Note for the P versus NP problem (II)

Frank Vega *

Posted Date: 6 November 2024

doi: 10.20944/preprints202409.2053.v7

Keywords: Complexity classes; Graph; Polynomial time; Completeness; Reduction

 doi:10.20944/preprints202409.2053.v7

Article

Note for the P Versus NP Problem (II)

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 SAT, a notoriously difficult NP-complete problem, can be solved efficiently, thereby potentially establishing the equivalence of P and NP. This work is an expansion and refinement of the article "Note for the P versus NP problem", published in IPI Letters.

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

MSC: Primary 68Q15; Secondary 68Q17, 68Q25

---

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].

 doi:10.20944/preprints202409.2053.v7

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].
- **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 [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 [11].
- **Partition into Triangles (PT) Problem:** Given a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, the PT problem asks whether the vertices of G can be partitioned into disjoint sets $V_1, V_2, \ldots, V_q$, each containing exactly 3 vertices, such that each V_i induces a triangle in G [11]. In other words, we want to know if we can divide the graph into q disjoint triangles.

The provided examples represent a small subset of the extensively studied NP-complete problems relevant to our current work. A bipartite graph, denoted as (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. 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'.

Definition 2.1. ***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 [11].

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 $2CNF$:

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

 doi:10.20944/preprints202409.2053.v7

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 now introduce some well-known problems:

Definition 2.2. ***Monotone One-in-three 3-Satisfiability (1-IN-3-3MSAT) Problem***

INSTANCE: Given a Boolean formula in $3CNF$ *with monotone clauses (meaning the variables are never negated)*

QUESTION: Is there exists a satisfying truth assignment in which exactly one variable is true for each clause?

REMARKS: This problem is complete for NP [10].

Definition 2.3. ***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?

By presenting the NP-completeness and an efficient 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 [11].

Theorem 3.1. *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 [9]. To reduce a general SAT formula to a 3SAT formula, we can follow these steps:

1. **Converting a Logic Expression to Clause Form:** To convert a logic expression into clause form (CNF), we need to transform it into a conjunction of clauses, where each clause is a disjunction of literals. Here's a general approach:
   (a) **Eliminate Implications:** Replace implications with equivalent disjunctions using the rule: $\neg p \vee q$ is equivalent to $p \Rightarrow q$.
   (b) **Push Negations Inward:** Use De Morgan's laws to move negations inward, so they apply directly to individual variables. For example, $\neg(p \wedge q)$ becomes $\neg p \vee \neg q$.
   (c) **Convert to Conjunctive Normal Form:** Use the distributive law to distribute disjunctions over conjunctions. This may involve introducing new variables to represent intermediate expressions.
   (d) **Standardize Variables:** Ensure that each variable appears with a unique name throughout the formula.
   (e) **Clause Form:** The resulting expression will be in CNF, where each clause is a disjunction of literals.
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.

By systematically applying this reduction to each long clause, we can transform any SAT formula 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. □

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2024 doi:10.20944/preprints202409.2053.v7

Theorem 3.2. *The problem 3SAT can be reduced to 1-IN-3-3MSAT in polynomial time.*

Proof. It is well-known that any Boolean formula ϕ in 3SAT can be reduced to an equivalent 1-IN-3-3MSAT 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.
  2. **Clause Variables:** For each clause $c_i = (x \vee y \vee z)$ in ϕ, introduce four new variables: a_i, b_i, c_i, and d_i.
- **Clause Construction:**
  1. **Clause Reduction:** For each clause $c_i = (x \vee y \vee z)$, construct three 1-IN-3-3MSAT clauses:
     - $(x_s \vee a_i \vee b_i)$, where $s = -$ if x is positive and $s = +$ otherwise.
     - $(y_s \vee b_i \vee c_i)$, where $s = +$ if y is positive and $s = -$ otherwise.
     - $(z_s \vee c_i \vee d_i)$, where $s = -$ if z is positive and $s = +$ otherwise.
  2. **Variable Consistency:** For each variable x in ϕ, construct three 1-IN-3-3MSAT clauses:
     - $(x_+ \vee x_- \vee a_x)$, $(x_+ \vee x_- \vee b_x)$, and $(a_x \vee b_x \vee c_x)$. These clauses ensure that exactly one of x_+ and x_- is true.

By construction, a satisfying truth assignment for ϕ corresponds to an appropriate satisfying truth assignment for the 1-IN-3-3MSAT instance, and vice versa. Thus, this reduction proves that 1-IN-3-3MSAT is NP-complete. □

Theorem 3.3. *The problem 1-IN-3-3MSAT can be reduced to PT in polynomial time.*

Proof. To represent a 1-IN-3-3MSAT formula φ as an undirected graph $G = (V, E)$, 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 tips of the triangles are labeled with x or $\neg x$ to indicate the corresponding truth assignment.

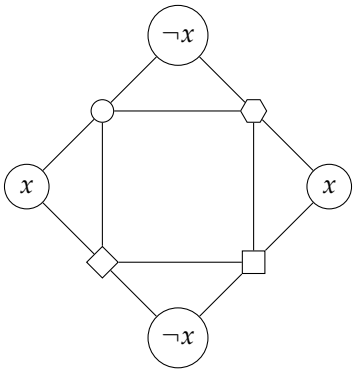

Figure 1. x gadget for two occurrences in φ.

To represent each clause $c = (x \vee y \vee z)$ in φ, we introduce three new vertices: U, V, and W. We connect U and V to the positive literals of c in the graph. As Figure 2 shows, exactly one positive literal from c, along with U and V, must form a triangle. This ensures that exactly one of the positive literals in c is true, thereby satisfying the 1-IN-3-3MSAT requirement.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2024 doi:10.20944/preprints202409.2053.v7

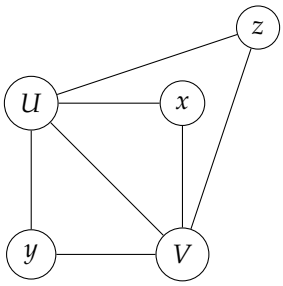

Figure 2. The connection between the positive literals of c and the new vertices U and V.

We then connect the negative literals to the logical constraints that involve their corresponding positive literals. As depicted in Figure 3, this design ensures that exactly two negative literals will be part of triangles if only one positive literal in the clause is chosen. This condition is necessary to satisfy the 1-IN-3-3MSAT requirement.

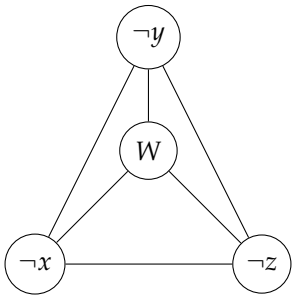

Figure 3. Fulfilling the conditions of the 1-IN-3-3MSAT problem.

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

A Boolean formula φ satisfies the 1-IN-3-3MSAT requirement if and only if its corresponding graph G can be partitioned into exactly $5 \cdot m$ disjoint triangles, where m is the number of clauses in φ. This equivalence is evident from the construction of G, which faithfully captures the logical structure of φ.

The graph G incorporates two types of triangles:

1. **Variable Triangles:** For each occurrence of a variable in φ, the variable gadget introduces a triangle. Since each clause contains the occurrence of three distinct variables, there are exactly $3 \cdot m$ such triangles.
2. **Clause Triangles:** For each clause in φ, the construction described in Figures 2 and 3 introduces two triangles if and only if exactly one variable in the clause is true. This ensures that there are exactly $2 \cdot m$ clause triangles in a satisfying truth assignment according to the 1-IN-3-3MSAT requirement.

Therefore, an appropriate satisfying truth assignment for φ directly corresponds to a partition of G into $5 \cdot m$ disjoint triangles: $3 \cdot m$ variable triangles and $2 \cdot m$ clause triangles. Conversely, any such partition of G 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 $5 \cdot m$-triangle partition in G.

Obviously PT problem is in NP, and it is also NP-hard because 1-IN-3-3MSAT can be reduced to PT. So PT is NP-complete. □

 doi:10.20944/preprints202409.2053.v7

This is a main insight.

Theorem 3.4. *The problem PT can be reduced to XX2MSAT in polynomial time.*

Proof. We can build an equivalent XX2MSAT instance for any given instance $G = (V, E)$ of the PT problem:

1. **Variables:**
   - Create a variable for each vertex v in the original graph G. Denote this variable as v itself.
   - For each pair of disconnected vertices u and v in G (i.e., $(u, v) \notin E$), introduce a new variable denoted by x_{uv}.
2. **Clauses:**
   - For every two disconnected vertices $u, v \in V$ (i.e., $(u, v) \notin E$), construct two clauses using the new variables:
     - $(u \oplus x_{uv})$: This enforces that either u is true or the new variable x_{uv} is true (XOR).
     - $(x_{uv} \oplus v)$: This enforces that either the new variable x_{uv} is true or v is true (XOR).

Key Points about the Construction:

- The two clauses for each pair of disconnected vertices u and v in G ensures that both variables u and v have the same truth value.
- Moreover, the two clauses together imply that x_{uv} is true exactly when both u and v are false. This means that for any pair of vertices u and v that are not directly connected in the graph, at least one of the three variables u, v, or x_{uv} must be assigned the value true.

Mapping between PT solutions and XX2MSAT assignments:

- A satisfying truth assignment in the XX2MSAT formula corresponds to a valid partition into triangles in the original graph G if:
  - Variables assigned false represent the vertices in the graph G.
  - If the new variable x_{uv} is assigned the value true, it means that the disconnected vertices u and v must belong to different triangles in the partition.

Why this construction works:

- The clauses guarantee that, in a satisfying truth assignment, the original variables (representing vertices) and the corresponding new variables must have compatible truth values.
- If a graph can be partitioned into q triangles, then there exists a corresponding truth assignment where the original variables (representing vertices) are false and the single new variable x_{uv} is true for each pair of disconnected vertices u and v.

Equivalence and Complexity:

- A q-triangle partition of G can be mapped to a satisfying truth assignment of the XX2MSAT formula with exactly $k = 9 \cdot \frac{q \cdot (q-1)}{2}$ true variables, and vice versa.
- If we partition the graph into q disjoint sets, each vertex in a set V_i will be disconnected from $3 \cdot (q - i)$ vertices that belong to the sets $V_{i+1}, V_{i+2}, \ldots, V_q$. By summing the number of disconnected pairs for all vertices in all sets, we arrive at a total of $9 \cdot \frac{q \cdot (q-1)}{2}$ disconnected pairs due to $9 \cdot (q-1) + 9 \cdot (q-2) + \ldots + 9 \cdot (1) = 9 \cdot \frac{q \cdot (q-1)}{2}$. For any pair of disconnected vertices u and v, the variable x_{uv} is assigned the value true if and only if both u and v are assigned the value false. As a result, there will be a total of $9 \cdot \frac{q \cdot (q-1)}{2}$ true variables in the satisfying truth assignment, since all the original variables corresponding to vertices are set to false. It's important to note that a satisfying truth assignment with exactly $9 \cdot \frac{q \cdot (q-1)}{2}$ true variables directly corresponds to a partition of the graph G into q triangles.

 doi:10.20944/preprints202409.2053.v7

- Given that PT is an NP-complete problem and we have demonstrated a polynomial-time reduction from PT to XX2MSAT, we can conclude that XX2MSAT is also NP-complete.

In essence, the proof demonstrates that solving XX2MSAT is equivalent to finding a q-triangle partition in the PT problem. This implies that XX2MSAT inherits the NP-completeness property from PT. □

These are the main theorems.

Theorem 3.5. $XIVC \in P$.

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 (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 array DP of dimensions $(p+1)$ by $(k+1)$ (i.e., using zero-based indexing) is initialized such that all elements are *false*, with the exception of $DP[0, 0]$ which is set to *true* (i.e., assuming that array indices start from zero). 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 $DP[p, k]$. 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. Similarly, the subset sum problem (in this specific variation) can be solved by systematically checking all possible values from 0 to k using each pair of partitions for every connected component. □

Theorem 3.6. $XX2MSAT \in P$ [12].

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.

 doi:10.20944/preprints202409.2053.v7

- **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 3.7. $P = NP$.

Proof. A polynomial-time solution to any NP-complete problem would establish the equivalence of P and NP [9]. 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]. 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.

 doi:10.20944/preprints202409.2053.v7

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 September 26, 2024.
2. Sudan, M. The P vs. NP problem. http://people.csail.mit.edu/madhu/papers/2010/pnp.pdf, 2010. Accessed September 26, 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. Schaefer, T.J. The complexity of satisfiability problems. Proceedings of the tenth annual ACM symposium on Theory of computing, 1978, pp. 216–226. doi:10.1145/800133.804350.
11. 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.
12. Vega, F. Note for the P versus NP Problem. *IPI Letters* **2024**, *2*, 14–18. doi:10.59973/ipil.92.