Unit-5
NP hard and NP Complete problems
Basic Concepts

K. RAGHAVA RAO
Professor in CSE
KL University
krraocse@gmail.com
http://mcadaa.blog.com

NP –HARD AND NP – COMPLETE PROBLEMS

Basic Concepts

Algorithm Running Time

Given a size n problem, an algorithm runs $O(f(n))$ time:

1. $O(f(n))$: upper bound. (Ω :lower θ: equal)
2. Polynomial: $f(n)=1$(constant), n(linear), n^2(quadratic), n^k.
3. Exponential: $f(n)=2^n$, $n!$, n^n.

Time	$n=1$	$n=10$	$n=100$	$n=1000$
n	1	10	10^2	10^3
n^2	1	10^2	10^4	10^6
n^{10}	1	10^{10}	10^{20}	10^{30}
2^n	2	$>10^3$	$>10^{30}$	$>10^{300}$
$n!$	1	$>10^6$	$>10^{150}$	$>10^{2500}$

NP –HARD AND NP – COMPLETE PROBLEMS

Basic Conepts

Decision Problems

To keep things simple, we will mainly concern ourselves with decision problems. These problems only require a single bit output: ``yes'' and ``no''.

How would you solve the following decision problems?

-Is this directed graph acyclic?
-Is there a spanning tree of this undirected graph with total weight less than w?
-Does this bipartite graph have a perfect (all nodes matched) matching?
-Does the pattern p appear as a substring in text t?

NP –HARD AND NP – COMPLETE PROBLEMS

BASIC CONCEPTS

- The computing times of algorithms fall into two groups.

- Group1– consists of problems whose solutions are bounded by the polynomial of small degree.

Example – Binary search o(log n) , sorting o(n log n), matrix multiplication 0($n^{2.81}$).

NP –HARD AND NP – COMPLETE PROBLEMS

- **Group2** – contains problems whose best known algorithms are non polynomial.

- Example –Traveling salesperson problem $0(n^2 2^n)$, knapsack problem $0(2^{n/2})$ etc.

- There are two classes of non polynomial time problems
  1) NP- hard
  2) NP-complete

- A problem which is NP complete will have the property that it can be solved in polynomial time iff all other NP – complete problems can also be solved in polynomial time.

NP –HARD AND NP – COMPLETE PROBLEMS
Basic Concepts

The class NP (meaning non-deterministic polynomial time) is the set of problems that might appear in a puzzle magazine: ``Nice puzzle.''

What makes these problems special is that they might be hard to solve, but a short answer can always be printed in the back, and it is easy to see that the answer is
correct once you see it.

Example... Does matrix A have an LU decomposition?

No guarantee if answer is ``no''.

NP –HARD AND NP – COMPLETE PROBLEMS
Basic Concepts

Another way of thinking of NP is it is the set of problems that can solved efficiently by a really good guesser.

The guesser essentially picks the accepting certificate out of the air (Non-deterministic Polynomial time). It can then convince itself that
it is correct using a polynomial
time algorithm. (Like a right-brain, left-brain sort of thing.)

Clearly this isn't a practically useful characterization: how could we build such a machine?

NP –HARD AND NP – COMPLETE PROBLEMS
Basic Conepts

Exponential Upperbound

Another useful property of the class NP is that all NP problems can be solved in exponential time (EXP).

This is because we can always list out all short certificates in exponential time and check all O(2nk) of them.

Thus, P is in NP, and NP is in EXP. Although we know that P is not equal to EXP, it is possible that NP = P, or EXP, or neither. Frustrating!

NP –HARD AND NP – COMPLETE PROBLEMS
Basic Conepts

NP-hardness

As we will see, some problems are at least as hard to solve as any problem in NP. We call such problems NP-hard.

How might we argue that problem X is at least as hard (to within a polynomial factor) as problem Y?

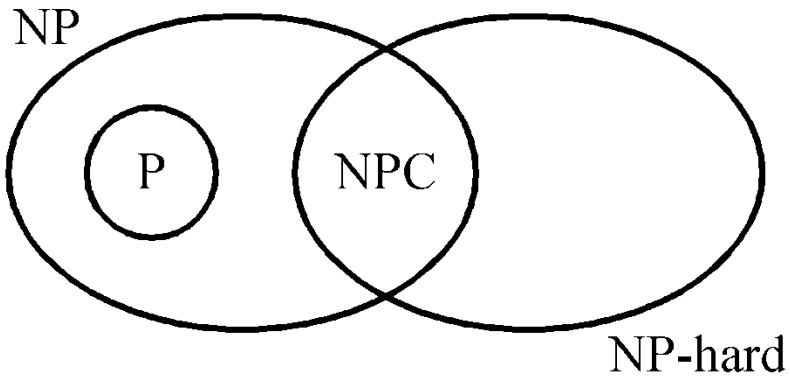
If X is at least as hard as Y, how would we expect an algorithm that is able to solve X to behave?

NP –HARD AND NP – COMPLETE PROBLEMS

- If an NP-hard problem can be solved in polynomial time, then all NP-complete problems can be solved in polynomial time.

- All NP-complete problems are NP-hard, but all NP-hard problems are not NP-complete.

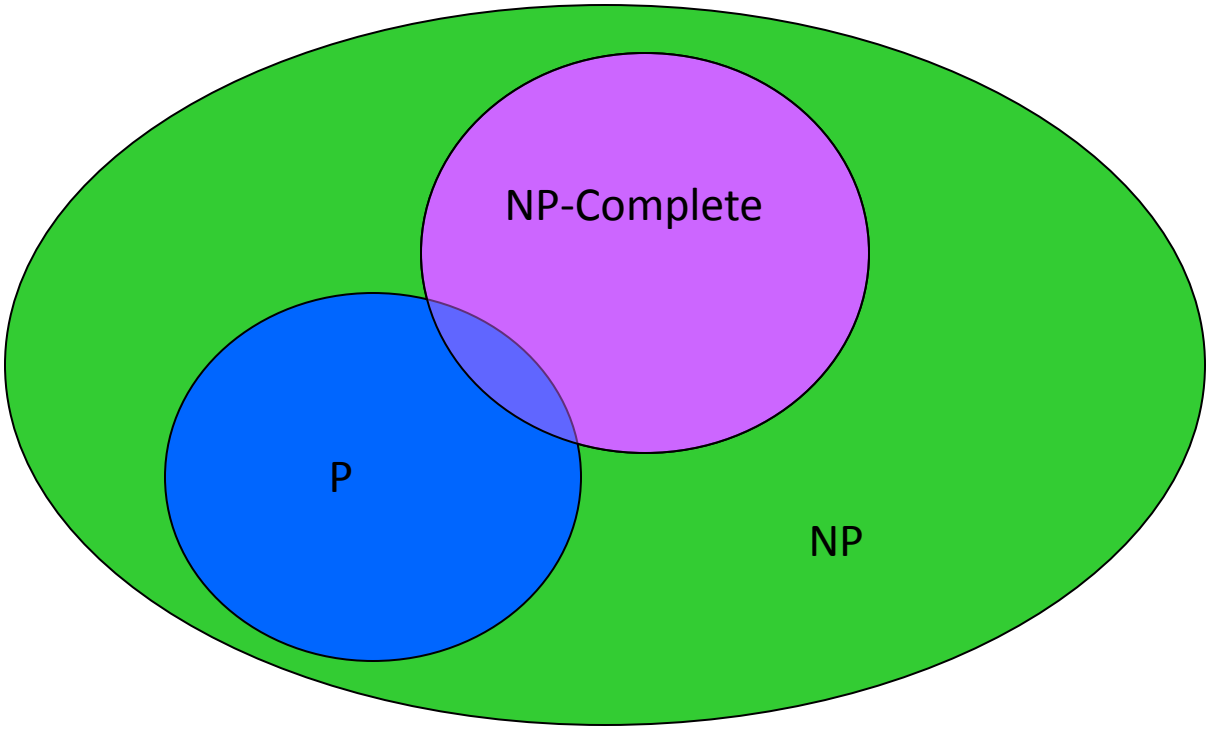
- The class of NP-hard problems is very rich in the sense that it contain many problems from a wide variety of disciplines.



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

NP-Completeness

- How we would you define NP-Complete?
- They are the “hardest” problems in NP



DETERMINISTIC and NONDETERMINISTIC ALGORITHMS

- Algorithms with the property that the result of every operation is uniquely defined are termed deterministic.

- Such algorithms agree with the way programs are executed on a computer.

- In a theoretical framework, we can allow algorithms to contain operations whose outcome is not uniquely defined but is limited to a specified set of possibilities.

Deterministic and Nondeterministic algorithms

- The machine executing such operations are allowed to choose any one of these outcomes subject to a termination condition.

- This leads to the concept of non deterministic algorithms.

- To specify such algorithms in SPARKS, we introduce three statements

  i) choice (s) …… arbitrarily chooses one of the elements of the set S.

  ii) failure …. Signals an unsuccessful completion.

  iii) Success : Signals a successful completion.

Deterministic and Nondeterministic algorithms

- Whenever there is a set of choices that leads to a successful completion then one such set of choices is always made and the algorithm terminates.

- A nondeterministic algorithm terminates unsuccessfully if and only if there exists no set of choices leading to a successful signal.

- A machine capable of executing a non deterministic algorithm is called a non deterministic machine.

- While non-deterministic machines do not exist in practice they will provide strong intuitive reason to conclude that certain problems cannot be solved by fast deterministic algorithms.

Nondeterministic algorithms

- A nondeterminstic algorithm consists of

  phase 1: guessing

  phase 2: checking

- If the checking stage of a nondeterministic algorithm is of polynomial time-complexity, then this algorithm is called an NP (nondeterministic polynomial) algorithm.
- NP problems : (must be decision problems)
  - e.g. searching, MST

    sorting

    satisfiability problem (SAT)

    traveling salesperson problem (TSP)

Example of a non deterministic algorithm

// The problem is to search for an element x //

// Output j such that A(j) =x; or j=0 if x is not in A //

j $\rightarrow$choice (1 :n)

if A(j) =x then print(j) ; success endif

print ('0') ; failure

complexity 0(1);

- Non-deterministic decision algorithms generate a zero or one as their output.
- Deterministic search algorithm complexity. Ω(n)

Deterministic and Nondeterministic algorithms

- Many optimization problems can be recast into decision problems with the property that the decision problem can be solved in polynomial time iff the corresponding optimization problem can .

EXAMPLE : [0/1 knapsack] :

- The decision is to determine if there is a 0/1 assignment of values to x_i $1 \leq i \leq n$ such that $\sum p_i x_i \geq R$, and $\sum w_i x_i \leq M$, R, M are given numbers p_i, $w_i \geq 0$, $1 \leq i \leq n$.

- It is easy to obtain polynomial time non deterministic algorithms for many problems that can be deterministically solved by a systematic search of a solution space of exponential size.

SATISFIABILITY

- Let $x_1, x_2, x_3 \ldots x_n$ denotes Boolean variables.
- Let $\overline{x}_i$ denotes the relation of x_i.
- A literal is either a variable or its negation.
- A formula in the prepositional calculus is an expression that can be constructed using literals and the operators and Λ or v.
- A clause is a formula with at least one positive literal.
- **The satisfiability problem is to determine if a formula is true for some assignment of truth values to the variables.**

SATISFIABILITY

- It is easy to obtain a polynomial time non determination algorithm that terminates successfully if and only if a given prepositional formula $E(x_1,x_2……x_n)$ is satiable.

- Such an algorithm could proceed by simply choosing (non deterministically) one of the 2^n possible assignments of truth values to $(x_1,x_2…x_n)$ and verify that $E(x_1,x_2…x_n)$ is true for that assignment.

The satisfiability problem

- The satisfiability problem
  - The logical formula :

    $x_1 \vee x_2 \vee x_3$

    $\& - x_1$

    $\& - x_2$

  the assignment :

  $x_1 \leftarrow F , x_2 \leftarrow F , x_3 \leftarrow T$

  will make the above formula true .

(-x_1, -x_2 , x_3) represents $x_1 \leftarrow F , x_2 \leftarrow F , x_3 \leftarrow T$

The satisfiability problem

- If there is at least one assignment which satisfies a formula, then we say that this formula is satisfiable; otherwise, it is unsatisfiable.

- An unsatisfiable formula :

$$
\begin{aligned}
& x_1 \vee x_2 \\
\& \; & x_1 \vee \neg x_2 \\
\& \; & \neg x_1 \vee x_2 \\
\& \; & \neg x_1 \vee \neg x_2
\end{aligned}
$$

The satisfiability problem

- Definition of the satisfiability problem:

  Given a Boolean formula, determine whether this formula is satisfiable or not.

- A literal : x_i or $-x_i$

- A clause : $x_1 \vee x_2 \vee -x_3 \equiv C_i$

- A formula : conjunctive normal form (CNF)

$$C_1 \& C_2 \& \ldots \& C_m$$

NP-HARD GRAPH AND SCHEDULING PROBLEMS

Some NP-hard Graph Problems :

The strategy to show that a problem L_2 is NP-hard is

(i) Pick a problem L_1 already known to be NP-hard.

(ii) Show how to obtain an instance I^1 of L_2 from any instance I of L_1 such that from the solution of I^1

- We can determine (in polynomial deterministic time) the solution to instance I of L_1.

NP-HARD GRAPH AND SCHEDULING PROBLEMS

(iii) Conclude from (ii) that $L_1 \alpha L_2$.

(iii) Conclude from (i),(ii), and the transitivity of α that

Satisfiability α L_1

$L_1 \alpha L_2$

$\therefore$ Satisfiability L_2

$\therefore$ L_2 is NP-hard

NP-HARD GRAPH AND SCHEDULING PROBLEMS

1. Chromatic Number Decision Problem (CNP)

- A coloring of a graph G = (V,E) is a function
  $f : V \rightarrow \{ 1,2, \ldots, k\} \ \forall\ i \in V$.

- If $(U,V) \in E$ then $f(u) \neq f(v)$.

- The CNP is to determine if G has a coloring for a given K.

- Satisfiability with at most three literals per clause α chromatic number problem.

$\therefore$ CNP is NP-hard.

NP-HARD GRAPH AND SCHEDULING PROBLEMS

2. Directed Hamiltonian Cycle (DHC)

- Let G = (V,E) be a directed graph and length n = 1V1
- The DHC is a cycle that goes through every vertex exactly once and then returns to the starting vertex.
- The DHC problem is to determine if G has a directed Hamiltonian Cycle.

Theorem : CNF (Conjunctive Normal Form) satisfiability α DHC

$\therefore$ DHC is NP-hard.

NP-HARD GRAPH AND SCHEDULING PROBLEMS

3. Travelling Salesperson Decision Problem (TSP) :

- The problem is to determine if a complete directed graph G = (V,E) with edge costs C(u,v) has a tour of cost at most M.

Theorem : Directed Hamiltonian Cycle (DHC) α TSP

- But from problem (2) satisfiability α DHC

  $\therefore$ Satisfiability α TSP

  $\therefore$ TSP is NP-hard.

NP-HARD SCHEDULING PROBLEMS

1. Sum of subsets

- The problem is to determine if $A=\{a_1,a_2,\ldots\ldots,a_n\}$ ($a_1,a_2,\ldots\ldots,a_n$ are positive integers) has a subset S that sums to a given integer M.

2. Scheduling identical processors

- Let P_i $1 \leq i \leq m$ be identical processors or machines P_i.
- Let J_i $1 \leq i \leq n$ be n jobs.
- Jobs J_i requires t_i processing time.

NP-HARD SCHEDULING PROBLEMS

- A schedule S is an assignment of jobs to processors.
- For each job J_i ,S specifies the time intervals and the processors on which this job is to be processed.
- A job can not be processed by more than one processor at any given time.
- The problem is to find a minimum finish time non-preemptive schedule.
- The finish time of S is $FT(S) = \max_{1 \leq i \leq m}\{Ti\}$
- Where T_i is the time at which processor P_i finishes processing all jobs (or job segments) assigned to it.

SOME SIMPLIIFIED NP-HARD PROBLEMS

- An NP-hard problem L cannot be solved in deterministic polynomial time.
- By placing enough restrictions on any NP hard problem, we can arrive at a polynomial solvable problem.

Examples.

(i) CNF- Satisfiability with at most three literals per clause is NP-hard.

(ii) If each clause is restricted to have at most two literals then CNF-satisfiability is polynomial solvable.

SOME SIMPLIIFIED NP-HARD PROBLEMS

(iii)Generating optimal code for a parallel assignment statement is NP-hard,

- however if the expressions are restricted to be simple variables, then optimal code can be generated in polynomial time.

(iv)Generating optimal code for level one directed a- cyclic graphs is NP-hard but optimal code for trees can be generated in polynomial time.

(v)Determining if a planner graph is three colorable is NP-Hard

- To determine if it is two colorable is a polynomial complexity problem. (We only have to see if it is bipartite)