COMP 761: Lecture 34 – Computational Complexity II

David Rolnick

November 23, 2020

Problem

Given an undirected, unweighted graph G, a *vertex cover* of G is a set S of vertices such that every edge of G has at least one of its endpoints in S (possibly both endpoints). Is it possible to efficiently find the smallest vertex cover in a graph?

(Please don't post your ideas in the chat just yet, we'll discuss the problem soon in class.)

Course Announcements

- Office hours after class

Review: P and NP

Review: P and NP

- Class P: the answer (yes/no) can be found in polynomial time.

Review: P and NP

- Class P: the answer (yes/no) can be found in polynomial time.
- Class NP: for every “yes” answer, there is a *certificate* that proves it is indeed “yes” and can be *checked* in polynomial time.

Review: P and NP

- Class P: the answer (yes/no) can be found in polynomial time.
- Class NP: for every “yes” answer, there is a *certificate* that proves it is indeed “yes” and can be *checked* in polynomial time.
- Class co-NP: for every “no” answer, there is a *certificate* that proves it is indeed “no” and can be *checked* in polynomial time.

Review: P and NP

- Class P: the answer (yes/no) can be found in polynomial time.
- Class NP: for every "yes" answer, there is a *certificate* that proves it is indeed "yes" and can be *checked* in polynomial time.
- Class co-NP: for every "no" answer, there is a *certificate* that proves it is indeed "no" and can be *checked* in polynomial time.
- We say that a problem B is *polynomial-time reducible* to a problem A if, given answers to A, we can solve B in polynomial time.

Review: P and NP

- Class P: the answer (yes/no) can be found in polynomial time.
- Class NP: for every "yes" answer, there is a *certificate* that proves it is indeed "yes" and can be *checked* in polynomial time.
- Class co-NP: for every "no" answer, there is a *certificate* that proves it is indeed "no" and can be *checked* in polynomial time.
- We say that a problem B is *polynomial-time reducible* to a problem A if, given answers to A, we can solve B in polynomial time.
- We say a problem A is *NP-complete* if (a) it is in NP, and (b) every problem B in NP is polynomial-time reducible to A.

Review: P and NP

- Class P: the answer (yes/no) can be found in polynomial time.
- Class NP: for every "yes" answer, there is a *certificate* that proves it is indeed "yes" and can be *checked* in polynomial time.
- Class co-NP: for every "no" answer, there is a *certificate* that proves it is indeed "no" and can be *checked* in polynomial time.
- We say that a problem B is *polynomial-time reducible* to a problem A if, given answers to A, we can solve B in polynomial time.
- We say a problem A is *NP-complete* if (a) it is in NP, and (b) every problem B in NP is polynomial-time reducible to A.
- If B is in NP, A is NP-complete, and A is polynomial-time reducible to B, then B is also NP-complete.

SAT

SAT

- *Boolean formulas* are expressions like this:

$$(x_1 \vee -x_2) \wedge x_3, \text{ where}$$

  - Each x_i is a Boolean variable (TRUE or FALSE),
  - $-x_i$ is the negation of x_i (TRUE if x_i is FALSE and vice versa),
  - $\vee$ means OR,
  - $\wedge$ means AND.

SAT

- *Boolean formulas* are expressions like this:

$$(x_1 \vee -x_2) \wedge x_3, \text{ where}$$

  - Each x_i is a Boolean variable (TRUE or FALSE),
  - $-x_i$ is the negation of x_i (TRUE if x_i is FALSE and vice versa),
  - $\vee$ means OR,
  - $\wedge$ means AND.
- A *Boolean satisfiability problem* is the problem of working out whether a Boolean formula has any solutions.

SAT

- *Boolean formulas* are expressions like this:

$$(x_1 \vee -x_2) \wedge x_3, \text{ where}$$

  - Each x_i is a Boolean variable (TRUE or FALSE),
  - $-x_i$ is the negation of x_i (TRUE if x_i is FALSE and vice versa),
  - $\vee$ means OR,
  - $\wedge$ means AND.
- A *Boolean satisfiability problem* is the problem of working out whether a Boolean formula has any solutions.
- For example, for the formula above, one solution is x_1=FALSE, x_2=FALSE, x_3=TRUE.

3-SAT

3-SAT

- A Boolean formula in *3-conjunctive normal form* has a series of clauses separated by $\wedge$, each clause having three variables or their negation separated by $\vee$:

$$\phi = (x_1 \vee -x_2 \vee x_4) \wedge (-x_2 \vee -x_3 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4).$$

3-SAT

- A Boolean formula in *3-conjunctive normal form* has a series of clauses separated by $\wedge$, each clause having three variables or their negation separated by $\vee$:

$$\phi = (x_1 \vee -x_2 \vee x_4) \wedge (-x_2 \vee -x_3 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4).$$

- It's possible to rewrite more general Boolean formulas in this form by rearranging and substituting variables.

3-SAT

- A Boolean formula in *3-conjunctive normal form* has a series of clauses separated by $\wedge$, each clause having three variables or their negation separated by $\vee$:

$$\phi = (x_1 \vee -x_2 \vee x_4) \wedge (-x_2 \vee -x_3 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4).$$

- It's possible to rewrite more general Boolean formulas in this form by rearranging and substituting variables.
- This standard form is sometimes more straightforward to work with.

3-SAT

- A Boolean formula in *3-conjunctive normal form* has a series of clauses separated by $\wedge$, each clause having three variables or their negation separated by $\vee$:

$$\phi = (x_1 \vee -x_2 \vee x_4) \wedge (-x_2 \vee -x_3 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4).$$

- It's possible to rewrite more general Boolean formulas in this form by rearranging and substituting variables.
- This standard form is sometimes more straightforward to work with.
- *3-SAT(ϕ)* is the problem of determining whether a formula ϕ in 3-conjunctive normal form has any solutions.

3-SAT

- A Boolean formula in *3-conjunctive normal form* has a series of clauses separated by $\wedge$, each clause having three variables or their negation separated by $\vee$:

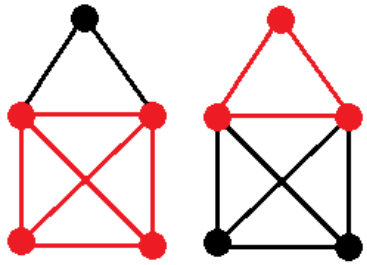
$$\phi = (x_1 \vee -x_2 \vee x_4) \wedge (-x_2 \vee -x_3 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4).$$

- It's possible to rewrite more general Boolean formulas in this form by rearranging and substituting variables.
- This standard form is sometimes more straightforward to work with.
- *3-SAT(ϕ)* is the problem of determining whether a formula ϕ in 3-conjunctive normal form has any solutions.
- 3-SAT is an NP-complete problem (though again, we won't prove this).

Cliques

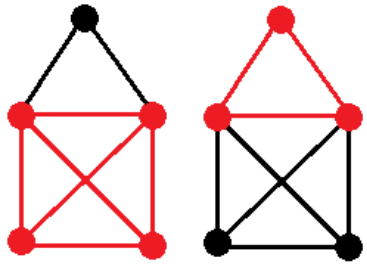
Cliques

- A *clique* in a graph G is a subset of the vertices where all the possible edges exist between them.



Cliques

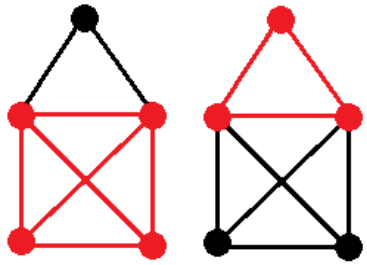
- A *clique* in a graph G is a subset of the vertices where all the possible edges exist between them.



- Given a graph G with n vertices, can we efficiently work out whether it contains a clique of size k?

Cliques

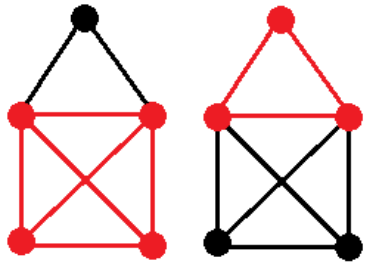
- A *clique* in a graph G is a subset of the vertices where all the possible edges exist between them.



- Given a graph G with n vertices, can we efficiently work out whether it contains a clique of size k?
- If k is small, this is doable – e.g. for $k = 3$ we can just check all $\binom{n}{3}$ groups of three vertices to see if there is a 3-clique (=triangle).

Cliques

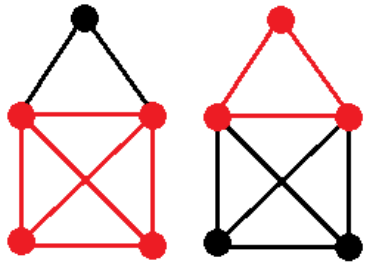
- A *clique* in a graph G is a subset of the vertices where all the possible edges exist between them.



- Given a graph G with n vertices, can we efficiently work out whether it contains a clique of size k?
- If k is small, this is doable – e.g. for $k = 3$ we can just check all $\binom{n}{3}$ groups of three vertices to see if there is a 3-clique (=triangle).
- More generally, we can check all $\binom{n}{k}$ by brute force.

Cliques

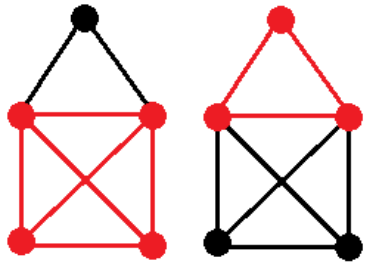
- A *clique* in a graph G is a subset of the vertices where all the possible edges exist between them.



- Given a graph G with n vertices, can we efficiently work out whether it contains a clique of size k?
- If k is small, this is doable – e.g. for $k = 3$ we can just check all $\binom{n}{3}$ groups of three vertices to see if there is a 3-clique (=triangle).
- More generally, we can check all $\binom{n}{k}$ by brute force.
- This is polynomial time if k is a constant.

Cliques

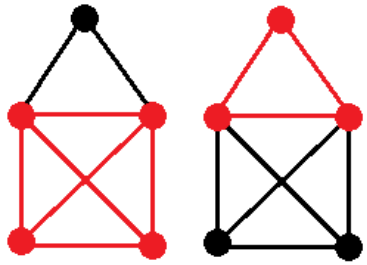
- A *clique* in a graph G is a subset of the vertices where all the possible edges exist between them.



- Given a graph G with n vertices, can we efficiently work out whether it contains a clique of size k?
- If k is small, this is doable – e.g. for $k = 3$ we can just check all $\binom{n}{3}$ groups of three vertices to see if there is a 3-clique (=triangle).
- More generally, we can check all $\binom{n}{k}$ by brute force.
- This is polynomial time if k is a constant.
- But we could have k bigger than that, e.g. $n/2$.

Cliques

- A *clique* in a graph G is a subset of the vertices where all the possible edges exist between them.



- Given a graph G with n vertices, can we efficiently work out whether it contains a clique of size k?
- If k is small, this is doable – e.g. for $k = 3$ we can just check all $\binom{n}{3}$ groups of three vertices to see if there is a 3-clique (=triangle).
- More generally, we can check all $\binom{n}{k}$ by brute force.
- This is polynomial time if k is a constant.
- But we could have k bigger than that, e.g. $n/2$.
- And $\binom{n}{n/2}$ is exponential in n.

Cliques

Cliques

- CLIQUE(G, k) is defined as the problem of whether or not there is a clique in G of size k.

Cliques

- CLIQUE(G, k) is defined as the problem of whether or not there is a clique in G of size k.
- Let's prove that CLIQUE is NP-complete by reduction from 3-SAT.

Cliques

- CLIQUE(G, k) is defined as the problem of whether or not there is a clique in G of size k.
- Let's prove that CLIQUE is NP-complete by reduction from 3-SAT.
- What's the first step?

Cliques

- CLIQUE(G, k) is defined as the problem of whether or not there is a clique in G of size k.
- Let's prove that CLIQUE is NP-complete by reduction from 3-SAT.
- CLIQUE is in NP because there exists a certificate to demonstrate a "yes" answer – namely, a k-clique is itself a certificate.

Cliques

- CLIQUE(G, k) is defined as the problem of whether or not there is a clique in G of size k.
- Let's prove that CLIQUE is NP-complete by reduction from 3-SAT.
- CLIQUE is in NP because there exists a certificate to demonstrate a "yes" answer – namely, a k-clique is itself a certificate.
- Now we will show that 3-SAT reduces to CLIQUE.

Cliques

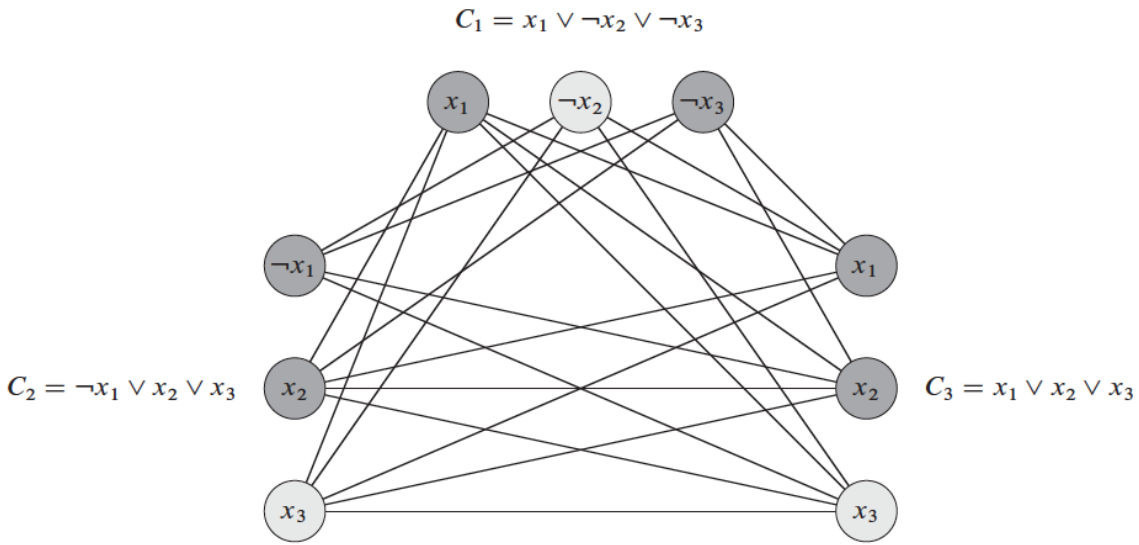
- CLIQUE(G, k) is defined as the problem of whether or not there is a clique in G of size k.
- Let's prove that CLIQUE is NP-complete by reduction from 3-SAT.
- CLIQUE is in NP because there exists a certificate to demonstrate a "yes" answer – namely, a k-clique is itself a certificate.
- Now we will show that 3-SAT reduces to CLIQUE.
- Suppose we have a formula ϕ with clauses $C_1, C_2, \ldots, C_k$.

Cliques

- CLIQUE(G, k) is defined as the problem of whether or not there is a clique in G of size k.
- Let's prove that CLIQUE is NP-complete by reduction from 3-SAT.
- CLIQUE is in NP because there exists a certificate to demonstrate a "yes" answer – namely, a k-clique is itself a certificate.
- Now we will show that 3-SAT reduces to CLIQUE.
- Suppose we have a formula ϕ with clauses $C_1, C_2, \ldots, C_k$.
- Define a graph G with 3 vertices for each clause C_i.

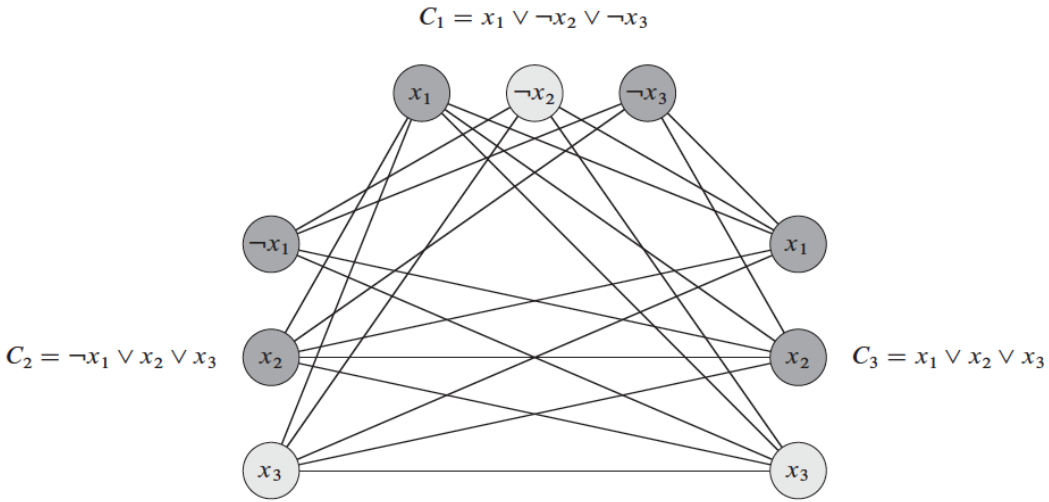
Cliques

- CLIQUE(G, k) is defined as the problem of whether or not there is a clique in G of size k.
- Let's prove that CLIQUE is NP-complete by reduction from 3-SAT.
- CLIQUE is in NP because there exists a certificate to demonstrate a "yes" answer – namely, a k-clique is itself a certificate.
- Now we will show that 3-SAT reduces to CLIQUE.
- Suppose we have a formula ϕ with clauses $C_1, C_2, \ldots, C_k$.
- Define a graph G with 3 vertices for each clause C_i.
- Two vertices are connected if they are in different clauses, unless they both represent the same variable, with one being x_j and the other $-x_j$.



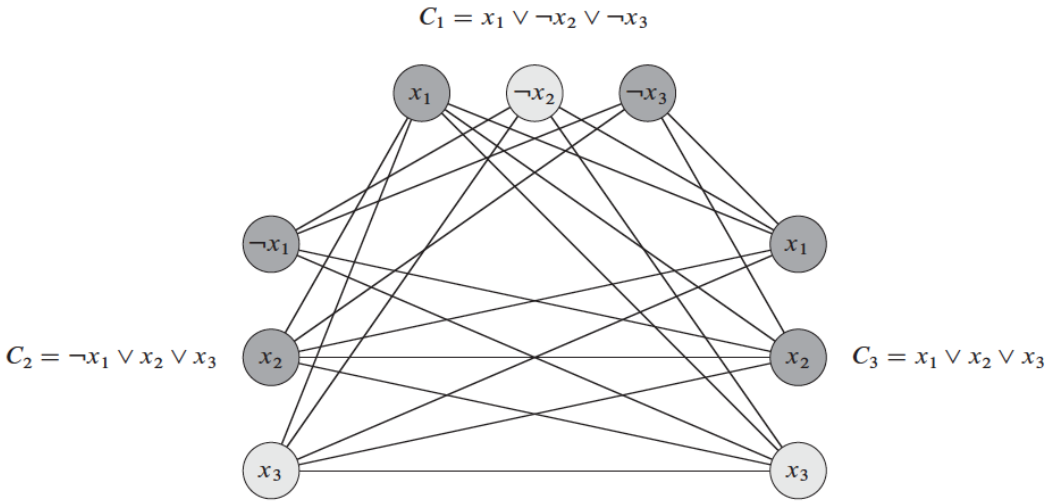
Cliques

- Two vertices are connected if they are in different clauses, unless they both represent the same variable, with one being x_j and the other $-x_j$.



Cliques

- Two vertices are connected if they are in different clauses, unless they both represent the same variable, with one being x_j and the other $-x_j$.

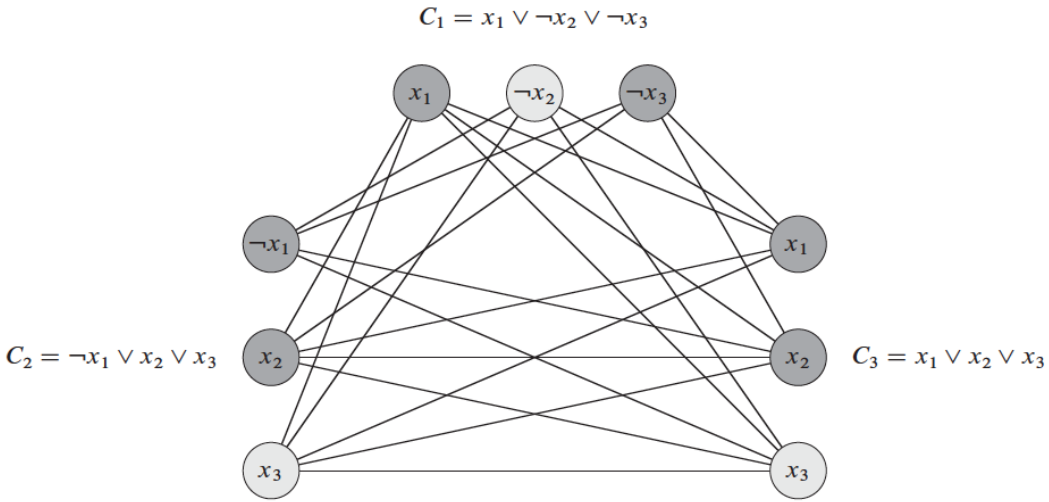


- We claim that finding a k-clique is equivalent to finding a solution to the formula ϕ, in this case:

$$\phi = (-x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee -x_2 \vee -x_3) \wedge (x_1 \vee x_2 \vee x_3).$$

Cliques

- Two vertices are connected if they are in different clauses, unless they both represent the same variable, with one being x_j and the other $-x_j$.



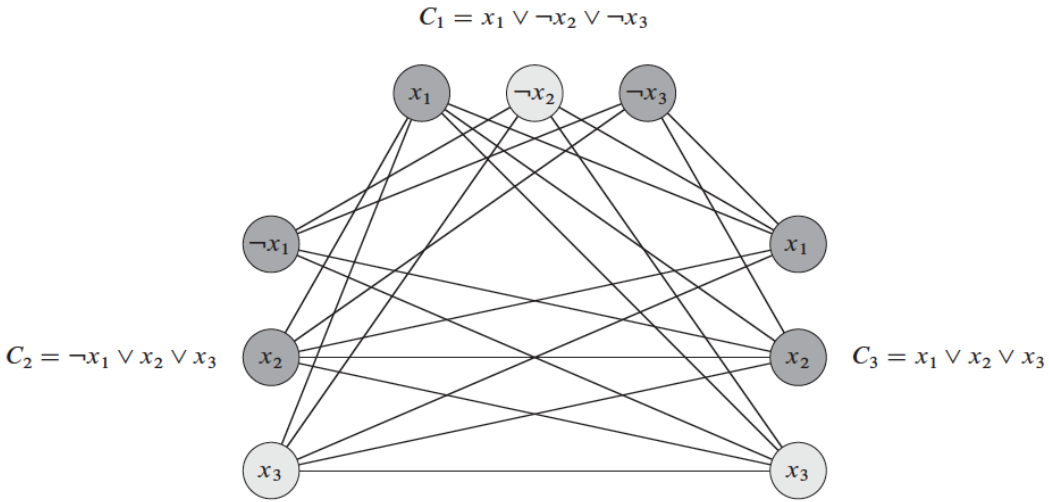
- We claim that finding a k-clique is equivalent to finding a solution to the formula ϕ, in this case:

$$\phi = (-x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee -x_2 \vee -x_3) \wedge (x_1 \vee x_2 \vee x_3).$$

- That's because a k-clique must have exactly one vertex from each clause.

Cliques

- Two vertices are connected if they are in different clauses, unless they both represent the same variable, with one being x_j and the other $-x_j$.



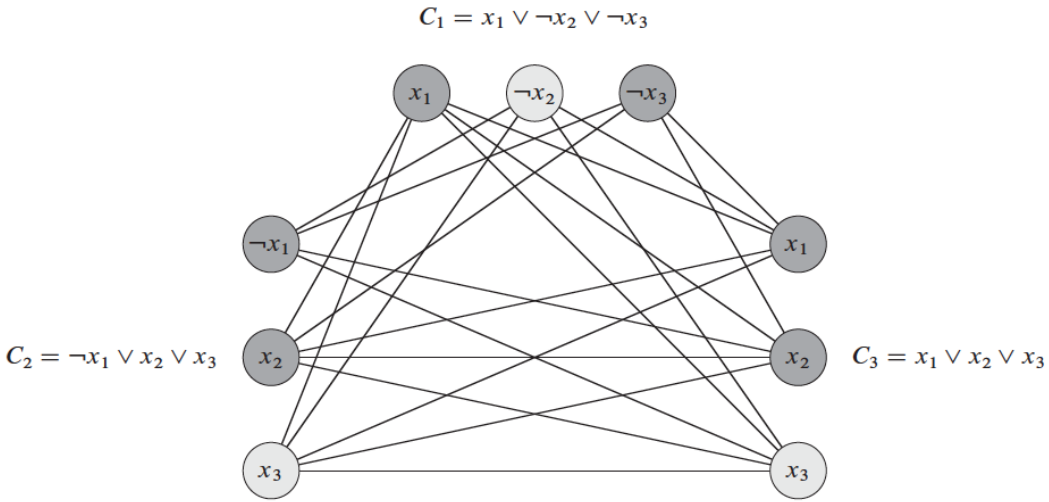
- We claim that finding a k-clique is equivalent to finding a solution to the formula ϕ, in this case:

$$\phi = (-x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee -x_2 \vee -x_3) \wedge (x_1 \vee x_2 \vee x_3).$$

- That's because a k-clique must have exactly one vertex from each clause.
- And two vertices are unconnected between different clauses exactly when they are incompatible (with one being x_j and the other $-x_j$).

Cliques

- Two vertices are connected if they are in different clauses, unless they both represent the same variable, with one being x_j and the other $-x_j$.



- We claim that finding a k-clique is equivalent to finding a solution to the formula ϕ, in this case:

$$\phi = (-x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee -x_2 \vee -x_3) \wedge (x_1 \vee x_2 \vee x_3).$$

- That's because a k-clique must have exactly one vertex from each clause.
- And two vertices are unconnected between different clauses exactly when they are incompatible (with one being x_j and the other $-x_j$).
- So we can assign TRUE to all the vertices in the clique to get a solution – and vice versa a solution gives us a clique.

Cliques

- We claim that finding a k-clique is equivalent to finding a solution to the formula ϕ, in this case:

$$\phi = (-x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee -x_2 \vee -x_3) \wedge (x_1 \vee x_2 \vee x_3).$$

- That's because a k-clique must have exactly one vertex from each clause.
- And two vertices are unconnected between different clauses exactly when they are incompatible (with one being x_j and the other $-x_j$).
- So we can assign TRUE to all the vertices in the clique to get a solution – and vice versa a solution gives us a clique.
- Therefore, answering CLIQUE(G, k) gives us an answer to 3-SAT(ϕ).

Cliques

- We claim that finding a k-clique is equivalent to finding a solution to the formula ϕ, in this case:

$$\phi = (-x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee -x_2 \vee -x_3) \wedge (x_1 \vee x_2 \vee x_3).$$

- That's because a k-clique must have exactly one vertex from each clause.
- And two vertices are unconnected between different clauses exactly when they are incompatible (with one being x_j and the other $-x_j$).
- So we can assign TRUE to all the vertices in the clique to get a solution – and vice versa a solution gives us a clique.
- Therefore, answering CLIQUE(G, k) gives us an answer to 3-SAT(ϕ).
- And it takes just polynomial time to build the graph G.

Cliques

- We claim that finding a k-clique is equivalent to finding a solution to the formula ϕ, in this case:

$$\phi = (-x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee -x_2 \vee -x_3) \wedge (x_1 \vee x_2 \vee x_3).$$

- That's because a k-clique must have exactly one vertex from each clause.
- And two vertices are unconnected between different clauses exactly when they are incompatible (with one being x_j and the other $-x_j$).
- So we can assign TRUE to all the vertices in the clique to get a solution – and vice versa a solution gives us a clique.
- Therefore, answering CLIQUE(G, k) gives us an answer to 3-SAT(ϕ).
- And it takes just polynomial time to build the graph G.
- Therefore, CLIQUE is NP-complete since 3-SAT is NP-complete.

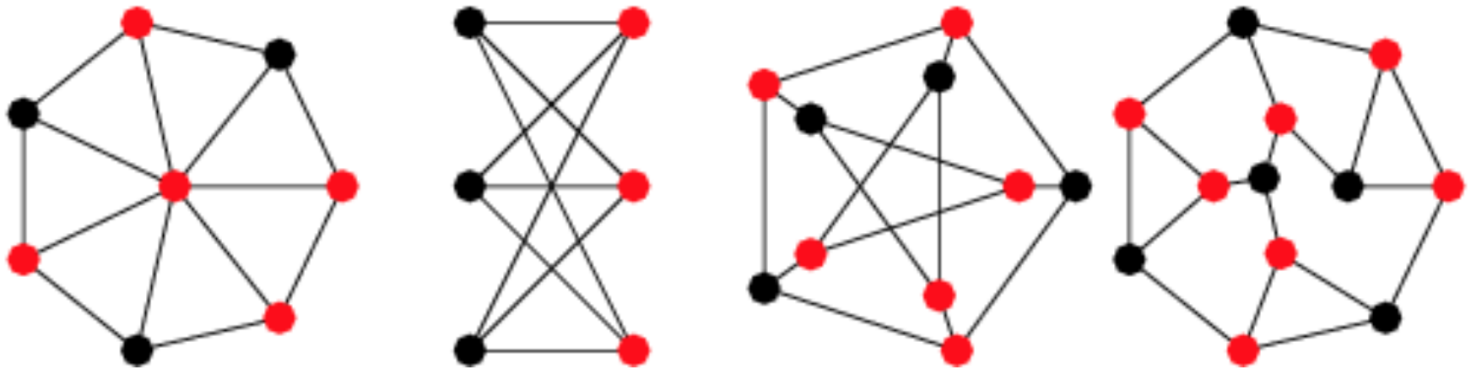
Vertex cover

Vertex cover

- Given an undirected, unweighted graph G, a *vertex cover* of G is a set S of vertices such that every edge of G has at least one of its endpoints in S (possibly both endpoints).

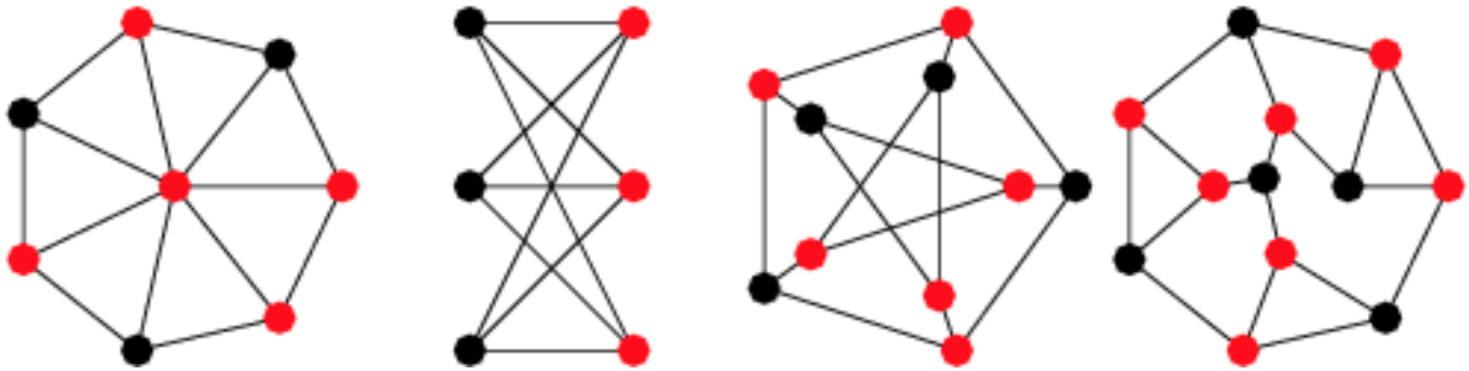
Vertex cover

- Given an undirected, unweighted graph G, a *vertex cover* of G is a set S of vertices such that every edge of G has at least one of its endpoints in S (possibly both endpoints).
- Some examples of vertex covers (shown in red):



Vertex cover

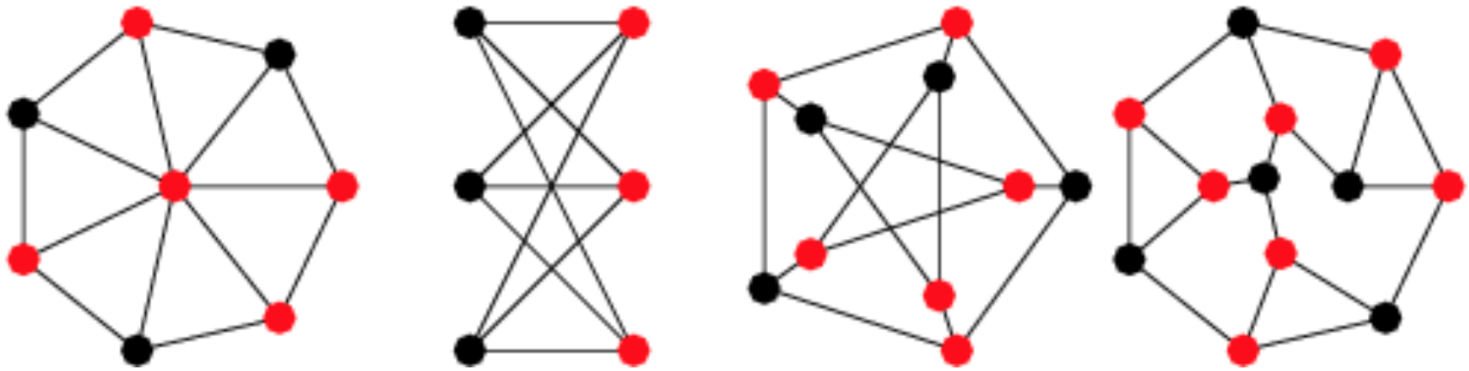
- Given an undirected, unweighted graph G, a *vertex cover* of G is a set S of vertices such that every edge of G has at least one of its endpoints in S (possibly both endpoints).
- Some examples of vertex covers (shown in red):



- Another way of thinking about it: The set of vertices *outside* the vertex cover must form an independent set (no edges between them).

Vertex cover

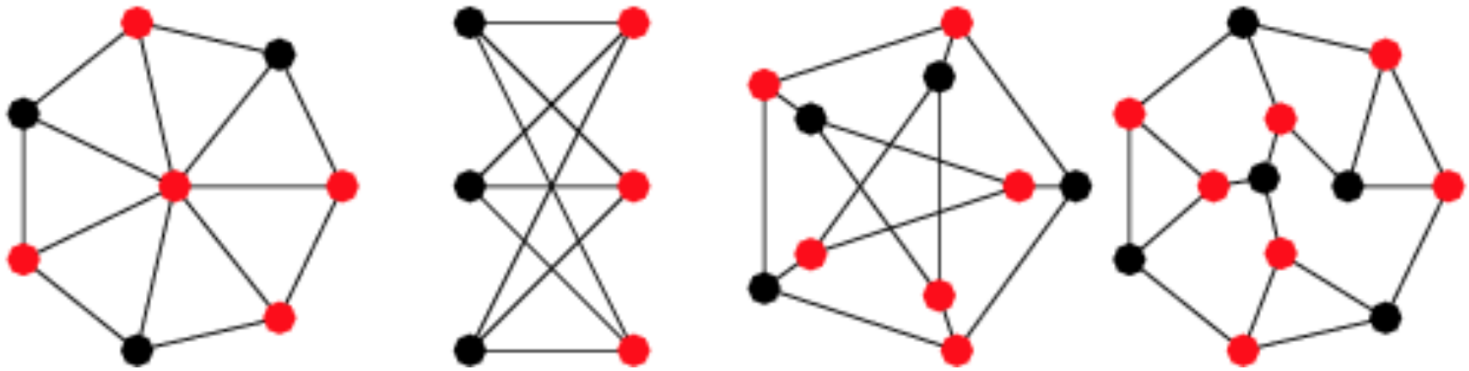
- Given an undirected, unweighted graph G, a *vertex cover* of G is a set S of vertices such that every edge of G has at least one of its endpoints in S (possibly both endpoints).
- Some examples of vertex covers (shown in red):



- Another way of thinking about it: The set of vertices *outside* the vertex cover must form an independent set (no edges between them).
- We define the problem VERTEX-COVER(G, k) as the problem of determining whether G has a vertex cover with k vertices.

Vertex cover

- Given an undirected, unweighted graph G, a *vertex cover* of G is a set S of vertices such that every edge of G has at least one of its endpoints in S (possibly both endpoints).
- Some examples of vertex covers (shown in red):



- Another way of thinking about it: The set of vertices *outside* the vertex cover must form an independent set (no edges between them).
- We define the problem VERTEX-COVER(G, k) as the problem of determining whether G has a vertex cover with k vertices.
- (If we could have an efficient algorithm for solving, we could apply it many times to work out the minimal vertex cover.)

Vertex cover

We define the problem VERTEX-COVER(G, k) as the problem of determining whether G has a vertex cover with k vertices. Prove that VERTEX-COVER is NP-complete.

Vertex cover

We define the problem VERTEX-COVER(G, k) as the problem of determining whether G has a vertex cover with k vertices. Prove that VERTEX-COVER is NP-complete.

- What's a good first step?

Vertex cover

We define the problem VERTEX-COVER(G, k) as the problem of determining whether G has a vertex cover with k vertices. Prove that VERTEX-COVER is NP-complete.

- We first need to show that it's in NP. What is a good certificate?

Vertex cover

We define the problem VERTEX-COVER(G, k) as the problem of determining whether G has a vertex cover with k vertices. Prove that VERTEX-COVER is NP-complete.

- We first need to show that it's in NP. What is a good certificate?
- A vertex cover of size k is itself a certificate. Easy to check.

Vertex cover

We define the problem VERTEX-COVER(G, k) as the problem of determining whether G has a vertex cover with k vertices. Prove that VERTEX-COVER is NP-complete.

- We first need to show that it's in NP. What is a good certificate?
- A vertex cover of size k is itself a certificate. Easy to check.
- Now what general strategy can we use?

Vertex cover

We define the problem VERTEX-COVER(G, k) as the problem of determining whether G has a vertex cover with k vertices. Prove that VERTEX-COVER is NP-complete.

- We first need to show that it's in NP. What is a good certificate?
- A vertex cover of size k is itself a certificate. Easy to check.
- We can find a polynomial-time reduction from a problem we know is NP-complete.

Vertex cover

We define the problem VERTEX-COVER(G, k) as the problem of determining whether G has a vertex cover with k vertices. Prove that VERTEX-COVER is NP-complete.

- We first need to show that it's in NP. What is a good certificate?
- A vertex cover of size k is itself a certificate. Easy to check.
- We can find a polynomial-time reduction from a problem we know is NP-complete.
- Let's try CLIQUE. We need to show we can solve CLIQUE using solutions to VERTEX-COVER.

Vertex cover

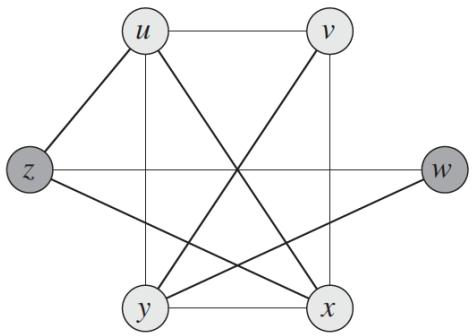
We define the problem VERTEX-COVER(G, k) as the problem of determining whether G has a vertex cover with k vertices. Prove that VERTEX-COVER is NP-complete.

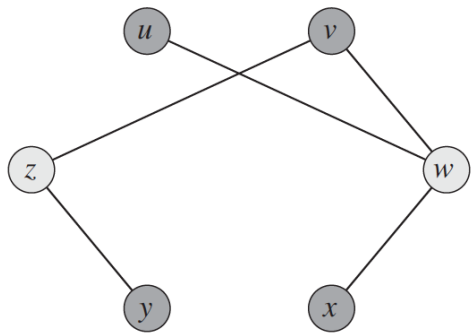
- We first need to show that it's in NP. What is a good certificate?
- A vertex cover of size k is itself a certificate. Easy to check.
- We can find a polynomial-time reduction from a problem we know is NP-complete.
- Let's try CLIQUE. We need to show we can solve CLIQUE using solutions to VERTEX-COVER.
- Suppose we have to work out if a graph G has a k-clique.

Vertex cover

We define the problem VERTEX-COVER(G, k) as the problem of determining whether G has a vertex cover with k vertices. Prove that VERTEX-COVER is NP-complete.

- We first need to show that it's in NP. What is a good certificate?
- A vertex cover of size k is itself a certificate. Easy to check.
- We can find a polynomial-time reduction from a problem we know is NP-complete.
- Let's try CLIQUE. We need to show we can solve CLIQUE using solutions to VERTEX-COVER.
- Suppose we have to work out if a graph G has a k-clique.
- Let's take the *complement* H of G – flipping edges and non-edges:

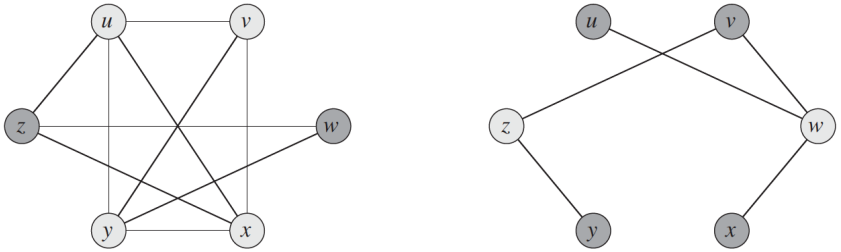




Vertex cover

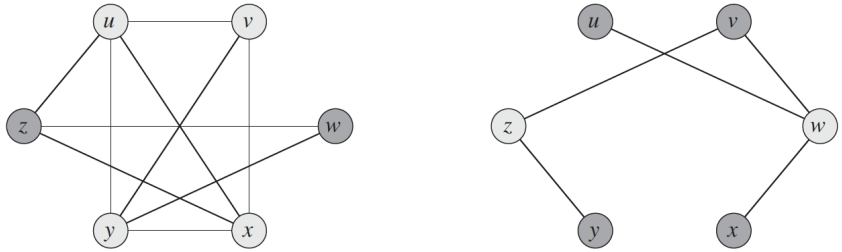
Vertex cover

- Suppose we have to work out if a graph G has a k-clique.
- Let's take the *complement* H of G – flipping edges and non-edges:



Vertex cover

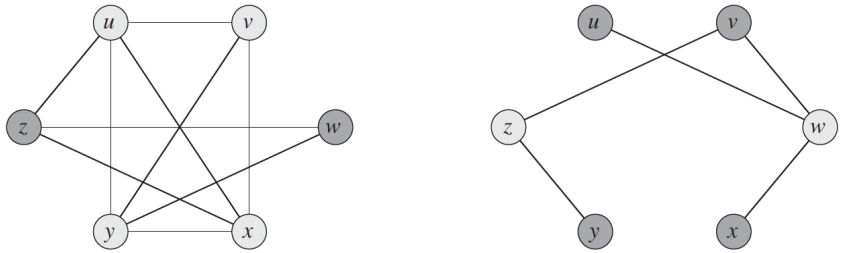
- Suppose we have to work out if a graph G has a k-clique.
- Let's take the *complement* H of G – flipping edges and non-edges:



- So a k-clique in G corresponds to an independent set of size k in H – i.e. a set of k vertices with no edges between them (here $k = 4$).

Vertex cover

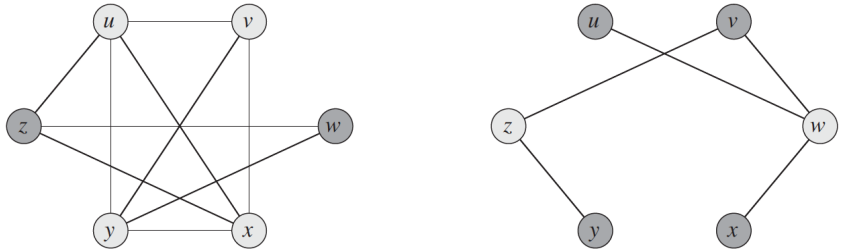
- Suppose we have to work out if a graph G has a k-clique.
- Let's take the *complement* H of G – flipping edges and non-edges:



- So a k-clique in G corresponds to an independent set of size k in H – i.e. a set of k vertices with no edges between them (here $k = 4$).
- How can we determine whether there is a k-clique?

Vertex cover

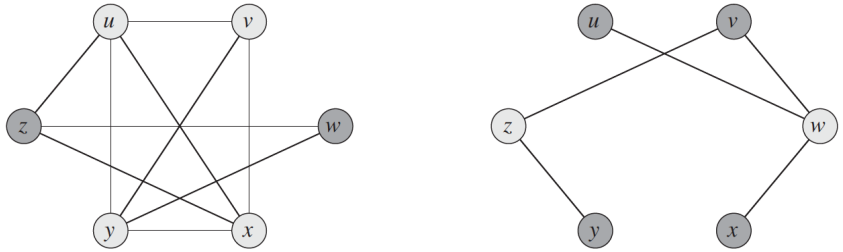
- Suppose we have to work out if a graph G has a k-clique.
- Let's take the *complement* H of G – flipping edges and non-edges:



- So a k-clique in G corresponds to an independent set of size k in H – i.e. a set of k vertices with no edges between them (here $k = 4$).
- Run VERTEX-COVER on $H, n - k$, where n is the total # of vertices.

Vertex cover

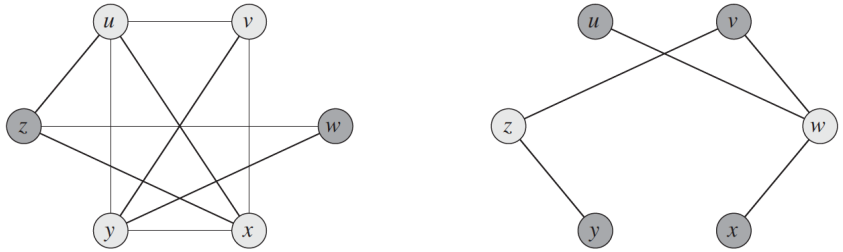
- Suppose we have to work out if a graph G has a k-clique.
- Let's take the *complement* H of G – flipping edges and non-edges:



- So a k-clique in G corresponds to an independent set of size k in H – i.e. a set of k vertices with no edges between them (here $k = 4$).
- Run VERTEX-COVER on $H, n - k$, where n is the total # of vertices.
- We claim VERTEX-COVER$(H, n - k)$ = CLIQUE(G, k).

Vertex cover

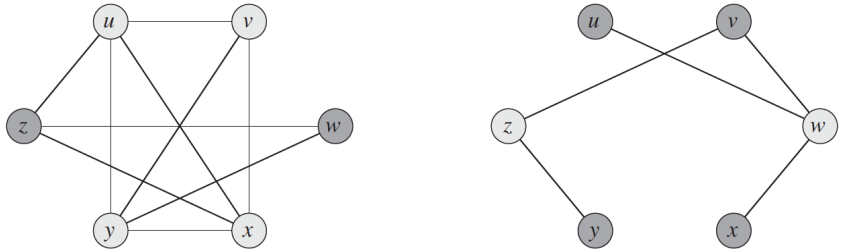
- Suppose we have to work out if a graph G has a k-clique.
- Let's take the *complement* H of G – flipping edges and non-edges:



- So a k-clique in G corresponds to an independent set of size k in H – i.e. a set of k vertices with no edges between them (here $k = 4$).
- Run VERTEX-COVER on $H, n - k$, where n is the total # of vertices.
- We claim VERTEX-COVER$(H, n - k)$ = CLIQUE(G, k).
- If VERTEX-COVER is "yes", there is a vertex cover in H with $n - k$ verts.

Vertex cover

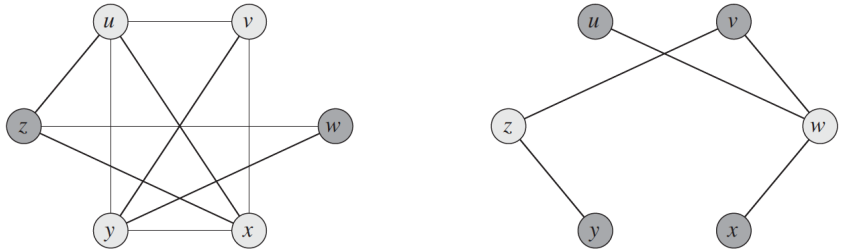
- Suppose we have to work out if a graph G has a k-clique.
- Let's take the *complement* H of G – flipping edges and non-edges:



- So a k-clique in G corresponds to an independent set of size k in H – i.e. a set of k vertices with no edges between them (here $k = 4$).
- Run VERTEX-COVER on $H, n - k$, where n is the total # of vertices.
- We claim VERTEX-COVER$(H, n - k)$ = CLIQUE(G, k).
- If VERTEX-COVER is "yes", there is a vertex cover in H with $n - k$ verts.
- Taking the k vertices that *are not in the cover* gives us an independent set in H, which is a k-clique in G.

Vertex cover

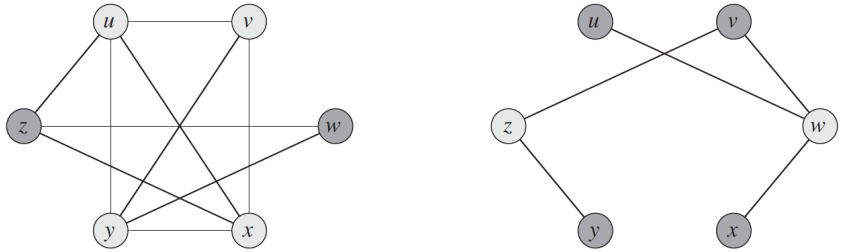
- Suppose we have to work out if a graph G has a k-clique.
- Let's take the *complement* H of G – flipping edges and non-edges:



- So a k-clique in G corresponds to an independent set of size k in H – i.e. a set of k vertices with no edges between them (here $k = 4$).
- Run VERTEX-COVER on $H, n - k$, where n is the total # of vertices.
- We claim VERTEX-COVER$(H, n - k)$ = CLIQUE(G, k).
- If VERTEX-COVER is "yes", there is a vertex cover in H with $n - k$ verts.
- Taking the k vertices that *are not in the cover* gives us an independent set in H, which is a k-clique in G.
- Conversely, if "no", there is no vertex cover in H with $n - k$ vertices.

Vertex cover

- Suppose we have to work out if a graph G has a k-clique.
- Let's take the *complement* H of G – flipping edges and non-edges:



- So a k-clique in G corresponds to an independent set of size k in H – i.e. a set of k vertices with no edges between them (here $k = 4$).
- Run VERTEX-COVER on $H, n - k$, where n is the total # of vertices.
- We claim VERTEX-COVER$(H, n - k)$ = CLIQUE(G, k).
- If VERTEX-COVER is "yes", there is a vertex cover in H with $n - k$ verts.
- Taking the k vertices that *are not in the cover* gives us an independent set in H, which is a k-clique in G.
- Conversely, if "no", there is no vertex cover in H with $n - k$ vertices.
- If there were a k-clique in G, it would be a k-independent set in H, and the other $n - k$ would be a vertex cover. So there is no k-clique in G.

Vertex cover

We define the problem VERTEX-COVER(G, k) as the problem of determining whether G has a vertex cover with k vertices. Prove that VERTEX-COVER is NP-complete.

Vertex cover

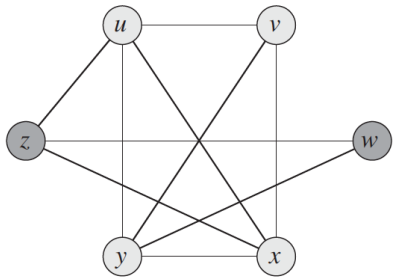
We define the problem VERTEX-COVER(G, k) as the problem of determining whether G has a vertex cover with k vertices. Prove that VERTEX-COVER is NP-complete.

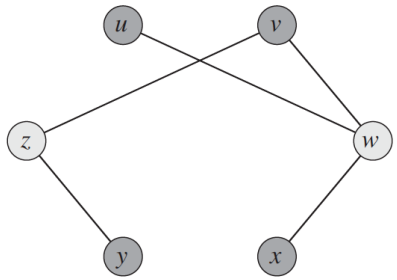
- To summarize, we first proved VERTEX-COVER is in NP.

Vertex cover

We define the problem VERTEX-COVER(G, k) as the problem of determining whether G has a vertex cover with k vertices. Prove that VERTEX-COVER is NP-complete.

- To summarize, we first proved VERTEX-COVER is in NP.
- Then we showed we could solve CLIQUE in polynomial time if we could solve VERTEX-COVER.

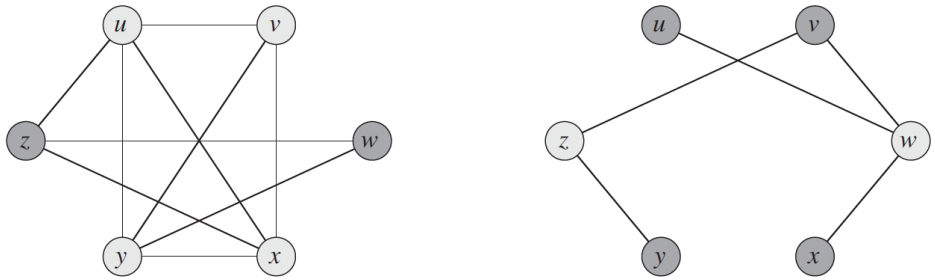




Vertex cover

We define the problem VERTEX-COVER(G, k) as the problem of determining whether G has a vertex cover with k vertices. Prove that VERTEX-COVER is NP-complete.

- To summarize, we first proved VERTEX-COVER is in NP.
- Then we showed we could solve CLIQUE in polynomial time if we could solve VERTEX-COVER.



- Since CLIQUE is NP-complete, that means VERTEX-COVER must also be NP-complete.