P versus NP on NP-completeness

FRANK VEGA, Joysonic, Serbia

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. We prove some interesting results regarding this class which affect directly to the P versus NP problem. Indeed, this study provides true facts that might address to think in the possibility of a definitely P = NP answer.

CCS Concepts: • **Theory of computation → Complexity classes**; *Problems, reductions and completeness*; • **Mathematics of computing →** Number-theoretic computations;

Additional Key Words and Phrases: Complexity Classes, Completeness, Theory of Numbers, Polynomial Time, Certificate

1 INTRODUCTION

The *P* versus *NP* problem is a major unsolved problem in computer science [3]. This 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]. It was essentially mentioned in 1955 and 1956 in letters written by John Nash and Kurt Gödel respectively [18], [11]. However, the precise statement of the P=NP problem was introduced in 1971 by Stephen Cook in a seminal paper [3].

In 1936, Turing developed his theoretical computational model [22]. The deterministic and nondeterministic Turing machines have become in two of the most important definitions related to this theoretical model for computation [22]. A deterministic Turing machine has only one next action for each step defined in its program or transition function [22]. 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 [22].

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 [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].

The set of languages decided by deterministic Turing machines within time f is an important complexity class denoted $TIME(f(n))$ [19]. In addition, the complexity class $NTIME(f(n))$ consists in those languages that can be decided within time f by nondeterministic Turing machines [19]. 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 [19]. At the same time, NP consists in all languages in $NTIME(n^k)$ for every possible positive constant k [19].

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].

Author's address: Frank Vega, Joysonic, Krunska 46, Belgrade, 11000, Serbia, vega.frank@gmail.com.

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 [4]. Such notations are convenient for describing the worst case running time function $T_M(n)$, which is usually defined only on integer input sizes [4]. For a given function $g(n)$, we denote by $\Theta(g(n))$ the set of functions

$$\Theta(g(n)) = \{a(n) : \exists c_1 > 0, c_2 > 0 \text{ and } n_0 > 0 \text{ such that } 0 \leq c_1 \times g(n) \leq a(n) \leq c_2 \times g(n)\ \forall n \geq n_0\}.$$

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

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

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 [4]. 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 [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 [19].

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

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

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

- $L \in NP$, 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* [4]. Moreover, if $L \in NP$, then $L \in$ *NP–complete* [4]. A principal *NP–complete* problem is *SAT* [7]. 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), $\rightarrow$(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 [7]. 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 [4]. 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 [4]. A Boolean formula is in 3-conjunctive normal form or 3*CNF*, if each clause has exactly three distinct literals [4].

For example, the Boolean formula

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

is in 3*CNF*. The first of its three clauses is $(x_1 \vee \rightharpoonup x_1 \vee \rightharpoonup x_2)$, which contains the three literals x_1, $\rightharpoonup x_1$, and $\rightharpoonup x_2$. Another relevant *NP–complete* language is 3*CNF* satisfiability, or 3*SAT* [4]. In 3*SAT*, it is asked whether a given Boolean formula ϕ in 3*CNF* is satisfiable. *HAMILTONIAN CIRCUIT* is another famous *NP–complete* problem [7]. An instance of *HAMILTONIAN CIRCUIT* is a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges, each edge being an unordered pair of vertices [7]. We say $(u, v) \in E$ is an edge in a graph $G = (V, E)$ where u and v are vertices. For a graph $G = (V, E)$ a simple circuit in G is a sequence of distinct vertices $< v_0, v_1, v_2, ..., v_k >$ such that $(v_k, v_0) \in E$ and $(v_{i-1}, v_i) \in E$ for $i = 1, 2, ..., k$ [4]. A Hamiltonian circuit is a simple circuit of the graph which contains all the vertices of the graph. The problem *HAMILTONIAN CIRCUIT* asks whether a graph has a Hamiltonian circuit [7].

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 [6]. 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 $P \neq NP$ [19]. Indeed, if $P = NP$ then there are stunning practical consequences [19]. For that reason, $P = NP$ is considered as a very unlikely event [19].

We prove some interesting results regarding the *NP–complete* class. On the one hand, the Quadratic Congruences is a known *NP–complete* problem [7]. We show this problem can be solved in polynomial time for the average case. Indeed, this proof validates one special kind of instances in this problem which was previously assumed that run in polynomial time under the assumption of the Extended Riemann Hypothesis. On the other hand, the *OTHER–L* is a problem of deciding given a solution y of an instance x in a language L whether x has another solution different of y. The class *OTHER–NP* contains those languages *OTHER–L* where L is in *NP*. We show a previous known language *OTHER–L* which is *NP–complete* and L is in *NP–complete* too. Moreover, we show *OTHER–SAT* and *OTHER–3SAT* are complete for *OTHER–NP*. Furthermore, we prove there is one language L that is *NP–complete* and the equivalent *OTHER–L* is in *P*.

2 RESULTS

2.1 P versus NP for the Average Case

Definition 2.1. 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 $x < c$ [10].

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 $Q(a, b, c, x) = true$?
We denote this problem as QC. $QC \in NP\text{–}complete$ [16].

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 2.3. *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 [7]. 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 [20], 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 [10]. 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 $Q(a, b, c, i) = true$ [4]. Since the Chinese Remainder Theorem can be solved in polynomial time ($O(\ln^{\beta} b)$ for a positive constant β) [21], 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]. Some work relating to average case complexity focused on problems for which polynomial time algorithms already existed, such as sorting. 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 [4]. 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 [10]. More precisely, suppose that

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

for every positive constant ϵ and almost all values of n [10]. There may be an exceptional infinitesimal set of n for which this inequality is false, and this exceptional set will naturally depend upon ϵ [10].

THEOREM 2.4. *Based on Theorem 2.3, 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 $\ln\ln n$ [10]. 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 2.5. *Based on Theorem 2.3, 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 [10]. For the j^{th} prime number p_j, the primorial $p_j\#$ is defined as the product of the first j primes [10]. If a number n is primorial, then $\omega(n) \sim \frac{\ln n}{\ln\ln n}$ [10]. 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 2.6. *The assumption of the Extended Riemann Hypothesis is not negated with this result.*

PROOF. Assuming the Extended Riemann Hypothesis, the problem QC is solvable in polynomial time when b is prime [7]. 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 [10]. Hence, the Theorem 2.4 corroborates in some way the Extended Riemann Hypothesis even though this might still be false. □

2.2 The Complexity of OTHER-NP

Definition 2.7. The class *OTHER–NP* contains those languages whether it is asked when an instance x with a given certificate y from a specific language in NP has another certificate different of y. More formally, a language L is in *OTHER–NP* if there is another $L' \in NP$ such that

$$L = \{(x,y) : \textit{where } M(x,y) = \text{“}yes\text{”} \textit{ and } \exists z \in \{0,1\}^* \textit{ such that } y \neq z \textit{ and } M(x,z) = \text{“}yes\text{”}\}$$

where M is the polynomial time verifier of L'. We denote this class as ONP.

Definition 2.8. OTHER–HAMILTONIAN CIRCUIT
INSTANCE: A graph G and a Hamiltonian circuit of G.
QUESTION: Has G another Hamiltonian circuit?
We denote this problem as OHC. $OHC \in$ *NP–complete* [19].

Definition 2.9. The completeness from *ONP* is defined as a language $L \subseteq \{0,1\}^*$ such that,

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

If L is a language such that $L' \leq_p L$ for some L' in *ONP–complete*, then L is *ONP–hard*. Moreover, if $L \in ONP$, then $L \in$ *ONP–complete*.

Definition 2.10. OTHER–SAT
INSTANCE: A Boolean formula ϕ in CNF and a satisfying truth assignment.
QUESTION: Has ϕ another satisfying truth assignment?
We denote this problem as $OSAT$.

Theorem 2.11. *OSAT ∈ ONP–complete.*

Proof. We can reduce every nondeterministic Turing machine which accepts an input string $x \in \{0,1\}^*$ in polynomial time into a polynomially bounded and satisfiable Boolean formula ϕ in CNF using the Cook's Theorem [7]. Consequently, given a language $L \in ONP$ we can transform the polynomial time verifier M from the respective $L' \in NP$ into a nondeterministic Turing machine N that guesses a string y in linear time and then check whether this y is a certificate for an input x in a deterministic polynomial time. Since we already know the value of the string $y \in \{0,1\}^*$ in the language $L \in ONP$, then the guesser process will be deterministic as well and therefore we will obtain a Boolean formula ϕ in CNF and also a satisfying truth assignment T for this formula after making the reduction of Cook's Theorem [7]. In this way, we will have

$$(x, y) \in L \textit{ if and only if } (\phi, T) \in OSAT$$

and thus $OSAT \in ONP$–*complete.* □

Definition 2.12. OTHER–3SAT
INSTANCE: A Boolean formula ϕ in $3CNF$ and a satisfying truth assignment.
QUESTION: Has ϕ another satisfying truth assignment?
We denote this problem as $O3SAT$.

Theorem 2.13. *O3SAT ∈ ONP–complete.*

Proof. In the reduction of a Boolean formula ϕ in CNF to another formula ϕ' in $3CNF$ such that ϕ is satisfiable if and only if ϕ' is satisfiable, we introduce some new two variables p and q for clauses of less than 3 literals where for any assignment for these variables both formulas remain algebraically and effectively equivalents [4]. For example, for a clause of one literal (x) we create

$$(x \vee \rightharpoonup p \vee \rightharpoonup q) \wedge (x \vee p \vee q) \wedge (x \vee \rightharpoonup p \vee q) \wedge (x \vee p \vee \rightharpoonup q)$$

and for a clause of two literals $(x \vee y)$ we create

$$(x \vee y \vee \rightharpoonup p) \wedge (x \vee y \vee p)$$

where p and q continue being the same variables. Therefore, for any satisfying truth assignment T of ϕ, then we could find another satisfying truth assignment T' for ϕ' based on T just choosing arbitrary values for these specific variables p and q introduced in the polynomial time reduction. In addition, we can create a new variable p_{c_i} for each clause c_i that contains more than 3 literals as much as it may be necessary. For example, for a clause of four literals $c_i = (x \vee y \vee z \vee v)$ we create

$$(x \vee y \vee \rightharpoonup p_{c_i}) \wedge (z \vee v \vee p_{c_i})$$

and for other of five literals $c_j = (x \vee y \vee z \vee v \vee w)$ we create another new variable p_{c_j}

$$(x \vee y \vee z \vee \rightharpoonup p_{c_j}) \wedge (v \vee w \vee p_{c_j})$$

and so forth recursively. Hence, for any satisfying truth assignment T of ϕ, then we could find another satisfying truth assignment T' for ϕ' based on T just choosing the corresponding and necessary values for these new variables introduced in the polynomial time reduction for each clause with more than 3 literals . In the case of the original clauses in ϕ contain exactly 3 literals, then they will remain in ϕ'. For that reason, we will obtain

$$(\phi, T) \in OSAT \textit{ if and only if } (\phi', T') \in O3SAT$$

and thus $O3SAT \in ONP$–*complete.* □

Definition 2.14. SIMULTANEOUS INCONGRUENCES

INSTANCE: Collection $\{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\}$ of ordered pair of positive integers with $a_i \leq b_i$ for $1 \leq i \leq n$.

QUESTION: Is there an integer x such that for $1 \leq i \leq n$, every congruence $x \equiv a_i(\mod b_i)$ is false?

We denote this problem as *SI*. $SI \in$ *NP–complete* [23].

Let's define the OTHER-version of this language.

Definition 2.15. OTHER–SIMULTANEOUS INCONGRUENCES

INSTANCE: Collection $\{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\}$ of ordered pair of positive integers with $a_i \leq b_i$ for $1 \leq i \leq n$ and a certificate x.

QUESTION: Is there another integer y such that for $1 \leq i \leq n$, every congruence $y \equiv a_i(\mod b_i)$ is false?

We denote this problem as *OSI*.

Theorem 2.16. $OSI \in P$.

Proof. For the collection $\{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\}$ if every congruence $x \equiv a_i(\mod b_i)$ is false, then every congruence $x + b_1 \times b_2 \times \ldots \times b_n \equiv a_i(\mod b_i)$ is false too. The reason is $b_1 \times b_2 \times \ldots \times b_n \equiv 0(\mod b_i)$ for every $1 \leq i \leq n$. However, the solution $x + b_1 \times b_2 \times \ldots \times b_n$ complies with $x + 2^{\log_2 b_1} \times 2^{\log_2 b_2} \times \ldots \times 2^{\log_2 b_n}$ that is equal to $x + 2^{\log_2 b_1 + \log_2 b_2 + \ldots + \log_2 b_n}$. Since $|x| + |b_1| + |b_2| + \ldots + |b_n|$ is polynomially bounded by the original instance (where $|\ldots|$ denotes the bit-length function), then this new solution is also a certificate and therefore, we can accept *OSI* in polynomial time and thus $OSI \in P$, because the multiplication of integers is polynomial [4]. □

3 DISCUSSION

3.1 On Average Case

The average case performance of algorithms has been studied since modern notions of computational efficiency were developed in the middle of the last century [4]. From the beginning the initial work was focused on problems for which worst case polynomial time algorithms were already known [14]. 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 [14].

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 [15]. He defined the average case complexity and completeness while giving an example of a complete problem for $distNP$, the average case analogue of NP [15]. 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 one *NP–complete* in $distP$ and in this way, we think we open a new path in the analysis on the expected complexity field [4].

3.2 On P = NP

SI is demonstrated that belongs to *NP–complete*, because of a transformation from $3SAT$ [23]. In this way, we think it might be feasible to prove the existence of a polynomial time reduction from

$O3SAT$ to OSI. If this can be proved, then we would be demonstrating that $P = ONP$ because of Theorems 2.13 and 2.16. Since $OHC \in NP\text{–}complete$ and $OHC \in ONP$, then there will be some $NP\text{–}complete$ in P and thus $P = NP$ [9].

4 CONCLUSION

No one has been able to find a polynomial time algorithm for any of more than 300 important known $NP\text{–}complete$ problems [7]. Most complexity theorists already assume P is not equal to NP, but no one has found an accepted and valid proof yet [8]. There are several consequences if P is not equal to NP, such as many common problems cannot be solved efficiently [3]. 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 [3]. The consequences, both positive and negative, arise since various $NP\text{–}complete$ problems are fundamental in many fields. This result does not explicitly conclude with an answer of the P versus NP problem. However, this study about some properties of the class $NP\text{–}complete$ might address to think in the possibility of a $P = NP$ answer.

Cryptography, for example, relies on certain problems being difficult. A constructive and efficient solution to an $NP\text{–}complete$ problem such as $3SAT$ will break most existing cryptosystems including: Public-key cryptography [12], symmetric ciphers [17] and one-way functions used in cryptographic hashing [5]. In that case, these would need to be modified or replaced by information-theoretically secure solutions not inherently based on $P\text{–}NP$ equivalence. Nevertheless, sometimes in cryptography a solution for an average case has been the definitely key to break some cryptosystems from the past [13].

There are enormous positive consequences that will follow from rendering tractable many currently mathematically intractable problems. For instance, many problems in operations research are $NP\text{–}complete$, such as some types of integer programming and the traveling salesman problem [9]. Efficient solutions to these problems have enormous implications for logistics [3]. Many other important problems, such as some problems in protein structure prediction, are also $NP\text{–}complete$, so this will spur considerable advances in biology [2].

But such changes may pale in significance compared to the revolution an efficient method for solving $NP\text{–}complete$ 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." [3]. For that reason, some speculative proofs derived from this paper in the future might help to contribute to the field of Computer Science and many other fields in case of this result being correct.

REFERENCES

[1] Sanjeev Arora and Boaz Barak. 2009. *Computational complexity: a modern approach.* Cambridge University Press.

[2] Bonnie Berger and Tom Leighton. 1998. Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete. *Journal of Computational Biology* 5, 1 (1998), 27–40.

[3] Stephen A Cook. 2000. The P versus NP Problem. (April 2000). Available at Millennium Prize Problems web site http://www.claymath.org/sites/default/files/pvsnp.pdf.

[4] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press.

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

[6] Lance Fortnow. 2009. The status of the P versus NP problem. *Commun. ACM* 52, 9 (2009), 78–86.

[7] Michael R Garey and David S Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness* (1 ed.). San Francisco: W. H. Freeman and Company.

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

[9] Oded Goldreich. 2010. *P, NP, and NP-Completeness: The basics of computational complexity.* Cambridge University Press.
[10] Godfrey Harold Hardy and Edward Maitland Wright. 1979. *An introduction to the theory of numbers.* Oxford University Press.
[11] Juris Hartmanis. 1989. Gödel, von Neumann, and the P = NP problem. *Bulletin of the European Association for Theoretical Computer Science* 38 (1989), 101–107.
[12] Satoshi Horie and Osamu Watanabe. 1997. Hard instance generation for SAT. *Algorithms and Computation* (1997), 22–31.
[13] Jonathan Katz and Yehuda Lindell. 2007. Introduction to Modern Cryptography (Chapman 8 Hall/Crc Cryptography and Network Security Series). (2007).
[14] Donald Knuth. 1973. *The Art of Computer Programming.* Vol. 3. Addison-Wesley.
[15] Leonid A Levin. 1986. Average case complete problems. *SIAM J. Comput.* 15, 1 (1986), 285–286.
[16] Kenneth L Manders and Leonard Adleman. 1978. NP-complete decision problems for binary quadratics. *J. Comput. System Sci.* 16, 2 (1978), 168–184.
[17] Fabio Massacci and Laura Marraro. 2000. Logical cryptanalysis as a SAT problem. *Journal of Automated Reasoning* 24, 1 (2000), 165–203.
[18] John Nash. 1955. Letter to the United States National Security Agency. (January 1955). Available at NSA web site https://www.nsa.gov/news-features/declassified-documents/nash-letters/assets/files/nash_letters1.pdf.
[19] Christos H Papadimitriou. 1994. *Computational complexity.* Addison-Wesley.
[20] René Schoof. 1985. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of computation* 44, 170 (1985), 483–494.
[21] Raymond Séroul. 2012. *Programming for mathematicians.* Springer Science & Business Media.
[22] Michael Sipser. 2006. *Introduction to the Theory of Computation.* Vol. 2. Thomson Course Technology Boston.
[23] Larry J Stockmeyer and Albert R Meyer. 1973. Word problems requiring exponential time (Preliminary Report). In *Proceedings of the fifth annual ACM symposium on Theory of computing.* ACM, 1–9.