International Journal of Research Studies in Computer Science and Engineering (IJRSCSE)
Volume 2, Issue 3, March 2015, PP 102-104
ISSN 2349-4840 (Print) & ISSN 2349-4859 (Online)
www.arcjournals.org

P vs NP: One of the Millennium Prize Problems Proposed by the Clay Mathematics Institute

Bharathi Devi Patnala[1], Shamim[2]

[1]HoD, Dept of MCA, KBN PG College
[2]HOD, Dept of M.Sc(CS), KBN PG College

Abstract: *The P versus NP question distinguished itself as the central question of theoretical computer science nearly four decades ago. The question to resolve it, and more generally, to understand the power and limits of efficient computation, has led to the development of computational complexity theory. While this mathematical discipline in general, and the P vs. NP problem in particular, have gained prominence within the mathematics community in the past decade, it is still largely viewed as a problemof computer science.In this paper I shall try to explain why this problem, and describe the underlying concepts and problems, the attempts to understand and solve them, and some of the research directions this led us to. I shall also explain some of the important results, as well as the major goals and conjectures which stilelude us.*

1. INTRODUCTION

In 1936, Turing developed his theoretical computational model. He based his model on how he perceived mathematicians think. As digital computers were developed in the 40's and 50's, the Turing machine proved itself as the right theoretical model for computation. Quickly though we discovered that the basic Turing machine model fails to account for the amount of time or memory needed by a computer, a critical issue today but even more so in those early days of computing. The key idea to measure time and space as a function of the length of the input came in the early 1960's by Hartmanis and Stearns. And thus computational complexity was born. In the early days of complexity, researchers just tried understanding these new measures and how they related to each other. We saw the first notion of efficient computation by using time polynomial in the input size. This led to complexity's most important concept, NP-completeness, and its most fundamental question, whether P = NP?.The work of Cook and Karp in the early 70's showed lalargenumber of combinatorial and logical problems were NP-complete, i.e., as hard as any problem computable in nondeterministic polynomial time. The P=NP question is equivalent to an efficient solution of any of these problems. In the thirty years hence this problem has become one of the outstanding open questions in computer science and indeed all of mathematics. In the 70's we saw the growth of complexity classes as researchers tried to encompass different models of computations. One of those models, probabilistic computation, started with a probabilistic test for primality, led to probabilistic complexity classes and a new kind of interactive proof system that itself led to hardness results for approximating certain NP-complete problems. We have also seen strong evidence that we can remove the randomness from computations and most recently a deterministic algorithm for the original primality problem. In the 80's we saw the rise of finite models like circuits that capture computation in an inherently different way. A new approach to problems like P = NP arose from these circuits and though they have had limited success in separating complexity classes, this approach brought combinatorial techniques into the area and led to a much better understanding of the limits of these devices.In the 90's we have seen the study of new models of computation like quantum computers and propositional proof systems. Tools from the past have greatly helped our understanding of these new areas. In the year 2006,On 6 August, VinayDeolalikar, a mathematician at Hewlett-Packard Labs in Palo Alto, California, sent out draft copies of a paper titled simply "P ≠ NP".This terse assertion could have profound implications for the ability of computers to solve many kinds of problem.

2. CLASSIFICATION OF P, NP, NP-COMPLETENESS, NP-HARD

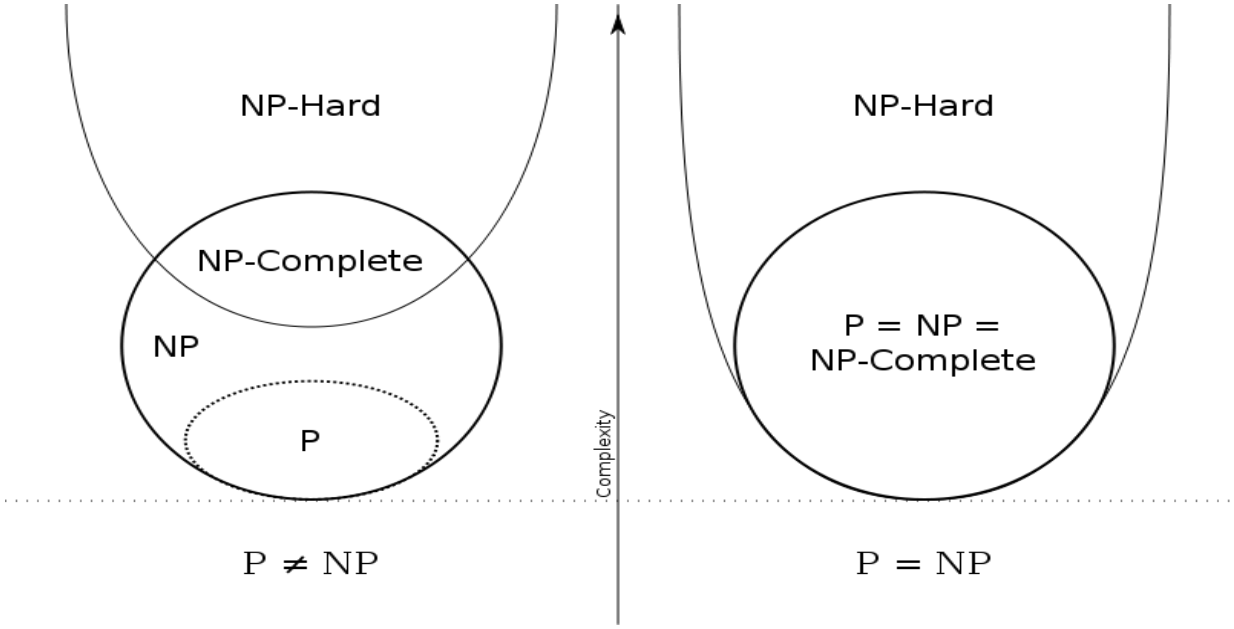
P: The complexity class of decision problems that can be solved on a deterministic Turing machine in polynomial time.

NP: The complexity class of decision problems that can be solved on a non-deterministic Turing machine in polynomial

NP-Hard: Class of problems which are at least as hard as the hardest problems in NP. Problems in NP-hard do not have to be elements of NP, indeed, they may not even be decidable problems.

NP-complete: Class of problems which contains the hardest problems in NP. Each element of NP-complete has to be an element of NP

The diagrammatical representation for classification of complexity when P=NP and when P≠NP.



As you can see from the diagram above, all P problems are NP problems. That is, if it's easy for the computer to solve, it's easy to verify the solution. So the P vs NP problem is just asking if these two problem types are the same, or if they are different, i.e. that there are some problems that are easily verified but not easily solved. According to the work of Cook and Karp showed a large number of combinatorial and logical problems were NP-complete, i.e., as hard as any problem computable in nondeterministic polynomial time. The P=NP question is equivalent to an efficient solution of any of these problems but after the work done by Vinay Deolalikar, proved that the problem P≠NP. It currently appears that P ≠ NP, meaning we have plenty of examples of problems that we can quickly verify potential answers to, but that we can't solve quickly. Let's look at a few examples:

- A traveling salesman wants to visit 100 different cities by driving, starting and ending his trip at home. He has a limited supply of gasoline, so he can only drive a total of 10,000 kilometers. He wants to know if he can visit all of the cities without running out of gasoline.
- A farmer wants to take 100 watermelons of different masses to the market. She needs to pack the watermelons into boxes. Each box can only hold 20 kilograms without breaking. The farmer needs to know if 10 boxes will be enough for her to carry all 100 watermelons to market.

All of these problems share a common characteristic that is the key to understanding the intrigue of P versus NP: In order to solve them you have to try all combinations.

It was alsoshown by Ladner that if P ≠ NP then there exist problems in NP that are neither in P nor NP-complete. Such problems are called NP-intermediate problems. The graph isomorphism problem, the discrete logarithm problem and the integer factorization problem are examples of problems believed to be NP-intermediate. They are some of the very few NP problems not known to be in P or to be NP-complete.

The graph isomorphism problem is the computational problem of determining whether two finite graphs are isomorphic. An important unsolved problem in complexity theory is whether the graph isomorphism problem is in P, NP-complete, or NP-intermediate. The answer is not known, but it is believed that the problem is at least not NP-complete.If graph isomorphism is NP-complete, the polynomial time hierarchy collapses to its second level. Since it is widely believed that the polynomial hierarchy does not collapse to any finite level, it is believed that graph isomorphism is not NP-complete. The best algorithm for this problem, due to Laszlo Babai and Eugene Luks has run time $2O(\sqrt{(n \log(n))})$ for graphs with n vertices.

The integer factorization problem is the computational problem of determining the prime factorization of a given integer. Phrased as a decision problem, it is the problem of deciding whether the input has a factor less than k. No efficient integer factorization algorithm is known, and this fact forms the basis of several modern cryptographic systems, such as the RSA algorithm. The integer factorization

problem is in NP and in co-NP (and even in UP and co-UP[9]). If the problem is NP-complete, the polynomial time hierarchy will collapse to its first level (i.e., NP will equal co-NP). The best known algorithm for integer factorization is the general number field sieve, which takes time O(e(64/9)1/3(n.log 2)1/3(log (n.log 2))2/3) to factor an n-bit integer. However, the best known quantum algorithm for this problem, Shor's algorithm, does run in polynomial time. Unfortunately, this fact doesn't say much about where the problem lies with respect to non-quantum complexity classes.

3. CONCLUSION

Finally, we conclude that the question of whether P equals NP is one of the most important open questions in theoretical computer science because of the wide implications of a solution. If the answer is yes, many important problems can be shown to have more efficient solutions. These include various types of integer programming problems in operations research, many problems in logistics, protein structure prediction in biology, and the ability to find formal proofs of pure mathematics theorems. The P versus NP problem is one of the Millennium Prize Problems proposed by the Clay Mathematics Institute. There is a US$1,000,000 prize for resolving the problem.Solving this problem would have profound effects on computing, and therefore on our society.

REFERENCES

- *Fundamentals of Computer Alogrithms, Ellis Horowiz, SartajSahni, SanguthevarRajasekaran.*
- *S.A.COOK, The complexity of theorem proving procedures, Proceedings of*
- *The 3rd Annual ACM Symposium on Theory of Computing, 151-158, 1971.*
- *History of this letter and its translation from Michael Sipser. "The History and Status of the P versus NP question.*