P Versus NP: More than just a prize problem

Dr Uday Singh Rajput
Department Of Mathematics and Astronomy,
University of Lucknow, Lucknow, India

email: usrajput07@gmail.com

Abstract

In May 2000, the Clay Mathematics Institute (CMI) in Cambridge, MA, put forth seven problems[1], named Millennium Prize problems, and offered $ 1 million for each problem to one who provides a verified proof. The problems are open to scientists/ researchers/teachers and public in general. P versus NP is one of these problems. However, due to its importance, P versus NP is much more than just a mathematical puzzle, or more appropriately more than a prize problem. It seeks to determine, finally, which kinds of problems present day computers can solve and which not. The aim of this article is to discuss the importance of P versus NP in academic and social life.

Subject class [2010]:03D15
Keywords: P versus NP, complexity theory

1 INTRODUCTION

Computability theory is an advanced topic of mathematics and computer science. It not only analyses the complexity of existing solutions but also encourages the researchers to develop general algorithms for solving real life problems. The effort is to find the solution of every problem in reasonable time. Some problems seem very simple to solve while others quite difficult. The difficulty of a problem is measured in terms of complexity[3, 5]. Broadly, two types of complexities have been defined: Space complexity and time complexity. Space complexity is the measure of space taken by the algorithm during the execution. It is not measured in actual unit but is expressed in a general notation (Big O), which may be the function of its data size, e.g. O(n), where n may be the number of integers in a sorting algorithm. Similarly, time complexity is the measure of time taken by the algorithm to solve a problem. Like space complexity, time complexity is also measured in terms of Big O. At present, the study of space complexity is becoming irrelevant due to the invention of modern computers equipped with relatively large space for execution of algorithms. Thus, we will mainly focus on time complexity.

2 CLASSIFICATION OF PROBLEMS

There are two types of problems: tractable and intractable[6]. Tractable problems are those, which can be solved easily and thus, have efficient algorithms. Efficient algorithms

are those, which are fast enough, i.e. have less execution time. In addition, these algorithms take less space. On the other hand, intractable problems are difficult to solve. Even if the solution of an intractable problem exists, it has very high time complexity. There are several intractable problems, which need attention of researchers. We further classify problems into following categories.
1. Polynomial time (P)
2. Nondeterministic polynomial time (NP)
3. NP complete and NP hard.

3 POLYNOMIAL TIME(P) ALGORITHMS

To model the problem it is necessary to give a formal model of a computer. The standard computer model in computability theory is the Turing machine model, introduced by Alan Turing in 1936 [7]. The class P is the class of problems solvable by algorithms within a reasonable time or within a reasonable number of steps. Turing machines can simulate efficient computer models that can execute in polynomial time. Thus, P is a class that has problems, which can be simulated on Turing machines with reasonable complexity. In terms of Big O notation, the time complexity of polynomial time algorithms is :
$O(n), O(n^2), O(n^3), O(n^4), O(1), O(n \log n)$.
The time complexity of non polynomial time algorithms is :
$O(2^n), O(n^n), O(n!)$ etc.
Examples of Polynomial time algorithms are bubble sort, quick sort, hashing etc. All trivial and fast running algorithms are obviously in class P.

4 NONDETERMINISTIC POLYNOMIAL TIME ALGORITHMS(NP)

The notation NP stands for nondeterministic polynomial time, since originally NP was defined in terms of nondeterministic machines that have more than one possible moves from a given configuration. Out of all moves, one may lead to the solution of the problem. Thus, it may offer choices, and one may have to guess for the correct choice. In contrast, deterministic machines have definite path to a solution. Obviously deterministic machines are more efficient, though both the models have same computational power. NP type problems are difficult to solve but easier to validate, i.e. if solution is given its correctness can be verified easily in polynomial time. Such problems are not only interesting but are very important too. Example: Sudoku (figure-1), Three-colorability etc. Solution of this Sudoku (figure-1), is given in figure-2. Three coloring is a similar problem in graph theory.

Figure-1

9					1		5	
7	6	5						
1			3					8
					6		4	
			2	1	8			
	9		4					
6					4			2
								7
	3		1					5

Figure-2

9	8	3	7	2	1	6	5	4
7	6	5	8	4	9	3	2	1
1	2	4	3	6	5	7	9	8
8	1	7	9	5	6	2	4	3
3	4	6	2	1	8	5	7	9
5	9	2	4	7	3	1	8	6
6	7	8	5	3	4	9	1	2
4	5	1	6	9	2	8	3	7
2	3	9	1	8	7	4	6	5

5 NP COMPLETE AND NP HARD

NP-complete problems are the hardest problems in NP. Sudoku and 3-colorability are also NP complete. It is general belief that if there is a fast (polynomial-time) algorithm for one NP-complete problem, then there is a fast algorithm for every problem in NP. For example, a fast algorithm for Sudoku may imply P = NP. NP-complete problems are, basically, in NP. The set of all decision problems whose solutions can be verified in polynomial time are in NP. NP may be equivalently defined as the set of decision problems that can be solved in polynomial time on a non-deterministic Turing machine. A problem p in NP is NP-complete if every other problem in NP can be transformed (or reduced) into p in polynomial time. NP-complete problems are studied because the ability to quickly verify solutions of NP problems seems to correlate with the ability to quickly solve that problem (P). It is not known whether every problem in NP can be quickly solvedthis is called the P versus NP problem. However, if any NP-complete problem can be solved quickly, then every problem in NP can also be solved quickly. Because of this, it is often said that NP-complete problems are harder or more difficult than NP problems in general. Formally, A decision problem p is NP-complete if
1. It is in NP, and
2. Every problem in NP is reducible to p in polynomial time.
A decision problem x is NP-hard when for any problem y in NP, there is a polynomial-time reduction from y to x. It does not restrict the class NP-hard to decision problems, for instance, it also includes search problems, or optimization problems. An example of an NP-hard problem is the optimization problem of finding the least-cost cyclic route through all nodes of a weighted graph. This is commonly known as the traveling salesman problem. There are decision problems that are NP-hard but not NP-complete, for example the halting problem of Turing machine(HTM). This is the problem which asks - given a program and its input, will it run forever? It is easy to prove that the HTM is NP-hard but not NP-complete. For example, the Boolean satisfiability problem may be reduced to the HTM by transforming it to the description of a Turing machine that tries all truth-value assignments, and when it finds one which satisfies the formula, it halts, and otherwise it goes into an infinite loop. It is not difficult to see that the HTM is not in NP, since all problems

in NP are decidable in a finite number of operations, while the HTM, is undecidable. There are also NP-hard problems that are neither NP-complete nor undecidable. For instance, the language of True quantified Boolean formulas is decidable in polynomial space, but not non-deterministic polynomial time.

6 P VERSUS NP

The P versus NP problem is to find whether every problem solved by some nondeterministic algorithm in polynomial time can also be solved by some (deterministic) algorithm in polynomial time.
Formally, the elements of the class P can be simulated by languages. Let Σ be a finite alphabet, and let Σ^* be the set of finite strings over Σ. Then a language over Σ is a subset L of Σ^*. Each Turing machine M has an associated input alphabet Σ. We say that M accepts a string w if the computation associated with this string terminates in the accepting state in finite number of steps. Further, M fails to accept w, either if this computation ends in the rejecting state, or if the computation fails to terminate. The language accepted by Turing machine M is defined by $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$. We denote by tM (x) the number of steps in the computation of M on input x. If this computation never halts, then tM (x) $= \infty$. For $n \in N$ we denote by $TM(n)$ the worst case run time of M; that is, $TM(n) = max\{tM(w)|w \in \Sigma n\}$, where Σn is the set of all strings over Σ of length n. We say that M runs in polynomial time if there exists i such that for all $n, TM(n) \leq n^i + i$. Now we define the class P of languages by $P = \{L \mid L = L(M)$ for some Turing machine M that runs in polynomial time$\}$. Look at the question: Does $P = NP$?
It is easy to see that the answer is independent of the size of the alphabet Σ
(we assume $\mid \Sigma \mid \geq 2$), since strings over an alphabet of any fixed size can be easily coded by strings over a binary alphabet. It is trivial to show that $P \subset NP$, since for each language L over Σ, if $L \in P$ then we can define the polynomial-time checking relation $R \subset \Sigma^* \cup \Sigma^*$ by $R(w, y) \iff w \in L$ for all $w, y \in \Sigma^*$. There are two simple examples,
1. The set of perfect squares is in P, since Newtons method can be used to easily approximate square roots.
2. The set of composite numbers is in NP, where the associated polynomial time checking relation R is given by $R(a, b) \iff 1 < b < a$ and $b \mid a$.

7 IMPORTANCE

The importance of the P vs NP question stems from the successful theories of NP-completeness and complexity-based cryptography[2], as well as the potentially stunning practical consequences of a constructive proof of P = NP. The theory of NP-completeness has its roots in computability theory, which originated in the work of Turing, Church, Godel, and others in the 1930s. If P=NP, then finding a solution is not much difficult than verifying it. Theorem proving can be automated. If this is the case, then one may imagine big relief to mathematicians. The complexity theory in mathematics and computer science will almost vanish. In biology, finding the minimum energy 3-dimensional configuration of a protein (NP-hard) will become easier. Further, the statistical methods may be developed

for predicting structural classes of proteins based on their amino acids composition. Several other NP-complete problems have been identified, including Sub-set Sum (given a set of positive integers presented in decimal notation, and a target T, is there a subset summing to T?). Many graph problems : given a graph G, does G possess a Hamiltonian cycle? Does G contain a clique consisting of half of the vertices? Can the vertices of G be colored with three colors with distinct colors for adjacent vertices. These problems give rise to many scheduling and routing problems with industrial importance. There are interesting examples of NP problems not known to be either in P or in NP-complete. One example is the graph isomorphism problem: Given two undirected graphs, determine whether they are isomorphic. Computational complexity theory[4] plays an important role in modern cryptography [2]. The security of the Internet, including most financial transactions, depends on complexity-theory assumptions such as the complexity of integer factoring or of breaking DES (the Data Encryption Standard). If P = NP, these assumptions may become false. Although a practical algorithm for solving an NP-complete problem i.e. P = NP would have shocking consequences for cryptography, it would also have striking practical consequences of more efficient solutions to many useful NP-hard problems important to industry. For instance, it would transform mathematics by allowing a computer to find a formal proof of any theorem that has a proof of reasonable length, since formal proofs can easily be recognized in polynomial time. Well, such theorems may include all of the CMI millennium prize problems.

8 CONCLUSION

It has become obvious that finding proof of P versus NP is important.
Clearly, P versus NP is not merely a prize problem but a problem of great importance and hence researchers must pay more attention for finding its proof.

References

[1] Carlson J, Jaffe A, and Wiles A, Editors, The millennium prize problems, Clay Mathematics Institute, American Mathematical Society, 2006

[2] Goldreich O, The Foundations of Cryptography Volume 1, Cambridge University Press, Cambridge, UK, 2000.

[3] Blum L, Cucker F, Shub M, and Smale S, Complexity and Real Computation, Springer-Verlag, New York, 1998.

[4] Sipser M, Introduction to the Theory of Computation, PWS Publ., Boston, 1997.

[5] Papadimitriou C, Computational Complexity, Addison-Wesley, Reading, MA, 1994.

[6] Garey M R and Johnson D S, Computers and Intractability, a Guide to the Theory of NP-Completeness, WH Freeman and Co., San Francisco, 1979.

[7] Turing, A M (1936). "On Computable Numbers, with an Application to the Entscheidungs problem". Proceedings of the London Mathematical Society. 2 (1937) 42: 230265.