able to definitively answer it. Closely related is the concept of NP-completeness. If even one NP-complete problem can be solved efficiently (in polynomial time), then it implies P equals NP. This work proposes that a specific NP-complete problem, ONE-IN-THREE 3SAT, can be solved efficiently. In this way, we prove that P is equal to NP.

Keywords: Complexity classes; Boolean formula; Graph; Completeness; Polynomial time

MSC: 68Q15; 68Q17; 68Q25

---

1. Introduction

The field of computer science grapples with one of its most significant and challenging unsolved problems: the P versus NP question [1]. At its core, this question asks whether efficient verification of a solution translates to efficient solving of the problem itself. Here, "efficient" refers to the existence of an algorithm that tackles the task in polynomial time, meaning the time it takes scales proportionally to the size of the input data.

The class of problems solvable by such efficient algorithms is denoted by P, or "class P." Another class, NP (standing for "nondeterministic polynomial time"), encompasses problems where solutions themselves can be verified efficiently. This verification relies on a "certificate," a piece of succinct information that quickly confirms the solution's validity [2].

The P versus NP problem essentially asks if P and NP are equivalent. If, as many believe, P is strictly contained within NP (meaning $P \neq NP$), then some problems in NP are inherently harder to solve than to verify. This distinction would have significant ramifications for various fields like cryptography and artificial intelligence [3,4].

Cracking the P versus NP problem is considered a pinnacle achievement in computer science. A solution would revolutionize our understanding of computation, potentially leading to groundbreaking algorithms that address some of humanity's most pressing challenges. The difficulty of this problem is reflected in its inclusion among the Millennium Prize Problems, a prestigious set of unsolved questions offering a million-dollar reward for a correct solution [1].

2. Materials and methods

NP-complete problems is the Mount Everest of computer science challenges. These problems are notoriously difficult because while solutions can be verified quickly, finding them efficiently remains a mystery. In computational complexity theory, a problem is considered NP-complete if it meets the following two criteria:

1. **Fast checking**: An NP-complete problem allows for speedy verification of a proposed solution using a succinct certificate [5].
2. **Universally tough relatives**: Any problem in NP can be efficiently transformed into an NP-complete problem [5].

The big implication: If we find a fast way to solve just one NP-complete problem, it becomes a golden key - unlocking efficient solutions for all NP problems! This would revolutionize

fields like cryptography, artificial intelligence, and planning [3,4]. Here are some examples of NP-complete problems:

- **Boolean satisfiability problem (SAT)**: Given a Boolean formula, determine whether there is an assignment of truth values to the variables that makes the formula true [6].
- **Independent Set**: Given an undirected graph $G = (V, E)$ (V is the set of vertices and E is the set of edges) and positive integer k, determine whether there is a set $V' \subseteq V$ of at least k vertices such that V' is an independent set in G [6]. An independent set V' is a non-empty subset of vertices in a graph G where no two vertices in the set are connected by an edge [6].

These are just a few examples of the many NP-complete problems that have been studied and have a close relation with our current result. An undirected graph $G = (V, E)$ is a comparability graph if there exists a partial order P on V, say $<_P$, such that $(u, v) \in E$ if and only if either $u <_P v$ or $v <_P u$ [7]. The problem of deciding whether an undirected graph is a comparability graph can be solved in polynomial time [7].

Definition 1. ***Independent Set for Comparability Graph (ISCG)***
INSTANCE: A comparability graph $G = (V, E)$ and a positive integer k.
QUESTION: Is there set V' of at least k vertices such that V' is an independent set in G?
REMARKS: This problem can be solved in polynomial time [8].

Formally, an instance of **Boolean satisfiability problem (SAT)** is a Boolean formula ϕ which is composed of:

1. Boolean variables: $x_1, x_2, \ldots, x_n$;
2. Boolean connectives: Any Boolean function with one or two inputs and one output, such as $\wedge$(AND), $\vee$(OR), $\neg$(NOT), $\Rightarrow$(implication), $\Leftrightarrow$(if and only if);
3. and parentheses.

A truth assignment for a Boolean formula ϕ is a set of values for the variables in ϕ. A satisfying truth assignment is a truth assignment that causes ϕ to be evaluated as true. A Boolean formula with a satisfying truth assignment is satisfiable. The problem SAT asks whether a given Boolean formula is satisfiable [6].

We define a CNF Boolean formula using the following terms: A literal in a Boolean formula is an occurrence of a variable or its negation [5]. A Boolean formula is in conjunctive normal form, or CNF, if it is expressed as an AND of clauses, each of which is the OR of one or more literals [5]. A Boolean formula is in 3-conjunctive normal form or 3CNF, if each clause has exactly three distinct literals [5].

For example, the Boolean formula:

$$(x_1 \vee \neg x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$$

is in 3CNF. The first of its three clauses is $(x_1 \vee \neg x_1 \vee x_2)$, which contains the three literals x_1, x_2 and $\neg x_1$.

We define the following problem:

Definition 2. ***ONE-IN-THREE 3SAT***
INSTANCE: A Boolean formula in 3CNF.
QUESTION: Is there exists a truth assignment such that each clause contains exactly one true literal?
REMARKS: This problem is a well-known NP-complete [6].

Putting all together yields a proof for the P versus NP problem.

3. Results

This is the main theorem.

Theorem 1. *ONE-IN-THREE 3SAT* $\in P$.

Proof. We can build a special kind of graph $G = (V, E)$ (comparability graph) from any Boolean formula ϕ with special properties (*ONE-IN-THREE 3SAT* formula). The Boolean formula ϕ contains n variables and m clauses. We assume that ϕ does not have any clause which contains a literal and its negation. This graph will help us solve the original formula.

1. Building the Graph:
   - Each literal in the formula gets its own vertex in the graph G. For each variable x, denote the positive literal x as x_+ and the negated literal $\neg x$ as x_-.
   - For each variable x in the formula, we introduce the edge (x_+, x_-) connecting both literal vertices.
   - For each clause $c_j = (x \vee y \vee z)$ in the formula, we add three new vertices to the graph G denoted by x_j, y_j and z_j.
   - We connect these new vertices with edges to existing literal vertices from the clause. Indeed, for each clause $c_j = (x \vee y \vee z)$ in ϕ, we construct six new edges using the new variables:
     - $(x_\diamond, x_j)$: This enforces that at most one vertex between $x_\diamond$ and x_j can be into an independent set ($\diamond = -$ if and only if the literal x_j represents the positive literal x; $\diamond = +$ if and only if the literal x_j represents the negated literal $\neg x$).
     - $(y_\diamond, y_j)$: This enforces that at most one vertex between $y_\diamond$ and y_j can be into an independent set ($\diamond = -$ if and only if the literal y_j represents the positive literal y; $\diamond = +$ if and only if the literal y_j represents the negated literal $\neg y$).
     - $(z_\diamond, z_j)$: This enforces that at most one vertex between $z_\diamond$ and z_j can be into an independent set ($\diamond = -$ if and only if the literal z_j represents the positive literal z; $\diamond = +$ if and only if the literal z_j represents the negated literal $\neg z$).
     - (x_j, y_j), (x_j, z_j) and (y_j, z_j): This enforces that at most one vertex between x_j, z_j and z_j can be into an independent set.
   - These edges imply the following conditions:
     - These edges ensure that at most one of the literal vertices for a single variable can be included in an independent set.
     - Whether we use a positive or negative version of the literal vertex depends on whether the original variable is positive or negated in the clause.
     - Additionally, edges are added between the new vertices created from every clause. This further enforces the rule that at most one of these vertices can be part of an independent set.
2. Understanding the Edges:
   - These edges ensure that a set of vertices in the graph (called an independent set) reflects a valid solution to the original formula.
   - A clause contains exactly one true literal in the formula only if at least one of its three new vertices created per clause is included in the independent set.
3. Mapping Between Solutions: An independent set in the graph corresponds to a solution for the formula if:
   - It includes at least one vertex created from every clause (ensuring all clauses contains exactly one true literal): An independent set in G cannot contain more that one vertex created for each clause.
   - It includes at least one literal vertex only if the corresponding literal is true in the solution: An independent set in G cannot contain both literal vertices for a single variable in ϕ.
4. Why it Works:

- The edges in the graph make sure the chosen vertices (independent set) represent a consistent truth assignment for the variables in the formula.
- A truth assignment solution with exactly one true literal per clause translates to an independent set V' with at least m vertices (for clauses) and n literal vertices (exactly one for each variable in ϕ). The existence of such independent set V' guarantee that G is a comparability graph. Certainly, the graph G would be a comparability graph since we can assign the number 3 for the vertices inside of V', the number 0 for the literal vertices outside of V' and the numbers 1 and 2 for the vertices created for each clause that are outside of V', respectively.

5. Equivalence and Complexity:
   - The existence of an independent set with at least $m + n$ vertices in the comparability graph of *ISCG* is equivalent to a truth assignment with exactly one true literal per clause in the Boolean formula of *ONE-IN-THREE 3SAT*.
   - Since *ISCG* is solvable in polynomial time, this proves that the original Boolean formula ϕ in *ONE-IN-THREE 3SAT* is also solvable in polynomial time. Besides, we can decide whether G is a comparability graph in polynomial time [7].

In simpler terms, this construction shows that solving the *ISCG* problem is the same as finding an appropriate certificate in an arbitrary instance of *ONE-IN-THREE 3SAT*. This implies that the original problem can also be solved efficiently. □

Theorem 2. $P = NP$.

Proof. This is a direct consequence of Theorem 1. □

4. Conclusion

A potential proof of $P = NP$, if verified and accepted by the mathematical community, would mark a monumental shift in our understanding of computational complexity. Here, we delve into the far-reaching consequences this equivalence could have:

1. Algorithmic Revolution:
   - Faster Solutions to Intractable Problems: Problems currently considered intractable, like protein folding, logistics optimization, and cryptography with certain algorithms, could be solved efficiently [3]. This would revolutionize fields like medicine, materials science, and cybersecurity [3].
   - Exponential Optimization: Everyday tasks involving complex optimization, like scheduling, resource allocation, and financial modeling, could be tackled with significantly faster algorithms [3]. This would lead to improved efficiency and decision-making across various industries [3].
2. Scientific Breakthroughs:
   - Accelerated Scientific Discovery: Complex simulations in physics, chemistry, and biology could be performed in a fraction of the current time [4]. This could unlock breakthroughs in areas like material design, drug discovery, and climate modeling [4].
   - Enhanced Big Data Analysis: The ability to efficiently analyze massive datasets would revolutionize fields like social science, economics, and healthcare. Patterns and insights hidden within vast amounts of data could be readily extracted [4].
3. Technological Advancements:
   - Artificial Intelligence Revolution: The development of more powerful AI algorithms could be significantly accelerated [4]. This could lead to significant advancements in areas like machine learning, natural language processing, and robotics [4].
   - Cryptography Redefined: The landscape of cryptography could be fundamentally altered. While $P = NP$ might render some current encryption methods

vulnerable, it could also pave the way for the development of new, provably secure cryptosystems [4].

4. Economic and Societal Impact:
    - Innovation Boost: The ability to solve complex problems efficiently would lead to a surge in innovation across various sectors. New products, services, and solutions could emerge at a much faster pace [3].
    - Resource Optimization: Efficient algorithms could help us better manage resources like energy, water, and transportation [3]. This would lead to a more sustainable future and a reduced environmental footprint [3].

In conclusion, a verified proof of $P = NP$ would be a scientific and technological game-changer. It would unlock a new era of computational efficiency, leading to advancements in diverse fields and potentially reshaping our world. While challenges and considerations remain, the potential benefits are vast and warrant continued exploration of this profound conjecture.

References

1. Cook, S.A. The P versus NP Problem, Clay Mathematics Institute. https://www.claymath.org/wp-content/uploads/2022/06/pvsnp.pdf, 2022. Accessed 19 June 2024.
2. Sudan, M. The P vs. NP problem. http://people.csail.mit.edu/madhu/papers/2010/pnp.pdf, 2010. Accessed June 23, 2024.
3. Fortnow, L. The status of the P versus NP problem. *Communications of the ACM* **2009**, *52*, 78–86. https://doi.org/10.1145/1562164.1562186.
4. Aaronson, S. $P \stackrel{?}{=} NP$. *Open Problems in Mathematics* **2016**, pp. 1–122. https://doi.org/10.1007/978-3-319-32162-2_1.
5. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*, 3rd ed.; The MIT Press, 2009.
6. Garey, M.R.; Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1 ed.; San Francisco: W. H. Freeman and Company, 1979.
7. Alvarez, C.; Greenlaw, R. A compendium of problems complete for symmetric logarithmic space. *Computational Complexity* **2000**, *9*, 123–145. https://doi.org/10.1007/PL00001603.
8. Golumbic, M.C. The complexity of comparability graph recognition and coloring. *Computing* **1977**, *18*, 199–208. https://doi.org/10.1007/BF02253207.

Short Biography of Authors

Frank Vega is essentially a Back-End Programmer and Mathematical Hobbyist who graduated in Computer Science in 2007. In May 2022, The Ramanujan Journal accepted his mathematical article about the Riemann hypothesis. The article "Robin's criterion on divisibility" makes several significant contributions to the field of number theory. It provides a proof of the Robin inequality for a large class of integers, and it suggests new directions for research in the area of analytic number theory.