NP versus PSPACE

Frank Vega

NP versus PSPACE

Frank Vega

Abstract

The P versus NP problem is one of the most important and unsolved problems in computer science. This consists in knowing the answer of the following question: Is P equal to NP? This incognita was first mentioned in a letter written by Kurt Gödel to John von Neumann in 1956. However, the precise statement of the P versus NP problem was introduced in 1971 by Stephen Cook in a seminal paper. Another major complexity class is PSPACE. Whether P = PSPACE is another fundamental question that it is as important as it is unresolved. All efforts to find polynomial-time algorithms for the PSPACE-complete problems have failed. We shall prove the existence of a problem in NP and PSPACE-complete. Since, PSPACE is closed under reductions and NP is contained in PSPACE, then we have that NP = PSPACE.

Keywords: P, NP, PSPACE, PSPACE-complete, GEOGRAPHY
2000 MSC: 68-XX, 68Qxx, 68Q15

1. Introduction

The *P* versus *NP* problem is a major unsolved problem in computer science. This problem was introduced in 1971 by Stephen Cook [1]. It is considered by many to be the most important open problem in the field [2]. It is one of the seven Millennium Prize Problems selected by the Clay Mathematics Institute to carry a US$1,000,000 prize for the first correct solution.

The argument made by Alan Turing in the twentieth century states that for any algorithm we can create an equivalent Turing machine [3]. There are some definitions related with this model such as the deterministic or nondeterministic Turing machine. A deterministic Turing machine has only one next action for each step defined in its program or transition function [4]. A nondeterministic Turing machine can contain more than one action defined for each step of the program, where this program is not a function, but a relation [4].

Another huge advance in the last century was the definition of a complexity class. A language *L* over an alphabet is any set of strings made up of symbols from that alphabet [5]. A complexity class is a set of problems, which are represented as a language, grouped by measures such as the running time, memory, etc [5].

In computational complexity theory, the class *P* consists in all those decision problems (defined as languages) that can be decided on a deterministic Turing machine in an amount of time that is polynomial in the size of the input; the class *NP* consists in all those decision problems whose positive solutions can be verified in polynomial-time given the right information, or equivalently, that can be decided on a nondeterministic Turing machine in polynomial-time [6]. On

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

the other hand, $PSPACE$ is the class of all languages recognizable by polynomial space bounded deterministic Turing machines that halt on all inputs [7].

The biggest open question in theoretical computer science is the following one:

Is P equal to NP?

There is another important complexity class called *PSPACE-complete* [7]. A language L is *PSPACE-complete* if L is in $PSPACE$, and every $PSPACE$ problem can be reduced in polynomial-time to L [7]. We shall define a new problem called *ODDPATH-HORNUNSAT*. We shall show this problem is NP and *PSPACE-complete*. Since, $PSPACE$ is closed under reductions and $NP \subseteq PSPACE$, then we have that $NP = PSPACE$ [4].

2. Theoretical framework

2.1. The SAT problem

We say that a language L_1 is polynomial-time reducible to a language L_2, written $L_1 \leq_p L_2$, if there exists a polynomial-time computable function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

$$x \in L_1 \textit{ if and only if } f(x) \in L_2.$$

There is an important complexity class called *NP-complete* [6]. A language $L \subseteq \{0,1\}^*$ is *NP-complete* if

- $L \in NP$, and
- $L' \leq_p L$ for every $L' \in NP$.

Furthermore, if L is a language such that $L' \leq_p L$ for some $L' \in$ *NP-complete*, then L is *NP-hard* [5]. Moreover, if $L \in NP$, then $L \in$ *NP-complete* [5].

One of the first discovered *NP-complete* problems was SAT [7]. An instance of SAT is a Boolean formula ϕ which is composed of

- Boolean variables: $x_1, x_2, \ldots$;
- 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); and
- parentheses.

A truth assignment for a Boolean formula ϕ is a set of values for the variables of ϕ, and a satisfying truth assignment is a truth assignment that causes it to evaluate to true. A formula with a satisfying truth assignment is a satisfiable formula. The SAT asks whether a given Boolean formula is satisfiable.

One convenient language is $3CNF$ satisfiability, or $3SAT$ [5]. We define $3CNF$ satisfiability using the following terms. A literal in a Boolean formula is an occurrence of a variable or its negation. 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. A Boolean formula is in 3-conjunctive normal form, or $3CNF$, if each clause has exactly three distinct literals.

For example, the Boolean formula

$$(x_1 \vee \rightharpoonup x_1 \vee \rightharpoonup x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\rightharpoonup x_1 \vee \rightharpoonup x_3 \vee \rightharpoonup x_4)$$

is in $3CNF$. The first of its three clauses is $(x_1 \vee \rightharpoonup x_1 \vee \rightharpoonup x_2)$, which contains the three literals x_1, $\rightharpoonup x_1$, and $\rightharpoonup x_2$. In $3SAT$, we are asked whether a given Boolean formula ϕ in $3CNF$ is satisfiable.

2.2. The HORNSAT problem

We say that a language L_1 is logarithmic-space reducible to a language L_2, written $L_1 \leq_{log} L_2$, if there exists a logarithmic-space computable function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

$$x \in L_1 \textit{ if and only if } f(x) \in L_2.$$

The logarithmic space reduction is frequently used for P and below [4].

There is an important complexity class called *P-complete* [4]. A language $L \subseteq \{0,1\}^*$ is *P-complete* if

- $L \in P$, and
- $L' \leq_{log} L$ for every $L' \in P$.

One of the *P-complete* problems is $HORNSAT$ [4]. We say that a clause is a Horn clause if it has at most one positive literal [4]. That is, all its literals, except possibly for one, are negations of variables. An instance of $HORNSAT$ is a Boolean formula ϕ in CNF which is composed only of Horn clauses [4].

For example, the Boolean formula

$$(\rightharpoonup x_2 \vee x_3) \wedge (\rightharpoonup x_1 \vee \rightharpoonup x_2 \vee \rightharpoonup x_3 \vee \rightharpoonup x_4) \wedge (x_1)$$

is a conjunction of Horn clauses. The $HORNSAT$ asks whether an instance of this problem is satisfiable [4].

2.3. Directed graph notions

A directed graph (or digraph) G is a pair (V, E), where V is a finite set and E is a binary relation on V [5]. The set V is called the vertex set of G, and its elements are called vertices (singular: Vertex) [5]. The set E is called the edge set of G, and its elements are called edges [5].

If (u, v) is an edge in a directed graph $G = (V, E)$, we say that (u, v) is outgoing from or leaves vertex u and is income to or enters vertex v. If (u, v) is an edge in a graph $G = (V, E)$, we say that vertex v is adjacent to vertex u [5]. In a directed graph, the adjacency relation is not necessarily symmetric [5]. If v is adjacent to u in a directed graph, we sometimes write $u \rightarrow v$.

A path of length k from a vertex u to a vertex u' in a graph $G = (V, E)$ is a sequence of vertices $< v_0, v_1, v_2, ..., v_k >$ such that $u = v_0$, $u' = v_k$, and $(v_{i-1}, v_i) \in E$ for $i = 1, 2, ..., k$ [5]. The length of the path is the number of edges in the path [5]. The path contains the vertices $v_0, v_1, ..., v_k$ and the edges $(v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)$ [5]. If there is a path p from u to u', we say that u' is reachable from u via p [5]. A path is simple if all vertices in the path are distinct [5].

2.4. The GEOGRAPHY problem

A language $L \subseteq \{0,1\}^*$ is *PSPACE-complete* if

- $L \in PSPACE$, and
- $L' \leq_p L$ for every $L' \in PSPACE$.

Perhaps one the most fundamental problem for $PSPACE$ is the game of $GEOGRAPHY$ [4]. $GEOGRAPHY$ is an elementary-school game played by two players, called here "$P1$" and "$P2$" [4]. We can formulate this games as follows: We have a directed graph $G = (V, E)$ whose nodes are all the cities of the world, and such that there is an edge from city i to city j if and only if the last letter in the name of i coincides with the first letter in the name of j [4]. Player $P1$ starts picking a node 1, then player $P2$ picks another node to which there is an edge from 1, say node 2. Player $P1$ then must reply picking a node to which there is an edge from 2 without taking the used nodes 1 and 2. In this way, with the player alternating, it is defined a simple path from G. The first player that cannot continue the path because all edges out of the current tip lead to nodes already used, loses [4].

We can generalize this to any given directed graph G; this generalization may imply not only a planet with arbitrarily many cities, but, even less realistically, one with an arbitrarily large alphabet [4]. Hence, the $GEOGRAPHY$ would be the following computational problem: Given a directed graph G and a starting node 1, is it a win for $P1$? It is a proved result that the $GEOGRAPHY$ belongs to *PSPACE-complete* [4].

3. Results

Definition 3.1. *A sat-graph, written SAT-$G = (V, E, \kappa)$, is a directed graph $G = (V, E)$ with a mapping function κ, such that κ maps each vertex $u \in V$ to a Boolean formula.*

Definition 3.2. *Given a sat-graph SAT-$G = (V, E, \kappa)$ and a starting node u, the problem called as ODDPATH-HORNUNSAT consists in deciding whether exists a simple path of vertices $< v_0, v_1, v_2, ..., v_k >$ such that $u = v_0$, $\kappa(v_0) \wedge \kappa(v_1) \wedge \kappa(v_2) \wedge ... \wedge \kappa(v_k) \in HORNUNSAT$, and the length of the path is odd.*

Note: See the definition of simple and length of a graph path in section 2. $HORNUNSAT$ would be the complement language of $HORNSAT$, that is, the instances of $HORNSAT$ which are unsatisfiable (see section 2).

Theorem 3.3. *ODDPATH-HORNUNSAT $\in NP$.*

Proof. Given a sat-graph *SAT-$G = (V, E, \kappa)$* and a starting node u, we can check in polynomial-time whether a path $p =< v_0, v_1, v_2, ..., v_k >$ is a certificate of this instance just verifying that $u = v_0$, checking that all vertices in the path are distinct, checking that the length of the path is odd, and verifying in polynomial-time whether $\kappa(v_0) \wedge \kappa(v_1) \wedge \kappa(v_2) \wedge ... \wedge \kappa(v_k) \in HORNUNSAT$ since $HORNUNSAT \in P$ due to P is closed under complement [4]. Consequently, there is a polynomial-time verifier for *ODDPATH-HORNUNSAT*, and thus, *ODDPATH-HORNUNSAT* $\in NP$ [4]. □

Theorem 3.4. *ODDPATH-HORNUNSAT $\in$ PSPACE-complete.*

Proof. $ODDPATH\text{-}HORNUNSAT \in PSPACE$, because $NP \subseteq PSPACE$ [4]. Given a directed graph G and a starting node u, we will create a sat-graph $SAT\text{-}G$, such that $(G, u) \in GEOGRAPHY$ if and only if $(SAT\text{-}G, u) \in ODDPATH\text{-}HORNUNSAT$. For this purpose, we will do the following actions:

(1) First, we take the directed graph $G = (V, E)$ and for each vertex $v \in V$, we create a new state v' and add an edge $v \rightarrow v'$. We will call the vertex v' as the clone vertex of v. In this way, we create a new graph $G' = (V', E')$.
(2) Next, for each vertex $v \in V'$ in the graph $G' = (V', E')$, we create a new Boolean variable x_v which will be linked to vertex v. We say that x_v is represented by the vertex v.
(3) After that, we create a mapping function κ, such that for each vertex $v \in V'$ in the graph $G' = (V', E')$, we have $\kappa(v) = x_v$ if v has not outgoing edges or $\kappa(v) = x_v \wedge (\rightharpoonup x_{v_1} \vee \rightharpoonup x_{v_2} \rightharpoonup x_{v_3} \vee ... \vee \rightharpoonup x_{v_m})$ when v has $m > 0$ outgoing edges, where x_v is represented by the vertex v and $x_{v_1}, x_{v_2}, x_{v_3}, ..., x_{v_m}$ are represented by the vertices $v_1, v_2, ..., v_m$ which are all the vertices that are adjacent to vertex v (see section 2 for definition of the adjacency relation). We will call this clause of all negated variables inside of $\kappa(v)$ as the adjacency clause of v.
(4) Finally, we obtain a sat-graph $SAT\text{-}G = (V', E', \kappa)$.

All these steps can be done in polynomial-time just iterating through the vertices and edges of G. In the step (1), we need to iterate through the set of vertices V to add new $|V|$ vertices and edges. The algorithm of insertion of a polynomial amount of new vertices and edges into a graph will take only a polynomial-time [5]. In the step (2), we need to iterate through the set of vertices V' to create the Boolean variables of each vertex. This will only need a polynomial-time since $|V'| = 2 \times |V|$. Finally, in the step (3), we need to iterate through the set of vertices V' and edges of E' to create the mapping function κ. This would take a polynomial-time in relation to $|V|$.

Now, suppose we take a simple path of vertices $p =< v_0, v_1, v_2, ..., v_k >$ in $SAT\text{-}G$, such that $u = v_0$ and $\phi = \kappa(v_0) \wedge \kappa(v_1) \wedge \kappa(v_2) \wedge ... \wedge \kappa(v_k)$. We can see the inner formula $x_{v_0} \wedge x_{v_1} \wedge x_{v_2} \wedge ... \wedge x_{v_k}$ inside of ϕ is satisfiable, and therefore, whether ϕ is in $HORNUNSAT$ or not depends principally of the adjacency clauses of vertices in p. Indeed, $\phi \in HORNUNSAT$ if and only if there is an adjacency clause of some vertex v_i inside of ϕ which has all its negated Boolean variables represented by a vertex in p. Moreover, this will only happen in the adjacency clause of v_{k-1}, because in the other vertices of the path the respective adjacency clauses can be true, because the Boolean variables represented by the respective clone vertices can be false. Certainly, if $\phi \in HORNUNSAT$, then v_k (the last vertex in the path p) should necessarily be a clone vertex. The reason is this one: If the path p does not contain a clone vertex, then it would be impossible that the adjacency clause of some vertex v_i (where p contains v_i) will be false for all truth assignment of ϕ, because we could always make true the adjacency clauses of vertices in p just evaluating the Boolean variables represented by the respective clone vertices in false. At the same time, it will be impossible that the path p could have a clone vertex different of the last one, because the clone vertices does not have outgoing edges.

But, what does this mean?

It would mean, we cannot reach any adjacent vertex from vertex v_{k-1} that is not already visited in the path, except for its clone vertex. But, this is exactly what happens when a player loses in the game of $GEOGRAPHY$. In addition, the path $p' =< v_0, v_1, v_2, ..., v_{k-1} >$ represents a winner path from the starting node u in the game of $GEOGRAPHY$, where the winner is player $P1$ if the length of p' is even, else the winner is player $P2$. Therefore, as the path p contains another vertex (v_k: The clone vertex of v_{k-1}), then the winner will be $P1$ if the length of p is odd. Hence, we can conclude with the following result:

$(G, u) \in GEOGRAPHY$ *if and only if* $(SAT\text{-}G, u) \in ODDPATH\text{-}HORNUNSAT$.

□

Since *GEOGRAPHY* belongs to *PSPACE-complete*, and the time of the reduction above is polynomial, then we obtain *ODDPATH-HORNUNSAT* ∈ *PSPACE-complete*.

Theorem 3.5. $NP = PSPACE$.

Proof. We prove *ODDPATH-HORNUNSAT* is NP and *PSPACE-complete* in Theorems 3.3 and 3.4. Since, $PSPACE$ is closed under reductions and $NP \subseteq PSPACE$, then we have that $NP = PSPACE$ [4]. □

References

[1] S. A. Cook, The complexity of Theorem Proving Procedures, in: Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing (STOC'71), ACM Press, 1971, pp. 151–158.

[2] L. Fortnow, The Status of the P versus NP Problem, Communications of the ACM 52 (9) (2009) 78–86, available at http://www.cs.uchicago.edu/~fortnow/papers/pnp-cacm.pdf. doi:10.1145/1562164.1562186.

[3] A. M. Turing, On Computable Numbers, with an Application to the Entscheidungsproblem, Proceedings of the London Mathematical Society 42 (1936) 230–265.

[4] C. H. Papadimitriou, Computational complexity, Addison-Wesley, 1994.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 2nd Edition, MIT Press, 2001.

[6] O. Goldreich, P, Np, and Np-Completeness, Cambridge: Cambridge University Press, 2010.

[7] 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.