ZERO-ONE-IN-THREE 3SAT is in P

Frank Vega

ZERO-ONE-IN-THREE 3SAT is in P

Frank Vega

Desarrollo de Aplicaciones, Tecnologías y Sistemas, Havana, Cuba

Abstract

$ONE-IN-THREE\ 3SAT$ is the problem of deciding whether a given boolean formula ϕ in $3CNF$ has a truth assignment such that each clause in ϕ has exactly one true literal. This problem is $NP-complete$. We define a similar language: $ZERO-ONE-IN-THREE\ 3SAT$ is the problem of deciding whether a given boolean formula ϕ in $3CNF$ has a truth assignment such that each clause in ϕ has at most one true literal. Indeed, $ONE-IN-THREE\ 3SAT \subseteq ZERO-ONE-IN-THREE\ 3SAT$. In this work, we prove $ZERO-ONE-IN-THREE\ 3SAT \in P$.

Keywords: P, NP, NP-complete, 3SAT
2000 MSC: 68-XX, 68Qxx, 68Q15

1. Introduction

The P versus NP problem is a major unsolved problem in computer science. It was introduced in 1971 by Stephen Cook [1]. Today is considered by many scientists as the most important open problem in this field [2].

Since the beginning of computation many tasks were done by computers, but sometimes some difficult and slow to resolve were not feasible for even the fastest computers. The only way to avoid the delay was to find a possible method that should not do the exhaustive search that was accompanied by "brute force". Even today, there are problems which have not a known method to solve easily yet.

If $P = NP$, then it would ensure there are hundreds of problems that have a feasible solution. This is largely derived from this result there will be a huge amount of problems that can be verified easily and have some practical solution at the same time [3]. This so called $P = NP$ question has been one of the deepest, most perplexing open research problems in theoretical computer science since it was posed in 1971.

The work is about an interesting class of problems, called the "$NP-complete$" problems, whose status is unknown. No polynomial-time algorithm has yet been discovered for an $NP-complete$ problem [4]. Most theoretical computer scientists believe that the $NP-complete$ problems are intractable. The reason is that if any single $NP-complete$ problem can be solved in polynomial time, then every $NP-complete$ problem has a polynomial-time algorithm [4]. In this work, we show a problem in P which has a close relation with an $NP-complete$ problem.

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

2. Theory

The argument made by Alan Turing in the twentieth century proves mathematically that for any computer program we can create an equivalent Turing Machine [5]. A deterministic Turing Machine is a Turing Machine that has only one next action for each step defined in the transition function [6]. However, a non-deterministic Turing Machine can contain more than one action defined for each step of the program, where this program was no longer a function but a relation [7].

A complexity class is a set of problems, which are represented as a language, grouped by measures such as the running time, memory, etc [4]. There are two complexity classes that have a close relationship with the previous concepts and are represented as P and NP. In computational complexity theory, the class P contains the languages that are decided by a deterministic Turing Machine in polynomial time [6]. The class NP contains the languages that are decided by a non-deterministic Turing Machines in polynomial time [7]. Moreover, a language $L \in NP$ if there is a polynomial time decidable and polynomially balanced relation R_L such that for all strings x: there is a string y with $R_L(x, y)$ if and only if $x \in L$ [8]. This string y is known as certificate.

On the other hand, there is a derived complexity class from NP that is the class $NP-complete$. Informally, the $NP-complete$ problems are a set of problems to which any other NP problem can be reduced in polynomial time, but whose solution may still be verified in polynomial time. 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 $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

$$x \in L_1 \; if \; and \; only \; if \; f(x) \in L_2 \tag{1}$$

and 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 $NP - hard$ [4]. Moreover, if $L \in NP$, then $L \in NP - complete$ [4].

There is an important $NP - complete$ problem known as SAT [4]. We formulate the formula satisfiability problem in terms of the language SAT as follows. An instance of SAT is a boolean formula composed of

- boolean variables: $x_1, x_2, ...$;
- 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); and
- parentheses.

A truth assignment for a boolean formula ϕ is a set of values for the variables of ϕ, and a satisfying assignment is a truth assignment that causes it to evaluate to true. A formula with a satisfying assignment is a satisfiable formula. The satisfiability problem asks whether a given boolean formula is satisfiable; in formal language terms,

$$SAT = \{\phi : \phi \; is \; a \; satisfiable \; boolean \; formula\} \tag{2}$$

One convenient language is $3CNF$ satisfiability, or $3SAT$ [4]. 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

$$(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4) \tag{3}$$

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

Many problems can be proved that belong to $NP-complete$ by a polynomial time reduction from $3SAT$. For example, the problem $ONE-IN-THREE\ 3SAT$ which is the following: Given a boolean formula ϕ in $3CNF$, is there a truth assignment such that each clause in ϕ has exactly one true literal?

3. Result

Definition 3.1. *$ZERO-ONE-IN-THREE\ 3SAT$ is the problem of deciding whether a given boolean formula ϕ in $3CNF$ has a truth assignment such that each clause in ϕ has at most one true literal.*

Lemma 3.2. $ONE-IN-THREE\ 3SAT \subseteq ZERO-ONE-IN-THREE\ 3SAT$.

Indeed, for every boolean formula ϕ in $3CNF$ if $\phi \in ONE-IN-THREE\ 3SAT$, then $\phi \in ZERO-ONE-IN-THREE\ 3SAT$.

Definition 3.3. *We could define the language $TWO-OR-THREE\ 3SAT$ as the boolean formulas ϕ in $3CNF$ which have a satisfying truth assignment such that each clause in ϕ has at least two true literals.*

Theorem 3.4. $TWO-OR-THREE\ 3SAT \in P$.

We could decide in polynomial time if any boolean formula ϕ in $3CNF$ of m clauses belongs to $TWO-OR-THREE\ 3SAT$ in the following way,

- in the set of clauses $\{c_1, c_2, ..., c_m\}$ of ϕ, we create for each clause $c_i = (a \vee b \vee c)$ a new boolean formula $d_i = (a \vee b) \wedge (b \vee c) \wedge (a \vee c)$ where a, b and c are literals , and thus,
- we create a new boolean formula ϕ' which is $d_1 \wedge d_2 \wedge ... \wedge d_m$ and is the conjunction of all d_i boolean formulas, and finally,
- we verify $\phi' \in 2SAT$ and accept ϕ for $TWO-OR-THREE\ 3SAT$ otherwise we reject ϕ.

We could state the clause $(a \vee b \vee c)$ has at least two true literals for some truth assignment if and only if the boolean formula $(a \vee b) \wedge (b \vee c) \wedge (a \vee c)$ is satisfiable with this truth assignment. Indeed, if we want to guarantee this property through all the clauses of ϕ, then each boolean formula d_i must have a satisfying truth assignment that should be contained into a single truth

assignment for ϕ. The union of simultaneous truth assignment of each boolean formula d_i could be done by joining the d_i boolean formulas with the AND function. The result would be a new boolean formula $\phi^{'}$ in $2CNF$. Therefore, a satisfying truth assignment to $\phi^{'}$ exists if and only if with this same truth assignment each clause in ϕ has at least two true literals, that is when $\phi \in TWO - OR - THREE\ 3SAT$.

The construction of $\phi^{'}$ is possible in polynomial time, because we only need to iterate with a polynomial amount of steps through the m clauses of ϕ. In conclusion, the decision of $\phi \in TWO - OR - THREE\ 3SAT$ could be done in polynomial time because $2SAT \in P$ [9].

Theorem 3.5. $ZERO - ONE - IN - THREE\ 3SAT \in P$.

Given a boolean formula ϕ in $3CNF$ of m clauses, we could do the following actions,

- in the set of clauses $\{c_1, c_2, ..., c_m\}$ of ϕ, we create for each clause $c_i = (a \vee b \vee c)$ a new clause $d_i = (\ ^{\neg}a \vee\ ^{\neg}b \vee\ ^{\neg}c)$ where a, b and c are literals , and thus,
- we create a new boolean formula $\phi^{'}$ which is $d_1 \wedge d_2 \wedge ... \wedge d_m$ and is the conjunction of all d_i clauses.

This construction of $\phi^{'}$ could be done in order $O(m)$. The clause $(a \vee b \vee c)$ has at most one true literal if and only if the clause $(\ ^{\neg}a \vee\ ^{\neg}b \vee\ ^{\neg}c)$ has at least two true literals. Therefore,

$$\phi \in ZERO - ONE - IN - THREE\ 3SAT\ if\ and\ only\ if\ \phi^{'} \in TWO - OR - THREE\ 3SAT \quad (4)$$

and thus, $ZERO - ONE - IN - THREE\ 3SAT \leq_p TWO - OR - THREE\ 3SAT$.

Acknowledgement

I thank my mother Iris Delgado for her support and confidence.

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.

[3] M. Sipser, Introduction to the Theory of Computation, International Thomson Publishing, 1996.

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

[5] A. M. Turing, On computable numbers, with an application to the entscheidungsproblem, Proceedings of the London Mathematical Society 42 (1936) 230–265.

[6] H. R. Lewis, C. H. Papadimitriou, Elements of the theory of computation (2. ed.), Prentice Hall, 1998.

[7] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences), first edition Edition, W. H. Freeman, 1979.

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

[9] M. R. Krom, The decision problem for a class of first-order formulas in which all disjunctions are binary, Zeitschrift fr Mathematische Logik und Grundlagen der Mathematik 13 (1967) 15–20.