UP VERSUS NP

Frank Vega

UP VERSUS NP

FRANK VEGA

Abstract. We define a problem that we call General Quadratic Congruences. We show General Quadratic Congruences is an NP-complete problem. Moreover, we prove General Quadratic Congruences is also in UP. In this way, we demonstrate that UP = NP.

Introduction

P versus NP is a major unsolved problem in computer science [3]. This problem was introduced in 1971 by Stephen Cook [1]. It is considered by many to be the most important open problem in the field [3]. 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 [3].

In 1936, Turing developed his theoretical computational model [1]. 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 [10]. 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 [10].

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 [2]. A complexity class is a set of problems, which are represented as a language, grouped by measures such as the running time, memory, etc [2].

In the computational complexity theory, the class P contains those languages that can be decided in polynomial time by a deterministic Turing machine [6]. The class NP consists in those languages that can be decided in polynomial time by a nondeterministic Turing machine [6].

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 [5].

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 [12]. It is obvious that $P \subseteq UP \subseteq NP$ [10]. Whether $P = UP$ is another fundamental question that it is as important as it is unresolved [10]. All efforts to solve the P versus UP problem have failed [10]. Nevertheless, we prove $UP = NP$.

2000 *Mathematics Subject Classification.* Primary 68Q15, Secondary 11A07.
Key words and phrases. P, NP, UP, NP-complete, Quadratic Congruences.

1. Theoretical notions

Let Σ be a finite alphabet with at least two elements, and let Σ^* be the set of finite strings over Σ [1]. A Turing machine M has an associated input alphabet Σ [1]. For each string w in Σ^* there is a computation associated with M on input w [1]. We say that M accepts w if this computation terminates in the accepting state, that is, $M(w) =$ "*yes*" [1]. Note that M fails to accept w either if this computation ends in the rejecting state, or if the computation fails to terminate [1].

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 [1]. 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 [1]. We say that M runs in polynomial time if there exists k such that for all n, $T_M(n) \leq n^k + k$ [1].

Definition 1.1. *A language L is in class P if $L = L(M)$ for some deterministic Turing machine M which runs in polynomial time* [1].

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 [11]. *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 [11].

Definition 1.3. *NP is the class of languages that have polynomial time verifiers* [11].

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* [8]. *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*"* [8].

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

$$x \in L_1 \text{ } \textit{iff} \text{ } f(x) \in L_2$$

where *iff* means "if and only if". An important complexity class is *NP–complete* [6]. A language $L \subseteq \{0,1\}^*$ is *NP–complete* if

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

Furthermore, if L is a language such that $L' \leq_p L$ for some $L' \in$ *NP–complete*, then L is in *NP–hard* [2]. Moreover, if $L \in NP$, then $L \in$ *NP–complete* [2]. If any single *NP–complete* problem can be solved in polynomial time, then every NP problem has a polynomial time algorithm [2]. No polynomial time algorithm has yet been discovered for any *NP–complete* problem [3].

2. Results

Definition 2.1. *Given five positive integers a, b ,c, d and x, the boolean function* $Q(a,b,c,d,x)$ *is true if and only if* $x < c$ *and* $d \times x^2 \equiv a(\mod b)$ [9].

Definition 2.2. *QUADRATIC CONGRUENCES*
INSTANCE: Positive integers a, b and c, such that we have the prime factorization of b.
QUESTION: Is there a positive integer x such that $Q(a,b,c,1,x) = true$*?*
We denote this problem as QC. QC ∈ NP–complete [4].

Let's define another problem.

Definition 2.3. *GENERAL QUADRATIC CONGRUENCES*
INSTANCE: Positive integers a, b, c and d, such that we have the prime factorization of b.
QUESTION: Is there a positive integer x such that $Q(a,b,c,d,x) = true$*?*
We denote this problem as GQC.

Theorem 2.4. *GQC ∈ NP–complete.*

Proof. Since we can check $Q(a,b,c,d,x) = true$ in polynomial time, then $GQC \in NP$. Indeed, the certificate x will be polynomially bounded by any instance (a,b,c,d) when $Q(a,b,c,d,x) = true$ because $x < c$. In addition, we can reduce every instance (a,b,c) of QC into an instance $(a,b,c,1)$ of GQC in polynomial time where

$$(a,b,c) \in QC \text{ } \textit{iff} \text{ } (a,b,c,1) \in GQC.$$

Since $QC \in$ *NP–complete* then $GQC \in$ *NP–complete.* □

The distinct prime factors of a positive integer $n >= 2$ are defined as the $\omega(n)$ numbers $p_1, \ldots, p_{\omega(n)}$ in the prime factorization

$$n = p_1^{a_1} \times p_2^{a_2} \times \ldots \times p_{\omega(n)}^{a_{\omega(n)}}.$$

Lemma 2.5. *There will exist a constant* α*, such that there are infinite positive integers n which complies with* $\omega(n) \leq \alpha \times \ln\ln n$*.*

Proof. The average order of $\omega(n)$ is $\omega(n) \sim \ln\ln n$ [7]. Consequently, it will exist the constant α. □

Theorem 2.6. *Given four positive integers a, b, c and d, such that we have the prime factorization of b and $\omega(b) \leq \alpha \times \ln\ln b$, then we can check whether a positive integer x is the minimum that complies $Q(a,b,c,d,x) = true$ in order $O(\ln^k b)$ for a constant k.*

Proof. Suppose we have a positive integer i such that $0 < i < x$ and $Q(a,b,c,d,i) = true$. Hence, we will obtain $d \times x^2 \equiv d \times i^2 (\mod b)$. Moreover, by a property of congruences we have $x^2 \equiv i^2 (\mod b')$ where $b' = \frac{b}{(d,b)}$ and (d,b) is the greatest common divisor of d and b [9]. We can find (d,b) in polynomial time in relation to $\ln b$ just multiplying into a single number each maximum prime power $p_i^{e_i}$ that divides b when also $p_i^{e_i}$ divides d. This is possible because we have the prime factorization of b. We are going to assume $b' \neq 1$, because in case of $(d,b) = b$ then x should be necessarily equal to 1.

If the congruence $x^2 \equiv i^2 (\mod b')$ has a solution, that solution is necessarily a solution to each of the prime power congruences $x^2 \equiv i^2 (\mod p_i^{e_i})$ when $p_i^{e_i}$ divides b' [9]. For any prime p_r, a necessary condition for $x^2 \equiv i^2 (\mod p_r^{e_r})$ to have a solution is for $x^2 \equiv i^2 (\mod p_r)$ to have a solution (to see this, note that if $x^2 - i^2$ is divisible by $p_r^{e_r}$ then it is certainly divisible by p_r).

Now, suppose $x^2 \equiv i^2 (\mod p_r^{e_r})$ where $p_r^{e_r}$ is a prime power which divides b'. Then $x^2 - i^2 \equiv (x-i) \times (x+i) \equiv 0 (\mod p_r^{e_r})$. Thus $p_r^{e_r}$ divides the product $(x-i) \times (x+i)$ and so p_r divides the product as well. If $p_r = 2$ and p_r divides $(x-i) \times (x+i)$, then this is because $x \equiv i (\mod p_r)$ since the sum and the subtraction of two integers is even when both are even or odd at the same time. If p_r is an odd prime and divides both $(x-i)$ and $(x+i)$, then p_r would divide both their sum and their difference, $2 \times x$ and $-2 \times i$. Since p_r is an odd prime, p_r does not divide 2 and so p_r would divide both x and i which can be translated to $x \equiv i (\mod p_r)$. It follows that p_r either divides $(x-i)$ or $(x+i)$ but not both. Since p_r divides $(x-i) \times (x+i)$, it only divides one of $(x-i)$ and $(x+i)$. Therefore, either $x \equiv i (\mod p_r)$ or $x \equiv -i (\mod p_r)$.

In this way, we prove for every prime p_r that divides b' we will have either $x \equiv i (\mod p_r)$ or $x \equiv -i (\mod p_r)$. Conversely, if we find all the possible solutions to each of the prime congruences, then we can use the Chinese Remainder Theorem to produce a solution to the original problem, that is to find the value of i [2]. Since the Chinese Remainder Theorem can be solved in polynomial time $O(\ln^\beta b)$, then the remaining order will depend on the computation of all possible solutions. Since we only have two possible choices for each prime factor, then the order will depend on $O(2^{\omega(b')})$. Since $\omega(b') \leq \omega(b) \leq \alpha \times \ln\ln b$, then the final order will be of $O(\ln^\beta b \times 2^{\alpha \times \ln\ln b}) = O(\ln^\beta b \times \ln^\alpha b) = O(\ln^k b)$ for a constant $k = \beta + \alpha$. □

Definition 2.7. *SIMPLE QUADRATIC CONGRUENCES*

INSTANCE: Positive integers a, b, c and d, such that we have the prime factorization of b and $\omega(b) \leq \alpha \times \ln\ln b$.

QUESTION: Is there a positive integer x such that $Q(a,b,c,d,x) = true$?

We denote this problem as SQC.

Theorem 2.8. *$SQC \in UP$.*

Proof. We show a polynomial time verifier, and this verifier machine only accepts at most one certificate for each problem instance of *SQC* [8]. Given five positive

integers a, b, c, d and x, we define the verifier machine M for SQC as follows:

$$M(a,b,c,d,x) = \text{“}yes\text{”} \textit{ iff } x \textit{ is the minimum such that } Q(a,b,c,d,x) = true.$$

SQC belongs to UP because the verifier M can run in polynomial time as we proved in Theorem 2.6 and there will be a constant e such that

if $(a,b,c,d) \in SQC$, then there is a unique certificate x with $|x| = O(|(a,b,c,d)|^e)$ such that $M(a,b,c,d,x) = \text{“}yes\text{”}$,

if $(a,b,c,d) \notin SQC$, there is no certificate x with $|x| = O(|(a,b,c,d)|^e)$ such that $M(a,b,c,d,x) = \text{“}yes\text{”}$ [8].

The constant e exists because $SQC \in NP$. □

Definition 2.9. *COMPLEX QUADRATIC CONGRUENCES ON I*

INSTANCE: Positive integers a, b, c and d, such that we have the prime factorization of b and $\omega(b) \leq i \times \alpha \times \ln\ln b$ *for a positive integer i.*

QUESTION: Is there a positive integer x such that $Q(a,b,c,d,x) = true$*?*

We denote this problem as CQC_i.

Theorem 2.10. *For every positive integer i we have that* $CQC_i \in UP$.

Proof. For $i = 1$, then $CQC_1 = SQC$ and thus $CQC_1 \in UP$. Suppose for some $i = k$, then $CQC_k \in UP$. Let's prove $CQC_{k+1} \in UP$. We will take an arbitrary instance (a,b,c,d) and some prime number $p > 2$ which does not divide b. The prime p can be taken in polynomial time in relation to $\log_2 b$. Certainly, this can be done choosing a candidate from 3 to $\log_2^2 b$ because $\omega(b) \leq \log_2 b$ and the nth prime number is approximately equal to $n \times \ln n < n^2$ [9]. Let's take the number $q = p^{\lceil \ln^2 b \rceil}$. Since the congruence property

$$d \times x^2 \equiv a(\mod b)$$

complies with

$$q \times d \times x^2 \equiv q \times a(\mod q \times b)$$

then $Q(a,b,c,d,x) = true$ if and only if $Q(q \times a, q \times b, c, q \times d, x) = true$.

However, if the instance $(a,b,c,d) \in CQC_{k+1}$, then the instance $(q \times a, q \times b, c, q \times d) \in CQC_k$ because $\omega(q \times b) = \omega(b) + 1 \leq (k+1) \times \alpha \times \ln\ln b + 1$. Since $p > 2$ then $q = p^{\lceil \ln^2 b \rceil} > b^{\ln b}$. Therefore $k \times \alpha \times \ln\ln(q \times b) > k \times \alpha \times \ln\ln b^{\ln b}$ and this complies for $k > 1$ with $k \times \alpha \times \ln\ln b^{\ln b} = k \times \alpha \times \ln(\ln b \times \ln b) = k \times \alpha \times \ln\ln^2 b = 2 \times k \times \alpha \times \ln\ln b > (k+1) \times \alpha \times \ln\ln b + 1 \geq \omega(q \times b)$.

In this way, we can reduce in polynomial time CQC_{k+1} to CQC_k, since the calculation of q will be polynomial in relation to $\ln b$ if we use the exponentiating by squaring [2]. Since UP is closed under reductions and $CQC_k \in UP$, it follows that $CQC_{k+1} \in UP$. Hence, by mathematical induction we have proved $CQC_i \in UP$ for every positive integer i [9]. □

Definition 2.11. *A Turing machine M has an infinite bit length, when M is encoded by the binary alphabet of the Universal Turing machine, if the size of the program of M is infinite.*

Theorem 2.12. *If a language L is not in UP, then it would have a polynomial time verifier* M_j *of infinite bit length such that this verifier machine only accepts at most one certificate for each problem instance of L.*

Proof. In this case, the certificate is not longer useful and thus we can always pass a single constant string ϵ as certificate such that we will have $M_j(x,\rho) =$ "yes" if and only if $x \in L$ and $\rho = \epsilon$ or $M_j(x,\rho) =$ "no" if and only if $x \notin L$ or $\rho \neq \epsilon$. Indeed, $M_j(x,\rho)$ can be decided in polynomial time because M_j could decide every $x \in L$ in polynomial time since it may store an infinite amount of program space for the decision of elements in L and this will be a valid verification too. In addition, for every string ρ, M_j can compute in polynomial time whether $\rho = \epsilon$. Furthermore, since ϵ is a constant string, then there exists a constant e such that $|\epsilon| = O(|x|^e)$ for every $x \in L$. Therefore, this kind of Turing machine M_j of infinite bit length only accepts at most one certificate for each problem instance of L. □

Lemma 2.13. *A language L is in UP if and only if it would have a polynomial time verifier M_j of finite bit length such that this verifier machine only accepts at most one certificate for each problem instance of L.*

Proof. This is a consequence of the self definition of the class UP. □

Theorem 2.14. $GQC \in UP$.

Proof. For every positive integer i the set CQC_i contains infinite elements. Indeed, for some positive integer b there are infinite numbers n such that $\omega(b) = \omega(n)$, because there are infinite prime numbers [9]. Since for every positive integer i we have that $CQC_i \in UP$, then every language CQC_i will have a polynomial time verifier M_i, and this verifier machine only accepts at most one certificate for each problem instance of CQC_i [8]. We denote $l(M_i)$ as the bit length of M_i. We also denote $|CQC_i|$ as the cardinality of CQC_i. Certainly, $\frac{l(M_i)}{|CQC_i|} = 0$ since CQC_i has infinite elements and the bit length of M_i is finite because $CQC_i \in UP$. We get $\lim_{i\to\infty} \frac{l(M_i)}{|CQC_i|} = 0$ and thus $\lim_{i\to\infty} \frac{l(M_i)}{|CQC_i|} = \frac{\lim_{i\to\infty} l(M_i)}{\lim_{i\to\infty} |CQC_i|} = 0$. Moreover, we can assure that $\lim_{i\to\infty} |CQC_i| = |GQC|$, because $\lim_{i\to\infty} CQC_i = GQC$. Therefore, we obtain $\frac{\lim_{i\to\infty} l(M_i)}{|GQC|} = 0$. By the definition of M_i, we get $\lim_{i\to\infty} M_i = M_{GQC}$ where M_{GQC} is a polynomial time verifier such that this verifier machine only accepts at most one certificate for each problem instance of GQC. However, M_{GQC} might have an infinite bit length. Indeed, M_{GQC} has a finite bit length if and only if $GQC \in UP$. Consequently, $\frac{\lim_{i\to\infty} l(M_i)}{|GQC|} = \frac{l(M_{GQC})}{|GQC|} = 0$. Since GQC is infinite then M_{GQC} has a finite bit length and thus $GQC \in UP$. □

Theorem 2.15. $UP = NP$.

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

Conclusions

There is a previous known result which states that $P = UP$ if and only if there are no one-way functions [10]. 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 [10]. For that reason, the proof of Theorem 2.15 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 he will be proving the outstanding and difficult P versus NP problem at the same time [3]. Furthermore, if we obtain a possible proof of $P \neq NP$, then this work would also contribute to show $P \neq UP$.

References

1. Sanjeev Arora and Boaz Barak, *Computational complexity: A modern approach*, Cambridge University Press, 2009.
2. Thomas H. Cormen, Charles Eric Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms*, 2 ed., MIT Press, 2001.
3. Lance Fortnow, *The Golden Ticket: P, NP, and the Search for the Impossible*, Princeton University Press. Princeton, NJ, 2013.
4. Michael R. Garey and David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1 ed., San Francisco: W. H. Freeman and Company, 1979.
5. William I. Gasarch, *The P=?NP poll*, SIGACT News **33** (2002), no. 2, 34–47.
6. Oded Goldreich, *P, Np, and Np-Completeness*, Cambridge: Cambridge University Press, 2010.
7. G. H. Hardy, *Ramanujan: Twelve Lectures on Subjects Suggested by His Life and Work*, 3 ed., New York: Chelsea, 1999.
8. Lane A. Hemaspaandra and Jorg Rothe, *Unambiguous Computation: Boolean Hierarchies and Sparse Turing-Complete Sets*, SIAM J. Comput. **26** (2006), no. 3, 634–653.
9. T. Nagell, *Introduction to Number Theory*, New York: Wiley, 1951.
10. Christos H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.
11. Michael Sipser, *Introduction to the Theory of Computation*, 2 ed., Thomson Course Technology, 2006.
12. Leslie G. Valiant, *Relative Complexity of Checking and Evaluating*, Information Processing Letters **5** (1976), 20–23.

E-mail address: frank.vega@yahoo.com

Joysonic, Uzun Mirkova 5, Belgrade, Serbia