Article

SAT in Polynomial Time: A Proof of P = NP

Frank Vega

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

Abstract: The P versus NP problem is a cornerstone of theoretical computer science, asking whether problems that are easy to check are also easy to solve. "Easy" here means solvable in polynomial time, where the computation time grows proportionally to the input size. While this problem's origins can be traced to John Nash's 1955 letter, its formalization is credited to Stephen Cook and Leonid Levin. Despite decades of research, a definitive answer remains elusive. Central to this question is the concept of NP-completeness. If even one NP-complete problem, like SAT, could be solved efficiently, it would imply that all NP problems could be solved efficiently, proving P=NP. This research proposes a groundbreaking claim: SAT, traditionally considered NP-complete, can be solved in polynomial time, establishing the equivalence of P and NP.

Keywords: complexity classes; graph; polynomial time; completeness; reduction

---

1. Introduction

The P versus NP problem is a fundamental question in computer science that asks whether problems whose solutions can be easily checked can also be easily solved [1]. "Easily" here means solvable in polynomial time, where the computation time grows proportionally to the input size [1,2]. Problems solvable in polynomial time belong to the class P, while NP includes problems whose solutions can be verified efficiently given a suitable "certificate" [1,2]. Alternatively, P and NP can be defined in terms of deterministic and non-deterministic Turing machines with polynomial time complexity [1,2].

The central question is whether P and NP are the same. Most researchers believe that P is a strict subset of NP, meaning that some problems are inherently harder to solve than to verify. Resolving this problem has profound implications for fields like cryptography and artificial intelligence [3,4]. The P versus NP problem is widely considered one of the most challenging open questions in computer science. Techniques like relativization and natural proofs have yielded inconclusive results, suggesting the problem's difficulty [5,6].

The P versus NP problem is often described as a "holy grail" of computer science. A positive resolution could revolutionize our understanding of computation and potentially lead to groundbreaking algorithms for critical problems. 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 [3]. A polynomial time algorithm for any NP-complete problem would directly imply P equals NP [7]. 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, 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 quickly checked using a concise proof [7].
2. **Universal Hardness:** Every problem in the class NP can be reduced to this problem without significant computational overhead [7].

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) Problem:** Given a logical expression in conjunctive normal form, determine if there exists an assignment of truth values to its variables that makes the entire expression true [8].
- **Boolean 3-Satisfiability (3SAT) Problem:** Given a Boolean formula in conjunctive normal form with exactly three literals per clause, determine if there exists a truth assignment to its variables that makes the formula evaluate to true [8].
- **Not-All-Equal 3-Satisfiability (NAE-3SAT) Problem:** Given a Boolean formula in conjunctive normal form with exactly three literals per clause, decide if there exists a satisfying truth assignment such that each clause has at least one true variable and at least one false variable [8].

The provided examples represent a small subset of the extensively studied NP-complete problems relevant to our current work. The following problem can be solved in polynomial time:

Definition 1. ***Maximum Three-Length Disjoint Paths (MAX-3LDP) Problem***

INSTANCE: A graph $G = (V, E)$, specified vertices s and t, and a positive integer k.

QUESTION: Does G contain k or more mutually vertex-disjoint paths from s to t, each involving exactly three edges?

REMARKS: Solvable in polynomial time [8,9]. The problem is NP-complete for paths longer than three edges [8,9]. We refer to these k mutually disjoint paths of exactly three edges connecting s to t as k vertex-disjoint s-t 3-edge paths.

Formally, a Boolean formula ϕ 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 mapping from the variables of ϕ to the Boolean values $\{true, false\}$. A truth assignment is satisfying if it makes ϕ evaluate to true. A Boolean formula is satisfiable if it has at least one satisfying truth assignment. A literal is a Boolean variable or its negation. A Boolean formula is in Conjunctive Normal Form (CNF) if it is a conjunction (AND) of clauses, where each clause is a disjunction (OR) of one or more literals [7]. The SAT problem asks whether a given Boolean formula in CNF is satisfiable [8]. A $3CNF$ formula is a CNF formula in which each clause contains exactly three distinct literals [7]. For example, the following formula is in $3CNF$:

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

The first clause, $(x_1 \vee \neg x_2 \vee x_3)$, contains the three literals x_1, $\neg x_2$ and x_3. The version of SAT where formulas are in $3CNF$ is called 3SAT [7]. We formally define the following problem:

Definition 2. ***Monotone Not-All-Equal 3-Satisfiability (NAE-3MSAT) Problem***

INSTANCE: A Boolean formula in $3CNF$ with monotone clauses (meaning the variables are never negated).

QUESTION: Is there exists a satisfying truth assignment such that each clause has at least one true variable and at least one false variable?

REMARKS: This problem is complete for NP [10]. Here, the certificate is a valid satisfying truth assignment, meaning it satisfies the NAE-3MSAT condition for each clause.

By presenting the NP-completeness and a polynomial time solution to SAT, we would establish a proof that P equals NP.

3. Main Result

Even though the following reductions are widely known, we will rely on them as supporting results in our analysis [8].

Theorem 1. *The problem SAT can be reduced to 3SAT in polynomial time.*

Proof. We will not delve into the specific steps of this reduction, as it is a standard technique in computer science [7]. To reduce a general SAT instance to a 3SAT formula, we can follow these steps:

1. **Expand Clauses with Few Literals:** To ensure that all clauses contain exactly three literals, we introduce two new variables and expand clauses with at most two literals into clauses with three literals by considering all possible combinations of the new variables, both negated and positive. For instance, consider the two new variables A and B. A single-literal clause (x) can be equivalently expressed as:

$$(x \vee A \vee B) \wedge (x \vee \neg A \vee \neg B) \wedge (x \vee A \vee \neg B) \wedge (x \vee \neg A \vee B).$$

   Similarly, a two-literal clause $(x \vee y)$ is equivalent to:

$$(x \vee y \vee A) \wedge (x \vee y \vee \neg A) \wedge (x \vee y \vee B) \wedge (x \vee y \vee \neg B).$$

   Note that the same variables A and B are used in both cases.
2. **Identify Long Clauses:** Find all clauses with more than three literals.
3. **Introduce New Variables:** For a clause with n literals (where $n > 3$), introduce $n - 3$ new variables.
4. **Create New Clauses:** Create a chain of clauses with three literals each, using the original literals and the new variables. Ensure that the satisfiability of the original clause is preserved in this chain of new clauses. To exemplify, consider a clause containing four literals, $(x \vee y \vee z \vee w)$. By introducing a single additional variable, D, this clause can be logically represented as the conjunction of the following two clauses:

$$(x \vee y \vee \neg D) \wedge (D \vee z \vee w).$$

By systematically applying this reduction to each clause, we can transform any SAT instance into an equivalent 3SAT formula. This reduction demonstrates that 3SAT is at least as hard as the general SAT problem, and thus, it is an NP-complete problem. □

Theorem 2. *The problem 3SAT can be reduced to NAE-3SAT in polynomial time.*

Proof. Any 3SAT formula ϕ can be reduced to an equivalent NAE-3SAT instance. We assume that no clause contains a literal and its negation. Such tautological clauses can be removed. We also remove clauses containing literals whose negations do not appear in the 3SAT instance. This reduction involves the following steps:

- **Variable Introduction:**
  1. **Global Variable:** Introduce a new variable w that does not appear in ϕ.
  2. **Clause Variables:** For each clause $c_i = (x \vee y \vee z)$ in ϕ, introduce a new variable a_i.
- **Clause Construction:**
  - **Clause Reduction:** For each clause $c_i = (x \vee y \vee z)$, construct two NAE-3SAT clauses:

* $(x \vee y \vee a_i) \wedge (z \vee \neg a_i \vee w)$.

By construction, a satisfying truth assignment for ϕ corresponds to a valid satisfying truth assignment for the NAE-3SAT instance when w is assigned the value false, and vice versa. When w is true, we can obtain a satisfying truth assignment for ϕ by negating all the values in a valid satisfying truth assignment for the NAE-3SAT instance. This reduction demonstrates that NAE-3SAT is NP-complete. □

Theorem 3. *The problem NAE-3SAT can be reduced to NAE-3MSAT in polynomial time.*

Proof. It is well-known that any Boolean formula ϕ in NAE-3SAT can be reduced to an equivalent NAE-3MSAT instance. This reduction involves the following steps:

- **Variable Introduction:**
  1. **Literal Variables:** For each variable x in ϕ, introduce two variables: x_+ representing the positive literal x and x_- representing the negative literal $\neg x$. Additionally, we introduce three new variables a_x, b_x, and c_x for each variable x in ϕ.
- **Clause Construction:**
  1. **Clause Reduction:** For each clause $c_i = (x \vee y \vee z)$, construct one NAE-3MSAT clause:
     - $(x_{s_x} \vee y_{s_y} \vee z_{s_z})$, where s_v is $+$ if literal $v \in \{x, y, z\}$ is positive and $-$ otherwise.
  2. **Variable Consistency:** For each variable x in ϕ, construct four NAE-3MSAT clauses:
     - $(x_+ \vee x_- \vee a_x)$, $(x_+ \vee x_- \vee b_x)$, $(x_+ \vee x_- \vee c_x)$, and $(a_x \vee b_x \vee c_x)$. These clauses ensure that exactly one of x_+ and x_- is true.

By construction, a valid satisfying truth assignment for ϕ corresponds to a valid satisfying truth assignment for the NAE-3MSAT instance, and vice versa. Thus, this reduction proves that NAE-3MSAT is NP-complete. □

This is a Main Insight.

Theorem 4. *The problem NAE-3MSAT can be reduced to MAX-3LDP in polynomial time.*

Proof. To better visualize this polynomial time reduction, we will use a graphical representation of the graph involved. To represent a NAE-3MSAT formula ϕ as a graph $G = (V, E)$, we introduce a gadget for each variable x in ϕ. To enforce consistency, we connect all positive literals of a variable x to the logical constraints of the clauses where they appear. This ensures that if a positive literal of x is chosen in one clause, all other positive literals of x must also be selected, effectively "covering" all occurrences of x. The connection between the variable x and the clauses c_h, c_i, and c_j is depicted in Figure 1.

Here, x_h, x_i, and x_j are the occurrences of x in clauses c_h, c_i, and c_j, respectively. Since x_h, x_i, and x_j are different occurrences of x in ϕ, then selecting x_i implies the selection of both x_j and x_h. Conversely, choosing $\neg x_h$ forces us to choose both $\neg x_i$ and $\neg x_j$. To ensure this, Figure 1 adds vertices a_x, b_x, d_x, e_x, f_x, g_x, p_x, q_x, and r_x, with dashed vertices representing repeated ones. Note that dashed vertices with identical labels represent the same unique vertex.

The topology of the variable gadget ensures consistency. The construction (Figure 1) is designed such that if a positive vertex is matched to a vertex outside the gadget via a s-t 3-edge path, then all negative vertices must be matched within the gadget using s-t 3-edge paths. The reverse implication also holds. Thus, the "availability" of a vertex to be matched by an outside vertex corresponds to the truth assignment. This is supported by the requirement that each vertex p_x, q_x, and r_x must be covered by at least one vertex-disjoint s-t 3-edge path.

We use the gadget in Figure 2 for each clause $c_i = (x \vee y \vee z)$ in ϕ. By selecting exactly one of these vertex-disjoint s-t 3-edge paths, we ensure that clause c_i is satisfied in the NAE-3MSAT constraint. If

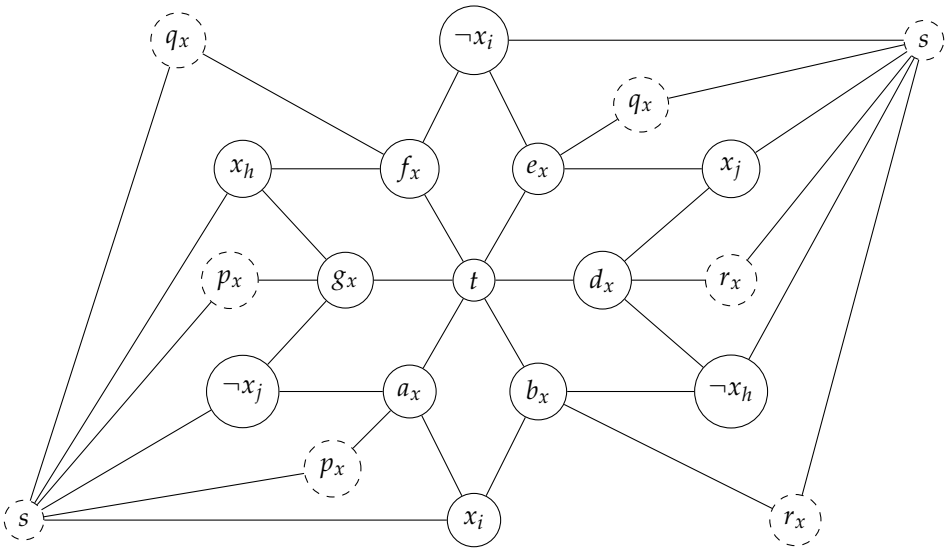

Figure 1. x-gadget for the occurrences of x in clauses c_h, c_i, c_j of ϕ.

all variables in c_i are assigned the same truth value, no s-t 3-edge path can be selected. Conversely, if c_i is satisfied under the NAE-3MSAT condition, exactly one s-t 3-edge path must be chosen.

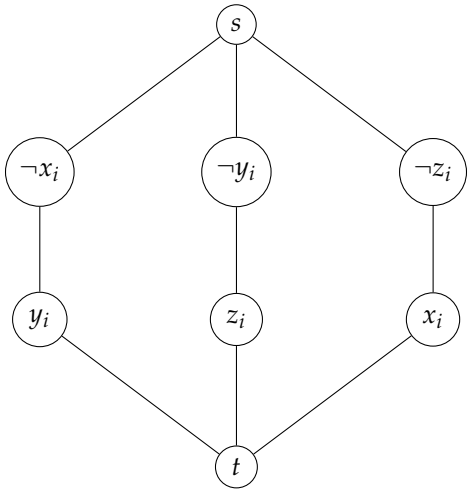

Figure 2. c_i-gadget corresponding to the clause $c_i = (x \vee y \vee z)$ in ϕ.

A Boolean formula ϕ satisfies the NAE-3MSAT constraints if and only if its corresponding graph G admits at least $7 \cdot m$ vertex-disjoint s-t 3-edge paths, where m denotes the number of clauses in ϕ. This equivalence is a direct consequence of the construction of G, which is meticulously designed to accurately reflect the logical structure of ϕ.

The graph G incorporates two types of vertex-disjoint s-t 3-edge path:

1. **Variable Gadgets:** The gadget in Figure 1 allows us to select two s-t 3-edge paths for each variable occurrence in a clause of ϕ. Since each clause comprises three distinct variables, there are precisely $6 \cdot m$ vertex-disjoint s-t 3-edge paths.
2. **Clause Gadgets:** The final step in Figure 2 ensures that exactly one s-t 3-edge path is chosen per clause, satisfying the NAE-3MSAT condition. This leads to a total of m vertex-disjoint s-t 3-edge paths, inducing a valid truth assignment for ϕ.
3. **All Gadgets:** Note that all gadgets include source vertex s and target vertex t.

Hence, a valid truth assignment for ϕ is directly equivalent to a graph G containing at least $7 \cdot m$ vertex-disjoint s-t 3-edge paths: $6 \cdot m$ arising from the variable gadgets and m from the clause gadgets.

Conversely, any such vertex-disjoint s-t 3-edge paths in G can be interpreted as a valid truth assignment for ϕ. This one-to-one correspondence establishes the equivalence between the valid satisfiability of ϕ and the existence of $7 \cdot m$ vertex-disjoint s-t 3-edge paths in G, thereby concluding the proof. □

This is a key finding.

Theorem 5. $SAT \in P$.

Proof. This follows directly from Theorems 1, 2, 3, and 4. □

This is the Main Theorem.

Theorem 6. $P = NP$.

Proof. Cook's Theorem states that every NP problem can be reduced to SAT in polynomial time [8]. Given that SAT is an NP-complete problem, a polynomial time solution for it, as presented here, would directly imply P equals NP. □

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 [3,4]. 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 [3,4].
- **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 [3,4]. The ability to efficiently analyze massive datasets would provide unparalleled insights in social sciences, economics, and healthcare, unlocking hidden patterns and correlations [3,4].
- **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 [3,4]. While the cryptographic landscape would face challenges, it would also present opportunities to develop new, provably secure encryption methods [3,4].
- **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 [3,4].

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 December 12, 2024.
2. Sudan, M. The P vs. NP problem. http://people.csail.mit.edu/madhu/papers/2010/pnp.pdf, 2010. Accessed December 12, 2024.
3. 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.
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. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*, 3rd ed.; The MIT Press, 2009.
8. 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.
9. Itai, A.; Perl, Y.; Shiloach, Y. The complexity of finding maximum disjoint paths with length constraints. *Networks* **1982**, *12*, 277–286. doi:10.1002/net.3230120306.
10. Schaefer, T.J. The complexity of satisfiability problems. STOC '78: Proceedings of the tenth annual ACM symposium on Theory of computing, 1978, pp. 216–226. doi:10.1145/800133.804350.