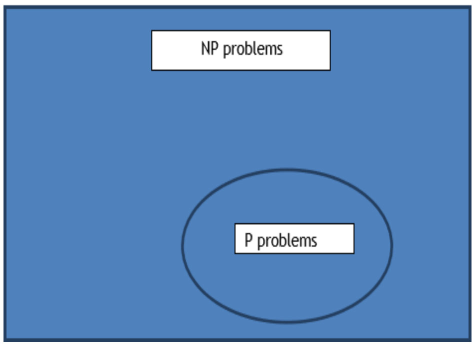
P vs np is An Exponential Problem, So Solution Can't Be Proved In Polynomial Time

Ravi Raja[#1], Piyush sharma[*2], Abhijeet singh[#3]
[1]*Scholar* , [2]*professor* , [3]*Scholar*
computer science department Aiet,jaipur india

Abstract
p vs np is a problem that has been there for centuries as a part of complexity theory. We are not certain about if p is equal or not equal to np. P is a set of problems that can be solved and verified in polynomial time and NP is the set of problems that can verified in polynomial time, but we are not certain whether we will find there solution in polynomial time using a deterministic algorithm or not, although we can find the solution using a lucky algorithm . If p is not equal to np then we can be in peace knowing that all our decrypted data is safe. And if p is equal to np then none of our encrypted data is safe because in case of p is equal to np, anything that can be verified in polynomial time, can be solved in polynomial time, like password, encrypted data, account number, access code etc. In this paper we will prove that presently it is not possible to prove that p is equal to np and nor it is possible to prove that p is not equal to np, because p vs np is itself an exponential problem or so called exp problem.

Keywords — *complexity theory, Polynomial time problem, non-deterministic polynomial time problem, lucky algorithm, exponential time problem*

I. INTRODUCTION



.

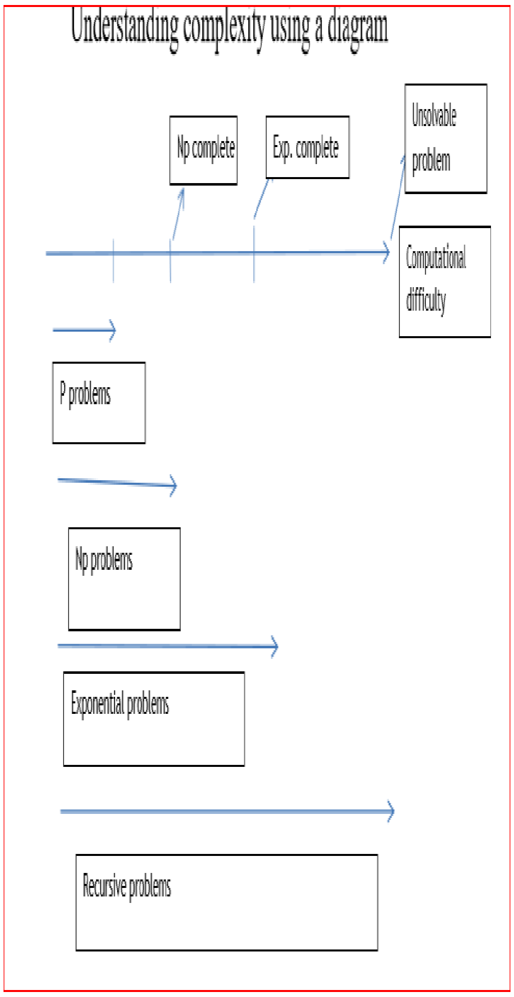
The P vs NP problem is one of the major unsolved problems in today's world and one of the most important one too. As show in the figure let's assume there are only two sets of problems, excluding all other types of problems. The two problems are problem that can be solved in polynomial time known as P problems and problems that are said to be solved in non-deterministic time, the NP problems. Now the question that we have been asking ourselves from decades is that is P and NP are the same thing or P and NP are different. P is set of problems that can be solved as well as verified in polynomial time, whereas NP is set of problems that can verified in polynomial time, but we don't have an efficient algorithm to solve that problem in polynomial time

IF P IS EQUAL TO NP

If p is equal to np ,this question is very important to us, because all our modern system, by all here I mean most of the modern system that we use today uses encryptions to keep there and user's data safe which is done using taking the fact in consideration that the solution will take exponential time to be discovered. And that means centuries, we know that if we know the correct decryption key than we can verify it in very less or polynomial time. And so if it's proven that something that can verified in polynomial time can also be solved in polynomial time using deterministic algorithm and we discover a way to do that, then we will easily be able to decrypt data and also find hack into any encrypted server easily, we will also be able to find the password of someone else's account easily. In this way nothing will be save and there will be no privacy left on internet and bit coins will be affected. And we have to find a new way to decrypt our data.

If P is not equal to NP

If p=! Np then it's a good and bad news, good news in the sense all our data is safe as a problem that can be verified in polynomial time doesn't assures that it will be solved in polynomial time using a deterministic algorithm. So our password issafe, until and unless we have a easy password which can be guessed using combination of common words used by us.Or if we use someone's name as our password. In other words one can guess our password but can't use a deterministic algorithm to solve it. Same with all other encrypted data online, we one can use a lucky guess or so called a lucky algorithm and try but can't solve it using a deterministic algorithm in polynomial time.



In this diagram the longest arrow represent the computational difficulty and first there the p problems which are said to be easiest one and can be solved in polynomial time, then there are np problem, so called non-deterministic polynomial time problem. And after that there is exponential time problems which need exponential time to be solved , an example is chess and then there is recursive problem, hardest of all. And after recursive problem there

Computational difficulty

Exp. complete Np complete

Unsolvable problem

Recursive problems

Exponential problems

Np problems

P problems

exist one more class of problem that are said to be unsolvable in finite time, an example is halting problem. And in NP class we have problems such as Tetris, jigsaw puzzle which can be solved using lucky guesses in polynomial time.

P vs NP is an exponential problem

Let's make an attempt to solve and prove that p=!np using prove by contradiction in order to understand why p vs np is un-provable in polynomial time. Let's assume first p=np Then, let say there is a problem in np and let's denote it by X. And let denote polynomial time as P, verification of solution as V, solution of problem as S.

X→S [POLYNOMIAL TIME]

S→V [POLYNOMIAL TIME]

Now let' s study the procedure how we will we be able to prove P is not equal to NP. Let's say there is a problem of NP and let's denote it by X and for now it can only be solved using a lucky algorithm in polynomial time and there is no efficient algorithm to solve it in polynomial time deterministically. So for proving that a NP problem is in P and can be solved in polynomial time , we have to first find out how many ways are there to solve a problem and then check for each method of solving a problem till we find a method that solve it in polynomial time. Because if we are making an attempt to prove that the P is equal to NP then, then we have to show that there are possible methods that can solve the NP problem in polynomial time and for doing this we have to first find out how many ways are there to solve a particular problem and once we have calculate the exact number of methods for solving a problem then we have to check each method and find each methods time complexity and see, if it solves the problem in polynomial time or not. We have to do so till we find a methods that solve the problem in polynomial time or till we have tried all the methods. And if we try all the methods and didn't find any method that is efficient enough to solve the problem in polynomial time then we can conclude that for that P is not equal to NP.Not let's assume that there is such a model that can take a problem as input and output all the possible ways of solving a problem. And for the sake of simplicity let's call this model as PWTSP [possible ways to solve a problem].This model is assumed hypothetical model to understand how P vs NP is an exponential problem, as this model plays a very important role in proving P=NP or P=!NP.

II. DEFINATION OF PWTSP

PWTSP is a hypothetical model with exponential time complexity for solving a problem because , PWTSP is in exponential class , as it's answer can nor found in polynomial time nor can be verified in polynomial time. It'sanswer cannot verified in polynomial time because we can't be exactly sure that there are only n number of ways for solving a problem and it could not be found in polynomial time because as we start from first methods and move towards n number of methods the time complexity keeps on increasing. And we cannot ever be sure if there is only n number of ways for solving a problem, so verifying PWTSP is not easy. PWTSP is assumed because it plays a very important role in the proof of P vs NP problem. But as given below that PWTSP model has exponential time complexity for finding a solution and then checking for the solutions by applying it to the problem, so presently we can't design a system that can run this model and give us an output in polynomial time, but in future maybe with enhancement in quantum computing we will be able to run this model in polynomial time, along with other exponential problems, as we will have very high speed processing pf the power. PWTSP is a model that takes a problem as input. Let say there is a problem Z, so PWTSP will take that problem as input and output all the possible ways of solving that problem. Let's say there is n number of ways for solving the problem. By n it means that there are n different ways of solving the problem and only n number of ways not more or less than that. But for some problems the value of n can be zero. n=0 The value of number of methods to solve a problem, denoted by n will be zero when the problem is unsolvable. And example can be halting problem. The value of n can also be infinity when there are infinite many ways to solve a problem. Now let's find the complexity of PWTSP system. Let say for problem Z, Set A is all possible ways in which it can be solved, with there are total n number of ways

A= {A1, A2, A3, A4, A5 …An}

Now as there is n number of ways, then if we check for each ways as if it is the efficient way or not that solve the problem in polynomial time then. For each method , let say the complexity is k then the total complexity for finding all the possible methods and checking for them will be n to the power k (2^n). As 2^n is the time complexity for PWTSP model. As power k (2^n) is exponential time then finding all the possible outcomes and checking them will take exponential time, and so does proving p=np. Hence, we can conclude that as solving the np problem and proving that there exist deterministic algorithms that can solve the problem in deterministic time will take exponential time. Therefore p=np itself can't prove in polynomial time will take exponential time. And same as P= NP, if we want to prove p is not equal to np (p=! np) then also we have to use the model to first find our all possible ways and then we have to check for all possible methods which will take exponential time.

Important: As PWTSP is a hypothetical model, so when we design it in real world then instead of polynomial time, it can have higher time complexity. In that case the time complexity for solving P vs NP also may increase. But in any case , it is at least polynomial time and even with polynomial time model, it will take exponential time to prove P vs NP problem.

IF P=NP then proof will be faster as compared to when P=!NP

As for proving P=NP we have to first use the model to find out all the possible ways for finding the solutions, then let say for using model against a problem Z, we get n number of methods to solve Z.

Case 1:P=NP

When proving P=NP, we have to start the iteration from initial from first methods and then move to the nth. If we find in-between 1 to n any methods that solve the problem in polynomial time we can break the iteration and we can prove that P=NP using that method, we don't have to check all methods till n.

Case 2:P=!NP

When proving P=!NP, we have to start the iteration from initial from first method and then move to the nth. As we will not find any methods that solve Z in polynomial time, so we have to keep looking for an efficient algorithm till n and at the end when after reaching n, still we don't find one. We can conclude that there is so efficient algorithm for solving Z in polynomial time so, p=!np.

III.CONCLUSION

P vs NP is a problem that is one of the most important problems in today's world, and many attempts are made to solve this problem. But in order to solve this problem we have to make a model like PWTSP so that we can solve real world NP problems and prove that either they are in P or not. But as we have seen PWTSP requires exponential time to check all . So one possible way can be a system that can solve exponential time problems in polynomial time, maybe we can use fast quantum computers in future for doing this. But one thing is clear, that proving p vs np is not possible in polynomial time for all problems on np, But even if we prove p vs np problem for a problem of np then we can use reduction techniques to prove it for others , but this can be correctly done using the model which will take exponential time to verify all the possible methods.

REFERENCE

[1] M. Agrawal, N. Kayal, and N. Saxena, Primes is in P, Ann. Math. 160 (2004), 781–793.

[2] N. Alon and R.B. Boppana, The monotone circuit complexity of boolean functions, Combinatorica (1987), 1–22.

[3] T. Baker, J. Gill, and R. Solovay, Relativizations of the P =? NP question, SICOMP: SIAM Journal on Computing, 1975.

[4] L. Blum, F. Cucker, M. Shub, and S. Smale, Complexity and Real Computation, Springer- 19 Verlag, New York, 1998.

[5] M. Blum and R. Impagliazzo, Generic oracles and oracle classes, in Proceedings of the 28th Annual Symposium on Foundations of Computer Science, A.K. Chandra, ed., IEEE Computer Society Press, Los Angeles, 1987, 118–126. THE P VERSUS NP PROBLEM

[6] R.B. Boppana and M. Sipser, The complexity of finite functions, Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity, J. Van Leeuwen, ed., Elsevier and The MIT Press, Cambridge, MA, 1990, 759–804.

[7] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, Introduction to Algorithms, 2nd edition, 28 McGraw Hill, New York, 2001.

[8] **A.** Cobham, The intrinsic computational difficulty of functions, in Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science, Y. Bar-Hille, ed., Elsevier/North-Holland, Amsterdam, 1964, 24–30.

[9] S. Cook, The complexity of theorem-proving procedures, in Conference Record of Third Annual 33 ACM Symposium on Theory of Computing, ACM, New York, 1971, 151–158.

[10] S. Cook, Computational complexity of higher type functions, in Proceedings of the International Congress of Mathematicians, Kyoto, Japan, Springer-Verlag, Berlin, 1991, 55–69.