UP versus NP

Frank Vega

Joysonic, Uzun Mirkova 5, Belgrade, 11000, Serbia

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. Another major complexity class is UP. Whether UP = NP is another fundamental question that it is as important as it is unresolved. To attack the UP = NP question the concept of NP-completeness is very useful. If any single NP-complete problem is in UP, then UP = NP. Quadratic Congruences is a well-known NP-complete problem. We prove Quadratic Congruences is also in UP. In this way, we demonstrate that UP = NP.

Keywords: P, NP, UP, NP-complete, Quadratic Congruences
2000 MSC: 68Q15, 11A07

Introduction

P versus *NP* is a major unsolved problem in computer science [1]. This problem was introduced in 1971 by Stephen Cook [2]. 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].

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 [3]. 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 [3].

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 [4]. 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].

In the computational complexity theory, the class *P* contains those languages that can be decided in polynomial time by a deterministic Turing machine [5]. The class *NP* consists in those languages that can be decided in polynomial time by a nondeterministic Turing machine [5].

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

The biggest open question in theoretical computer science concerns the relationship between these classes: Is P equal to NP? In 2002, a poll of 100 researchers showed that 61 believed that the answer was not, 9 believed that the answer was yes, and 22 were unsure; 8 believed the question may be independent of the currently accepted axioms and so impossible to prove or disprove [6].

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 [7]. It is obvious that $P \subseteq UP \subseteq NP$ [3]. Whether $P = UP$ is another fundamental question that it is as important as it is unresolved [3]. All efforts to solve the P versus UP problem have failed [3]. Nevertheless, we prove $UP = NP$.

1. 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, $M(w) =$ "yes" [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 1.1. *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 1.2. *A verifier for a language L is a deterministic Turing machine M, where*

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

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 [8]. 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 [8].

Definition 1.3. *NP is the class of languages that have polynomial time verifiers [8].*

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

Definition 1.4. *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 [9]. 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" [9].

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 [8]. 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 $x \in \{0, 1\}^*$,

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

An important complexity class is *NP–complete* [5]. 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 [4]. No polynomial time algorithm has yet been discovered for any *NP–complete* problem [1].

2. Results

Definition 2.1. *QUADRATIC CONGRUENCES*
INSTANCE: Positive integers a, b and c.
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 ∈ NP–complete [10].

Theorem 2.2. *$QC \in UP$.*

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 [9]. 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 also be true due to when $\langle a, b, c\rangle \notin QC$ then there is no possible positive integer x, that 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 $x^2 - i^2 \equiv (x - i) * (x + i) \equiv 0(mod\ b)$ by the properties of congruences [11]. Hence, this will imply $x - i$ or $x + i$ is multiple of b [11]. In this way, 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 only 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. Nevertheless, this is true if and only if $2 \times x - 1 < b$. On the one hand, consider the case when $2 \times x - 1 \geq b$: 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 $0 \leq i < x$. On the other hand, 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. Since we can check whether the given certificate x complies with $2 \times x - 1 < b$ in polynomial time, then our verifier M will be polynomial and has the properties that guarantee the problem QC can be considered as a UP language. □

Theorem 2.3. $UP = NP$.

Proof. Since QC is complete for NP, thus all language in NP will reduce to UP [3]. Since UP is closed under reductions, it follows that $UP = NP$ [3]. □

Conclusions

There is a previous known result which states that $P = UP$ if and only if there are no one-way functions [3]. Indeed, for many years it has been accepted the P versus UP question as the correct complexity context for the discussion of the cryptography and one-way functions [3]. For that reason, the proof of Theorem 2.3 negates this current idea and also the belief that $UP = NP$ is a very unlikely event. In addition, this demonstration might be a shortcut to prove $P = NP$, because if anybody proves that $P = UP$, then this person would be proving the outstanding and difficult P versus NP problem at the same time [1]. Furthermore, if we obtain a possible proof of $P \neq NP$, then this work would also contribute to show $P \neq UP$.

References

[1] L. Fortnow, The Golden Ticket: P, NP, and the Search for the Impossible, Princeton University Press. Princeton, NJ, 2013.

[2] S. Arora, B. Barak, Computational complexity: A modern approach, Cambridge University Press, 2009.

[3] C. H. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.

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

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

[6] W. I. Gasarch, The P=?NP poll, SIGACT News 33 (2) (2002) 34–47.

[7] L. G. Valiant, Relative Complexity of Checking and Evaluating, Information Processing Letters 5 (1976) 20–23.

[8] M. Sipser, Introduction to the Theory of Computation, 2nd Edition, Thomson Course Technology, 2006.

[9] L. A. Hemaspaandra, J. Rothe, Unambiguous Computation: Boolean Hierarchies and Sparse Turing-Complete Sets, SIAM J. Comput. 26 (3) (2006) 634–653. doi:10.1137/S0097539794261970.

[10] 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.

[11] T. Nagell, Introduction to Number Theory, New York: Wiley, 1951.