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]. Similar problems, such as the VP versus VNP problem in algebraic complexity, remain unsolved [7].

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 [8]. 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 [8].
2. **Universal Hardness:** Every problem in the class NP can be reduced to this problem without significant computational overhead [8].

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 [9].
- **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 [9].
- **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 [9].

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

Definition 1. ***Maximum Cover By 3-Sets (3MSC) Problem***

INSTANCE: A universe set U, a collection of n sets $C = S_1, \ldots, S_n$ of 3-elements sets with $S_i \subseteq U$ and a positive integer $k \leq n$, where every element in U appears exactly twice in the collection C.

QUESTION: Is there a partition of m sets $S'_1, \ldots, S'_m$ with $k \leq m$ 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: By employing matching techniques, this problem can be efficiently solved in polynomial time.

Definition 2. ***Edge-Weighted Matching (EWM) Problem***

INSTANCE: An undirected graph $G = (V, E)$ with positive weights on the edges.

ANSWER: Find a matching of maximum weight.

REMARKS: A matching is a subset of edges $E' \subseteq E$ such that no two edges in E' share a common endpoint. Solvable in polynomial time [10,11]. Here's an implementation of this problem using NetworkX [12]:

```
import networkx as nx

if __name__ == "__main__":
    # Create an undirected graph
    G = nx.Graph()

    # Define the edges with weights as a list of tuples.
    # Each tuple represents an edge: (node1, node2, {attributes}).
    # Here, the 'weight' attribute is used to assign weights to the edges.
    edges = [
        (0, 4, {'weight': 2.5}), (0, 6, {'weight': 4}),
        (1, 5, {'weight': 1}), (1, 6, {'weight': 1.5}),
        (2, 4, {'weight': 3}), (3, 4, {'weight': 2}),
    ]

    # Add the edges to the graph G. The attributes (including weights) are also added.
    G.add_edges_from(edges)

    # Calculate the maximum weight matching of the graph.
    # The result is a set of edges represented as tuples (u, v). Converting to list for
                                                    easy iteration.
    maximum_weight_matching = list(nx.max_weight_matching(G))

    # Calculate the total weight of the maximum weight matching.
    # Iterate through the edges in the matching and sum their weights.
    weight = sum(
        G.edges[u, v]['weight']  # Access the 'weight' attribute of the edge (u, v).
```

```
        for u, v in maximum_weight_matching # Iterate through the edges in the matching
                                                .
    )

    # Print the total weight of the maximum weight matching. The expected output is 8.
    print(weight)
```

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 [8]. The SAT problem asks whether a given Boolean formula in CNF is satisfiable [9]. A $3CNF$ formula is a CNF formula in which each clause contains exactly three distinct literals [8]. 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 [8]. We formally define the following problem:

Definition 3. ***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 [13]. Here, the certificate is a valid satisfying truth assignment, meaning it satisfies the NAE-3MSAT condition for each clause.

In addition, a $2CNF$ formula is a Boolean formula in conjunctive normal form with exactly two distinct literals per clause. Now, we introduce the last problem:

Definition 4. ***Maximum Monotone XOR 2-Satisfiability (2MXSAT) Problem***
INSTANCE: A Boolean formula in $2CNF$ with monotone clauses (meaning the variables are never negated) using XOR logic operators $\oplus$ (instead of using the operator $\vee$) and a positive integer k.
QUESTION: Is there exists a truth assignment that satisfies at least k clauses?

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 [9].

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 [8]. 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. □

These are key findings.

Theorem 4. *The problem NAE-3MSAT can be reduced to 2MXSAT in polynomial time.*

Proof. We reduce an instance of NAE-3MSAT to an instance of 2MXSAT, consisting of a Boolean formula ϕ and a positive integer k. For each clause $c_i = (x \vee y \vee z)$ in the NAE-3MSAT formula, we construct the following expression using new variables a_i, b_i, and d_i:

$$F_i = (a_i \oplus b_i) \wedge (b_i \oplus d_i) \wedge (a_i \oplus d_i) \wedge (x \oplus a_i) \wedge (y \oplus b_i) \wedge (z \oplus d_i).$$

If c_i satisfies the NAE-3MSAT condition (i.e., not all literals have the same truth value), then there exists a truth assignment to a_i, b_i, and d_i such that at least five clauses in F_i are satisfied. Conversely, if all literals in c_i have the same truth value, it is impossible to satisfy five clauses in F_i. The complete 2MXSAT formula ϕ is the conjunction of all F_i:

$$\phi = \bigwedge_{i=1}^{m} F_i = F_1 \wedge F_2 \wedge F_3 \wedge \ldots \wedge F_{m-1} \wedge F_m,$$

where m is the number of clauses in the original NAE-3MSAT formula.

Setting $k = 5 \cdot m$, we claim that the NAE-3MSAT formula has a valid satisfying truth assignment if and only if there is a truth assignment that satisfies at least k clauses in ϕ. This is because each NAE-satisfying clause in the original formula corresponds to at least five satisfiable clauses in the constructed formula, and vice-versa. Therefore, satisfying $k = 5 \cdot m$ clauses in ϕ guarantees a valid NAE-satisfying truth assignment for the original NAE-3MSAT formula. □

Theorem 5. *3MSC admits a polynomial-time algorithm based on matching techniques.*

Proof. Given an instance of 3MSC defined by a universe U and a collection $C = S_1, \ldots, S_n$ of 3-element sets $S_i \subseteq U$, where each element in U appears exactly twice in C, and the goal is to select at least k mutually disjoint sets from C that cover U, we construct an equivalent instance of Edge-Weighted Matching (EWM) as follows:

1. **Graph Construction:** We create a graph $G = (V, E)$:
    - **Vertices:** The vertex set V contains the elements of the universe U. For each 3-element set S_i in C, we create a corresponding vertex s_i. Besides, for each pair of non-disjoint 3-element sets S_i, S_j in C, we introduce vertices $a_{(i,j)}$ and $b_{(i,j)}$.

- **Edges:** For non-disjoint sets S_i, S_j in C and element $u = S_i \cap S_j$, construct a gadget using vertices u, s_i, s_j, $a_{(i,j)}$ and $b_{(i,j)}$:

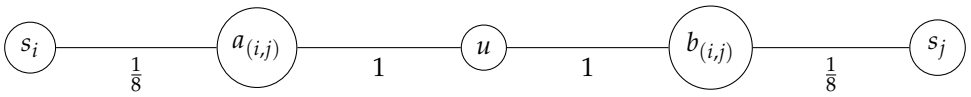

Figure 1. gadget corresponding to the element $u = S_i \cap S_j$.

2. **Maximum Weight Matching:**

   - A matching in G is a set of edges with no shared vertices.
   - A maximum weight matching is a matching with the largest possible total weight. In this construction, the weights are the positive numbers 1 and $\frac{1}{8}$.
   - The total weight of a maximum weight matching in G corresponds to the maximum number of disjoint sets of 3-elements that can be selected from C divided by 8 and the cardinality of U (i.e., denoted $\mid U \mid$). In contrast, the weight of selecting at least one element in S_i among the sum of the weights of the edges choosing S_i and the intersecting sets S_j in C is significantly smaller (i.e., $1 > \frac{1}{2}$). This fact supports the coverage of the universe U by the sets in the collection C.

3. **Correspondence to 3MSC Partition:**

   - For each gadget in the maximum weight matching, we can select one element u from the set U and exactly one of the two 3-element sets that include u, corresponding to a possible solution to the 3MSC instance. Excluding pairs of sets that both contain the same single element of the universe U ensures that the resulting solution consists entirely of disjoint sets.

The crucial observation is the bijective correspondence between selecting at least k disjoint sets in the 3MSC instance and finding a maximum weight matching of total weight at least $\frac{k}{8} + \mid U \mid$ in the constructed graph G. Because the maximum weight matching problem can be solved in polynomial time, this reduction provides a polynomial-time algorithm for 3MSC. By transforming the 3MSC problem into a maximum weight matching problem, we can leverage efficient graph algorithms to find an optimal solution in polynomial time. □

This is a Main Insight.

Theorem 6. *The problem 2MXSAT can be reduced to 3MSC in polynomial time.*

Proof. To better visualize this polynomial-time reduction, we will use a graphical representation of the sets involved. To represent a 2MXSAT formula ϕ and a positive integer k as a collection of set C over a universe U, we introduce a gadget for each variable x in ϕ. This gadget consists of $2 \cdot t$ triangles, where t 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 2) guarantees that if any positive vertex is matched with some vertices outside of the gadget then all negative vertices 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. For instance, in Figure 2, these sets would be:

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

where dashed vertices correspond to the literals of x that are absent from clauses c_i, c_j, c_h in ϕ.

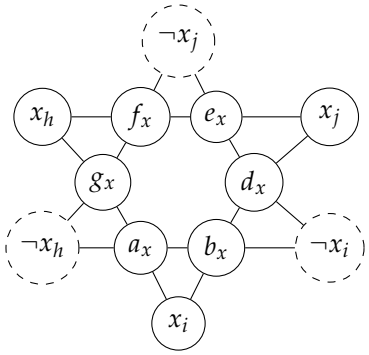

Figure 2. variable gadget for the occurrences of x in clauses c_i, c_j, c_h of ϕ.

To represent each clause $c_i = (x \oplus y)$ in ϕ, we employ a gadget consisting of four sets:

$$\{x_i, a_i, b_i\}, \{y_i, b_i, d_i\},$$
$$\{\neg y_i, d_i, e_i\}, \{\neg x_i, e_i, a_i\},$$

where dashed vertices correspond to the literals that are absent in c_i.

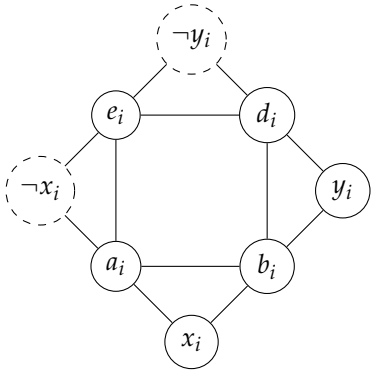

Figure 3. clause gadget corresponding to the clause $c_i = (x \oplus y)$ in ϕ.

This gadget, illustrated in Figure 3, is used to encode each clause within the overall construction. Satisfying clause c_i requires selecting exactly two disjoint sets. This is impossible if all variables in c_i are either all true or all false. Conversely, if c_i is satisfied, it guarantees that exactly two disjoint sets can be chosen.

A truth assignment satisfies at least k clauses of ϕ if and only if its corresponding set collection C can be partitioned into at least $2 \cdot (m + k)$ 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 2 enables the selection of exactly one set for each variable occurrence within a clause of ϕ. Since each clause comprises two distinct variables, there are precisely $2 \cdot m$ such sets.
2. **Clause Sets:** The final step in Figure 3 ensures clause satisfaction in ϕ by forcing the selection of exactly two sets. This ensures exactly $2 \cdot k$ clause sets induce a truth assignment satisfying k clauses of ϕ.

Hence, a truth assignment for ϕ that satisfies exactly k clauses can be directly mapped to a partition of C into precisely $2 \cdot m$ variable sets and $2 \cdot k$ clause sets. Conversely, any such partition of C can be

interpreted as a truth assignment for ϕ that satisfies exactly k clauses. This one-to-one correspondence rigorously establishes the equivalence between two distinct properties: first, the satisfiability of at least k clauses within the formula ϕ; and second, the existence of a specific partitioning of the set C into at least $2 \cdot (m + k)$ disjoint sets. This partition must completely cover the entire universe U, meaning that every element within U is contained within the union of these disjoint sets. Furthermore, a crucial constraint is imposed: each element in U must belong to precisely two of the sets within the partition of C. This bijection thus demonstrates that finding a satisfying assignment for at least k clauses in ϕ is precisely equivalent to constructing such a partition of C. □

This is the Main Theorem.

Theorem 7. $SAT \in P$.

Proof. This follows directly from Theorems 1, 2, 3, 4, 5, and 6. □

This is the definitive result.

Theorem 8. $P = NP$.

Proof. Cook's Theorem states that every NP problem can be reduced to SAT in polynomial time [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 [3,4]. 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.

Acknowledgements

The author would like to thank Iris, Marilin, Sonia, Yoselin, and Arelis for their support.

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 20, 2024.
2. Sudan, M. The P vs. NP problem. http://people.csail.mit.edu/madhu/papers/2010/pnp.pdf, 2010. Accessed December 20, 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. Wigderson, A. *Mathematics and Computation: A Theory Revolutionizing Technology and Science*; Princeton University Press, 2019.
8. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*, 3rd ed.; The MIT Press, 2009.
9. 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.
10. Karp, R.M.; Upfal, E.; Wigderson, A. Constructing a perfect matching is in random NC. *Combinatorica* **1986**, *6*, 35–48. doi:10.1007/BF02579407.
11. Greenlaw, R.; Hoover, H.J.; Ruzzo, W.L. *Limits to Parallel Computation: P-Completeness Theory*; Oxford University Press, USA, 1995.
12. Galil, Z. Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys (CSUR)* **1986**, *18*, 23–38. doi:10.1145/6462.6502.
13. 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.