

P versus NP under codings

Frank Vega

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? This question was first mentioned in a letter written by John Nash to the National Security Agency in 1955. 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. We define a coding to be a mapping from symbols of some alphabet (not necessarily one-to-one). NP is closed under codings. However, P is closed under codings if and only if $P = NP$. We show a coding of a NP language which produces a NEXP-complete problem when the empty string is considered as a symbol. If $P = NP$, then this NEXP-complete language would be in P, but this is not possible due to the Hierarchy Theorem. In this way, we prove P is not equal to NP when the empty string is taken as a symbol.

1. Introduction

The P versus NP problem is a major unsolved problem in computer science [6]. This is considered by many to be the most important open problem in the field [6]. 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 [6]. It was essentially mentioned in 1955 from a letter written by John Nash to the United States National Security Agency [6]. However, the precise statement of the $P = NP$ problem was introduced in 1971 by Stephen Cook in a seminal paper [6].

In 1936, Turing developed his theoretical computational model [3]. The deterministic and nondeterministic Turing machines have become in two of the most important definitions related to this theoretical model for computation [3]. A deterministic Turing machine has only one next action for each step defined in its program or transition function [3]. 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 [3]. 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 [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].

The set of languages decided by deterministic Turing machines within time f is an important complexity class denoted $TIME(f(n))$ [3]. In addition, the complexity class $NTIME(f(n))$ consists in those languages that can be decided within time f by nondeterministic Turing machines [3]. 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 fixed constant k [3]. At the same time, NP consists in all languages in $NTIME(n^k)$ for every possible positive fixed constant k [3]. NP is also the complexity class of languages whose solutions may be verified in polynomial time [3]. 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 [4].

2. Theory

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 $M(w) = \text{"yes"}$ [1]. Note that M fails to accept w either if this computation ends in the rejecting state, that is $M(w) = \text{"no"}$, 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:

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

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 is a constant k such that for all n , $T_M(n) \leq n^k + k$ [1]. 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 [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 [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 [3].

A function $f : \Sigma^* \rightarrow \Sigma^*$ 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 [1]. 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 $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ 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 [5]. 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 [2]. Moreover, if $L \in NP$, then $L \in NP$ -complete [2].

HAMILTON-PATH is an important NP -complete problem [5]. An instance of the language **HAMILTON-PATH** is a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges, each edge being an ordered pair of vertices [5]. 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 path in G is a sequence of distinct vertices $\langle v_0, v_1, v_2, \dots, v_k \rangle$ such that $(v_{i-1}, v_i) \in E$ for $i = 1, 2, \dots, k$ [2]. A Hamilton path is a simple path of the graph which contains all the vertices of the graph. The problem **HAMILTON-PATH** asks whether a graph has a Hamilton path [5].

Another *NP-complete* problem is *CIRCUIT-SAT* [5]. A Boolean circuit is an acyclic graph $C = (V, E)$, where the nodes $V = \{1, \dots, n\}$ are called the gates of C . We can assume that all edges are of the form (i, j) where $i < j$. All nodes in the graph have in-degree (number of incoming edges) equal to 0, 1 and 2. Also, each gate $i \in V$ has a sort $c(i)$ associated with it, where $c(i) \in \{\text{true}, \text{false}, \wedge, \vee, \neg\} \cup \{x_1, x_2, \dots\}$. If $c(i) \in \{\text{true}, \text{false}\} \cup \{x_1, x_2, \dots\}$, then the in-degree of i is 0, that is, i must have no incoming edges. Gates with no incoming edges are called the inputs of C . If $c(i) = \neg$, then i has in-degree one. If $c(i) \in \{\wedge, \vee\}$, then the in-degree of i must be two. Finally, node n (the largest numbered gate in the circuit, which necessarily has no outgoing edges), is called the output gate of the circuit. Let $X(C)$ be the set of all Boolean variables that appear in the circuit C (that is, $X(C) = \{x \in X : c(i) = x \text{ for some gate } i \text{ in } C\}$). We say that a truth assignment T is appropriate for C if it is defined for all the variables in $X(C)$. The problem *CIRCUIT-SAT* asks whether a given circuit C has a truth assignment T , appropriate to C , such that $C(T) = \text{true}$. Consider, however, the same problem for circuits with no variable gates. This problem, known as *CIRCUIT-VALUE*, obviously has a polynomial time algorithm [3].

On the other hand, *EXP* is the complexity class of languages that can be accepted in exponential time by deterministic Turing machines [2]. *NEXP* is the complexity class of languages defined by exponential time verifiers [3]. *NEXP-complete* is also defined under polynomial time reductions but each problem is in *NEXP*. One of the most important problems related to circuits and graph is *SUCCINCT-HAMILTON-PATH*. A succinct representation of a graph with $2 \times n - 1$ nodes is a Boolean circuit C with $2 \times b$ input gates where $n = 2^b$ is a power of two [3]. The graph represented by C , denoted G_C , is defined as follows: The nodes of G_C are $\{0, 1, 2, \dots, 2 \times n - 1\}$. 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 [3]. The problem *SUCCINCT-HAMILTON-PATH* is now this: Given the succinct representation C of a graph G_C with $2 \times n - 1$ nodes, does G_C have a Hamilton path? The problem *SUCCINCT-HAMILTON-PATH* is in *NEXP-complete* [3].

3. Results

DEFINITION 1. **CIRCUIT-HAMILTON-PATH**

Instance: A Boolean circuit C and a graph $G = (V, E)$.

Question: Does G have a Hamilton path where C is a succinct representation of G ?

THEOREM 3.1. *CIRCUIT-HAMILTON-PATH* $\in NP$.

Proof. We can check whether a simple path in G is a Hamilton path in polynomial time since *HAMILTON-PATH* $\in NP$. Moreover, we can check in polynomial time whether G has $2 \times n - 1$ nodes where $n = 2^b$ is a power of two. Furthermore, we can measure whether the size of C is upper bounded by b^k for a “feasible” positive integer k . Finally, we can verify in polynomial time whether every ordered pair of vertices (u, v) complies with $(u, v) \in E$ if and only if C accepts the binary representations of the b -bits integers u, v as inputs. \square

DEFINITION 2. We define a coding κ to be a mapping from Σ to Σ (not necessarily one-to-one) [3]. If $x = \sigma_1 \dots \sigma_n$, we define $\kappa(x) = \kappa(\sigma_1) \dots \kappa(\sigma_n)$ [3]. Finally, if $L \subseteq \Sigma^*$ is a language, we define $\kappa(L) = \{\kappa(x) : x \in L\}$ [3].

DEFINITION 3. **ENCODED-CIRCUIT-HAMILTON-PATH**

Instance: A string $\kappa(C)$ where C is a Boolean circuit and a graph $G = (V, E)$.

Question: Does G have a Hamilton path where C is a succinct representation of G ?

κ is a one-to-one mapping defined as $\kappa(0) = +$ and $\kappa(1) = -$.

THEOREM 3.2. *ENCODED-CIRCUIT-HAMILTON-PATH* $\in NP$.

Proof. *ENCODED-CIRCUIT-HAMILTON-PATH* is in NP , because we can evaluate in polynomial time κ^{-1} on $\kappa(C)$ to obtain C and *CIRCUIT-HAMILTON-PATH* is in NP . \square

THEOREM 3.3. *If we take the empty string ϵ as a symbol, then we obtain:*

$\kappa'(\text{ENCODED-CIRCUIT-HAMILTON-PATH}) = \text{SUCCINCT-HAMILTON-PATH}$,

κ' is a coding defined as $\kappa'(+) = 0$, $\kappa'(-) = 1$, $\kappa'(1) = \epsilon$ and $\kappa'(0) = \epsilon$.

Proof. The string $\kappa(C)G$ encoded in κ' is $C\epsilon \dots \epsilon$, but $C\epsilon \dots$ is equal to the Boolean circuit C because the empty string ϵ complies with $\epsilon\epsilon = \epsilon$. \square

THEOREM 3.4. *P is not closed under codings when we take the empty string as a symbol.*

Proof. If P is closed under codings and we take the empty string as a symbol, then *SUCCINCT-HAMILTON-PATH* would be P . However, there is not any *NEXP-complete* in P due to the Hierarchy Theorem. \square

THEOREM 3.5. *$P \neq NP$ when we take the empty string as a symbol.*

Proof. P is closed under codings if and only if $P = NP$ [3]. Hence, we prove $P \neq NP$ when we assume the empty string is a symbol. \square

References

1. S. ARORA and B. BARAK, *Computational complexity: a modern approach*, CAMBRIDGE UNIVERSITY PRESS, 2009.
2. T. H. CORMEN and C. E. LEISERSON and R. L. RIVEST and C. STEIN, *Introduction to Algorithms*, 3RD EDITION, THE MIT PRESS, 2009.
3. C. H. PAPADIMITRIOU, *Computational complexity*, ADDISON-WESLEY, 1994.
4. W. I. GASARCH, *Guest column: The second $P \stackrel{?}{=} NP$ poll*, ACM SIGACT NEWS (43) (2012) 53–77.
5. M. R. GAREY and D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1ST EDITION, SAN FRANCISCO: W. H. FREEMAN AND COMPANY, 1979.
6. S. AARONSON, *$P \stackrel{?}{=} NP$* , ELECTRONIC COLLOQUIUM ON COMPUTATIONAL COMPLEXITY, REPORT NO. 4 (2017).

Frank Vega

Joysonic, Uzun Mirkova 5, Belgrade, 11000
Serbia

vega.frank@gmail.com