Satisfied with Satisfiability?– A Note on the First NP-complete Problem

John Paul C. Vergara
Department of Information Systems and Computer Science
Ateneo de Manila University
Quezon City 1108 Philippines

ABSTRACT

In this paper, we revisit the theory of NP-completeness and present an alternative to Satisfiability (SAT) as the "first" NP-complete problem. We define the decision problem Nondeterministic Turing Machine Computation (NTMC) and present it as the alternative. An NTMC instance consists of a non-deterministic Turing machine, an input string, and an integer time bound. The decision problem seeks to answer the question of whether the machine accepts the input string within the given time. The approach was first presented by Lewis and Papadimitriou in their classic textbook, *Elements of the Theory of Computation* [6]. We review Cook's Theorem and its proof and present arguments on why NTMC is a reasonable alternative as the first NP-complete problem.

1. INTRODUCTION

Author's note: A version of this paper has appeared in Philippine Journal of Computing. Corrections on some technical definitions and various clarifications have been incorporated in this version.

The theory of NP-completeness is a central concept in the area of computational complexity. Thousands of decision problems have been analyzed and proven to be NP-complete, thereby placing them in a special class of intractable problems. Three classes of problems are important in this discussion:

- P: the class of decision problems whose acceptable instances can be decided in (deterministic) polynomial time.
- NP: the class of decision problems whose acceptable instances can be decided in *non-deterministic* polynomial time.
- NP-complete problems: the subclass of NP that contains all problems π such that for each $\pi' \in NP$, there exists a polynomial-time transformation from π' to π.

NP-complete problems are critical in our understanding of the $P = NP$ question, one of the most important unresolved questions in computer science. If even one NP-complete problem is shown to be in P, then the $P = NP$ question would be resolved in the affirmative, although this is generally thought to be unlikely. Showing that a problem is NP-complete is thus a definitive statement of intractability; that is, the problem is so difficult that discovering an efficient solution to it will be groundbreaking.

Due to the transitivity of polynomial transformations, proving that a problem $\pi \in NP$ is NP-complete typically involves providing a polynomial-time transformation from a known NP-complete problem to π. This, however, necessitates a "first" NP-complete problem, whose NP-completeness requires a more complicated proof. Specifically, the proof needs to satisfy the conditions of completeness as defined and ensure that *every* problem in NP polynomially transforms to that problem. That first NP-complete problem is Satisfiability (SAT), and the proof of its NP-completeness is due to Cook [1]. A SAT instance involves a set of variables and clauses over those variables; we seek to determine whether there is a truth assignment for the variables that satisfies all the clauses.

In this paper, we propose an alternative first NP-complete problem, Non-deterministic Turing Machine Computation (NTMC). An NTMC instance consists of a non-deterministic Turing machine, an input string, and an integer time bound; here, we seek to determine whether the machine accepts the input string within the given time. We are in fact reviving an approach first introduced by Lewis and Papadimitriou in the first edition of their textbook, *Elements of the Theory of Computation* [6]. While almost all textbooks that discuss computational complexity use SAT as the first NP-complete problem, Lewis and Papadimitriou begin by establishing the NP-completeness of NTMC, after which they provide a polynomial transformation from NTMC to the Bounded Tiling problem. Savelsberg and van Emde Boas [8] refer to this unique approach and propose Bounded Tiling as an alternative to Satisfiability.

The rest of this paper is organized as follows. Section 2 provides some formalisms on Turing machines and complexity classes. In Section 3, we review Cook's Theorem and provide a proof sketch for the theorem. In Section 4, we define NTMC and present arguments on its suitability as the first NP-complete problem. We conclude in Section 5 where we

summarize our discussions.

2. PRELIMINARIES

We use the Turing machine as a model for computation, and review its definition and significance in this section. A Turing machine consists of a finite control with a tape head moving along cells of a two-way infinite tape. Computation on the machine begins with an input string on the tape and the tape head pointing at the first symbol of the string. It then proceeds with the control carrying out actions on the cells of the tape until it reaches a final state.

Formally, a deterministic Turing machine M is a tuple $(Q, \Sigma, \delta, b, q_0, q_{fy}, q_{fn})$, where Q is a set of states, Σ is the tape alphabet, $b \in \Sigma$ is the special blank symbol, $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R\}$ is the transition function mapping a state-symbol pair to an action (state-symbol-direction triple), $q_0 \in Q$ is the start state, and $q_{fy}, q_{fn} \in Q$ are accepting and rejecting final states, respectively. The literature has defined Turing machines in different ways (see [2, 3, 7, 9] for some examples). The definition given was chosen to fit the purposes of this paper.

A computation on a Turing machine M involves an input string over the alphabet Σ. The machine M starts in state q_0 with a string $s \in \Sigma*$ bounded by infinitely many blank symbols placed on its tape and the tape head pointing at the leftmost symbol of s. The machine proceeds by repeated applications of the transition function δ, each carrying out an action based on the current state and the current symbol. An action involves overwriting the symbol on the tape, moving the head left (L) or right (R), and transitioning to a different state. The machine halts once a final state (q_{fy} or q_{fn}) is reached. It is of course possible that a computation will never reach a final state for a given input string. If it does halt, we say that the machine completes in t steps, where t is the number of times an action is carried out, equivalently, the number of applications of δ.

Turing machines can serve as deciders for strings; that is, a machine M *accepts* a string s if the resulting computation reaches q_{fy}, and *rejects* s if the computation reaches q_{fn}. On the other hand, Turing machines can also compute functions on strings; The string left on the tape after the machine halts serves as the computed output that corresponds to the input string. In this case, M computes a function $f_M : \Sigma* \rightarrow \Sigma*$, if for all $s \in \Sigma*$, M halts on input s (we may just require that all computations end in q_{fy}). The Church-Turing thesis in fact states that the set of problems that are computable (even when using computers with more powerful features) is the same as the set of problems that can be solved by a Turing machine.

Turing machines pave the way for the definitions of complexity classes. The reader is referred to textbooks on algorithms or computational complexity [2, 3, 7, 9] for more detailed formalisms on the rest of the statements in this section. Noting that problem instances can always be encoded by strings over some alphabet, a Turing machine may represent a solution to a given problem. Specifically, for a decision problem, a Turing machine *solves* that problem if it accepts only those strings whose correponding instances have "yes" answers to the problem's question, and rejects those that have "no" answers. The class P is defined to contain all decision problems π such that there exists a deterministic Turing machine that solves π within $p(n)$ steps for all its instances, where p is some polynomial over n. Here, n is the length of an encoding of an instance of π.

A *non-deterministic* Turing machine M follows the same definition as its deterministic counterpart except that δ is now a transition *relation*. In effect, multiple actions may correspond to a given state-symbol pair. The meaning of a machine accepting an input string is adjusted accordingly: for a non-deterministic machine M and a given input string s, M accepts s as long as there exists a sequence of actions that lead to q_{fy}, consistent with δ. Analogous statements for a non-deterministic Turing machine M solving a decision problem π holds.

The class NP is defined to contain all decision problems π such that there exists a non-deterministic Turing machine that solves π within $p(n)$ steps for all its instances, where p is some polynomial over n. Clearly, $P \subseteq NP$. It is not yet known if $P = NP$, although this has been conjectured to be not the case.

Let π_1 and π_2 be decision problems, a *transformation* f from π_1 to π_2 is a mapping from instances of π to instances of π_2, such that an instance I is a "yes"-instance of π_1 if and only if the instance $f(I)$ is "yes"-instance of π_2. We say that π_1 transforms to π_2.

The transformation is a *polynomial transformation* if it can be carried out in polynomial time; that is, there exists a deterministic Turing machine (this time used as a machine that computes a function on strings) that performs the transformation within $p(n)$ steps, for some polynomial (n is the length of the encoding of an instance).

A decision problem π is NP-complete if $\pi \in NP$ and for all $\pi' \in NP$, there exists a polynomial-time transformation from π' to π. NP-complete problems are significant because they represent a class of intractable problems virtually equivalent to each other via polynomial-time transformations. If a polynomial-time solution is discovered for even one NP-complete problem, then all problems in NP are rendered solvable in polynomial time, meaning $P = NP$. Showing that a problem is NP-complete is thus a definitive statement of intractability.

There are two ways to prove that a problem π is NP complete:

1. Show that $\pi \in NP$, and then show that for every problem $\pi' \in NP$, there is a polynomial transformation from π' to π.
2. Show that $\pi \in NP$, and then provide a polynomial transformation from a known NP-complete problem to π.

The second approach is valid because of the transitivity of polynomial-time transformations. However, it necessitates a "first" NP-complete problem, whose proof has to involve the first approach. That proof is the topic of the next section.

3. COOK'S THEOREM

Satisfiability is traditionally the first NP-complete problem. We provide a formal definition for this problem:

SATISFIABILITY (SAT)

INSTANCE: A set U of boolean variables and a collection C of clauses over U (each clause in C is a set of positive or negative literals over U).

QUESTION: Is there a truth assignment for U that satisfies all the clauses in C?

The following result is due to Cook [1], and was independently arrived at by Levin [5]. We sketch a proof similar to that described in Garey and Johnson's definitive reference to the theory of NP-completeness [2].

THEOREM 1. *SAT is NP-complete.*

PROOF. It is straightforward to show that SAT is in NP. A non-deterministic Turing machine can non-deterministically generate a truth assignment ($|U|$ true-or-false values) on the tape, and then verify (deterministically) in polynomial time if the assignment satisfies at least one literal for each clause, before proceeding to an accepting state.

To show that for every problem $\pi' \in NP$, there is a polynomial transformation from π' to SAT, we use the only characterization available for a problem in NP: the existence of a non-deterministic Turing machine M that solves π' within $p(n)$ steps. We create a SAT instance from this characterization; that is, we define a set of variables U and clauses C, such that there is a truth assignment for U that satisfies C if and only if the instance of π' is accepted by M within $p(n)$ steps.

The set U of variables are as follows:

- $Q[i,k]$ (for each time step i and each state k) holds the value true if at time i, M is in state q_k.
- $H[i,j]$ (for each time step i and tape cell j) holds the value true if at time i, the tape head is on cell j.
- $S[i,j,k]$ (for each time step i, tape cell j and symbol k) holds the value true if at time i, the contents of tape cell j is the symbol k.

The set C of clauses over U are arranged to represent the following:

1. At each time i, M is in exactly one state.
2. At each time i, the tape head is on exactly one tape cell.
3. At each time i, each tape cell contains exactly one symbol.
4. At time 0, the tape contains an input string of length n (the encoding of the instance of π'), the tape head is on the cell containing the leftmost symbol of the string, and M is in its initial state.
5. By time $p(n)$, M has reached the accepting state.
6. For each time i, the transitions to time $i+1$ (change in state and symbol) are consistent with M's transition relation δ.

We omit the details of the actual literals in the clauses (see [2] for a complete proof), but with the above construction, it can be verified that the transformation is a polynomial transformation and that there is a truth assignment for U that satisfies C if and only if the instance of π' is accepted by M within $p(n)$ steps. □

The above proof is a classic result and launched the growth of the class of NP-complete problems [2, 4]. It provided a first NP-complete problem, which meant that succeeding proofs for NP-completeness need only select an existing NP-complete problem to transform from to render a given problem NP-complete.

In the next section, we propose an alternative first NP-complete problem, and suggest it as a more natural choice.

4. THE ALTERNATIVE: NTMC

We begin by providing a formal definition for the problem of Non-deterministic Turing Machine Computation.

NON-DETERMINISTIC TURING MACHINE COMPUTATION (NTMC)

INSTANCE: A non-deterministic Turing machine M with alphabet Σ, a string $s \in \Sigma*$, and an integer t. [1]

QUESTION: Does M accept s within t steps?

We then propose that the proof of Cook's Theorem be broken down into two parts as follows:

1. A polynomial transformation from any problem in NP to NTMC.
2. A polynomial transformation from NTMC to SAT.

The first part was an approach actually taken by Lewis and Papadimitriou in the first edition of their textbook, *Elements of the Theory of Computation* [6]. (Curiously, the approach was not used in the second edition of their book [7].) We state and prove it here as a lemma.

LEMMA 1. *NTMC is NP-complete.*

[1]For the purposes of this paper, we assume that t is unary-encoded or that t is a polynomial on $|s|$.

PROOF. It is straightforward to show that NTMC is in NP. A non-deterministic Turing machine can non-deterministically generate a sequence of t transitions from M's δ (where each transition contains a state-symbol pair and a state-symbol-direction triple) and then verify (deterministically) in polynomial time if the sequence of transitions represents a valid simulation of M on s and leads to M's accepting final state.

Let π' be any problem in NP. By definition, there exists a Turing machine M' that solves π' in polynomial time. Therefore, an acceptable instance I' of π' brings M' to its accepting final state within $p(n)$ steps.

We create an instance of NTMC as follows: $M = M'$, $I = I'$, and $t = p(n)$. Trivially, M' accepts I' within $p(n)$ steps if and only if M accepts I within t steps. The lemma follows. □

The proof of Cook's Theorem would then invoke the above lemma and provide a transformation from NTMC to SAT.

The alternative proposed would appear superfluous on the surface, but it provides several conveniences:

1. The approach manages the complexity of the proof of Cook's Theorem. Transforming from *every* NP problem to a given problem is not too easy to comprehend for those who are new to the theory of NP-completeness. The directness of the transformation, if NTMC is used as the destination problem, helps simplify the proof. The second stage of the proof is then a simpler transformation from a single problem (NTMC) to another (SAT).

2. We can say that NP-complete problems are those problems that are "polynomially equivalent to the problem of Non-deterministic (Polynomial) Turing Machine Computation", providing a compact description consistent with the definition of the class.

3. We found at least one reference that proposed an alternative to Satisfiability. Savelsberg and van Emde Boas [8] propose Bounded Tiling problem as an alternative, particularly because they found that a transformation from a Turing machine instance was more appropriate. Had NTMC been a declared NP-complete problem, then the proof for the problem's NP-completeness would have been slightly less complex, involving a transformation from a single problem to another, instead of following an approach similar to Cook's Theorem.

5. SUMMARY

At the very least, this paper provides a review of the theory of NP-completeness and Cook's Theorem. It also proposes an approach that attempts to improve the explanation of the theory by providing an alternative to Satisfiability as the first NP-complete problem. The alternative problem is called Non-deterministic Turing Machine Computation (NTMC). The NP-completeness of NTMC is established, and arguments regarding the suitability of this problem as the first NP-complete problem are presented.

6. REFERENCES

[1] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *STOC*, pages 151–158. ACM, 1971.

[2] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[3] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation - (2. ed.)*. Addison-Wesley series in computer science. Addison-Wesley-Longman, 2001.

[4] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[5] L.A. Levin. Universal sequential search problems. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973.

[6] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, 1981.

[7] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the theory of computation (2. ed.)*. Prentice Hall, 1998.

[8] W.P. Savelsbergh and P. van Emde Boas. *Bounded Tiling, an Alternative to Satisfiability?* Report. Centrum voor Wiskunde en Informatica. Stichting Mathematisch Centrum. OS. Centrum voor Wiskunde en Informatica, 1984.

[9] Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.