P vs NP

Frank Vega

P VS NP

FRANK VEGA
La Portada, Cotorro
Havana, Cuba
vega.frank@gmail.com

P versus NP is one of the most important and unsolved problems in computer science. This consists in knowing the answer of the following question: Is P equal to NP? This incognita was first mentioned in a letter written by Kurt Gï¿½del to John von Neumann in 1956. However, the precise statement of the P versus NP problem was introduced in 1971 by Stephen Cook in a seminal paper. Under the assumption of P = NP, we show that P = EXP is also hold. Since P is not equal to EXP, we prove that P is not equal to NP by the Reductio ad absurdum rule.

Keywords: P; NP; EXP; NP-complete; SUBSET-PRODUCT; SUBSET-SUM.

1. Introduction

P versus NP is a major unsolved problem in computer science. 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].

The argument made by Alan Turing in the twentieth century states that for any algorithm we can create an equivalent Turing machine [9]. There are some definitions related to this model such as the deterministic Turing machine. A deterministic Turing machine has only one next action for each step defined in its program or transition function [7].

Another huge advance in the last century was 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 computational complexity theory, the class P contains those languages that can be decided in polynomial-time by a deterministic Turing machine [8]. A language L is in class NP if there is a polynomial-time decidable and polynomially balanced relation R_L such that for all strings x: There is a string y in $R_L(x, y)$ if and only if $x \in L$ [7]. The string y is known as the certificate.

The biggest open question in theoretical computer science concerns the relationship between these two classes:

Is P equal to NP?

In a 2002 poll of 100 researchers, 61 believed the answer to be no, 9 believed the answer is 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].

We shall define an interesting problem called MAS. We show MAS is a succinct version of the known problem *SUBSET–PRODUCT*. When we accept or reject the succinct instances of MAS, then we are accepting or rejecting the equivalent large instances of *SUBSET–PRODUCT*. If we assume that $P = NP$, then the problems MAS and *SUBSET–PRODUCT* could be in *P–complete*. However, this would imply that MAS is also in *EXP–complete*, because MAS is a succinct version of *SUBSET–PRODUCT* and *SUBSET–PRODUCT* would be in *P–complete*. Indeed, in Papadimitriou's book is proved the following statement: "$NEXP$ and EXP are nothing else but P and NP on exponentially more succinct input" [7]. Nevertheless, MAS cannot be in *EXP–complete* and *P–complete* at the same time, because this will be a contradiction. Therefore, we can claim that $P \neq NP$ as a result of applying the Reductio ad absurdum rule.

2. The class NP-complete

We say that a language L_1 is polynomial-time reducible to a language L_2, written $L_1 \leq_p L_2$, if there exists a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

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

A fundamental 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 *NP–hard* [2]. Moreover, if $L \in NP$, then $L \in$ *NP–complete* [2]. No polynomial-time algorithm has yet been discovered for any *NP–complete* problem [3].

A principal *NP–complete* problem is *SUBSET–SUM* [2]. In this problem we are given a finite set $S \subset \mathbf{N}$ and a target $t \in \mathbf{N}$. We ask whether there is a subset $S' \subseteq S$ whose elements sum to t. We may define this problem as a language:

$$SUBSET\text{–}SUM = \{\langle S; t\rangle : \exists S' \subseteq S \text{ such that } t = \sum_{k \in S'} k\}.$$

Another important *NP–complete* problem is *SUBSET–PRODUCT* [4]. An instance of this problem is a finite list L of natural numbers not necessarily distinct and a target $t \in \mathbf{N}$. We ask whether there is a sublist $L' \subseteq L$ such that the product

of the elements in L' is exactly t. This problem is defined as a language in the following way:

$$SUBSET\text{–}PRODUCT = \{\langle L; t\rangle : \exists L' \subseteq L \text{ such that } t = \prod_{k \in L'} k\}.$$

3. The problem MULTIPLE-ARRAYS-SUM

We will define a new problem as follows:

Definition 1. ***MULTIPLE-ARRAYS-SUM (MAS)***

INSTANCE*: A dictionary D and two natural numbers m and n, where D will not contain a key $\langle i, j\rangle$ when $i > m$ or $j > n$ and if D contains the key $\langle i, j\rangle$ for some $1 \leq i \leq m$ and $1 \leq j \leq n$, then $D[\langle i, j\rangle] \in \mathbf{Z}^+$. Besides one array B where $length(B) = m$ and $B[i] \in \mathbf{Z}^+$ for each $1 \leq i \leq m$.*

QUESTION*: Is there a subset $T \subseteq \{1, 2, 3, \ldots, n-1, n\}$ such that $B[1] = \sum_{j \in T} H(D, 1, j), B[2] = \sum_{j \in T} H(D, 2, j), B[3] = \sum_{j \in T} H(D, 3, j), \ldots, B[m] = \sum_{j \in T} H(D, m, j)$, where the function $H(D, i, j)$ returns $D[\langle i, j\rangle]$ when D contains the key $\langle i, j\rangle$ otherwise returns 0?*

Theorem 2. *$MAS \in NP$.*

Proof. Given an instance $\langle D; m; n; B\rangle$ of MAS, we could verify whether a subset $T \subseteq \{1, 2, 3, \ldots, n-1, n\}$ is a certificate for $\langle D; m; n; B\rangle$ in polynomial-time. We verify whether each element of T is an integer between 1 and n. Then, we check whether $B[1] = \sum_{j \in T} H(D, 1, j), B[2] = \sum_{j \in T} H(D, 2, j), B[3] = \sum_{j \in T} H(D, 3, j), \ldots, B[m] = \sum_{j \in T} H(D, m, j)$. In addition, the function H can be executed in polynomial time [2]. Since this iteration over the elements in the set T and searching in the dictionary D just comparing or summing their values can be done in polynomial time, we obtain that $MAS \in NP$. □

We say that a language L_1 is logarithmic-space reducible to a language L_2, written $L_1 \leq_L L_2$, if there exists a logarithmic-space computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

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

The logarithmic-space reduction is frequently used for P and below [7].

Theorem 3. *$SUBSET\text{–}SUM \leq_L MAS$.*

Proof. Given an arbitrary instance $\langle S; t\rangle$ of *SUBSET–SUM*, we will do the following actions:

(1) We define a one-to-one function κ_s which maps to a single integer between 1 and $|S|$ each element of S. This means $\kappa_s(x) \neq \kappa_s(y)$ if and only if $x \neq y$ for $x, y \in S$. We may define this function as the order from left to right in which the elements of S appear over the input, such that the first element has mapped the integer 1 on κ_s, the second element has mapped the integer 2 on κ_s and so on ... until the last element which has mapped the integer $|S|$.
(2) Next, we create a dictionary D and define the numbers $m = 1$ and $n = |S|$.
(3) After that, we iterate for each $x \in S$ adding the value of x in D as follows: $D[\langle 1, \kappa_s(x)\rangle] = x$.
(4) Finally, we create the array B of length 1 such that $B[1] = t$.

A subset $S' \subseteq S$ whose elements sum to t can be interpreted as a subset $T' \subseteq \{1, 2, 3, \ldots, |S| - 1, |S|\}$ such that $x \in S'$ if and only if $\kappa_s(x) \in T'$. Certainly, if S' is a certificate of $\langle S; t\rangle$, then the instance $\langle D; m; n; B\rangle$ belongs to MAS, because $t = B[1] = \sum_{j \in T'} H(D, 1, j)$. Moreover, if T' is a certificate of $\langle D; m; n; B\rangle$, then the instance $\langle S; t\rangle$ belongs to *SUBSET–SUM*, since there will be $|T'|$ elements in S that sum t. The reason is because the elements of $D[\langle 1, j\rangle]$ iterating through all j between 1 and $|S|$ is exactly the set S and $t = B[1]$. Hence, we obtain the following result:

$$\langle S; t\rangle \in \textit{SUBSET–SUM} \text{ if and only if } \langle D; m; n; B\rangle \in MAS.$$

In addition, the construction of dictionary D and array B can be done in logarithmic-space. Certainly, we can write the dictionary D directly to the output only using one string k that simulates the value of κ_s for each element $x \in S$. In the beginning the value of k is 0 and when we find the first element $x \in S$ from left to right in the input, then we increment k in 1 and write the entry $D[\langle 1, k\rangle] = x$ to the output. After that, we find the second element $y \in S$ from left to right in the input and increment k in 1 again to write the entry $D[\langle 1, k\rangle] = y$ to the output. Consequently, we repeat the same procedure over and over again until we reach the last element of S, and thus, we obtain that $\textit{SUBSET–SUM} \leq_L MAS$. □

Theorem 4. *$MAS \in$ NP–complete.*

Proof. Since every logarithmic-space reduction is also a polynomial-time reduction and $\textit{SUBSET–SUM} \leq_L MAS$, then $MAS \in$ *NP–hard*. Furthermore, $MAS \in NP$, and therefore, $MAS \in$ *NP–complete*. □

4. P versus NP

We will state our principal result:

Theorem 5. *MAS is actually SUBSET–PRODUCT on exponentially more succinct representation.*

Proof. Given an arbitrary instance $\langle L; t\rangle$ of *SUBSET–PRODUCT*, we will do the following actions:

(1) We create a function κ_c which maps to a single integer between 1 and $|L|$ each element of L, such that if we pick two elements $x, y \in L$, then they will have different values on κ_c, even though they might represent the same number.
(2) If the number N is the amount of different prime factors in t, then we create a one-to-one function κ_t which maps to a single integer between 1 and N each prime factor of t. This means $\kappa_t(p) \neq \kappa_t(q)$ if and only if $p \neq q$ for every two primes p and q that divide to t.
(3) Next, we create a dictionary D and define the numbers $m = (N + 1)$ and $n = |L|$.
(4) After that, we iterate for each $x \in L$ filling the values of D as follows: If x can be expressed as $p^w \times r$ where p is a prime factor of t, $w \in \mathbf{N}$ and the natural number r is not divisible by p, then $D[\langle \kappa_t(p), \kappa_c(x)\rangle] = w$ otherwise D will not contain the key $\langle \kappa_t(p), \kappa_c(x)\rangle$.
(5) Then, we iterate for each $x \in L$ continue filling the values of D in the following way: If x has at least one prime factor p such that p does not divide to t, then $D[\langle N + 1, \kappa_c(x)\rangle] = 1$ otherwise D will not contain the key $\langle N + 1, \kappa_c(x)\rangle$.
(6) Finally, we create the array B of length $N+1$ such that if the number t can be expressed as $p^w \times r$ where p is a prime factor, $w \in \mathbf{N}$ and the natural number r is not divisible by p, then we assign $B[\kappa_t(p)] = w$. In addition, we make $B[N + 1] = 0$.

A sublist $L' \subseteq L$ whose elements multiplication is exactly t can be interpreted as a subset $T' \subseteq \{1, 2, 3, \ldots, |L|-1, |L|\}$ such that $x \in L'$ if and only if $\kappa_c(x) \in T'$. Since t is equal to the product of the powers of its different prime factors as $p^{B[\kappa_t(p)]} \times q^{B[\kappa_t(q)]} \times \ldots \times r^{B[\kappa_t(r)]}$, we obtain that $B[\kappa_t(p)] = \sum_{j\in T'} H(D, \kappa_t(p), j), B[\kappa_t(q)] = \sum_{j\in T'} H(D, \kappa_t(q), j), \ldots, B[\kappa_t(r)] = \sum_{j\in T'} H(D, \kappa_t(r), j)$, because the product of powers A^i and A^j under the same base A will imply the addition of their exponents i and j in the result $A^{(i+j)}$. In addition, the keys that contain $(N+1)$ in D guarantee there will not be included any other prime in the multiplication which is not a prime factor of t, since $0 = B[N+1] = \sum_{j\in T'} H(D, N+1, j)$. Indeed, T' will be a certificate for $\langle D; m; n; B\rangle$ in MAS.

We can make the reverse transformation in the other direction too. Certainly, if $\langle D; m; n; B\rangle$ is an arbitrary instance of MAS, then we could translate it into an instance $\langle L; t\rangle$ of *SUBSET–PRODUCT*. For this purpose, we create a one-to-one function κ_r which maps to a single prime number the integers between 1 and m in the following way: $\kappa_r(i) = p$ if and only if p is the *i-th* prime. Then, we iterate for each $j = 1, 2, 3, \ldots, n$ defining the elements of list L as follows: $((\kappa_r(1))^{H(D,1,j)} \times (\kappa_r(2))^{H(D,2,j)} \times (\kappa_r(3))^{H(D,3,j)} \times \ldots \times (\kappa_r(m))^{H(D,m,j)})$. Finally,

we define t as $((\kappa_r(1))^{B[1]} \times (\kappa_r(2))^{B[2]} \times (\kappa_r(3))^{B[3]} \times \ldots \times (\kappa_r(m))^{B[m]})$. Now, if $\langle D; m; n; B\rangle$ is in MAS, then there is a sublist $L' \subseteq L$ whose elements multiplication is exactly t. Indeed, we can affirm that we can always find for every instance $\langle L; t\rangle$ of *SUBSET–PRODUCT* another equivalent instance $\langle D; m; n; B\rangle$ of MAS and viceversa. Consequently, we obtain that

$$\langle L; t\rangle \in \textit{SUBSET–PRODUCT} \text{ if and only if } \langle D; m; n; B\rangle \in MAS$$

where $\langle D; m; n; B\rangle$ always contains the exponents of the prime factors of t over the natural numbers in the instance $\langle L; t\rangle$ just ignoring the case in which the exponent value is 0. But definitely, the size of a number $|(p^u \times q^v \times \ldots \times r^w)|$ is exponentially larger than $|u| + |v| + \ldots + |w|$ when $p, q, \ldots, r$ are different prime numbers and $u, v, \ldots, w \in \mathbf{N}$. In conclusion, we can confirm the problem MAS is just a succinct version of *SUBSET–PRODUCT*. □

Theorem 6. $P \neq NP$.

Proof. We start assuming that $P = NP$. But, if $P = NP$, then MAS and *SUBSET–PRODUCT* would be in *P–complete*, because all currently known *NP–complete* are *NP–complete* under logarithmic-space reduction including our new problem MAS [4]. A succinct version of a problem that is complete for P can be shown not to lie in P, because it will be complete for EXP [7]. Since MAS is a succinct version of *SUBSET–PRODUCT* and *SUBSET–PRODUCT* would be in *P–complete*, then we obtain that MAS should be also in *EXP–complete*.

Since the classes P and EXP are closed under reductions, and MAS is complete for both P and EXP, then we could state that $P = EXP$ [7]. However, as result of Hierarchy Theorem the class P cannot be equal to EXP [7]. To sum up, we obtain a contradiction under the assumption that $P = NP$, and thus, we can claim that $P \neq NP$ as a direct consequence of the Reductio ad absurdum rule. □

5. Conclusions

This proof explains why after decades of studying the NP problems no one has been able to find a polynomial-time algorithm for any of more than 300 important known *NP–complete* problems [4]. Indeed, it shows in a formal way that many currently mathematically problems cannot be solved efficiently, so that the attention of researchers can be focused on partial solutions or solutions to other problems.

Although this demonstration removes the practical computational benefits of a proof that $P = NP$, it would represent a very significant advance in computational complexity theory and provide guidance for future research. In addition, it proves that could be safe most of the existing cryptosystems such as the public-key cryptography [6]. On the other hand, we will not be able to find a formal proof for every theorem which has a proof of a reasonable length by a feasible algorithm.

References

[1] S. A. Cook, The complexity of theorem-proving procedures, *Proceedings of the 3rd IEEE Symp. on the Foundations of Computer Science*, (1971), pp. 151–158.
[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, 2 edn. (MIT Press, 2001).
[3] L. Fortnow, The Status of the P versus NP Problem, *Communications of the ACM* **52**(9) (2009) 78–86.
[4] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1 edn. (San Francisco: W. H. Freeman and Company, 1979).
[5] W. I. Gasarch, The P=?NP poll, *SIGACT News* **33**(2) (2002) 34–47.
[6] O. Goldreich, *P, Np, and Np-Completeness* (Cambridge: Cambridge University Press, 2010).
[7] C. H. Papadimitriou, *Computational complexity* (Addison-Wesley, 1994).
[8] M. Sipser, *Introduction to the Theory of Computation*, 2 edn. (Thomson Course Technology, 2006).
[9] A. M. Turing, On Computable Numbers, with an Application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society* **42** (1936) 230–265.