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? 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. To attack the P = NP question the concept of NP-completeness is very useful. If any single NP-complete problem is in P, then P = NP. Quadratic Congruences is a well-known NP-complete problem. We prove Quadratic Congruences is also in P. In this way, we demonstrate that P = NP.

2012 ACM Subject Classification Theory of computation → Complexity classes, Theory of computation → Problems, reductions and completeness, Mathematics of computing → Number-theoretic computations

Keywords and phrases Polynomial Time, NP, UP, NP-complete, Quadratic Congruences

1 Issues

P versus NP is a major unsolved problem in computer science [1]. It is considered by many to be the most important open problem in the field [1]. 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 [1]. It was essentially mentioned in 1955 from a letter written by John Nash to the United States National Security Agency [1].

In 1936, Turing developed his theoretical computational model [2]. The deterministic and nondeterministic Turing machines have become in two of the most important definitions related to this theoretical model for computation. A deterministic Turing machine has only one next action for each step defined in its program or transition function [2]. 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 [2].

Another huge 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 [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 the computational complexity theory, the class P contains those languages that can be decided in polynomial time by a deterministic Turing machine [9]. The class NP consists in those languages that can be decided in polynomial time by a nondeterministic Turing machine [9]. Another major complexity class is UP. The class UP has all the languages that are decided in polynomial time by a nondeterministic Turing machines with at most one accepting computation for each input [16]. It is obvious that $P \subseteq UP \subseteq NP$ [14]. Whether $P = UP$ or $UP = NP$ are fundamental questions that they are as important as they are unresolved [14]. In this work, we prove the complexity class P is equal to NP. In this way, we solve one of the most important open problems in computer science.

2 Motivation

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 [8]. It is fully expected that $P \neq NP$ [14]. Indeed, if $P = NP$ then there are stunning practical consequences [14]. For that reason, $P = NP$ is considered as a very unlikely event [14]. 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 [1].

3 Summary

We prove the problem Quadratic Congruences is in UP. We deduce this from itself definition of a polynomial verifier for the class UP that complies the language Quadratic Congruences. We guarantee this since we define the uniqueness of the certificate based on the statement when there is a finite set of positive integers, then there must be only one minimum [13]. In addition, we define this finite set of positive integers as the set of certificates from an instance of Quadratic Congruences. Moreover, we show that this verifier can decide its inputs in polynomial time. In this way, we guarantee the problem Quadratic Congruences can be considered as a UP language. Moreover, we use the properties of this polynomial verifier to search a solution for an instance of this problem. This is possible since we can start from a candidate that might not be a solution and then, we will start decreasing this value until we found the minimum possible value which might be a certificate or not. In addition, we show this algorithm can be done in polynomial time. Since Quadratic Congruences is a well-known NP-complete and P is closed under reductions, then we demonstrate that P is equal to NP [7].

4 Significance

No one has been able to find a polynomial time algorithm for any of more than 300 important known *NP–complete* problems [7]. 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 [4]. The consequences, both positive and negative, arise since various *NP–complete* problems are fundamental in many fields [4]. This result explicitly concludes supporting the existence of a practical solution for the *NP–complete* problems because $P = UP = NP$.

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 [11], symmetric ciphers [12] and one-way functions used in cryptographic hashing [6]. 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 [7]. Efficient solutions to these problems have enormous implications for logistics [4]. Many other important problems, such as some problems in protein structure prediction, are also *NP–complete*, so this will spur considerable advances in biology [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. 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." [4].

Research mathematicians spend their careers trying to prove theorems, and some proofs have taken decades or even centuries to find after problems have been stated. For instance, Fermat's Last Theorem took over three centuries to prove. A method that is guaranteed to find proofs to theorems, should one exist of a "reasonable" size, would essentially end this struggle.

Indeed, with a polynomial algorithm for an *NP–complete* problem, we 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 [1].

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].

5 Basic Definitions

Let Σ be a finite alphabet with at least two elements, and let Σ^* be the set of finite strings over Σ [2]. A Turing machine M has an associated input alphabet Σ [2]. For each string w in Σ^* there is a computation associated with M on input w [2]. We say that M accepts w if this computation terminates in the accepting state, that is, $M(w) = \text{“}yes\text{”}$ [2]. Note that M fails to accept w either if this computation ends in the rejecting state, or if the computation fails to terminate [2].

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 [2]. 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 [2]. We say that M runs in polynomial time if there exists k such that for all n, $T_M(n) \leq n^k + k$ [2].

▶ **Definition 1.** A language L is in class P when $L = L(M)$ for some deterministic Turing machine M which runs in polynomial time [2].

We state the complexity class NP using the following definition.

▶ **Definition 2.** 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 [9]. 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.

Observe that, for polynomial time verifiers, the certificate is polynomially bounded by the length of w, because that is all the verifier can access in its time bound [9].

▶ **Definition 3.** NP is the class of languages that have polynomial time verifiers [9].

In addition, we can define another complexity class called UP.

▶ **Definition 4.** A language L is in UP if every instance of L with a given certificate can be verified by a polynomial time verifier, and this verifier machine only accepts at most one certificate for each problem instance [10]. More formally, a language L belongs to UP if there exists a polynomial time verifier M and a constant c such that

if $x \in L$, then there exists a unique certificate y with $|y| = O(|x|^c)$ such that $M(x, y) =$ "*yes*",

if $x \notin L$, there is no certificate y with $|y| = O(|x|^c)$ such that $M(x, y) =$ "*yes*" [10].

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 [9]. 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 exists 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* [9]. A language $L \subseteq \{0, 1\}^*$ is *NP–complete* if

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

If any single *NP–complete* problem can be solved in polynomial time, then every NP problem has a polynomial time algorithm [5]. No polynomial time algorithm has yet been discovered for any *NP–complete* problem [1].

6 Results

▶ **Definition 5.** *QUADRATIC CONGRUENCES*

INSTANCE: Positive integers a, b and c, such that we have the prime factorization of b where b is not prime.

QUESTION: Is there a positive integer x such that $x < c$ and $x^2 \equiv a (mod\ b)$?

We denote this problem as QC. $QC \in NP\text{–}complete$ [7]. Indeed, Schoof (1985) gives an algorithm for finding x with running time $O(\ln^{10} b)$ when b is prime [15].

▶ **Theorem 6.** *$QC \in P$.*

Proof. We will show QC can be verified by a polynomial time verifier M, and this verifier machine only accepts at most one certificate for each problem instance [10]. Given a certificate x for the instance $\langle a, b, c \rangle$ of the problem QC, our polynomial time verifier M will verify whether x is the minimum positive integer such that x complies with $x < c$ and

$x^2 \equiv a(mod\ b)$. Certainly, if $\langle a, b, c \rangle \in QC$, then there exists a unique certificate x with $|x| = O(|\langle a, b, c \rangle|^c)$ such that $M(\langle a, b, c \rangle, x) =$ "*yes*" where c is a constant and $|\ldots|$ is the bit-length function. The uniqueness of the certificate is valid since there must be only one minimum positive integer x which complies with $x < c$ and $x^2 \equiv a(mod\ b)$.

Moreover, x is polynomially bounded by $\langle a, b, c \rangle$, because of the itself definition of the problem QC as an NP language. Furthermore, if $\langle a, b, c \rangle \notin QC$, there is no certificate x with $|x| = O(|\langle a, b, c \rangle|^c)$ such that $M(\langle a, b, c \rangle, x) =$ "*yes*". Actually, this will be true due to when $\langle a, b, c \rangle \notin QC$ then there is no possible positive integer x, which is the minimum or not, such that $x < c$ and $x^2 \equiv a(mod\ b)$. Finally, we will only need to prove the verifier M decides in polynomial time. Given a certificate x for the instance $\langle a, b, c \rangle$ of the problem QC, we can check in polynomial time whether $x < c$ and $x^2 \equiv a(mod\ b)$ because QC is in NP. Now, how M can verify when x is the minimum positive integer such that $x < c$ and $x^2 \equiv a(mod\ b)$?

Suppose that x is not the minimum. Therefore, there must be a positive integer i such that $0 \leq i < x$ and $i^2 \equiv a(mod\ b)$. However, this will also imply $x^2 - i^2 \equiv (x - i) \times (x + i) \equiv 0(mod\ b)$ by the properties of congruences [13]. Hence, this will imply $x - i$ or $x + i$ is multiple of b or the two numbers are factors of a multiple of b [13]. Indeed, if we want to check whether a positive integer x is the minimum such that $x < c$ and $x^2 \equiv a(mod\ b)$, then we will need to verify whether there is no possible positive integer i such that $0 \leq i < x$ and $x - i$ or $x + i$ is multiple of b or the two numbers are factors of a multiple of b. Nevertheless, we cannot verify this if $2 \times x - 1 \geq b$. Certainly, in the case of $2 \times x - 1 \geq b$: If $x = b$, we can take the positive integer $i = 0$ as a disqualification or if $x \neq b$, then we can take some candidate i such that $x - i = b$ when $x > b$ or some candidate i such that $x + i = b$ when $x < b$. In addition, this positive integer i that we can take as a disqualification will always comply with $0 \leq i < x$. In this way, when $2 \times x - 1 < b$ then there is no possible positive integer i such that $0 \leq i < x$ and $x - i$ or $x + i$ is multiple of b.

Therefore, if $2 \times x - 1 < b$, then the only chance to find a disqualification is when the two numbers $x - i$ and $x + i$ are factors of a multiple of b where $0 \leq i < x$. However, this can be checked in polynomial time because we have the prime factorization of b. We already know the sum of both numbers $x - i$ and $x + i$ is equal to $2 \times x$. We could split the number b within two factors $a_1 > 1$ and $a_2 > 1$ such that $a_1 \times a_2 = b$ and $a_1 \times x_1 + a_2 \times x_2 = 2 \times x$ where x_1 and x_2 are positive integers greater than 0. We can find the possible values of x_1 and x_2 in a feasible way, because this kind of Diophantine equations can be solved in polynomial time [7]. Certainly, if there is a solution of some positive integers $x_1 > 0$ and $x_2 > 0$ which satisfies the equation $a_1 \times x_1 + a_2 \times x_2 = 2 \times x$ such that $a_1 > 1$ and $a_2 > 1$ comply with $a_1 \times a_2 = b$, then x will not be the minimum certificate since there will exist another positive integer $0 \leq i < x$ where $x - i = a_1 \times x_1$ and $x + i = a_2 \times x_2$.

However, the number of distinct divisors a_1 and a_2 of b that we must check with that kind of equation is polynomially bounded by the logarithm of b [17]. In general, if b is written as the product of prime factors: $b = p^{n_p} \times q^{n_q} \times r^{n_r} \ldots$ then the number of distinct divisors is equal to $(n_p + 1) \times (n_q + 1) \times (n_r + 1) \ldots$ [17]. In fact, Hardy proved that a "typical" number b has about $\log \log b$ distinct divisors [17]. Only a tiny proportion has many more divisors than this [17]. Since we can check in polynomial time when x is not the minimum, then our verifier M will be polynomial and has the properties that guarantee the problem QC can be considered as a UP language.

QC is solvable in polynomial time if $c = \infty$ when the prime factorization of b is given [7]. Therefore, we can find in polynomial time a positive integer x that is polynomially bounded by $\langle a, b, c \rangle$ such that $x^2 \equiv a(mod\ b)$. If this solution x complies with $x < c$, then we can accept $\langle a, b, c\rangle \in QC$ in polynomial time. However, when $x \geq c$, then we can use our polynomial verifier M to check whether x is the minimum such that $x < c'$ and $x^2 \equiv a(mod\ b)$ where $c' = x + 1$. On the one hand, if x is the minimum, then we can reject $\langle a, b, c\rangle \notin QC$ in polynomial time. On the other hand, if x is not the minimum, then it will exist another positive integer i such that $0 \leq i < x$ and $i^2 \equiv a(mod\ b)$.

In the case of $2 \times x - 1 \geq b$: If $x = b$, we can take the positive integer $i' = 0$ as a possible value or if $x \neq b$, then we can take some candidate i' such that $x - i' = b \times k$ or $x + i' = b \times k$ where $b \times k$ is the closest multiple of b in relation to x. Certainly, we can find the closest multiple of b in relation to x that is greater than 0 within the integers $\lfloor \frac{x}{b} \rfloor \times b$ when $\lfloor \frac{x}{b} \rfloor > 0$ or $\lceil \frac{x}{b} \rceil \times b$. Consequently, this can be done in polynomial time. At the same time, we can find the minimum possible value of i'' through all the solutions of the Diophantine equations $a_1 \times x_1 + a_2 \times x_2 = 2 \times x$ for every pair of divisors a_1 and a_2 of b such that $a_1 \times a_2 = b$, $x - i = a_1 \times x_1$ and $x + i = a_2 \times x_2$ where x_1 and x_2 are positive integers greater than 0. In addition, this can be solved in polynomial time since the amount of Diophantine equations that we must solve is poly-logarithmic in relation to b since the number of distinct divisors of b is poly-logarithmic on b. Moreover, this kind of Diophantine equations can be solved in polynomial time [7].

Thus, we select the minimum possible value of i such that $0 \leq i < x$ and $i^2 \equiv a(mod\ b)$ from the candidates i' and i'' that we found. Since we select the minimum possible value of i such that $0 \leq i < x$ and $i^2 \equiv a(mod\ b)$, then we accept $\langle a, b, c\rangle$ in QC if and only if $i < c$. In this way, we can guarantee the problem QC belongs to P, because the search of the possible values of i' and i'' can be done in polynomial time. ◀

▶ **Theorem 7.** *$P = NP$.*

Proof. Since QC is complete for NP, thus all language in NP will reduce to P [14]. Since P is closed under reductions, it follows that $P = NP$ [14]. ◀

References

1 Scott Aaronson. P $\overset{?}{=}$ NP. *Electronic Colloquium on Computational Complexity, Report No. 4*, 2017.

2 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

3 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.

4 Stephen A Cook. The P versus NP Problem, April 2000. at `http://www.claymath.org/sites/default/files/pvsnp.pdf`.

5 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

6 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.

7 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.

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

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

10 Lane A. Hemaspaandra and Jorg Rothe. Unambiguous Computation: Boolean Hierarchies and Sparse Turing-Complete Sets. *SIAM J. Comput.*, 26(3):634–653, 2006. doi:10.1137/S0097539794261970.

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

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

13 Trygve Nagell. *Introduction to Number Theory.* New York: Wiley, 1951.

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

15 René Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of computation*, 44(170):483–494, 1985.

16 Leslie G. Valiant. Relative Complexity of Checking and Evaluating. *Information Processing Letters*, 5:20–23, 1976.

17 David G. Wells. *Prime numbers: the most mysterious figures in math.* John Wiley & Sons, Inc., 2005.