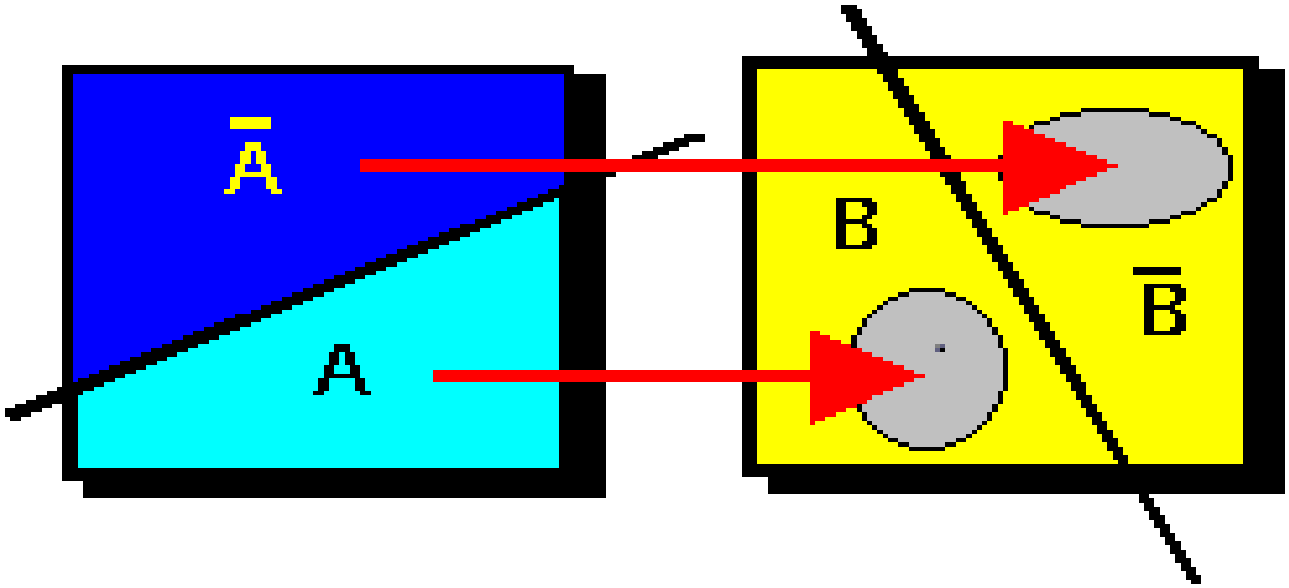
NP complete

- *The class of **nondeterministic polynomially acceptable problems**, **NP**, contains all sets in which membership can be verified in polynomial time.*
- *For every set A in NP there is a polynomial p(n) such that the problem of determining whether a data item of size n is a member of A can be solved in 2p(n) time.*

- A useful tool in studying the relationships between members of a class is the translation or mapping of one to another.
- If we can translate one set into another, we can often deduce properties of one by the properties that we know the other possesses. This is called reducibility.

- **Definition.** *The set A is **many-one polynomial-time reducible** to the set B (this is written as A £$_p$ B) if and only if there is a recursive function g(x) which can be computed in polynomial time such that for all x: x Î A if and only if g(x) Î B.*



- Note that all of the members of A map into a portion of B and all elements not in A map into a part of B's complement. This gives us a way to solve membership in A if we know how to solve membership in B. If A is reducible to B via the function g(x), then all we need do to determine if x is in A is to check to see if g(x) is in B.

List of the many problems that are members of NP, and are in NP-complete

- **0-1 Integer Programming (0-1 INT).** *Given a matrix A and a vector b, is there a vector x with values from {0, 1} such that Ax ³ b?*
- **CLIQUE**. *Given a graph and an integer k, are there k vertices in the graph which are all adjacent to each other?*
- **Vertex Cover (VC).** *Given a graph and an integer k, is there a collection of k vertices such that each edge is connected to one of the vertices in the collection?*
- **Chromatic Number (COLOR).** *Given a graph and an integer k, is there a way to color the vertices with k colors such that adjacent vertices are colored differently?*

List - contd

- **Examination Scheduling (EXAM).** *Given a list of courses, a list of conflicts between them, and an integer k; is there an exam schedule consisting of k dates such that there are no conflicts between courses which have examinations on the same date?*
- **Closed Tour (TOUR).** *Given n cities and an integer k, is there a tour, of length less than k, of the cities which begins and ends at the same city?*
- **Rectilinear Steiner Spanning Tree (STEINER).** *Given n points in Euclidean space and an integer k, is there a collection of vertical and horizontal lines of total length less than k, which spans the points?*
- **Knapsack.** *Given n items, each with a weight and a value, and two integers k and m, is there a collection of items with total weight less than k, which has a total value greater than m?*

- Any problem which is NP-complete is a candidate for approximation since no subexponential time bounded algorithms are known for these problems.
- The process of proving a problem NP-complete:

  1. show that the problem is in NP

  2. reduce an NP-complete problem to it

  3. show that the reduction is a polynomial time function.

Problem:

- **Satisfiability with 3 literals per clause (3-SAT).** *Given a finite set of clauses, each containing exactly three literals, is there some truth assignment for the variables which satisfies all of the clauses?*
- **Theorem .** *3-SAT is NP-complete.*

- we examine the clauses below which are made up of literals from the set of Boolean variables $\{v_1, ..., v_n\}$.

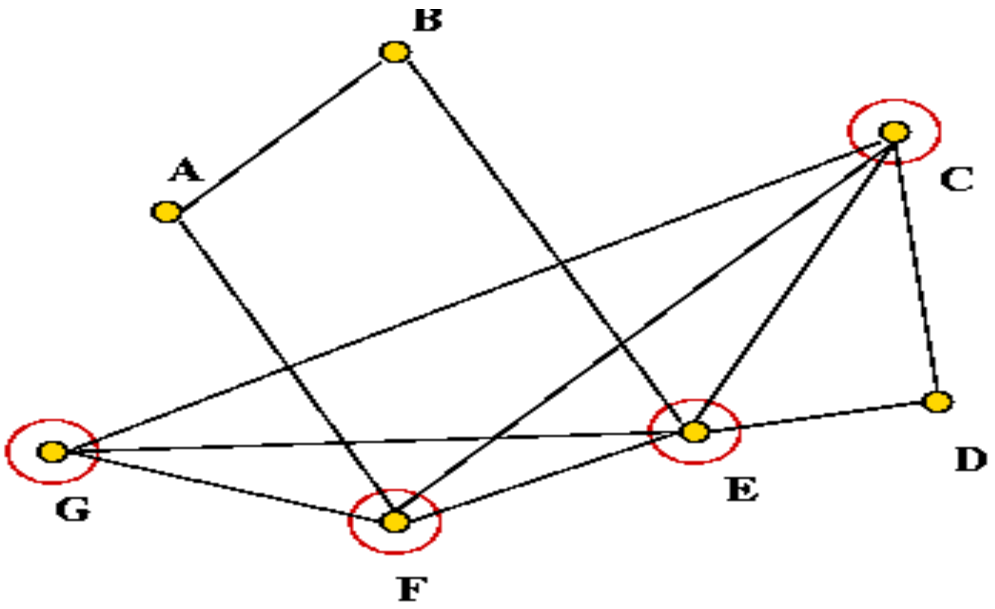
$$(v_1, \overline{v_2}, v_3), (v_2), (\overline{v_1}, v_3)$$

- The first clause is satisfiable if either v_1 or v_3 are true or v_2 is false. Now let us consider at the entire collection of clauses. All three are true (at once) when all three variables are true. Thus we shall say that a collection of clauses is *satisfiable if and only if there is some assignment of truth values to the variables which makes all of the clauses true simultaneously.*
- The collection: $(v_1, \overline{v_2}, v_3), (v_2), (\overline{v_2}, \overline{v_3}), (\overline{v_1}, v_3)$
- is not satisfiable because at least one of the three clauses will be false no matter how the truth values are assigned to the variables. Now for the first decision problem which is NP-complete. It is central to theorem proving procedures and the propositional calculus.
- **The Satisfiability Problem (SAT).** *Given a set of clauses, is there an assignment of truth values to the variables such that the collection of clauses is satisfiable?*

- This was the first problem to be proven as NP and NP Complete. This was proved by Scientist “ Cook” with the help of Turing machines.
- & Hence used in further problems to prove that they belong to NP Complete class.

Problem to prove by reduction: Clique

- What is a *clique*? In a graph, a *clique* is a subset of vertices with the property: there's an edge between every pair of vertices

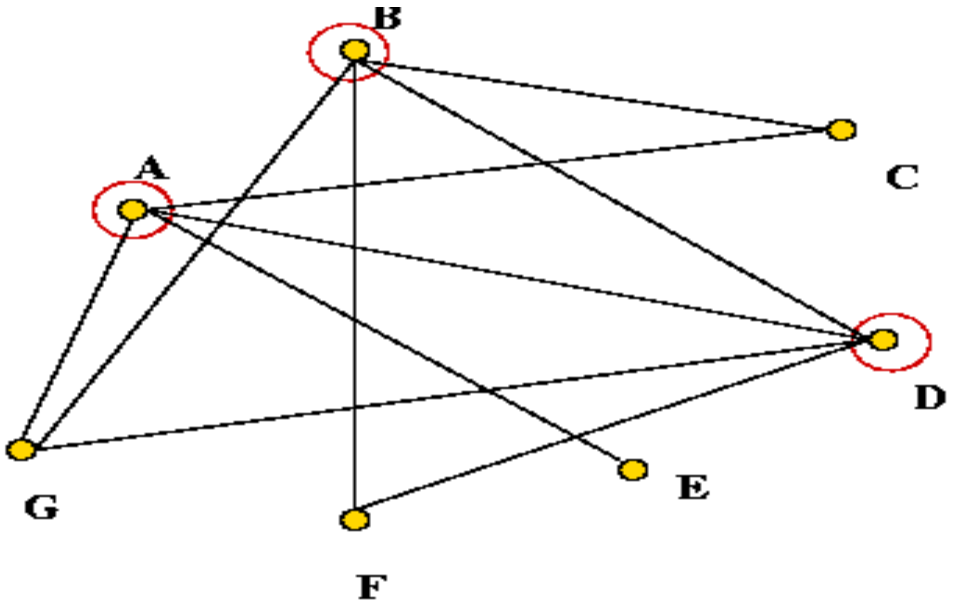


Vertices C, E, F and G form a clique of size 4

- Optimization version: Given a graph *G=(V,E)* find the largest-size clique in the graph.
- Decision version: Given a graph *G=(V,E)* and a number *K*, is there a clique of size at least *K*?

Problem: Vertex cover

- What is a *vertex cover*?
  - In a graph, it's a subset of vertices such that every edge is incident to at least one of these vertices.
  - Example:



Vertices A, B and D form a vertex cover of size 3

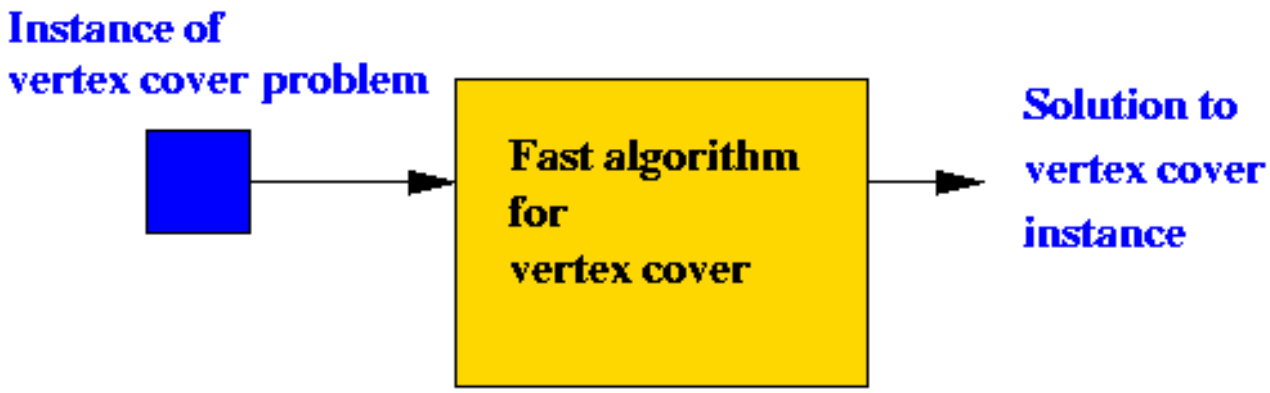
- Optimization version: Given a graph *G=(V,E)* find the smallest-size vertex cover in the graph.
- Decision version: Given a graph *G=(V,E)* and a number *M* is there a vertex cover of size at most *M*?

Transforming

- **Transforming an Instance of one Problem into an Instance of Another Problem**

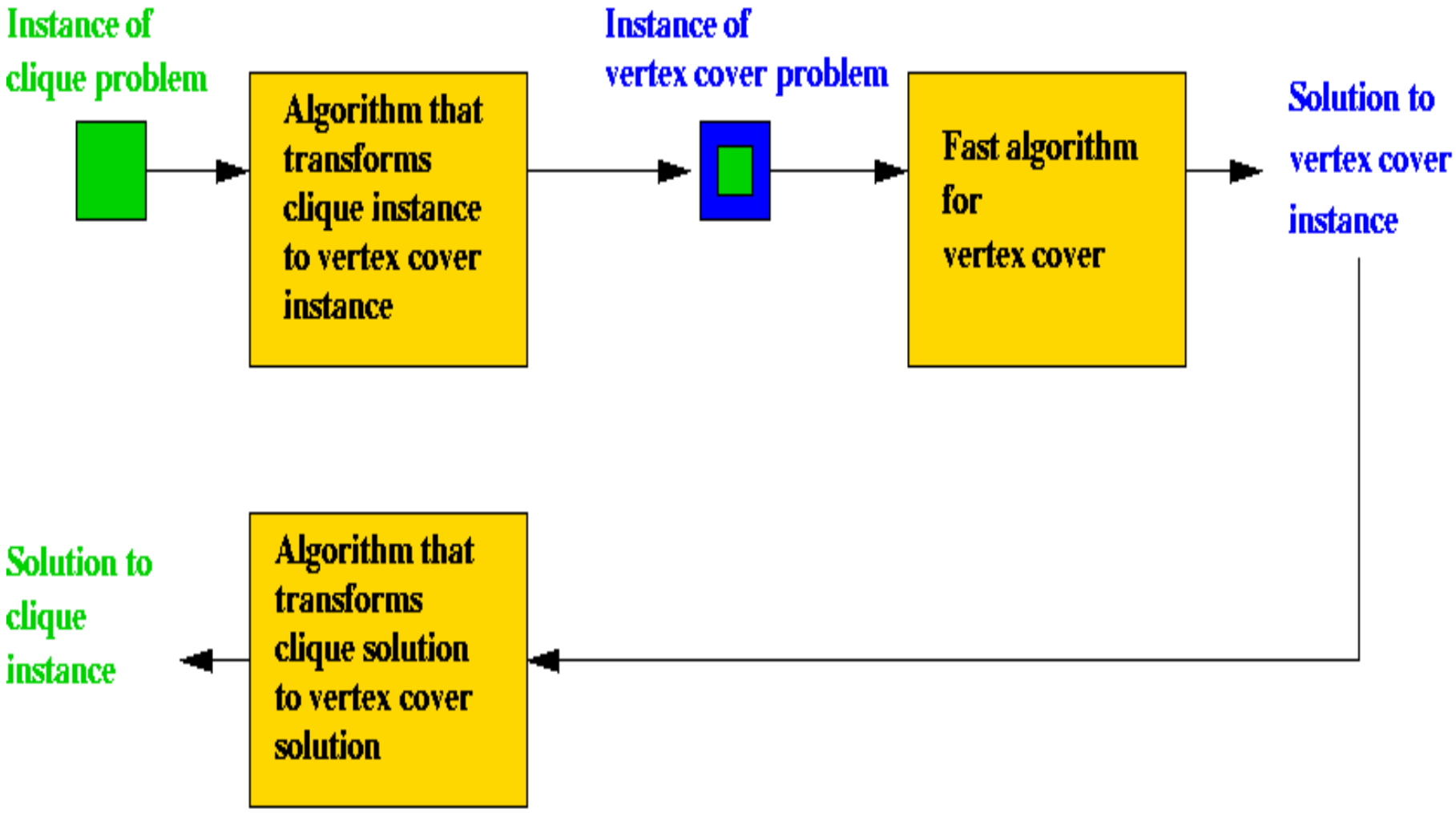
- Motivation:
- Suppose that: We can transform a *problem instance* of **clique** into an equivalent *problem instance* of **vertex cover**.
- We can transform a solution to the **vertex cover** instance into a solution for the **clique** instance.

- If a fast algorithm was found for **Vertex cover**:



- we could use it to solve **clique**:

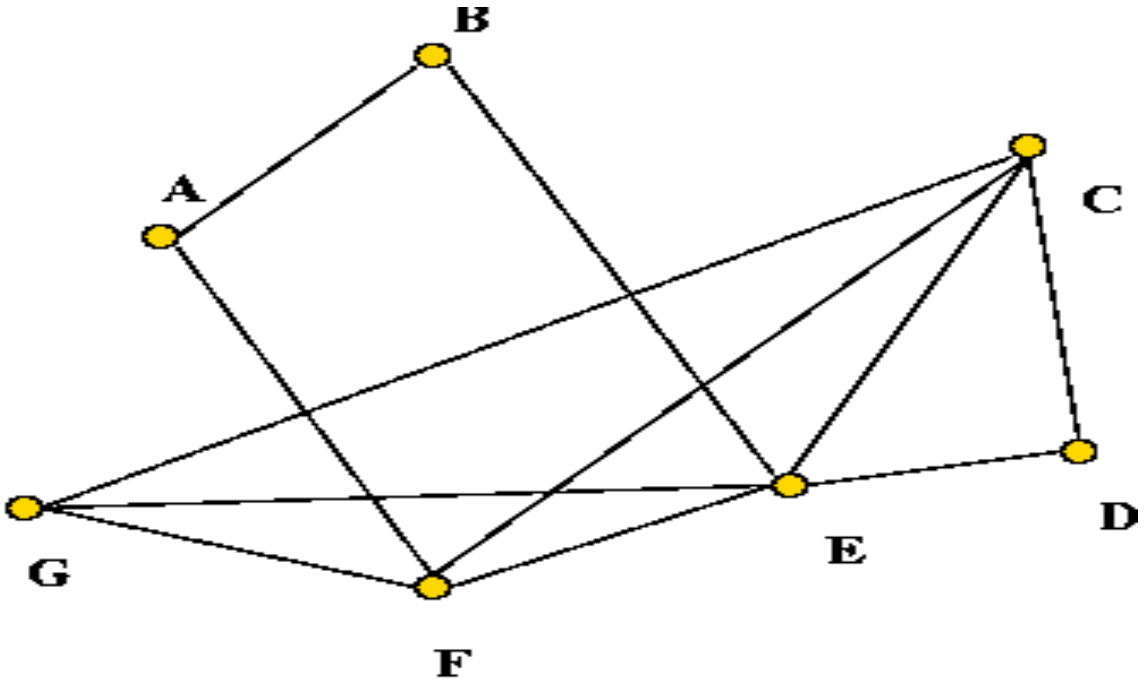
we could use it to solve **clique**



Example: transforming **clique** to **vertex cover**

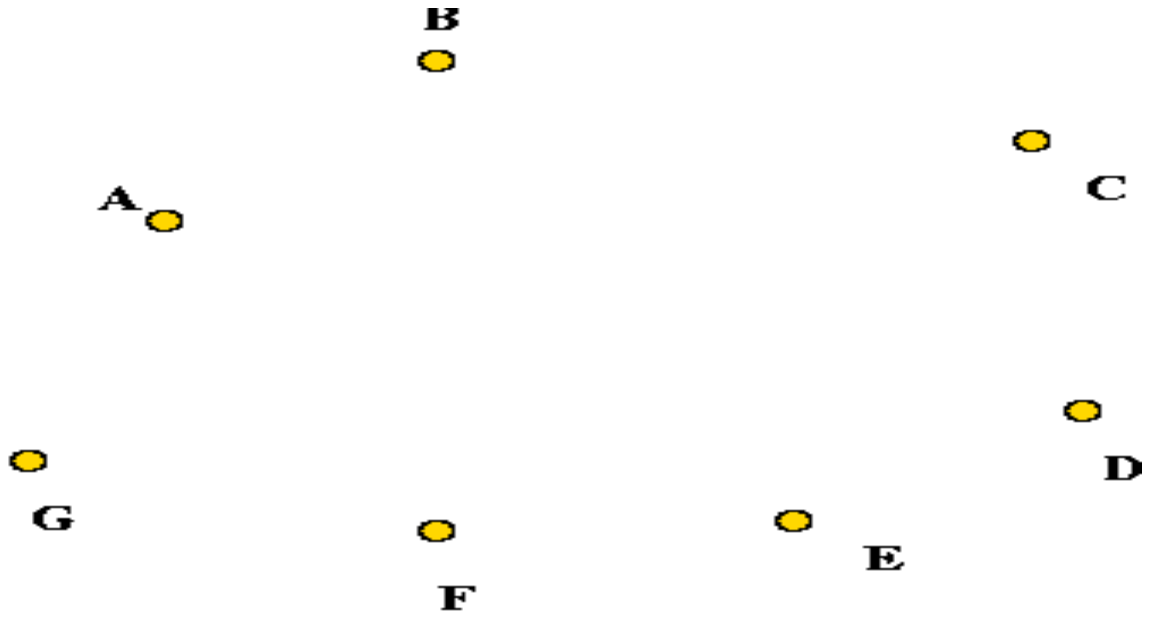
- Suppose we are given an instance of the **clique** problem, *G = (V, E)*.
  => Goal is to find largest clique.

- Example:



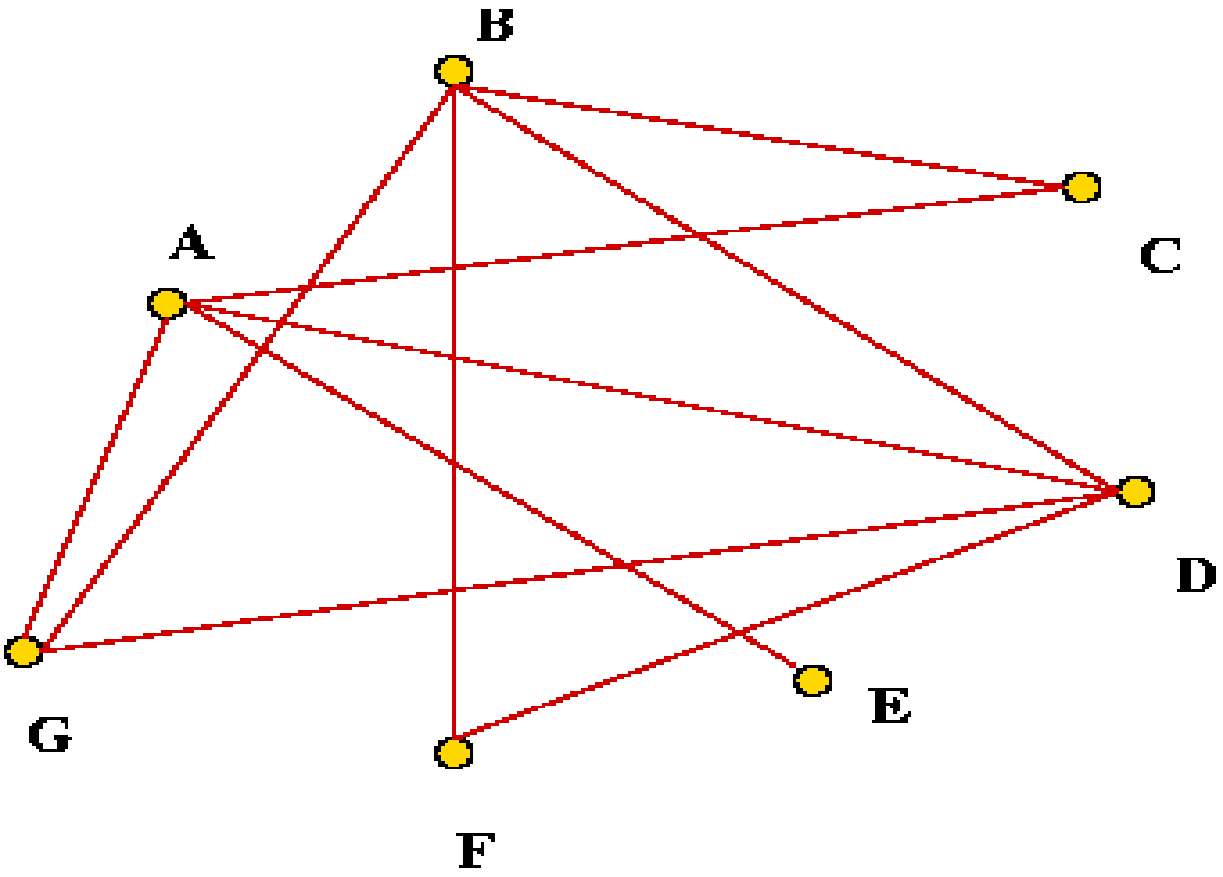
1.

- Define *G′*, the complement of *G* as follows: *G′* has the same vertex set *V* of *G*.



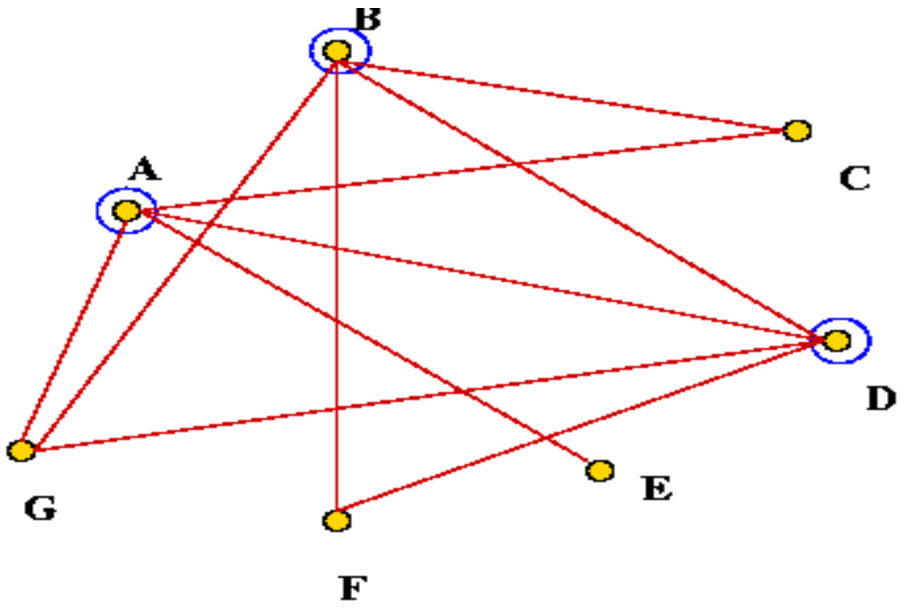
2.

For every two vertices, *u, v* not connected in *G*, place an edge in *G'*.



3.

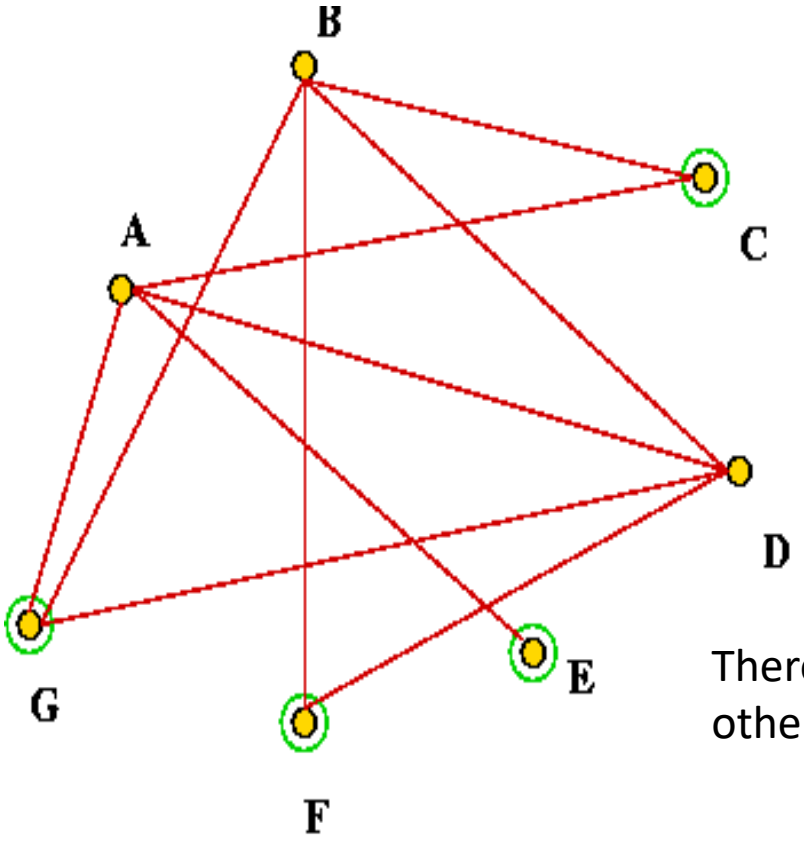
- Claim: if *G'* has a vertex cover of size *k* then *G* has a clique of size *n* - *k*: Suppose *G'* has vertex cover of size *k* (*k = 3* in example).



Vertex cover: A, B and D.

4.

- Consider the other vertices:

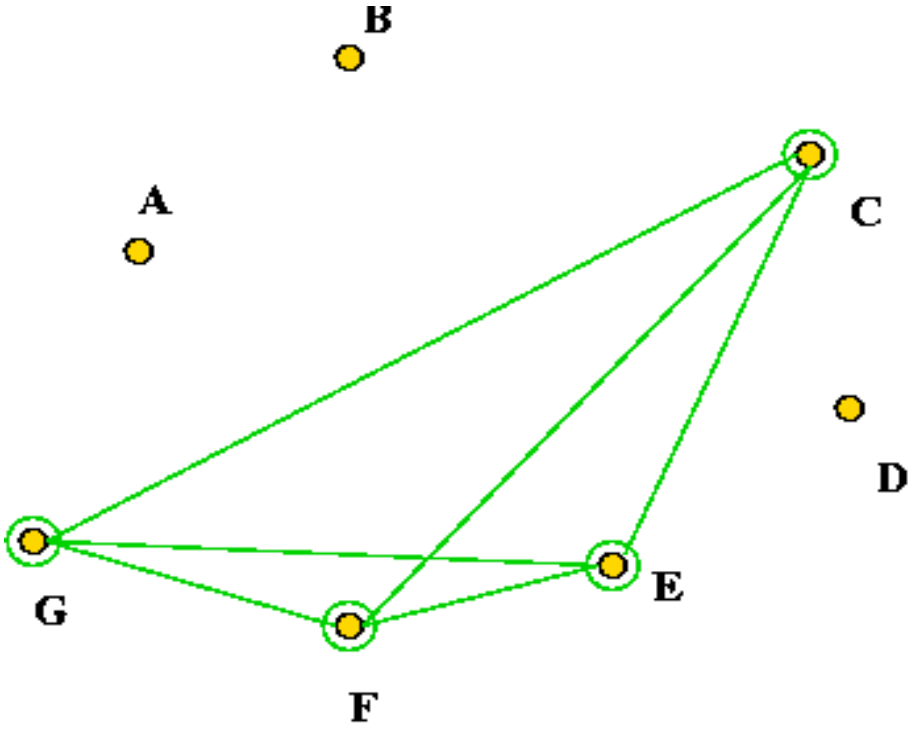


Vertices in G' not in the vertex cover
(no edges between them)

There are no edges between them (because all other edges are "covered" by the vertex cover).

5.

- This means, there MUST be an edge between every such pair in *G*:



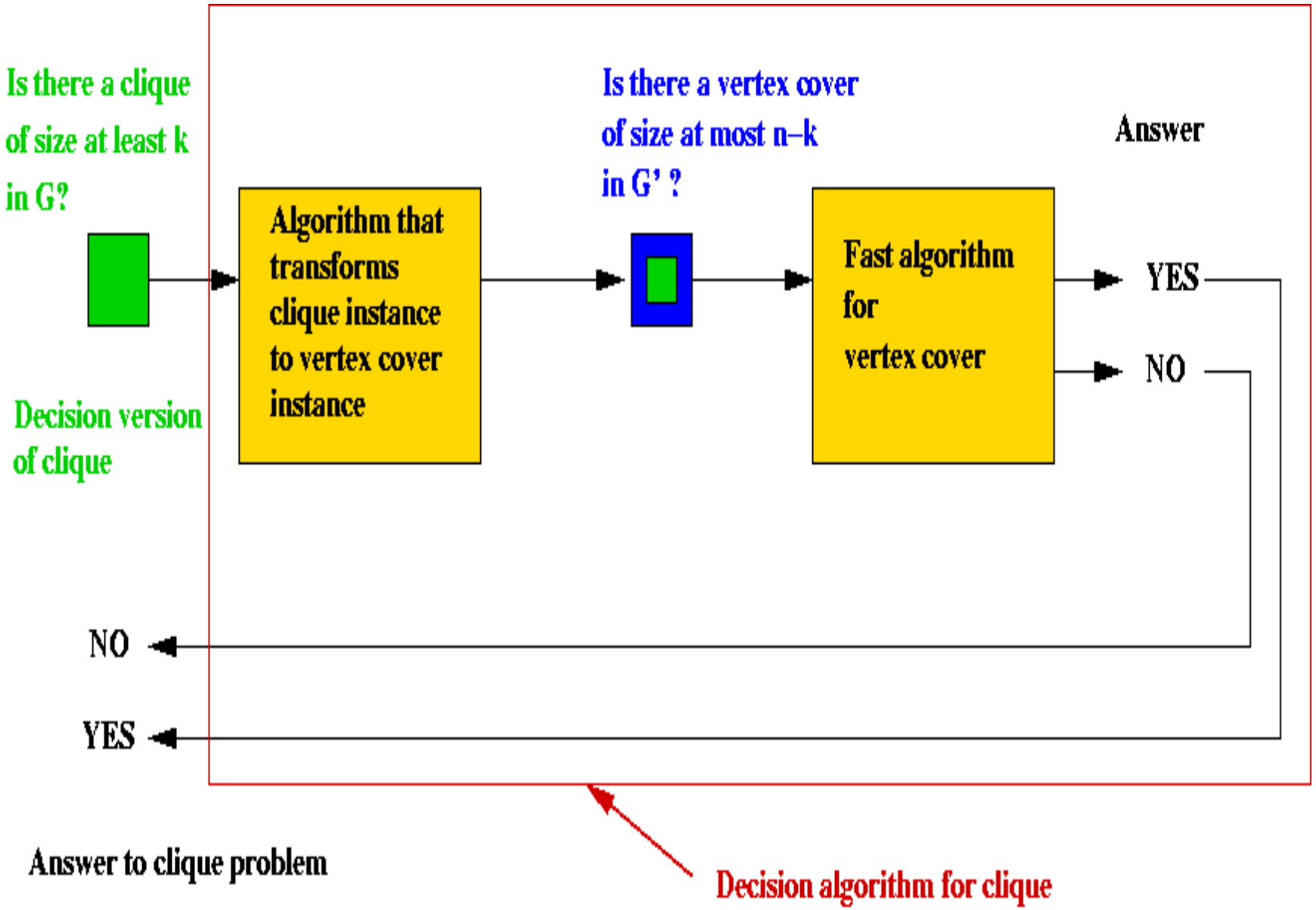
Clique of size 4 in G

- Thus, there is a clique of size *n* - *k* in *G*.

- Performing the transformation has two parts:
  - Transforming *G* to *G'*:
    - Processes each pair of vertices once
      => $O(n^2)$ time to transform an instance of **clique** into an instance of **vertex cover**.
  - Transform **vertex cover** solution to **clique** solution:
    - Build clique graph
      => $O(n^2)$ time (worst-case).
- Thus, both transformations take polynomial-time.

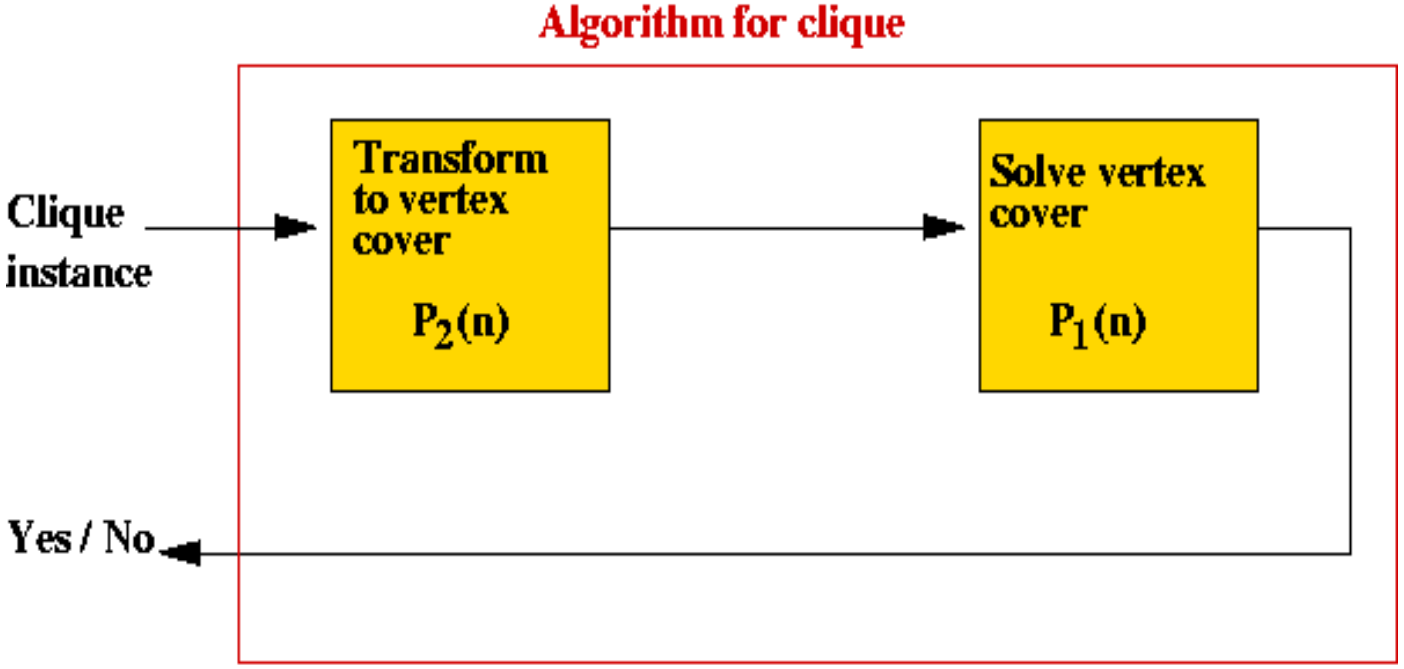
Decision version

- Decision version of **clique**: Is there a clique of size at least *k*?
- Decision version of **vertex cover**: Is there a vertex cover of size at most *m*?
- What we have shown: size-*(n-k)* vertex-cover in *G'* <==> size-*k* clique in *G*.
- Thus, we can create a solution for the decision version of clique:


Is there a clique of size at least k in G?
Decision version of clique
Algorithm that transforms clique instance to vertex cover instance
Is there a vertex cover of size at most n–k in G' ?
Fast algorithm for vertex cover
Answer
YES
NO
NO
YES
Answer to clique problem
Decision algorithm for clique

Time taken

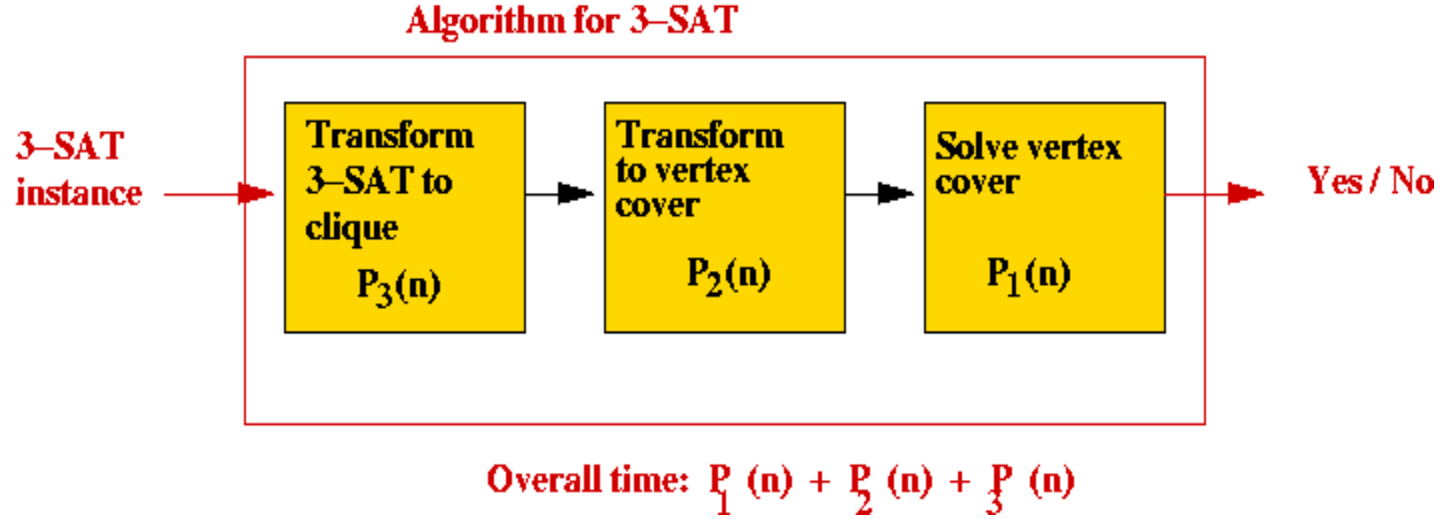
- Suppose vertex-cover algorithm takes time $P_1(n)$.
- suppose the transformation takes time $P_2(n)$.
- We have a clique algorithm that takes time $P_1(n) + P_2(n)$, still a polynomial.



Overall time: $P_1(n) + P_2(n)$

Known Transformations

- **Satisfiability (3-SAT)** to **clique**



- **Hamiltonian-tour** to **3-SAT**.