

P versus NP Problem



Safi Khan,
MS in Computer Sciences,
[Software Engineering]

Theory of computation,
Virtual University of Pakistan
Dated: 01-05-2016

Abstract:

There are still unsolvable problems in the computer science history known as P versus NP. The phenomena is that, is the every given problem solve quickly by a computer program once it verified the solution of that given problem. The matter posed in 1956 by Kurt Gödel while asking to John von Neumann in a written letter. The question was arising is that, "whether a certain NP-complete problem could be solved in quadratic or linear time" [2]. In 1971 Stephen Cook one of the most senior computer scientist was introduced very first time the term, "P versus NP" in his paper titled "The complexity of theorem proving procedure" [3]. That problem is considered as most important in the computer science field [4], and prize for that solution is in US\$1,000,000 announced by the Clay Mathematics Institute.

The problem solved quickly by a computer program only when it solves the given problem in polynomial time. That's mean that there should be an algorithm exists that solve the given problem in polynomial time. If that kind of an algorithm exists that solve the problem in polynomial time then that given problem is fall in "class P" or some time just "P". However there are some problems that do not fall in "class P" or in other words, not solve "quickly" or "not in polynomial time" then it may be possible that the given answer by showing information, one it can verify the answer "quickly" or "polynomial time". That kind of problems falls in "class NP" because only the answer of these questions can be verified in polynomial time. Where, NP stands for "nondeterministic polynomial time".

The subset sum problem can be verified easily but it is very difficult to compute its answer. So that problem can be checkable quickly (NP) but not quickly solvable (P). For example, there are a set of some integers and we want to know is there any of the subset which has the sum of 0? In practically, if there is a set $\{-5, -10, -15, -20, -25, 50, 75, 100\}$ and we quickly verify that there are two subset with the sum 0. First one is $\{-5, -10, -15, -20, 50\}$, and the second one is $\{-5, -10, -15, -20, -25, 75\}$. When we add the integer of both these sets it will return the answer is 0. Both are quickly verified by adding some integers but there are not any known algorithms that can find these kinds of subsets in polynomial time. But at the same time there may be an exponential time algorithm can be exists if and only if when $P = NP$.

A problem that can be verified in polynomial time can also solved in polynomial time if $P = NP$. If it is not then it termed as $P \neq NP$, means that there are a problem that is in NP and it is easy to verify and hard to compute. So that the given problem (subset sum problem) cannot be solved in polynomial time but its answer can be verified in polynomial time. So the P versus NP is an important problem in history of computer science as well as in computational theory, so the proof may be profound implication for cryptography, mathematics, artificial intelligence, algorithm research, multimedia processing, economics, game theory, philosophy and many other fields.

1 – Introduction:

Computational theory defines the complexity relation between the P and NP classes. The resources those are required to solve the computational problem are deals within theory of computation. These resources are considered as time and space. Time specifies the steps to solve a particular problem and space deals with the memory required to solve a particular problem.

A computer model is required for such kind of analysis which analyzed the required time. These types of models are deterministic and sequential. The deterministic models are those are taking values as input and a present sate and perform only one possible action that a given computer might take. On the other hand, the sequential models are those that can be perform a single action one after the other.

A deterministic sequential machine solves those decision problems that falls in class P, in polynomial size of the input. Whereas the class NP verified the solution of given decision problems in polynomial time, or in other words the solution of those problems in be found in the polynomial time on a non-deterministic machine [5]. Hence the $P \subseteq NP$ and the computational theory define the relationship between these two classes. Is $P = NP$?

A poll of researcher was conduct in 2002 and also in 2012 to answer that question, in which many researchers was participates and replied various answer [6]. The table in figure 1 below shows the actual figures. The researcher those are not certain about the possibility of prove or disprove, believe that the question may be independent of currently accepted axioms [7].

Year	2002	2012
No. of Researchers	100	151
Yes	9	12 (9%)
No	61	126 (83%)
Unsure	22	8 (5%)
Impossible to prove or disprove	8	5 (3%)

Figure 1: The pool of researcher in 2002 and 2012

2 – NP – Complete:

The concept of NP-Complete is very useful in order to find the solution of $P = NP$. All the NP-problems which can be reduced in the polynomial time fall in the set of NP-complete problems, because the solution of these problems can be verified in polynomial time. In the set of NP-complete problem each problem can be easily transform in any other problem, hence all these problems are equally tough.

On the other hand, NP-hard problems are equally hard then NP problems, and can be reduced in polynomial time by NP problems. There is no need to equate the NP-hard problems with NP because there is no need to verify them in polynomial time. Following figure 2 shows the relationship between the P, NP, NP-complete, and NP-hard problems.

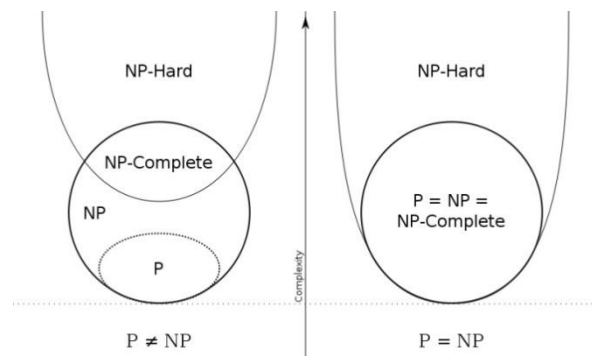


Figure 2: Euler diagram for P, NP, NP-complete, and NP-hard set of problems

By the Cook-Levin theorem, the entire Boolean satisfiability problem falls in the NP-complete class. So if a problem in NP then it can be mechanically transformed into Boolean satisfiability problem in the polynomial time. Hence, these Boolean satisfiability problems are one of the NP-complete problems. The $P = NP$ if and only if the NP-complete problem is in P, but there are a large number of problems that are in NP-complete, and there are not any known solution or algorithm for them.

If we examine clearly only on the definition then we do not found obviously that the problems of NP-complete are exist. However a NP-complete problem can be formulated trivially as: if we give a description of a Turing machine M that guaranteed to be halt the Turing machine in polynomial time, so the question is that, is there any polynomial-size input exists that M will accept[8]? Obviously, this is in class NP because on given an input one it can simply check via a simulation that M accepts the input. It is also in NP-complete due to the verifier of an instance for a problem that is in NP, and one it can be encoded with machine M in polynomial-time as an input and verified the solution. So that instance is based on valid input.

The Boolean satisfiability problem is the very first NP-complete problem to be proven. Actually this is the Cook-Levin theorem because it's proof that the satisfiability is the NP-complete. The technical details which it

contains about the Turing machines can be relate to the NP definition. In that way if we reduce the other problem in Boolean satisfiability problem than those problems are also in NP-complete, i.e. subset sum problem. In that way a large class of related problems can be reduce to one another and treated as same problem. The following figure 3 shows the complexity relationship between these classes. That figure shows that $P \neq NP$, the problem within class NP are clearly outside both the classes of P and NP-complete, the Lander's theorem base on that assumption [1].

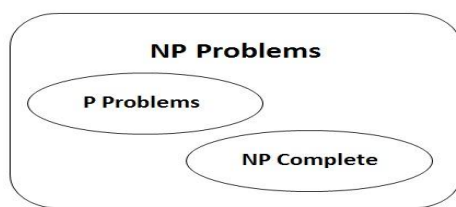


Figure 3: complexity relationship between the classes.

3 – Harder problems:

Yet it is unknown that whether $P = NP$, but the problems outside of P are known. There are a small number of problems that are in EXPTIME-complete. These kinds of problems are not operating on the normal input; its input requires the computational description. At the same time these problems are treated as $P \neq EXPTIME$, hence they are outside of class P, because more than polynomial time is required to solve them. In fact, if we examine the theorem of time hierarchy then it can found that these problems cannot be solved in less than exponential time significantly. Board games, i.e. chess (on the $N \times N$ board) [9] are examples to find a perfect strategy [10].

On the other hand, the problems in Presburger arithmetic statements requires more time to finding the truth. Rabin and Fischer found in 1974 that the decision algorithm to known the truth of Presburger statements requires $2^{2^{cn}}$, where c is some constant and the length of the Presburger statement denoted by the n . Hence, there are more than exponential run time is

required for known problem, and this run time can be more critical in case of more difficult problems, i.e. undecidable problems which are known as halting problems. The algorithms cannot be solved these problems completely because there are a particular input of each algorithm that produce the wrong answer or do not produce right answer or it do not produce the conclusive answer. Some time it may be run forever and do not producing any right answer at all.

4 – Problems in NP not known to be in P or NP-complete:

Lander believes that if $P \neq NP$ then there are some problems exists in NP that many not are both in P and NP-complete [1]. These problems known as the intermediate problems, the example of NP-intermediate problems are; discrete logarithm problems, isomorphism problems and integer factorization problems. However there are some problems in NP that are still not known if it falls in class P or NP-complete.

The graph isomorphism problem which is known as computational problem basically are used to determination weather two graphs are isomorphic or not. That is the unsolved problem in the theory of complexity. That is no known whether the graph isomorphic problems in P, NP-complete, or NP-intermediate. The scientist sill not believes but they sure that the problem in not in class NP-complete [11]. The hierarchy of polynomial time is collapses to the second level if the class NP-complete contains the graph isomorphic problem [12] [13]. Sine at any finite level the hierarchy of polynomial does not collapse and at the same time the class NP-complete do not contain the graph isomorphism problems. For the graph that have n vertices, Laszlo Babi and Eugene Lucks denotes the run time $2^{O(n \log(n))}$ as an best algorithm.

Another computational problem that determines the prime factorization of a given integer is known as integer factorization

problem. Like the decision problem this problem also is decided that if input has factor less than k . Like RSA algorithm, that factor is based on sever modern cryptographic system. Although, still yet there are not any best known integer factorization algorithm is exists. The class NP and co-NP contains the integer factorization problem, sometime known as UP and co-UP [14]. The general number field sieve is known as best integer factorization algorithm and it takes the expected time to factor an n -bit integer.

$$O(\exp((64n/9 \log(2))^{1/3} (\log(n \log(2)))^{2/3}))$$

Shor's algorithm is also known as best quantum algorithm to solve that problem, does not run in polynomial time. With respect to non-quantum complexity classes that problem does not sure about the problem where it is lies.

5 – Does P mean "easy"?

While reading the above discussion it is assumed that the problems those are fall in class P mean "easy", and the problem those are not fall in class P means as a "hard", so these assumptions known as Cobham's thesis. Some people or literatures known these assumptions accurate reasonably in complexity theory but some time it is assume that it has some caveats. First of all in practical that is not true always because a polynomial algorithm may have theoretically a constant factors or exponents which are extremely large and thus most of the time it rendered as impractical. However, in practice there may still some effective approaches that can tackle the problem if it is in NP-complete and even if $P \neq NP$. For NP-complete problems there are still many algorithms that can be solve the many of real-world problem in the reasonable amount of time; such as traveling salesman problem, Boolean satisfiability problem and knapsack problem. For such kind of algorithms the empirical average-case complexity (which is based on time vs. problem size) can be very low surprisingly. For instance the linear programming the simplex algorithm that works

well in practice; however it have worst-case time complexity exponentially but it known best polynomial-time algorithms [16].

Second, some computation problems that do not conforms the model of the Turing machine in which the P and NP are defined; these problems are as randomized algorithms as well as quantum computation.

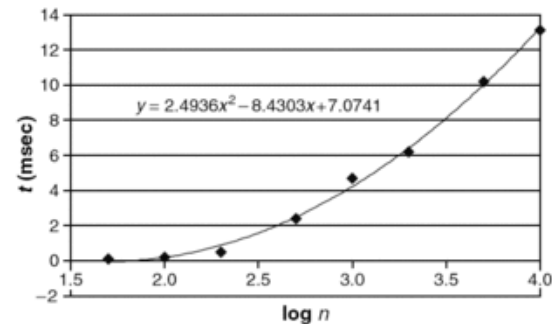


Figure 4: Time vs. problem size for knapsack problem.

In the figure 4, the graph shows the 100 instances in ms which are using a 933 MHz Pentium III, which is as a state-of-art specialized algorithm of time vs. problem size for knapsack problems. The empirical algorithmic complexity which is suggests as quadratic fit for instances with 50–10,000 variables is $O((\log(n))^2)$. [15]

6 – Reasons to believe $P \neq NP$:

Many computer scientists, according to the polls held [6][17], believes the $P \neq NP$. The reason is that still now no one can find the polynomial time algorithm, after a decades studying, i.e. studying more than three thousand NP-complete problems. These kinds of algorithms were difficult to search before the NP-complete concept. $NP = co-NP$ and $P = PH$ were believed as false, so the $P = NP$ result startling then. The problems exist that are hard to solve but the solutions of these problems are easy for the verification also argued intuitively to matches the real-world experience [18].

According to the Scott Aaronson, MIT:

"If $P = NP$, then the world would be a profoundly different place than we usually assume it to be.

There would be no special value in "creative leaps," no fundamental gap between solving a problem and recognizing the solution once it's found."

At the same time some of the other researchers believe that some researchers are overconfident about the $P \neq NP$ believing and they should try to explore the proof for $P = NP$. Following statements were made in 2002 as [6]:

According to the Moshe Y. Vardi, Rice University:

"The main argument in favor of $P \neq NP$ is the total lack of fundamental progress in the area of exhaustive search. This is, in my opinion, a very weak argument. The space of algorithms is very large and we are only at the beginning of its exploration. [...] The resolution of Fermat's Last Theorem also shows that very simple questions may be settled only by very deep theories."

According to the Anil Nerode, Cornell University:

"Being attached to a speculation is not a good guide to research planning. One should always try both directions of every problem. Prejudice has caused famous mathematicians to fail to solve famous problems whose solution was opposite to their expectations, even though they had developed all the methods required."

7 – Consequences of solution:

There are some factors that may attract the consequences of answer; it may be either that enormously the resolution direction would be the advance theory or it may have huge practical consequences as well.

7.1 – $P = NP$:

If we can solve important problems that belong the NP then in that case the proof of $P = NP$ is practically best consequences. If somehow the polynomial bounds are very large in practice that are not efficient or the proof is not to be constructive then in that case the proof possibly

no leads to an efficient methods. So both the positive and negative consequences arise due to these NP-complete problems which are fundamental in nature in many of fields.

Cryptography which relies some of the certain problem being difficult, 3-SAT which have constructive as well as efficient algorithm break down the cryptosystems such as;

- Public-key cryptography: that is the foundation of most important security application systems [19], i.e. financial transaction on internet.
- Symmetric ciphers: i.e. AES or 3DES uses for communication data encryption.
- Cryptographic hashing: it is a one way function, i.e. find pre-image hashes to a given value [21], sometime difficult to use due to exponential time. Via reduction to SAT then finding pre-image can done in polynomial time if and only if $P = NP$ [22].

There are some other positive consequences that may be enormous and rendering intractable problems mathematically. For example a lot of problems in operation research are NP-complete, i.e. travelling salesman problem, integer programming. Efficient solution for these kinds of problems is the logistically implicated. A lot of other problems like protein structure predication are also in NP-complete [23]. If the solution of this problem found efficiently than one it could be a considerable advancement in the life of biotechnology as well as science.

But at the same time such kind of changes may less significant than the methods that evolutionally efficient to solve the NP-complete problems that would be cause in mathematics itself. In the computational complexity the Gödel thoughts the mechanical methods than can be solved any problem would be considerable revolution mathematics [24] [25].

The greatest important consequences where a machine rely is $\phi(n) \sim k \cdot n$ (or even $\sim k \cdot n^2$). That's mean it is the undecidability of

Entscheidungs problem, as the mathematician's mental work concern only on Yes or No questions and this is replaced completely by a machine. If we choose the natural number n is so large, then one possibility is that the machine does not provide the result that is it makes no sense to think more about the problem.

Stephen Cook says:

"It would transform mathematics by allowing a computer to find a formal proof of any theorem which has a proof of a reasonable length, since formal proofs can easily be recognized in polynomial time. Example problems may well include all of the CMI prize problems." [26]

Mathematicians as well as the researchers spend their life and careers to prove the theorems. Some of the proofs took centuries to resolves, i.e. Fermat's Last Theorem took near about three centuries to prove. The reasonable size of the problem is main concern while guarantee to proof a theorem.

A researcher Donald Knuth believes that the $P = NP$ but is reserved about the impact of the possible proof [27], "I don't believe that the equality $P = NP$ will turn out to be helpful even if it is proved, because such a proof will almost surely be non-constructive".

7.2 – $P \neq NP$:

The computational benefits that $P = NP$ provides is reduced by the proof of $P \neq NP$, but at the same time it provide the computational advancement in the complexity theory as well as guidance for future research. Formally it shows that we cannot solve the common problems efficiently. At that point the partial solution becomes the attractive by the researchers. So the believe-ness of $P \neq NP$ by a lot of researchers is the result to focusing to other similar problems [28].

So the focusing the average-case complexity some of the harder problems in the class NP is due to that believe-ness of $P \neq NP$. SAT requires the exponential time but on the other hand the

other instances that are selected randomly can be solved efficiently. To deal with the question of average-case complexity the Russell Impagliazzo describes the five hypothetical worlds that lead to the different possible resolutions [29]. In this regard three ranges can be defined as Algorithmica, Cryptomania, Heuristica;

Algorithmica: In that case the $P = NP$ and SAT problems can be efficiently solved by all its instances.

Cryptomania: In that case $P \neq NP$ and hard instances of the problems that are generated outside P is easy. Different possible distributions of difficulty are reflecting the three possible intermediate all over the instances of NP -hard problems.

Heuristica: The "world" where $P \neq NP$ but at the same time all possible problems in NP are tractable in the average case, in the paper "A Princeton University workshop in 2009 studied the status of the five worlds" [30].

8 – Results about difficulty of proof:

Is the $P = NP$? This is however million dollar prize, a huge amount offered to researcher solve this problem, some efforts are made and it have led to various other new approaches. Some of the advance researches about the $P = NP$ show that the techniques that has already existing are not enough and powerful which can answer our question, so some of the novel techniques are required.

The problem is really difficult, so the all known proofing techniques essentially computational complexity theory can be categorize in one of the following class, however each of which is failed to proof that $P \neq NP$.

8.1 – Relativizing proofs:

Suppose a scenario in which each algorithm made queries to a fixed subroutine called oracle, and the running time of an algorithm in not counted against the running time of oracle.

Most of the classical proofs can be applied uniformly with the oracles regardless of what the oracle does. These proofs are known as relativizing. With the respect of oracle, in 1975 some researchers namely, Baker, Gill and Solovay proved that $P = NP$, while $P \neq NP$ for other oracles [31]. These relativizing proofs can prove only uniformly true statements regarding the oracle, and that way it shows that the relativizing techniques cannot solve the $P = NP$.

8.2 – Natural proofs:

For the circuit complexity of lower bounds, a general class of proving techniques is defined by the Alexander Razborov and Steven Rudich in 1993, which is known as natural proof. This circuit complexity approach was well known to resolve the $P = NP$, since previously all known circuit techniques of lower bounds were natural. Both of the researchers show that the no method of natural proof can distinguish between P and NP , if one-way functions exist. However, it has never been proven that formally they exist, however one believes that they do by most of the mathematicians. At the same time the proof or disproof of their existence is much stronger than quantification of P relative to NP . So it is unlikely that these natural proofs can alone resolve the $P = NP$.

8.3 – Algebrizing proofs:

Some result of Baker, Gill and Solovay shows that newly techniques which are non-relativizing can be successfully used to prove that $IP = PSPACE$. Scott Aaronson and Avi Wigderson in 2008 shows that some of the technical tool which used to prove $IP = PSPACE$ known as arithmetization was insufficient also to resolve $P = NP$ [32].

All of these barriers are another reason that why the NP -complete problems are very useful. If the demonstration of the polynomial-time algorithm for these NP -complete problems than one could solve $P = NP$ problem, including the above result.

These are some massive barriers which lead to computer scientists that P vs NP may be independent in nature of standard axiom system, i.e. ZFC which cannot be prove or disprove. By examining these independent result, one can conclude that there are no polynomial-time algorithm exists for the NP -complete problems, so the construction of the proof is failed, i.e. ZFC, or if the polynomial-time algorithm exists for NP -complete problem exists but it is impossible to prove [33]. Using technique of sort one it can be shown that the problem is undecidable even with the weaker assumption with extending the Peano axioms (PA) for the integer arithmetic, then for every NP problem the nearly-polynomial-time algorithms exist [34]. However it is believed that there are not efficient algorithms for the entire problem in NP leads to the phenomena that independence proof using those techniques cannot be possible. This shows that using those known techniques to prove the independence for PA or ZFC are not easier than proving that the efficient algorithms exist for all NP problems.

9 – Claimed solutions:

Some of the researchers claimed the solution of unsolved problem of P vs NP [35]. A comprehensive list is held by Gerhard J. Woeginger [36]. Vinay Deolalikar and Palo Alto also claimed the proof in August 2010 that $P \neq NP$ and they got heavy attention on the internet and press [37]. Publicly this proof has been reviewed by some academics [38][39] and an expert Neil Immerman pointed out some of the fatal error in that proof [40].

Deolalikar work in detail on that proof in September 2010, [41] and the other studies show by some of the theoretical computer scientists that the proof is not correct nor have any significant advancement to understand the problem [42]. In 2013 the assessment is prompted by The New Yorker article to call that proof attempt thoroughly discredited [43].

10 – Logical characterizations:

Some of the certain classes which contain the logical statements to measure the descriptive complexity to restated the problem of $P = NP$. For that all languages with the finite structures is considered in a linear order relation. Then all the classes which fall in the language P could be express in 1st order logic with the some addition of some suitable combinatory of fixed-point. This combinational order allows the recursive functions definition effectively. At least one function or predicates contain by the signature which additionally distinguished the order relation. These kinds of functions take polynomial space amount to store the finite structures, characterizes as P .

The languages that are logically fall in 2nd order in class NP . The functions, relations and subset by the universal quantification are however excluded in that 2nd order logic. As well as that 2nd order logic is related to all the languages that are in polynomial hierarchy (PH). So the question arises is that, is P which is a subset of NP can be reformulated as; is the 2nd order logic existentially able to describe the languages that the 1st order logic cannot with the fixed point? [44] That language is based on finite linearly ordered structures with nontrivial signature. In the previous characterization the word “existential” can be dropped, since $P = NP \leftrightarrow P = PH$ as we establish that $NP = co-NP$ implies that $NP = PH$.

11 – Polynomial time algorithms:

For the NP -complete problem there is no polynomial time algorithm exists. However if $P = NP$ then the polynomial time algorithm for NP -complete may be exists with some enormous constant which make the algorithm impractical. Levin represent the following algorithm which accepts the NP -complete SUBSET-SUM language correctly without any citation in the polynomial time if and only if $P = NP$.

The algorithm runs in polynomial time if it is accepting, i.e. answer is “yes”. On the other

hand, if the answer is “no” the algorithm runs forever. If $P = NP$ then the algorithm is enormously impractical. The algorithm try to other programs first at least in $2b - 1$ if a short program that can be solve the SUBSET-SUM in the polynomial time if it is b bits long.

// Algorithm that accepts the NP -complete language SUBSET-SUM. This is a polynomial-time algorithm if and only if $P = NP$. "Polynomial-time" means it returns "yes" in polynomial time when the answer should be "yes", and runs forever when it is "no"

// Input: S = a finite set of integers

// Output: "yes" if any subset of S adds up to 0.

// Runs forever with no output otherwise.

// Note: "Program number P " is the program obtained by

// writing the integer P in binary, then

// considering that string of bits to be a

// program. Every possible program can be

// generated this way, though most do nothing

// because of syntax errors.

FOR $N = 1 \dots \infty$

FOR $P = 1 \dots N$

Run program number P for N steps with input S

IF the program outputs a list of distinct integers

AND the integers are all in S

AND the integers sum to 0

THEN

OUTPUT "yes" and HALT

12 – Formal Definition:

The formal definition contains the following points.

12.1 – P and NP :

A decision problem takes string ‘ w ’ as an input from an alphabet Σ and output the Boolean value, i.e. “yes” or “no”. If there are a TM, i.e. an algorithm that takes the input string of length n and produce the answer correctly at most cn^k steps (k and c both are constant and independent of the input string), then one can guess that the problem can be solved in the polynomial time that fall in class P (set of the

languages which can be decided by a deterministic polynomial time TM.

$P = \{L : L = L(M) \text{ for some deterministic polynomial-time Turing machine } M\}$

Where $L(M) = \{w \in \Sigma^* : M \text{ accepts } w\}$

Hence, a deterministic polynomial-time TM is a deterministic TM M that satisfies the conditions;

1. M halts on all input w
2. $k \in n$ Such that $T_M(n) \in O(n^k)$; here O refers to big O notation and,
 $T_M(n) = \max \{t_M(w) : w \in \Sigma^*, |w| = n\}$
 $t_M(w) = \text{number of steps } M \text{ takes to halt on input } w$

NP can be defined traditionally by nondeterministic TM. The major concepts of certificate and verifier can be used to define the NP in modern way. Verifiers in form of set of languages contain the finite alphabet is used to defined NP formally which runs in polynomial time. The notion of verifier is defined as:

Let L be a language over a finite alphabet, Σ .

$L \in NP$ if, and only if, there exist a binary relation $R \subset \Sigma^* \times \Sigma^*$ and a positive integer k such that the following two conditions are satisfied:

1. For all $x \in \Sigma^*, x \in L \leftrightarrow \exists y \in \Sigma^*$ such that
 $(x, y) \in R$ and $(x, y) \in R \mid |y| \in O(|x|^k)$
2. The language $L_R = \{x \# y : (x, y) \in R\}$ over $\Sigma \cup \{\#\}$ is decidable by a TM in polynomial time.

LR is the verifier for L of the TM, where y is the certificate such that $(x, y) \in R$. In that case the verifier does not have the polynomial time, but verifier should be run in polynomial time for the L to be in NP.

12.1.1 – Example:

For example let;

$COMPOSITE = \{x \in N \mid x = pq \text{ for integers } p, q > 1\}$

$R = \{(x, y) \in N \times N \mid 1 < y \leq \sqrt{x} \text{ and } y \text{ divides } x\}$

The question is that if x is composite is same as if x is member of composite, where $COMPOSITE \in NP$ by the definition of the verifier satisfying the definition narrated above. This can be done only if the natural numbers can be identifying via binary representations. However via [45] [46] $COMPOSITE$ can also be happened in P .

12.2 – NP-completeness:

There are many ways for describing the concepts of NP-completeness.

Let L be a language over a finite alphabet Σ

The following two conditions should be satisfied to prove that L in NP-complete.

1. $L \in NP$
2. L in NP is polynomial time reducible to L as $L' \leq_p L$ if and only if following two conditions satisfied.
 - a. There exists $f : \Sigma^* \rightarrow \Sigma^*$ such that for all w in Σ^* we have:
 $f : \Sigma^* \rightarrow \Sigma^* (w \in L' \leftrightarrow f(w) \in L)$
 - b. There exists a polynomial-time TM that halts with $f(w)$ on its tape on any input w

13 – Conclusion:

The entire problems that falls in NP-completeness have some order parameter (at least one), and some time hard to solve due to those critical values that are near around to the order parameter. Separation of a region from one to other is performed by those critical values such as over and under constrained regions of the problem space. In that situation the transition in phase occurs 0 to 1 due to change in solution probability. However the phase transition does not arise in P problems unless it arises to bounded N , so it has bounded cost.

For the particular problem process I have provide the conjectures as empirical evidence. I also show that one hard problem in one space can be map to another hard problem in the other space. In that way preserve the phase boundary under the mapping. I have also shown some of the cases where P and NP classes have the distinction and also show that how the P classes was excluded from the critical region.

In that cases where the conjectures are true, then an NP problem turn into the P problem adding some restrictions excluding the order parameter of the critical values.

Note that only the reduced problems carry these results explaining why not these results previously noticed particularly. So there are a lot of outstanding question arises, such that:

1. What happened with NP-hard problems?
2. Is there any situation where the hard problem occurs in the non-critical region?
3. Do the other types of problems, i.e. games, optimization problems, etc. have the same properties?

14 – References:

- [1] R. E. Ladner "On the structure of polynomial time reducibility," Journal of the ACM, 22, pp. 151–171, 1975. Corollary 1.1.
- [2] Hartmanis, Juris. "Gödel, von Neumann, and the P = NP problem" (PDF). Bulletin of the European Association for Theoretical Computer Science 38: 101–107.
- [3] Cook, Stephen (1971). "The complexity of theorem proving procedures". Proceedings of the Third Annual ACM Symposium on Theory of Computing. pp. 151–158.
- [4] Fortnow, Lance (2009). "The status of the P versus NP problem" (PDF). Communications of the ACM 52 (9): 78–86. doi:10.1145/1562164.1562186.
- [5] Sipser, Michael: Introduction to the Theory of Computation, Second Edition, International Edition, page 270. Thomson Course Technology, 2006. Definition 7.19 and Theorem 7.20.
- [6] William I. Gasarch (June 2002). "The P=?NP poll." (PDF). SIGACT News 33 (2): 34–47. doi:10.1145/1052796.1052804. Retrieved 29 December 2008.
- [7] William I. Gasarch. "The Second P=?NP poll" (PDF). SIGACT News 74.
- [8] Scott Aaronson. "PHYS771 Lecture 6: P, NP, and Friends". Retrieved 27 August 2007.
- [9] Aviezri Fraenkel and D. Lichtenstein (1981). "Computing a perfect strategy for $n \times n$ chess requires time exponential in n ". J. Comb. Th. A (31): 199–214.
- [10] David Eppstein. "Computational Complexity of Games and Puzzles"
- [11] Arvind, Vikraman; Kurur, Piyush P. (2006). "Graph isomorphism is in SPP". Information and Computation 204 (5): 835–852.
- [12] Schöning, Uwe. "Graph isomorphism is in the low hierarchy". Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science 1987: 114–124.
- [13] Schöning, Uwe (1988). "Graph isomorphism is in the low hierarchy". Journal of Computer and System Sciences 37: 312–323.
- [14] Lance Fortnow. Computational Complexity Blog: Complexity Class of the Week: Factoring. 13 September 2002.
- [15] Pisinger, D. 2003. "Where are the hard knapsack problems?" Technical Report 2003/08, Department of Computer Science, University of Copenhagen, Copenhagen, Denmark
- [16] Gondzio, Jacek; Terlaky, Tamás (1996). "3 A computational view of interior point methods". In J. E. Beasley. Advances in linear and integer programming. Oxford Lecture Series in Mathematics and its Applications 4. New York: Oxford University Press. pp. 103–144. MR 1438311. Postscript file at website of Gondzio and at McMaster University website of Terlaky
- [17] Rosenberger, Jack (May 2012). "P vs. NP poll results". Communications of the ACM 55 (5): 10.
- [18] Scott Aaronson. "Reasons to believe". point 9.
- [19] See Horie, S. and Watanabe, O.; Watanabe (1997). "Hard instance generation for SAT". Algorithms and Computation. Lecture Notes in Computer Science (Springer) 1350: 22–31. arXiv:cs/9809117. Bibcode:1998cs.....9117H. doi:10.1007/3-540-63890-3_4. ISBN 978-3-540-63890-2. for a reduction of factoring to SAT. A 512 bit factoring problem (8400 MIPS-years when factored) translates to a SAT problem of 63,652 variables and 406,860 clauses.
- [20] See, for example, Massacci, F. and Marraro, L. (2000). "Logical cryptanalysis as a SAT problem". Journal of Automated Reasoning (Springer) 24 (1): 165–203. doi:10.1023/A:1006326723002. CiteSeerX: 10.1.1.104.962. in which an instance

- of DES is encoded as a SAT problem with 10336 variables and 61935 clauses. A 3DES problem instance would be about 3 times this size.
- [21] Find a message M that when hashed by the function $H()$ gives a digest h , or $H(M)=h$
 - [22] De, Debapratim and Kumarasubramanian, Abishek and Venkatesan, Ramarathnam (2007). "Inversion attacks on secure hash functions using SAT solvers". Springer. pp. 377–382.
 - [23] Berger B, Leighton T (1998). "Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete". *J. Comput. Biol.* 5 (1): 27–40. doi:10.1089/cmb.1998.5.27. PMID 9541869.
 - [24] History of this letter and its translation from Michael Sipser. "The History and Status of the P versus NP question"
 - [25] David S. Johnson. "A Brief History of NP-Completeness, 1954–2012" (PDF). From pages 359–376 of *Optimization Stories*, M. Grötschel (editor), a special issue of *Documenta Mathematica*, published in August 2012 and distributed to attendees at the 21st International Symposium on Mathematical Programming in Berlin.
 - [26] Cook, Stephen (April 2000). "The P versus NP Problem" (PDF). Clay Mathematics Institute. Retrieved 18 October 2006.
 - [27] Knuth, Donald E. (May 20, 2014). "Twenty Questions for Donald Knuth". *informit.com*. InformIT. Retrieved 20 July 2014.
 - [28] L. R. Foulds (October 1983). "The Heuristic Problem-Solving Approach". *Journal of the Operational Research Society* 34 (10): 927–934. doi:10.2307/2580891. JSTOR 2580891
 - [29] R. Impagliazzo, "A personal view of average-case complexity," *sct*, pp.134, 10th Annual Structure in Complexity Theory Conference (SCT'95), 1995
 - [30] "Tentative program for the workshop on "Complexity and Cryptography: Status of Impagliazzo's Worlds"". Archived from the original on 2013-11-15.
 - [31] T. P. Baker, J. Gill, R. Solovay. Relativizations of the $P \stackrel{?}{=} NP$ Question. *SIAM Journal on Computing*, 4(4): 431–442 (1975)
 - [32] S. Aaronson and A. Wigderson (2008). *Algebrization: A New Barrier in Complexity Theory* (PDF). *Proceedings of ACM STOC'2008*. pp. 731–740. doi:10.1145/1374376.1374481.
 - [33] Aaronson, Scott. "Is P Versus NP Formally Independent?"
 - [34] Ben-David, Shai; Halevi, Shai (1992). "On the independence of P versus NP". Technical Report 714. Technion.
 - [35] John Markoff (8 October 2009). "Prizes Aside, the P-NP Puzzler Has Consequences". *The New York Times*.
 - [36] Gerhard J. Woeginger. "The P-versus-NP page". Retrieved 25 May 2014.
 - [37] Markoff, John (16 August 2010). "Step 1: Post Elusive Proof. Step 2: Watch Fireworks.". *The New York Times*. Retrieved 20 September 2010.
 - [38] Polymath Project wiki. "Deolalikar's P vs NP paper"
 - [39] Science News, "Crowdsourcing peer review"
 - [40] Dick Lipton (12 August 2010). "Fatal Flaws in Deolalikar's Proof?"
 - [41] Dick Lipton (15 September 2010). "An Update on Vinay Deolalikar's Proof". Retrieved 31 December 2010
 - [42] Gödel's Lost Letter and $P=NP$, Update on Deolalikar's Proof that $P \neq NP$
 - [43] Alexander Nazaryan (2 May 2013). "A Most Profound Math Problem". Retrieved 1 May 2014
 - [44] Elvira Mayordomo. "P versus NP" *Monografías de la Real Academia de Ciencias de Zaragoza* 26: 57–68 (2004)
 - [45] M. Agrawal, N. Kayal, N. Saxena. "Primes is in P" (PDF). Retrieved 29 December 2008
 - [46] AKS primality test
 - [47] Geere, Duncan. "'Travelling Salesman' movie considers the repercussions if P equals NP". *Wired*. Retrieved 26 April 2012