UNIT-IV

Unit IV	**Intractable Problems and NP-Completeness**

Time-Space trade off,

Tractable and Non-tractable Problems,

Polynomial and non-polynomial problems, deterministic and non-deterministic algorithms, P-class problems,

NP-class of problems, **Polynomial problem reduction,**

NP complete problems- **Vertex cover** and **3-SAT**

And

NP hard problem - **Hamiltonian cycle**

Book Chapters

Content	Book Name	Page No	Additional
P-class problems, NP-class of problems, Polynomial problem reduction NP complete problems- vertex cover and 3-SAT and NP hard problem - Hamiltonian cycle.	Fundamentals of Computer Algorithms (Horowitz and Sahani)	Page No.514 to 536	Design and Analysis of Algorithms -Parag Dave (Page no.617 to 631)

Basic Concepts Required in Unit IV

- Polynomial Time Algorithms (Deterministic),Exponential Algorithms (Deterministic)
- Tractable Problems : problem that is solvable by a polynomial-time algorithm.
  - The upper bound is polynomial
- Intractable Problems: problem that cannot be solved by a polynomial-time algorithm.
  - The lower bound is exponential
- Polynomial, Non-Polynomial Problems [Decision problems and Optimization (/ Search) Problems][Sample problems: SAT, Node (Vertex) Cover, Max Clique, Hamiltonian Cycles]
- Non-Deterministic Algorithms (Choice(S), Success, Failure)
- P-Class, NP-Class of problems
- Reductions, Polynomial-Time Reductions
- NP-Hard problems
- NP-Complete problems

Unit IV	**Intractable Problems and NP-Completeness**

Time-Space trade off,

Tractable and Non-tractable Problems,

Polynomial and non-polynomial problems, deterministic and non-deterministic algorithms, P-class problems,

NP-class of problems, **Polynomial problem reduction,**

NP complete problems- **Vertex cover** and **3-SAT**

And

NP hard problem - **Hamiltonian cycle**

Intro: Time-space tradeoff in algorithm design

Example of a **space–versus–time tradeoff**

- An important question in the study of computation (time) is how best to use the registers of a CPU and/or the random-access memory of a general-purpose computer.
- In most computations, the number of registers (space) available is insufficient to hold all the data on which a program operates and registers must be reused.
- If the space is increased, the number of computation steps (time) can generally be reduced.

Unit IV	**Intractable Problems and NP-Completeness**

Time-Space trade off,

Tractable and Non-tractable Problems,

Polynomial and non-polynomial problems,

deterministic and non-deterministic algorithms,

P-class problems, NP-class of problems,

Polynomial problem reduction,

NP complete problems- **Vertex cover** and **3-SAT**

And

NP hard problem - **Hamiltonian cycle**

Polynomial time Algorithm

- A (Deterministic) algorithm whose order-of-magnitude time performance is bounded from above by a polynomial function of n, where n is the size of its inputs.

Exponential Algorithm

- An algorithm whose order-of-magnitude time performance is not bounded, from above, by a polynomial function of n.

Why divide algorithms into these two broad classes – Polynomial and Exponential?

- Assuming that one instruction can be executed every microsecond, following table shows times

	10	20	50	100	300
n^2	$\frac{1}{10000}$ second	$\frac{1}{2500}$ second	$\frac{1}{400}$ second	$\frac{1}{100}$ second	$\frac{9}{100}$ second
n^5	$\frac{1}{10}$ second	3.2 seconds	5.2 minutes	2.8 hours	28.1 days
2^n	$\frac{1}{1000}$ second	1 second	35.7 years	400 trillion centuries	a 75-digit number of centuries
n^n	2.8 hours	3.3 trillion years	a 70-digit number of centuries	a 185-digit number of centuries	a 728-digit number of centuries

(The Big Bang was approximately 15 billion years ago.)

Tractable and Intractable Problems

Tractable Problem: a problem that is solvable by a polynomial-time algorithm. The upper bound is polynomial.

-Searching an unordered list

-Searching an ordered list

-Sorting a list

-Finding a minimum spanning tree in a graph (even though there's a gap)

Intractable Problem: a problem that cannot be solved by a polynomial-time algorithm. The lower bound is exponential.

Real life intractable problems

- Many Real life problems are intractable
  - Provably intractable (proven to have no polynomial-time algorithm)
  - Or, have no known polynomial-time algorithm even if it is not yet proven intractable

How to solve - Real life intractable problems ?

- In practice, **try out improvements** (backtrack, eliminate symmetry, sacrifice Space Complexity, use heuristics (rule-of-thumb) dynamically, etc.), while accepting that exponential cases are still possible
- **Solve simpler/restricted versions** of the problem (when a solution to a slight variant of the problem would still be useful), while avoiding exponential complexity
- Use a polynomial-time **probabilistic algorithm**: one which gives the right answer only with very high probability.
  - Trading program correctness, in the interests of speed.
- For **optimisation** problems, use a polynomial-time **approximation algorithm**: one which is not guaranteed to find the best answer

Third Class of problems?

There is a third class of problem

- Dividing problems into **two classes** (Tractable/Intractable) ignores ‘gaps’ between lower and upper bounds

- There are problems for which, the state of our knowledge is such that, the gap spans this **coarse division (tractable/intractable)**

Three broad classes of problems

1. Problems with **known** polynomial-time algorithms.
2. Problems that are **provably intractable** (proven to have no polynomial-time algorithm)
3. *Problems with **no known** polynomial-time algorithm **but not yet proven** to be intractable*

Unit IV	**Intractable Problems and NP-Completeness**

Time-Space trade off,

Tractable and Non-tractable Problems,

Polynomial and non-polynomial problems,

deterministic and non-deterministic algorithms,

P-class problems, NP-class of problems,

Polynomial problem reduction,

NP complete problems- **Vertex cover** and **3-SAT**

And

NP hard problem - **Hamiltonian cycle**

Problem types

- Problems with **Polynomial amount** of inputs (say, n)
  - Can have polynomial time algorithms **or otherwise**
- Problems with **polynomial amount** of outputs
  - E.g. Decision problems (Yes/No)
  - Can have polynomial time algorithms **or otherwise**
- Problems with **non-polynomial** amounts of outputs
  - Cannot have polynomial time algorithms

... Problems with **non-polynomial** amounts of outputs

If a problem has an exponential amount of output, then no amount of cleverness is going to find an algorithm that is polynomial

- E.g. Towers of Hanoi, finding all permutations of n numbers, enumerating all Hamiltonian cycles.

Examples of non-polynomial problems

- Some problems require a **non-polynomial amount** of **output**, so they clearly will take a **non-polynomial amount of time**
  - **Towers of Hanoi**: we can prove that any algorithm that solves this problem must have a worst-case running time that is at least $2^n - 1$.
  - **List all permutations** (all possible orderings) of n numbers

- Others have polynomial amounts of output, but still cannot be solved in polynomial time:
  - For an n × n draughts board with an arrangement of pieces, determine whether there is a winning strategy for White (i.e. a sequence of moves so that, no matter what Black does, White is guaranteed to win).
    - We can prove that any algorithm that solves this problem must have a worst-case running time that is at least 2^n.

Decision Problems vs Optimization Problems

Decision problems

- The solution is simply "Yes" or "No".
- Optimization problems are more difficult.
- e.g. the traveling salesperson problem
  - Optimization version:

    Find the shortest tour
  - Decision version:

    Is there a tour whose total length is less than or equal to a constant c ?

Decision problem algorithms

- Suppose you've a decision problem and an algorithm A that solves that decision problem.
- We'll say that algorithm A accepts input x if it returns YES
- We'll say that algorithm A rejects input x if it returns NO

Importance of decision problems

- Research work in Complexity Theory often concentrate on **decision problems**

- Many **optimization problems** can be solved using **decision problems, because for** many non-decision problems, there are related decision problems

Converting Non-Decision Problem to Decision Problem

A non-decision problem (such as the TSP Search Problem) can often be turned into a decision problem by introducing a parameter k and asking if there is an answer that costs at least or at most k.

If you can't find a polynomial-time algorithm for a decision problem, then you're not likely to find one for the non-decision problems to which it is related.

Solving an Optimization problem by a Decision algorithm :

- Solving TSP optimization problem by decision algorithm :
  - Give c_1 and test (decision algorithm)
    Give c_2 and test (decision algorithm)

    Give c_n and test (decision algorithm)
  - We can easily find the smallest c_i

Similarly for- Graph coloring(using m-coloring), etc.

Introduction to More Problems

SAT problem

Graph Problems (Vertex(Node) cover, Max Clique, Hamiltonian Cycle)

SAT Problem

Satisfiability Problem

The satisfiability problem

- The satisfiability problem
  - A sample logical formula with Boolean literals, x1,x2, (in CNF form):

    $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2)$

The assignment : $x_1 \leftarrow F$, $x_2 \leftarrow T$
will make the above formula true

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

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

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

3-Conjugate Normal Form (3-CNF)-SAT Problem

- Meaning of 3-CNF: A propositional formula is in **3-CNF** if it consists of clauses connected by ANDs, and each clause is the OR of three literals. A literal is either a variable or its negation.

Example:

$(x1 \vee x2 \vee \neg x3) \wedge (x4 \vee x5 \vee x1)$

- The above 3-CNF-SAT = {<P>: P is a **satisfiable** 3-CNF propositional formula}

3-CNF SAT Problem

- Given a 3-CNF formula, SAT decision problem (Y/N), is a problem which requires to understand if there is at least one assignment which satisfies the given 3-CNF formula

Graph problems

Vertex(Node) Cover, Max-Clique, Hamiltonian Cycle

Recap: Concepts from Graph theory

Incident: Edge $\{u, v\}$ is said to be *incident* on vertices u and v. E.g. $\{A, B\}$ is incident on A and B.

Degree: The *degree* of a vertex is the number of edges incident to that vertex. E.g. A has degree 3.

Path: A *path* from vertex u to vertex z in graph G is a sequence of vertices $u, v, w, \ldots, y, z$ such that $\{u, v\}$, $\{v, w\}, \ldots,$ $\{y, z\}$ are edges in G. E.g. A, D, F, C is one path from A to C.

Cycle: A *cycle* is a path in which the first and last vertices are the same. E.g. A, D, B, A is a cycle.

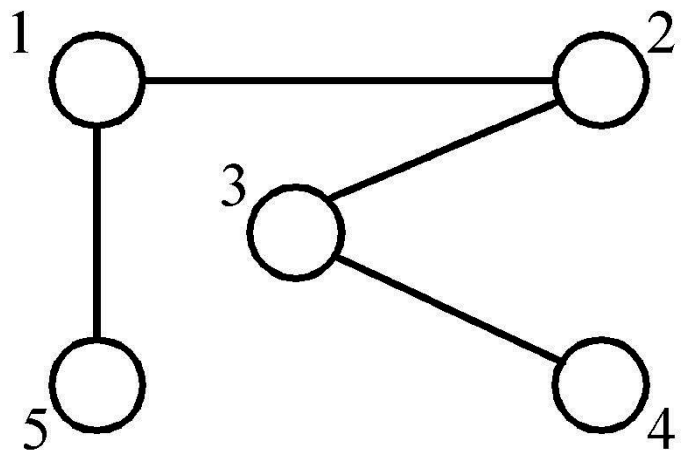
Connected vertices: In a graph G, two *vertices* u and v are *connected* iff there is a path in G from u to v. E.g. A and C are connected.

Connected graph: A *graph* G is *connected* iff for every pair of distinct vertices u and v $(u \neq v)$, there is a path from u to v. E.g. the graph shown earlier *is* connected.

The Vertex (/node) Cover problem

Def: Given a graph G=(V, E),

- S is the node cover, if $S \subseteq V$ and for every edge $(u, v) \in E$, either $u \in S$ or $v \in S$ or both.



node cover :
{1, 3}
{5, 2, 4}

- Decision problem : $\exists S \ni |S| \leq K$?

Vertex-Cover Problem

- Given a graph G, find the **smallest set of vertices** such that every edge is incident to at least one of them.

- Given a (undirected non-weighted) graph G = (V, E), a vertex cover C is a set of vertices $C \subseteq V$ such that for every edge $(u, v) \in E$, either $u \in C$ or $v \in C$ (or, possibly, both).

- Decision problem: “Given G and integer k, does G contain a vertex cover of size ≤ k?”

..cntd..Vertex Cover Problem

In the **Minimum Vertex Cover problem**, given a graph G we want to find a **smallest vertex cover**.

In the **decision version**, given G and a parameter k, we want to know whether or not G contains a vertex cover of size at most k.

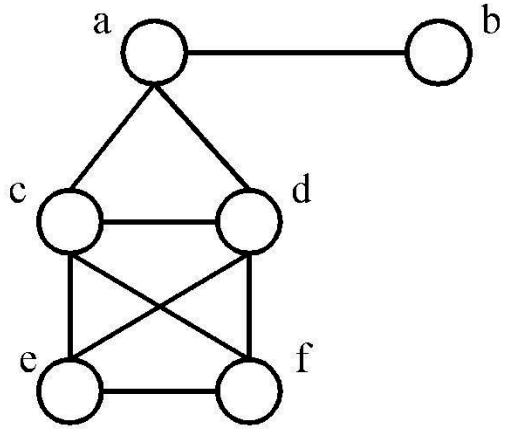
It should be clear that the **problem is in NP.**

Max clique problem

- **Def**: A **maximal complete subgraph** of a graph G=(V,E) is a clique.

The max (maximum) clique problem is to determine the size of a largest clique in G.

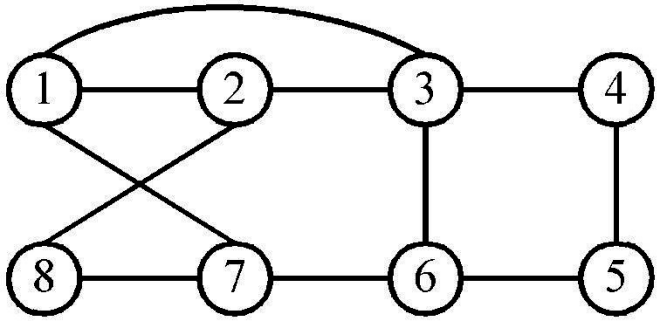
- e. g.



maximal cliques :
{a, b}, {a, c, d}
{c, d, e, f}
maximum clique :
(largest)
{c, d, e, f}

Hamiltonian cycle

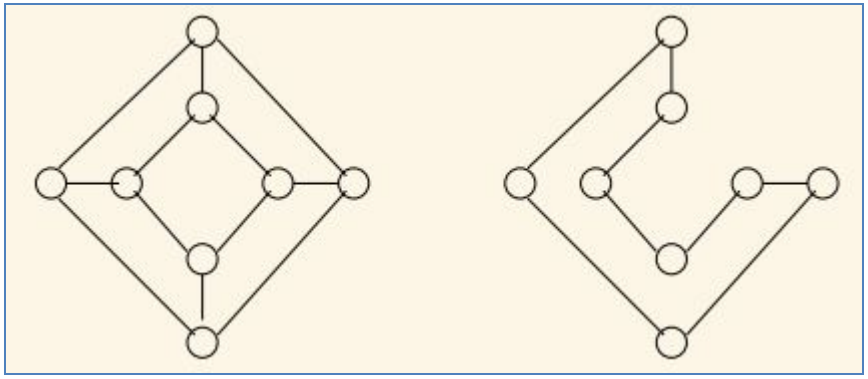
- **Def**: A Hamiltonian cycle is a round trip path along n edges of G which visits every vertex once and returns to its starting vertex.
- e.g.



Hamiltonian cycle : 1, 2, 8, 7, 6, 5, 4, 3, 1.

Hamiltonian Cycle problem

- A Hamiltonian cycle is a **cycle** that contains **every vertex exactly once** (except that the initial and final vertices will be equal)



Problem *Hamiltonian Cycle Decision Problem*
Parameters: *A graph $G = \langle V, E \rangle$.*
Returns: *YES if there is a Hamiltonian cycle; NO otherwise.*

Problem *Hamiltonian Cycle Search Problem*
Parameters: *A graph $G = \langle V, E \rangle$.*
Returns: *A Hamiltonian cycle in G, if there is one.*

Hamiltonian Cycle Decision Problem and Search Problem

* The only known algorithms are exponential.
* If n is the number of vertices,
  - E.g. generate-and-test all permutations of the vertices: $O(n^n)$
  - E.g. use a backtracking algorithm: $O(n!)$
  - There may be better algorithms, but they're all exponential.
  - But the problem has not been proven to be intractable.

Problem complexity : Hamiltonian Cycle Decision Problem vs Search Problem

- The only known algorithms are exponential
  - If n is the number of vertices, and following solutions
    - Generate-and-test all permutations of the vertices: $O(n^n)$
    - Backtracking algorithm: O(n!)
    - There may be better algorithms, but they're all exponential
    - But the problem has **not been proven to be intractable**

Sample - Provably intractable problem

Hamiltonian Cycle Enumeration Problem

Parameters: A graph G = <V, E>.

Returns: All Hamiltonian cycles in G.

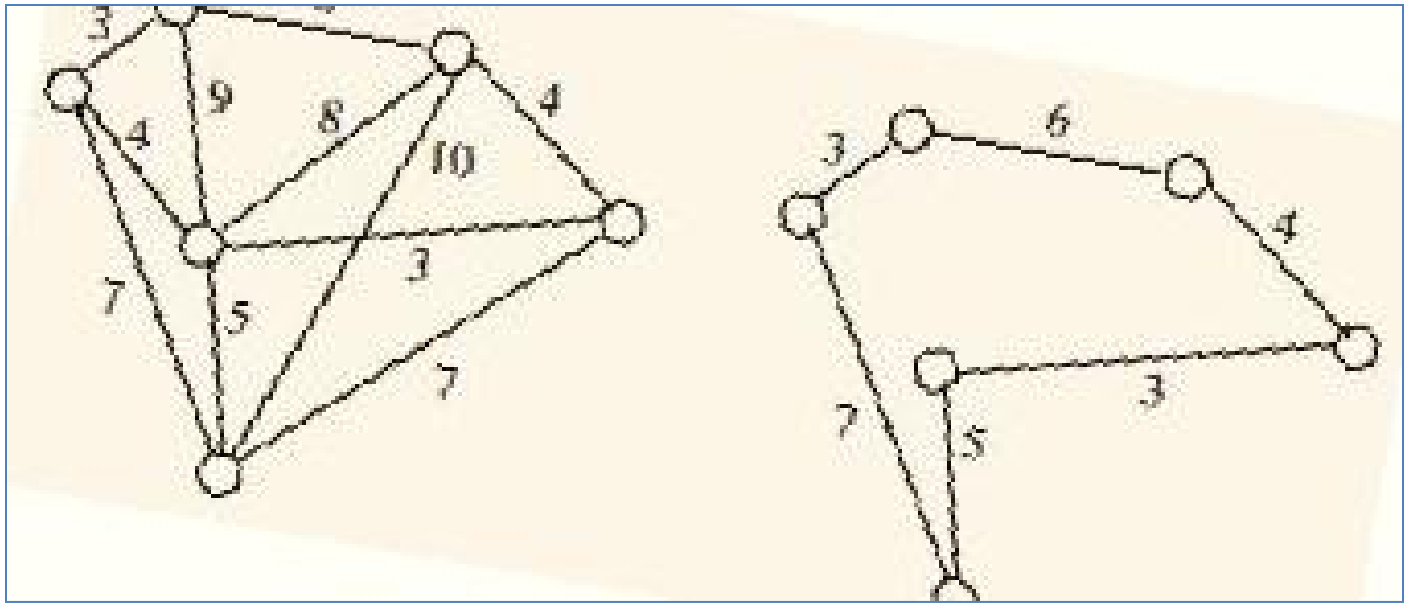
In the worst case, there can be an exponential number of Hamiltonian cycles.

Therefore, this is a problem that is provably intractable.

TSP

- TSP - applies to weighted graphs (ones with costs on the edges).
- TSP - finds **minimum cost Hamiltonian cycles**

GRAPH	**MIN COST** Hamiltonian cycle



TSP Decision Problem vs (Optimization) Search Problem

TSP Decision Problem

- Parameters:
  - A weighted graph G = <V, E> and an integer k
- Returns: YES if there is a Hamiltonian cycle whose total cost is less than or equal to k; NO otherwise

TSP Search Problem

- Parameters: A weighted graph G = <V, E>.
- Returns: A minimum cost Hamiltonian cycle

The **problem complexity** here is the same as that for the Hamiltonian cycle problems:

– TSP Decision Problem: no known polynomial-time algorithm but not yet proven to be intractable.

– TSP Search Problem: no known polynomial-time algorithm but not yet proven to be intractable.

Unit IV	**Intractable Problems and NP-Completeness**

Time-Space trade off,

Tractable and Non-tractable Problems,

Polynomial and non-polynomial problems,

deterministic and non-deterministic algorithms,

P-class problems, NP-class of problems,

Polynomial problem reduction,

NP complete problems- **Vertex cover** and **3-SAT**

And

NP hard problem - **Hamiltonian cycle**

Deterministic VS Non-deterministic Algorithms

- **Deterministic Algorithms** : **Result** of every operation in the algorithm is **uniquely defined**

- *THEORETICALLY removing the above restriction – we can allow operation whose result is **NOT uniquely defined, but restricted to a set of possibilities** – (Non-deterministic operation)*

- **Non-Deterministic Algorithms:** Algorithms, which contain operations whose **result** is **NOT uniquely defined,** but **restricted** to a **set of possibilities**

Non-deterministic Algorithms

Non-deterministic Operations and Functions

Sample Non-deterministic Algorithms/Functions

Statements in Non-deterministic (ND) Algorithm and ND Search

(i) **choice** (S) . . . arbitrarily chooses one of the elements of set S
(ii) **failure** . . . signals an unsuccessful completion
(iii) **success** . . . signals a successful completion.

Non-Deterministic Search Algorithm

$j \leftarrow$ **choice**$(1{:}n)$
if $A(j) = x$ **then print**(j); **success endif**
print(‘0’); **failure**

Nondeterministic algorithms

- A nondeterminstic algorithm consists of

  phase 1: guessing

  phase 2: checking

(Repeat) Nondeterministic operations and functions

- **Choice(S)** : arbitrarily chooses one of the elements in set S
- **Failure** : an unsuccessful completion
- **Success** : a successful completion

- Nonderministic searching algorithm:

```
j ← choice(1 : n)    /* guessing */
if A(j) = x then success   /* checking */
            else failure
```

Nondeterministic operations and functions

- A nondeterministic algorithm **terminates unsuccessfully** iff there exist no set of choices leading to a success signal.

- The time required for *choice(1 : n)* is O(1).

- A deterministic interpretation of a non-deterministic algorithm can be made by allowing unbounded parallelism in computation.

Non-deterministic Sorting

```
    procedure NSORT(A, n)
       //sort n positive integers//
       integer A(n), B(n), n, i, j
1      B ← 0   //initialize B to zero//
2      for i ← 1 to n do
3         j ← choice(1:n)
4         if B(j) ≠ 0 then failure endif
5         B(j) ← A(i)
6      repeat
7      for i ← 1 to n - 1 do    //verify order//
8         if B(i) > B(i + 1) then failure endif
9      repeat
10     print(B)
11     success
12  end NSORT
```

Nondeterministic sorting (Guessing & Checking)

```
B ← 0
/* guessing */
for i = 1 to n do
    j ← choice(1 : n)
    if B[j] ≠ 0 then failure
    B[j] = A[i]
/* checking */
for i = 1 to n-1 do
   if B[i] > B[i+1] then failure
   success
```

Need for Non-deterministic Machine and Algorithm?

- Concept of Non-Determinism may seem pointless now, but it is a theoretical technicality in part of a broader argument that we will develop later
- We will use it to **group a class of Problems**, so that, if we ever find something out about the complexity of one of them, we may learn something about the complexity of all of them.

Recap Questions

Q) What are the 2 class of problems? Explain.

Q) What are PT and NPT Algorithms?

Q) What are deterministic and non-deterministic algorithms?

Q) What can you say about the deterministic time complexity vs non-deterministic t c of a problem's algorithm?

Unit IV	Intractable Problems and NP-Completeness

Time-Space trade off,

Tractable and Non-tractable Problems,

Polynomial and non-polynomial problems,

deterministic and non-deterministic algorithms,

P-class problems, NP-class of problems,

Polynomial problem reduction,

NP complete problems- **Vertex cover** and **3-SAT**

And

NP hard problem - **Hamiltonian cycle**

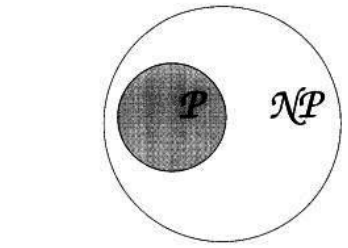
The Complexity Class P

- **Definition**: The complexity class P is the set of all decision problems that can be solved with worst-case polynomial time-complexity.
  - In other words, a problem is P-Class problem if it is a decision problem and there exists an algorithm that solves any instance of size n in $O(n^k)$ time, for some integer k.
- So P is just the **set of tractable decision problems**: the decision problems for which we have polynomial-time algorithms.

The Complexity Class NP

- Second class of decision problems that is called NP (non-deterministically polynomial)

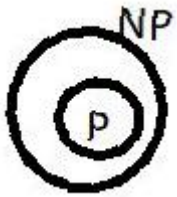
- Definition of NP involves the idea of a **non-deterministic algorithm**
- Commonly believed relationship between P and NP



Commonly believed relationship between $\mathcal{P}$ and $\mathcal{NP}$

NP-Class of Problems

- Technically: If a problem can be solved in polynomial time, by a Non-deterministic turing machine, it is in NP-Class
- Informally: If a problem's solution can be checked for correctness in polynomial time, then it is in class NP
  - *Are there problems, where checking their correctness is hard??*
- P $\subseteq$ NP i.e. all P problems are NP (*WHY??)*



but are P = NP????(Million Dollar question)

P and NP problems

Million dollar question: is P = NP?

P-Class and NP-Class problems

- **P – Class Problems** : - have DETERMINISTIC polynomial time algorithms. Problem can be solved in polynomial time

- **NP – Class Problems** : - have NON-DETERMINISTIC polynomial time algorithms

Need for NP Class?

- Concept of **Non-Determinism** and **NP-Class** may seem **pointless now**, but it is a theoretical technicality in part of a broader argument that we will develop later
- It is a part of a way of showing that some **problems are related** and so, if we ever find something out about the complexity of one of them, we may learn something about the complexity of all of them

Non-Deterministic Decision Knapsack

```
1   Algorithm DKP(p, w, n, m, r, x)
2   {
3       W := 0; P := 0;
4       for i := 1 to n do
5       {
6           x[i] := Choice(0, 1);
7           W := W + x[i] * w[i]; P := P + x[i] * p[i];
8       }
9       if ((W > m) or (P < r)) then Failure();
10      else Success();
11  }
```

Non-Deterministic Decision Clique

```
1   Algorithm DCK(G, n, k)
2   {
3       S := ∅; // S is an initially empty set.
4       for i := 1 to k do
5       {
6           t := Choice(1, n);
7           if t ∈ S then Failure();
8           S := S ∪ {t} // Add t to set S.
9       }
10      // At this point S contains k distinct vertex indices.
11      for all pairs (i, j) such that i ∈ S, j ∈ S, and i ≠ j do
12          if (i, j) is not an edge of G then Failure();
13      Success();
14  }
```

Non-Deterministic SAT

```
1   Algorithm Eval(E, n)
2   // Determine whether the propositional formula E is
3   // satisfiable. The variables are x_1, x_2, ..., x_n.
4   {
5       for i := 1 to n do  // Choose a truth value assignment.
6           x_i := Choice(false, true);
7       if E(x_1, ..., x_n) then Success();
8       else Failure();
9   }
```

Nondeterministic satisfiability

Q)

- If we write a non-deterministically polynomial algorithm for decision TSP, can we say TSP is NP? Why?

- Write a non-deterministic TSP decision algorithm.

Unit IV	**Intractable Problems and NP-Completeness**

Time-Space trade off,

Tractable and Non-tractable Problems,

Polynomial and non-polynomial problems,

deterministic and non-deterministic algorithms,

P-class problems, NP-class of problems,

Polynomial problem reduction,

NP complete problems- **Vertex cover** and **3-SAT**

And

NP hard problem - **Hamiltonian cycle**

Concept of Polynomial Reductions

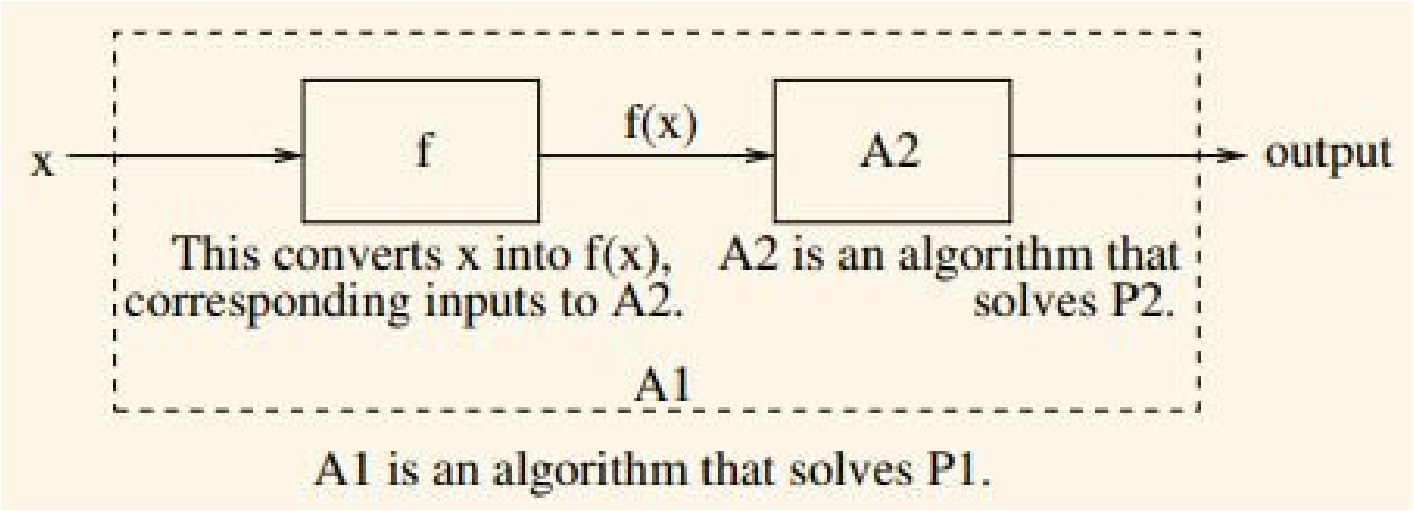
Reducing a problem into another problem in Polynomial Time

Reduction

- Suppose you have a problem P2 which you know how to solve, (say, by using algorithm A2)
- If another problem P1, similar to P2, needs to be solved, we may use a technique called REDUCTION
- A reduction of P1 to P2 involves:
  - Transforming inputs to P1 into inputs to P2
  - Running A2 (which solves P2) as a ‘black-box’
  - and, interpreting the outputs from A2 as answers to P1

...contd..Reduction

- A problem P1 is reducible to a problem P2 if there is a function **f** that takes any input **x** to P1 and transforms it to an input f(x) of P2, such that the solution to P2 on f(x) is the solution to P1 on x.



If “-> “ symbolizes “reduces to”,

Sample problem of reduction : Squaring a matrix “ -> ” matrix multiplication

“Reduces to” is a transitive relation

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

Polynomial-Time Reductions

- Reductions which take polynomial time (**≤p**)
- We use **Polynomial-Time Reductions** in Complexity Theory

  Suppose, P1->P2 (or P1 **≤p** P2), then it means that:-
  - P2 is 'as hard as' P1
  - If P2 is tractable, then P1 is tractable
    - Or, the lower bound we prove for P1, may apply to P2 also
    - Equivalently (the contrapositive): if P1 is proved to be intractable, then P2 is also intractable

Q)

- If Squaring a Matrix reduces to Matrix Multiplication in polynomial time, then, if Matrix Multiplication is tractable, what can we conclude about Squaring a Matrix and why?

Polynomially Equivalent Problems

- 2 Problems L1, L2 are polynomially equivalent iff L1 -> L2 and L2 -> L1

NP-Hard problems

- A problem P is NP-Hard iff SAT (Satisfiability) reduces to P
- SAT->P

Strategy to show a problem L2 is NP -Hard

1. Pick a problem L_1 already known to be $\mathcal{NP}$-hard.

2. Show how to obtain (in polynomial deterministic time) an instance I' of L_2 from any instance I of L_1 such that from the solution of I' we can determine (in polynomial deterministic time) the solution to instance I of L_1 (see Figure 11.3).

3. Conclude from step (2) that $L_1 \propto L_2$.

4. Conclude from steps (1) and (3) and the transitivity of $\propto$ that L_2 is $\mathcal{NP}$-hard.

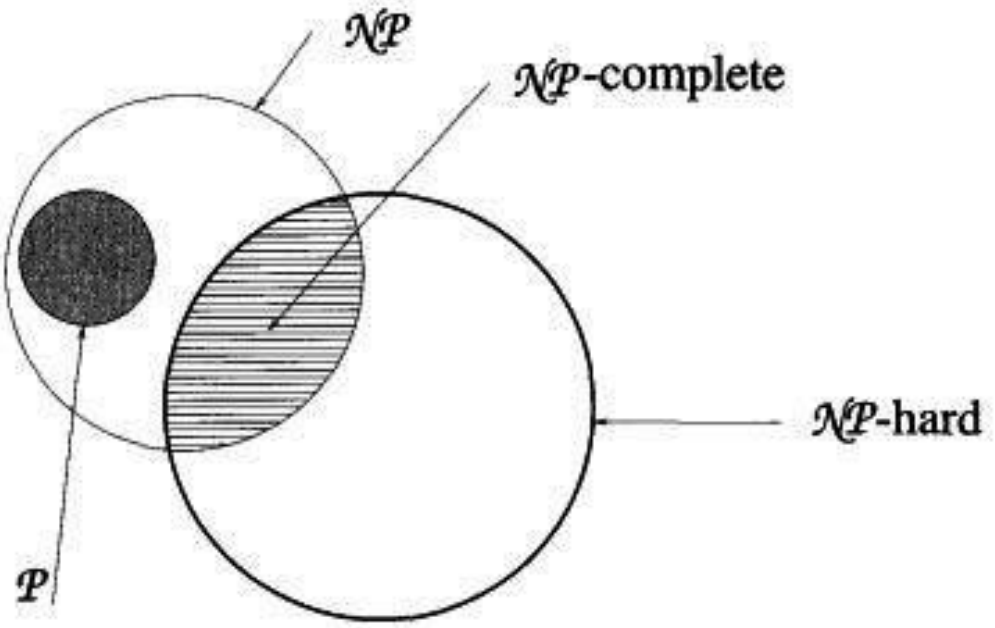
Importance of SAT Problem

- **(S. Cook's) Research Question** – Is there any problem in NP such that if we showed it to be in P then it would imply P=NP?
- **(Answer) S. Cook's theorem**
  - **Satisfiability (SAT) is in P, iff P = NP**

- SAT has been proved to be **NP-Hard**

NP-Complete problems

- A problem P is NP-Complete iff it is NP-Hard(i.e SAT (Satisfiability) reduces to P) and P belongs to NP



Commonly believed relationship among $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-complete, and $\mathcal{NP}$-hard problems

NP-Complete Problems

(1)The problem itself is in NP class.

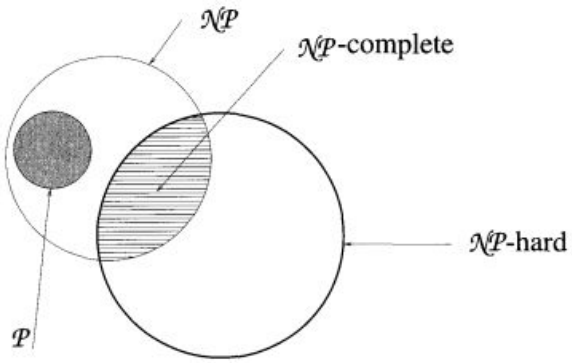
(2)All other problems in NP class can be polynomial time reducible to that.

(B is polynomial time reducible to C is denoted as B ***->p*** *C)*

NP-Hard and NP-Complete Problems

It is easy to see that there are $\mathcal{NP}$-hard problems that are not $\mathcal{NP}$-complete. Only a decision problem can be $\mathcal{NP}$-complete. However, an optimization problem may be $\mathcal{NP}$-hard. Furthermore if L_1 is a decision problem and L_2 an optimization problem, it is quite possible that $L_1 \propto L_2$. One can trivially show that the knapsack decision problem reduces to the knapsack optimization problem. For the clique problem one can easily show that the clique decision problem reduces to the clique optimization problem. In fact, one can also show that these optimization problems reduce to their corresponding decision problems

Yet, optimization problems cannot be $\mathcal{NP}$-complete whereas decision problems can. There also exist $\mathcal{NP}$-hard decision problems that are not $\mathcal{NP}$-complete. Figure shows the relationship among these classes.



Commonly believed relationship among $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-complete, and $\mathcal{NP}$-hard problems

Q) Which of the following statements are TRUE?

1. The problem of determining whether there exists a cycle in an undirected graph is in P.
2. The problem of determining whether there exists a cycle in an undirected graph is in NP.
3. If a problem A is NP−Complete, there exists a non-deterministic polynomial time algorithm to solve A.

Q)

- For a problem P, if P1 is its decision problem and P2 is its optimization problem then can P1 reduce to P2?
- Can Decision Clique reduce to Max –Clique?

- Can knapsack decision problem reduce to knapsack optimization problem?

- Can optimization problems be NP-Complete?
- Are all decision problems NP-Complete?

Unit IV	Intractable Problems and NP-Completeness

Time-Space trade off,

Tractable and Non-tractable Problems,

Polynomial and non-polynomial problems,

deterministic and non-deterministic algorithms,

P-class problems, NP-class of problems,

Polynomial problem reduction,

NP complete problems- 3-SAT, Clique Decision Problem and Vertex cover

NP hard problem - **Hamiltonian cycle**

Unit IV	**Intractable Problems and NP-Completeness**

Time-Space trade off,
Tractable and Non-tractable Problems,
Polynomial and non-polynomial problems,
deterministic and non-deterministic algorithms,
P-class problems, NP-class of problems,
Polynomial problem reduction,
NP complete problems- 3-SAT, **Clique Decision Problem** and Vertex cover
And
NP hard problem - Hamiltonian cycle

Show that **Clique Decision Problem (CDP) is NP-Complete**

1) Show CDP is NP

AND

1) Show that CDP is NP – Hard

To Show that
Clique Decision Problem (CDP) is NP-Complete

1) Show CDP is NP

- i) It has Non-deterministically polynomial time algorithm

  OR

- ii)It's solution can be checked for correctness in polynomial time

AND

2) Show that CDP is NP – Hard

- Reduce a known NP-Hard problem to CDP in polynomial time

1) Show CDP is NP:

using (i) It has Non-deterministically polynomial time algorithm *(as seen previously)*

```
1   Algorithm DCK(G, n, k)
2   {
3       S := ∅; // S is an initially empty set.
4       for i := 1 to k do
5       {
6           t := Choice(1, n);
7           if t ∈ S then Failure();
8           S := S ∪ {t} // Add t to set S.
9       }
10      // At this point S contains k distinct vertex indices.
11      for all pairs (i, j) such that i ∈ S, j ∈ S, and i ≠ j do
12          if (i, j) is not an edge of G then Failure();
13      Success();
14  }
```

2) Reduce a known NP-Hard problem to CDP

3 SAT - > Clique Decision Problem (CDP)

3-SAT and Clique problem

Given a boolean formula in 3 CNF, **3-SAT** problem is to find whether the formula is satisfiable

For a given graph $G = (V, E)$ and integer k, the Clique problem is to find whether G contains a clique of size $>= k$.

Reduction of 3-SAT to Clique

A formula Φ of k three-literal clauses can be reduced to a k-Clique problem in the following way:

Construct a graph G of k clusters with a maximum of 3 nodes in each cluster.

Each cluster corresponds to a clause in Φ.

Each node in a cluster is labeled with a literal from the clause.

An edge is put between all pairs of nodes in different clusters except for pairs of the form $(x, \bar{x})$

No edge is put between any pair of nodes in the same cluster.

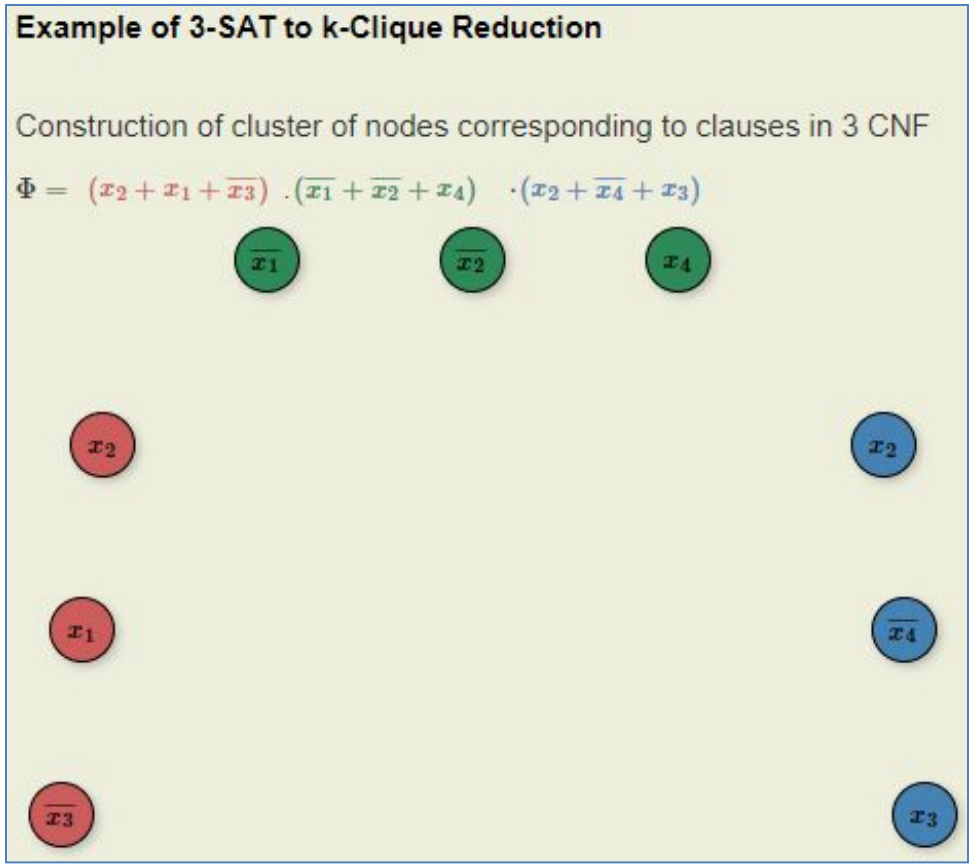
If length of the Formula is m, then O(m) time will be required to obtain G from Formula

Example of 3-SAT to k-Clique Reduction

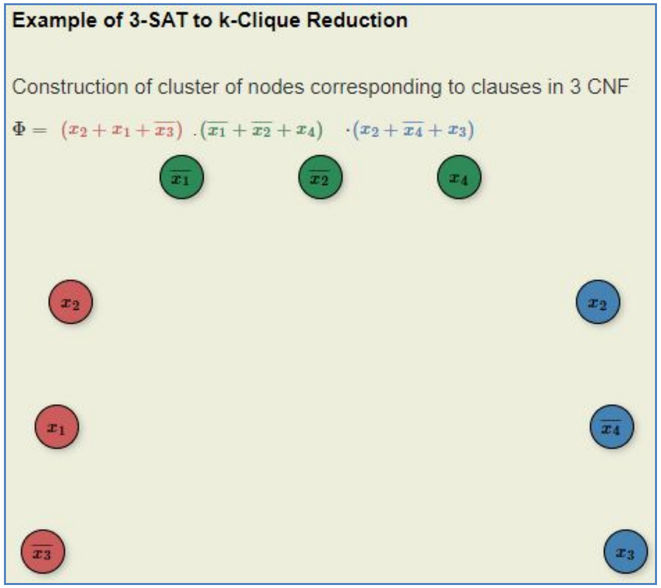
$\Phi = (x_2 + x_1 + \overline{x_3}) \cdot (\overline{x_1} + \overline{x_2} + x_4) \cdot (x_2 + \overline{x_4} + x_3)$

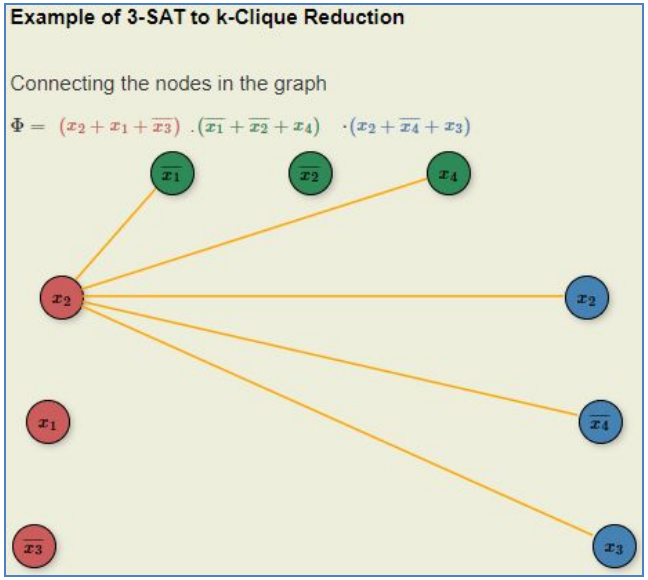
Φ is $True$ for which assignment

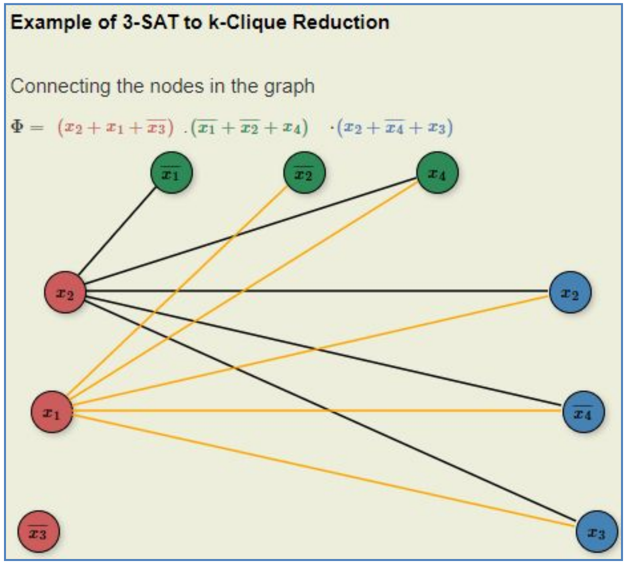
Example 1: Converting a 3 –SAT decision problem to Clique Decision Problem (CDP) in polynomial time

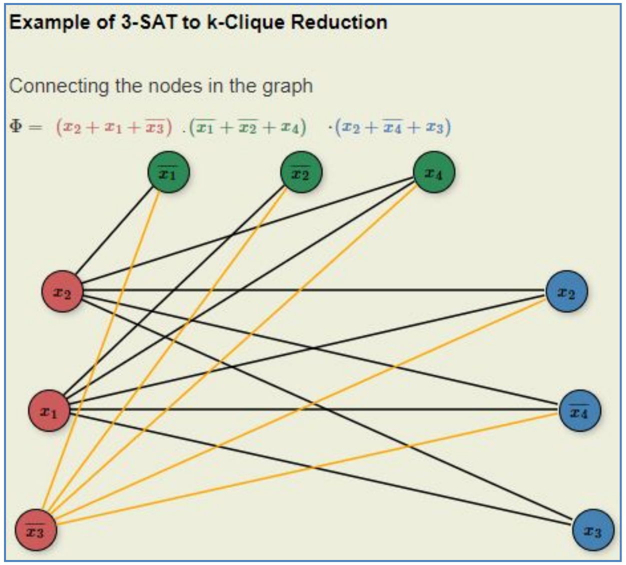


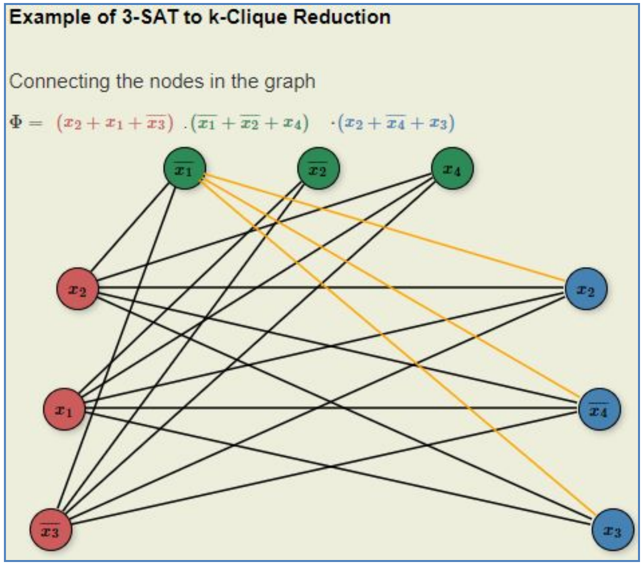
Example 1: ...contd.. Converting a 3 –SAT problem to Clique Decision Problem (CDP) in polynomial time



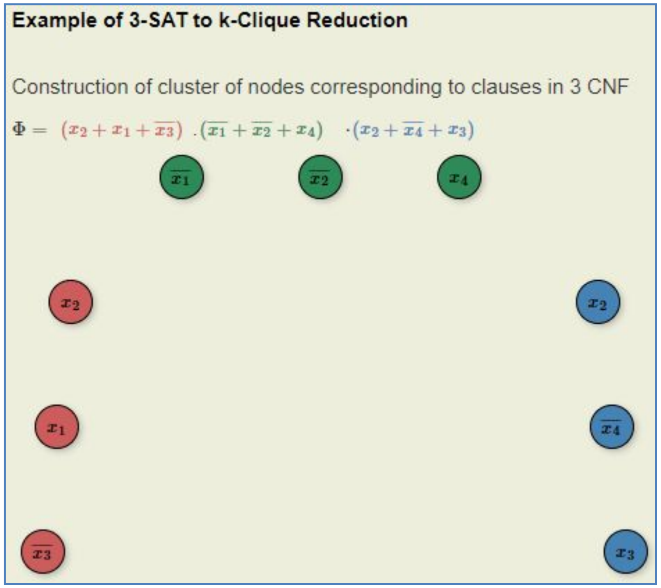


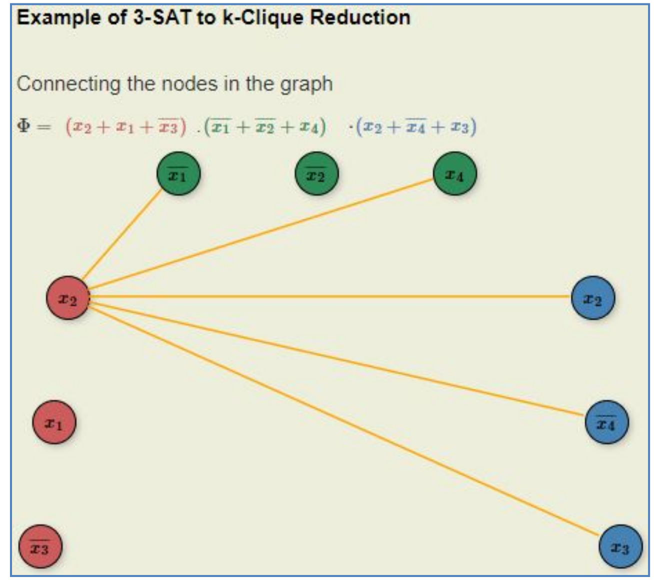


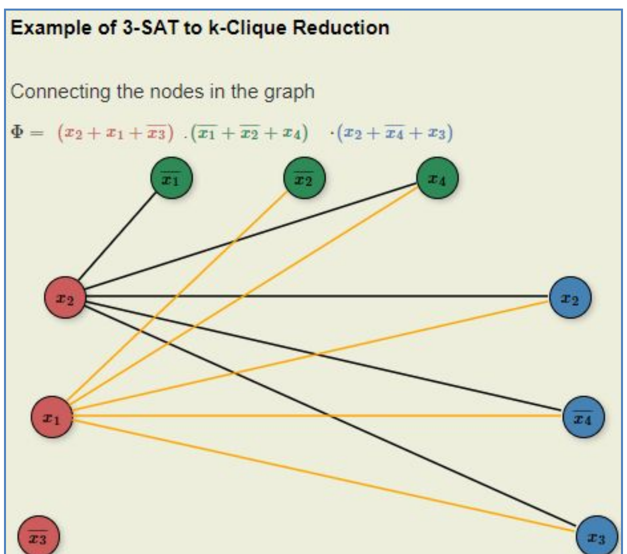


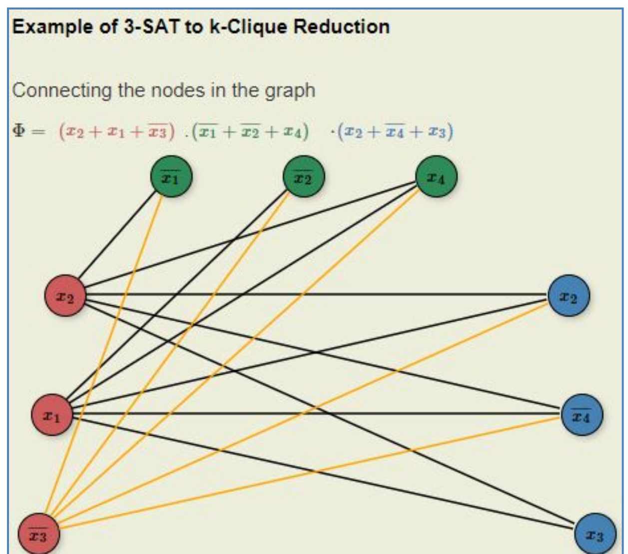


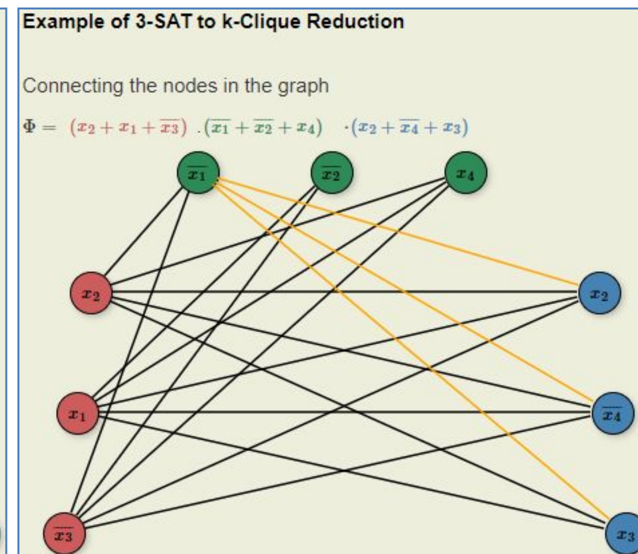
Example 1: : ...contd.. Converting a 3 –SAT problem to Clique Decision Problem (CDP) in polynomial time



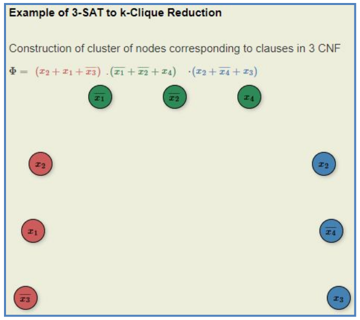


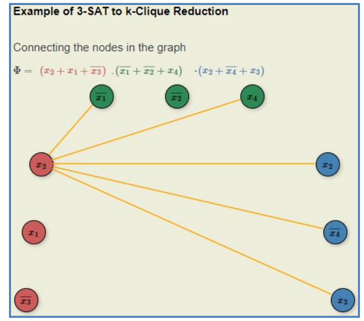


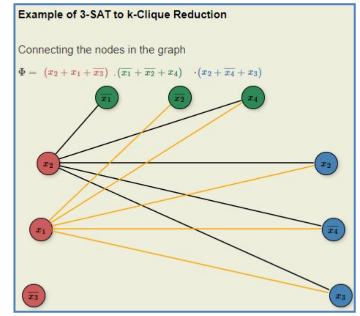


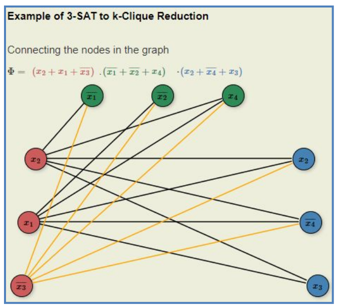


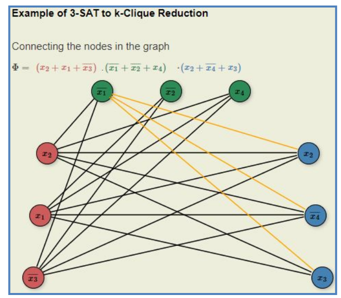
Example : 1 ...contd...

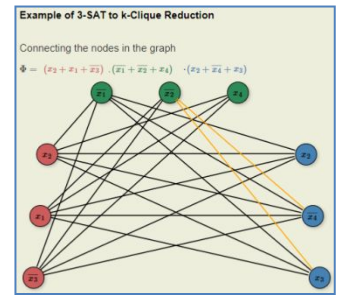


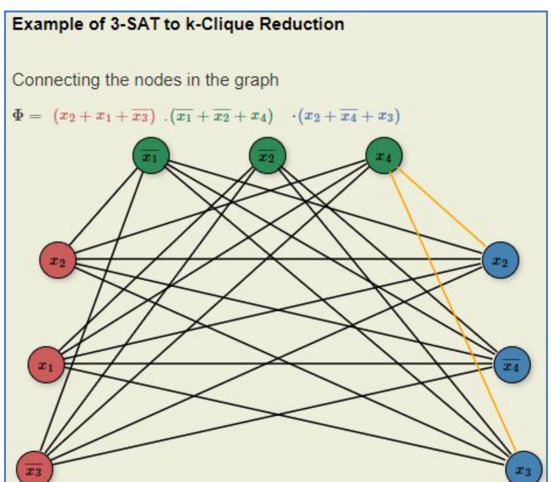


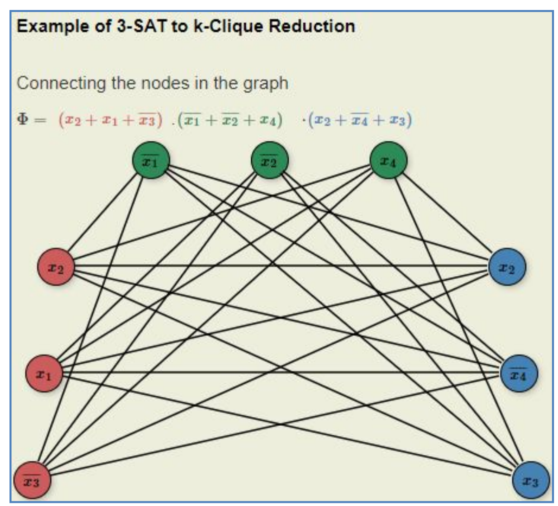












Insights about the graph

1. If two nodes in the graph are connected, the corresponding literals can be simultaneously be assigned $True$.
(This is true since there is no edge between nodes corresponding to literals of type x and $\overline{x}$).

2. If two literals, not from the same clause can be assigned $True$ simultaneously, the nodes corresponding to these literals in the graph are connected.

3. Construction of the graph can be performed in polynomial time

3-SAT to k-Clique Reduction

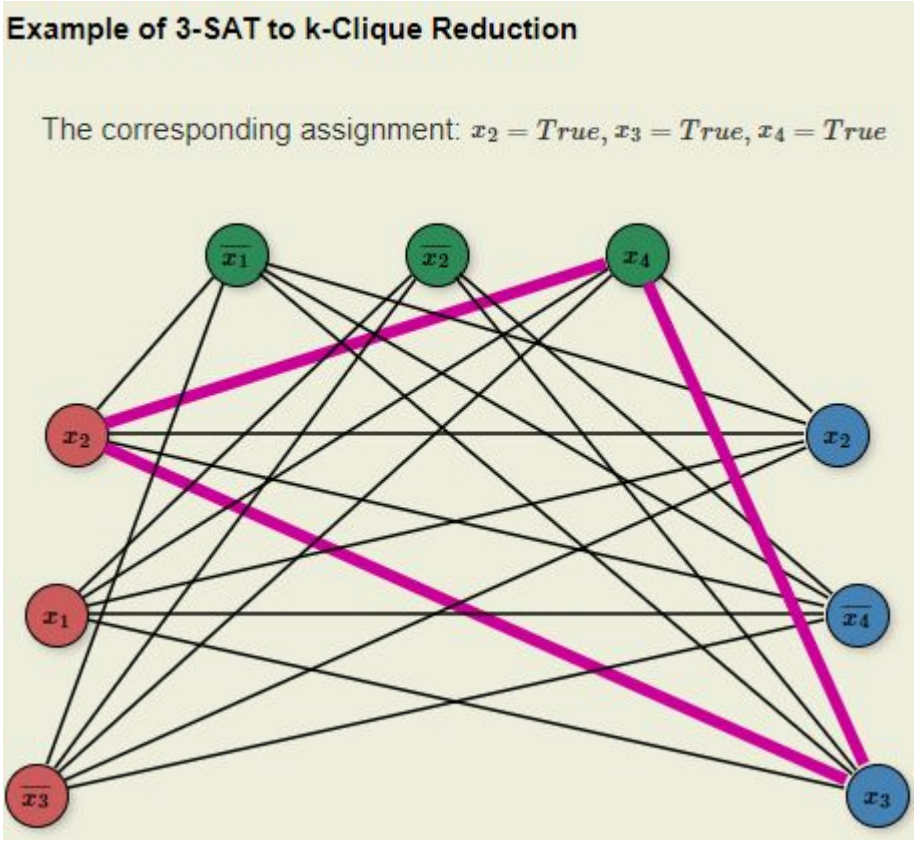
G has a k-clique if and only if Φ is satisfiable.

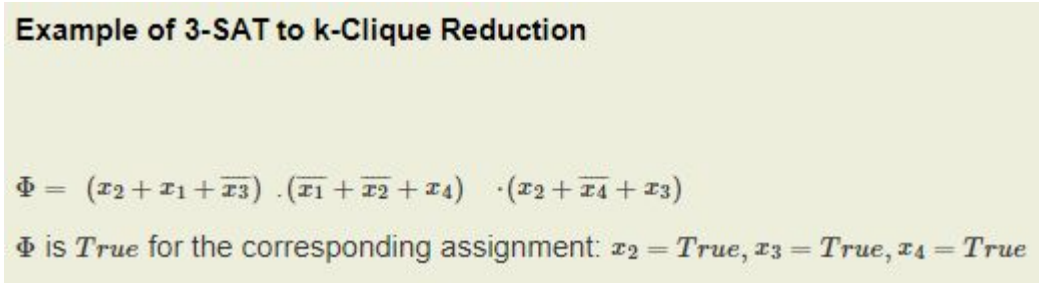
1. **If the graph G has a k-clique,** the clique has exacty one node from each cluster.
(This is because no two nodes from the same cluster are connected to each other, hence they can never be a part of the same clique.)
All nodes in a clique are connected, hence all corresponding literals can be assigned $True$ simultaneously.
Each literal belong to exactly one of the k-clauses. Hence Φ **is satisfiable**

2. If Φ is satisfiable, let A be a satisfying assignment. Select from each clause a literal that is $True$ in A to construct a set S. $||S|| = k$.Since no two literals in A are from the same clause and all of them are simultaneously $True$, all the corresponding nodes in the graph are connected to each other, forming a k-clique.Hence **the graph has a k-clique**

3-SAT ≤p CDP

Solve 3-Clique Problem, if answer is yes, then 3-SAT problem is solved

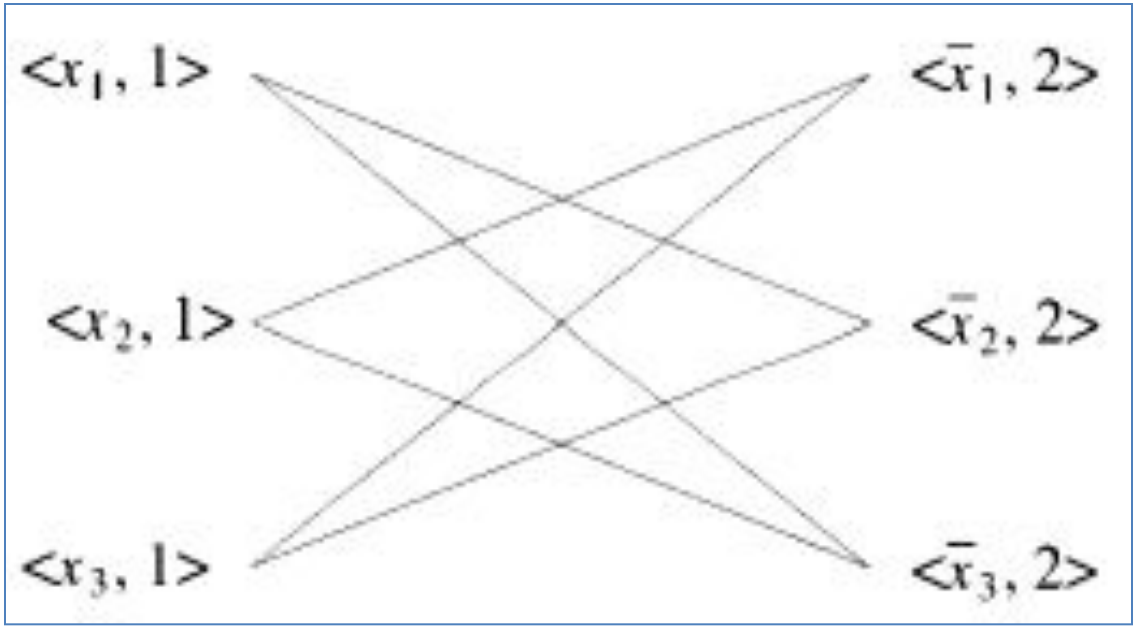




Example 2

Consider $F = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$.

A sample graph and satisfiability



Non-Deterministic Decision Clique

```
1   Algorithm DCK(G, n, k)
2   {
3       S := ∅; // S is an initially empty set.
4       for i := 1 to k do
5       {
6           t := Choice(1, n);
7           if t ∈ S then Failure();
8           S := S ∪ {t} // Add t to set S.
9       }
10      // At this point S contains k distinct vertex indices.
11      for all pairs (i, j) such that i ∈ S, j ∈ S, and i ≠ j do
12          if (i, j) is not an edge of G then Failure();
13      Success();
14  }
```

Unit IV	**Intractable Problems and NP-Completeness**

Time-Space trade off,

Tractable and Non-tractable Problems,

Polynomial and non-polynomial problems,

deterministic and non-deterministic algorithms,

P-class problems, NP-class of problems,

Polynomial problem reduction,

NP complete problems- 3-SAT, Clique Decision Problem **and Vertex cover**

And

NP hard problem - **Hamiltonian cycle**

Show that Vertex (Node) Cover Decision Problem (NCDP) is NP-Complete

To Show NCDP is NP-Complete

1) Show NCDP is NP

- i) It has Non-deterministically polynomial time algorithm

  OR

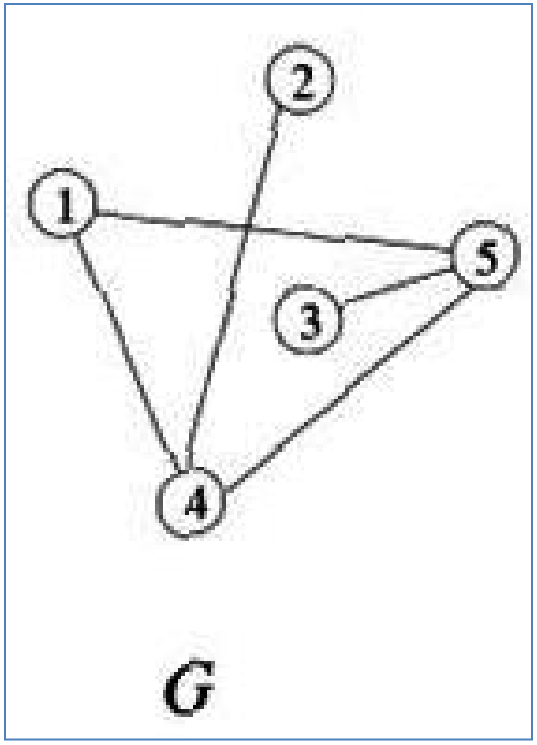
- ii)It's solution can be checked for correctness in polynomial time

AND

2) Show that it is NP – Hard

- Reduce a known NP-Hard problem to NCDP in polynomial time

1) Show NCDP is NP
using *(ii) Solution checking in polynomial time*



(ii) Polynomial Time Checking of Solution of NCDP

Let Subset V' of V={1,2,3,4,5}, contain the vertices in the vertex cover of various sizes
size k= 2, V'={4,5} - *correct;* {1,5} - *incorrect*
size k= 3, V'={1,3,5} – *incorrect* ; {1,2,5} – *correct*
size k= 4, V'={1,2,3,5} – *correct* ; {1,2,3,4} – *correct*

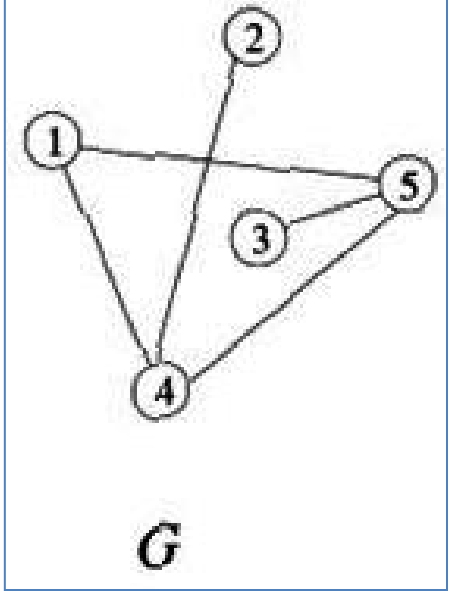
We need to check whether the set V' is a correct vertex cover of size k

…contd……

(ii) Polynomial Time Checking of Solution of NCDP

Let Subset V' of V={1,2,3,4,5}, contain the vertices in the vertex cover of various sizes
size k= 2, V'={4,5} - *correct;* {1,5} - *incorrect*
size k= 3, V'={1,3,5} – *incorrect* ; {1,2,5} – *correct*
size k= 4, V'={1,2,3,5} – *correct* ; {1,2,3,4} – *correct*

We need to check whether the set V' is a correct vertex cover of size k
STRATEGY for a graph G(V, E) :



STRATEGY for checking the correctness of NCDP solution V' of size K, of a graph G(V, E) :

```
count=0
for each vertex v in V'
    remove all edges adjacent to v from set E ;  count ++
    if ( E is empty)
      given solution is correct
    else
      given solution is incorrect
```

2) Show that NCDP is NP – Hard by Reducing a known NP-Hard problem (CDP) to NCDP in polynomial time

2) To Show that NCDP (Vertex Cover D Prob) is NP – Hard

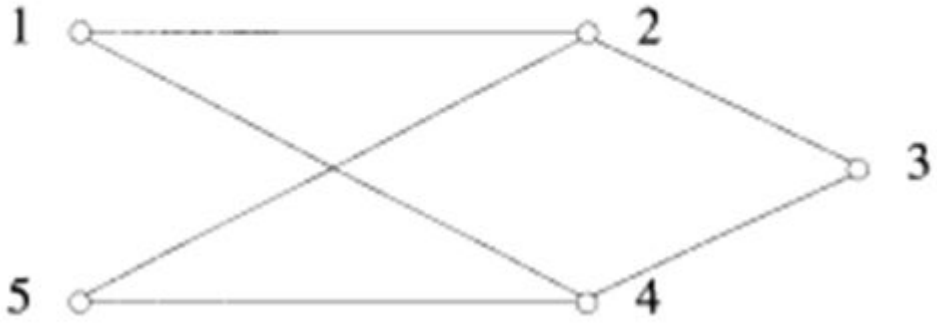
- To prove that Vertex Cover is NP Hard, we take some problem which has already been proven to be NP Hard (CDP) , and show that this problem can be reduced to the Vertex Cover problem.
- Using instance (G, k) of the Clique problem we reduced it to an instance of the vertex cover problem

The clique decision problem $\propto$ the node cover decision problem.

Example Consider the graph

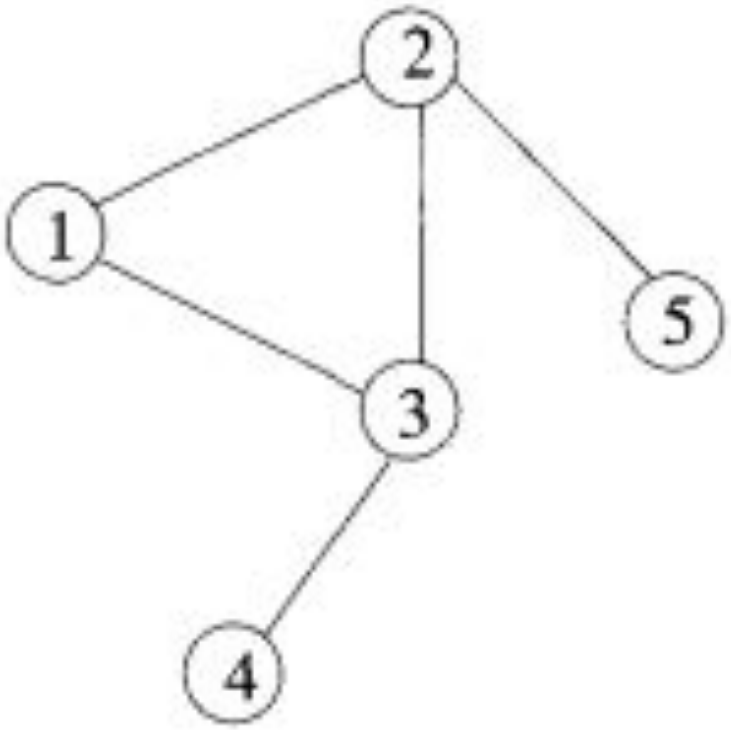
$S = \{2, 4\}$ is a node cover of size 2.

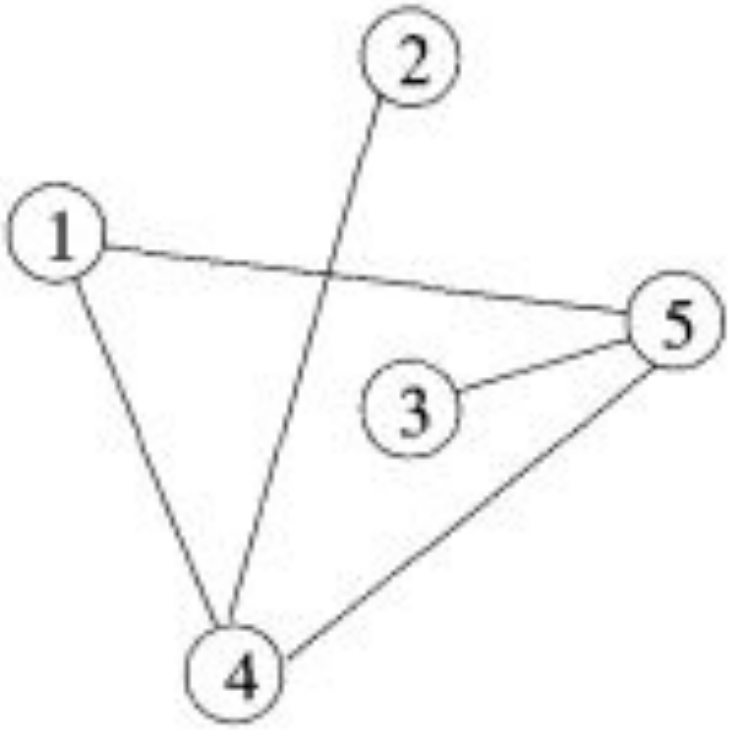
$S = \{1, 3, 5\}$ is a node cover of size 3.



—**Figure** A sample graph and node cover —

In the node cover decision problem we are given a graph G and an integer k. We are required to determine whether G has a node cover of size at most k.

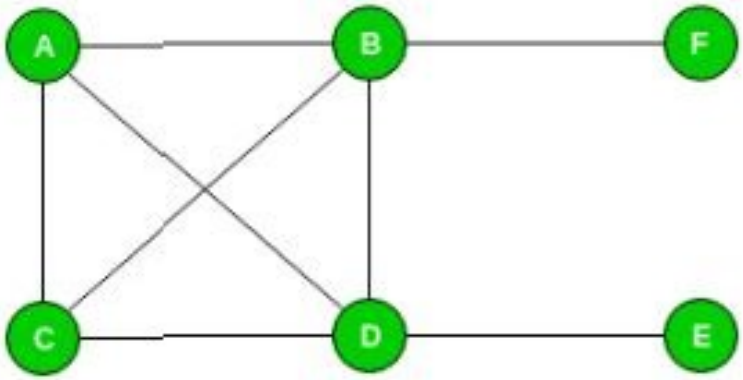




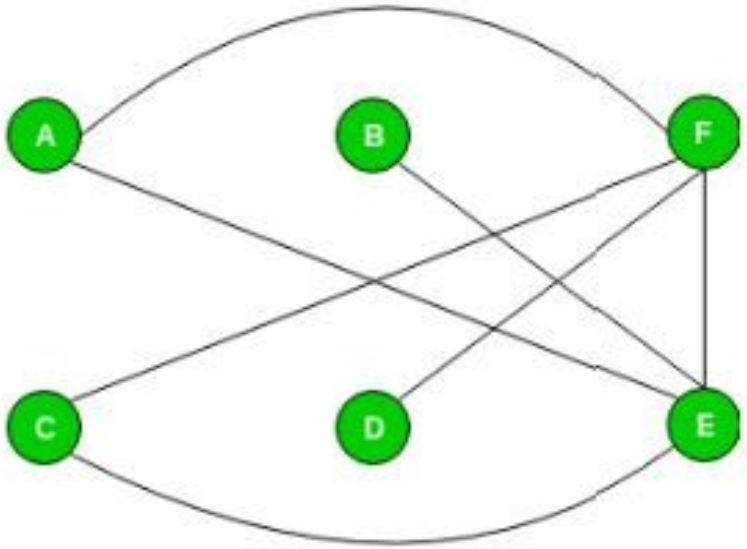
G *G'*

A graph and its complement

Graph G' consists of all edges not in G, but in the complete graph using all vertices in G



G
V' ={A, B, C, D}



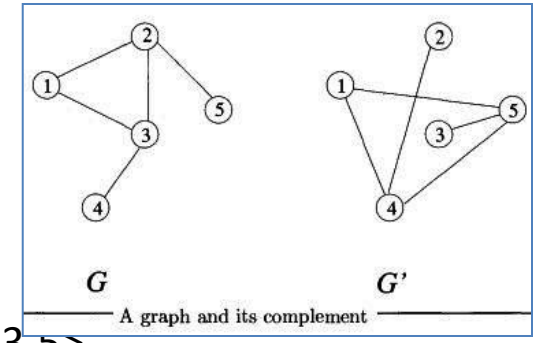
G'
V'' ={E, F}

Solving NCDP

- Problem of finding whether a **clique of size k** exists in the graph G is the same as the problem of

  finding whether there is a **vertex cover** of size |V| – k in G’ i.e. (n-k)

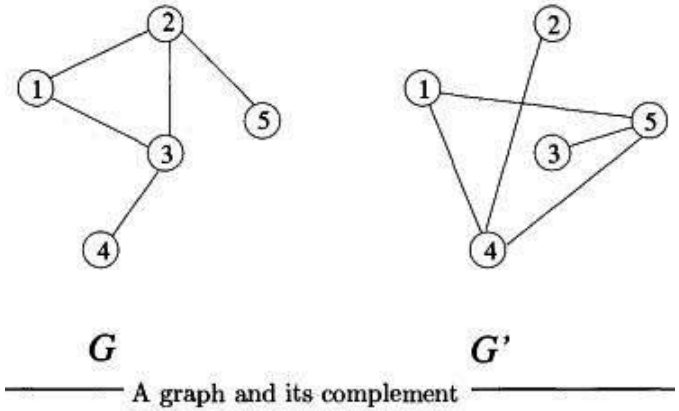
Finding whether there is a **vertex cover** of size (n-k) in G’



A graph and its complement

- Assume that there is a **clique of size k** in G.
- Let the set of vertices in the clique be V’.
- This means |V’| = k, say 3
- In the complement graph G’, let us pick any edge (u, v)., say <3,5>
- Then at least one of u or v must be in the set V – V’.
  - This is because, if both u and v were from the set V’, then the edge (u, v) would belong to V’, which, in turn would mean that the edge (u, v) is in G.
  - This is not possible since (u, v) is not in G.
  - Thus, all edges in G’ are covered by vertices in the set V – V’.
- Now assume that there is a vertex cover V’’ of size |V| – k i.e. (n-k)=2 in G’. V’’={4,5}
- This means that all edges in G’ are connected to some vertex in V’’.
- As a result, if we pick any edge (u, v) from G’, both of them cannot be outside the set V’’.
- This means, all edges (u, v) such that both u and v are outside the set V’’ are in G, i.e., these edges constitute a clique of size k.

…contd…Q)
Show that Clique Decision Problem ->p Vertex Cover Decision Problem



A graph and its complement

3 Clique Decision Problem for 3 k's: Does graph G have clique of size at least **k = 2, 3, 4 ?**

Ans: = TRUE if G' has node cover(vertex cover) of size at most (n-k) i.e. **3, 2, 1** respectively for the 3 k's

Now, we show that G has a clique of size at least k if and only if G' has a node cover of size at most $n - k$. Let K be any clique in G. Since there are no edges in $\bar{E}$ connecting vertices in K, the remaining $n - |K|$ vertices in G' must cover all edges in $\bar{E}$. Similarly, if S is a node cover of G', then $V - S$ must form a complete subgraph in G.

Since G' can be obtained from G in polynomial time, CDP can be solved in polynomial deterministic time if we have a polynomial time deterministic algorithm for NCDP.

Unit IV	Intractable Problems and NP-Completeness

Time-Space trade off,

Tractable and Non-tractable Problems,

Polynomial and non-polynomial problems,

deterministic and non-deterministic algorithms,

P-class problems, NP-class of problems,

Polynomial problem reduction,

NP complete problems- 3-SAT, Clique Decision Problem and Vertex cover

NP hard problem - Hamiltonian cycle

3-CNF SAT ->p Hamiltonian Path ->p Hamiltonian Cycle

Self Study / Done Later

References

- https://www.cs.ucc.ie/~dgb/courses/toc
- https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/threeSAT_to_clique.html
- https://www.geeksforgeeks.org/proof-that-vertex-cover-is-np-complete/

Thank You!!