A Mathematical Conjecture from P versus NP

Frank Vega

May 16, 2020

A Mathematical Conjecture from P versus NP

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

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? It was essentially mentioned in 1955 from a letter written by John Nash to the United States National Security Agency. However, a precise statement of the P versus NP problem was introduced independently by Stephen Cook and Leonid Levin. Since that date, all efforts to find a proof for this problem have failed. 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. Another major complexity class is NP-complete. To attack the P versus NP question the concept of NP-completeness has been very useful. If any single NP-complete problem can be solved in polynomial time, then every NP problem has a polynomial time algorithm. We state the following conjecture for a natural number B greater than 3: The number of divisors of B is lesser than or equal to the quadratic value from the integer part of the logarithm of B in base 2. This conjecture has been checked for large numbers: Specifically, from every integer between 4 and 10 millions. If this conjecture is true, then the NP-complete problem Subset Product is in P and thus, the complexity class P is equal to NP.

2012 ACM Subject Classification Theory of computation → Complexity classes

Keywords and phrases complexity classes, completeness, polynomial time, logarithm, tuple

1 Introduction

The P versus NP problem is a major unsolved problem in computer science [4]. This is considered by many to be the most important open problem in the field [4]. The precise statement of the $P = NP$ problem was introduced in 1971 by Stephen Cook in a seminal paper [4]. 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].

The $P = NP$ question is also singular in the number of approaches that researchers have brought to bear upon it over the years [6]. From the initial question in logic, the focus moved to complexity theory where early work used diagonalization and relativization techniques [6]. It was showed that these methods were perhaps inadequate to resolve P versus NP by demonstrating relativized worlds in which $P = NP$ and others in which $P \neq NP$ [3]. This shifted the focus to methods using circuit complexity and for a while this approach was deemed the one most likely to resolve the question [6]. Once again, a negative result showed that a class of techniques known as "Natural Proofs" that subsumed the above could not separate the classes NP and P, provided one-way functions exist [11]. There has been speculation that resolving the $P = NP$ question might be outside the domain of mathematical techniques [6]. More precisely, the question might be independent of standard axioms of set theory [6]. Some results have showed that some relativized versions of the $P = NP$ question are independent of reasonable formalizations of set theory [9].

It is fully expected that $P \neq NP$ [10]. Indeed, if $P = NP$ then there are stunning practical consequences [10]. For that reason, $P = NP$ is considered as a very unlikely event [10]. Certainly, P versus NP is one of the greatest open problems in science and a correct solution for this incognita will have a great impact not only in computer science, but for many other fields as well [1]. Whether $P = NP$ or not is still a controversial and unsolved problem [1]. We show some results that could help us to prove this outstanding problem.

2 Theory and Methods

2.1 Preliminaries

In 1936, Turing developed his theoretical computational model [12]. The deterministic and nondeterministic Turing machines have become in two of the most important definitions related to this theoretical model for computation [12]. A deterministic Turing machine has only one next action for each step defined in its program or transition function [12]. 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 [12].

Let Σ be a finite alphabet with at least two elements, and let Σ^* be the set of finite strings over Σ [2]. A Turing machine M has an associated input alphabet Σ [2]. For each string w in Σ^* there is a computation associated with M on input w [2]. We say that M accepts w if this computation terminates in the accepting state, that is $M(w) = \text{“}yes\text{”}$ [2]. Note that M fails to accept w either if this computation ends in the rejecting state, that is $M(w) = \text{“}no\text{”}$, or if the computation fails to terminate, or the computation ends in the halting state with some output, that is $M(w) = y$ (when M outputs the string y on the input w) [2].

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 [5]. A complexity class is a set of problems, which are represented as a language, grouped by measures such as the running time, memory, etc [5]. The language accepted by a Turing machine M, denoted $L(M)$, has an associated alphabet Σ and is defined by:

$$L(M) = \{w \in \Sigma^* : M(w) = \text{“}yes\text{”}\}.$$

Moreover, $L(M)$ is decided by M, when $w \notin L(M)$ if and only if $M(w) = \text{“}no\text{”}$ [5]. We denote by $t_M(w)$ the number of steps in the computation of M on input w [2]. 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 [2]. We say that M runs in polynomial time if there is a constant k such that for all n, $T_M(n) \leq n^k + k$ [2]. In other words, this means the language $L(M)$ can be decided by the Turing machine M in polynomial time. Therefore, P is the complexity class of languages that can be decided by deterministic Turing machines in polynomial time [5]. A verifier for a language L_1 is a deterministic Turing machine M, where:

$$L_1 = \{w : M(w, c) = \text{“}yes\text{”} \textit{ for some string } c\}.$$

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 [2]. A verifier uses additional information, represented by the symbol c, to verify that a string w is a member of L_1. This information is called certificate. NP is the complexity class of languages defined by polynomial time verifiers [10].

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

$x \in L_1$ *if and only if* $f(x) \in L_2$.

An important complexity class is *NP–complete* [7]. A language $L_1 \subseteq \{0,1\}^*$ is *NP–complete* if:

- $L_1 \in NP$, and
- $L' \leq_p L_1$ for every $L' \in NP$.

If L_1 is a language such that $L' \leq_p L_1$ for some $L' \in$ *NP–complete*, then L_1 is *NP–hard* [5]. Moreover, if $L_1 \in NP$, then $L_1 \in$ *NP–complete* [5].

2.2 Definitions on Tuples

▶ **Definition 1.** *We consider a tuple* $(a_1, a_2, \ldots, a_m)$ *as an m-tuple.*

▶ **Definition 2.** *We consider the addiction of two m-tuples* $(a_1, a_2, \ldots, a_m)$ *and* $(b_1, b_2, \ldots, b_m)$ *as the m-tuple* $(a_1 + b_1, a_2 + b_2, \ldots, a_m + b_m)$.

▶ **Definition 3.** *We consider the subtraction of two m-tuples* $(a_1, a_2, \ldots, a_m)$ *and* $(b_1, b_2, \ldots, b_m)$ *as the m-tuple* $(a_1 - b_1, a_2 - b_2, \ldots, a_m - b_m)$.

▶ **Definition 4.** *We consider an m-tuple* $(a_1, a_2, \ldots, a_m)$ *is equal to an m-tuple* $(b_1, b_2, \ldots, b_m)$ *if and only if for every integer* $1 \leq i \leq m$ *we have that* $a_i = b_i$.

▶ **Definition 5.** *For a positive integer* k*, we consider* k_m *as the m-tuple* $(\underbrace{k, k, \ldots, k}_{m})$*. Besides, an m-tuple* $(a_1, a_2, \ldots, a_m)$ *is lesser than* 0_m*, when there is an integer* $1 \leq i \leq m$ *such that* $a_i < 0$.

▶ **Definition 6.** *For some natural number* $B > 3$ *with the prime factorization* $p_1^{a_1} \times p_2^{a_2} \times \ldots \times p_m^{a_m}$ *such that* $p_1 < p_2 < \ldots < p_m$*, then we consider the value of* $h(B)$ *as the m-tuple* $(a_1, a_2, \ldots, a_m)$.

▶ **Definition 7.** *Consider two natural numbers* $B > 3$ *and* $C \geq 1$ *when* C *divides* B *and the prime factorization of* B *is* $p_1^{a_1} \times p_2^{a_2} \times \ldots \times p_m^{a_m}$ *such that* $p_1 < p_2 < \ldots < p_m$*, then we consider the value of* $h_B(C)$ *as the m-tuple* $(a'_1, a'_2, \ldots, a'_m)$ *where* a'_1 *is the exponent of the power* $p_1^{a'_1}$ *in the prime factorization of* C *from the prime* p_1 *and so forth until* m *(the value of* a'_i *could be* 0 *when the prime* p_i *does not divide* C*).*

3 Results

We show a previous known *NP–complete* problem:

▶ **Definition 8.** ***Subset Product***

INSTANCE: Finite set X*, a size* $s(x) \in \mathbb{Z}^+$ *for each* $x \in X$*, and a positive integer* B.

QUESTION: Is there a subset $X' \subseteq X$ *such that the product of the sizes of the elements in* X' *is* B*?*

REMARKS: We denote this problem as SP [10]. $SP \in$ *NP–complete [7]. This problem remains in NP–complete even if we know the prime factorization of* B *[7].*

▷ Conjecture 9. For some natural number $B > 3$ with the prime factorization $p_1^{a_1} \times p_2^{a_2} \times \ldots \times p_m^{a_m}$, then we could always obtain that $(a_1 + 1) \times (a_2 + 1) \times \ldots \times (a_m + 1) \leq (\lfloor \log_2 B \rfloor)^2$, which means that the number of divisors of B is lesser than or equal to $(\lfloor \log_2 B \rfloor)^2$ [13].

▶ **Theorem 10.** *If the Conjecture 9 is true, then $SP \in P$.*

Proof. Suppose the set X is

$$x_1, x_2, \ldots, x_N$$

and we wish to determine if there is a nonempty subset $X' \subseteq X$ such that the product of the sizes of the elements in X' is B. We assume that we have the prime factorization of B. We ignore when $B \leq 3$, since these cases are trivial. We assume also that each size $s(x_i)$ divides B otherwise we just remove the element x_i from our set X. We consider the sequence of tuples

$$h_B(s(x_1)), h_B(s(x_2)), \ldots, h_B(s(x_N))$$

where $c_i = s(x_i)$ is the size of the element x_i and the function $h_B(c_i)$ returns an m-tuple for some m using the Definition 7. We can calculate the tuple $h_B(s(x_i))$ for every element $x_i \in X$ just in $O(N \times (\lfloor \log_2 B \rfloor)^3)$, since we have the prime factorization of B.

Now, define the Boolean-valued function $Q(i, y)$ to be the value (true or false) of "there is a nonempty subset of $s(x_1), \ldots, s(x_i)$ which products to y" which is equivalent to the Boolean-valued function $Q(i, h_B(y))$ "there is a nonempty subset of m-tuples $h_B(s(x_1)), \ldots, h_B(s(x_i))$ which sums to $h_B(y)$", because the product of two prime powers p^r and p^t from a same prime p is equal to p^{r+t}, where we sum the exponents r and t of the prime powers. Thus, the solution to the problem "Given a nonempty subset $X' \subseteq X$ such that the product of the sizes of the elements in X' is B?" is the value of $Q(N, h(B))$ using the Definition 6.

Clearly, $Q(i, h_B(y)) =$ false, if $h_B(y) < 0_m$ or $y > B$ using the Definition 5. So these values do not need to be stored or computed. Create an array to hold the values $Q(i, h_B(y))$ for $1 \leq i \leq N$, $0_m \leq h_B(y)$ and $y \leq B$ such that y divides B. The array can now be filled in using a simple recursion. Initially, for $0_m \leq h_B(y)$ and $y \leq B$ such that y divides B, set

$$Q(1, h_B(y)) = (h_B(s(x_1)) == h_B(y))$$

where $==$ is a Boolean function that returns true if $h_B(s(x_1))$ is equal to $h_B(y)$ using the Definition 4, false otherwise. Then, for $i = 2, \ldots, N$, set for $0_m \leq h_B(y)$ and $y \leq B$ such that y divides B

$$Q(i, h_B(y)) = Q(i-1, h_B(y)) \vee (h_B(s(x_i)) == h_B(y)) \vee Q(i-1, h_B(y) - h_B(s(x_i)))$$

where the substraction of tuples is stated using the Definition 3 and $\vee$ is the OR Boolean function. For each assignment, the values of Q on the right side are already known, either because they were stored in the table for the previous value of i or because $Q(i-1, h_B(y) - h_B(s(x_i))) =$ false if $h_B(y) - h_B(s(x_i)) < 0_m$. Therefore, the total number of arithmetic operations is $O(N \times q \times (\lfloor \log_2 B \rfloor))$, where q is equal to the number of the valid m-tuples between 0_m and $h(B)$ (that is, the amount of different integers $1 \leq y \leq B$ such that y divides B) and $(\lfloor \log_2 B \rfloor) \geq m$ is greater than or equal to the number of indexes in the m-tuples that we need to compare in each iteration. Certainly, the amount of the valid m-tuples between 0_m and $h(B)$ is equal to $q = (a_1 + 1) \times (a_2 + 1) \times \ldots \times (a_m + 1)$ when the prime factorization of $B > 3$ is $p_1^{a_1} \times p_2^{a_2} \times \ldots \times p_m^{a_m}$, where this is actually the number of divisors of B [13]. In this way, if this Conjecture 9 is true, then the solution has runtime of $O(N \times (\lfloor \log_2 B \rfloor)^3)$ and thus, the problem SP would be in P, because the runtime is polynomial according to the bit-length of the input. ◀

▶ **Lemma 11.** *If the Conjecture 9 is true, then $P = NP$.*

Proof. This is a direct consequence of Theorem 10, because when any single *NP–complete* problem can be solved in polynomial time, then every NP problem has a polynomial time algorithm [5]. ◀

References

1 Scott Aaronson. P $\stackrel{?}{=}$ NP. *Electronic Colloquium on Computational Complexity, Report No. 4*, 2017.

2 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

3 Theodore Baker, John Gill, and Robert Solovay. Relativizations of the $\mathcal{P} = ?\mathcal{NP}$ Question. *SIAM Journal on computing*, 4(4):431–442, 1975. `doi:10.1137/0204037`.

4 Stephen A. Cook. The P versus NP Problem, April 2000. In Clay Mathematics Institute at `http://www.claymath.org/sites/default/files/pvsnp.pdf`. Retrieved 26 April 2020.

5 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

6 Vinay Deolalikar. P $\neq$ NP, 2010. In Woeginger Home Page at `https://www.win.tue.nl/~gwoegi/P-versus-NP/Deolalikar.pdf`. Retrieved 26 April 2020.

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

8 William I. Gasarch. Guest column: The second P $\stackrel{?}{=}$ NP poll. *ACM SIGACT News*, 43(2):53–77, 2012. `doi:10.1145/2261417.2261434`.

9 Juris Hartmanis and John E. Hopcroft. Independence Results in Computer Science. *SIGACT News*, 8(4):13–24, October 1976. `doi:10.1145/1008335.1008336`.

10 Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

11 Alexander A. Razborov and Steven Rudich. Natural Proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, August 1997. `doi:10.1006/jcss.1997.1494`.

12 Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

13 David G. Wells. *Prime Numbers, The Most Mysterious Figures in Math*. John Wiley & Sons, Inc., 2005.