NP COMPLETE

NP - Complete

- Problem P is said to be NPC if

1. $P \in NP$, and
2. $Q \leq_p P \; \forall \; Q \in NP$

- That is, the problem is in NP and every other problem in NP is polynomial time reducible to P so that P is at least as hard as any other problem in NP.

Why Prove NP-Completeness?

- Though nobody has proven that **P != NP**, if you prove a problem NP-Complete, most people accept that it is probably intractable
- Therefore it can be important to prove that a problem is NP-Complete
  - Don’t need to come up with an efficient algorithm
  - Can instead work on *approximation algorithms*

Proving NP-Completeness

- *What steps do we have to take to prove a problem P is NP-Complete?*
  – Pick a known NP-Complete problem Q
  – Reduce Q to P
    - Describe a transformation that maps instances of Q to instances of P, s.t. "yes" for P = "yes" for Q
    - Prove the transformation works
    - Prove it runs in polynomial time
  – Oh yeah, prove P $\in$ **NP** (*What if you can't?*)

The SAT Problem

- One of the first problems to be proved NP-Complete was *satisfiability* (SAT):
  - Given a Boolean expression on *n* variables, can we assign values such that the expression is TRUE?
  - Ex: $((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$
  - **Cook's Theorem:** The satisfiability problem is NP-Complete
    - **Note: Argue from first principles, not reduction**
    - **Proof: not here**

Conjunctive Normal Form

- Even if the form of the Boolean expression is simplified, the problem may be NP-Complete
  - *Literal*: an occurrence of a Boolean or its negation
  - A Boolean formula is in *conjunctive normal form*, or *CNF*, if it is an AND of clauses, each of which is an OR of literals
    - Ex: $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_5)$
  - *3-CNF*: each clause has exactly 3 distinct literals
    - Ex: $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_5 \vee x_3 \vee x_4)$
    - Notice: true if at least one literal in each clause is true

The 3-CNF Problem

- Thm 36.10: Satisfiability of Boolean formulas in 3-CNF form (the *3-CNF Problem*) is NP-Complete
  - Proof: Nope
- The reason we care about the 3-CNF problem is that it is relatively easy to reduce to others
  - Thus by proving 3-CNF NP-Complete we can prove many seemingly unrelated problems NP-Complete

3-CNF → Clique

- *What is a clique of a graph G?*
- A: a subset of vertices fully connected to each other, i.e. a complete subgraph of G
- The *clique problem*: how large is the maximum-size clique in a graph?
- *Can we turn this into a decision problem?*
- A: Yes, we call this the *k-clique problem*
- *Is the k-clique problem within **NP**?*

Clique is in NP

- CLIQUE = {<G,k>: G has a clique of size k}
- For x = <G,k> in CLIQUE, does there exist a certificate y: |y| = polynomial in the length of x and a polynomial time algorithm that can use y to verify that x is in CLIQUE?
  - Show the existence of y,
  - Show that |y| = polynomial in the length of |x|,
  - Give an algorithm that verifies x using y,
  - Show that the algorithm runs in polynomial time in |y| and |x| and hence in polynomial time in the length of |x|.

SO, FOUR STEPS TO SHOW THAT A PROBLEM IS IN NP

CLIQUE is in NPC

- 2nd step to show that CLIQUE is in NPC is
- Pick up a problem known to be NPC and
  - Transform (reduce) the known problem to CLIQUE
  - 0 Give the transformation
    1. Show that under the transformation : solution of known problem is yes => solution to CLIQUE is yes.
    2. Show that under the transformation : solution of CLIQUE is yes => solution of the known problem is yes.
    3. Show that the transformation can be done in time polynomial in the length of an instance of the known problem.

SO, THREE STEPS TO REDUCE A KNOWN PROBLEM TO CLIQUE.

3-CNF $\rightarrow$ Clique

- *What should the reduction do?*
- A: Transform a 3-CNF formula to a graph, for which a *k*-clique will exist (for some *k*) iff the 3-CNF formula is satisfiable

3-CNF $\rightarrow$ Clique

- The reduction:
  - Let $B = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ be a 3-CNF formula with *k* clauses, each of which has 3 distinct literals
  - For each clause put a triple of vertices in the graph, one for each literal
  - Put an edge between two vertices if they are in different triples and their literals are *consistent*, meaning not each other's negation
  - Run an example:
    $B = (x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (x \vee y \vee z)$

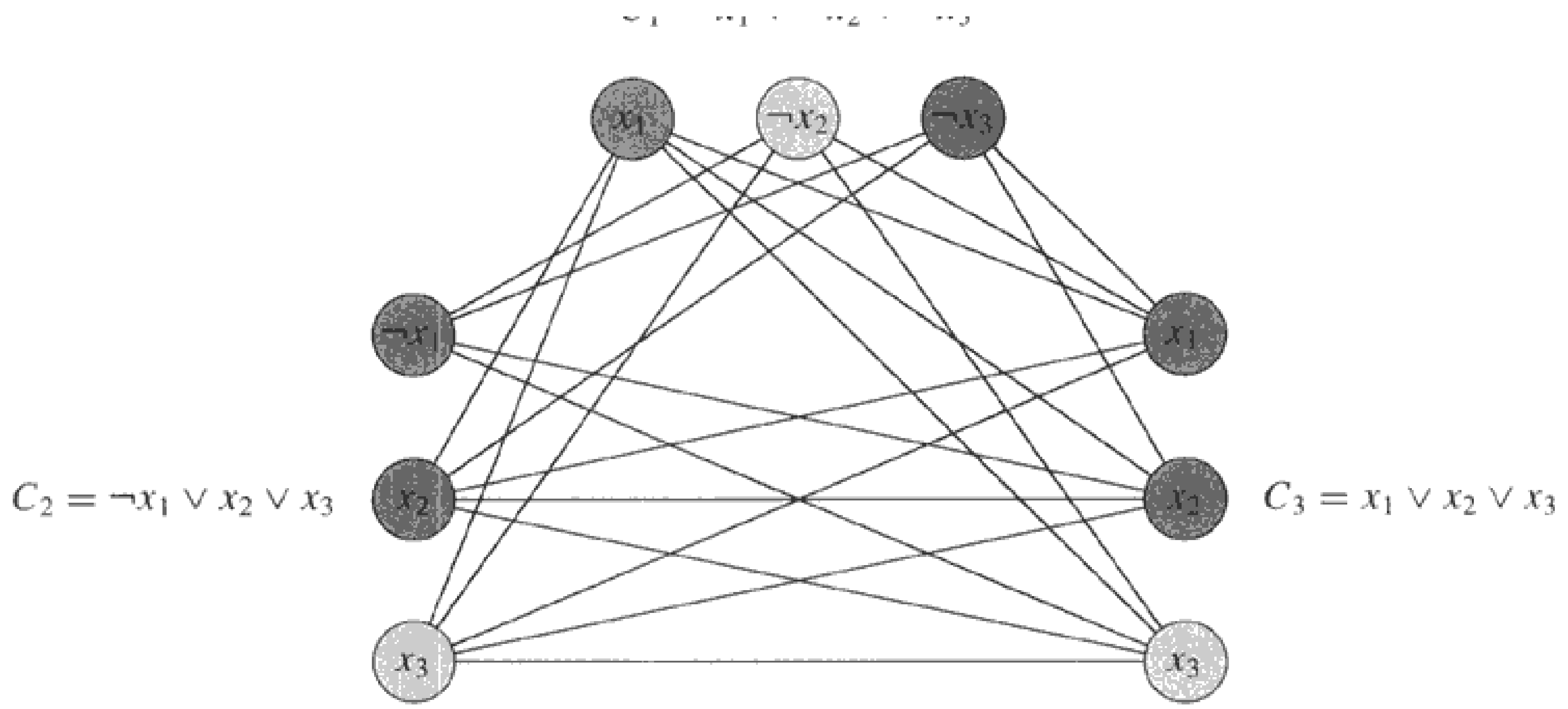

x_1
$\neg x_2$
$\neg x_3$
$\neg x_1$
x_2
x_3
x_1
x_2
x_3
$C_2 = \neg x_1 \vee x_2 \vee x_3$
$C_3 = x_1 \vee x_2 \vee x_3$

Figure 34.14 The graph G derived from the 3-CNF formula $\phi = C_1 \wedge C_2 \wedge C_3$, where $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$, $C_2 = (\neg x_1 \vee x_2 \vee x_3)$, and $C_3 = (x_1 \vee x_2 \vee x_3)$, in reducing 3-CNF-SAT to CLIQUE. A satisfying assignment of the formula has $x_2 = 0$, $x_3 = 1$, and x_1 may be either 0 or 1. This assignment satisfies C_1 with $\neg x_2$, and it satisfies C_2 and C_3 with x_3, corresponding to the clique with lightly shaded vertices.

3-CNF $\rightarrow$ Clique

- Prove the reduction works:
  - If B has a satisfying assignment, then each clause has at least one literal (vertex) that evaluates to 1
  - Picking one such “true” literal from each clause gives a set V’ of *k* vertices. V’ is a clique (*Why?*)
  - If G has a clique V’ of size k, it must contain one vertex in each triple (clause) (*Why?*)
  - We can assign 1 to each literal corresponding with a vertex in V’, without fear of contradiction

Reduction takes polynomial time

- Let there be n variables in the 3-CNF with k clauses
- Then, the input size is at least(>=) B = max{k,n}
- Any algorithm at most(<=) polynomial in B is polynomial in the input size.
- Creating 3k vertices with no more than k^2 edges with n variables takes no more than max{k^2 max{log 3k, log n}, n log n} time …a polynomial in n and k and hence in B.

Make life simpler with an assumption for future

- From now on we understand that a graph G with |V| vertices and |E| edges can be created and represented in time polynomial in |V| and |E|.
- Hence in future we’ll just show that |V| and |E| are polynomial in the input size of …..
- You can use this in the exam.

Vertex Cover Problem

- A *vertex cover* for a graph G is a set of vertices incident to every edge in G
- The *vertex cover problem*: what is the minimum size vertex cover in G?
- Restated as a decision problem: does a vertex cover of size *k* exist in G?
- Thm 36.12: vertex cover is NP-Complete

VC is in NP

- How?

- Four steps

  --- Show the existence of y,
  - Show that |y| = polynomial in the length of |x|,
  - Give an algorithm that verifies x using y,
  - Show that the algorithm runs in polynomial time in |y| and |x| and hence in polynomial time in the length of |x|.

Pick up a problem known in NPC

– CLIQUE

Clique → Vertex Cover

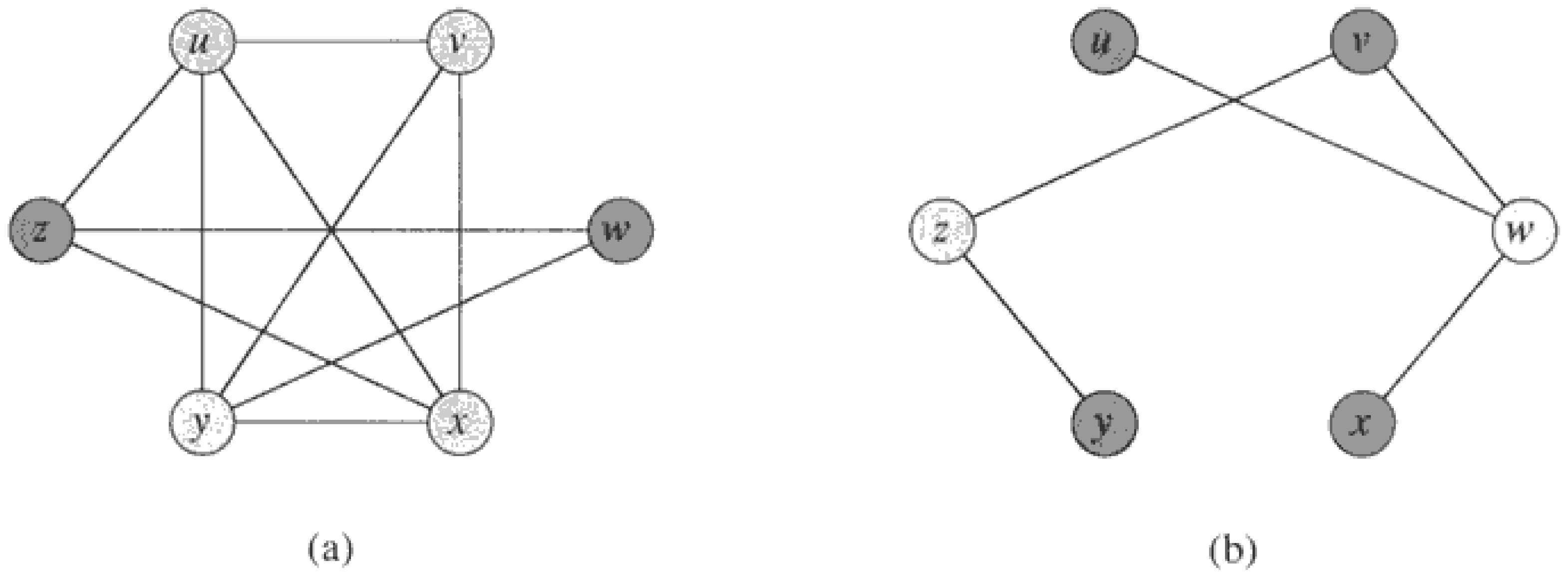


Figure 34.15 Reducing CLIQUE to VERTEX-COVER. **(a)** An undirected graph $G = (V, E)$ with clique $V' = \{u, v, x, y\}$. **(b)** The graph $\overline{G}$ produced by the reduction algorithm that has vertex cover $V - V' = \{w, z\}$.

Clique $\rightarrow$ Vertex Cover

- Reduce *k*-clique to vertex cover
  - The *complement* G_C of a graph G contains exactly those edges not in G
  - Compute G_C in polynomial time
  - G has a clique of size *k* iff G_C has a vertex cover of size $|V| - k$

Clique $\rightarrow$ Vertex Cover

- Claim: If G has a clique of size *k*, G_C has a vertex cover of size |V| - *k*
  - Let V' be the *k*-clique
  - Then V - V' is a vertex cover in G_C
    - Let (*u*,*v*) be any edge in G_C
    - Then *u* and *v* cannot both be in V' (*Why?*)
    - Thus at least one of *u* or *v* is in V-V' (*why?*), so edge (*u*, *v*) is covered by V-V'
    - Since true for *any* edge in G_C, V-V' is a vertex cover

Clique $\rightarrow$ Vertex Cover

- Claim: If G_C has a vertex cover $V' \subseteq V$, with $|V'| = |V| - k$, then G has a clique of size k
  - For all $u,v \in V$, if $(u,v) \in G_C$ then $u \in V'$ or $v \in V'$ or both (*Why?*)
  - Contrapositive: if $u \notin V'$ and $v \notin V'$, then $(u,v) \in E$
  - In other words, all vertices in V-V' are connected by an edge, thus V-V' is a clique
  - Since $|V| - |V'| = k$, the size of the clique is k

Vertex Cover → HCP

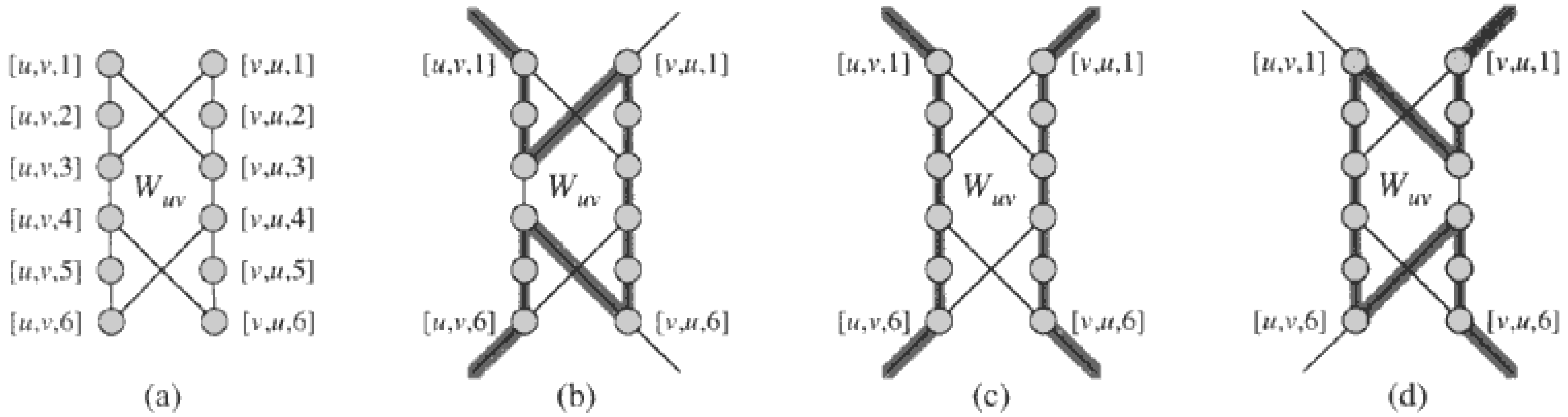
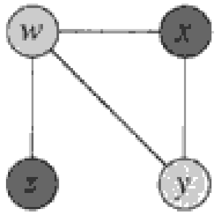


Figure 34.16 The widget used in reducing the vertex-cover problem to the hamiltonian-cycle problem. An edge (u, v) of graph G corresponds to widget W_{uv} in the graph G' created in the reduction. **(a)** The widget, with individual vertices labeled. **(b)–(d)** The shaded paths are the only possible ones through the widget that include all vertices, assuming that the only connections from the widget to the remainder of G' are through vertices $[u, v, 1]$, $[u, v, 6]$, $[v, u, 1]$, and $[v, u, 6]$.



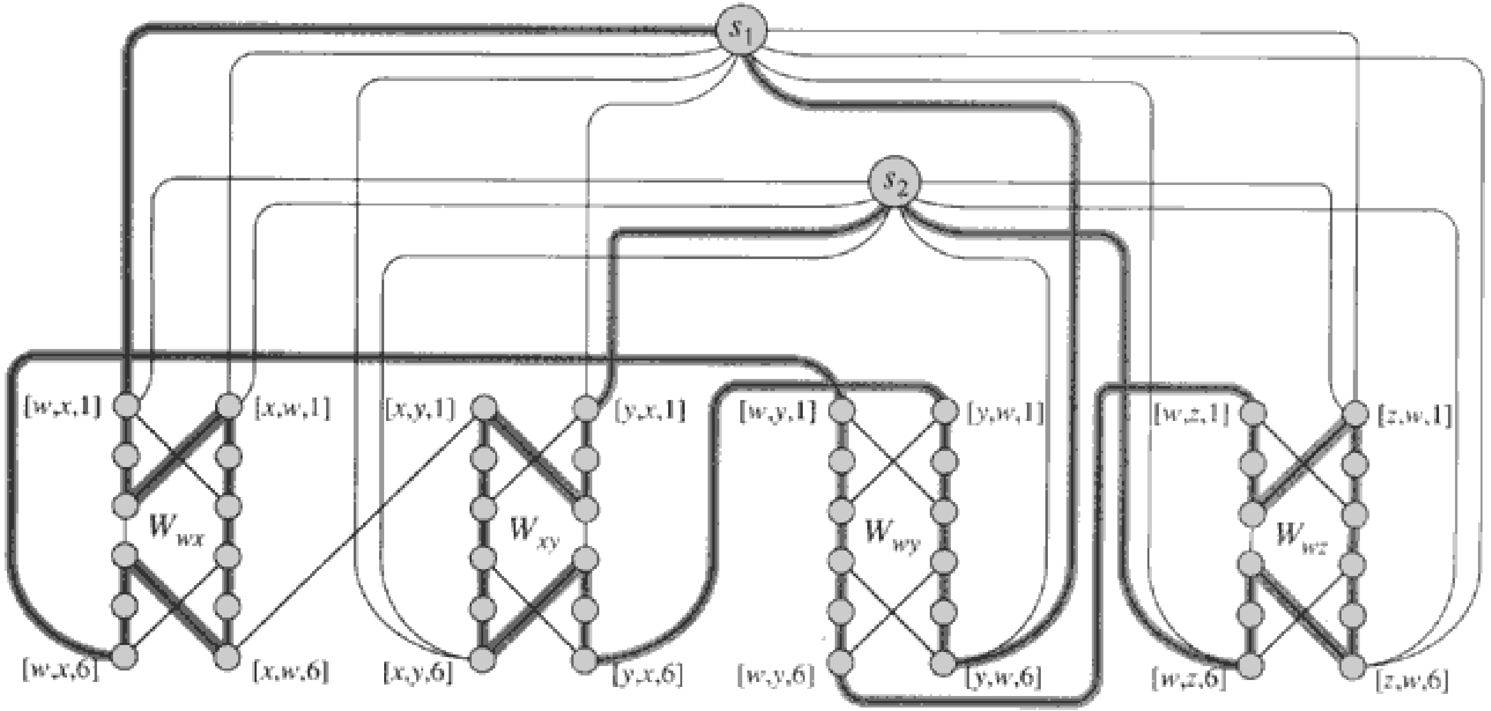


Figure 34.17 The reduction of an instance of the vertex-cover problem to an instance of the hamiltonian-cycle problem. **(a)** An undirected graph G with a vertex cover of size 2, consisting of the lightly shaded vertices w and y. **(b)** The undirected graph G' produced by the reduction, with the hamiltonian path corresponding to the vertex cover shaded. The vertex cover $\{w, y\}$ corresponds to edges $(s_1, [w, x, 1])$ and $(s_2, [y, x, 1])$ appearing in the hamiltonian cycle.

Application & scope of research of NP-Complete Problems

- *Subset-sum*: Given a set of integers, does there exist a subset that adds up to some target *T*?
- *0-1 knapsack*: when weights not just integers
- *Hamiltonian path*: Obvious
- *Graph coloring*: can a given graph be colored with *k* colors such that no adjacent vertices are the same color?
- *Scope of research is finding an algorithm for these problem whose run time complexity is Polynomial time.*

Assignment

Q.1)Show that clique is NP Complete problem.

Q.2)How to prove that problem is NP Complete