Note for the P versus NP problem (II)

Frank Vega *

Posted Date: 26 September 2024

doi: 10.20944/preprints202409.2053.v1

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

 doi:10.20944/preprints202409.2053.v

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 fundamental question in computer science. It asks whether problems whose solutions can be quickly verified can also be quickly solved. Here, "quickly" refers to computational time that grows proportionally to the size of the input (polynomial time). While the problem's roots trace back to a 1955 letter from John Nash, its formalization is attributed to Stephen Cook and Leonid Levin. Despite extensive research, a definitive answer remains elusive. Closely tied to this is the concept of NP-completeness. If a single NP-complete problem could be solved efficiently, it would imply that all problems in NP can be solved efficiently, proving that P equals NP. Our work presents a polynomial-time algorithm for the CLOSURE problem. We implemented a Python-based solution for this polynomial time algorithm, which is available on GitHub under the username "frankvegadelgado". In addition, we propose that CLOSURE is actually an NP-complete problem, which would imply that P equals 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; Boolean formula; graph; completeness; polynomial time

MSC: *2020 Mathematics Subject Classification*. Primary 68Q15; Secondary 68Q17, 68Q25

---

1. Introduction

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

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

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

2. Background and Ancillary Results

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

 doi:10.20944/preprints202409.2053.v

(1) **Efficient Verifiability:** Solutions can be swiftly checked using a concise proof.
(2) **Universal Hardness:** Every problem in the class NP can be transformed into an instance of this problem without significant computational overhead [9].

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

Illustrative examples of NP-complete problems include:

- **Boolean satisfiability (SAT)**: Given a logical expression, determine if there exists an assignment of truth values to its variables that makes the entire expression true [10].
- **K-CLOSURE problem**: Given a directed graph $G = (V, A)$ (V is the set of vertices and A is the set of edges) and positive integer k, determine whether there is a set V' of at most k vertices such that for all $(u, v) \in A$ either $u \in V'$ or $v \notin V'$ (Please refer to [Queyranne, 1976] in the Johnson and Garey book) [10]. Note that in this problem, the phrase "either $u \in V'$ or $v \notin V'$" is equivalent to either "($u \in V'$ and $v \in V'$) or ($u \notin V'$ and $v \notin V'$)". This is because the logical implication of the phrase "**either ... or ...**" requires that exactly one of the two conditions be true.

The provided examples represent a small subset of the extensively studied NP-complete problems relevant to our current work. 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. Determining whether a given graph is a bipartite graph can be accomplished efficiently (in polynomial time) [9]. 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.

Definition 2.1. Independent Vertex in Bipartite Graph
INSTANCE: A bipartite graph $B = (U, V, E)$ and a positive integer k.
QUESTION: Is there set V' of at least k vertices such that V' is an independent set in B?
REMARKS: This problem can be easily solved in polynomial time [10].

Formally, an instance of **Boolean satisfiability problem (SAT)** 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 set of values for the variables in ϕ. A satisfying truth assignment is a truth assignment that causes ϕ to be evaluated as true. A Boolean formula with a satisfying truth assignment is satisfiable. The problem SAT asks whether a given Boolean formula is satisfiable [10].

We define a CNF Boolean formula using the following terms: A literal in a Boolean formula is an occurrence of a variable or its negation [9]. A Boolean formula is in conjunctive normal form, or CNF, if it is expressed as an AND of clauses, each of which is the OR of one or more literals [9]. A Boolean formula is in 2-conjunctive normal form or $2CNF$, if each clause has exactly two distinct literals [9].

For example, the Boolean formula:

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

is in $2CNF$. The first of its three clauses is $(x_1 \vee \neg x_2)$, which contains the two literals x_1 and $\neg x_2$. We define the following problem:

 doi:10.20944/preprints202409.2053.v

Definition 2.2. Monotone Weighted Xor 2-satisfiability problem (MWX2SAT)

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 at least k of the variables are true?

Finally, we introduce the last problem:

Definition 2.3. CLOSURE

INSTANCE: A directed graph $G = (V, A)$ and a positive integer k.

QUESTION: Is there set V' of at least k vertices such that for all $(u, v) \in A$ either $u \in V'$ or $v \notin V'$?

REMARKS: The *CLOSURE* problem is NP-complete. This follows from the fact that it is equivalent to the NP-complete *K-CLOSURE* problem, where the target closure size is simply complemented by reversing the direction of all edges in the graph.

By presenting an efficient solution to *CLOSURE*, we would establish a proof that P equals NP.

3. Main Result

This is a main insight.

Theorem 3.1. *The problem CLOSURE can be reduced to MWX2SAT in polynomial time [11].*

Proof. We can build an equivalent $MWX2SAT$ instance for any given instance $G = (V, A)$ of the *CLOSURE* problem:

(1) Variables:

- Create a variable for each vertex v in the original graph G. Denote this variable as v itself.
- For each edge (u, v) in G, introduce two new variables denoted by x_{uv} and x_{vu}.

(2) Clauses:

- For each edge (u, v) in G, construct three clauses using the new variables:
  - $(u \oplus x_{uv})$: This enforces that either vertex 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 vertex v is true (XOR).
  - $(x_{vu} \oplus x_{uv})$: This guarantees that x_{uv} and x_{vu} have different truth values. Note that x_{vu} is not used elsewhere, so it only enforces there is exactly one true variable per each edge (u, v) over the new variables x_{uv} and x_{vu}.

Key Points about the Construction:

- The first two clauses for each edge (u, v) ensures that both variables u and v for an edge have the same truth value. This is because they represent the "state" of the edge (both in the closure or both outside). By definition, a vertex closure cannot have any outgoing edges pointing to vertices outside the closure. Therefore, no edge can exist where one vertex belongs to the solution and the other does not.
- The third clause for each edge (u, v) together ensure that exactly one of x_{uv} or x_{vu} is true in a satisfying truth assignment. Take into account this condition enforces always a true variable for each edge for every possible satisfying truth assignment.

Mapping between CLOSURE solutions and MWX2SAT assignments:

- A satisfying truth assignment in the $MWX2SAT$ formula corresponds to a valid closure of at least k vertices in the original graph G if:

 doi:10.20944/preprints202409.2053.v

- Vertices assigned true represent the vertices in the closure V'.
- New variable x_{uv} assigned true represents that the corresponding edge (u, v) has both endpoints outside the closure.
- New variable x_{vu} assigned true indicates that the corresponding edge (u, v) has both endpoints within the closure.

Why this construction works:

- The clauses enforce that a satisfying truth assignment must have consistent values for a vertex and its corresponding edge variables.
- A k $>$-vertex closure property translates to k original variables (vertices) being true in the satisfying truth assignment, along with exactly one true variable from the pair of new variables x_{uv} and x_{vu} per each edge depending on the specific closure.

Equivalence and Complexity:

- There exists a satisfying truth assignment for the $MWX2SAT$ formula with at least $k+ \mid A \mid$ true variables if and only if there exists a closure of at least k vertices in the original graph. ($\mid A \mid$ represents the number of edges in the graph).
- Since *CLOSURE* is known to be NP-complete, this shows that $MWX2SAT$ is also NP-complete.

In essence, the proof demonstrates that solving $MWX2SAT$ is equivalent to finding a closure of at least k vertices in the *CLOSURE* problem. This implies that $MWX2SAT$ inherits the NP-completeness property from *CLOSURE*. □

This is the main theorem.

Theorem 3.2. $MWX2SAT \in P$ [11].

Proof. There is a connection between finding a satisfying truth assignment in $MWX2SAT$ with at least k true variables and finding a set of at least k vertices that is 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 $MWX2SAT$ 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. $MWX2SAT$ and the Graph:
   - A truth assignment in $MWX2SAT$ where at least k variables are true directly translates to a set of at least k vertices in the constructed graph where true variables correspond to the vertices included in the set.
   - The properties of $MWX2SAT$ clauses ensure that:
     - 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.
     - Bipartite Graph: The Boolean formula in $MWX2SAT$ is satisfiable if and only if the corresponding graph is bipartite.

 doi:10.20944/preprints202409.2053.v

Therefore, finding a satisfying truth assignment with at least k true variables in $MWX2SAT$ is indeed equivalent to finding a set of at least k vertices that fulfills an independent set requirements in the corresponding bipartite graph. However, we know the problem of finding a set of at least k vertices that is an independent set in a bipartite graph can be easily solved in polynomial time [10]. Additionally, verifying if the constructed graph is indeed a bipartite graph can be done in polynomial time [9]. Consequently, the instances of the problem $MWX2SAT$ can be solved in polynomial time as well. □

This is a key finding.

Theorem 3.3. $P = NP$.

Proof. A polynomial-time solution to any NP-complete problem would establish the equivalence of P and NP [9]. Given that $CLOSURE$ is an NP-complete problem, a polynomial-time solution for it, as presented here, would directly imply P equals NP. □

4. Discussion

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' [10]. In general, V'' is an independent set of an arbitrary undirected graph $G = (V, E)$ if and only if $V - V''$ is a vertex cover of G [10]. The **Independent Vertex Set in Bipartite Graph** problem is equivalent to finding a vertex cover in a bipartite graph $B = (U, V, E)$ using at most $(\mid U \mid + \mid V \mid -k)$ vertices. We implemented a Python-based solution for this algorithm, which is available on GitHub under the username "frankvegadelgado" [12].

5. Conclusion

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. 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].

 doi:10.20944/preprints202409.2053.v

- **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].

Acknowledgments: Many thanks to Sergi Simon, Jorge Félix and Melvin Vopson 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 September 19, 2024.
2. Sudan, M. The P vs. NP problem. http://people.csail.mit.edu/madhu/papers/2010/pnp.pdf, 2010. Accessed September 19, 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. Vega, F. Note for the P versus NP Problem. *IPI Letters* **2024**, *2*, 14–18. doi:10.59973/ipil.92.
12. Vega, F. ALMA | MWX2SAT Solver. https://github.com/frankvegadelgado/alma, 2024. Accessed September 19, 2024.