Article

On the P versus NP Problem

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 could be solved efficiently, it would imply that all problems in NP could be solved efficiently, proving P equals NP. This research proposes that a notoriously difficult NP-complete problem can be solved efficiently, thereby potentially 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" [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]. Similar problems, such as the VP versus VNP problem in algebraic complexity, remain unsolved [7].

Resolving 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 [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, 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 [9].
2. **Universal Hardness:** Every problem in the class NP can be reduced to 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) Problem:** Given a logical expression, determine if there exists an assignment of truth values to its variables that makes the entire expression true [10].
- **Exact k-Coloring Problem:** Given a graph G and a positive integer k, determine if there exists a valid coloring of G such that exactly k vertices have the same color and no adjacent vertices have the same color. This problem is equivalent to finding an independent set of size k, an NP-complete problem [10].

The provided examples represent a small subset of the extensively studied NP-complete problems relevant to our current work.

The L = SL theorem is a significant breakthrough in computational complexity theory [11]. It establishes that deterministic logarithmic space (L) and symmetric logarithmic space (SL) are equivalent complexity classes [12]. This means that any problem solvable by a deterministic Turing machine using only a logarithmic amount of space can also be solved by a symmetric Turing machine with the same space constraints [12]. The following problem is complete for SL [13]:

Definition 1. ***Exact Cover-2 (EC-2) Problem***

INSTANCE: A universe set U and a family of n sets $S_i \subseteq U$, where every element in U appears at most twice in the list $S_1, \ldots, S_n$.

QUESTION: Is there exists a subfamily $S'_1, \ldots, S'_m$ with $m \leq n$ such that $S'_i \cap S'_j = \emptyset$ for $1 \leq i \neq j \leq m$ and $S'_1 \cup \ldots \cup S'_m = U$?

REMARKS: This problem can be solved in logarithmic space [11,14].

We consider a variant of the problem.

Definition 2. ***Exact K-Cover-2 (K-EC-2) Problem***

INSTANCE: A universe set U, a family of n sets $S_i \subseteq U$ and a positive integer $k \leq n$, where every element in U appears exactly twice in the list $S_1, \ldots, S_n$.

QUESTION: Is there exists a subfamily of exactly k sets $S'_1, \ldots, S'_k$ such that $S'_i \cap S'_j = \emptyset$ for $1 \leq i \neq j \leq k$ and $S'_1 \cup \ldots \cup S'_k = U$?

REMARKS: To simplify notation, we denote the family of n sets $S_i \subseteq U$ as the collection C.

An independent set V' is a subset of vertices in a graph G where no two vertices in the set are connected by an edge. In addition, a vertex cover (sometimes called a node cover) of a graph G is a subset of its vertices, denoted by V', such that every edge in G has at least one endpoint in V'. A bipartite graph, denoted as $B = (U, V, E)$, is an undirected graph characterized by the existence of two node sets U, V and edges in E that only connect nodes from opposite sets.

Definition 3. ***Exact Independent Vertex Cover (XIVC) Problem***

INSTANCE: An undirected graph $G = (V, E)$ and a positive integer k.

QUESTION: Is there set V' of exactly k vertices such that V' is both a vertex cover and an independent set in G?

REMARKS: Solving the XIVC problem is akin to determining a 2-coloring of a bipartite graph such that one of the partitions contains exactly k vertices. This task can be accomplished in polynomial time, given the efficient solvability of the 2-coloring problem in bipartite graphs.

Formally, an instance of **Boolean Satisfiability (SAT) Problem** 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 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. The SAT problem asks whether a given Boolean formula is satisfiable [10].

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 [9]. A $2CNF$ formula is a CNF formula in which each clause contains exactly two distinct literals [9].

For example, the following formula is in $2CNF$:

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

The first clause, $(x_1 \vee \neg x_2)$, contains the two literals x_1 and $\neg x_2$. In addition, a $3CNF$ formula is a Boolean formula in conjunctive normal form with exactly three literals per clause.

We formally define the following problem:

Definition 4. ***Exact Monotone Xor 2-Satisfiability (XX2MSAT) Problem***

INSTANCE: An n-variable $2CNF$ *formula with monotone clauses (meaning the variables are never negated) using logic operators* $\oplus$ *(instead of using the operator* $\vee$*) and a positive integer k.*

QUESTION: Is there exists a satisfying truth assignment in which exactly k of the variables are true?

Finally, we introduce the last problem:

Definition 5. ***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 [15]. Here, the certificate is an appropriate satisfying truth assignment, meaning it satisfies the NAE-3MSAT condition for each clause.

By presenting the NP-completeness and a polynomial time algorithm to NAE-3MSAT, we would establish a proof that P equals NP.

3. Main Result

This is an important result.

Theorem 1. *The problem NAE-3MSAT can be reduced to K-EC-2 in polynomial time.*

Proof. To better visualize this polynomial time reduction, we will use a graphical representation of the sets involved. To represent a NAE-3MSAT formula ϕ as a collection of sets C over a universe U, we introduce a gadget for each variable x in ϕ. This gadget consists of $2 \cdot k$ triangles, where k is the larger of the number of occurrences of x in ϕ. Each triangle in the gadget corresponds to a possible truth assignment for the variable x. The apexes of the triangles are labeled with x_i or $\neg x_i$ to denote the truth assignment required for clause c_i in ϕ.

The topology of the gadget ensures consistency. The construction (e.g., Figure 1) guarantees that if any positive vertex is matched with some vertices outside of the this gadget then all negative vertices can only be matched by the triangles inside this gadget, and vice versa. Thus the "availability" of a

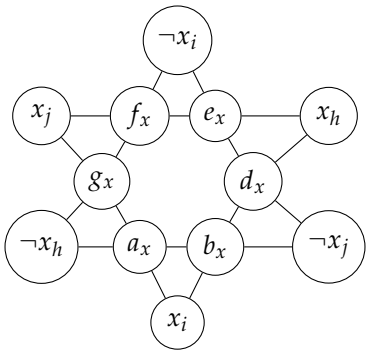

Figure 1. x-gadget for the occurrences of x in clauses c_i, c_j, c_h of ϕ.

vertex to be matched by an outside vertex corresponds to the truth assignment. For instance, in Figure 1, these sets would be:

$$\{x_i, a_x, b_x\}, \{\neg x_i, f_x, e_x\},$$
$$\{x_j, f_x, g_x\}, \{\neg x_j, b_x, d_x\},$$
$$\{x_h, e_x, d_x\}, \{\neg x_h, g_x, a_x\}.$$

To represent each clause $c_i = (x \vee y \vee z)$ in ϕ, we define three sets:

$$\{\neg x_i, y_i\}, \{\neg y_i, z_i\}, \{\neg z_i, x_i\}.$$

By ensuring that exactly one of these sets is chosen, we satisfy the NAE-3MSAT condition for clause c_i. If all variables in c_i have the same truth assignment, none of these sets can be selected. Conversely, if c_i is satisfied under the NAE-3MSAT condition, then exactly one of these sets will be covered.

A Boolean formula ϕ satisfies the NAE-3MSAT constraints if and only if its corresponding collection of sets C can be partitioned into exactly $4 \cdot m$ disjoint sets, where m is the number of clauses in ϕ. This equivalence follows directly from the construction of C, which is designed to faithfully represent the logical structure of ϕ over the universe U.

The collection of sets C incorporates two types of set:

1. **Variable Sets:** The construction depicted in Figure 1 enables the selection of one set for each variable occurrence within a clause of ϕ. Since each clause comprises three distinct variables, there are precisely $3 \cdot m$ such sets.
2. **Clause Sets:** The final step enforces the NAE-3SAT condition by requiring each clause in ϕ to choose exactly one set. This guarantees the existence of precisely m clause sets that induce an appropriate satisfying truth assignment for ϕ.

Hence, an appropriate satisfying truth assignment for ϕ directly corresponds to a partition of C into $4 \cdot m$ disjoint sets: $3 \cdot m$ variable sets and m clause sets. Conversely, any such partition of C can be interpreted as an appropriate satisfying truth assignment for ϕ. This one-to-one correspondence establishes the equivalence between the appropriate satisfiability of ϕ and the existence of a $4 \cdot m$-set partition in C, where each element in U belongs to exactly two sets in C. The proof is hereby concluded. □

This is a main insight.

Theorem 2. *The problem K-EC-2 can be reduced to XX2MSAT in polynomial time.*

Proof. We present a reduction from the K-EC-2 problem to the XX2MSAT problem. Given an instance (U, C, k) of K-EC-2, we construct an equivalent XX2MSAT instance as follows:

1. **Formula Construction:**
    - **Variables:** For each set $S_i \in C$, introduce a Boolean variable x_i.
    - **Clauses:** For every pair of sets $S_i, S_j \in C$ that share a common element $u \in U$, create a clause $(x_i \oplus x_j)$.
2. **Equivalence of Solutions:**
    - A solution to the K-EC-2 instance, consisting of k mutually disjoint sets that cover U, directly corresponds to a truth assignment to the XX2MSAT instance where exactly k variables are true.
    - Conversely, a truth assignment to the XX2MSAT instance with k true variables corresponds to a selection of k sets in C that are mutually disjoint and cover U.

    To see why, consider the following:
    - **Covering the Universe:** The clause structure ensures that every element in U is covered by at least one selected set. If an element is not covered, then the corresponding clause would be unsatisfied.
    - **Mutual Disjointness:** The XOR clauses between pairs of intersecting sets enforce mutual disjointness. If two sets with a common element are both selected, the corresponding clause would be unsatisfied.

Therefore, the K-EC-2 problem and the XX2MSAT problem are equivalent, and a solution to one can be efficiently transformed into a solution to the other. □

These are the main theorems.

Theorem 3. *$XIVC \in P$.*

Proof. Given the efficient solvability of the 2-coloring problem in bipartite graphs, we claim that the XIVC problem can be accomplished within polynomial time. This is a straightforward dynamic programming algorithm similar to solve subset sum: Let $(A_1, B_1), (A_2, B_2) \ldots, (A_p, B_p)$ be the sides of partitions A_i and B_i in a connected component i of the bipartite graph $B = (U, V, E)$, such that every vertex in a single partition has the same color.

Now, we create a dynamic programming table $DP[i, t]$ that stores whether it is possible to have a bipartite graph with exactly t vertices on one color using the i first components. The bi-dimensional boolean array DP, having dimensions $(p + 1)$ by $(k + 1)$ and zero-based indexing, is initialized. All elements are assigned the value *false*, with the exception of the element at index $(0, 0)$ which is assigned the value *true* (i.e., $DP[0, 0] = true$). Using the recurrence

$$DP[i, t] = DP[i - 1, t - \mid A_i \mid] \vee DP[i - 1, t - \mid B_i \mid],$$

we correctly decide whether there exists an entire partitioning of exactly k vertices with the same color after by examining $DP[p, k]$, where $\mid \ldots \mid$ is the cardinality set function. The recurrence evaluates $DP[i, t]$ as false for any i and t that do not satisfy $0 \leq i \leq p$ and $0 \leq t \leq k$. This is a polynomial time algorithm since the running time is bounded by $O(\mid U \mid + \mid V \mid + \mid E \mid + p \cdot k)$. Identifying 2-color partitions takes $O(|U| + |V| + |E|)$ time using breadth-first search algorithm (BFS), while finding k vertices of the same color requires $O(p \cdot k)$ iterations. We can easily determine if a graph is two-colorable by performing a breadth-first search and assigning alternating colors to the nodes. Every connected component is partitioned into two sets using two colors. For isolated vertices, one of the sets is empty. Similarly, the dynamic programming algorithm to solve subset sum (in this specific variation) can be solved by systematically checking all possible values from 1 to k using each pair of partitions for every connected component. □

Theorem 4. *$XX2MSAT \in P$.*

Proof. There is a connection between finding a satisfying truth assignment in XX2MSAT with exactly k true variables and finding a set of exactly k vertices that is both a vertex cover and an independent set in a specific graph construction.

Here's a breakdown of the equivalence:

1. **Graph Construction:**
   - Each vertex in the new graph represents a variable in the XX2MSAT formula.
   - Edges are created between variables based on the structure of the $2CNF$ clauses: If two variables appear in a clause (e.g., $(x \oplus y)$), then an edge is drawn between the corresponding vertices in the graph.
2. **XX2MSAT and the Graph:**
   - A truth assignment in XX2MSAT where exactly k variables are true directly translates to a set of exactly k vertices in the constructed graph where true variables correspond to the vertices included in the set.
   - The properties of XX2MSAT clauses ensure that:
     - **Vertex Cover:** The chosen vertices cover all the edges (due to the structure of the clauses and the way edges are formed). This satisfies the vertex cover condition.
     - **Independent Set:** The chosen vertices don't have any edges connecting them (because the variables are connected in the graph, and only one variable from each clause can be true). This satisfies the independent set condition.

Therefore, finding a satisfying truth assignment with exactly k true variables in XX2MSAT is indeed equivalent to finding a set of exactly k vertices that fulfills both vertex cover and independent set requirements in the corresponding graph. However, we know the problem of finding a set of exactly k vertices that is both a vertex cover and an independent set can be solved in polynomial time. Consequently, the instances of the problem XX2MSAT can be solved in polynomial time as well. □

This is a key finding.

Theorem 5. *NAE-3MSAT is in P.*

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

This is a significant result.

Theorem 6. *$P = NP$.*

Proof. A polynomial time algorithm to any NP-complete problem would establish the equivalence of P and NP [9]. Given that NAE-3MSAT is an NP-complete problem, a polynomial time algorithm 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]. 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 December 2, 2024.
2. Sudan, M. The P vs. NP problem. http://people.csail.mit.edu/madhu/papers/2010/pnp.pdf, 2010. Accessed December 2, 2024.
3. Fortnow, L. The status of the P versus NP problem. *Communications of the ACM* **2009**, *52*, 78–86. doi: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.
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. Reingold, O. Undirected connectivity in log-space. *Journal of the ACM (JACM)* **2008**, *55*, 1–24. doi:10.1145/1391289.1391291.
12. Michel, P. A survey of space complexity. *Theoretical Computer Science* **1992**, *101*, 99–132. doi:10.1016/0304-3975(92)90151-5.
13. Alvarez, C.; Greenlaw, R. A compendium of problems complete for symmetric logarithmic space. *Computational Complexity* **2000**, *9*, 123–145. doi:10.1007/PL00001603.

14. Jones, N.D.; Lien, Y.E.; Laaser, W.T. New problems complete for nondeterministic log space. *Mathematical Systems Theory* **1976**, *10*, 1–17. doi:10.1007/BF01683259.
15. 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.