SEQUENCES OF LANGUAGES WHEN THE LIMIT GOES TO INFINITY

Frank Vega

SEQUENCES OF LANGUAGES WHEN THE LIMIT GOES TO INFINITY

FRANK VEGA

ABSTRACT. This work is specifically about an interesting class of problems called the NP-complete problems, whose status is unknown. No polynomial-time algorithm has yet been discovered for an NP-complete problem. If any single NP-complete problem can be solved in polynomial-time, then every NP problem has a polynomial-time algorithm. We study two new complexity classes which have a close relation to the NP-complete problems. We call these classes infinite-UP and infinite-P. We show two NP-complete problems which are infinite-UP and infinite-P respectively. In this way, we demonstrate some new properties of the NP-complete problems which can help us to understand better the P versus NP problem.

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

2000 *Mathematics Subject Classification.* Primary 68Q15, Secondary 11A07, 11A51.
Key words and phrases. P, NP, UP, NP-complete, Quadratic Congruences, Subset Product.

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]. All efforts to solve the P versus NP problem have failed [10].

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].

We consider two new complexity classes that are called *infinite–UP* and *infinite–P* which have a close relation to the *NP–complete* problems. 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 *infinite–UP*. In addition, we define another problem that we call Simple Subset Product. We show Simple Subset Product is an *NP–complete* problem. Furthermore, we prove Simple Subset Product is also in *infinite–P*.

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*

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

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 $x \in \{0,1\}^*$,

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

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].

A principal *NP–complete* problem is SAT [6]. 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), $\neg$(NOT), $\Rightarrow$(implication), $\Leftrightarrow$(iff);
(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 [6].

Another *NP–complete* language is $3CNF$ satisfiability, or $3SAT$ [2]. We define $3CNF$ satisfiability using the following terms. A literal in a Boolean formula is an occurrence of a variable or its negation [2]. 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 [2]. A Boolean formula is in 3-conjunctive normal form or $3CNF$, if each clause has exactly three distinct literals [2].

For example, the Boolean formula

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

is in $3CNF$. The first of its three clauses is $(x_1 \vee \neg x_1 \vee \neg x_2)$, which contains the three literals x_1, $\neg x_1$, and $\neg x_2$. In $3SAT$, it is asked whether a given Boolean formula ϕ in $3CNF$ is satisfiable.

It can be demonstrated that many problems belong to *NP–complete* using a polynomial time reduction from $3SAT$ [6]. For example, the well-known problem *1–IN–3 3SAT* which is defined as follows: Given a Boolean formula ϕ in $3CNF$, is there a truth assignment such that each clause in ϕ has exactly one true literal?

2. Results

2.1. infinite-UP.

Definition 2.1. *We say that a language L belongs to UP_∞ if there exist an infinite sequence of languages $L_1, L_2, L_3 \ldots$ where for each $i \in \mathbb{N}$ every language L_i is in UP, has an infinite cardinality, the set $L_{i+1} - L_i$ has infinite elements and $L_i \subset L_{i+1}$ such that*

$$\lim_{i \to \infty} L_i = L.$$

We call the complexity class UP_∞ as "infinite–UP".

Definition 2.2. *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 2.3. *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. $QC \in$ NP–complete [4].

Let's define another problem.

Definition 2.4. *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 2.5. *$GQC \in$ 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

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

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 2.6. *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 2.7. *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 2.8. *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 2.9. *$SQC \in UP$.*

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:

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

SQC belongs to UP because the verifier M can run in polynomial time as we proved in Theorem 2.7 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 $M(a,b,c,d,x) =$ "*yes*",

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

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

Definition 2.10. *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 2.11. *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 n^{th} 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]. □

Theorem 2.12. $GQC \in UP_\infty$.

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]. Certainly, $CQC_i \subset CQC_{i+1}$ and the sets $CQC_{i+1} - CQC_i$ have infinite elements. Moreover, we can assure that

$$\lim_{i \to \infty} CQC_i = GQC.$$

Hence, we can affirm $GQC \in UP_{\infty}$. □

2.2. infinite-P.

Definition 2.13. *We say that a language L belongs to P_{∞} if there exist an infinite sequence of languages $L_1, L_2, L_3 \dots$ where for each $i \in \mathbb{N}$ every language L_i is in P, has an infinite cardinality, the set $L_{i+1} - L_i$ has infinite elements and $L_i \subset L_{i+1}$ such that*

$$\lim_{i \to \infty} L_i = L.$$

We call the complexity class P_{∞} as "infinite–P".

Definition 2.14. *SIMPLE SUBSET PRODUCT*

INSTANCE: A list of numbers L and a positive integer k with its prime factorization, such that the prime factorization of k does not contain any prime power with exponent greater than 1.

QUESTION: Is there a subset of numbers from L whose product is k?

We denote this problem as SSP.

Theorem 2.15. *$SSP \in$ NP–complete.*

Proof. Since SSP is only a restriction of problem *Subset Product*, then $SSP \in NP$ [4]. Certainly, we can check whether the product of a subset of numbers from L is indeed k in polynomial time. We present a polynomial time reduction from *1–IN–3 3SAT* to SSP. For a $3CNF$ formula ϕ with m clauses and n variables, we consider the sets $S_{1,t}, S_{1,f} \dots S_{n,t}, S_{n,f} \subseteq \{1, \dots, m\}$ such that $S_{i,t}$ (resp., $S_{i,f}$) is the set of the indices of the clauses (of ϕ) that are satisfied by setting the i^{th} variable to true (resp., false). That is, if the i^{th} variable appears un-negated in the j^{th} clause then $j \in S_{i,t}$, whereas if the i^{th} variable appears negated in the j^{th} clause then $j \in S_{i,f}$. Indeed, $S_{i,t} \cup S_{i,f}$ equals the set of clauses containing an occurrence of the i^{th} variable, and the union of all these $2 \times n$ sets equals $\{1, \dots, m\}$. Besides, we augment the universe with n additional elements and add the i^{th} such element to both $S_{i,t}$ and $S_{i,f}$. Thus, the reduction proceeds as follows:

(1) On input a $3CNF$ formula ϕ (with n variables and m clauses), the reduction computes the sets $S_{1,t}, S_{1,f} \dots S_{n,t}, S_{n,f}$ such that $S_{i,t}$ (resp., $S_{i,f}$) is the set of the indices of the clauses in which the i^{th} variable appears un-negated (resp., negated).
(2) The reduction creates the sets $S_1 \dots S_{2 \times n}$, where $i = 1 \dots n$ it holds that $S_{2 \times i - 1} = S_{i,t} \cup \{m + i\}$ and $S_{2 \times i} = S_{i,f} \cup \{m + i\}$.
(3) Establish a bijective mapping between the numbers $\{1, \dots, m + n\}$ and the first $m + n$ prime numbers. Replace the members inside of $S_1 \dots S_{2 \times n}$ with the mapped primes.
(4) For each set in $S_1 \dots S_{2 \times n}$ multiply its members together, the resulting list of products is L for the SSP instance. Because prime numbers are used for the mapping in the previous step, the products are guaranteed to be equivalent iff the sets are equivalent by the unique factorization theorem [9].
(5) Multiply the first $m + n$ primes $p_1, \dots, p_{m+n}$ together, the resulting product is the value k for the SSP instance.

Note that (L, k) is a yes-instance of SSP iff ϕ is a yes-instance of *1–IN–3 3SAT*. Assume, on the one hand, that ϕ has the certificate $\tau_1, \dots, \tau_n$. Then, for every

$j \in \{1, \ldots, m\}$ there exists an $i \in \{1, \ldots, n\}$ such that setting the i^{th} variable to τ_i, satisfies the j^{th} clause. Since, k does not contain prime powers with exponent greater than 1 then we guaranteed that exactly one literal is true per each clause.

The transformation steps involves operations that are polynomial to the size of the input ϕ. The first $m + n$ primes can be generated in time $O(m + n)$ using the sieve of Eratosthenes and are guaranteed to fit into $O((m+n)^2 \times \ln(m+n))$ space by the prime number theorem [9]. □

Definition 2.16. *COMPLEX SUBSET PRODUCT ON I-TH PRIME*

INSTANCE: A list of numbers L and a positive integer k with its prime factorization, such that the prime factorization of k does not contain any prime power with exponent greater than 1 and no prime factor is greater than the i^{th} prime number.

QUESTION: Is there a subset of numbers from L whose product is k?

We denote this problem as CSP_i.

Theorem 2.17. *For every positive integer i we have that $CSP_i \in P$.*

Proof. $CSP_1 \in P$, since the target k will necessarily be equal to 2. Indeed, we would only need to check whether the list L contains the number 2. Assume for some natural number i, we have $CSP_i \in P$. Let's prove $CSP_{i+1} \in P$.

Suppose we have an instance (L, k) of CSP_{i+1}. If the target k does not contain the $(i+1)^{th}$ prime number, then $(L, k) \in CSP_i$ iff $(L, k) \in CSP_{i+1}$. If the target k is equal to the $(i+1)^{th}$ prime number, then we replace the occurrence of the number p_{i+1} by the previous prime p_i inside the list L and target k. The resulting instance (L', k') complies with $(L', k') \in CSP_i$ iff $(L, k) \in CSP_{i+1}$.

Now, suppose the target k contains the prime factor p_{i+1} and another p_j where obviously $p_j < p_{i+1}$. Thus we multiply each pair of numbers inside of L such that those number contains either p_{i+1} or p_j but not both. Next, we add these new integers within the list L. After that, we remove those previous numbers that contains only the prime factor p_{i+1} or p_j but not both. Finally, we replace the product $p_{i+1} \times p_j$ by the prime p_j in the factorization of the elements in the resulting list. We do exactly the same with the target k which as well contains the product $p_{i+1} \times p_j$ as a factor number of its factorization. The resulting instance (L', k') complies with $(L', k') \in CSP_i$ iff $(L, k) \in CSP_{i+1}$. Certainly, if $(L, k) \in CSP_{i+1}$ then every possible certificate of (L, k) contains a pair of numbers which has the corresponding prime factor p_{i+1} or p_j or just one single number that contains both. Besides, in every possible reduction the instance (L', k') is polynomially bounded by (L, k).

In this way, we can reduce in polynomial time CSP_{i+1} to CSP_i. Since P is closed under reductions and $CSP_i \in P$, it follows that $CSP_{i+1} \in P$. Hence, for every $i \in \mathbb{N}$ we have proved $CSP_i \in P$ by mathematical induction [9]. □

Theorem 2.18. $SSP \in P_\infty$.

Proof. For every positive integer i the set CSP_i contains infinite elements. Indeed, the list L could vary into an infinite ways even though the target could be the same. Certainly, $CSP_i \subset CSP_{i+1}$ and the sets $CSP_{i+1} - CSP_i$ have infinite elements. Since there are infinite prime numbers, we can assure that

$$\lim_{i \to \infty} CSP_i = SSP.$$

Hence, we can affirm $SSP \in P_{\infty}$. □

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