Noname manuscript No.
(will be inserted by the editor)

Quadratic Congruences on Average Case

Frank Vega

the date of receipt and acceptance should be inserted later

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. However, 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 NP-complete. To attack the P versus NP question the concept of NP-completeness has been very useful. The Quadratic Congruences is a known NP-complete problem. We show this problem can be solved in polynomial time for the average case. It is true that Hamilton cycle and some NP-complete problems could be solved in average case over inputs. However, this algorithm, in the same way as Quicksort, is polynomial for a large amount of inputs because of the infinite set of elements that cannot be solved in polynomial time is infinitesimal.

Keywords Complexity Classes · Completeness · Theory of Numbers · Polynomial Time · Certificate

Mathematics Subject Classification (2000) 68Q15 · 68Q17 · 68R01

CR Subject Classification F.1.3 · F.1.3.2 · G.1.0

1 Introduction

The P versus NP problem is a major unsolved problem in computer science [2]. This is considered by many to be the most important open problem in the field [2]. 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

Frank Vega
Joysonic, Uzun Mirkova 5, Belgrade, 11000, Serbia
E-mail: vega.frank@gmail.com

correct solution [2]. It was essentially mentioned in 1955 and 1956 in letters written by John Nash and Kurt Gödel respectively [12], [9]. However, the precise statement of the P=NP problem was introduced in 1971 by Stephen Cook in a seminal paper [2].

In 1936, Turing developed his theoretical computational model [16]. The deterministic and nondeterministic Turing machines have become in two of the most important definitions related to this theoretical model for computation [16]. A deterministic Turing machine has only one next action for each step defined in its program or transition function [16]. 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 [16].

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 [3]. A complexity class is a set of problems, which are represented as a language, grouped by measures such as the running time, memory, etc [3].

The set of languages decided by deterministic Turing machines within time f is an important complexity class denoted $TIME(f(n))$ [13]. In addition, the complexity class $NTIME(f(n))$ consists in those languages that can be decided within time f by nondeterministic Turing machines [13]. 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 constant k [13]. At the same time, NP consists in all languages in $NTIME(n^k)$ for every possible positive constant k [13].

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, that is $M(w) =$ "no", 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]. The notations we use to describe the asymptotic running time of an algorithm are defined in terms of functions whose domains are the set of natural numbers [3]. Such notations are convenient for describing the worst case running time function $T_M(n)$, which is usually defined only on integer input sizes [3]. For a given function $g(n)$, we denote by $\Theta(g(n))$ the set of functions

$$\Theta(g(n)) = \{a(n) : \exists c_1 > 0, c_2 > 0 \ and \ n_0 > 0$$

$$\textit{such that } 0 \leq c_1 \times g(n) \leq a(n) \leq c_2 \times g(n)\ \forall n \geq n_0\}.$$

The Θ-notation asymptotically bounds a function from above and below. When we have only an asymptotic upper bound, we use O-notation. For a given function $g(n)$, we denote by $O(g(n))$ the set of functions

$$O(g(n)) = \{a(n) : \exists c > 0 \textit{ and } n_0 > 0 \textit{ such that } 0 \leq a(n) \leq c \times g(n)\ \forall n \geq n_0\}.$$

We say that M runs in polynomial time if there is a constant k such that for all n, $T_M(n) \leq n^k + k$ [1]. This would be equivalent to say there is a constant k such that M runs in time $O(n^k)$. 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 [3]. 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 [1]. 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 [13].

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 [16]. 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* [7]. 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* [3]. Moreover, if $L \in NP$, then $L \in$ *NP–complete* [3]. If any single *NP–complete* problem can be solved in polynomial time, then every NP problem has a polynomial time algorithm [3]. No polynomial time algorithm has yet been discovered for any *NP–complete* problem [4]. 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 [6]. It is fully expected that $P \neq NP$ [13]. Indeed, if $P = NP$ then there are stunning practical consequences [13]. For that reason, $P = NP$ is considered as a very unlikely event [13]. We prove an interesting result regarding the *NP–complete* class. The Quadratic Congruences is a known *NP–complete* problem [5]. We show this problem can be solved in polynomial time for the average case.

2 Results

Definition 1 Given four positive integers a, b, c and x, the following Boolean function $Q(a, b, c, x)$ is true if and only if $x^2 \equiv a(mod\ b)$ and $x < c$.

Definition 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 $Q(a, b, c, x) = true$?

We denote this problem as QC. $QC \in$ *NP–complete* [5].

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

$$n = p_1^{i_1} \times p_2^{i_2} \times \ldots \times p_{\omega(n)}^{i_{\omega(n)}}.$$

Theorem 1 *Given an instance (a, b, c) of QC, this can be decided in time $O(|a, b|^{\alpha} + \ln^{\beta} b \times 2^{\omega(b)})$ for positive constants α and β where $|\ldots|$ denotes the bit-length function.*

Proof QC is solvable in polynomial time if $c = \infty$ when the prime factorization of b is given [5]. We say this can be solved in $O(|a, b|^{\alpha})$ for a positive constant α. Since we can have a candidate solution x in polynomial time which is not upper bounded by c [14], then we can find another positive integer i such that $i < x$ and $Q(a, b, c, i) = true$. Hence, we obtain $x^2 \equiv i^2(mod\ b)$. If the congruence $x^2 \equiv i^2(mod\ b)$ has a solution, then the solution is necessarily a solution for the prime power congruences $x^2 \equiv i^2(mod\ p_j^{i_j})$ when $p_j^{i_j}$ divides b [8]. For every prime p_r, a necessary condition for $x^2 \equiv i^2(mod\ p_r^{i_r})$ 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 $p_r^{i_r}$ then it is certainly divisible by p_r).

Now, suppose $x^2 \equiv i^2(mod\ p_r^{i_r})$ where $p_r^{i_r}$ is a prime power which divides b. Then $x^2 - i^2 \equiv (x - i) \times (x + i) \equiv 0(mod\ p_r^{i_r})$. Thus $p_r^{i_r}$ 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 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, that is the numbers $(x - i) + (x + i) = 2 \times x$ and $(x - i) - (x + i) = -2 \times i$. 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 that for every prime p_r that divides b we have either $x \equiv i(mod\ p_r)$ or $x \equiv -i(mod\ p_r)$. Conversely, if we find all the possible solutions for each of these prime congruences, then we can use the Chinese Remainder Theorem to produce a solution for the problem of finding the minimum value of i which complies with $Q(a,b,c,i) = true$ [3]. Since the Chinese Remainder Theorem can be solved in polynomial time ($O(\ln^{\beta} b)$ for a positive constant β) [15], then the running time depends mostly on the computation of all possible solutions. Since we have at most two possible choices for each prime factor ($x \equiv i(mod\ p_r)$ or $x \equiv -i(mod\ p_r)$), then the running time is affected directly by $O(2^{\omega(b)})$ in many cases. Therefore, we can verify whether there is any positive integer i within all the analyzed solutions which complies with $Q(a,b,c,i) = true$ or not in $O(|a,b|^{\alpha} + \ln^{\beta} b \times 2^{\omega(b)})$ for the positive constants α and β.

In computational complexity theory, the average case complexity of an algorithm is the amount of some computational resource (typically time) used by the algorithm, averaged over all possible inputs [1]. It is frequently contrasted with worst case complexity which considers the maximal complexity of the algorithm over all possible inputs [1]. The average case performance of algorithms has been studied since modern notions of computational efficiency were developed in the middle of the last century [3]. From the beginning the initial work was focused on problems for which worst case polynomial time algorithms were already known [10]. In 1973, Donald Knuth published an extensively surveys average case performance of algorithms for problems solvable in worst case polynomial time, such as sorting and median-finding [10]. For example *Quicksort*, have a worst case running time of $O(n^2)$, but an average case running time of $O(n \times \log n)$, where n is the length of the input to be sorted [3].

Definition 3 We shall say, roughly, that a function $f(n)$ has the Normal Order $F(n)$ if $f(n)$ is approximately $F(N)$ for almost all values of n [8]. More precisely, suppose that $F(n)$ complies with

$$(1-\epsilon) \times F(n) < f(n) < (1+\epsilon) \times F(n)$$

for every positive ϵ and almost all values of n [8]. There may be an exceptional infinitesimal set of n for which this inequality is false, and this exceptional set will naturally depend upon ϵ [8].

Theorem 2 *Based on Theorem 1, QC can be solved in* $\Theta(|a,b|^{\alpha} + \ln^{\beta+1} b)$ *for the average case.*

Proof The Normal Order of $\omega(n)$ is $\ln \ln n$ [8]. Consequently, for the average case we will have $2^{\omega(b)} = \Theta(2^{\ln \ln b}) = \Theta(\ln b)$ and thus we can guarantee this Theorem.

Theorem 3 *Based on Theorem 1, QC can be solved in* $\Theta(|a,b|^{\alpha} + \ln^{\beta} b \times \sqrt[\ln\ln b]{b})$ *for the worst case.*

Proof The worst case for the value of $\omega(b)$ is when b is a primorial [8]. For the j^{th} prime number p_j, the primorial $p_j\#$ is defined as the product of the first j primes [8]. If a number n is primorial, then $\omega(n) \sim \frac{\ln n}{\ln\ln n}$ [8]. Consequently, for the worst case we will have $2^{\omega(b)} = \Theta(2^{\frac{\ln b}{\ln\ln b}}) = \Theta(\sqrt[\ln\ln b]{b})$ and thus we can guarantee this Theorem.

Lemma 1 *The infinite set of elements in QC that cannot be solved in polynomial time is infinitesimal.*

Proof This is a consequence of the Definition 3 in which the average case is proved [8].

Lemma 2 *The assumption of the Extended Riemann Hypothesis is not negated with this result.*

Proof Assuming the Extended Riemann Hypothesis [17], the problem QC is solvable in polynomial time when b is prime [5]. Each prime number p_r complies with $\omega(p_r) = 1$ and thus $\omega(p_r) \leq k \times \ln\ln p_r$, except for $p_r = 2$ (for $p_r \geq 3$ is enough to take $k = 11$). We can decide when $(a, 2, c) \in QC$ in polynomial time, due to x^2 is an odd number when x is odd respectively [8]. Hence, the Theorem 2 corroborates in some way the Extended Riemann Hypothesis (even though this might still be false), because this shows QC is always solvable in polynomial time when b is prime.

3 Conclusion

An efficient algorithm for *NP–complete* problems is generally characterized as one which runs in polynomial time for all inputs: This means requiring efficient worst case complexity. However, an algorithm which is inefficient on a "small" number of inputs may still be efficient for "most" inputs that occur in practice [1]. Thus, it is desirable to study the properties of these algorithms where the average case complexity may differ from the worst case complexity and find methods to relate the two [1].

The fundamental notions of average case complexity were developed by Leonid Levin in 1986 [11]. He defined the average case complexity and completeness while giving an example of a complete problem for $distNP$, the average case analogue of NP [11]. The equivalent average case analogue for P is called $distP$ [1]. There are several results regarding this topic [1]. However, our result shows there is another *NP–complete* which is "easy on average" and therefore, we think we open another path in the analysis on the expected complexity field [3].

References

1. Arora, S., Barak, B.: Computational complexity: a modern approach. Cambridge University Press (2009)
2. Cook, S.A.: The P versus NP Problem (2000). Available at Millennium Prize Problems web site http://www.claymath.org/sites/default/files/pvsnp.pdf
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Third Edition, 3rd edn. The MIT Press (2009)
4. Fortnow, L.: The status of the p versus np problem. Communications of the ACM **52**(9), 78–86 (2009)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness, 1 edn. San Francisco: W. H. Freeman and Company (1979)
6. Gasarch, W.I.: Guest column: The second p=? np poll. ACM SIGACT News **43**(2), 53–77 (2012)
7. Goldreich, O.: P, NP, and NP-Completeness: The basics of computational complexity. Cambridge University Press (2010)
8. Hardy, G.H., Wright, E.M.: An introduction to the theory of numbers. Oxford University Press (1979)
9. Hartmanis, J.: Gödel, von neumann, and the p = np problem. Bulletin of the European Association for Theoretical Computer Science **38**, 101–107 (1989)
10. Knuth, D.: The Art of Computer Programming, vol. 3. Addison-Wesley (1973)
11. Levin, L.A.: Average case complete problems. SIAM Journal on Computing **15**(1), 285–286 (1986)
12. Nash, J.: Letter to the united states national security agency (1955). Available at NSA web site https://www.nsa.gov/news-features/declassified-documents/nash-letters/assets/files/nash_letters1.pdf
13. Papadimitriou, C.H.: Computational complexity. Addison-Wesley (1994)
14. Schoof, R.: Elliptic curves over finite fields and the computation of square roots mod p. Mathematics of computation **44**(170), 483–494 (1985)
15. Séroul, R.: Programming for mathematicians. Springer Science & Business Media (2012)
16. Sipser, M.: Introduction to the Theory of Computation, vol. 2. Thomson Course Technology Boston (2006)
17. Wells, D.G.: Prime numbers: the most mysterious figures in math. John Wiley & Sons, Inc. (2005)