Parul® University

# Design and Analysis of Algorithms

**Prof. Ankita Gandhi,** Assistant Professor
Computer Science & Engineering

Parul®
University

DIGITAL LEARNING CONTENT

# CHAPTER-4

# Tractable and Intractable Problems


# What is Tractable and Intractable Problems?

- **Polynomial Time:**

The time required for a computer to solve a problem, where this time is a simple polynomial function of the size of the input.

- **Tractable Problem:**

A **problem** that is solvable by a polynomial-time algorithm. The upper bound is polynomial.

**Example:**

❖Searching an unordered list
❖Searching an ordered list
❖Multiplication of integers
❖Finding a MST in a graph

# What is Tractable and Intractable Problems?

- **Intractable Problem:**

A **problem** that cannot be solved by a polynomial-time algorithm. The lower bound is exponential.

❖List all permutations of n numbers

❖Tower of Hanoi (worst-case running time that is at least 2n − 1.)

**What constitutes reasonable time?**

- Standard working definition: *polynomial time* (On an input of size *n* the worst-case running time is $O(n^k)$ for some constant *k* ).
- Polynomial time: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$
- Not in polynomial time: $O(2^n)$, $O(n^n)$, $O(n!)$

# What is Polynomial Time Algorithm?

- **Polynomial Time:**

The time required for a computer to solve a problem, where this time is a simple polynomial function of the size of the input.

- **Can all computational problems solved by computer?**

1. **Tractable or Easy Problems:**

Problems that are solvable by polynomial-time algorithms i.e. $O(n^k)$ algorithm is available, k constant

2. **Intractable or Hard Problems:**

Problems that require super polynomial time i.e. **No $O(n^k)$** algorithm is available **$O(n^n)$ or $O(n!)$ or $O(k^n)$ time required**

**NP-Complete**


# What is Polynomial Time Algorithm?

**3. Another Class of problems:**

- No reasonably fast algorithms have been found.
- Intractability of these problems cannot be proved

# Computability of Algorithms

❖Branch of the theory of computation in theoretical computer science that focuses on classifying computational problems according to their inherent difficulty and relating those classes to each other.

❖Closely related fields in theoretical computer science are analysis of algorithms and computability theory.

**That could be used to solve the same problem**

**particular algorithm to solve a problem**

Parul®
University

DIGITAL LEARNING CONTENT

# Computability of Algorithms

❖ **Computability** is the ability to solve a problem in an effective manner. The **computability** of a problem is closely linked to the existence of an **algorithm** to solve the problem.

There are divided in different part of algorithm:

- **P Class**
- **NP Class**
- **NP-Complete**
- **NP-Hard**

# Computability of Algorithms- P Class

**1. P Class**

The **class P** consists of those problems that are solvable in polynomial time, i.e. these problems can be solved in time $O(n^k)$ in worst-case, where k is constant. These problems are called tractable, while others are called intractable or super polynomial.

**Examples:**

- Basic multiplication of two numbers.

# Computability of Algorithms- P Class

**Why Polynomial?**

- Time comparisons of the most common algorithmorders

| n | log n | n log n | n (lg n)$^2$ | n$^2$ |
|---|---|---|---|---|
| 10 | 3 | 33 | 110 | 100 |
| 100 | 7 | 664 | 4414 | 10000 |
| 1000 | 10 | 9966 | 99317 | $10^6$ |
| 10000 | 13 | 132877 | 1765633 | $10^8$ |
| 100000 | 17 | 16660964 | 27588016 | $10^{10}$ |
| 1000000 | 20 | 19931569 | 397267426 | $10^{12}$ |

# Computability of Algorithms- P Class

**Order of Growth**

- Various classes for expressing efficiency of an algorithm

| | |
|---|---|
| Constant | 1 |
| Logarithmic | log n |
| Linear | n |
| n long n | n long n |
| Quadratic | $n^2$ |
| Cubic | $n^3$ |
| Exponential | $2^n$ |
| Factorial | n ! |


# NP Complete Problems

**The list below contains some well-known problems that are NP complete when expressed as decision problems.**

- Boolean satisfiability **problem** (SAT)
- Knapsack **problem**.
- Hamiltonian path **problem**.
- Travelling salesman **problem** (decision version)
- Sub graph isomorphism **problem**.
- Subset sum **problem**.
- Clique **problem**.
- Vertex cover **problem**.

# NP Complete Problems

NP-complete problems seem similar to problems that have polynomial-time algorithms.

**Examples:**

**Ex.1: Shortest vs. longest simple paths**

- Even with negative edge weights, **we can find shortest paths from a single source in a directed graph G = (V, E) in O(VE) time.**
- **Finding a longest simple path between two vertices is difficult, however.**
- **Determining whether a graph contains a simple path with at least a given number of edges is NP-complete.**


# NP Complete Problems

**Examples:**

**Ex.2: Euler tour vs. hamiltonian cycle**

- **An Euler tour** of a connected, directed graph G = (V, E) is a cycle that traverses each edge of G exactly once, although it is allowed to visit each vertex more than once.

**we can determine whether a graph has an Euler tour in only O(E) time.**

- A **hamiltonian cycle of** a directed graph G = (V, E) is a simple cycle that contains each vertex in V.

**Determining whether a directed graph has a hamiltonian cycle is NP-complete.**

# NP Complete Problems

**Examples:**

**Ex.3: 2-CNF satisfiability vs. 3-CNF satisfiability**

**A Boolean formula contains**

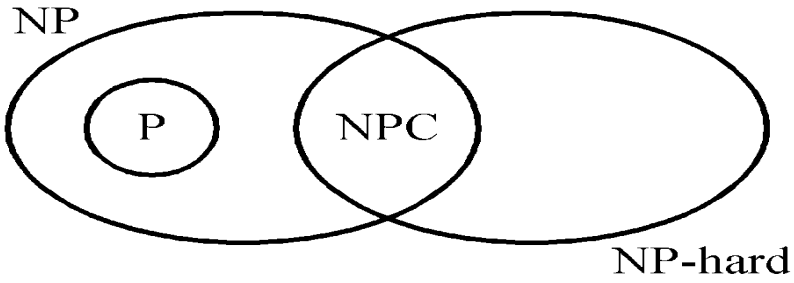
Variables whose values are 0 or 1

Boolean connectives such as ^ (AND), v (OR), and -(NOT) parentheses

**A Boolean formula is *satisfiable***

*if there exists* some assignment of the values 0 and 1 to its variables that causes it to evaluate to 1.

# NP Completeness and Classes P & NP




$$P \subset NP, NPC \subset NP, P \cap NPC = \varnothing$$

**P**: the class of problems **solvable in polynomial time.**

**NP**: the class of problems that are **verifiable in polynomial time .**

**NP-complete (NPC):** the class of problems which are **NP-hard and belong to NP.**

**NP-hard:** the class of problems to which every NP problem reduces.

# Class P – Polynomial Time

**The class of decision problems that have polynomial-time deterministic algorithms.**

- A deterministic algorithm is (essentially) one that always computes the correct answer.
- For a given particular input, the computer will produce the same output.

| Problem | Description | Yes | No |
|---|---|---|---|
| MULTIPLE | Is x a multiple of y? | 51, 17 | 51, 16 |
| RELPRIME | Are x and y relatively prime? | 34, 39 | 34, 51 |
| PRIMES | Is x prime? | 53 | 51 |

- Knapsack
- MST
- Sorting
- Others?

# Class NP – Non- deterministic Polynomial Time

**The class of decision problems that are solvable in polynomial time with a Non-deterministic algorithm.**

- A deterministic algorithm is (essentially) one that always computes the correct answer.
- For a given particular input, the computer will produce the same output.

**Examples:**

- Fractional Knapsack, MST, Sorting, Others?
- Hamiltonian Cycle (Traveling Salesman)
- Graph Coloring
- Satisfiability (SAT)

  The problem of deciding whether a given Boolean formula is satisfiable?

# Class NP – Non- deterministic Polynomial Time

NP consists of those problems that are **"verifiable" in polynomial time.** If we were somehow given a "certificate" of a solution, then we could verify that the certificate is correct in polynomial time.

**Examples:**

- In the hamiltonian cycle problem, given a directed graph G = (V, E), a certificate would be a **sequence ($V_1$, $V_2$, $V_s$, …, $V_{|v|}$) of |v| vertices**. It is easy to check in polynomial time that (vi, vi+1) ϵ E for I = 1, 2, 3, …, |v| and that ($V_{|v|}$, $V_1$) ϵ E as well.

- For 3-CNF satisfiability, a certificate would be an assignment of values to variables. We could check in polynomial time that this assignment satisfies the boolean formula.

# Class NPC – (NP-Complete)

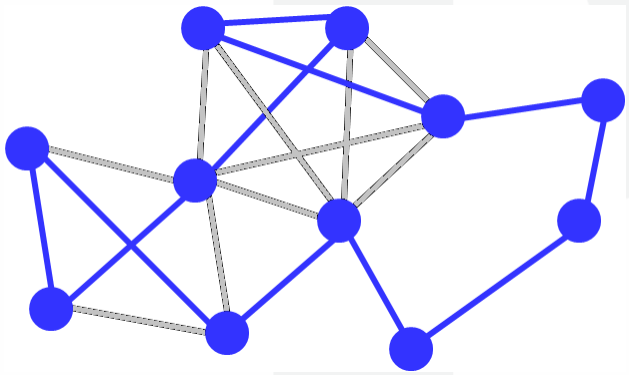
A decision problem D is **NP-complete** iff

1. D $\in$ NP
2. Is as hard as any problem in NP

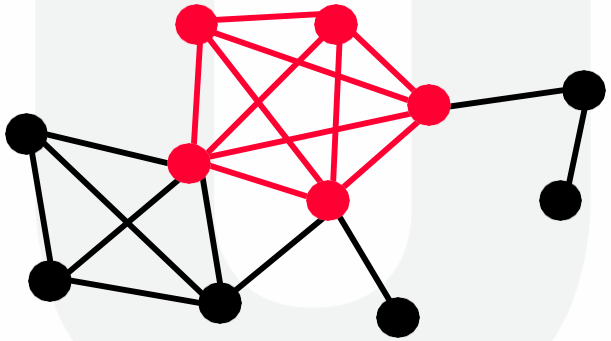
**Cook's theorem (1971): CNF-sat is NP-complete – Discussed this topic later on…**

# Example of NPC Problem

## Traveling Salesman



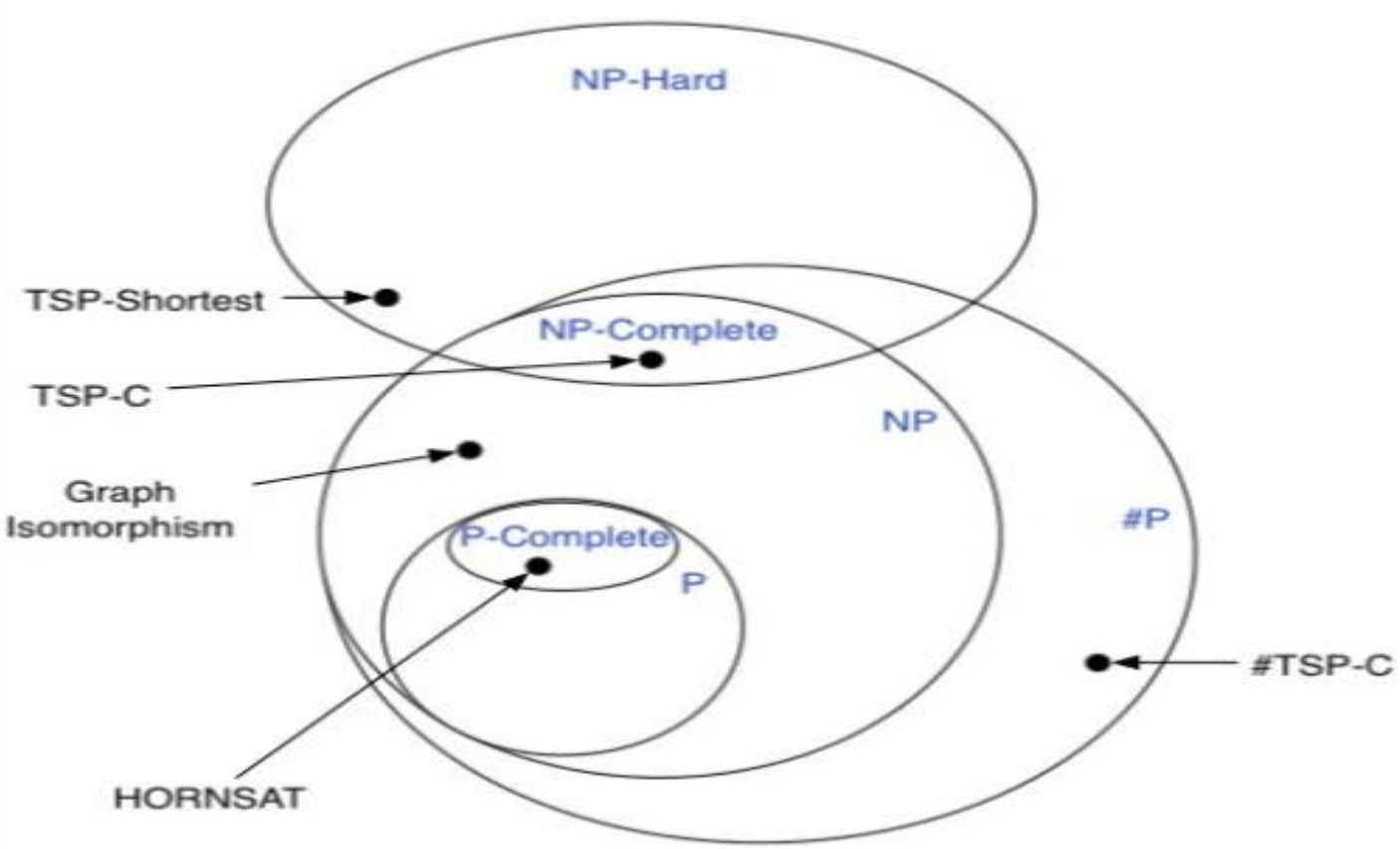
## 5-Clique




# NP Hard

- NP hard are problems that are at least as hard as the hardest problems in NP.
- Note that NP-Complete problems are also NP-hard. However not all NP-hard problems are NP (or even a decision problem), despite having 'NP' as a prefix.
- That is the NP in NP-hard does not mean 'non-deterministic polynomial time'.

# Comparison between P-NP-NP-Complete-NP-Hard



# Difference between P-NP-NP-Complete-NP-Hard

**P**- Polynomial time **solving** . Problems which can be solved in polynomial time, which take time like O(n), O(n2), O(n3). Eg: finding maximum element in an array or to check whether a string is palindrome or not. so there are many problems which can be solved in polynomial time.

**NP**- Non deterministic Polynomial time solving. Problem which can't be solved in polynomial time like TSP( travelling salesman problem) or An easy example of this is subset sum: given a set of numbers, does there exist a subset whose sum is zero?. but NP problems are checkable in polynomial time means that given a solution of a problem , we can check that whether the solution is correct or not in polynomial time.

# Difference between P-NP-NP-Complete-NP-Hard

Now we will discuss about NP-Complete and NP-hard. but first we need to know what is **reducibility** .

Here suppose we can take two problems A and B both are NP problems.

**Reducibility**- If we can convert one instance of a problem A into problem B (NP problem) then it means that A is reducible to B.

**NP-hard**-- Now suppose we found that A is reducible to B, then it means that B is at least as hard as A.

**NP-Complete** -- The group of problems which are both in NP and NP-hard are known as NP-Complete problem.

Now suppose we have a NP-Complete problem R and it is reducible to Q then Q is at least as hard as R and since R is an NP-hard problem. therefore Q will also be at least NP-hard , it may be NP-complete also.


# How to be show Problems to be NP-Complete

- **Three key concepts to show a problem to be NP-complete**
  1. Decision problems vs. optimization problems
  2. Reductions
  3. A first NP-complete problem

# How to be show Problems to be NP-Complete

**1. Decision problems vs. optimization problems**

- **Optimization Problems**
  - ➢Each feasible solution has an associated value
  - ➢We wish to find a feasible solution with best value

  **Examples**:

  0-1 Knapsack

  Fractional Knapsack

  Minimum Spanning Tree

- **Decision Problems**
  - ➢Solving the problem by giving an answer “YES” or “NO”

  **Examples:**

  Does a graph G have a MST of weight ≤W?


# How to be show Problems to be NP-Complete

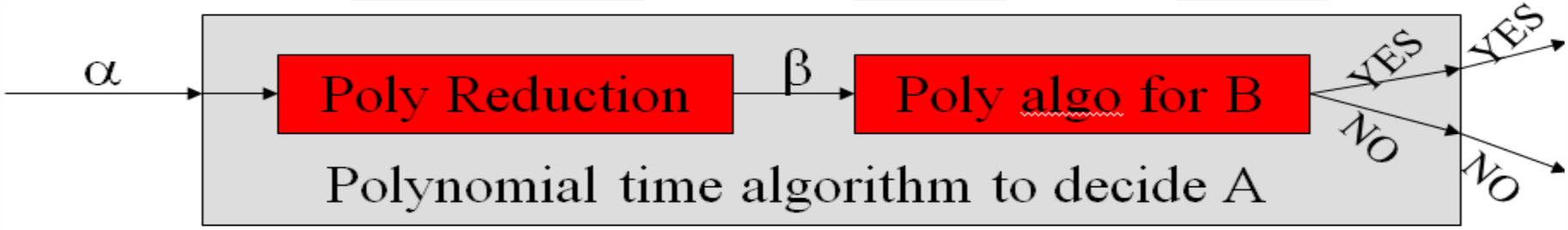
**2. Reductions**

- A problem A can be reduced to another problem B if any instance of A can be rephrased to an instance of B, the solution to which provides a solution to the instance of A. This rephrasing is called **a transformation.**
- Intuitively: If A reduces in polynomial time to B, A is “no harder to solve” than B
- **Example:** lcm(m, n) = m * n / gcd(m, n), lcm(m,n) problem is reduced to gcd(m, n) problem

# How to be show Problems to be NP-Complete

## 2. Reductions (Continue…)

- **Way to solve problem A in polynomial time**
  1. Given an instance $\alpha$ of problem A, use polynomial time reduction algorithm to transform it to an instance $\beta$ of problem B
  2. Run polynomial time algorithm for B for the instance $\beta$
  3. Use the answer for $\beta$ as the answer for $\alpha$



# How to be show Problems to be NP-Complete

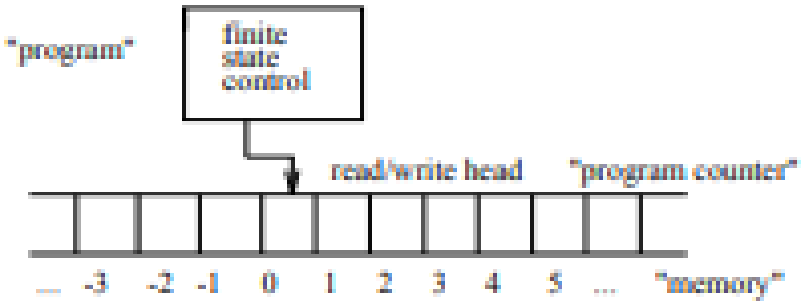
**3. A First NP-Complete Problem**

**Question: What is and how to prove the first NPC problem?**

- The technique of reduction relies on having a problem already known to be NP-complete.
- **Circuit-satisfiability problem**

# Cook's Theorem

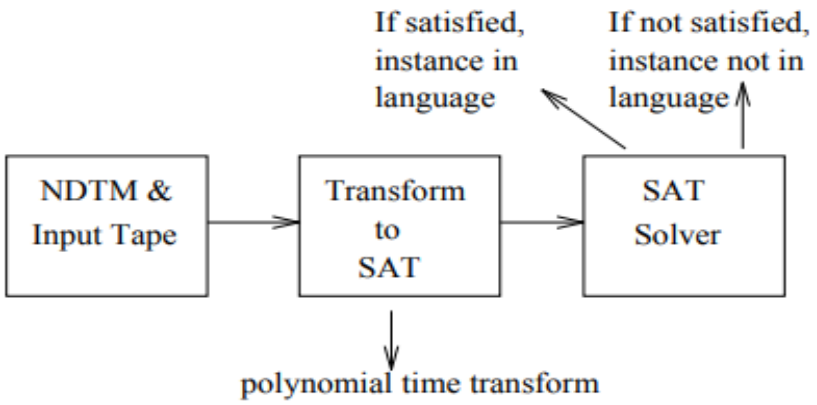
**Cook's Theorem** proves that satisfiability is NP-complete by reducing all non-deterministic Turing machines to SAT. Each Turing machine has access to a two-way infinite tape (read/write) and a finite state control, which serves as the program.



# Cook's Theorem- Satisfiability is NP-Complete

**Proof:** We must show that any problem in NP is at least as hard as SAT. Any problem in NP has a non-deterministic TM program which solves it in polynomial time, specifically P(n).

We will take this program and create from it an instance of satisfiability such that it is satisfiable if and only if the input string was in the language.



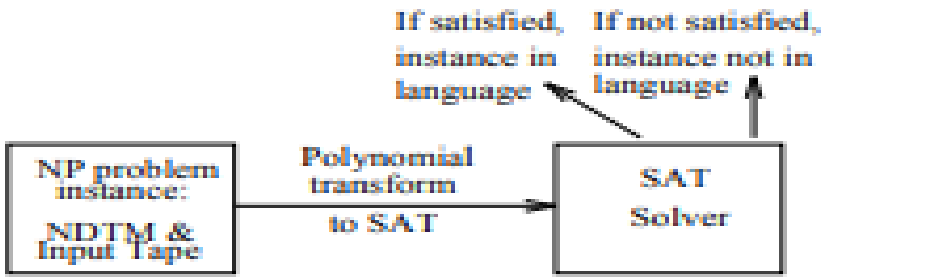

# Cook's Theorem- Satisfiability is NP-Complete

If a polynomial time transform exists, then SAT must be NP-complete, since a polynomial solution to SAT gives a polynomial time algorithm to anything in NP.

# Polynomial Time Reduction

**A decision problem is NP-hard** if the time complexity on a deterministic machine is within a polynomial factor of the complexity of any problem in NP.

**A problem is NP-complete** if it is NP-hard and in NP. Cook's theorem proved SATISFIABILITY was NP-hard by **using a polynomial time reduction** translating each problem in NP into an instance of SAT:





# Polynomial Time Reduction

- Since a polynomial time algorithm for SAT would imply a polynomial time algorithm for everything in NP, SAT is NP-hard.

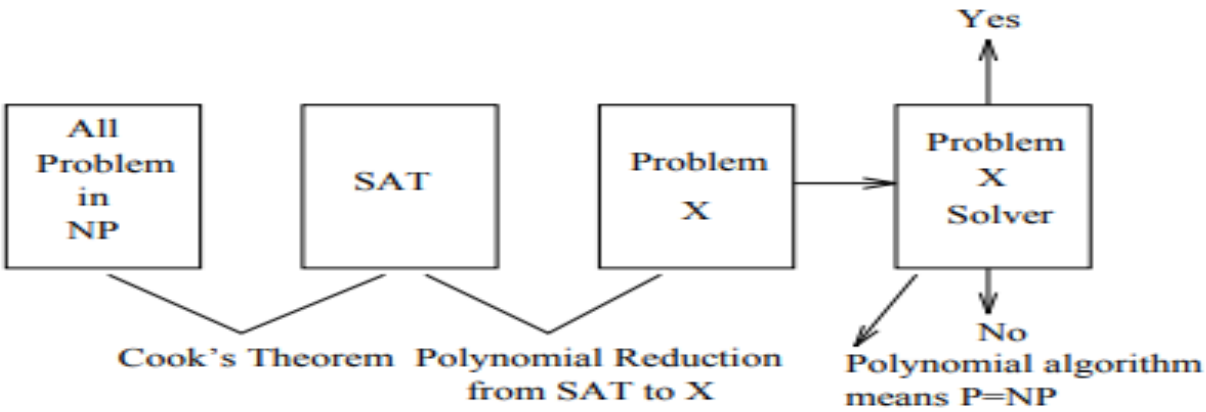
- Since we can guess a solution to SAT, it is in NP and thus NP-complete.

- The proof of Cook's Theorem, while quite clever, was certainly difficult and complicated.

- We had to show that all problems in NP could be reduced to SAT to make sure we didn't miss a hard one. But now that we have a known NP-complete problem in SAT.

- For any other problem, we can prove it NP-hard by polynomially transforming SAT to it!

# Polynomial Time Reduction




Since the composition of two polynomial time reductions can be done in polynomial time, all we need show is that SAT, i.e. any instance of SAT can be translated to an instance of x in polynomial time.

# DIGITAL LEARNING CONTENT


Parul® University

www.paruluniversity.ac.in