iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 11 **Issue:** V **Month of publication:** May 2023

DOI: https://doi.org/10.22214/ijraset.2023.52268

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue V May 2023- Available at www.ijraset.com

Analysis of Polynomial Time and Non-Polynomial Time of Algorithms

Vatsal Saxena[1], Kumar Shivam[2], Sonu Yadav[3], Khushbu Trivedi[4]
[1, 2, 3]*Faculty, Computer Engineering Dept. Ajeenkya D.Y Patil University (of Affiliation), Pune, India*
[4]*Student, Bachelors of Technology CE), Ajeenkya D.Y Patil University (of Affiliation), Pune, India*

Abstract: ***The P vs NP problem is one of the most significant open problems in computer science and mathematics. This problem asks whether every problem that can be solved in polynomial time can also be verified in polynomial time. The purpose of this research paper is to explore the P vs NP problem and its relevance in the analysis of algorithms. We will discuss the techniques used to design and analyze algorithms, such as divide-and-conquer, dynamic programming, and greedy algorithms, and their relation to the P vs NP problem. We will also examine some examples of polynomial-time algorithms and NP-hard problems and analyze their time and space complexity, addition to which we will analyze the NP-Complete Problem and finally looking the current scenario of the P & NP and stating its significance.***
Keywords: Polynomial, analysis, algorithm P and NP, NP complete

I. INTRODUCTION

In computer science, the complexity of algorithms is classified into different categories based on their time and space requirements. One of the most significant problems in the field of algorithm analysis is the P vs NP problem. This problem asks whether every problem that can be solved in polynomial time can also be verified in polynomial time. If P = NP, then it would mean that any problem that can be solved in polynomial time can also be verified in polynomial time. This would have significant implications for many fields, including cryptography, optimization, and machine learning

II. LITERATURE REVIEW

The P vs NP problem is one of the most critical open problems in computer science and mathematics. The problem was first introduced in the 1970s by Stephen Cook and Leonid Levin, and since then, it has remained unsolved. The problem is NP-complete, meaning that any NP problem can be reduced to an instance of this problem in polynomial time. The problem has significant implications for cryptography, optimization, and machine learning, among other fields.[1]The problem has also been studied extensively in the context of algorithm analysis. Many algorithms that are currently used in practice, such as sorting algorithms and graph algorithms, are polynomial-time algorithms. However, there are many problems that are known to be NP-hard, meaning that no polynomial-time algorithm is known to exist for them. The problem has also been studied in the context of computational complexity theory, where researchers have developed many techniques for analyzing the time and space complexity of algorithms.

III. METHODOLOGY

We'll look at some exemplifications of NP-hard issues and polynomial- time algorithms to more grasp the P vs NP problem in the environment of algorithm analysis. Algorithms with polynomial- time complexity have a worse- case time complexity that's constrained by a polynomial function of the input size.[3]

A. Polynomial- time Algorithms include the following Exemplifications

1) Bubble sort This straightforward sorting algorithm evaluates neighboring rudiments and barters them if they're in the wrong order as it iteratively moves through the list to be sorted. However, also the worst- case time complexity is O(n2), If there are n rudiments in the list.
2) Double hunt This fashion locates a target value's position within a sorted array. Its worst- case temporal complexity, where n is the array's element count, is O(log n).
3) Dijkstra's algorithm This system determines the shortest route between any two bumps in a graph. Where E is the number of edges and V is the number of vertices in the graph, its worst- case time complexity is O(E V log V).

©IJRASET: All Rights are Reserved

B. NP-hard Problems Include the Following

NP-hard problems are those that are at least as grueling as the toughest issues in the NP(non-deterministic polynomial time) complexity class. These issues are allowed to be unnaturally delicate to break and have no given polynomial- time results[3]

1) Traveling salesperson problem This is a problem in which a salesperson must visit a set of metropolises, each only formerly, and return to the starting megacity, with the thing of minimizing the total distance travelled. Its worst- case time complexity is exponential, making it NP-hard.

2) Backtrack problem This is a problem in which a set of particulars must be packed into a backpack, subject to certain constraints, with the thing of maximizing the total value of the particulars packed. Its worst- case time complexity is also exponential, making it NP-hard.

3) Boolean satisfiability problem In order to discover whether a formula in propositional sense is satiable(i.e., whether there's a setting of verity values for its variables that makes it true), it must be assessed. It's NP-hard because of its worst- case exponential time complexity.

Point A & B discusses the P vs NP problem in algorithm analysis and provides examples of polynomial-time algorithms and NP-hard problems. Polynomial-time algorithms have a worst-case time complexity that can be expressed as a polynomial function of the input size. Examples of polynomial-time algorithms include bubble sort, double hunt, and Dijkstra's algorithm. NP-hard problems are those that are at least as hard as the hardest problems in the NP complexity class and do not have polynomial-time solutions. Examples of NP-hard problems include the traveling salesperson problem, the backpack problem, and the Boolean satisfiability problem. These problems have exponential worst-case time complexity.

C. Designing and Analysing an Algorithms

We'll bandy the ways used to design and dissect algorithms, similar as Divide- and- conquer, dynamic programming, and greedy algorithms and also dissect the time and space complexity of these algorithms, and bandy how they relate to the P vs NP problem, Designing and assaying algorithms involves several ways, including peak- and- conquer, dynamic programming, and greedy algorithms:[2]

1) Divide- and- conquer This fashion involves breaking down a problem into lower subproblems, working each subproblem recursively, and combining the results to break the original problem. exemplifications of algorithms that use peak- and- conquer include merge sort and quicksort. The time complexity of peak- and- conquer algorithms is generally O(n log n) or better.

2) Dynamic programming This fashion involves breaking down a problem into lower subproblems, working each subproblem formerly, and storing the results in a table to avoid spare calculations. exemplifications of algorithms that use dynamic programming include the Fibonacci sequence and the shortest path problem. The time complexity of dynamic programming algorithms is generally O(n2) or better.

3) Greedy algorithms This system makes opinions that are locally optimal at each stage in the expedients of locating a global optimum. The least gauging tree problem and the exertion selection problem are two exemplifications of algorithms that make use of greedy algorithms. Greedy algorithms frequently have an O(nlogn) or lower time complexity.Equations

The time and space complexity of an algorithm is determined by the number of operations and the quantum of memory needed to break a problem of size n. The time complexity is generally expressed in terms of big- O memorandum, which gives an upper bound on the number of operations needed as a function of n. The space complexity is generally expressed in terms of the quantum of memory needed as a function of n.

D. Its Relation with P and NP:

A abecedarian question in computer wisdom is whether all issues that can be vindicated in polynomial time can also be resolved in polynomial time. This is known as the P vs NP dilemma. Polynomial- time algorithms are considered effective, while algorithms with exponential time complexity are considered inefficient. However, it would mean that all problems that can be vindicated in polynomial time can also be answered in polynomial time, which would have significant counteraccusations for cryptography, If P equals NP.[2]

Divide- and- conquer and dynamic programming are known to be effective algorithms that can break a wide range of problems, including some that are NP-hard. Greedy algorithms are effective in some cases, but may not always find the optimal result. While these algorithms are useful for working practical problems, they don't give a definitive answer to the P vs NP problem. The difficulty of working NP-hard problems remains an open question in computer wisdom, and it isn't yet known whether a polynomial- time algorithm exists for these problems.

E. Examine some examples of polynomial-time algorithms and analyze their time and space complexity.

We'll begin by agitating polynomial- time algorithms, similar as sorting algorithms, graph algorithms, and dynamic programming algorithms. We'll examine the ways used to design and dissect these algorithms, similar as divide - and- conquer, dynamic programming, and greedy algorithms. We'll also bandy their time and space complexity, and dissect their performance on colorful problem cases.'

Polynomial- time algorithms are algorithms that have a worst- case time complexity that's bounded by a polynomial function of the input size. These algorithms are considered to be effective and can break numerous practical problems in a reasonable quantum of time. Some exemplifications of polynomial- time algorithms include sorting algorithms, graph algorithms, and dynamic programming algorithms.

1) Sorting Algorithms Sorting algorithms are used to arrange a set of rudiments in a specific order The sorting algorithms bubble sort, insertion sort, selection sort, quicksort, and combine sort are all examples.. These algorithms use colorful ways similar as peak- and- conquer, dynamic programming, and greedy algorithms to sort the rudiments. The time complexity of these algorithms ranges from O(n2) to O(n log n), depending on the algorithm used.[6]

2) Graph Algorithms Graph algorithms are used to assay connections between objects, similar as roads, computer networks, or social networks. exemplifications of graph algorithms include Dijkstra's algorithm for chancing the shortest path, Kruskal's algorithm for chancing the minimal gauging tree, and depth-first hunt and breadth-first hunt for covering a graph. These algorithms use colorful ways similar as dynamic programming, greedy algorithms, and divide- and- conquer to break the problems. The time complexity of these algorithms ranges from O(E log V) to O(V3), where E is the number of edges and V is the number of vertices in the graph.[6]

3) Dynamic Programming Algorithms Dynamic programming algorithms are used to break problems by breaking them down into lower subproblems and working them recursively. The results of the subproblems are stored in a table to avoid spare calculations. exemplifications of dynamic programming algorithms include the Knapsack problem, the longest common subsequence problem, and the Bellman- Ford algorithm for chancing the shortest path in a graph with negative edges. The time complexity of these algorithms ranges from O(nX) to O(n3), where n is the number of rudiments and W is the capacity of the backpack or the length of the strings.[6]

The time and space complexity of these algorithms depend on colorful factors, similar as the input size, the data structures used, and the specific algorithmic ways employed. The time complexity is generally expressed in big- O memorandum, which gives an upper bound on the number of operations needed as a function of the input size. The space complexity is generally expressed in terms of the quantum of memory needed as a function of the input size.

In general, divide - and- conquer algorithms have a time complexity of O(n log n), dynamic programming algorithms have a time complexity of O(n2), and greedy algorithms have a time complexity of O(n log n). still, the factual time and space complexity of an algorithm depend on the specific problem case and the algorithmic fashion used. thus, it's important to dissect the performance of an algorithm on colorful problem cases to determine its practical utility.[9]

In conclusion, polynomial- time algorithms similar as sorting algorithms, graph algorithms, and dynamic programming algorithms are essential tools for working numerous practical problems efficiently. The specific algorithmic fashion used and the problem case play an important part in determining the time and space complexity of the algorithm.

F. Examine some Examples of Non-polynomial-time Algorithms and Analyze their time and Space Complexity.

We'll move on to NP-hard problems, similar as the Traveling salesperson Problem and the Knapsack Problem. We'll bandy the ways used to dissect these problems, similar as reduction, approximation, and heuristics. We'll also dissect the time and space complexity of these algorithms, and bandy their performance on colorful problem cases. [7]

NP-hard problems are a class of decision problems that are at least as hard as the hardest problems in NP. These problems have no given polynomial- time algorithm to break them, and are considered to be computationally intractable. exemplifications of NP-hard problems include the Traveling salesperson Problem(TSP), the Knapsack Problem, and the Boolean Satisfiability Problem:

1) Traveling salesperson Problem(TSP) The TSP is a classic optimization problem in which a salesperson must visit a set of metropolises exactly formerly, returning to the starting megacity, while minimizing the total distance traveled. The TSP is known to be NP-hard, and no polynomial- time algorithm is known to break it optimally. colorful ways similar as heuristics, approximation algorithms, and metaheuristics similar as simulated annealing and inheritable algorithms have been proposed to break the TSP. The time complexity of these algorithms ranges from O(n2 2n) to O(n2 log n).

2) Knapsack Problem The Knapsack Problem is another classic optimization problem in which a backpack of fixed capacity must be filled with a subset of particulars, each with a weight and a value, similar that the total value of the particulars in the backpack is maximized. The Knapsack Problem is also known to be NP-hard, and no polynomial- time algorithm is known to break it optimally. colorful ways similar as dynamic programming, approximation algorithms, and metaheuristics similar as simulated annealing and inheritable algorithms have been proposed to break the Knapsack Problem. The time complexity of these algorithms ranges from O(nW) to O(n2 log n).

3) Boolean Satisfiability Problem(SAT) The SAT problem is a classic decision problem in which a Boolean formula is given, and the task is to determine if there exists an assignment of verity values to the variables that makes the formula true. The SAT problem is known to be NP-hard, and no polynomial- time algorithm is known to break it optimally. colorful ways similar as resolution, Davis- Putnam- Logemann- Loveland(DPLL) algorithm, and countermanding algorithms have been proposed to break the SAT problem. The time complexity of these algorithms ranges from O(2n) to O(n22n).In general, the time and space. complexity of algorithms for NP-hard problems are much advanced than those for polynomial- time problems. In addition, there's no given algorithm that can break these problems optimally in polynomial time.

Reduction is a powerful technique used to analyze NP-hard problems. Reduction entails changing a specific instance of one problem into an other problem while maintaining the original problem's structure. If the second problem is known to be NP-hard, then the first problem is also NP-hard. This technique is often used to prove that a new problem is NP-hard by reducing it to a known NP-hard problem.

In conclusion, NP-hard problems such as the TSP, Knapsack Problem, and SAT Problem are computationally intractable, and no polynomial-time algorithm is known to solve them optimally. Various approximation algorithms and heuristics have been proposed to obtain approximate solutions to these problems. Reduction is a powerful technique used to analyze NP-hard problems and is often used to prove that a new problem is NP-hard by reducing it to a known NP-hard problem

G. The Concept of NP-Complete and its analysis.

The concept of NP-completeness, which is used to classify problems that are at least as hard as the hardest problems in NP. We will examine the techniques used to prove NP-completeness, such as reduction and the Cook-Levin theorem. We will also discuss the implications of NP-completeness for the P vs NP problem.[7]

NP-completeness is a concept used to classify problems that are at least as hard as the hardest problems in NP. A problem is said to be NP-complete if it is in NP and any problem in NP can be reduced to it in polynomial time. In other words, an NP-complete problem is one that is "at least as hard" as any other problem in NP.[7]

The concept of NP-completeness was introduced by Stephen Cook and Leonid Levin in the 1970s. They showed that the Boolean satisfiability problem (SAT) is NP-complete, which means that any problem in NP can be reduced to SAT in polynomial time. This result was later extended to other problems, including the traveling salesman problem (TSP), the knapsack problem, and many others.[9]

The implications of NP-completeness for the P vs NP problem are significant. If there exists a polynomial-time algorithm for any NP-complete problem, then there exists a polynomial-time algorithm for all problems in NP. This is because any problem in NP can be reduced to an NP-complete problem in polynomial time, and then solved using the polynomial-time algorithm for the NP-complete problem. Therefore, proving that an NP-complete problem is not in P would prove that $P \neq NP$.[8]

In conclusion, NP-completeness is a concept used to classify problems that are at least as hard as the hardest problems in NP. The proof of NP-completeness typically involves reduction, which is the process of transforming one problem into another problem in a way that preserves the solution. The Cook-Levin theorem is an important result in the theory of NP-completeness, and shows that the Boolean satisfiability problem is NP-complete. The implications of NP-completeness for the P vs NP problem are significant, as proving that an NP-complete problem is not in P would prove that $P \neq NP$.

Finally, we will examine some recent developments in the field of computational complexity theory, such as the development of quantum computing and its potential implications for the P vs NP problem. We will also discuss the limitations of current techniques for analyzing algorithms and the need for new approaches to solve the P vs NP problem

In recent years, one of the most significant developments in the field of computational complexity theory has been the development of quantum computing. Quantum computing is a new model of computation that uses quantum mechanics to manipulate information. Unlike classical computers, which use bits to represent information, quantum computers use qubits, which can be in a superposition of states, allowing them to perform certain calculations much faster than classical computers.

H. Based Upon The Analysis And The Researches Made Till Date

The potential implications of quantum computing for the P vs NP problem are significant. While it is currently unknown whether P = NP or P ≠ NP, it is known that there are some problems that can be solved efficiently by a quantum computer that cannot be solved efficiently by a classical computer. This means that quantum computing may be able to solve some problems in NP much faster than classical computers, which could have important implications for cryptography and other fields.

However, the limitations of current techniques for analyzing algorithms remain a significant challenge in the field of computational complexity theory. While techniques such as reduction and approximation have been successfully used to analyze many problems, there are still many problems for which these techniques are not effective. In addition, there are some problems for which no efficient algorithm is known, but for which it is not known whether an efficient algorithm exists or not. These problems are known as open problems, and solving them would require new approaches and techniques that have not yet been developed.[10]

To address these challenges, researchers are exploring new approaches to computational complexity theory, such as quantum complexity theory and circuit complexity theory. These approaches aim to understand the computational power of different types of models of computation, such as quantum computers and circuits. By developing new techniques and approaches for analyzing algorithms, researchers hope to make progress on some of the most challenging problems in computational complexity theory, including the P vs NP problem.

IV. EXAMPLE STATING THE PROOF OF NP

The Travelling Salesperson Problem (TSP) and the Hamiltonian Circuit Problem are both NP-hard problems, which means that they are at least as hard as the hardest problems in the NP complexity class.[5]

The TSP asks for the shortest possible route that visits a set of cities and returns to the starting city, while the Hamiltonian Circuit Problem asks for a cycle that visits every vertex of a given graph exactly once. [5] Both problems are optimization problems and are known to be NP-hard, meaning that there is no known polynomial-time algorithm that can solve them.

If the TSP and Hamiltonian Circuit Problems could be solved in polynomial time, it would mean that P equals NP, and all problems in the NP class could be solved efficiently. However, no polynomial-time algorithm has been discovered for these problems yet, and they remain open questions in the field of computer science.

Many algorithms have been developed for solving the TSP and the Hamiltonian Circuit Problem, such as the brute force approach, dynamic programming, and approximation algorithms. While these algorithms may provide good solutions for small instances of the problems, they become computationally infeasible for larger instances. [5]

In conclusion, both the TSP and Hamiltonian Circuit Problem are NP-hard problems, and their resolution in polynomial time would have significant implications for the field of computer science. However, no such algorithm has been discovered yet, and they remain challenging optimization problems for researchers and practitioners alike

V. CONCLUSION

The P vs NP problem is one of the most significant open problems in computer science and mathematics. It has significant implications for many fields, including cryptography, optimization, and machine learning. In the context of algorithm analysis, it has led to the development of many techniques for designing and analyzing algorithms, as well as the classification of problems into different complexity classes. While there has been much progress in this area, the problem remains unsolved, and new approaches are needed to make further progress.

The P and NP problem has significant implications in computer science. The P problem is a class of problems that can be solved efficiently. The NP problem is a class of problems that are difficult to solve efficiently. The P and NP problem has significant implications for cryptography, optimization, and artificial intelligence. Many important problems in computer science are in the NP problem class, and finding efficient algorithms to solve these problems is critical.

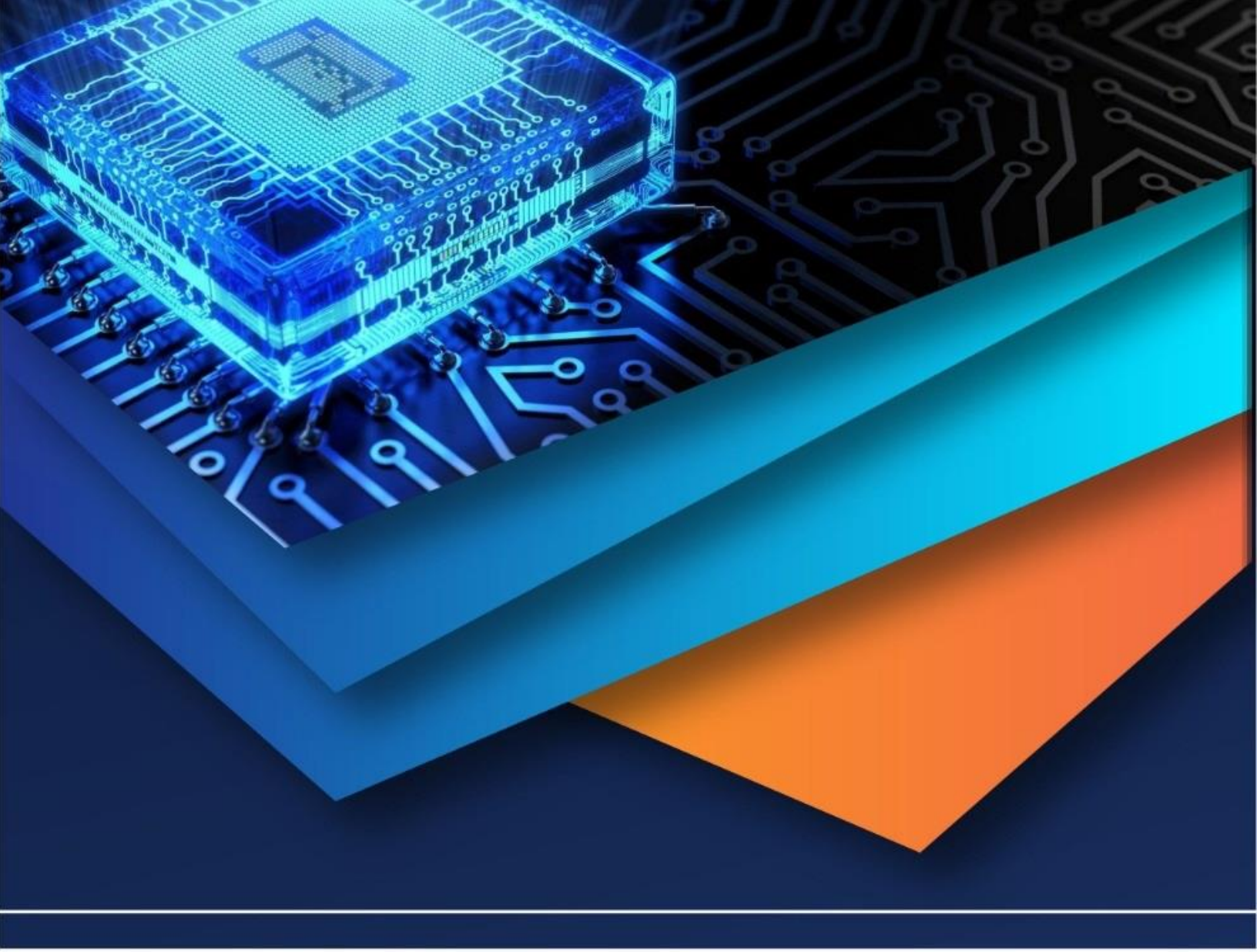
VI. FUTURE WORK

Future research in this area could focus on developing new techniques for analyzing algorithms and solving the P vs NP problem. One area of interest is quantum computing, which has the potential to solve certain problems in polynomial time that are currently believed to be NP-hard. Another area of interest is the development of new mathematical techniques for analyzing the complexity of algorithms.[10] Ultimately, the solution to the P vs NP problem could have significant implications for many fields, and continued research in this area is critical

REFERENCES

[1] RICHARD E. LADNER"On the Structure of Polynomial Time Reducibility"
[2] Garey, M. R., & Johnson, D. S. (1979). Computers and intractability: A guide to the theory of NP-completeness. W. H. Freeman.
[3] Papadimitriou, C. H. (1994). Computational complexity. Addison-Wesley.
[4] Lance Fortnow,"The status of the P versus NP problem",Communications of the ACMVolume 52Issue 9, September 2009,pp 78–86
[5] G. Cornuéjols, D. Naddef & W. R. PulleyblankHalin graphs and the travelling salesman problem, 287–294 (1983), October 1983
[6] Michael Sipser, "The history and status of the P versus NP question", STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of Computing ,July 1992 ,Pages 603–618
[7] M. R. Garey, D. S. Johnson, L. Stockmeyer , "Some simplified NP-complete problems"STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing, April 1974,Pages 47–63
[8] NP-complete Problems and What To Do About Them (article): https://www.cs.cmu.edu/~15251/previous-semesters/2013-spring/resources/np-complete-problems.pdf
[9] NP-Completeness (lecture notes): http://www.cs.cmu.edu/afs/cs/academic/class/15451-s14/LectureNotes/lecture5.pdf
[10] Arora, S., & Barak, B. (2009). Computational complexity: A modern approach. Cambridge University Press.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)