ACROPOLIS INSTITUTE OF TECHNOLOGY & RESEARCH, INDORE

ACROPOLIS
Enlightening Wisdom

Session: 2021-2022

Article NP-Complete

Submitted by:

Akshat Singh Rathore

4th semester **SEC**: CS-1

Enrollment no. - 0827CS201021

❖ What is NP Completeness

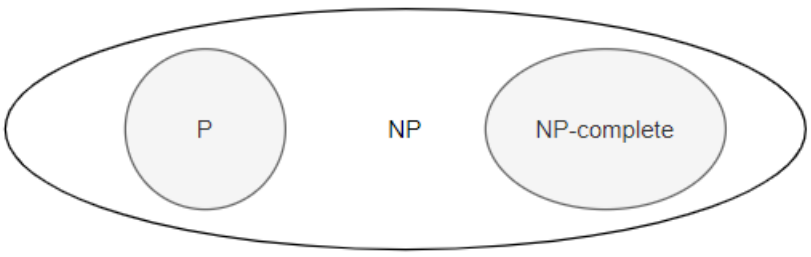
In computational complexity theory, a problem is **NP-complete** when:

1. it is a problem for which the correctness of each solution can be verified quickly (namely, in polynomial time) and a brute-force search algorithm can find a solution by trying all possible solutions.
2. the problem can be used to simulate every other problem for which we can verify quickly that a solution is correct. In this sense, NP-complete problems are the hardest of the problems to which solutions can be verified quickly. If we could find solutions of some NP-complete problem quickly, we could quickly find the solutions of every other problem to which a given solution can be easily verified.

The name "NP-complete" is short for "nondeterministic polynomial-time complete". In this name, "nondeterministic" refers to nondeterministic Turing machines, a way of mathematically formalizing the idea of a brute-force search algorithm.

Polynomial time refers to an amount of time that is considered "quick" for a deterministic algorithm to check a single solution, or for a nondeterministic Turing machine to perform the whole search. "Complete" refers to the property of being able to simulate everything in the same complexity class.

A problem is in the class NPC if it is in NP and is as **hard** as any problem in NP. A problem is **NP-hard** if all problems in NP are polynomial time reducible to it, even though it may not be in NP itself.



If a polynomial time algorithm exists for any of these problems, all problems in NP would be polynomial time solvable. These problems are called **NP-complete**. The phenomenon of NP-completeness is important for both theoretical and practical reasons.

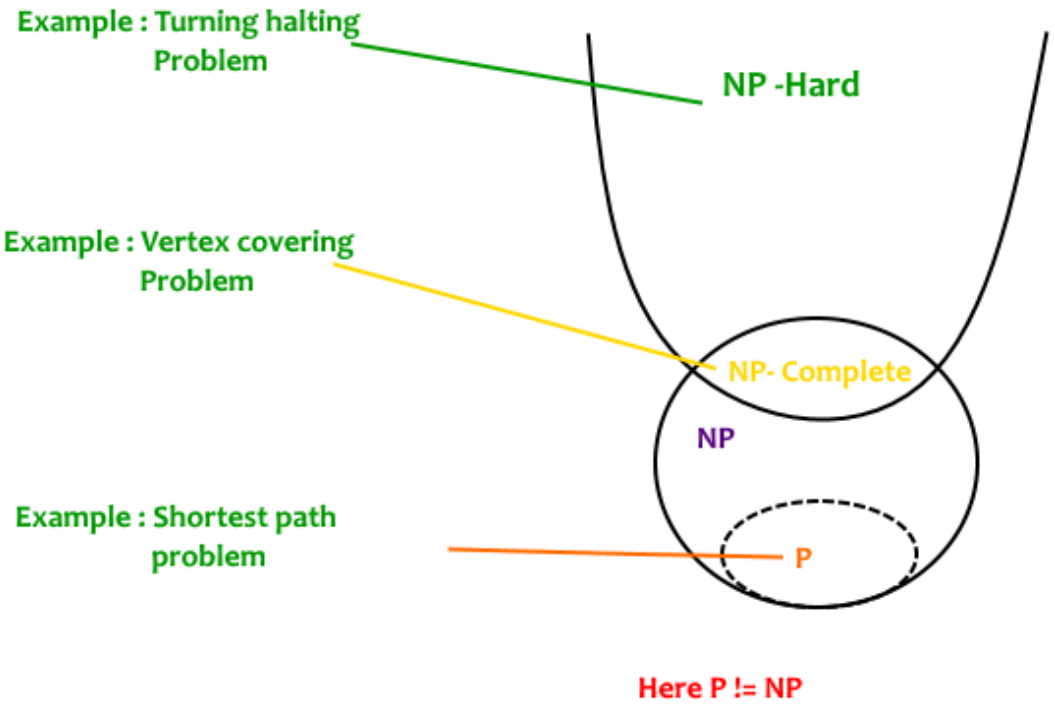
❖ Definition of NP-Completeness

NP-complete problems are the hardest problems in the NP set. A decision problem L is NP-complete if:

1) L is in NP (Any given solution for NP-complete problems can be verified quickly, but there is no efficient known solution).

2) Every problem in NP is reducible to L in polynomial time (Reduction is defined below).

A problem is NP-Hard if it follows property 2 mentioned above, doesn't need to follow property 1. Therefore, the NP-Complete set is also a subset of the NP-Hard set.



The problem in NP-Hard cannot be solved in polynomial time, until **P = NP**. If a problem is proved to be NPC, there is no need to waste time on trying to find an efficient algorithm for it. Instead, we can focus on design approximation algorithm.

A problem X is NP-Complete if there is an NP problem Y, such that Y is reducible to X in polynomial time. NP-Complete problems are as hard as NP problems. A problem is NP-Complete if it is a part of both NP and NP-Hard Problem. A non- deterministic turning machine can solve NP-Complete problem in polynomial time.

Status of NP Complete problems is another failure story, NP complete problems are problems whose status is unknown. No polynomial time algorithm has yet been discovered for any NP complete problem, nor has anybody yet been able to prove that no polynomial-time algorithm exists for any of them. The interesting part is, if any one of the NP complete problems can be solved in polynomial time, then all of them can be solved.

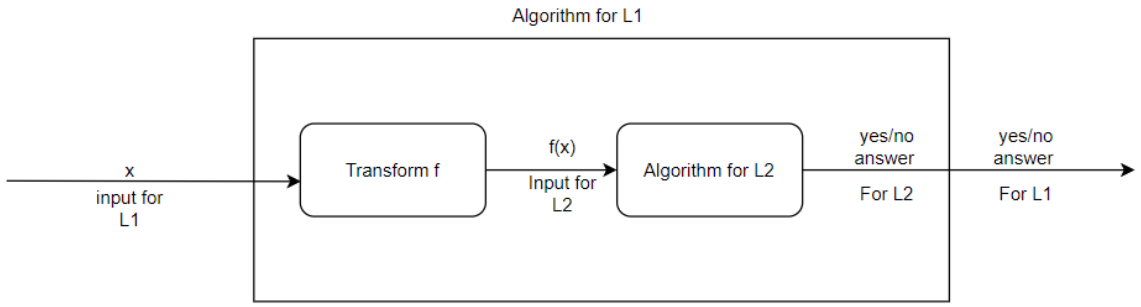
❖ Decision vs Optimization Problems

NP-completeness applies to the realm of decision problems. It was set up this way because it's easier to compare the difficulty of decision problems than that of optimization problems. In reality, though, being able to solve a decision problem in polynomial time will often permit us to solve the corresponding optimization problem in polynomial time (using a polynomial number of calls to the decision problem). So, discussing the difficulty of decision problems is often really equivalent to discussing the difficulty of optimization problems.
For example, consider the vertex cover problem (Given a graph, find out the minimum sized vertex set that covers all edges). It is an optimization problem. Corresponding decision problem is, given undirected graph G and k, is there a vertex cover of size k?

❖ What is Reduction?

Let L_1 and L_2 be two decision problems. Suppose algorithm A_2 solves L_2. That is, if y is an input for L_2 then algorithm A_2 will answer Yes or No depending upon whether y belongs to L_2 or not.

The idea is to find a transformation from L_1 to L_2 so that algorithm A_2 can be part of an algorithm A_1 to solve L_1.



Learning reduction, in general, is very important. For example, if we have library functions to solve certain problems and if we can reduce a new problem to one of the solved problems, we save a lot of time. Consider the example of a problem where we have to find the minimum product path in a given directed graph where the product of path is the multiplication of weights of edges along the path. If we have code for Dijkstra's algorithm to find the shortest path, we can take the log of all weights and use Dijkstra's algorithm to find the minimum product path rather than writing a fresh code for this new problem.

❖ How to prove that a given problem is NP complete?

From the definition of NP-complete, it appears impossible to prove that a problem L is NP-Complete. By definition, it requires us to that show every problem in NP in polynomial time reducible to L. Fortunately, there is an alternate way to prove it. The idea is to take a known NP-Complete problem and reduce it to L. If polynomial time reduction is possible, we can prove that L is NP-Complete by transitivity of reduction (If a NP-Complete problem is reducible to L in polynomial time, then all problems are reducible to L in polynomial time).

❖ What was the first problem proved as NP-Complete?

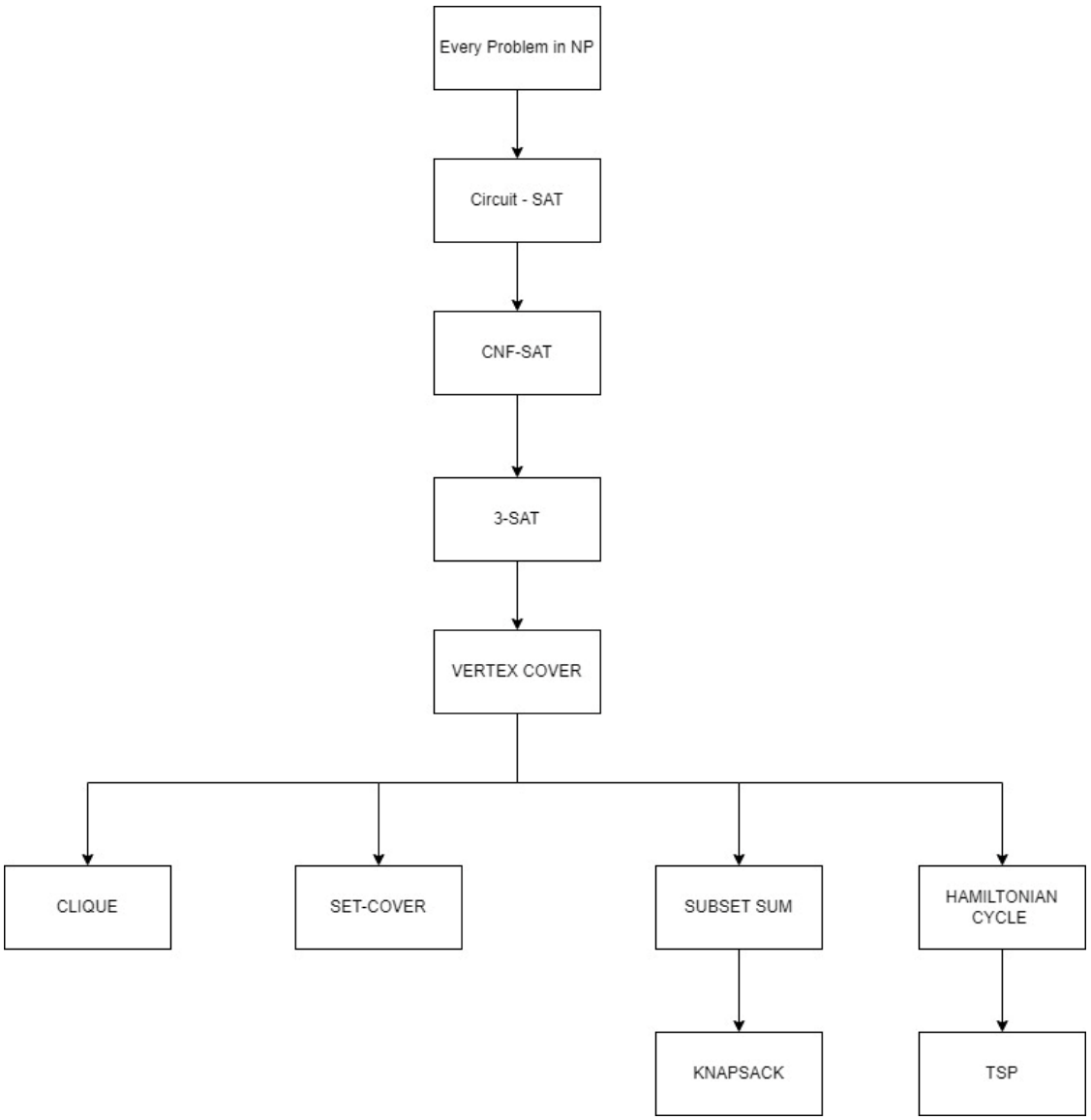
There must be some first NP-Complete problem proved by definition of NP-Complete problems. SAT (Boolean satisfiability problem) is the first NP-Complete problem proved by Cook (See CLRS book for proof).

It is always useful to know about NP-Completeness even for engineers. Suppose you are asked to write an efficient algorithm to solve an extremely important problem for your company. After a lot of thinking, you can only

come up exponential time approach which is impractical. If you don't know about NP-Completeness, you can only say that I could not come with an efficient algorithm. If you know about NP-Completeness and prove that the problem is NP-complete, you can proudly say that the polynomial time solution is unlikely to exist. If there is a polynomial time solution possible, then that solution solves a big problem of computer science many scientists have been trying for years.

❖ NP Complete Problems

To prove whether particular problem is NP complete or not we use polynomial time reductibility.The reduction is an important task in NP completeness proofs . this can be illustrated by the following figure



Various types of reduction are

LOCAL REPLACEMENT- this reduction A->B by dividing input A in the form of components and then these components can be converted to components of B

COMPONENT DESIGN- I this reduction A->B by building special component for input B that enforce propertied required by A

❖ The satisfiability problem

❖ CNF-SAT problem

SAT Problem: SAT(Boolean Satisfiability Problem) is the problem of determining if there exists an interpretation that satisfies a given boolean formula. It asks whether the variables of a given boolean formula can be consistently replaced by the values **TRUE or FALSE** in such a way that the formula evaluates to **TRUE**. If this is the case, the formula is called *satisfiable*. On the other hand, if no such assignment exists, the function expressed by the formula is **FALSE** for all possible variable assignments and the formula is *unsatisfiable*.

Problem: Given a boolean formula ***f***, the problem is to identify if the formula ***f*** has a satisfying assignment or not.

Explanation: An instance of the problem is an input specified to the problem. An instance of the problem is a boolean formula ***f***. Since an NP-complete problem is a problem which is both **NP** and **NP-Hard**, the proof or statement that a problem is NP-Complete consists of two parts:

1. *The problem itself is in NP class.*
2. *All other problems in NP class can be polynomial-time reducible to that.*
   (B is polynomial-time reducible to C is denoted as $\leq P^C$)

If the 2nd condition is only satisfied then the problem is called **NP-Hard**. But it is not possible to reduce every NP problem into another NP problem to show its NP-Completeness all the time i.e., to show a problem is NP-complete then prove that the problem is in NP and any NP-Complete problem is reducible to that i.e. if B is NP-Complete and B $\leq P^C$ For C in NP, then C is NP-Complete. Thus, it can be verified that the **SAT Problem** is NP-Complete using the following propositions:

SAT is in NP:

It any problem is in NP, then given a 'certificate', which is a solution to the problem and an instance of the problem (a boolean formula ***f***) we will be able

to check (identify if the solution is correct or not) certificate in polynomial time. This can be done by checking if the given assignment of variables satisfies the boolean formula.

SAT is NP-Hard:
In order to prove that this problem is NP-Hard then reduce a known problem, Circuit-SAT in this case to our problem. The boolean circuit **C** can be corrected into a boolean formula as:

- For every input wire, add a new variable $\mathbf{y_i}$.
- For every output wire, add a new variable **Z**.
- An equation is prepared for each gate.
- These sets of equations are separated by ∩ values and adding ∩**Z** at the end.

This transformation can be done in linear time. The following propositions now hold:

- If there is a set of input, variable values satisfying the circuit then it can derive an assignment for the formula ***f*** that satisfies the formula. This can be simulated by computing the output of every gate in the circuit.
- If there is a satisfying assignment for the formula ***f***, this can satisfy the boolean circuit after the removal of the newly added variables.

Therefore, the **SAT Problem** is NP-Complete.

Some other NP complete problems

1.the 0/1 Kanpsack problem- it can be proved as NP complete by reduction from Sum of SUBSET problem.

2.Hamiltonian Cycle-it can be proved as NP complete,by reduction from vertex cover.

3.Travelling salesperson problem-it can be proved as NP complete, by reduction from Hamiltonian cycle.

❖ Properties-

Viewing a decision problem as a formal language in some fixed encoding, the set NPC of all NP-complete problems is **not closed** under:

- union
- intersection
- concatenation
- Kleene star