IPI Letters

ISSN 2976 - 730X
IPI Letters 2024,Vol 2 (2):14-18
https://doi.org/10.59973/ipil.92

Received: 2024-05-28
Accepted: 2024-06-05
Published: 2024-06-07

Article

Note for the P versus NP Problem

Frank Vega[1,*]

[1]Information Physics Institute, Miami, Florida, United States

[*]Corresponding author: vega.frank@gmail.com

Abstract - P versus NP is considered as one of the most fundamental open problems in computer science. This consists in knowing the answer of the following question: Is P equal to NP? It was essentially mentioned in 1955 from a letter written by John Nash to the United States National Security Agency. However, a precise statement of the P versus NP problem was introduced independently by Stephen Cook and Leonid Levin. Since that date, all efforts to find a proof for this problem have failed. Another major complexity class is NP-complete. It is well-known that P is equal to NP under the assumption of the existence of a polynomial time algorithm for some NP-complete. We show that the Monotone Weighted Xor 2-satisfiability problem (MWX2SAT) is NP-complete and P at the same time. Certainly, we make a polynomial time reduction from every directed graph and positive integer k in the K-CLOSURE problem to an instance of MWX2SAT. In this way, we show that MWX2SAT is also an NP-complete problem. Moreover, we create and implement a polynomial time algorithm which decides the instances of MWX2SAT. Consequently, we prove that P = NP.

Keywords - Complexity classes; Boolean formula; Graph; Completeness; Polynomial time.

1 Introduction

P versus *NP* is one of the most important and challenging problems in computer science [1]. It asks whether every problem whose solution can be quickly verified can also be quickly solved. The informal term "quickly" here refers to the existence of an algorithm that can solve the task in polynomial time [1]. The general class of problems for which such an algorithm exists is called *P* or "class *P*"[1].

Another class of problems called *NP*, which stands for "nondeterministic polynomial time", is defined by the property that if an input to a problem is a solution, then it can be quickly verified [1]. The *P* versus *NP* problem asks whether *P* equals *NP*. If it turns out that $P \neq NP$, which is widely believed to be the case, it would mean that there are problems in *NP* that are harder to compute than to verify [1]. This would have profound implications for various fields, including cryptography and artificial intelligence [2].

Solving the *P* versus *NP* problem is considered to be one of the greatest challenges in computer science [1]. A solution would have a profound impact on our understanding of computation and could lead to the development of new algorithms and techniques that could solve many of the world's most pressing problems [1]. The problem is so difficult that it is considered to be one of the seven Millennium Prize Problems, which are a set of seven unsolved problems that have been offered a 1 million prize for a correct solution [1].

2 Materials and methods

NP-complete problems are a class of computational problems that are at the heart of many important and challenging problems in computer science. They are defined by the property that they can be quickly verified, but there is no known efficient algorithm to solve them. This means that finding a solution to an *NP*-complete problem can be extremely time-consuming, even for relatively small inputs. In computational complexity theory, a problem is considered *NP*-complete if it meets the following two criteria:

1. **Membership in NP**: A solution to an *NP*-complete problem can be verified in polynomial time. This means that there is an algorithm that can quickly check whether a proposed solution is correct [3].

2. **Reduction to NP-complete problems**: Any problem in *NP* can be reduced to an *NP*-complete problem in polynomial time. This means that any *NP*-problem can be transformed into an *NP*-complete problem by making a small number of changes [3].

If it were possible to find an efficient algorithm for solving any one *NP*-complete problem, then this algorithm could be used to solve all *NP* problems in polynomial time. This would have a profound impact on many fields, including cryptography, artificial intelligence, and operations research [2]. Here are some examples of *NP*-complete problems:

- **Boolean satisfiability problem (SAT)**: Given a Boolean formula, determine whether there is an assignment of truth values to the variables that makes the formula true [4].

- **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 $v \notin V'$ (see reference [Queyranne, 1976] from the Johnson and Garey book) [4]. Note that in this problem the phrase "either $u \in V'$ or $v \notin V'$" does mean the same as ($u \in V'$ and $v \in V'$) or ($u \notin V'$ and $v \notin V'$) since the logical implication of the words "**either ... or ...**" indicates that exactly one of the following statements can be true.

These are just a few examples of the many *NP*-complete problems that have been studied and have a close relation with our current result. On the one hand, 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'. On the other hand, 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 Cover***
INSTANCE: An undirected graph $G = (V, E)$ and a positive integer k.
QUESTION: Is there set V' of at most k vertices such that V' is both a vertex cover and an independent set in G?
REMARKS: This problem can be easily solved in polynomial time [4].

In this work, we show there is an *NP*-complete problem that can be solved in polynomial time using the previous problem. Consequently, we prove that *P* is equal to *NP*.

3 Results

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), $\rightarrow$(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 [4].
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 [3]. 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 [3]. A Boolean formula is in 2-conjunctive normal form or 2*CNF*, if each clause has exactly two distinct literals [3]. For example, the Boolean formula:

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

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

Definition 3.1. ***Monotone Weighted Xor 2-satisfiability problem (MWX2SAT)***
INSTANCE: An n-variable 2*CNF 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 most k of the variables are true?

The following is key Lemma.

Lemma 3.2. *MWX2SAT* $\in$ *NP–complete.*

Proof. We can build an equivalent *MWX2SAT* instance for any given instance $G = (V, A)$ of the *K-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 k-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 K-CLOSURE solutions and MWX2SAT assignments:

- A satisfying truth assignment in the *MWX2SAT* formula corresponds to a valid closure of at most k vertices in the original graph G if:
  - 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.
- The 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 most $k + |A|$ true variables if and only if there exists a closure of at most k vertices in the original graph. ($|A|$ represents the number of edges in the graph).
- Since *K-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 most k vertices in the *K-CLOSURE* problem. This implies that $MWX2SAT$ inherits the *NP*-completeness property from *K-CLOSURE*. □

This is the main theorem.

Theorem 3.3. $MWX2SAT \in P$.

Proof. There is a connection between finding a satisfying truth assignment in $MWX2SAT$ with at most k true variables and finding a set of at most 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 original 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 most k variables are true directly translates to a set of at most k vertices in the constructed graph where true variables correspond to the vertices included in the set.
   - The properties of $MWX2SAT$ 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 at most k true variables in $MWX2SAT$ is indeed equivalent to finding a set of at most 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 at most k vertices that is both a vertex cover and an independent set can be easily solved in polynomial time [4]. Consequently, the instances of the problem $MWX2SAT$ can be solved in polynomial time as well. □

4 Discussion

In the definition of the **K-CLOSURE problem**, which is the original *NP*-complete, it is important to notice that is used the words "**either ... or ...**" which is equivalent to the XOR Boolean operation in most cases when it is used in English. There are contexts in which is used as OR, but we check out the whole reference paper [4] and the authors Garey and Johnson always used that statement in the book as the XOR Boolean operation. Finally, we reduce our principal problem to the **Independent Vertex Cover** problem which in the same book states that "can be easily solved in polynomial time" in the page 190 [4]. The **Independent Vertex Cover** problem (where the solution should be simultaneously a vertex cover and independent set inside of an undirected graph) is the same as solving the 2-colorable problem in a bipartite graph using at most k vertices in one of the bipartition set. Indeed, it is quite similar to the 2-colorable algorithm on bipartite graph, but we only need to additionally introduce a trivial greedy algorithm. We create a software programming implementation in Python for this algorithm (this code in Python would be outside of the necessary correctness of the paper and thus, this can only be considered as an appendix that will not compromise the whole result)[5]. This is placed into a GitHub repository under my GitHub username "frankvegadelgado"[5]. The last commit was on March 19th of 2024 with a SHA commit **bd2d963b34a5b01b7c78dc602aa8f01acd6273f5** [5].

5 Conclusion

A proof of $P = NP$ will have stunning practical consequences, because it possibly leads to efficient methods for solving some of the important problems in computer science [1]. The consequences, both positive and negative, arise since various *NP*-complete problems are fundamental in many fields [2]. But such changes may pale in significance compared to the revolution an efficient method for solving *NP*-complete problems will cause in mathematics itself [1]. Research mathematicians spend their careers trying to prove theorems, and some proofs have taken decades or even centuries to be discovered after problems have been stated [1]. A method that guarantees to find proofs for theorems, should one exist of a "reasonable" size, would essentially end this struggle [1].

Acknowledgements

Many thanks to Sergi Simon, Jorge Félix and Melvin Vopson for their support.

References

[1] Stephen Arthur Cook. The P versus NP Problem, Clay Mathematics Institute. https://www.claymath.org/wp-content/uploads/2022/06/pvsnp.pdf, June 2022. Accessed 25 May 2024.

[2] Lance Fortnow. The status of the P versus NP problem. Communications of the ACM, 52(9):78–86,2009. doi:10.1145/1562164.1562186.

[3] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Introduction to Algorithms. The MIT Press, 3rd edition, 2009.

[4] Michael R Garey and David S Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. San Francisco: W. H. Freeman and Company, 1 edition, 1979.

[5] Frank Vega. ALMA— MWX2SAT Solver. https://github.com/frankvegadelgado/alma, Febru- ary 2024. Accessed 25 May 2024.