International Journal of Combinatorial Optimization Problems and Informatics, 12(1), Jan-April 2021, 1-8. ISSN: 2007-1558.

www.editada.org

Editorial: P versus NP problem from Formal Languages Theory View

Jorge A. Ruiz-Vanoye, Ocotlán Díaz-Parra, Francisco Rafael Trejo-Macotela, Julio Cesar Ramos-Fernández

Universidad Politécnica de Pachuca, México.
jorge@ruizvanoye.com, ocotlan@diazparra.net

Abstract. P versus NP is an unsolved problem in mathematics and computational complexity. In this paper, we use the formal language theory to the computational complexity to analyze P versus NP problem from a new point of view. P versus NP problem is to determine whether some deterministic algorithm also accepts every language accepted by some nondeterministic algorithm in polynomial time in polynomial time. Then, we use the theory of formal languages to determine whether some deterministic algorithm also accepts every language accepted by some nondeterministic algorithm in polynomial time in polynomial time. We use different problems to display the question of P versus NP from Formal Languages Theory View.
Keywords: Oil Platform, Transport Problem, Waste.

1 Introduction

P versus NP is an unsolved problem in mathematics and computational complexity. The computational complexity contains diverse elements such as the classes of problems complexity, the complexity of algorithms, the complexity of instances and other items [1].

The theory of Problems Computational Complexity is computational complexity classes to determine the complexity of the problems [1]. The Computational Complexity introduces diverse categories of complexity (P, NP, NP-hard and NP-complete, and others) of real problems [2]. Some examples of the real problems are [2]: A) The design of networks, which contains the problems of minimization of route costs. B) Storage and recovery, which contains problems to maximize the allocation of weights (products) in partitions (storage spaces) to obtain savings in the expenses of storage, among other problems. C) The scheduling and allocation of priorities that contain the problems of scheduling of manufacture processes (saving in the idling of the manufacturing processes), of transport vehicle fleets (saving in gasoline), and other problems. Some definitions of the problems complexity classes are:

a. P class. It is the class of recognizable languages by a determinist Turing Machine of one tape in polynomial time [3].
b. NP class. It is the class of recognizable languages by a Non-determinist Turing Machine of one tape in polynomial time [3].
c. NP-equivalent class. It is the class of problems that are considered NP-easy and NP-hard [4].
d. NP-easy class. It is the class of problems that are recognizable in polynomial time by a Turing Machine with one Oracle (subroutine).
e. NP-hard class. The *Q* problem is NP-hard if each problem in NP is reducible to *Q* [2, 5]. It is the class of problems classified as problems of combinatorial optimization at least as complex as an NP.
f. NP-complete class. An *L* language is NP-complete if *L* is in NP, and Satisfiability $\leq_p L$ [6, 3, 7]. It is the class of problems classified as decision problems.

The Theory of Algorithms Computational Complexity is computational complexity measures (time and space) to determine the relations between the size of algorithms or machines and their efficiency. The Computational complexity of algorithms is a way to classify how efficient is an algorithm by means the execution time (asymptotic analysis) to solve a problem with the worst-case input. It is expressed by $O(f(x_1, x_2,\ldots x_n))$ where *f*

is a function of x_i parameters of instance [2]. Von Neumann [8] propose the definitions: polynomial time algorithm and exponential time algorithm. The time complexity of an algorithm is commonly expressed using the big O notation (it was popularized in computer science by Donald Knuth [9]. The most common classes of complexity of algorithms are: Polynomial time algorithms (constant time, linear time, quadratic time, cubic time, polynomial time, strongly polynomial time, weakly polynomial time, super-polynomial time and quasi-polynomial time), Sub-linear time algorithms (logarithmic time, log-logarithmic time, and poly-logarithmic time), Super-polynomial time algorithms (sub-exponential time, exponential time, factorial time, and double exponential time) [1].

The theory of Instances Computational Complexity is computational complexity measures (time and space) to determine the complexity of the problem instances. The Computational complexity of cases is a measure of the computational complexity of individual instances (the specification of particular values of the parameters of a problem [2]) of a string x on a set A and time-bound t [9]. Instance complexity [9] or $ic^t(x:A)$ is defined as the size of the smallest special-case program for A that runs in time t. The complexity of instance of combinatorial optimization problems could be calculated by a mathematical expression based on the descriptive statistics [20]. In this paper, we introduce a formal language theory to the computational complexity to analyze the P versus NP problem from a new point of view. Section II shows the status of the works on the topic of P versus NP; Section III introduces a formal language theory to the computational complexity to analyze P versus NP problem from a new point of view, later are experimentation and the conclusions.

2 Related works on P versus NP

In this section, we show the status of the works on the topic of P versus NP briefly.

Hemmerling shows relationships to quantifier elimination and a computation tree analysis using first-order formulas to find results for P versus NP problems, and other results of structural complexity theory [27].

Cook mentions that P versus NP problem is to determine whether every language accepted by some nondeterministic algorithm in polynomial time is also accepted by some (deterministic) algorithm in polynomial time [26].

Mainhardt shows P versus NP and computability theoretic constructions in complexity theory over algebraic structures [25].

Jukna presents the analogy of P versus NP ∩ co-NP question for the traditional two-party communication protocols where polynomial time is replaced with polylogarithmic communication [24].

Fortnow surveys in a short paper P versus NP problem, its importance to prove P ≠ NP and the approaches to deal with the NP-complete problems [16]. Fortnow presents a non-technical point of view of the P versus NP Problem.

Allender surveys P versus NP question; he summarizes some of the progress that has been made in 2009 [23]. Landsberg describes geometric approaches to variants of P versus NP, results on the role of group actions in complexity theory, and a geometric definition of complexity classes [17].

De Figueiredo [18] contributes to graph theory in the classification of 2 classes of problems for which every problem is classified into P or NP-complete [18].

Pérez-Jiménez [19] analysed the P versus NP problem from the membrane computing view provided by an unconventional bio-inspired model of computing.

3 The P versus NP problem from Formal Languages Theory

P versus NP problem is to determine whether every language accepted by in some nondeterministic algorithm in polynomial time is also accepted by some deterministic algorithm in polynomial time [15]. In this section, we analyze P versus NP problem from formal language theory view.

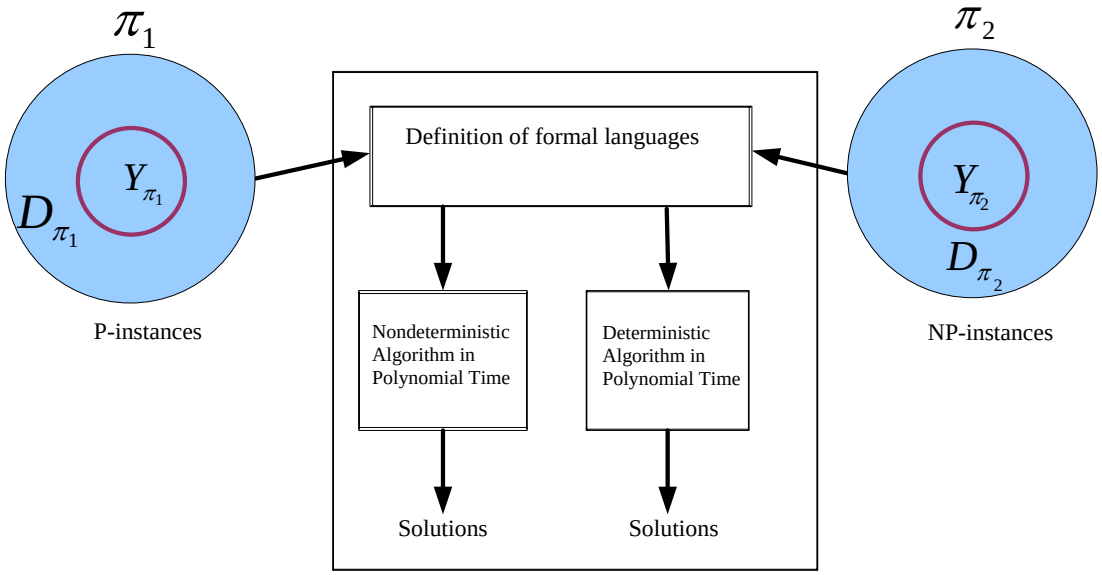

Fig. 1. Every language accepted by in some nondeterministic algorithm in polynomial time is also accepted by some deterministic algorithm in polynomial time.

We propose the next definitions:
Definition. P class from Formal Languages Theory view is the class of recognizable formal languages (alphabet and grammar) by a deterministic algorithm in polynomial time.

Definition. P-instances are obtained by specifying particular values of the parameters of a P class. $Y_{\pi 1}$ defines yes-instances of the problem or feasible solutions and The $D_{\pi 1}$ defines no-instances or infeasible solutions.

Definition. NP class from Formal Languages Theory view is the class of recognizable formal languages (alphabet and grammar) by a nondeterministic algorithm in polynomial time.

Definition. NP-instances areobtained by specifying particular values of parameters of an NP class. $Y_{\pi 2}$defines yes-instances of the problem or feasible solutions and The $D_{\pi 2}$ defines no-instances or infeasible solutions.

Definition. NP-hard class from Formal Languages Theory is the class of the Combinatorial Optimization Problems that could be transformed by a polynomial transformation that uses the formal language theory to another NP Combinatorial Optimization Problem.

Definition. NP-complete Class from Formal Languages Theory view is the class of the Decision Problems that could be transformed by a polynomial transformation that uses the formal language theory to another NP Decision Problem.

An algorithm is a finite sequence of instructions (a procedure that always terminates) that can be mechanically carried out [28].

Definition. A nondeterministic algorithm in polynomial time is an algorithm with a finite sequence of instructions that performs several steps or computations to obtain different feasible or infeasible solutions in polynomial time function.

Definition. A deterministic algorithm in polynomial time is an algorithm with a finite sequence of instructions that perform the same steps or computations to obtain the same feasible or infeasible solution in polynomial time function.

Definition. A nondeterministic algorithm in exponential time is an algorithm with a finite sequence of instructions that performs several steps or computations to obtain different feasible or infeasible solutions in exponential time function.

Definition. A deterministic algorithm in exponential time is an algorithm with a finite sequence of instructions that perform the same steps or computations to obtain the same feasible or infeasible solution in exponential time function.

There are different definitions of languages: Brown (1960) [10] described engineering languages with the objective of making a language distinct from the natural language. The engineering languages are languages that are designed to specify objective criteria and modelled to meet criteria. Hopcroft and Ullman (1969) [11] defined the language as any set V* of sentences on an alphabet V. A sentence of an alphabet is any string of finite length composed of symbols from alphabet *V* [11]. Cook (1971) [12] defined a language as a set *G* of chains of symbols on a fixed, large, and finite alphabet {0, 1, *}. Karp (1972) [13] defined a language as a subset of Σ* (the set of all the finite chains of 0's and 1's). Karp defined NP-complete problems (*L*-complete) as *L* (polynomial) complete if $L \in$ NP and every language in NP is reducible to *L*[13]. Garey and Johnson (1979) [2] mentioned a language as any finite set Σ of symbols, denoted by Σ* the set of all finite strings of symbols from Σ, if Σ= {0, 1}. A language corresponds to an NP-complete problem when the values of the instance parameters of the problem can be codified in a hypothetical language [20].

A formal language is defined abstractly as a mathematical system [28]. In this paper, we use the theory of formal languages [21] to determine whether some deterministic algorithm also accepts every language accepted by some nondeterministic algorithm in polynomial time in polynomial time. Specifically, we use polynomial transformations of a language *A* to language *B* performed using a computer program that translates text written in a source language into another target language [28]. A polynomial transformation is a mechanism that is useful for finding out if a problem belongs to a class of problems, determining if a problem is more complex than another, and for helping solve complex real-life optimization problems for which no algorithms can be found that guarantee to yield exact solutions. Polynomial transformation is possible through transformation expressions, NP-completeness theory, graph theory and formal language theory [22].A polynomial transformation between NP-hard problems allows a language (L_1) and a language (L_2) to be transformed in polynomial time.

For polynomial transformations exist several definitions [22]: Karp (1972) [13] defined a polynomial reduction as follows: considering two languages *L* and *M*, then *L* is reducible to *M* if there exists a function $f \in M \Leftrightarrow x \in L$. Cook (1971) [12] defined a polynomial reduction as follows: a set *S* of chains of symbols (on a fixed, large, and finite alphabet {0.1, *}) is polynomial reducible to a set *T* of chains of symbols (on a fixed, large, and finite alphabet {0.1, *}) if there exists a query machine *M* and a polynomial $Q(n)$, such that for each input string *w* the computation of *M* with input *w* halts in $Q(|w|)$ steps ($|w|$ is the length of *w*) and ends in the accept state iff $w \in S$.Garey and Johnson (1979) [2] defined a polynomial transformation from a language $L_1 \subseteq \Sigma_1^*$ to a language $L_2 \subseteq \Sigma_2^*$ as a function f: $\Sigma_1^* \rightarrow \Sigma_2^*$ that satisfies the following two conditions: (1) There is a polynomial time deterministic Turing machine (DTM) program that computes *f*. (2) For all $x \in \Sigma_1^*$, $x \in L_1$ if and only if $f(x) \in L_2$.The polynomial transformation approach mentions that a language L_1 of a complex real-life optimization problem *A*is transformable in polynomial time to a language L_2 of a complex real-life optimization problem*B* ($L_1 \leq_P L_2$) if there exists a transformation *f* from problem *A* to problem *B*. This implies a transformation from each instance *x* of L_1 to an instance $f(x)$ of L_2.

Table 1. Differences between definitions of NP using Turing Machine / polynomial reduction and definitions from the Formal Languages Theory View / polynomial transformation.

Definitions from Turing Machine	**Definitions from the Formal Languages Theory View**
P class. It is the class of recognizable languages by a determinist Turing Machine of one tape in polynomial time [3].	*P class from Formal Languages Theory view is the class of recognizable formal languages (alphabet and grammar) by a deterministic algorithm in polynomial time.*
NP class. It is the class of recognizable languages by a Non-determinist Turing Machine of one tape in polynomial time [3].	*NP from Formal Languages Theory view is the class of recognizable formal languages (alphabet and grammar) by a nondeterministic algorithm in polynomial time.*
NP-hard class. The Q problem is NP-hard if each problem in NP is reducible to Q [2, 5]. It is the class of problems classified as problems of combinatorial optimization at least as complex as an NP.	*NP-hard class from Formal Languages Theory view is the class of the Combinatorial Optimization Problems that could be transformed by a polynomial transformation that uses the formal language theory to another NP Combinatorial Optimization Problem.*
NP-complete class. An L language is NP-complete if L is in NP, and Satisfiability $\leq_P$ L [6, 3, 7]. It is the class of problems classified as decision problems.	*NP-complete Class from Formal Languages Theory view is the class of the Decision Problems that could be transformed by a polynomial transformation that uses the formal language theory to another NP Decision Problem.*

The steps for polynomial transformation using formal language theory are [21, 22]: 1. Select areal-life optimization problem *A* (source problem). 2. Define a formal language L_1 (source language) for the real-life optimization problem *A*.3. Select areal-life optimization problem *B* (target problem). 4. Define a formal language L_2 (target language) for the real-life optimization problem *B*.5. Construct a compiler that transforms in polynomial time ($L_1 \leq_P L_2$) a source language L_1into a target language L_2. 6. Optionally, it is possible to add to the phase of language generation an algorithm that solves target language L_2.

In the figure 2 is the codification scheme of real-life optimization problem based on formal languages by using a compiler (with phases of lexical analysis, syntactic analysis, semantic analysis and language generator) from the source language L_1 (which defines yes-instances of the problem or feasible solutions $Y_{\pi 1}$) to target language L_2 (which defines yes-instances of the problem $Y_{\pi 2}$).The *D* defines no-instances of both problems.

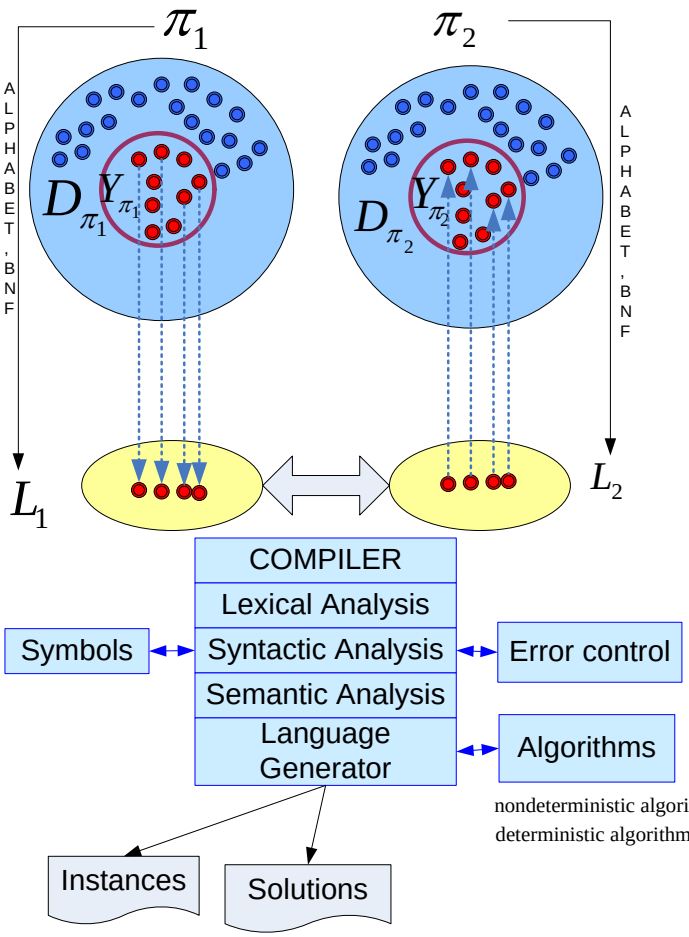

Fig. 2. Codification scheme of the real-life optimization problem based on formal languages.

The steps for polynomial transformation from P-instances to NP-instances (P-instances $\leq_P$NP-instances) and from NP-instances to P-instances (NP-instances $\leq_P$P-instances) are:
1. Select an NP class optimization problem.
2. Define a formal language L_1 for the NP class optimization problem.
3. Solve the L_1 by a nondeterministic algorithm in polynomial time.
4. Select a P class optimization problem.
5. Define a formal language L_2 for the P class optimization problem.
6. Construct a compiler that transforms in polynomial time ($L_1 \leq_P L_2$) a source language L_1 into a target language L_2.
7. Add to the phase of language generation an algorithm that solves the target language $L_2$by a deterministic algorithm in polynomial time.

4 Conclusions

This article contained a formal language theory to the computational complexity to analyze the P versus NP problem from a new point of view. P class and NP class from formal languages theory view, with the fundamental difference of updating the concepts of languages recognizable by determinist and Non-determinist Turing Machine of one tape in polynomial time a recognizable formal language (alphabet and grammar) by a deterministic and nondeterministic algorithm in polynomial time.
Also, new definitions were proposed on P-instances, NP-instances, the nondeterministic algorithm in polynomial time, the deterministic algorithm in polynomial time, the nondeterministic algorithm in exponential time, the deterministic algorithm in exponential time.

ACKNOWLEDGEMENT
This work was supported by the Laboratorio Nacional en Vehículos Autónomos y Exoesqueletos (LN 299146).

5 References

1 Ruiz-Vanoye, J. A., & Díaz-Parra, O. (2011). An overview of the theory of instances computational complexity. *International Journal of Combinatorial Optimization Problems and Informatics*, *2*(2), 21-27.
2 Hartmanis, J. (1982). Computers and intractability: a guide to the theory of NP-completeness (michael r. garey and david s. johnson). *Siam Review*, *24*(1), 90.
3 Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations* (pp. 85-103). Springer, Boston, MA.
4 Jonsson, P., & Bäckström, C. (1995). *Complexity results for state-variable planning under mixed syntactical and structural restrictions* (p. 205). Universitetet i Linköping/Tekniska Högskolan i Linköping. Institutionen för Datavetenskap.
5 Papadimitriou, C. H., & Steiglitz, K. (1998). *Combinatorial optimization: algorithms and complexity*. Courier Corporation.
6 Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms*. MIT press.
7 Cook, S. A. (1971, May). The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing* (pp. 151-158).
8 Von Neumann, J. (1953). A certain zero-sum two-person game equivalent to the optimal assignment problem. *Contributions to the Theory of Games*, *2*(0), 5-12.
9 Knuth, D. E. (1976). Big omicron and big omega and big theta. *ACM Sigact News*, *8*(2), 18-24.
10 Brown, J.C., 1960. Loglan. Scientific American, 202:43-63.
11 Hopcroft, J., Ullman, J., 1969. Formal Languages and Their Relation to Automata. Addison-Wesley, USA, p.1-7.
12 Cook, S. A. (1971, May). The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing* (pp. 151-158).
13 Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations* (pp. 85-103). Springer, Boston, MA.
14 Shannon, C. E. (1949). The synthesis of two-terminal switching circuits. *The Bell System Technical Journal*, *28*(1), 59-98.
15 Cook, S. (2003). The importance of the P versus NP question. *Journal of the ACM (JACM)*, *50*(1), 27-29.
16 Fortnow, L. (2009). The status of the P versus NP problem. *Communications of the ACM*, *52*(9), 78-86.
17 Landsberg, J. M. (2010). P versus NP and geometry. *Journal of Symbolic Computation*, *45*(12), 1359-1377.
18 De Figueiredo, C. M. (2012). The P versus NP–complete dichotomy of some challenging problems in graph theory. *Discrete Applied Mathematics*, *160*(18), 2681-2693.
19 Pérez-Jiménez, M. J. (2014). The P versus NP problem from the membrane computing view. *European Review*, *22*(1), 18-33.
20 Ruiz-Vanoye, J. A., Diaz-Parra, O., Perez-Ortega, J., Salgado, G. R., & Gonzalez-Barbosa, J. J. (2010). Complexity of instances for combinatorial optimization problems. In *Computational Intelligence and Modern Heuristics*. IntechOpen.
21 Ruiz-Vanoye, J. A., Pérez-Ortega, J., Rangel, R. A. P., Díaz-Parra, O., Fraire-Huacuja, H. J., Frausto-Solís, J., & Cruz-Reyes, L. (2013). Application of formal languages in polynomial transformations of instances between NP-complete problems. *Journal of Zhejiang University SCIENCE C*, *14*(8), 623-633.
22 Ruiz-Vanoye, J. A., Pérez-Ortega, J., Díaz-Parra, O., Frausto-Solís, J., Huacuja, H. J. F., & Cruz-Reyes, L. (2011). Survey of polynomial transformations between NP-complete problems. *Journal of computational and applied mathematics*, *235*(16), 4851-4865.

23 Allender, E. (2009). A status report on the P versus NP question. *Advances in Computers*, *77*, 117-147.
24 Jukna, S. (2005). On the P versus NP intersected with co-NP question in communication complexity. *Information processing letters*, *96*(6), 202-206.
25 Mainhardt, G. (2004). P versus NP and computability theoretic constructions in complexity theory over algebraic structures. *The Journal of Symbolic Logic*, *69*(1), 39-64.
26 Cook, S. (2003). The importance of the P versus NP question. *Journal of the ACM (JACM)*, *50*(1), 27-29.
27 Hemmerling, A. (2001). On P Versus NP for Parameter-Free Programs Over Algebraic Structures. *Mathematical Logic Quarterly: Mathematical Logic Quarterly*, *47*(1), 67-92.
28 Aho, A. V., Sethi, R., & Ullman, J. D. (1986). Compilers, principles, techniques. *Addison wesley*, *7*(8), 9.
29 Toth, P., & Vigo, D. (Eds.). (2002). *The vehicle routing problem*. Society for Industrial and Applied Mathematics.