Introduction

25 April 2025 01:26

Polynomial time algorithm :→ $O(n^k)$ for some +ve k

can all problems be solved in $O(n^k)$:→

NO , if a problem is solvable by TURINGS MACHINE (works on Von Neumann architecture not sure.)

⇒ solvable by computers.

TRACTABLE → easily solvable in polynomial time.

INTRACTABLE ⇒ require superpolynomial time. (hard)

NP COMPLETE PROBLEMS → unknown states

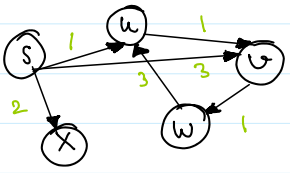
- i.) No polynomial time algo exists
- ii.) Now proved that there can't exist any solution.

SHORTEST V/S LONGEST PATH.

shortest from source → Bellman ford algorithm, Dijkstra's algorithm.

But longest path can be hard.

even finding if shortest path exist with given number of edges is NP



NP → whether ∃ a simple path from v → x. having atleast 3 edges.

EULER TOUR / TRAIL & HAMILTONIAN GRAPH

* Euler Tour → each edge exactly once.
  - can be solved using strongly connected components in $O(E)$ time.
  - Category : P

* Hamiltonian cycle :→ if each vertex only once, start = end. (cycle)
  - Category : NP complete

#2CNF , 3CNF → conjective Normal form.

eg:→ 2CNF = $(x_1 \vee x_2) \wedge (\bar{x_1} \vee x_3)$

3CNF = $(x_1 \vee x_2 \vee \bar{x_3}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\bar{x_1} \vee x_2 \vee \bar{x_3})$

Conj of clauses

Clauses

→ No. of literals = 3

satisfiable if value = 1.

Variables = $\{x_1, x_2, x_3\}$

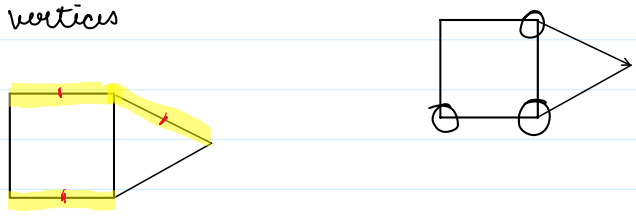
Literals = $\{x_1, x_2, \bar{x_3}\}$

2 CNF → Psolvable
3 CNF → NP complete.

Edge cover / Vertex cover :→

↓ Min no of edges that covers all vertices

↓ Min no of vertices that covers all edges



classes P, NP and NPC.

P :→ all the problems solvable in polynomial time. are in P.

NP :→ all the problems verifiable in polynomial time

↓

can be checked if provided a "certificate"

↳ a solution.

eg:→ In 3 CNF if we are given a sequence, we can check in polynomial time.

In Hamiltonian graph, certificate → set of vertices

NPC :→ it is in NP and is as hard as any problem in NP

If NPC gets a poly time sol^n → all NP problems has a polynomial time solution.

Showing problems to be N-P Complete

* show how hard the problem is rather than easy

* Show how hard the problem is rather than easy

* 3 concepts :→

  i.) decision problem V/S optimization problem

  ii.) Reducibility

  iii.) A first NPC problem.

Optimization Problem → decision problem

Optimization Problem → value required

eg :→ min no of edges in single pair – shortest path. in an undirected graph, unweighted graph

Decision problem :→ ans 'yes' or 'no' (1 or 0)

eg :→ whether exist a path, with 'k' no of edges between source & dest.

NPC problems → confined to decision problems

- So we convert OPt. Prob → Decision Problem.
- Helps in proving hardness of the problem.

  since decision problem is easier or "no harder" than optimization problem.

∴ if decision problem is hard, optimization will also be atleast hard.

Reduction :→

for 2 decision problems, we use Reductions

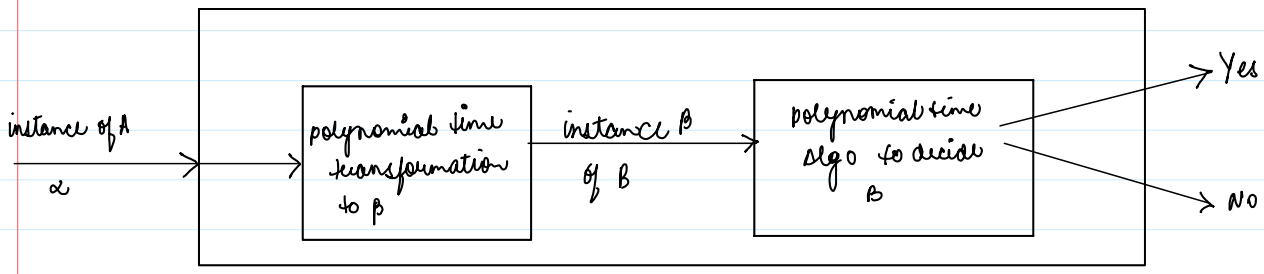
INSTANCES → given inputs in a problem. eg :→ graph, vertices u and v etc.

Problem A :→ we would like to solve in poly time.

Problem B :→ already know how to solve in poly time.

∴ Instance α of A —Reduced→ Instance β of B. if :→

- The transformation takes polynomial time.
- The answers are same. A = 1 iff B = 1.



Polynomial time Algo to decide A.

* using easiness of B to prove easiness of A

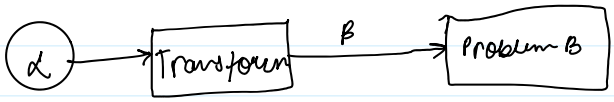
* can also use to show that problem is NP complete.

showing B is NPC

→ suppose a problem A (we know, no poly time sol^n exist suppose)

→ and we somehow transform A → B in polynomial time. somehow first do this

→ so B can't be solved, else A would have been solved.



Instance of an NPC problem

using hardness of 'A' to prove hardness of 'B'.

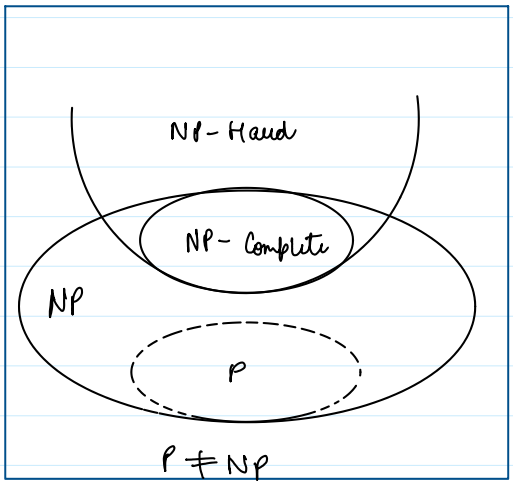
A first NP complete Problem :→

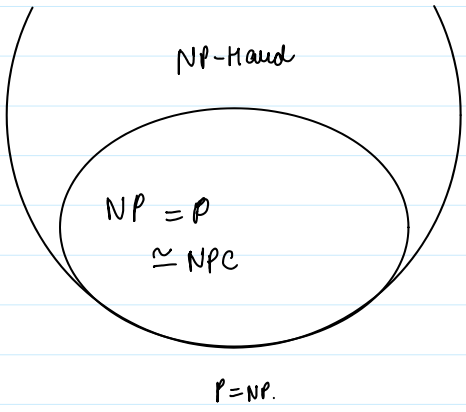
Since we need A (an NPC) to prove hardness of B. We need an (A)

first NPC → 3SAT

this A will be the first NPC.

Classes





A formal language framework

Alphabets Σ be a set of alphabets. eg :→ $\{0,1\}$.

Language $L = \{01, 001, 0001 \ldots\}$ is a language defined on Σ ,

empty string → ε

empty lang → ϕ.

all possible string set of $\Sigma = \Sigma^*$

eg :→ $\Sigma^* = \{0, 01, 10, 11, 100 \ldots\}$ set of all binary.

UNION and INTERSECTION → just like set operation. * COMPLIMENT :→

$\overline{L} = \Sigma^* - L$

* CONCATENATION :→ say $L_1 \& L_2$.

$L_1 L_2 = \{(x,y) \mid x \in L_1, y \in L_2\}$.

* CLOSURE OR KLEENE STAR :→

$L^* = \{\varepsilon\} \cup L \cup L^2 \cup L^3 \cup L^4 \ldots \ldots L^k$

where L^k is the k times concatenation of L to itself $\{L^2 = L \times L\}$ x → concatenation.

Usage :→ a decision problem Q → based on instances. (only decision problem)

set of instances = set of $\Sigma = \Sigma^*$.

where $\Sigma = \{0,1\}$

Q → characterized by instances which gives ans '1'.

so those instances in a language

$L = \{x \in \Sigma^* \mid Q(x) = 1\}$.

eg :→ decision problem Path has corresponding language

PATH = $\{(G, u, v, k)$: $G = (V, E)$ is an undirected graph.

$u, v \rightarrow$ vertices.

$k \rightarrow$ no. of edges, $k \geq 0$. $\}$

PATH → decision problem, language both under same Name.

* A language

i) accepts a string $x \in \{0,1\}^*$ if Algo outputs '1'.

ii) Rejects a string $x \in \{0,1\}^*$ if algo outputs '0'

eg :→ A Tentative definition of class P : class → set of languages

class P = $\{ L \subseteq \{0,1\}^* : \exists$ an algo that decides L in polynomial time $\}$

Verification argument

A 2 argument problem, $x \rightarrow$ normal input, $y \rightarrow$ certificate } accepted if $A(x, y) = 1$.

Complexity class NP.

A class verifiable in polynomial time.

$L = \{ x \in \{0,1\}^* : \exists\, y$ with $|y| = O(|x|^c)$ ← verifying with y

such that $A(x, y) = 1 \}$ ← if verified.

$L_1 \leq_P L_2$, L_1 is not more than polynomial factor harder L_2

NP complete class :→

$L \subseteq \{0,1\}^*$ is NP complete if :→

i) $L \in NP$

ii) $L' \leq_P L$ for every $L' \in NP$

If only (ii) satisfied then NP-Hard.

THE CLIQUE PROBLEM

The clique of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that each and every pair of V' is connected by an edge.

∴ V' forms complete subgraph of G.

.. V' forms complete subgraph of G.

* K-partite graph

Prob → Find max size of clique in a graph.

CLIQUE = $\{\langle G, K\rangle$, where G is a graph containing clique of size K$\}$

corresponding decision problem → clique decision prob (C.D.P)

CNF - satisfiability → NP complete (COOK LEVIN THEOREM)

Koup reduced CNF-SAT to CDP.

3CNF to CDP

Let 3CNF with K clauses.

Let $\varphi = C_1 \wedge C_2 \wedge C_3 \ldots \wedge C_K$

and $r = 1, 2, 3, \ldots K$

φ satisfiable iff G has a clique of size k

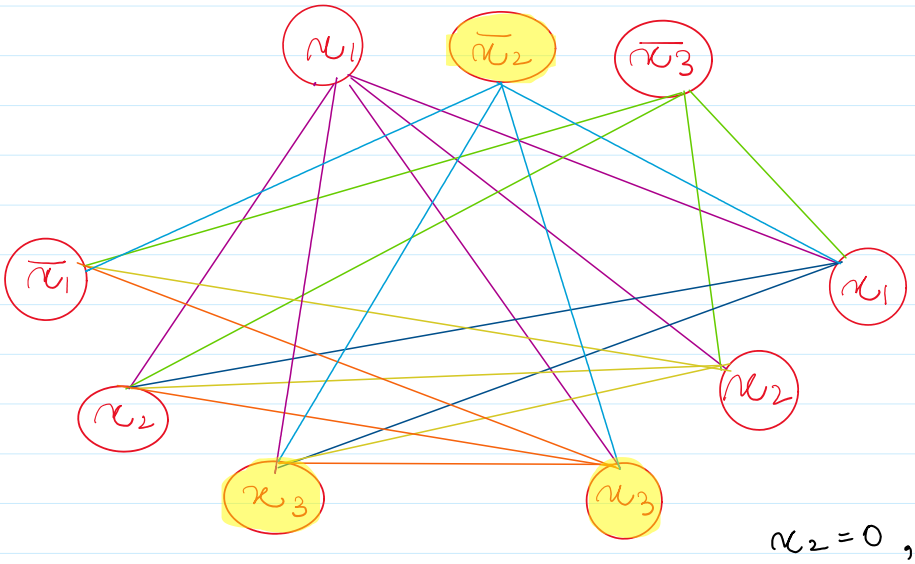
* * edge b/w x_1 & x_2 if

i) x_1 & x_2 are in diff tuplets.

ii) Their corresponding literals are consistent, i.e.

$x_1 \neq \neg x_2$

Eg:→ $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1, x_2, x_3) \wedge (x_1 \vee x_2 \vee x_3)$



$x_2 = 0, x_3 = 1$

satisfies the equation

* No. of literals = no. of edges → $O(n^2)$

* Since any CNF → CDP

any α of CNF → β of CDP → CDP is NP-hard.

3NP - complete problems.

* Independent set :→ $S \subseteq V$ such that no pair of S has an edge in between.

* Max independent set → Max no. of element in S possible from a graph
  → say S_{max}.

  Then Min Vertex Cover = $V - S_{max}$.

* Complement of independent set produces clique, say G_c.

Max Independent set is NP-Hard.

any instance of clique → instance of independent set.

clique: NP Hard

independent set: NP Hard

Vertex-cover is NP Hard

any instance of independent set S → instance of vertex-cover (V-S)

independent set: NP Hard

vertex-cover (V-S): NP Hard.

Summary :→

NP complete

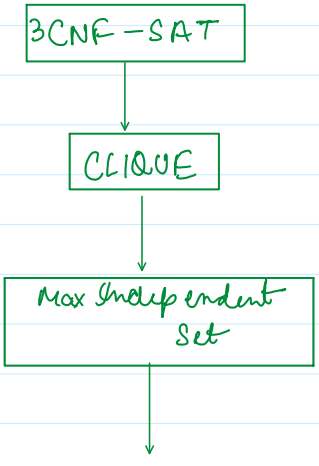
→ Longest Path
→ Hamiltonian graph
→ 3CNF

P soluable

→ Shortest Path.
→ Eulerian graph
→ 2CNF

NP Hard

→ CDP (Clique decision prob)
→ Independent set
→ Vertex cover.



Vertex cover