Is P equal to NP?

Frank Vega

Is P equal to NP?

Frank Vega

Abstract

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 Gödel 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, NEXP, coNP
2000 MSC: 68-XX, 68Qxx, 68Q15

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 [2]. It is one of the seven Millennium Prize Problems selected by the Clay Mathematics Institute to carry a US$1,000,000 prize for the first correct solution [2].

In 1936, Turing developed his theoretical computational model [3]. The deterministic and nondeterministic Turing machine have become in some 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 [4]. 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 [4].

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 [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].

In 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 two classes:

Is P equal to NP?

Email address: `vega.frank@gmail.com` (Frank Vega)

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 [7].

In addition, we have the class EXP contains those languages that can be decided in exponential-time by a deterministic Turing machine [6]. The class $NEXP$ is the set of all languages that can be decided in exponential-time by a nondeterministic Turing machine [6]. EXP and $NEXP$ are nothing else but P and NP on exponentially more succinct input [4]. It is known the succinct version of the problem HAMILTON PATH, that is called SUCCINCT HAMILTON PATH, is in *NEXP–complete* [4]. We shall prove if we assume that $P = NP$, then the language SUCCINCT HAMILTON PATH would be in P too. However, this would imply $P = EXP$ [4]. But, this is a false result [4]. In this way, we shall claim that $P \neq NP$ as a consequence of applying the Reductio ad absurdum rule.

2. Results

A graph G is a pair (V, E), where V is a finite set and E is a binary relation on V [5]. The set V is called the vertex set of G, and its elements are called vertices or nodes [5]. The set E is called the edge set of G, and its elements are called edges [5]. If (u, v) is an edge in a graph $G = (V, E)$, we say that vertex v is adjacent to vertex u [5]. A path of length k from a vertex u to a vertex u' in a graph $G = (V, E)$ is a sequence of vertices $\langle v_0, v_1, v_2, \ldots, v_k \rangle$ such that $u = v_0$, $u' = v_k$, and $(v_{i-1}, v_i) \in E$ for $i = 1, 2, \ldots, k$ [5]. One of the most basic problems on graphs is this: Given a graph, is there a path that visits each node exactly once? We call this problem as HAMILTON PATH [4]. HAMILTON PATH is in *NP–complete* [4].

A succinct representation of a graph with n nodes, where $n = 2^b$ is a power of two, is a Boolean circuit C with $2 \times b$ input gates [4]. The graph represented by C, denoted G_C, is defined as follows: The nodes of G_C are $\{1, 2, \ldots, n\}$. And (i, j) is an edge of G_C if and only if C accepts the binary representations of the *b-bits* integers i, j as inputs [4]. The problem SUCCINCT HAMILTON PATH is now this: Given the succinct representation C of a graph G_C with n nodes, does G_C have a Hamilton path? The problem SUCCINCT HAMILTON PATH is in *NEXP–complete* [4].

Theorem 2.1. *If $P = NP$, then SUCCINCT HAMILTON PATH would be in P.*

Proof. Let's take an arbitrary succinct representation C of a graph G_C with n nodes, where $n = 2^b$ is a power of two and C will be a Boolean circuit of $2 \times b$ input gates. The circuit C computes a Boolean function $f_C : \{true, false\}^{2 \times b} \to \{true, false\}$ [4]. Now, if C is a "yes" instance of SUCCINCT HAMILTON PATH, then there will be a linear order Q on the nodes of G_C, that is, a binary relationship isomorphic to $<$ on the nodes of G_C, such that consecutive nodes are connected in G_C [4].

This linear order Q must require several things:

1. All distinct nodes of G_C are comparable by Q,
2. next, Q must be transitive but not reflexive,
3. and finally, any two consecutive nodes in Q must be adjacent in G_C.

Any binary relationship Q that has these properties must be a linear order, any two consecutive elements of which are adjacent in G_C—that is, it must be a Hamilton path [4].

Let R be a binary relation on strings. R is called polynomially decidable if there is a deterministic Turing machine deciding the language $\{x; y : (x, y) \in R\}$ in polynomial time [4]. We say that R is polynomially balanced if $(x, y) \in R$ implies $|y| < |x|^k$ for some $k \geq 1$ [4].

The linear order Q can be represented as a graph G_Q. In this way, the succinct representation C_Q of the graph G_Q will represent the linear order Q too. We can define a polynomially balanced relation R_Q, where for all succinct representation C of a graph: There is another Boolean circuit C_Q that will represent a linear order Q on the nodes of G_C such that $(C, C_Q) \in R_Q$ if and only if $C \in$ SUCCINCT HAMILTON PATH [4]. Indeed, the graphs G_C and G_Q will comply with $|G_Q| < |G_C|^3$ when $(C, C_Q) \in R_Q$, since both graphs would have the same number of nodes and G_C would contain a Hamilton path. Certainly, if the graph G_C of n nodes contains a Hamilton path, then it would have at least $(n - 1)$ edges. But, if G_C is a pair (V_{G_C}, E_{G_C}), G_Q is (V_{G_Q}, E_{G_Q}) and $(C, C_Q) \in R_Q$, where V_{G_C} and V_{G_Q} are vertex sets and E_{G_Q} and E_{G_C} are edge sets, then $|V_{G_C}| = |V_{G_Q}|$ and $|E_{G_Q}| < |E_{G_C}|^3$ when $|E_{G_C}| > 1$, because the maximum number of edges in a graph of n nodes is lesser than $n \times (n - 1)$ [5]. Consequently, we obtain the same property for their succinct representations, that is, C_Q should be polynomially bounded by C. Indeed, for a sufficiently large n, the Boolean circuits C and C_Q will be exponentially more succinct than G_C and G_Q respectively [4].

A language L is in class NP if there is a polynomially decidable and polynomially balanced relation R such that $L = \{x : (x, y) \in R \text{ for some } y\}$ [4]. We shall show the binary relation R_Q would be polynomially decidable if $P = NP$. In this way, we show that SUCCINCT HAMILTON PATH would be in NP under this assumption. Moreover, since P would be equal to NP, we obtain that SUCCINCT HAMILTON PATH would be in P too.

Given the chosen arbitrary Boolean circuit C, we will show we could decide in polynomial-time whether $(C, C_Q) \in R_Q$ for some circuit C_Q. This circuit C_Q must compute a Boolean function $f_Q : \{true, false\}^{2\times b} \to \{true, false\}$ [4].

First, let's define two simple languages:

Definition 2.2. *Problem* ***BETWEEN****:*
INSTANCE*: Three positive integers x, i, and j.*
QUESTION*: Is x between i and j?*

Definition 2.3. *Problem* ***EQUAL****:*
INSTANCE*: Two positive integers x and y.*
QUESTION*: Is x equal to y?*

It is easy to see these problems are in P. Hence, they will have uniformly polynomial circuits [4]. Certainly, a language L has uniformly polynomial circuits if and only if $L \in P$ [4]. Consequently, we can obtain in logarithmic-space a Boolean circuit C_{bt} of polynomial size in relation to b, where C_{bt} will have $(2 \times b) + 1$ input gates, such that for some positive integer x represented on a binary string of length b, x is between 1 and n if and only if the output of C_{bt} is true, when the ith input variable is true if $x_i = 1$, and false otherwise [4]. The other input gates in C_{bt} do not require a truth assignment, because they are not variable gates. Indeed, in these input gates we use the same conversion for the binary sequence of 1 and n into a Boolean sequence of final length $(b + 1)$: That is *true* in a bit 1, and false otherwise. The circuit C_{bt} computes a Boolean function $f_{bt} : \{true, false\}^b \to \{true, false\}$ [4]. Similarly, we could obtain a Boolean circuit C_{eq} in logarithmic-space, where C_{eq} has only $2 \times b$ input gates and its size is polynomial in relation to b, such that for two positive integers x and y represented on binary strings of length b, the output of C_{eq} is true if and only if x is equal to y. The circuit C_{eq} computes a Boolean function $f_{eq} : \{true, false\}^{2\times b} \to \{true, false\}$ [4].

If NP is the class of problems that have succinct certificates, then the complexity class $coNP$ contains those problems that have succinct disqualifications [4]. That is, a "no" instance of a problem in *coNP* possesses a short proof of its being a "no" instance [4]. An interesting language is TAUTOLOGY which is defined as follows: Given a Boolean formula ϕ, is there not any truth assignment that makes ϕ false? TAUTOLOGY is in *coNP–complete*, because its complement is *NP–complete* [8]. A Boolean formula in TAUTOLOGY is frequently called a tautology [4].

Using the previous functions f_Q, f_{bt} and f_{eq}, we can define another Boolean functions as follows:

$$\psi(X, Y) = \delta(X, Y) \Rightarrow (f_Q(X, Y) \vee f_Q(Y, X) \vee f_{eq}(X, Y))$$

$$\varphi(X, Y, Z) = \gamma(X, Y, Z) \Rightarrow ((\neg\ f_Q(X, X)) \wedge ((f_Q(X, Y) \wedge f_Q(Y, Z)) \Rightarrow f_Q(X, Z)))$$

where

$$\delta(X, Y) = f_{bt}(X) \wedge f_{bt}(Y)$$

and

$$\gamma(X, Y, Z) = f_{bt}(X) \wedge f_{bt}(Y) \wedge f_{bt}(Z).$$

We let X be a set of Boolean variables $\{x_i \in X : i \in N \text{ and } 1 \leq i \leq b\}$ that represent a *b-bits* integer m, where the ith variable x_i is true if $m_i = 1$, and false otherwise. We will define the set of Boolean variables Y and Z in the same way. The functions δ and γ guarantee the *b-bits* integers represented by X, Y and Z will always evaluate ψ and φ to true when they are not between 1 and n.

In analogy with Boolean circuits compute Boolean functions, the Boolean functions could be expressed by Boolean expressions [4]. All distinct nodes of G_C are comparable by a binary relationship Q on nodes of G_C if and only if the Boolean expression that expresses the function ψ is a tautology. Moreover, a binary relationship Q on nodes of G_C is transitive but not reflexive if and only if the Boolean expression that expresses the function φ is a tautology. But, if $P = NP$, then $P = NP = coNP$ [4]. In this way, TAUTOLOGY would be in P, and thus, it has been proved that we can check in polynomial-time the first and second property of a linear order Q on nodes of G_C.

For the verification of the third property we need to define a new language:

Definition 2.4. *Problem **EVALUATION**:*

***INSTANCE**: Two Boolean formulas ϕ_1 and ϕ_2. The formula ϕ_2 contains all the variables of ϕ_1, but ϕ_2 might have some variables that are not in ϕ_1.*

***QUESTION**: Does every truth assignment T of ϕ_1 which converts ϕ_2 into a tautology after its evaluation in T, a satisfying truth assignment of ϕ_1?*

A truth assignment for a Boolean formula ϕ is a set of values for the variables of ϕ and a satisfying truth assignment is a truth assignment that causes it to evaluate to true.

Let's see one example of this language:

$$\phi_1 = p \wedge q$$

$$\phi_2 = (p \wedge r) \vee (q \wedge \neg r).$$

The only truth assignment T of ϕ_1 that makes ϕ_2 a tautology after its evaluation in T is $p = true$ and $q = true$. Certainly, the formula ϕ_2 after the evaluation in T would be $r \vee \neg r$, that is, a trivial tautology [4]. However, T will be a satisfying truth assignment of ϕ_1, and thus, $\langle \phi_1; \phi_2 \rangle \in EVALUATION$.

Theorem 2.5. *If $P = NP$, then EVALUATION would be in P.*

Proof. We can find a succinct disqualification of an instance $\langle \phi_1; \phi_2 \rangle$, when this one would be a "no" instance of EVALUATION, if $P = NP$. Given a truth assignment T of ϕ_1, we could check in polynomial-time whether ϕ_2 is a tautology after its evaluation in T, because TAUTOLOGY would be in P. A language $L_{NT} = \{\langle \phi_1; \phi_2 \rangle\}$, such that there is not any truth assignment T of ϕ_1 that makes ϕ_2 a tautology after its evaluation in T, would be in $coNP$. Certainly, its complement would be in NP, since as we mentioned before, we can verify in polynomial-time whether a truth assignment T of ϕ_1 converts ϕ_2 into a tautology when $P = NP$. But, these kind of "no" instances of EVALUATION, which are the elements of L_{NT}, could be checked in polynomial-time. Indeed, L_{NT} would be in $coNP$, and therefore, this language would be in P. At the same time, if some truth assignment T of ϕ_1, such that ϕ_2 is converted into a tautology after its evaluation in T, is not a satisfying truth assignment of ϕ_1, then this could be verified in polynomial-time. Certainly, we could verify in polynomial-time whether a truth assignment is not a satisfying truth assignment of ϕ_1 [4]. Hence, we have proved that EVALUATION would be in $coNP$ if we assume that $P = NP$, and thus, it would be in P too. □

Now, let's build the following Boolean functions from the previous functions f_C, f_Q, δ and γ:

$$\psi'(X, Y) = \delta(X, Y) \Rightarrow (f_Q(X, Y) \Rightarrow f_C(X, Y))$$

$$\varphi'(X, Y, Z) = \gamma(X, Y, Z) \Rightarrow (\neg f_Q(X, Z) \vee \neg f_Q(Z, Y)).$$

In addition, we obtain the Boolean expressions ψ'_1 and φ'_2 from the Boolean functions ψ' and φ' respectively. We can see φ'_2 is a tautology for a truth assignment of ψ'_1 if and only the nodes represented by X and Y are consecutive nodes in the binary relationship Q, or $(\neg f_Q(X, Y))$ is true, or the *b-bits* integers represented by X and Y are not between 1 and n. In this way, any two consecutive nodes in a binary relationship Q must be adjacent in G_C if and only if $\langle \psi'_1; \varphi'_2 \rangle \in EVALUATION$. But, as we just proved before, EVALUATION would be in P, and thus, we could verify in polynomial-time the third and last property of a linear order Q on nodes of G_C too.

Finally, we have checked in polynomial-time whether a succinct representation C_Q of a graph G_Q with n nodes could be a linear order Q on nodes of the graph G_C if $P = NP$. That is equivalent to show the graph represented by the arbitrary Boolean circuit C can contain a Hamilton path. In this way, we have shown the polynomially balanced relation R_Q would be polynomially decidable when $P = NP$. For this purpose, the Boolean functions f_C, f_Q, f_{bt} and f_{eq} can be created in polynomial-time from the circuits C, C_Q, C_{bt} and C_{eq} [4]. Furthermore, we have obtained in polynomial-time the Boolean circuits C_{bt} and C_{eq}, since $BETWEEN \in P$ and $EQUAL \in P$ [4]. In conclusion, we have demonstrated if $P = NP$, then SUCCINCT HAMILTON PATH would be in P. □

Theorem 2.6. $P \neq NP$.

Proof. We start assuming that $P = NP$. The Theorem 2.1 states when $P = NP$, the problem SUCCINCT HAMILTON PATH would be in P. But, we already know if $P = NP$, then $EXP = NEXP$ [4]. Since SUCCINCT HAMILTON PATH is in *NEXP–complete*, then it would be in *EXP–complete*, because the completeness of both classes uses the polynomial-time reduction [4]. But, if some *EXP–complete* problem is in P, then P should be equal to EXP, because P and EXP are closed under reductions and P is a subset of EXP [4]. However, as result of the Hierarchy Theorem the class P cannot be equal to EXP [4]. 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 applying the Reductio ad absurdum rule. □

3. 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 [8]. 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 [9]. 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, in: Proceedings of the 3rd IEEE Symp. on the Foundations of Computer Science, 1971, pp. 151–158.
[2] L. Fortnow, The Status of the P versus NP Problem, Communications of the ACM 52 (9) (2009) 78–86. doi:10.1145/1562164.1562186.
[3] A. M. Turing, On Computable Numbers, with an Application to the Entscheidungsproblem, Proceedings of the London Mathematical Society 42 (1936) 230–265.
[4] C. H. Papadimitriou, Computational complexity, Addison-Wesley, 1994.
[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 2nd Edition, MIT Press, 2001.
[6] M. Sipser, Introduction to the Theory of Computation, 2nd Edition, Thomson Course Technology, 2006.
[7] W. I. Gasarch, The P=?NP poll, SIGACT News 33 (2) (2002) 34–47.
[8] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, 1st Edition, San Francisco: W. H. Freeman and Company, 1979.
[9] O. Goldreich, P, Np, and Np-Completeness, Cambridge: Cambridge University Press, 2010.