Note for the P versus NP Problem

Frank Vega

NataSquad, 10 rue de la Paix 75002 Paris, France

Abstract

P versus *NP* is considered as one of the most important 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 2-satisfiability problem (MW2SAT) is *NP*-complete and *P* at the same time.

Keywords: computational algorithm, complexity classes, completeness, polynomial time, reduction
2000 MSC: 68Q15, 68Q17, 68Q25

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].

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].

Email address: vega.frank@gmail.com (Frank Vega)

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].
- **Subset sum problem**: Given a set of positive integers and target T, determine whether there is a subset of the integers which sum to precisely T [4].
- **Vertex cover problem**: Given an undirected graph and positive integer k, determine whether there is a set of exactly k vertices that together touch all the edges of the given undirected graph [4].

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

2. Summary of the Main 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 $2CNF$, if each clause has exactly two distinct literals [3].

For example, the Boolean formula:

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

is in $2CNF$. The first of its three clauses is $(x_1 \vee \rightarrow x_1)$, which contains the two literals x_1 and $\rightarrow x_1$.

In the **Weighted 2-satisfiability problem (W2SAT)**, the input is an n-variable $2CNF$ instance and an integer k, and the problem is to decide whether there exists a satisfying truth assignment in which exactly k of the variables are true. We define another variant:

Definition 2.1. ***Monotone Weighted 2-satisfiability problem (MW2SAT)***
INSTANCE: An n-variable 2CNF instance and an integer k, where all clauses are monotone (meaning that variables are never negated)
QUESTION: Is there exists a satisfying truth assignment in which exactly k of the variables are true?

We state the following result:

Lemma 2.2. *$MW2SAT \in NP$–complete.*

Finally, we deduce our main goal:

Theorem 2.3. *$MW2SAT \in P$.*

3. Main Results

3.1. Proof of Lemma 2.2

Proof. For any given instance of the vertex cover problem, one can construct an equivalent $MW2SAT$ problem with a variable for each vertex of a graph. Each edge (u, v) of the graph may be represented by a $2CNF$ clause $(u \vee v)$ that can be satisfied only by including either u or v among the true variables of the solution. Then the satisfying instances of the resulting $2CNF$ formula encode solutions to the vertex cover problem, and there is a satisfying assignment with k true variables if and only if there is a vertex cover with k vertices. Therefore, like vertex cover, $MW2SAT$ is NP-complete. □

3.2. Proof of Theorem 2.3

Proof. Suppose we have the following sequence of variables in a given instance of $MW2SAT$:

$$x_1, \ldots, x_n.$$

For each variable x_i in the instance, we define the functions f and g for the following cases:

1. If the variable x_i is assigned as true, then $f(x_i, true)$ is the number of variables that should be necessarily true while $f(x_i, false)$ is the number of variables that should be necessarily false. It is trivial to see that $f(x_i, true) = 1$ and $f(x_i, false) = 0$.
2. If the variable x_i is assigned as false, then $g(x_i, true)$ is the number of variables x_j that should be necessarily true such that $j > i$ while $g(x_i, false)$ is the number of variables that should be necessarily false. It is trivial to see that $g(x_i, false) = 1$ which is the variable x_i itself. Moreover, $g(x_i, true)$ is the number of clauses in the form of $(x_i \vee x_j)$ or $(x_j \vee x_i)$ whenever $j > i$ (Note that, this value could be 0).

We define a state as a pair (i, s, r) of integers. This state represents the fact that during an evaluation of the variables,

"*there is a nonempty subset of variables such that* $x_1, \ldots, x_i$ *sums to* s *true variables and sums to* r *false variables*."

Each state (i, s, r) has two next states:

- $(i + 1, s + f(x_{i+1}, true), r + f(x_{i+1}, false))$, implying that x_{i+1} is included in the subset and it is evaluated as true;
- $(i + 1, s + g(x_{i+1}, true), r + g(x_{i+1}, false))$, implying that x_{i+1} is included in the subset and it is evaluated as false.

Starting from the initial state $(0, 0, 0)$, it is possible to use any graph search algorithm (e.g. **Breadth-first search (BFS)** [3]) to search the state $(n, k, n - k)$. The run-time of this algorithm is at most linear in the number of states. The number of states is at bounded by n^2 times which is the number of different possible ways for which two numbers could sum the value of n. Since all values of i are positive and bounded by n, then the whole time required is $O(n^3)$. □

4. Conclusion

No one has been able to find a polynomial time algorithm for any of more than 300 important known NP-complete problems [4]. 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]. A polynomial solution for any NP-complete problem will imply a feasible solution to every problem in NP and thus, P would be equal to NP [3].

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]. For instance, Fermat's Last Theorem took over three centuries to be proved [1]. A method that guarantees to find proofs for theorems, should one exist of a "reasonable" size, would essentially end this struggle [1].

References

[1] S. A. Cook, The P versus NP Problem, https://www.claymath.org/wp-content/uploads/2022/06/pvsnp.pdf, Clay Mathematics Institute. Accessed 9 September 2023 (June 2022).

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

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 3rd Edition, The MIT Press, 2009.

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