Design & Analysis of Algorithms

(Prof. Parminder Kaur Wadhwa)

CAMPUS 2 Date _____ Page _____

P, NP, NP-hard & NP-Complete Problem Classes

Topics Covered:-

1. Polynomial time problems
2. Non-polynomial time problems (exponential problems)
3. Deterministic Algorithms
4. Non-Deterministic Algorithms
5. Decision problem, Decision Algorithms
6. Optimization problem, optimization algorithms
7. Conjunctive Normal form (CNF)
8. Disjunctive Normalform (DNF)
9. CNF-Satisfiability [i.e. a satisfiability problem for CNF formula]
10. P problem class
11. NP problem class
12. Commonly believed relationship between P and NP problem classes.
13. Cook's Theorem
14. Problem L_1 reduces to problem L_2 [i.e. $L_1 \propto L_2$]
15. Satisfiability is exponential in nature
16. NP-hard problem class
17. NP-complete problem class
18. Commonly believed relationship among P, NP, NP-hard and NP-complete problem classes.
19. Some facts about the P, NP, NP-hard and NP-complete problem classes.
20. Halting Problem for Deterministic Algorithm - an NP-hard decision problem that is not NP-complete.

1. Polynomial Time problems

There are some problems whose best-known algorithms are polynomial.

e.g. Sorting algorithm = $O(n \log n)$

2. Non-Polynomial Time problems (exponential problems)

There are some problems whose best-known algorithms are non-polynomial i.e. exponential in nature.

e.g. sum of subsets problem (using backtracking) = $O(q(n)\, 2^n)$.

3. Deterministic Algorithm

The algorithm which has the property that the result of every operation is uniquely defined is called as deterministic algorithm.

4. Non-Deterministic Algorithm

If an algorithm contains operations whose outcomes are not uniquely defined but are limited to specific sets of possibilities is called as non-deterministic algorithm.

In such algorithms, a choice is made out of a set of elements. This choice is arbitrary and has no specific rule. The choice may lead to a successful completion or a failure of the algorithm. Thus, its output is

not unique.

For example:-

To search element x in an array A[1..n]

A non-deterministic algorithm for this is as follows:-

```
{
    int j = Choice (1,n)
    if ( A[j] == x)
    {
        cout << j;
        Success ();          // display successful completion
    }
    else
    { cout << '0';
      failure();             // declare unsuccessful completion
    }
}
```

5. Decision problem, Decision Algorithm

Any problem for which the answer is either zero or one [or True/False or Yes/No] is called as a decision problem.

An algorithm for a decision problem is termed as a decision algorithm.

For example:-

n-Queens' problem is a decision problem.

e.g. 4-Queens' problem is YES solvable.
3-Queens' problem is NOT solvable

6. Optimization Problem, Optimization Algorithm

Any problem that involves the identification of an optimal (either minimum or maximum) value of a given cost function is known as an optimization problem.

An optimization algorithm is used to solve an optimization problem.

For example:-

Knapsack problem involves the identification of maximum profit, so it is an optimization problem.

7. Conjunctive Normal Form (CNF)

A formula is in conjunctive Normal form (CNF) iff it is represented as $\bigwedge_{i=1}^{k} C_i$

(where C_i are clauses each represented as $\vee$ of boolean variables x_i or their negations $\bar{x}_i$.)

$\wedge$ = and
$\vee$ = or

for example :-

$(x_1 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee x_4)$ is in CNF.

8. Disjunctive Normal Form (DNF)

A formula is in disjunctive normal form (DNF) iff it is represented as $\bigvee_{i=1}^{k} C_i$

(where C_i are clauses each represented as $\wedge$ of boolean variables x_i or their negations $\bar{x}_i$)

$\wedge$ means AND
$\vee$ means OR

for example :-

$(x_2 \wedge \bar{x}_1) \vee (x_3 \wedge x_4)$ is in DNF.

9. CNF Satisfiability [i.e. a satisfiability problem (for CNF formulas)]

Given a propositional formula:-

$$E(x_1, \ldots, x_n)$$

Choosing truth values of boolean variables and verifying that the formula is true or not for that assignment is called as satisfiability problem.

i.e. to check whether $E(x_1, \ldots, x_n)$ is satisfiable or not for the given values of x_i.

There are 2^n possible assignments of n boolean variables [(i.e. x_1 to x_n)].

Thus, satisfiability problem is exponential in nature.

Actually it is exponential type problem, but we can have a polynomial time algorithm for it, if we don't try all the values of x_i but instead we use a non-deterministic algorithm.

Such an algorithm could proceed by simply choosing (non-deterministically) one of the 2^n possible assignments of x_i and will verify whether $E(x_1, \ldots, x_n)$ is true for that assignment and may be successful leading to solve it in polynomial time only.

10. P problem class

P is the set of all decision problems solvable by deterministic algorithms in polynomial time.

11. NP problem class

NP is the set of all decision problems solvable by non-deterministic algorithms in polynomial time.

Problem class	Type of problem	Type of Algorithm	Time taken by algorithm
P	decision problem	Deterministic Algorithm	Polynomial Time
NP	decision problem	Non-Deterministic Algorithm	Polynomial Time

P ⟹ (decision problem, Deterministic Algo., Polynomial Time)

NP → (" " , Non-Deterministic Algo, ")

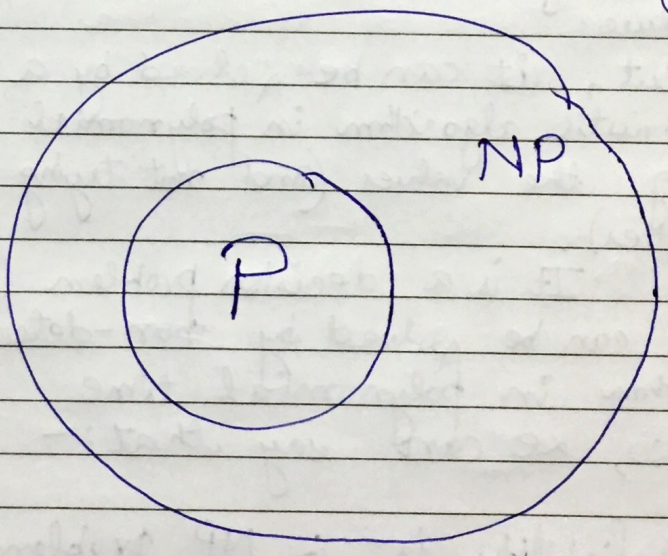
12. **Commonly believed relationship between P and NP problem classes :-**

Since, deterministic algorithms are just a special case of non-deterministic ones, we conclude that

$$P \subseteq NP$$

i.e. P is a proper subset of NP.

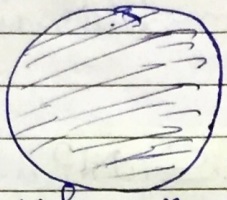
(Note: If you have only one CHOICE in non-deterministic algorithm, you will get a unique result every time, making the algorithm as a deterministic algorithm.
Thus, deterministic algorithms are special cases of non-deterministic algorithms.).



$$P \subset NP$$

13. Cook's Theorem

Satisfiability is in P if and only if P = NP.



Satisfiability is in P

The satisfiability problem is to determine whether a formula is true for some assignment of truth values to the variables.

It is having 2^n possible assignments of truth values of variables. Thus, is exponential in nature.

But, it can be solved by a non-deterministic algorithm in polynomial time by choosing the values (and not trying all possibilities).

It is a decision problem (True/False type), can be solved by non-deterministic algorithm in polynomial time.

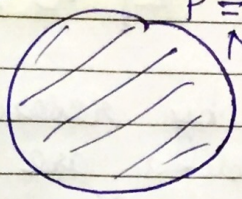
So, we can say that :-

Satisfiability is in NP problem class.

But if, the non-deterministic algorithm has only one choice, producing a uniquely

-defined output, then it becomes a deterministic algorithm.

i.e.



Then, Satisfiability problem (a decision problem) is being solved by a deterministic algorithm in polynomial time.

This implies that, now:-

Satisfiability is in P problem class.

Thus, Cook's Theorem says that:-

Satisfiability is in P, iff P = NP.

14. **Problem L_1 reduces to Problem L_2**

(i.e. $L_1 \propto L_2$)

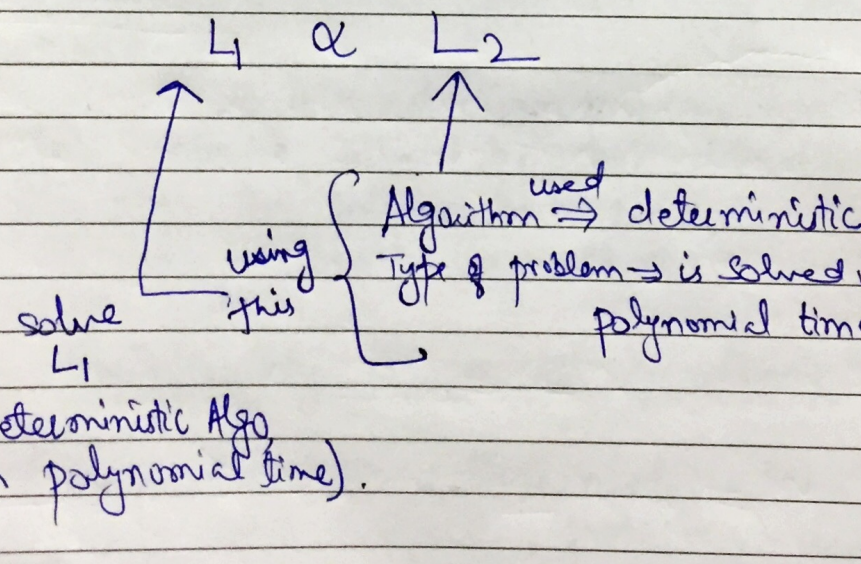
This is studied by researchers to understand that the problems are computationally related to each other.

For this they represent

$$L_1 \propto L_2$$

Explanation :-

Let L_1 and L_2 be problems. Problem L_1 reduces to L_2 (also written as $L_1 \propto L_2$) if and only if there is a way to solve L_1 by a deterministic polynomial time algorithm using a deterministic algorithm that solves L_2 in polynomial time.



The transitive relationship exists:-

e.g. if $L_1 \propto L_2$

(use L_2 to solve L_1)

$L_2 \propto L_3$

(L_2 is solved using L_3)

Then, this implies that:-

$L_1 \propto L_3$

(L_3 can be used to solve L_1)

Summarizing:-

if, $L_1 \propto L_2$
and $L_2 \propto L_3$
then $L_1 \propto L_3$

Let,

L_1 be a decision problem

L_2 be an optimization problem,

then

$L_1 \propto L_2$

means

Decision problem $\propto$ Optimization problem

means,

use optimization problem's deterministic algo that solve it in polynomial time to solve the corresponding decision problem using deterministic algorithm in polynomial time.

For example:-

The Knapsack decision problem reduces to the Knapsack optimization problem.

The Knapsack decision problem is to determine whether there is a 0/1 assignment of values to x_i, $1 \le i \le n$, such that $\sum p_i x_i \ge r$ and $\sum w_i x_i \le m$.

The r is a given number.

In fact,
one can also show that the optimization problem reduces to their corresponding decision problem.

optimization problem $\propto$ decision problem

also, it may be possible that:-

decision problem $\propto$ optimization problem

So, the problems are computationally related.

---

15. Satisfiability is exponential in nature

The satisfiability problem is to determine whether a formula is true for some assignment of truth values to the variables.

There are 2^n possible assignments of truth values.

So, satisfiability is exponential in nature.

16. NP- hard Problem class

A problem L is NP- hard if and only if satisfiability $\propto$ L.

i.e. satisfiability $\propto$ L

mean L is used to solve satisfiability problem (which is exponential in nature)

17. NP- Complete Problem class

A problem L is NP- complete if and only if L is NP- hard and L $\in$ NP.

Explanation:-

L is NP-hard

means, Satisfiability $\propto$ L

but L $\in$ NP, now

i.e.

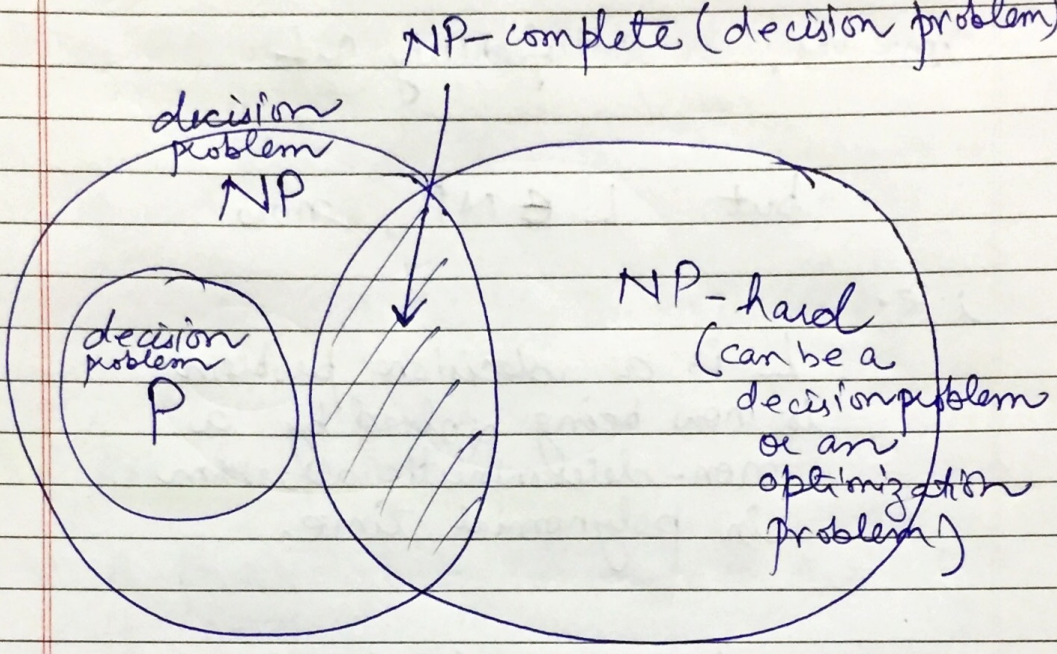
L is a decision problem is now being solved by a non-deterministic algorithm in polynomial time.

if L is NP-hard
and L $\in$ NP

then L is NP-complete

L was hard to solve but you have solved it in polynomial time (by using a non-deterministic algorithm), but, the nature of problem is the decision type of problem, as NP is a decision problem only. (So, reducing an optimization problem to decision problem to make it NP) [if required]

Q8. Commonly believed relationship among P, NP, NP-hard & NP-complete problem classes:-



NP-hard may be a decision problem or an optimization problem.

But are hard to solve.

We can say, they are solved in exponential time.

NP-complete — That NP-hard problem which is made to solve in polynomial time (using a non-deterministic algorithm), but it has to be ~~an~~ a decision problem.

That NP-hard which is a decision problem can be made to solve in polynomial time using a non-deterministic algorithm.

i.e. NP-hard decision problem can be made NP.

Then it is called as NP-complete problem.

That NP-hard problem which is an optimization problem can not become NP.

For this, we need to reduce that optimization problem into a decision problem.

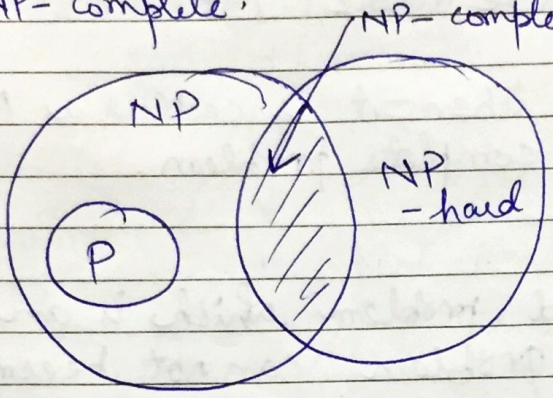
Optimization problem	$\propto$	decision problem

Converting an optimization problem into its corresponding decision problem, so that we can make that NP-hard optimization problem into NP (as only decision problems can be NP), and to finally make it NP-complete.

NP problems are decision problems being solved by non-deterministic algorithm in polynomial time. For, practical applications or actual implementation, we try to make NP as P.

19. Some facts about the P, NP, NP-hard and NP-complete problem classes

(a) There are NP-hard problems that are not NP-complete.



(b) Only a decision problem can be NP-complete.

(c) An optimization problem may be NP-hard.

(d) Optimization problems cannot be NP-complete whereas a decision problem can be NP-complete.

(e) There also exist NP-hard decision problems that are not NP-complete.

e.g. Halting problem for deterministic algorithms

(f) All NP-complete problems are NP-hard, but some NP-hard problems are not known to be NP-complete.

20. An example of a NP-hard decision problem that is not NP-complete.

Halting Problem for Deterministic Algorithm

The halting problem is to determine for an arbitrary deterministic algorithm A and an input I, whether algorithm A with Input I ever terminates (or enters an infinite loop).

This is a decision problem, as it will tell whether the algorithm will halt or not.

This is deterministic algorithm, as for the given Input I, for the given problem, it may either halt or will go infinite loop.

But, as the algorithm is arbitrary, the problem may be any and it is undecidable that whether the algorithm will ~~halt~~ halt successfully or will remain in infinite loop.

The halting problem for an arbitrary deterministic algorithm A and an input I is undecidable.

Hence, there exists no algorithm (of any complexity) to solve this problem.

So, it cannot be solved in polynomial time.

Hence, it cannot be in NP.

For example:-

An arbitary deterministic algorithm for halting problem

```
Algorithm A (Input I)
{
    do
    {
        check input I for required answer
        and halt successfully;
        break;
    } while (1);                    // infinite loop
```

As, algorithm A is an arbitary deterministic algorithm, the given problem may be any and we don't know whether for Input I, it will halt or will remain in infinite loop.

Thus, it is undecidable and thus cannot be solved in polynomial time.

∴ Halting problem is not in NP.

Halting problem is NP-hard :-

For a problem L to be NP-hard, it is checked if

$$\text{Satisfiability} \propto L$$

Checking, if

$$\text{Satisfiability} \propto \text{Halting problem}$$

Let us assume that an algorithm A whose input is a propositional formula X.

If X has n variables, then A tries out all 2^n possible truth assignments and verifies whether X is satisfiable.

If it is, then A stops.

If it is not, then A enters an infinite loop.

Hence, A halts on input X if and only if X is satisfiable.

for Satisfiability $\propto$ Halting problem

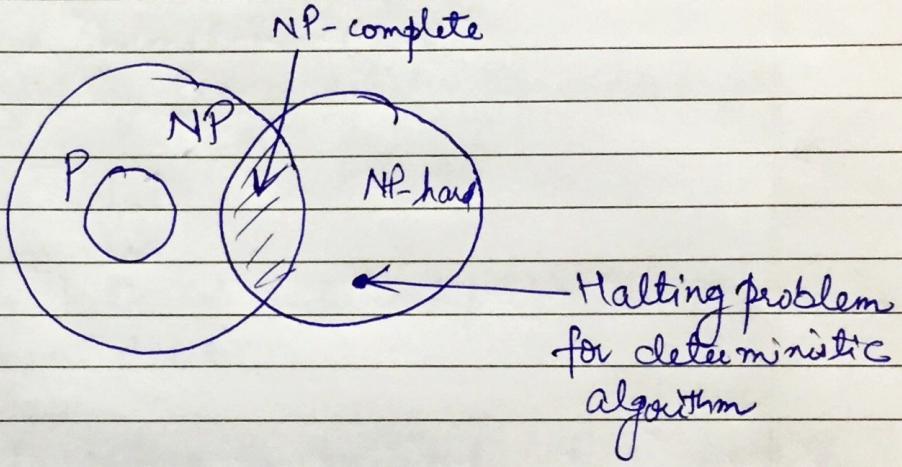
we are using this algo to solve satisfiability

If we had a polynomial time algorithm for the halting problem, then we could solve the satisfiability problem in polynomial time using A and X as input to the algorithm for the halting problem.

Hence, the halting problem is an NP-hard problem, as

satisfiability $\propto$ halting problem

But, halting problem is not NP.



Halting problem is a decision problem which is NP hard but not NP-complete.

[Thus, every NP-hard decision problem is not necessarily to be NP-complete.

Or

There exist NP-hard decision problems that are not NP-complete.]