P versus NP

Frank Vega
Joysonic, Uzun Mirkova 5, Belgrade, 11000, Serbia
vega.frank@gmail.com
https://orcid.org/0000-0001-8210-4126

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? This question was first mentioned in a letter written by John Nash to the National Security Agency in 1955. A precise statement of the P versus NP problem was introduced independently in 1971 by Stephen Cook and Leonid Levin. Since that date, all efforts to find a proof for this problem have failed. Another major complexity classes are LOGSPACE and NLOGSPACE. Whether LOGSPACE = NLOGSPACE is another fundamental question that it is as important as it is unresolved. SAT is easier if the number of literals in a clause is limited to at most 2, in which case the problem is called 2SAT. This problem can be solved in polynomial time, and in fact is complete for the complexity class NLOGSPACE. If additionally all OR operations in literals are changed to XOR operations, the result is called exclusive-or 2-satisfiability, which is a problem complete for the complexity class LOGSPACE. Given an instance of exclusive-or 2-satisfiability and a positive integer K, the problem maximum exclusive-or 2-satisfiability consists in deciding whether this Boolean formula has a truth assignment with at leat K satisfiable clauses. We prove that maximum exclusive-or 2-satisfiability is in NLOGSPACE. Moreover, we demonstrate this problem is NP-complete. To attack the P versus NP question the concept of NP-completeness has been very useful. If any single NP-complete problem can be solved in polynomial time, then every NP problem has a polynomial time algorithm. Since every language in the class NLOGSPACE is in P, then we show that our problem is in P and NP-complete and thus, P = NP.

2012 ACM Subject Classification Theory of computation → Complexity classes, Theory of computation → Problems, reductions and completeness

Keywords and phrases Complexity Classes, NP-complete, LOGSPACE, NLOGSPACE, exclusive-or 2-satisfiability

1 Introduction

The P versus NP problem is a major unsolved problem in computer science [5]. This is considered by many to be the most important open problem in the field [5]. 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 [5]. It was essentially mentioned in 1955 from a letter written by John Nash to the United States National Security Agency [1]. However, the precise statement of the $P = NP$ problem was introduced in 1971 by Stephen Cook in a seminal paper [5].

In 1936, Turing developed his theoretical computational model [19]. The deterministic and nondeterministic Turing machines have become in two of the most important definitions related to this theoretical model for computation [19]. A deterministic Turing machine has only one next action for each step defined in its program or transition function [19]. A nondeterministic Turing machine could contain more than one action defined for each step of its program, where this one is no longer a function, but a relation [19].

Another relevant advance in the last century has been the definition of a complexity class. A language over an alphabet is any set of strings made up of symbols from that alphabet [6]. A complexity class is a set of problems, which are represented as a language, grouped by measures such as the running time, memory, etc [6].

The set of languages decided by deterministic Turing machines within time f is an important complexity class denoted $TIME(f(n))$ [16]. In addition, the complexity class $NTIME(f(n))$ consists in those languages that can be decided within time f by non-deterministic Turing machines [16]. The most important complexity classes are P and NP. The class P is the union of all languages in $TIME(n^k)$ for every possible positive fixed constant k [16]. At the same time, NP consists in all languages in $NTIME(n^k)$ for every possible positive fixed constant k [16]. Whether $P = NP$ or not is still a controversial and unsolved problem [1]. Our goal is to prove the answer $P = NP$.

2 Motivation

If any single NP–*complete* problem can be solved in polynomial time, then every NP problem has a polynomial time algorithm [6]. No polynomial time algorithm has yet been discovered for any NP–*complete* problem [8]. The biggest open question in theoretical computer science concerns the relationship between these classes: Is P equal to NP? In 2012, a poll of 151 researchers showed that 126 (83%) believed the answer to be no, 12 (9%) believed the answer is yes, 5 (3%) believed the question may be independent of the currently accepted axioms and therefore impossible to prove or disprove, 8 (5%) said either do not know or do not care or don't want the answer to be yes nor the problem to be resolved [10]. It is fully expected that $P \neq NP$ [16]. Indeed, if $P = NP$ then there are stunning practical consequences [16]. For that reason, $P = NP$ is considered as a very unlikely event [16]. Certainly, P versus NP is one of the greatest open problems in science and a correct solution for this incognita will have a great impact not only for computer science, but for many other fields as well [8].

3 Theory

Let Σ be a finite alphabet with at least two elements, and let Σ^* be the set of finite strings over Σ [3]. A Turing machine M has an associated input alphabet Σ [3]. For each string w in Σ^* there is a computation associated with M on input w [3]. We say that M accepts w if this computation terminates in the accepting state, that is $M(w) = \text{“}yes\text{”}$ [3]. Note that M fails to accept w either if this computation ends in the rejecting state, that is $M(w) = \text{“}no\text{”}$, or if the computation fails to terminate [3].

The language accepted by a Turing machine M, denoted $L(M)$, has an associated alphabet Σ and is defined by:

$$L(M) = \{w \in \Sigma^* : M(w) = \text{“}yes\text{”}\}.$$

We denote by $t_M(w)$ the number of steps in the computation of M on input w [3]. For $n \in \mathbb{N}$ we denote by $T_M(n)$ the worst case run time of M; that is:

$$T_M(n) = max\{t_M(w) : w \in \Sigma^n\}$$

where Σ^n is the set of all strings over Σ of length n [3]. We say that M runs in polynomial time if there is a constant k such that for all n, $T_M(n) \leq n^k + k$ [3]. In other words, this means the language $L(M)$ can be accepted by the Turing machine M in polynomial time. Therefore, P is the complexity class of languages that can be accepted in polynomial time by deterministic Turing machines [6]. A verifier for a language L is a deterministic Turing machine M, where:

$$L = \{w : M(w, c) = \text{“}yes\text{” } \textit{for some string } c\}.$$

We measure the time of a verifier only in terms of the length of w, so a polynomial time verifier runs in polynomial time in the length of w [3]. A verifier uses additional information, represented by the symbol c, to verify that a string w is a member of L. This information is called certificate. NP is also the complexity class of languages defined by polynomial time verifiers [16].

A function $f : \Sigma^* \to \Sigma^*$ is a polynomial time computable function if some deterministic Turing machine M, on every input w, halts in polynomial time with just $f(w)$ on its tape [19]. Let $\{0,1\}^*$ be the infinite set of binary strings, we say that a language $L_1 \subseteq \{0,1\}^*$ is polynomial time reducible to a language $L_2 \subseteq \{0,1\}^*$, written $L_1 \leq_p L_2$, if there is a polynomial time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$:

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

An important complexity class is *NP–complete* [11]. A language $L \subseteq \{0,1\}^*$ is *NP–complete* if

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

If L is a language such that $L' \leq_p L$ for some $L' \in$ *NP–complete*, then L is *NP–hard* [6]. Moreover, if $L \in NP$, then $L \in$ *NP–complete* [6]. A principal *NP–complete* problem is SAT [9]. An instance of 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), $\neg$(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 formula with a satisfying truth assignment is a satisfiable formula. The problem SAT asks whether a given Boolean formula is satisfiable [9]. 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 [6]. 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 [6]. A Boolean formula is in 3-conjunctive normal form or $3CNF$, if each clause has exactly three distinct literals [6].

For example, the Boolean formula:

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

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 $\neg x_2$. Another relevant *NP–complete* language is $3CNF$ satisfiability, or $3SAT$ [6]. In $3SAT$, it is asked whether a given Boolean formula ϕ in $3CNF$ is satisfiable. Many problems have been proved that belong to *NP-complete* by a polynomial time reduction from $3SAT$ [9]. For example, the problem *1-IN-3 3SAT* defined as follows: Given a Boolean formula ϕ in $3CNF$, is there a truth assignment such that each clause in ϕ has exactly one true literal?

A logarithmic space Turing machine has a read-only input tape, a write-only output tape, and a read/write work tape [19]. The work tape may contain $O(\log n)$ symbols [19]. In computational complexity theory, $LOGSPACE$ is the complexity class containing those decision problems that can be solved by a logarithmic space Turing machine which is deterministic [16]. $NLOGSPACE$ is the complexity class containing the decision problems that can be solved by a logarithmic space Turing machine which is nondeterministic [16]. A Boolean formula is in 2-conjunctive normal form, or $2CNF$, if it is in CNF and each clause has exactly two distinct literals. There is a problem called $2SAT$, where we asked whether a given Boolean formula ϕ in $2CNF$ is satisfiable. $2SAT$ is complete for $NLOGSPACE$ [16]. Another special case is the class of problems where each clause contains XOR (i.e. exclusive or) rather than (plain) OR operators. This is in P, since an XOR SAT formula can also be viewed as a system of linear equations mod 2, and can be solved in cubic time by Gaussian elimination [15]. We denote the XOR function as $\oplus$. The *XOR 2SAT* problem will be equivalent to *XOR SAT*, but the clauses in the formula have exactly two distinct literals. *XOR 2SAT* is complete for $LOGSPACE$ [2], [18].

4 Results

We can give a certificate-based definition for $NLOGSPACE$ [3]. The certificate-based definition of $NLOGSPACE$ assumes that a logarithmic space Turing machine has another separated read-only tape [3]. On each step of the machine the machine's head on that tape can either stay in place or move to the right [3]. In particular, it cannot reread any bit to the left of where the head currently is [3]. For that reason this kind of special tape is called "read once" [3].

▶ **Definition 1.** A language L is in $NLOGSPACE$ if there exists a deterministic logarithmic space Turing machine and a with an additional special read-once input tape polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for every $x \in \{0,1\}^*$,

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ such that } M \text{ accepts } \langle x, u\rangle$$

where by $M(x,u)$ we denote the computation of M where x is placed on its input tape and u is placed on its special read-once tape, and M uses at most $O(\log |x|)$ space on its read/write tape for every input x.

▶ **Definition 2. MAXIMUM EXCLUSIVE-OR 2-SATISFIABILITY**

INSTANCE: A positive integer K and a formula ϕ that is an instance of *XOR 2SAT*.

QUESTION: Is there a truth assignment in ϕ such that at least K clauses are satisfiable?

We denote this problem as $MAX \oplus 2SAT$.

▶ **Theorem 3.** *$MAX \oplus 2SAT \in NLOGSPACE$.*

Proof. Given a Boolean formula ϕ that is an instance of *XOR 2SAT* with n variables and m clauses, we enumerate from left to right the clauses in ϕ such that the leftmost clause has the index 1 and the rightmost the number m. Therefore, a combination of K clauses in ϕ corresponds to a subset of size K from the set $\{1, 2, 3, \ldots, m-1, m\}$. This subset of K numbers will be a regular language, because it is finite [13]. Since it is a regular language, then it will be computable by a linear size NC^1 circuit [13]. Since the number of input gates is at most $\lceil \log m \rceil$, then the size of the circuit C which computes this subset is bounded by $O(\log m)$.

There will be a deterministic logarithmic space Turing machine which receives the circuit C in the special read-once input tape and in the input tape the given Boolean formula ϕ that is an instance of *XOR 2SAT*. Next, we copy the circuit C to the read/write work tape just reading each bit in the special read-once input tape from left to right until we find the blank symbol. We can copy it to the read/write work tape, because the size of C is $O(\log m)$. After that, we evaluate in ascending order the numbers in the set $\{1, 2, 3, \ldots, m-1, m\}$ just to verify if there are at least K numbers which leads to an acceptance. This can be done in $O(\log m)$, because *CIRCUIT VALUE* can be solved in linear time [16]. Besides, we can count the number of different acceptances with a positive integer $d \leq m$ that will have at most $\lceil \log m \rceil$ bit-length. Furthermore, if we obtain in the counting that $d > m$, then we reject. In this way, the process remains in $O(\log m)$ space.

Finally, if there are at least K acceptances between the numbers 1 and m, then we compute in the read/write work tape the Boolean formula $\psi = c_{i_1} \wedge c_{i_2} \ldots \wedge c_{i_K} \ldots$ such that each number i_j is accepted by C. Since *XOR 2SAT* is complete for $LOGSPACE$ [2], [18], then we can decide ψ in logarithmic space. Notice that, we do not need to copy ψ to the read/write work tape since the membership in ψ of any clause c_{i_j} in the input tape can be done in logarithmic space by an evaluation in C. In this way, we finally accept in case of ψ is satisfiable otherwise we reject the chosen input and certificate. All this process can be done in deterministic logarithmic space just reading at once the additional special tape. Moreover, the size of the certificate is polynomial due to the size and the depth of C is logarithmic. In conclusion, we show $MAX \oplus 2SAT$ complies with the certificate-based definition of $NLOGSPACE$ and thus, $MAX \oplus 2SAT \in NLOGSPACE$ [3]. ◀

▶ **Theorem 4.** *$MAX \oplus 2SAT \in$ NP–complete.*

Proof. $MAX \oplus 2SAT \in NP$, because $NLOGSPACE \subseteq NP$ [16]. Given a Boolean formula ϕ in $3CNF$ with n variables and m clauses. For each clause $c_i = (x \vee y \vee z)$ in ϕ, where x, y and z are literals, we create the following formulas,

$$P_i = (\neg x \oplus \neg y) \wedge (\neg y \oplus \neg z) \wedge (\neg x \oplus \neg z).$$

We can see P_i has exactly two satisfiable clauses if and only if exactly 1 member of $\{x, y, z\}$ is true. Hence, we can create the Boolean formula ψ as the conjunction of the P_i formulas for every clause c_i in ϕ, such that $\psi = P_1 \wedge \ldots \wedge P_m$. Finally, we obtain that

$$\phi \in \textit{1-IN-3 3SAT if and only if } (\psi, 2 \times m) \in MAX \oplus 2SAT.$$

To sum up, we show $MAX \oplus 2SAT \in$ *NP–hard* and $MAX \oplus 2SAT \in NP$ and thus, $MAX \oplus 2SAT \in$ *NP–complete.* ◀

▶ **Theorem 5.** *$P = NP$.*

Proof. If any single *NP–complete* problem can be solved in polynomial time, then every NP problem has a polynomial time algorithm [6]. Every language in the class $NLOGSPACE$ is in P, and therefore, $MAX \oplus 2SAT \in P$ [16]. Hence, as a consequence of Theorems 3 and 4, then $P = NP$. ◀

5 Conclusion

No one has been able to find a polynomial time algorithm for any of more than 300 important known *NP–complete* problems [9]. Most complexity theorists already assume P is not equal to NP, but no one has found an accepted and valid proof yet [10]. There are several consequences if P is not equal to NP, such as many common problems cannot be solved efficiently [5]. However, a proof of $P = NP$ will have stunning practical consequences, because it leads to efficient methods for solving some of the important problems in NP [5]. The consequences, both positive and negative, arise since various *NP–complete* problems are fundamental in many fields [5]. This result explicitly concludes with the answer of the P versus NP problem: $P = NP$.

Cryptography, for example, relies on certain problems being difficult. A constructive and efficient solution to an *NP–complete* problem such as $3SAT$ will break most existing cryptosystems including: Public-key cryptography [12], symmetric ciphers [14] and one-way functions used in cryptographic hashing [7]. These would need to be modified or replaced by information-theoretically secure solutions not inherently based on *P–NP* equivalence.

There are enormous positive consequences that will follow from rendering tractable many currently mathematically intractable problems. For instance, many problems in operations research are *NP–complete*, such as some types of integer programming and the traveling salesman problem [11]. Efficient solutions to these problems have enormous implications for logistics [5]. Many other important problems, such as some problems in protein structure prediction, are also *NP–complete*, so this will spur considerable advances in biology [4].

But such changes may pale in significance compared to the revolution an efficient method for solving *NP–complete* problems will cause in mathematics itself. Stephen Cook says: "...it would transform mathematics by allowing a computer to find a formal proof of any theorem which has a proof of a reasonable length, since formal proofs can easily be recognized in polynomial time." [5].

Indeed, this proof of $P = NP$ could solve not merely one Millennium Problem but all seven of them [1]. This observation is based on once we fix a formal system such as the first-order logic plus the axioms of ZF set theory, then we can find a demonstration in time polynomial in n when a given statement has a proof with at most n symbols long in that system [1]. This is assuming that the other six Clay conjectures have ZF proofs that are not too large such as it was the Perelman's case [17].

Besides, a $P = NP$ proof reveals the existence of an interesting relationship between humans and machines [1]. For example, suppose we want to program a computer to create new Mozart-quality symphonies and Shakespeare-quality plays. When $P = NP$, this could be reduced to the easier problem of writing a computer program to recognize great works of art [1].

References

1. Scott Aaronson. P $\stackrel{?}{=}$ NP. *Electronic Colloquium on Computational Complexity, Report No. 4*, 2017.
2. Carme Alvarez and Raymond Greenlaw. A compendium of problems complete for symmetric logarithmic space. *Computational Complexity*, 9(2):123–145, 2000.
3. Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
4. Bonnie Berger and Tom Leighton. Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete. *Journal of Computational Biology*, 5(1):27–40, 1998.
5. Stephen A Cook. The P versus NP Problem, April 2000. Available at Millennium Prize Problems Web Site `http://www.claymath.org/sites/default/files/pvsnp.pdf`.
6. Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

7 Debapratim De, Abishek Kumarasubramanian, and Ramarathnam Venkatesan. Inversion attacks on secure hash functions using SAT solvers. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 377–382. Springer, 2007.

8 Lance Fortnow. The status of the P versus NP problem. *Communications of the ACM*, 52(9):78–86, 2009.

9 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.

10 William I Gasarch. Guest column: The second P $\stackrel{?}{=}$ NP poll. *ACM SIGACT News*, 43(2):53–77, 2012.

11 Oded Goldreich. *P, NP, and NP-Completeness: The basics of computational complexity.* Cambridge University Press, 2010.

12 Satoshi Horie and Osamu Watanabe. Hard instance generation for SAT. *Algorithms and Computation*, pages 22–31, 1997.

13 Michal Koucky. Circuit complexity of regular languages, April 2012. Available at Jayalal's Home Page `http://www.cse.iitm.ac.in/~jayalal/teaching/CS6840/2012/project/Regular-Sunil-slides.pdf`.

14 Fabio Massacci and Laura Marraro. Logical cryptanalysis as a SAT problem. *Journal of Automated Reasoning*, 24(1):165–203, 2000.

15 Cristopher Moore and Stephan Mertens. *The Nature of Computation.* Oxford University Press, 2011.

16 Christos H Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

17 Grigori Perelman. The entropy formula for the Ricci flow and its geometric applications, November 2002. Available at arXiv Web Site `http://www.arxiv.org/abs/math.DG/0211159`.

18 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4):1–24, 2008.

19 Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.