

P, NP Theory - II

Prof Devesh C Jinwala, PhD
Professor, Department of Computer Engineering
<http://www.svnit.ac.in/dcj/>

Indian Institute of Technology Jammu

Contents

- What is an Algorithm ? Solving problems algorithmically
- Classifying the problems
- A Motivating Example
- Some Hard Problems
- Reductions
- Non-determinism
- Class P, NP
- NP Complete Problems
- Concluding Remarks

How to we deal with the hard problems ?

If we are not able to solve these problems,

at least

we would be happy

if we were able to verify a given solution

to be optimal or not.....

in polynomial time ???

Nondeterminism

- Deterministic Algorithms

- determinism - the property that the result of every operation is uniquely defined.
- that can be implemented on the actual machine

- Deterministic Polynomial Algorithms

- Deterministic algorithms that can be solved reasonably fast
 - whose complexity is bounded by $O(n^c)$, c a positive constant, n input size as seen before.

Nondeterminism

- Theoreticians treat even n^{million} as solvable reasonably fast
 - even while practically their execution time is very long
- That is the reason why
 - we have to conceptually conceive non-determinism.

Understanding Nondeterminism

- A nondeterministic algorithm can be understood using three new functions

- Choice(S)

- Takes a set S as input & arbitrarily chooses one of the elements of the set S
 - e.g. $x=\text{choice}(l,n)$
 - x is assigned any one of the integers in $[l \dots n]$ can be qualified in any manner we desire...
 - takes time $O(1)$

- Failure()

- signals an unsuccessful completion of the algorithm
 - takes time $O(1)$

- Success()

- signals a successful completion of the algorithm
 - takes time $O(1)$

Which function(s)
amongst these
is/are
deterministic ?
Nondeterministic ?

Nondeterministic Search Algorithm

- Problem
 - Searching for an element x in a given set of elements $A[n]$, $n \geq 1$.
 - Output An index j , such that $A[j]=x$ OR $j=0$ if x is not in A .
 - Design the algorithm **NonDeterministicSearch (x, A)**

```
Algorithm NonDeterministicSearch(x, A)
```

```
1. j = Choice(1,n);  
2. if A[j]=x, then write(j); Success();  
3. write(0); Failure();`
```

- Algorithm has nondeterministic complexity $O(1)$. Why ?

Nondeterministic Sorting Algorithm

■ Problem

- Sorting $A[i]$, $1 \leq i \leq n$, an unsorted array of positive integer
- Output $A[i]$, sorted in nondecreasing order

```
Algorithm NonDeterministicSorting(A, n)
```

```
1. for i= 1 to n, B[i]=0;  
2. for i= 1 to n {  
3.     j=Choice(1,n);  
4.     if B[j] ≠ 0 then Failure();  
5.     B[j] = A[i];  
6. }  
7. for i=1 to n-1 //Verify the order  
8.     if B[i]>B[i+1] then Failure();  
9. write B[1..n];  
10.Success();
```

What is the overall Nondeterministic Complexity ??

Why Nondeterminism ?

- Nondeterministic algorithms in computational complexity theory.
 - are typically studied only on a theoretical framework.
 - outcome of an operation is not uniquely defined.....
 - but is limited to specified sets of possibilities. i.e. these can allow for multiple choices to be available for the next step of computation.....
 - at every step, creating many possible computational paths.
 - these algorithms do *not* arrive at a solution for every possible computational path,
 - but are guaranteed to arrive **at a correct solution** for some path i.e., **if the right choices** are made underway.

.....contd

Why Nondeterminism ?.....

- Nondeterministic algorithms in computational complexity theory.
 - the choices can be interpreted as guesses in a search process.
 - a large number of real-life problems can be naturally stated/solved in this way
 - hence nondeterminism can help define commonality amongst the problems.

Problem Boolean Satisfiability

- Input
 - A boolean formula F in CNF
- Goal
 - Check if F is satisfiable or not.
 - e.g. if $F = (x_1 + x_2)$ can we assign at least one set of values to the literals of the formula so that $F=1$.
- How to solve this problem deterministically ?
 - What could be the Brute-force approach ?
 - Time complexity ??
 - $O(2^n |F|)$
 - Cannot do any better than that.
- Hence, the nondeterministic solution.....

Nondeterministic Boolean Satisfiability

- Phase1
 - 1. for i=1 to n do
 - 2. use Choice() to determine the value of x_i 's that will satisfy F
- Phase2
 - 1. Check if F is satisfied under the above criterion
- Runtime
 - Phase1..... $O(1) * n$ i.e. $O(n)$
 - Phase2..... $O(|F|)$
 - Total..... $O(n + |F|)$

Problem k -Clique

- Input $G(V, E)$, k
- Output “Yes”, if G has a clique of size k and “No” otherwise
 - e.g.

k-Clique Deterministic approach

- Input $G(V, E)$, k
- Output “Yes”, if G has a clique of size k and “No” otherwise
- Naïve/Deterministic approach
 - For every subset of k nodes, check if these k nodes form a clique
 - Time complexity
 - $\binom{n}{k} \binom{k}{2}$
 - i.e. $O(n^k k^2)$
- The issue is can we do better than this ?

k-Clique Non-deterministic approach

Phase 1

Using the Choice(), we investigate whether "for every node in the graph is this node a k-clique?"

Guess k nodes that form a clique

Phase 2

Verify whether the above nodes indeed form a k-clique.

Time Complexity

- Phase I takes k units of time.....
- Phase two takes (k choose 2 i.e.) units of time
- $O(k + (k \text{ choose } 2))$ i.e. $O(k^2)$.

Complexity Classes

- A Complexity Class

- Is a collection of **decision problems** (???) all of which can be solved in SAME resource bounds
 - i.e. within same time, space or randomness....

- The question is

Why do we bring in the discussion and then focus on decision problems and not on search problems ?

Different versions of problems

- Combinatorial Optimization problems
 - Answer is a number
- Decision problems
 - Answer is yes or no
- Construction problems
 - Answer is some object (set of vertices, function, ...)

Classes of Problems

- Combinatorial Optimization problems
 - how to we define these ?
- Output

*Our Focus on
decision problems*

Classes of Problems

- Decision problems
 - variation of optimization problems
 - answer yes/no to a question
- Statement of a decision problem has two parts
 - instance description
 - question with variables defined in instance description with actual yes/no asked
- Many problems will have decision and optimization versions.
- For example ??

Illustrating the difference

■ Clique

■ optimization

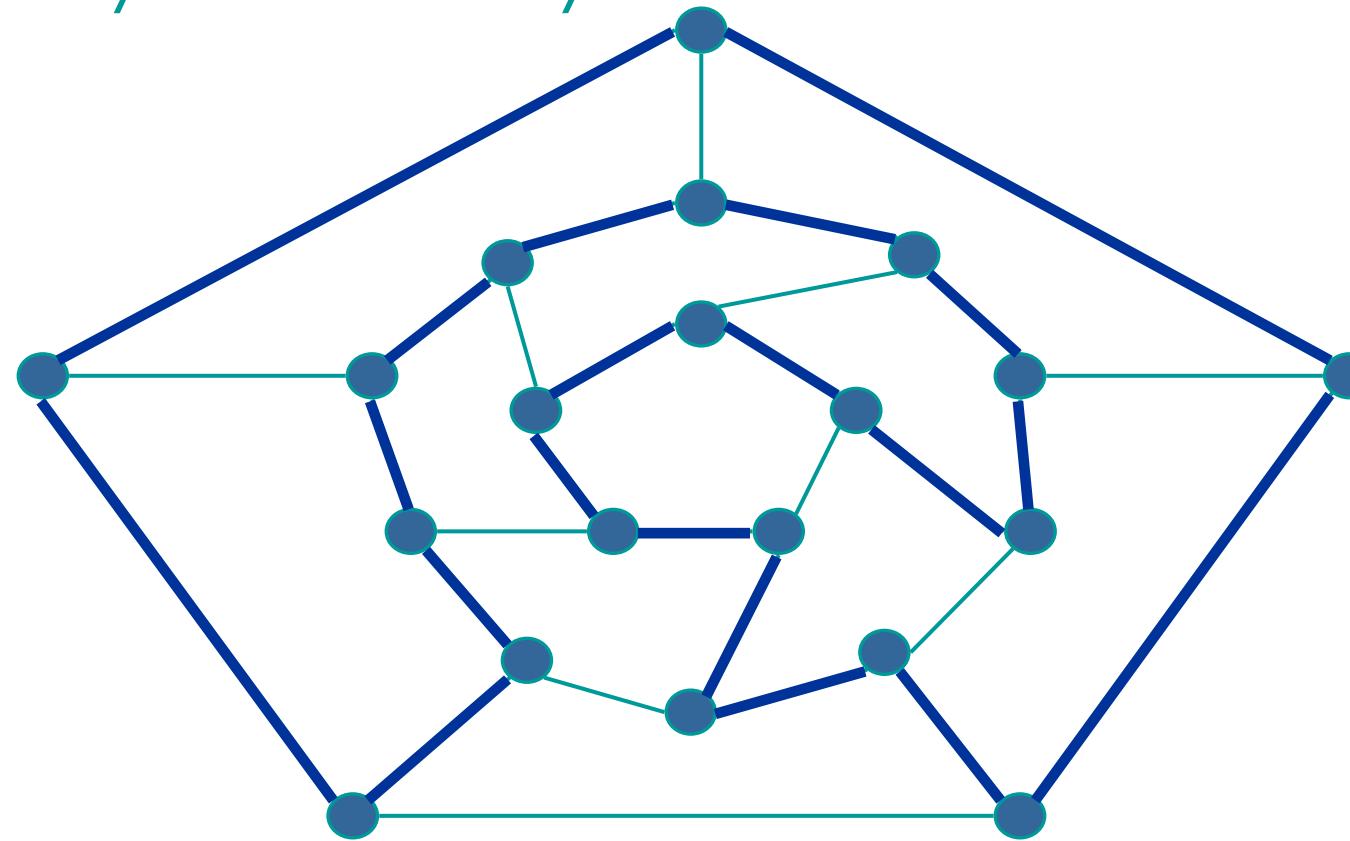
- instance For an undirected graph $G = (V, E)$
- question does G contain a clique of k vertices ?
 - say running time $O(k^2n^k)$.
 - Is it polynomial ?depends on the nature of k

■ Decision

- instance For an undirected graph $G = (V, E)$ and an integer k
- question does G contain a clique of k vertices ?
 - say running time $O(k^2n^k)$.
 - Is it polynomial ?depends on the nature of k

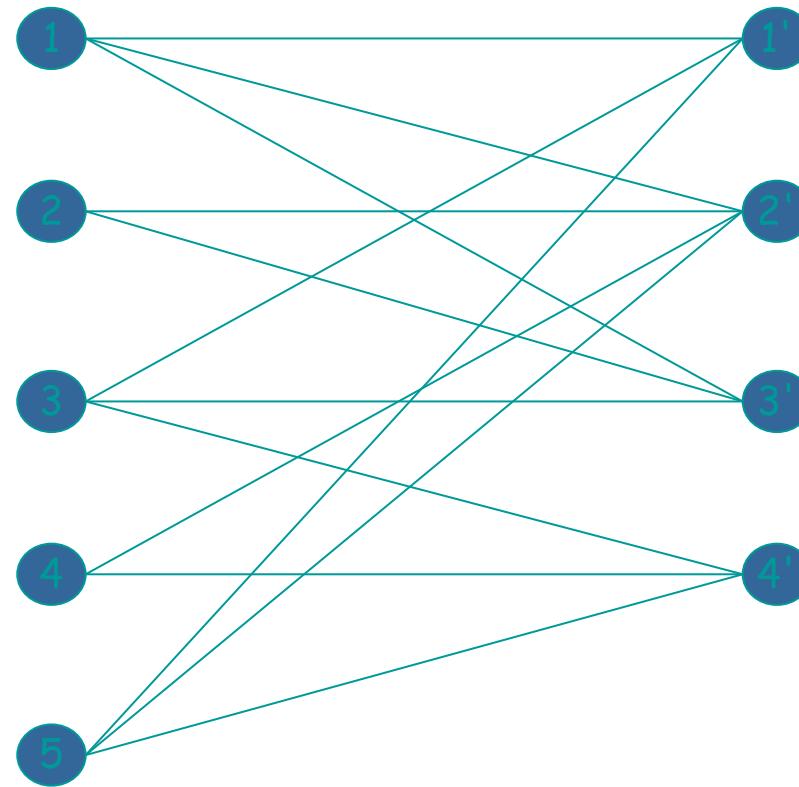
Hamiltonian Cycle

- given an undirected graph $G = (V, E)$, does there exist a simple cycle Γ that contains every node in V exactly once ?



Hamiltonian Cycle (contd)

- HAM-CYCLE given an undirected graph $G = (V, E)$, does there exist a simple cycle Γ that contains every node in V .



Traveling Salesperson problem

- Minimum Tour problem
 - Optimization problem
 - Given a complete, weighted graph find a minimum-weight Hamiltonian cycle
 - Decision problem
 - Given a complete, weighted graph and an integer k , is there a Hamiltonian cycle with total weight at most k ?

Contents

- What is an Algorithm ? Solving problems algorithmically
- Classifying the problems
- A Motivating Example
- Some Hard Problems
- Reductions
- Non-determinism
- **Class P, NP**
- NP Complete Problems
- Concluding Remarks

Class P

■ P

- the class of decision problems that are solvable deterministically in $O(p(n))$, where $p(n)$ is a polynomial on n
- the class of decision problems that are polynomially bounded.

Solving the Decision Problems

- Can every decision problem be solved in polynomial time ?
 - Obviously not !!
 - Some decision problems cannot be solved at all by any algorithm
 - Undecidable problems e.g. The Halting problem
 - Proving that the Halting problem can not be solved at all.....
 - Are there any decidable but intractable problems ?
 - Yes, they are

Solving the Decision Problems...

- However, as seen before.....the important aspect is
 - that there are decidable problems for which neither the polynomial time algorithm has been found nor impossibility of doing so can be proved

Class NP

- NP – Nondeterministic Polynomial and not Nonpolynomial !!
 - the set of all the **decision problems** that **can be solved** using nondeterministic algorithms in polynomial time
 - Loosely
 - the class of decision problems for which a given proposed solution for a given input can be checked quickly in polynomial time.

Class NP

■ Recollect that

- A nondeterministic algorithm is a two stage procedure that takes as input an instance i of a decision problem
 - Guessing (nondeterministic) stage
 - an arbitrary string S is generated that can be thought of as a candidate solution to the given instance i
 - Verification (Deterministic) stage
 - a deterministic algorithm takes both i and S as its input and outputs yes if S represents a solution to instance i ; otherwise returns no.

Class NP (contd)

- Thus, NP can also be thought of as
 - the collection of problems that have polynomially i.e. efficiently **verifiable** solutions
 - that can be solved in polynomial time on a machine that can pursue infinitely many paths of the computation in parallel

Certification in NP

- def

- Algorithm $C(s, t)$ is a **certifier** for problem X if for every string s , $s \in X$ there exists a string t such that $C(s, t) = \text{yes}$.

← "certificate" or "witness"
↑

- Certification algorithm intuition.

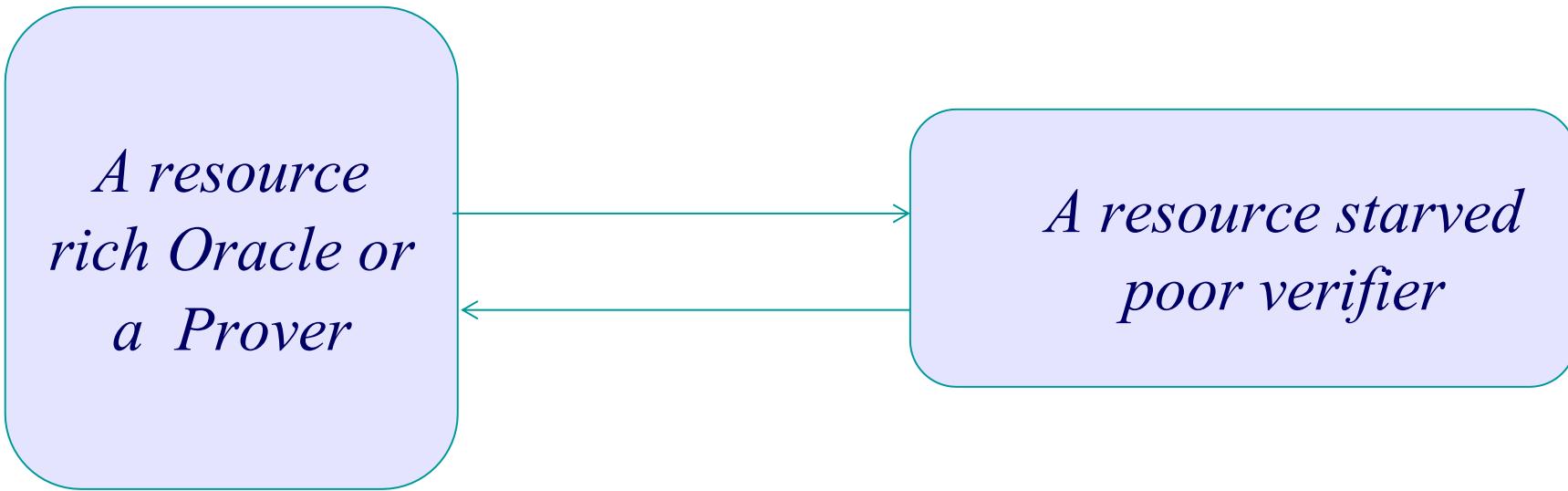
$C(s, t)$ is a poly-time algorithm and
 $|t| \leq p(|s|)$ for some polynomial $p(\cdot)$.

- Certifier views things from "managerial" viewpoint.
- Certifier doesn't determine whether $s \in X$ on its own; rather, it checks a proposed proof t that $s \in X$.

- NP

- Decision problems for which there exists a **poly-time certifier**.

Prover Verifier View



Certifiers-Certificates Composite

- COMPOSITES. Given an integer s , is s composite?
- Certificate
 - A nontrivial factor t of s . Note that such a certificate exists iff s is composite. Moreover $|t| \leq |s|$.
- Certifier
- Instance. $s = 437,669$.
- Certificate. $t = 541$ or 809 .
- Conclusion. COMPOSITES is in NP.

```
boolean C(s, t) {  
    if (t ≤ 1 or t ≥ s)  
        return false  
    else if (s is a multiple of t)  
        return true  
    else  
        return false  
}
```

$$437,669 \leftarrow 541 \times 809$$

Certifiers-Certificates 3-Satisfiability

- SAT
 - Given a CNF formula Φ , is there a satisfying assignment?
- Certificate
 - An assignment of truth values to the n boolean variables.
- Certifier
 - Check that each clause in Φ has at least one true literal.
- Ex.

instance s $(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3} \vee \overline{x_4})$

$$x_1=1, x_2=1, x_3=0, x_4=1$$

certificate t

- Conclusion. SAT is in NP.

Certifiers-Certificates Hamiltonian Cycle

- HAM-CYCLE
 - Given an undirected graph $G = (V, E)$, does there exist a simple cycle C that visits every node?
- Certificate
 - A permutation of the n nodes.
- Certifier
 - Check that the permutation contains each node in V exactly once, and that there is an edge between each pair of adjacent nodes in the permutation.
- Conclusion. HAM-CYCLE is in NP.

Certifiers-Certificates k-CLIQUE

- A graph $G=(V, E)$ is a Clique if every two nodes in V are connected by an edge
- K-CLIQUE
 - Given a graph $G=(V,E)$, prove that k-clique is in NP.
- Proof
 - define CLIQUE = { $\langle G, k \rangle | G$ is an undirected graph with a k -clique}
 - Certificate
 - the collection of k vertices of the graph
 - Certifier
 - Take as input some string c as $(\langle G, k \rangle, c)$
 - Test whether c is a set of k nodes in G
 - Test whether G contains all edges connecting nodes in c
 - If so, proved.

Certifiers-Certificates SUBSET SUM

- Tutorial Assignment.....

Revisiting P, NP, EXP

- P
 - Decision problems for which there is a **poly-time algorithm**.
- EXP
 - Decision problems for which there is an **exponential-time algorithm**.
- NP
 - Decision problems for which there is a **poly-time certifier**.

Revisiting P, NP, EXP (contd)

- Claim. $P \subseteq NP$.

- Proof

- Consider any problem X in P .

- By definition, there exists a poly-time algorithm $A(s)$ that solves X .
 - Certificate $t = \varepsilon$, certifier $C(s, t) = A(s)$.
 - Therefore, the fact.



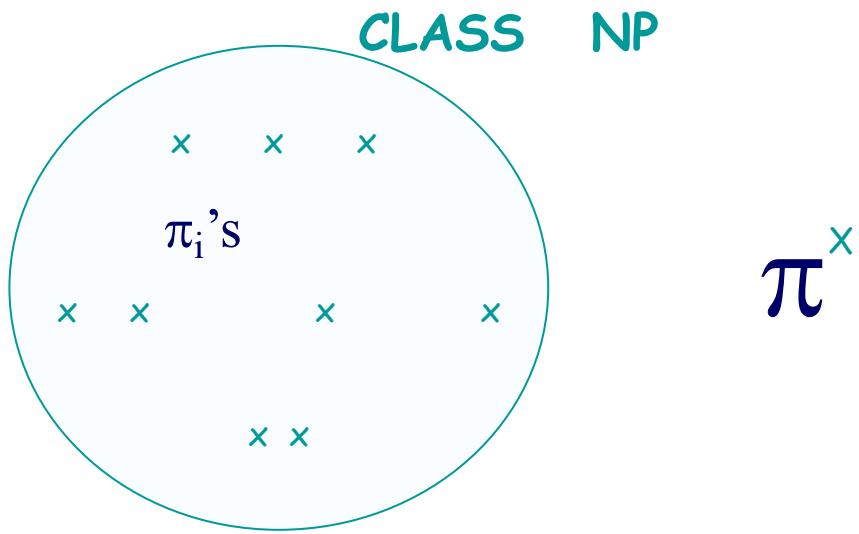
Remember, we said an Algorithm $C(s, t)$ is a **certifier** for problem X ; iff for every string s , $s \in X$ there exists a string t such that $C(s, t) = \text{yes}$.

Revisiting P, NP, EXP (contd)

- Claim. $\text{NP} \subseteq \text{EXP}$.
- Proof
 - Consider any problem X in NP.
 - By definition, there exists a poly-time certifier $C(s, t)$ for X .
 - To solve input s , run $C(s, t)$ on all strings t with $|t| \leq p(|s|)$.
 - Return yes, if $C(s, t)$ returns yes for any of these

NP-Hard

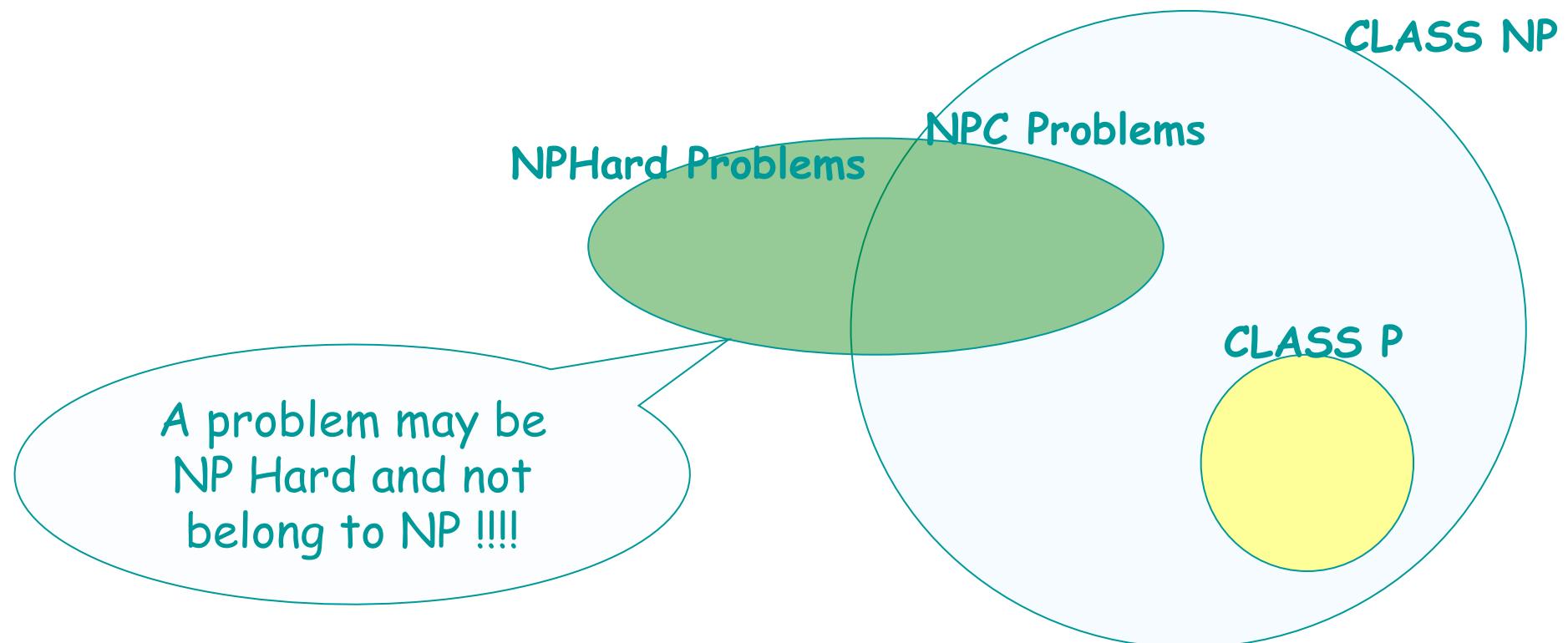
- A problem π is NP-Hard if for every problem $\pi_i \in \text{NP}$, π_i reduces to π .
 - i.e. π is at least as difficult as π_i OR
 - i.e. π is at least as difficult as every problem in class NP.
- Graphically,



Here, every π_i in CLASS NP is polynomially reducible to a problem π that is outside NP,

NP Complete Problems

- A problem π is NPC if π is NP-Hard and $\pi \in \text{NP}$.
- Thus, based on the above we can conceive of the Venn-diagram as below



NP Completeness

- Cook and Levin 's theory
 - There are certain problems in NP
 - whose individual complexity is related to that of the entire class.
 - i.e. if a poly time algorithm were to exist for any problem in this class, all problems in NP would be polynomial time solvable.
- These problems are called NP-complete problems.

NP Completeness

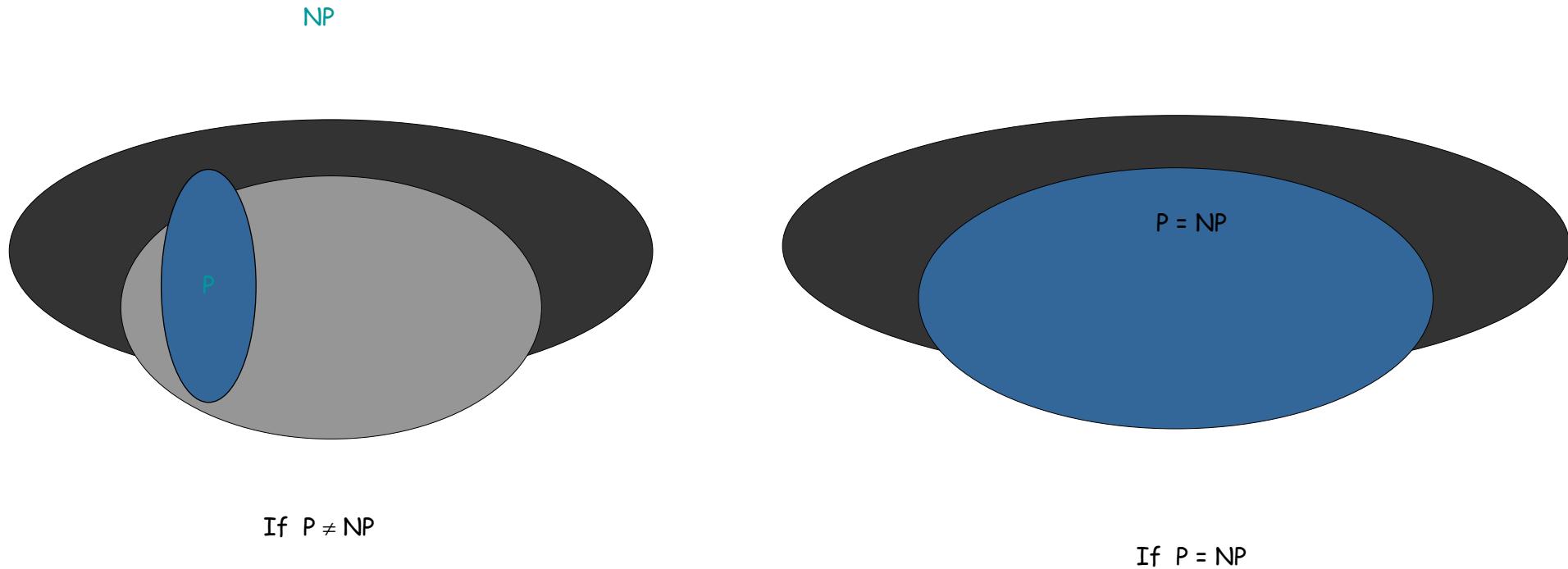
- A problem P is NP-complete if one can prove that
 - P is in NP, and
 - show that there is some other problem Q - known to be an NPC – is poly-time reducible to the problem P.
 - the hard part of that was proving the *first* example of an NP-complete problem
 - that was done by Steve Cook in Cook's Theorem

Theorem $P = NP$

- Fact Let π be any NPC problem. If $\pi \in P$, then $P = NP$.
- Proof
 - Let π_1 be any problem in NP.
 - Now, if π_1 is an NPC problem, then by definition of NPC, π_1 is also NP-Hard and so $\pi_1 \alpha \pi$.
 - Now, if $\pi \in P$, then since $\pi_1 \alpha \pi$, π_1 is also in P
 - Therefore, $P = NP$.
- However, the belief is that if π is any NPC problem, then $\pi \notin P$.
- So, $P \neq NP$.

The Main Question P Versus NP

- Does $P = NP$? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]
 - Is the decision problem as easy as the certification problem?



The Main Question P Versus NP...

- Does $P = NP$? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]
 - Is the decision problem as easy as the certification problem?
- If yes
 - Efficient algorithms for 3-COLOR, TSP, FACTOR, SAT, ...
- If no
 - No efficient algorithms possible for 3-COLOR, TSP, SAT, ...
- Consensus opinion on $P = NP$? Probably no.

Theorem

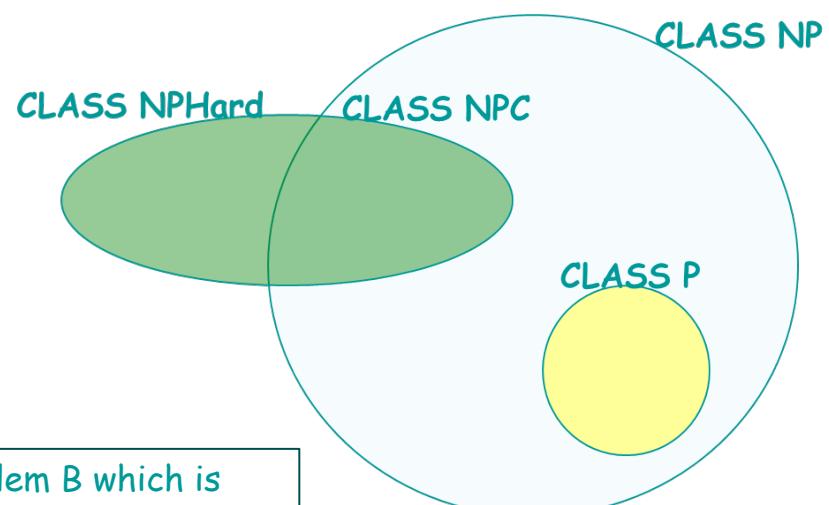
- Prove that if a problem B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-Complete.

A problem Π is NPC if it belongs to the class NP and for every problem $\Pi_1 \in NP$, Π_1 reduces to Π .

- Proof

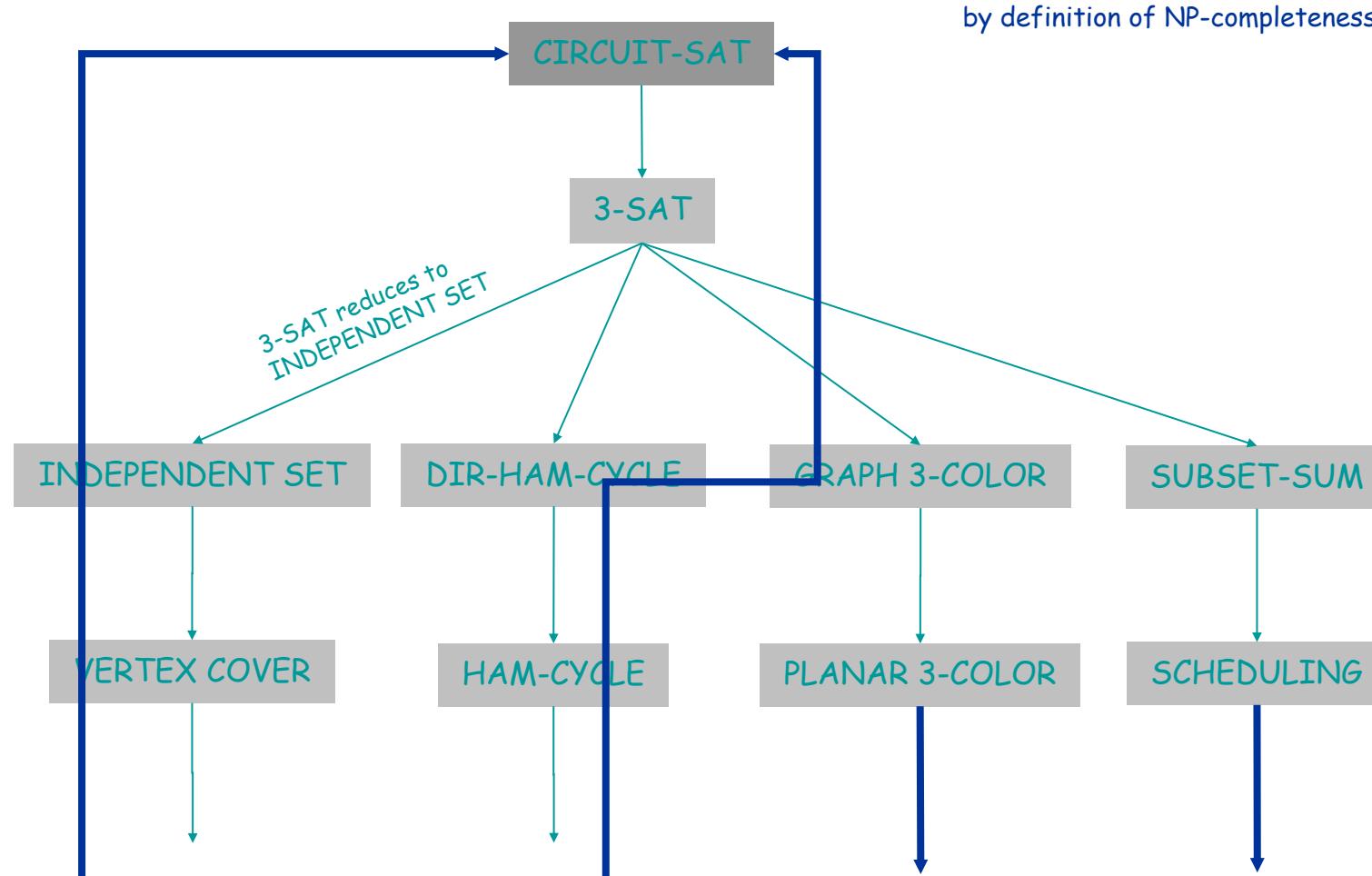
- We are given C is in NP and
 - to prove that C is NPC we must now show that every problem in class NP is polynomial time reducible to C.
- We are given that B is NP-complete.
 - That is, every problem in NP is \leq_P to B
 - But, we are given that $B \leq_P C$
 - Therefore, every problem A in NP is $\leq_P C$
 - Therefore, C is NP-complete

A problem A is NP-complete if one can prove that A is in NP, and there is some problem B which is already proven to be an NPC is poly-time reducible to a problem A.



NP-Completeness

- Observation. All problems below are NP-complete and polynomial reduce to one another!



Some NP-Complete Problems

- Six basic genres of NP-complete problems and paradigmatic examples.
 - Packing problems SET-PACKING, INDEPENDENT SET.
 - Covering problems SET-COVER, VERTEX-COVER.
 - Constraint satisfaction problems SAT, 3-SAT.
 - Sequencing problems HAMILTONIAN-CYCLE, TSP.
 - Partitioning problems 3D-MATCHING 3-COLOR.
 - Numerical problems SUBSET-SUM, KNAPSACK.
 - Practice. Most NP problems are either known to be in P or NP-complete.
- Notable exceptions. Factoring, graph isomorphism, Nash equilibrium.

Concluding Remarks

References

- Text by CLRS
- Text by Garey and Johnson
- Text by Kleinberg Tardos
- NPTEL Videos
- Special gratitude to Prof Rajsekaran, IUCEE Course Instructor at Infosys, July 2009.

To Teach is to Learn twice
!!!!

Hence, Thank you !!!