Article

Solving NP-Complete Problems Efficiently

Frank Vega *

Posted Date: 22 August 2024

doi: 10.20944/preprints202408.1631.v1

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

 doi:10.20944/preprints202408.1631.v1

Article

Solving NP-Complete Problems Efficiently

Frank Vega

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

Abstract: The P versus NP problem is a fundamental question in computer science. It asks whether problems whose solutions can be quickly verified can also be quickly solved. Here, "quickly" refers to computational time that grows proportionally to the size of the input (polynomial time). While the problem's roots trace back to a 1955 letter from John Nash, its formalization is attributed to Stephen Cook and Leonid Levin. Despite extensive research, a definitive answer remains elusive. Closely tied to this is the concept of NP-completeness. If a single NP-complete problem could be solved efficiently, it would imply that all problems in NP can be solved efficiently, proving that P equals NP. This work posits that ONE-IN-THREE 3SAT, a notoriously difficult NP-complete problem, can be solved efficiently, thereby establishing the equivalence of P and NP.

Keywords: complexity classes; graph; polynomial time; Boolean formula

MSC Classification: 68Q15; 68Q17; 68Q25

---

1. Introduction

Computer science is confronted by the formidable challenge of the P versus NP problem [1]. Fundamentally, this inquiry seeks to determine if the ability to swiftly verify a solution implies the capacity to swiftly compute it. Here, "swiftly" denotes algorithms with a polynomial time complexity, where computational time grows proportionally to input size. Problems solvable within polynomial time constitute the class P. Conversely, NP encompasses problems whose solutions can be verified efficiently given a suitable "certificate" - a piece of information enabling rapid validation [2].

The crux of the P versus NP question lies in whether P and NP are identical. A prevailing belief is that P is a strict subset of NP ($P \neq NP$), signifying that certain problems are inherently more difficult to solve than to verify. Resolving this enigma holds profound implications for fields such as cryptography and artificial intelligence [3,4]. The P versus NP problem is widely considered one of the most challenging open questions in computer science. Evidence supporting its difficulty arises from techniques like relativization and natural proofs, which have yielded inconclusive results [5,6]. Similar problems, such as the VP versus VNP problem in algebraic complexity, remain unsolved [7].

Resolving the P versus NP question is often described as a "holy grail" of computer science. A positive resolution would revolutionize our understanding of computation, potentially leading to groundbreaking algorithms for critical problems. Reflecting its significance, the problem is listed among the Millennium Prize Problems. While recent years have seen progress in related areas, such as finding efficient solutions to specific instances of NP-complete problems, the core question of P versus NP remains unanswered [8]. A polynomial-time algorithm for any NP-complete problem would directly imply P equals NP [9]. Our work focuses on presenting such an algorithm for a well-known NP-complete problem.

2. Background and ancillary Results

NP-complete problems are the Everest of computational challenges. Despite the ease of verifying proposed solutions with a succinct certificate [9], finding these solutions efficiently remains an elusive goal. A problem is classified as NP-complete if it satisfies two stringent criteria within computational complexity theory:

1. Efficient Verifiability: Solutions can be swiftly checked using a concise proof.

 doi:10.20944/preprints202408.1631.v1

2. Universal Hardness: Every problem in the class NP can be transformed into an instance of this problem without significant computational overhead [9].

The implications of finding an efficient algorithm for a single NP-complete problem are profound. Such a breakthrough would serve as a master key, unlocking efficient solutions for all problems in NP, with transformative consequences for fields like cryptography, artificial intelligence, and planning [3,4].

Illustrative examples of NP-complete problems include:

- **Boolean satisfiability (SAT)**: Given a logical expression, determine if there exists an assignment of truth values to its variables that makes the entire expression true [10].
- **Independent Set**: In a given graph, identify a maximum-sized subset of vertices where no two vertices are connected by an edge [10].

The provided examples represent a small subset of the extensively studied NP-complete problems relevant to our current work. A comparability graph, denoted as $G = (V, E)$, is characterized by the existence of a partial order P on its vertex set V such that an edge connects vertices u and v if and only if either u precedes v or v precedes u in P [11]. Determining whether a given graph is a comparability graph can be accomplished efficiently (in polynomial time) [11].

Definition 1. ***Independent Set for Comparability Graph (ISCG):***

Given a comparability graph $G = (V, E)$ and a positive integer k, the ISCG problem asks whether there exists a subset V' of V containing at least k vertices such that no two vertices in V' are connected by an edge in G. This problem can be solved efficiently in polynomial time [12].

A **Boolean satisfiability problem (SAT)** instance is a Boolean formula constructed from:

1. Boolean variables: $x_1, x_2, ..., x_n$, which can take on the values true or false.
2. Boolean connectives: Logical operators such as AND ($\wedge$), OR ($\vee$), NOT ($\neg$), implication ($\Rightarrow$), and equivalence ($\Leftrightarrow$).
3. Parentheses: To specify the order of operations.

A truth assignment for a Boolean formula is a complete mapping of its variables to the values true or false. A satisfying truth assignment is one that evaluates the formula to true. If such an assignment exists, the formula is satisfiable. The SAT problem asks whether a given Boolean formula is satisfiable [10].

A literal is a single variable or its negation within a Boolean formula [9]. A Boolean formula is in conjunctive normal form (CNF) when it is expressed as a conjunction (AND) of clauses, where each clause is a disjunction (OR) of one or more literals [9]. A 3-conjunctive normal form ($3CNF$) formula is a specific type of CNF where each clause contains exactly three distinct literals [9].

For instance, the formula

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

in $3CNF$. Its first clause, $(x_1 \vee \neg x_1 \vee x_2)$, consists of the three literals x_1, $\neg x_1$ and x_2.

We introduce the *ONE-IN-THREE 3SAT* problem:

Definition 2. ***ONE-IN-THREE 3SAT:***

Given a Boolean formula in $3CNF$, determine if there exists a truth assignment such that exactly one literal is true in each clause. This problem is a well-known NP-complete problem [10].

By presenting an efficient solution to *ONE-IN-THREE 3SAT*, we would establish a proof that P equals NP.

 doi:10.20944/preprints202408.1631.v1

3. Main Result

This is a key finding.

Theorem 1. *ONE-IN-THREE 3SAT can be solved in polynomial time.*

Proof. Let φ be a Boolean formula satisfying the specific constraints of the *ONE-IN-THREE 3SAT* problem. We can transform φ in $3CNF$ into another Boolean formula ϕ in CNF such that each variable in ϕ is restricted to appear at most three times, and each literal at most twice (the use of CNF in ϕ requires only that each clause has at most 3 literals and contains at least 2 literals).

We proceed to convert φ into ϕ in the following way:

- Suppose that some variable x appears k times in φ.
- Replace the first occurrence of x by x_1, the second by x_2 and so on, where $x_1, x_2, \ldots, x_k$ are k new variables.
- Add $(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge \cdots \wedge (\neg x_k \vee x_1)$ to the expression.
  - This is logically equivalent to
  $$x_1 \Rightarrow x_2 \Rightarrow \cdots \Rightarrow x_k \Rightarrow x_1.$$
  - Note that each clause above has exactly 2 literals.
- The resulting equivalent expression in ϕ satisfies the condition for x.
- Suppose that we are given the following expression in φ:
$$\cdots \wedge (\neg x \vee a \vee b) \wedge \cdots \wedge (x \vee y \vee z) \cdots .$$
- The transformed expression is
$$\cdots (\neg x_1 \vee a \vee b) \wedge \cdots \wedge (x_2 \vee y \vee z) \cdots (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_1).$$
  - Variable x_1 appears thrice.
  - Literal x_1 appears once.
  - Literal $\neg x_1$ appears twice.
- We complete this transformation iterating over all the new expressions for each variable and putting them together in order to create the Boolean formula ϕ.

A truth assignment satisfying the *ONE-IN-THREE* condition for φ exists if and only if such an assignment exists for ϕ. The modified formula ϕ contains m clauses with three literals each and m' clauses with two literals each.

A special type of graph, $G = (V, E)$, known as a comparability graph, can be constructed from the Boolean formula ϕ. This graph will serve as a tool to solve the original formula φ.

1. **Building the Graph.**
   - **Vertex Creation:** Each literal in a three-literal clause of ϕ is represented by a unique vertex in G, denoted u_x. Similarly, each literal y in a two-literal clause is represented by a vertex v_y. For each variable x, both its positive (u_x, v_x) and negative ($u_{\neg x}$, $v_{\neg x}$) forms are represented as vertices, regardless of whether they appear in three- or two-literal clauses.
   - **Edge Creation for Variable Consistency:** For each variable x, an edge connects v_x and $v_{\neg x}$ to ensure at most one can be included in an independent set. If x appears in both three- and two-literal clauses, an edge connects u_x and $v_{\neg x}$ (or v_x and $u_{\neg x}$ in case of $\neg x$ could appear in both three- and two-literal clauses) to enforce consistency.
   - **Edge Creation for Clause Constraints:** For each three-literal clause $(x \vee y \vee z)$, edges are added between u_x, u_y and u_z to guarantee at most one can be in an independent set. For each two-literal clause $(x \vee y)$, an edge is added between v_x and v_y for the same purpose.

 doi:10.20944/preprints202408.1631.v1

2. **Edge Implications.** The introduced edges serve two primary purposes:
   - **Mutual Exclusion:** They prevent the simultaneous inclusion of literal vertices representing a variable and its negation within an independent set.
   - **Clause Restriction:** By connecting vertices from the same clause, they enforce the constraint that at most one literal per clause can be part of an independent set.
3. **Understanding the Edges.** The edges in the graph are designed to ensure the following:
   - **Solution Mapping:** An independent set in the graph corresponds to a valid solution for the formula ϕ.
   - **Clause Satisfaction:** A clause in ϕ contains exactly one true literal if and only if at least one of its corresponding vertices is included in the independent set.
4. **Mapping Between Solutions.** An independent set in the graph represents a valid solution to the formula if:
   - **Clause Coverage:** It includes at least one vertex from every clause, ensuring that each clause contains exactly one true literal.
   - **Literal Consistency:** It includes at most two vertices representing a specific literal (positive or negative) for each variable. This guarantees that the solution assigns a consistent truth value to each variable.
5. **Why it Works.**
   - **Consistency Enforcement:** The graph's structure ensures that any chosen set of vertices (independent set) corresponds to a valid truth assignment for the formula's variables.
   - **Solution Equivalence:** A truth assignment with exactly one true literal per clause in ϕ is directly equivalent to an independent set V' containing at least $m + m'$ vertices (where m and m' represent the number of three- and two-literal clauses, respectively). The existence of such an independent set guarantees that G is a comparability graph. This can be demonstrated by assigning a numerical rank to each vertex: 3 for vertices in V', 0 for literals in two-literal clauses outside V', and 1 or 2 for literals in three-literal clauses outside V'.
6. **Equivalence and Complexity.**
   - **Problem Equivalence:** A solution to the *ONE-IN-THREE 3SAT* problem (a truth assignment with exactly one true literal per clause) exists if and only if an independent set of size at least $m + m'$ exists in the corresponding comparability graph.
   - **Polynomial Time Solvability:** The *ISCG* problem, which involves finding such an independent set in a comparability graph, is solvable in polynomial time. Consequently, the original *ONE-IN-THREE 3SAT* problem can also be solved in polynomial time. This is because determining the existence of a suitable truth assignment is equivalent to finding the independent set, which is a computationally efficient task. Additionally, verifying if the constructed graph is indeed a comparability graph can be done in polynomial time [11].

In essence, this construction establishes a direct connection between solving the *ISCG* problem and finding a valid solution (or certificate) for any *ONE-IN-THREE 3SAT* instance. Since *ISCG* can be solved efficiently, it follows that the original, seemingly more complex problem can also be solved efficiently. □

This is the main theorem.

Theorem 2. $P = NP$.

Proof. A polynomial-time solution to any NP-complete problem would establish the equivalence of P and NP [9]. Despite extensive research on over 300 significant NP-complete problems, no such polynomial-time algorithm has been discovered [9]. Given that *ONE-IN-THREE 3SAT* is a well-known NP-complete problem [9], a polynomial-time solution for it, as presented in Theorem 1, would directly imply P equals NP. □

 doi:10.20944/preprints202408.1631.v1

4. Conclusion

A definitive proof that P equals NP would fundamentally reshape our computational landscape. The implications of such a discovery are profound and far-reaching:

- **Algorithmic Revolution.**
  - The most immediate impact would be a dramatic acceleration of problem-solving capabilities. Complex challenges currently deemed intractable, such as protein folding, logistics optimization, and certain cryptographic problems, could become efficiently solvable [3]. This breakthrough would revolutionize fields from medicine to cybersecurity. Moreover, everyday optimization tasks, from scheduling to financial modeling, would benefit from exponentially faster algorithms, leading to improved efficiency and decision-making across industries [3].
- **Scientific Advancements.**
  - Scientific research would undergo a paradigm shift. Complex simulations in fields like physics, chemistry, and biology could be executed at unprecedented speeds, accelerating discoveries in materials science, drug development, and climate modeling [3]. The ability to efficiently analyze massive datasets would provide unparalleled insights in social sciences, economics, and healthcare, unlocking hidden patterns and correlations [3].
- **Technological Transformation.**
  - Artificial intelligence would be profoundly impacted. The development of more powerful AI algorithms would be significantly accelerated, leading to breakthroughs in machine learning, natural language processing, and robotics [8]. While the cryptographic landscape would face challenges, it would also present opportunities to develop new, provably secure encryption methods [8].
- **Economic and Societal Benefits.**
  - The broader economic and societal implications are equally significant. A surge in innovation across various sectors would be fueled by the ability to efficiently solve complex problems. Resource optimization, from energy to transportation, would become more feasible, contributing to a sustainable future [3].

In conclusion, a proof of $P = NP$ would usher in a new era of computational power with transformative effects on science, technology, and society. While challenges and uncertainties exist, the potential benefits are immense, making this a compelling area of continued research.

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 August 7, 2024.
2. Sudan, M. The P vs. NP problem. http://people.csail.mit.edu/madhu/papers/2010/pnp.pdf, 2010. Accessed August 7, 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. doi:10.1007/978-3-319-32162-2_1.
5. Baker, T.; Gill, J.; Solovay, R. Relativizations of the $\mathcal{P} =? \mathcal{NP}$ Question. *SIAM Journal on computing* **1975**, *4*, 431–442. doi:10.1137/0204037.
6. Razborov, A.A.; Rudich, S. Natural Proofs. *Journal of Computer and System Sciences* **1997**, *1*, 24–35. doi:10.1006/jcss.1997.1494.
7. Wigderson, A. *Mathematics and Computation: A Theory Revolutionizing Technology and Science*; Princeton University Press, 2019.

 doi:10.20944/preprints202408.1631.v1

8. Fortnow, L. Fifty Years of P vs. NP and the Possibility of the Impossible. *Communications of the ACM* **2022**, *65*, 76–85. doi:10.1145/3460351.
9. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*, 3rd ed.; The MIT Press, 2009.
10. 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.
11. Alvarez, C.; Greenlaw, R. A compendium of problems complete for symmetric logarithmic space. *Computational Complexity* **2000**, *9*, 123–145. doi:10.1007/PL00001603.
12. Golumbic, M.C. The complexity of comparability graph recognition and coloring. *Computing* **1977**, *18*, 199–208. doi:10.1007/BF02253207.