International Journal in IT & Engineering (IJITE)
Volume 6 Issue 8, August 2018 ISSN: 2321-1776 Impact Factor: 6.341
Journal Homepage: http://ijmr.net.in, Email: irjmss@gmail.com
Blind Peer Reviewed Refereed Open Access International Journal

P COMPLETE PROBLEM WITH CIRCUIT AND MONOTONE CIRCUIT AND NP COMPLETENESS PEOBLEM WITH SATISFIABILITY AND 3-COLORING

Amit Kumar Nahar[1], Dr. Om Parkash[2]
Department of Computer Science
[1,2]OPJS University, Churu (Rajasthan) – India

ABSTRACT

NP Complete (truncated as NPC) problems, remaining at the core of choosing whether P=NP, are among hardest problems in computer science and other related areas. Through decades, NPC problems are treated as one class. Seeing that NPC problems have various natures, it is impossible that they will have a similar complexity. Our escalated investigation demonstrates that NPC problems are not all equivalent in computational complexity, and they can be additionally classified. In most of this course, we will look at the asymptotic complexity of problems. As opposed to considering, express, the time required to solve 3-coloring on graphs with 10; 000 nodes on some particular model of computation, we will ask what is the best asymptotic running time of an algorithm that solves 3-coloring on all instances. Surely, we will be significantly less ambitious, and we will basically ask whether there is a \feasible" asymptotic algorithm for 3-coloring.

1. INTRODUCTION

As we solve larger and progressively complex problems with more prominent computational power and cleverer algorithms, the problems we can't handle start to emerge. The theory of NP-completeness causes us comprehend these limitations and the P versus NP problems starts to pose a potential threat not similarly as an interesting hypothetical inquiry in computer science, however as an essential principle that saturates every one of the sciences. So while we don't expect the P versus NP problem to be resolved sooner rather than later, the inquiry has driven research in a number of subjects to enable us to get, handle and even exploit the hardness of different computational problems.

P problems imply that the class of problems can be solved precisely in polynomial time while NPC problems represent a class of problems which can be solved in nondeterministic polynomial time by Turing machine. NPC problems have extensive consequences to different problems in mathematics, science, theory and cryptography. All the more explicitly, in Big O-notation of computational complexity for asymptotic efficiency of algorithms, P problems can be solved in polynomial time of O(n k) for some consistent k where n is the size of input to the problem, while NPC problems may have computational complexity of $O(2c^n)$ including both exponential time and sub-exponential time, where c is steady larger than zero.

- **P:**The class of problems which can be solved by a deterministic polynomial algorithm.
- **NP:**The class of decision problem which can be solved by a non-deterministic polynomial algorithm.

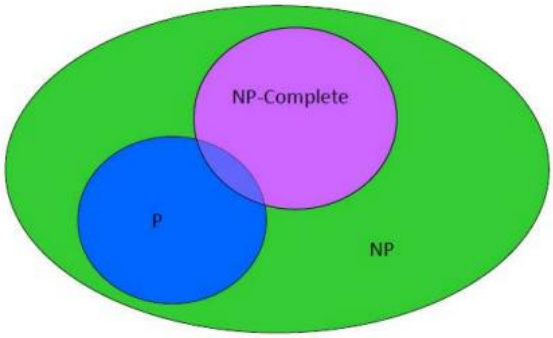

Figure 1: Complete Problem

- **NP-hard:**The class of problems to which each NP problem reduces
- **NP-complete (NPC):** the class of problems which are NP-hard and have a place with NP.

The complexity class P is contained in NP, and NP contains numerous significant problems. The hardest of which are NP Complete (NPC) problems. A decision problem d is NPC if: d is in NP, and each NP problem is reducible to d in polynomial time. The most significant open inquiry in complexity theory is the P versus NP problem which asks whether polynomial time algorithms actually exist for NPC problems and all NP problems.

2. P-COMPLETE PROBLEMS

To demonstrate that a problem is P-complete we should demonstrate that it is in P and that all problems in P can be reduced to by $\leq_P$ means of a log-space reduction. The task of demonstrating this is simplified by the knowledge that log-space reductions are transitive: if another problem Q has just been demonstrated to be P-complete, to demonstrate that P will be P-complete it gets the job done to appear there is a log-space reduction from Q to P and that $P \in P$.

- ✓ Circuit Value

Instance: A circuit depiction with fixed qualities for its input variables and a designated output gate.

Answer: "Yes" if the output of the circuit has esteem 1.

We demonstrate that the CIRCUIT VALUE problem depicted above is P-complete by showing that for each decision problem in P an instance w of and a DTM M that perceives "Yes"

instances of can be interpreted by a log-space DTM into an instance c of CIRCUIT VALUE with the end goal that w is a "Yes" instance of if and just if c is a "Yes" instance of CIRCUIT VALUE.

Since P is shut under supplements, it pursues that if the "Yes" in-positions of a decision problem can be resolved in polynomial time, so can the "No" instances. Along these lines, the CIRCUIT VALUE problem is equivalent to deciding the estimation of a circuit from its depiction. Note that for CIRCUIT VALUE the estimations of all variables of a circuit are incorporated into its depiction.

CIRCUIT VALUE is in P on the grounds that, as appeared in Theorem, a circuit can be assessed in a number of steps corresponding at the very least to the square of the length of its depiction. In this manner, an instance of CIRCUIT VALUE can be assessed in a polynomial number of steps.

Monotone circuits are built of and additionally gates. The functions computed by monotone circuits' structure an asymptotically small subset of the set of Boolean functions. Likewise, numerous significant Boolean functions are not monotone, for example, binary addition. Be that as it may, despite the fact that monotone circuits are a limited class of circuits, the monotone version of CIR-CUIT VALUE, characterized underneath, is additionally P-complete.

- ✓ Monotone Circuit Value

Instance: A portrayal for a monotone circuit with fixed qualities for its input variables and a designated output gate.

Answer: "Yes" if the output of the circuit has esteem 1.

CIRCUIT VALUE is a beginning stage to demonstrate that numerous different problems are P-complete. We start by diminishing it to MONOTONE CIRCUIT VALUE.

Theorem 1MONOTONE CIRCUIT VALUE *is* **P**-*complete.*

Proof As appeared in Problem, each Boolean function can be realized with just AND as well as gates (this is known as double rail logic) if the estimations of input variables and their supplements are made accessible. We reduce an instance of CIRCUIT VALUE to an instance of MONOTONE CIRCUIT VALUE by supplanting each gate with the pair of monotone gates depicted in Problem 2.12. Such descriptions can be worked out in log-space if the gates in the monotone circuit are numbered appropriately. The reduction should likewise write out the estimations of variables of the first circuit and their complements.

The class of P-complete problems is rich. Space limitations expect us to limit our treatment of

this subject to two additional problems. We presently demonstrate that LINEAR INEQUALITIES depicted underneath is P-complete. LINEAR INEQUALITIES is significant in light of the fact that it is straightforwardly related to LINEAR PROGRAMMING, which is broadly used to describe optimization problems. The reader is approached to demonstrate that LINEAR PROGRAMMING is P-complete.

LINEAR INEQUALITIES

Instance: An integer-valued ×n framework A and segment m-vector b.

Answer: "Yes" if there is a rational column n-vector x > 0 (all segments are non-negative and in any event one is non-zero) with the end goal that$\boldsymbol{Ax \leq b}$.

We demonstrate that LINEAR INEQUALITIES is P-hard, that will be, that each problem in P can be reduced to it in log-space. The evidence that LINEAR INEQUALITIES is in P, a significant and troublesome result in its own right, isn't given here.

Theorem 2 LINEAR INEQUALITIES *is* P-*hard*.

Proof We give a log-space reduction of CIRCUIT VALUE to LINEAR INEQUALITIES. That is, we demonstrate that in log-space an instance of CIRCUIT VALUE can be changed to an in-position of LINEAR INEQUALITIES so an instance of CIRCUIT VALUE is a "Yes" instance if and just if the corresponding instance of LINEAR INEQUALITIES is a "Yes" instance.

The log-space reduction that we use converts each gate and input in an instance of a circuit into a set of inequalities. The inequalities portraying each gate are demonstrated as follows. (A balance connection a = b is equivalent to two inequality relations, a b and b a.) The reduction likewise writes the uniformity z = 1 for the output gate z. Since every variable must be non-negative, this last condition protects that the resulting vector of variables, x, fulfills x > 0.

	Input		Gates		
Type	**TRUE**	**FALSE**	**NOT**	**AND**	**OR**
Function	xi = 1	xi = 0	w = ¬u	w = u ∧ v	w = u ∨ v
Inequalities	xi = 1	xi = 0	$0 \leq w \leq 1$ w = 1 − u	$0 \leq w \leq 1$ $w \leq u$ $w \leq v$ u + v − 1 ≤ w	$0 \leq w \leq 1$ $u \leq w$ $v \leq w$ $w \leq u + v$

Given an instance of CIRCUIT VALUE, every task to a variable is converted into a uniformity articulation of the structure xi = 0 or xi = 1. Correspondingly, each AND, OR, and NOT gate is

converted into a set of inequalities of the structure appeared. Logarithmic transitory space does the trick to hold gate numbers and to write these inequalities in light of the fact that the number of bits needed to speak to each gate number is logarithmic in the length of an instance of CIRCUIT VALUE.

To see that an instance of CIRCUIT VALUE is a "Yes" instance if and just if the instance of LINEAR INEQUALITIES is likewise a "Yes" instance, see that inputs of 0 or 1 to a gate result in the correct output if and just if the corresponding set of inequalities powers the output variable to have a similar worth. By induction on the size of the circuit instance, the qualities computed by each gate are actually equivalent to the estimations of the corresponding output variables in the set of inequalities.

We give as our last case of a P-complete problem DTM ACCEPTANCE, the problem of choosing if a string is acknowledged by a deterministic Turing machine in a number of steps indicated as unary number. (The integer *k* is spoken to as unary number by a string of k characters.) For this problem it is increasingly helpful to give an immediate reduction from all problems in P to DTM ACCEPTANCE.

DTM ACCEPTANCE

Instance: A description of a DTM M, a string w, and an integer n written in unary. Answer: "Yes" if and just if M, when begun with input w, halts with the answer "Yes" in at most n steps.

Theorem 3 DTM ACCEPTANCE *is* P-*complete.*

Proof To demonstrate that DTM ACCEPTANCE is log-space complete for P, consider an arbitrary problem P in P and an arbitrary instance of P, in particular x. There is some Turing machine, say M_P, that acknowledges instances x of P of length n in time p (n), p a polynomial. We assume that p is incorporated with the specification of M_P. For example, if $p(y) = 2y^4 + 3y^2 + 1$, we can speak to it with the string ((2, 4), (3, 2), (1, 0)). The log-space Turing machine that makes an interpretation of M_P and x into an instance of DTM ACCEPTANCE writes the description of M_P together with the input x and the estimation of p(n) in unary. Steady temporary space gets the job done to move the descriptions of M_P and x to the output tape. To complete the verification we need just demonstrate that O (log n) temporary space does the trick to write the incentive in p (n) in unary, where n is the length of x.

Since the length of the input x is given in unary, that is, by the number of characters it contains, its length n can be written in binary on a work tape in space O(log n) by including the number of characters in x. Since it isn't hard to demonstrate that any intensity of a k-bit binary number can be computed by a DTM in work space O(k), it pursues that any fixed polynomial in n can be

computed by a DTM in work space O(k) = O(log n).

To demonstrate that DTM ACCEPTANCE is in P, we design a Turing machine that acknowledges the "Yes" instances in polynomial time. This machine copies the unary string of length n to one of its work tapes. Given the description of the DTM M, it simulates M with an all inclusive Turing machine on input w. When it completes a step, it progresses the head on the work tape containing n in unary, pronouncing the instance of DTM ACCEPTANCE acknowledged whether M ends without utilizing more than n steps. By definition, it will complete its simulation of M in at most n of M's steps every one of which uses a steady number of steps on the recreating machine. That is, it acknowledges a "Yes" instance of DTM ACCEPTANCE in time polynomial in the length of the input.

We signify by P the class of decision problems that are solvable in polynomial time. We state that a search problem characterized by a connection R is a NP search problem if the connection is productively computable and with the end goal that solutions, on the off chance that they exist, are short. Officially, R is a NP search problem if there is a polynomial time algorithm that, given x and y, chooses whether (x; y) 2 R, and if there is a polynomial p with the end goal that in the event that (x; y) 2 R at that point **j**y**j** · p(**j**x**j**).

We state that a decision problem L is a NP decision problem if there is some NP connection R with the end goal that x 2 L if and just if there is a y to such an extent that (x; y) 2 R. Equivalently, a decision problem L is a NP decision problem if there is a polynomial time algorithm V (¢; ¢) also, a polynomial p with the end goal that x 2 L if and just if there is a y, **j**y**j** · p(**j**x**j**) such that V (x; y) accepts.

We signify by NP the class of NP decision problems.

Equivalently, NP can be characterized as the set of decision problems that are solvable in polynomial time by a non-deterministic Turing machine. Assume that L is solvable in polynomial time by a non-deterministic Turing machine M: at that point we can define the connection R with the end goal that (x; t) 2 R if and just if t is a transcript of a tolerant computation of M on input x and it's anything but difficult to demonstrate that R is a NP connection and that L is in NP as indicated by our first definition. Assume that L is in NP as per our first definition and that R is the corresponding NP connection. At that point, on input x, a non-deterministic Turing machine can figure a string y of length not exactly p(jxj) and afterward acknowledge whether and just if $(x; y)_2$ R. Such a machine can be executed to run in non-deterministic polynomial time and it chooses L.

For a function t: N! N, we characterize by DTIME(t(n)) the set of decision problems that are solvable by a deterministic Turing machine inside time t(n) on inputs of length n, and by

NTIME(t(n)) the set of decision problems that are solvable by a non-deterministic Turing machine inside time t(n) on inputs of length n. In this way, **P** = ^{S}kDTIME(O(n^k)) and **NP** = ^{S}kDTIME(O(n^k)).

3. NP-COMPLETENESS

- **Reductions** -Let A and B be two decision problems. We state that A reduces to B, indicated A • B, if there is a polynomial time computable function f with the end goal that x 2 An if and just if f(x)inB Two immediate observations: in the event that A • B and B is in P, at that point additionally A 2 P (on the other hand, on the off chance that A • B, and A 62P then likewise B 62P); in the event that A • B and B • C, at that point additionally A • C.

- **NP-completeness** -A decision problem A is NP-hard if for each problem L 2 NP we have L • A. A decision problem A is NP-complete in the event that it is NP-hard and it has a place with NP. It is a simple observation that in the event that A is NP-complete, at that point An is solvable in polynomial time if and just if P = NP.

➢ NP-COMPLETE PROBLEM

Think about the accompanying decision problem, that we call U: we are given in input (M; x; t; l) where M is a Turing machine, x **2 f**0; 1**g**¤ is a conceivable input, and t and l are integers encoded in unary[2], and the problem is to decide if there is a y **2 f**0; 1**g**¤, **jyj** · l, such that M(x; y) accepts in · t steps.

It is immediate to see that U is in NP. One can characterize a technique VU that on input (M; x; t; l) and y acknowledges whether and as it were if **jyj** · l, and M(x; y) accepts in at most t steps.

Let L be an **NP** decision problem. Then there are algorithm V_L, and polynomials T_L and p_L, such that x[2] L if and only if there is y, **jyj** ·p_L(**j**x**j**) such that V_L(x; y) accepts; furthermore V_L runs in time at most T_L(**j**x**j** + **jyj**). We give a reduction from L to U. The reduction maps x into the instance f(x) = (V_L; x; T_L(**j**x**j** + p_L(**j**x**j**)); p_L(**j**x**j**)). Just by applying the definitions, we can see that x[2] L if and only f(x) **2** U.

As referenced over, the NP-complete problems are the problems in NP that are the most hard to solve. We have demonstrated that **NP ⊆ PSPACE ⊆ EXPTIME** or that each problem in NP, including the NP-complete problems, can be solved in exponential time. Since the NP-complete problems are the most difficult problems in NP, each of these is even under the least favorable conditions an exponential-time problem. Consequently, we realize that the NP-complete problems require either polynomial or exponential time, yet we don't know which.

The CIRCUIT SAT problem is to decide from a description of a circuit whether it tends to be fulfilled; that is, regardless of whether esteems can be doled out to its inputs with the end goal that the circuit output has esteem 1. As referenced over, this is our canonical NP-complete problem.

CIRCUIT SAT

Instance: A circuit description with n input variables $\{x_1,x_2. . .x_n\}$for some integer n and a designated output gate.

Answer: "Yes" if there is a task of qualities to the variables with the end goal that the output of the circuit has esteem 1.

CIRCUIT SAT is a NP-complete problem. The goal of this problem is to perceive the "Yes" instances of CIRCUIT SAT, instances for which there are esteems for the input variables with the end goal that the circuit has esteem 1.

We demonstrated that CIRCUIT SAT depicted above is NP-complete by evil spirit starting that for each decision problem P in NP an instance w of P and a NDTM M that acknowledges "Yes" instances of P can be deciphered by a polynomial-time (actually, a log-space) DTM into an instance c of CIRCUIT SAT with the end goal that w is a "Yes" instance of P if and just if c is a "Yes" instance of CIRCUIT SAT.

Despite the fact that it gets the job done to reduce problems in NP by means of a polynomial-time transformation to a NP-complete problem, every one of the reductions given in this part should be possible by a log-space transformation. We currently demonstrate that an assortment of different problems is NP-complete.

4. NP-COMPLETE SATISFIABILITY PROBLEMS

We demonstrated that SATISFIABILITY characterized beneath is NP-complete. In this section we exhibit that two variations of this language are NP-complete by simple augmentations of the fundamental verification that CIRCUIT SAT is NP-complete.

SATISFIABILITY

Instance: A set of**literals***X* = $\{x_1,x_1,x_2,x_2. . .x_n,x_n\}$and a sequence of**clauses**

= (c_1, c_2. . . c_m), where each clause c_i is a subset of *X*.

Answer: "Yes" if there is a (satisfying) assignment of values for the variables$\{x_1,x_2. . .x_n\}$ over the set B with the end goal that every provision has at any rate one exacting whose worth is 1.

The two variations of SATISFIABILITY are 3-SAT, which has all things considered three literals in every provision, and NAESAT, in which not all literals in every proviso have a similar worth.

3-SAT

Instance: A set of literals$X = \{x_1, x_1, x_2, x_2, \ldots, x_n, x_n\}$, and a sequence of clauses C = (c_1, c_2, . . . , c_m), where each clause c_i is a subset of X containing all things considered three literals.

Answer: "Yes" if there is an assignment of values for variables$\{x_1, x_2 \ldots x_n$*is a subset of X containing everything thought about three literals.*

Answer: "Yes" if there is an assignment of values for variables 1.

Theorem 43-SAT*is*NP-*complete.*

Proof The verification that SATISFIABILITY is NP-complete likewise applies to 3-SAT on the grounds that every one of the clauses produced in the transformation of instances of CIRCUIT SAT has all things considered three literals for each condition.

NAESAT

Instance: An instance of 3-SAT.

Answer: "Truly" if every proviso is satisfiable when not all literals have a similar value.

NAESAT contains as its "Yes" instances those instances of 3-SAT in which the literals in every statement are not all approach.

Theorem 5NAESAT*is*NP-*complete.*

Proof We reduce CIRCUIT SAT to NAESAT utilizing nearly a similar reduction concerning 3-SAT. Each gate is replaced by a set of clauses. The main contrast is that we add the new strict y to every two-exacting provision related with AND or potentially gates and to the condition related with the output gate. Unmistakably, this reduction can be performed in deterministic log-space. Since a "Yes" instance of NAESAT can be confirmed in nondeterministic polynomial time, NAESAT is in NP. We currently demonstrate that it is NP-hard.

Given a "Yes" instance of CIRCUIT SAT, we demonstrate that the instance of 3-SAT is a "Yes" instance. Since each condition is fulfilled in a "Yes" instance of CIRCUIT SAT, each provision of the corresponding instance of NAESAT has in any event one exacting with value 1. The clauses that don't contain the strict y by their temperament have not all literals equivalent. Those

containing y can be made to fulfill this condition by setting y to 0, along these lines giving a "Yes" instance of NAESAT.

Presently consider a "Yes" instance of NAESAT produced by the mapping from CIRCUIT SAT. Supplanting each strict by its supplement generates another "Yes" instance of NAESAT since the literals in every proviso are not all rise to, a property that applies when complementation. In one of these "Yes" instances y is doled out the value 0. Since this is a "Yes" instance of NAESAT, at any rate one strict in every condition has value 1; that is, every provision is satisfiable. This infers the first CIRCUIT SAT problem is satisfiable. It pursues that an instance of CIRCUIT SAT has been converted into an instance of NAESAT with the goal that the previous is a "Yes" instance if and just if the last is a "Yes" instance.

Step Type	*Corresponding Clauses*
$(i$ READ $x)$	$(\overline{g}_i \vee x)$ $(g_i \vee \overline{x})$
$(i$ NOT $j)$	$(\overline{g}_i \vee \overline{g}_j)$ $(g_i \vee g_j)$
$(i$ OR j $k)$	$(g_i \vee \overline{g}_j \vee y)$ $(g_i \vee \overline{g}_k \vee y)$ $(\overline{g}_i \vee g_j \vee g_k)$
$(i$ AND j $k)$	$(\overline{g}_i \vee g_j \vee y)$ $(\overline{g}_i \vee g_k \vee y)$ $(g_i \vee \overline{g}_j \vee \overline{g}_k)$
$(i$ OUTPUT $j)$	$(g_j \vee y)$

Figure 2:A reduction from CIRCUIT SAT to NAESAT

Reduction from CIRCUIT SAT to NAESAT is gotten by supplanting each gate in a "Yes" instance of CIRCUIT SAT by a set of clauses. The clauses utilized in the reduction from CIRCUIT SAT to 3-SAT are those appeared with the strict y evacuated. In the reduction to NAESAT the exacting y is added to the 2-strict clauses utilized for AND as well as gates and to the output condition.

THEOREM 63-COLORING*is*NP-*complete.*

Proof To demonstrate that 3-COLORING is in NP, see that a three-coloring of a graph can be proposed in nondeterministic polynomial time and confirmed in deterministic polynomial time.

We reduce NAESAT to 3-COLORING. Review that an instance of NAESAT is an instance of 3-SAT. A "Yes" instance of NAESAT is one for which every statement is satisfiable with not all literals equivalent. Let an instance of NAESAT comprise of m clauses$C$ = (c_1, c_2. . . c_m) containing exactly three literals from the set X = {x_1, x_1, x_2, x_2, . . . , x_n, x_n} of literals in n variables. (Utilize the method presented in the proof of Theorem 6 to guarantee that every proviso in an instance of 3-SAT has precisely three literals for each statement)

Given an instance of NAESAT, we build a graph G in log-space and demonstrate that this graph is three-colorable if and just if the instance of NAESAT is a "Yes" instance.

The graph G has a set of n variable triangles, one for every variable. The vertices of the triangle related with variable x_i are $\{v, x_i, x_i\}$. Accordingly, all the variable triangles share one vertex for all intents and purpose. For every condition containing three literals we build one statement triangle for each proviso. On the off chance that provision c_j contains literals λ_{j1} ,λ_{j2} , and λ_{j3} , its associated clause triangle has vertices labeled (j, λ_{j1}), (j, λ_{j2}), and (j, λ_{j3}). Finally, we add an edge between the vertex (j, λ_{jk}) and the vertex associated with the literal λ_{jk} .We currently demonstrate that an instance of NAESAT is a "Yes" instance if and just if the graph G is three-colorable. Assume the graph is three-colorable and the colors are $\{0, 1, 2\}$. Since three colors are needed to color the vertices of a triangle and the variable triangles share a vertex labeled ν for all intents and purpose, assume without loss of generality that this basic vertex has color 2. The other two vertices in every variable triangle are doled out value 0 or 1, values we provide for the related variable and its supplement.

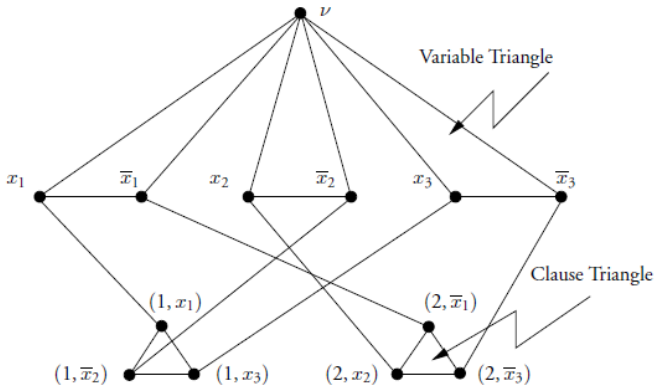

Figure 3:*G*corresponding to the clauses$c_1=\{x_1,x_2,x_3\}$and$c_2=\{x_1,x_2,x_3\}$in an instance of NAESAT.

It has one variable triangle for each variable and one clause triangle for each clause.

Consider now the coloring of provision triangles. Since three colors are needed to color vertices of a condition triangle, think about vertices with colors 0 and 1. The edges between these provision vertices and the corresponding vertices in variable triangles have various colors at each end. Give the literals access the condition triangles be given values that are the Boolean supplement of their colors. This gives values to literals that are predictable with the values of variables and guarantees that not all literals in a proviso have a similar value. The third vertex in every triangle has color 2. Give its literal a value steady with the value of its variable. It pursues that the clauses are a "Yes" instance of NAESAT.

Assume, then again, that a set of clauses is a "Yes" instance of NAESAT. We demonstrate that the graph G is three-colorable. Allocate color 2 to vertex ν and colors 0 and 1 to vertices

labeledx_i and x_i in view of the values of these literals in the "Yes" instance. Think about two literals in proviso c_j that are not both satisfied. If x_i (x_i) is one of these, give the vertex labeled (j, x_i) ((j, x_i)) the value that is the Boolean complement of the color of x_i (x_i) in its variable triangle. Do likewise for the other literal. Since the third literal has a similar value as one of the other two literals (they have various values), let its vertex have color 2. At that point G is three-colorable. Along these lines, G is a "Yes" instance of 3-COLORING if and just if the corresponding set of clauses is a "Yes" instance of NAESAT.

Instance: A set$S=\{u_1,u_2. . .u_p\}$and a family$\{S_1,S_2,. . .,S_n\}$of subsets of$S$.

Answer: "Yes" if there are disjoint subsets$S_{j1},S_{j2},. . .,S_{jt}$such that$\cup_{1\leq i\leq t}S_{ji}=S$.

5. CONCLUSION

In this section we discuss the complexity class NP that aims to capture the set of problems whose solutions can be efficiently verified. The famous P versus NP question asks whether or not the two are the same. We also introduce NP-completeness, an important class of computational problems that are in P if and only if P = NP

The class NP consists of all the languages for which membership can be certified to a polynomial-time algorithm. It contains many important problems not known to be in P. NP can also be defined using non-deterministic Turing machines. NP-complete problems are the hardest problems in NP, in the sense that they have a polynomial-time algorithm if and only if P =NP. Many natural problems that seemingly have nothing to do with Turing machines turn out to be NP-complete. One such example is the language 3SAT of satisfiable Boolean formulae in 3CNF form. If P = NP then for every search problem for which one can efficiently verify a given solution, one can also efficiently find such a solution from scratch.

REFERENCES

[1]. WenhongTian (2018) – "On The Classification Of NP Complete Problems And Their Duality Feature", International Journal of Computer Science & Information Technology (IJCSIT) Vol 10, No 1, February 2018
[2]. Tian, Wenhong. (2017). On the Transformability of P and NP Problems.
[3]. F. L. Traversa, C. Ramella, F. Bonani and M.D. Ventra, memcomputing NP-complete problems in polynomial time using polynomial resources and collective states, Science, Vol. 1, no. 6, e1500031, Nov. 2015.
[4]. Shivam Sharma (2015) – "P VS NP PROBLEM", (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 6 (6) , 2015, 5022-5025
[5]. Lance Fortnow, (2015) - The Status of the P versus NP problem, Communications of the ACM. 52 (9): 78–86, 2009.

[6]. V. Conitzer and T. Sandholm. New complexity results about Nash equilibria. Games and Economic Behavior, 63(2):621{641, July 2008.

[7]. Aaronson. NP-complete problems and physical reality. SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory), 36, 2005.

[8]. O'Donnell. Hardness amplification within NP. JCSS: Journal of Computer and System Sciences, 69, 2004.

[9]. G. J. Woeginger, Exact algorithms for NP-hard Problems: A Survey, Combinatorial optimization - Eureka, you shrink! Pages 185 - 207 , Springer-Verlag New York, Inc. New York, NY, USA ©2003

[10]. PierluigiCrescenzi and ViggoKann. A compendium of np optimization problems. http: //www.nada.kth.se/~viggo/problemlist/, 2000. Website tracking the tractability of many NP problems.