Algorithms and Data Structures
Discussion 15 (Week 16)

Keyi Yuan
Teaching assistant
Dec.23th 2019

P and NP

Decision Problems

- Def: assignment of inputs to No(0) or Yes(1)
- Inputs are either **No instances** or **Yes instances** (i.e. satisfying instances)

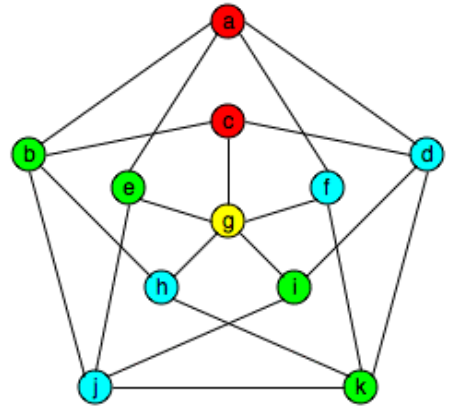
Problem	Decision
Independent Set	Given a graph, does it contains a subset of k (or more) vertices such that no two are adjacent?
3-SAT	Given a CNF formula, dost it have a satisfying truth assignment?
Hamilton Cycle	Given an undirected graph, does there exist a cycle that visits all vertices?
3-coloring	Given an undirected graph, can the nodes be colored black, white, and blue so that no adjacent nodes have the same color?

P and NP

- **P:** the set of decision problems for which there is an algorithm A such that for every instance I of size n, A on I runs in poly(n) time and solves I correctly.
- **NP:** the set of decision problems for which there is an algorithm V , a "certifier", that takes as input an instance I of the problem, and a "certificate" bit string of length polynomial in the size of I, so that:
  - V always runs in polynomial time, and the input and output should be in the polynomial size of I
  - if I is a YES-instance, then there is some certificate c so that V on input (I, c) returns **YES**, and
  - if I is a NO-instance, then no matter what c is given to V together with I, V will always output **NO** on (I, c).
- **For NP Problem, an instance can be verified in polynomial time.**
- You can think of the certificate as a proof that I is a YES-instance. If I is actually a NO-instance then no proof should work.

Example: 4-coloring is in NP

- Given a graph, can we assign each vertex one of 4 colors, such that adjacent vertices have different colors?
- Certifier
  - **Certificate y is an assignment of colors to the vertices of graph x.**
  - Check y uses at most 4 colors. If not, output no.
  - Go through all edges of x, and checks endpoints of each edge have different colors.
  - If true for all edges, output 1. Else output 0.



- If x has solution
  - Then x is 4-colorable.
  - So there's way to assign each vertex one of 4 colors s.t. endpoints of each edge have different colors.
  - Let y be this assignment, and give y to V.
  - Clearly V outputs 1.
- x has no solution
  - Then x is not 4-colorable.
  - So no matter how we assign 4 colors to vertices of x, some edge has endpoints with the same order.
  - So V outputs 0, for any input y.
- V runs in polynomial time.
  - If x has n vertices, then it has $O(n^2)$ edges, so V runs in $O(n^2)$ time.

All problems in P are in NP

- Notice that for problems in P, V doesn't need a certificate y.
  - For problems in P, it's easy to determine if they're solvable or not by itself.
  - P $\subseteq$ NP

P vs NP

- NP != Not Polynomial Time Problem
- NP = Nondeterministic Polynomial Time Problem

- P = NP? P ≠ NP? This is an open question.
- Why do we care? If can show a problem is hardest problem in NP, then problem cannot be solved in polynomial time if P ≠ NP.

- How to define hardest? How do we relate difficulty of problems? Reductions!

NP-Completeness

Reduction Review

- Suppose you want to solve problem A.
- One way to solve is to convert A into a problem B you know how to solve.
- Solve using an algorithm for B and use it to compute solution to A.
- This is called a reduction from problem A to problem B $(A \rightarrow B)$
- Because B can be used to solve A, B is **at least as hard** $(A \leq B)$
- If you can solve B, you can also solve A.

NP-completeness

- Out of all the NP problems, there's a subset of NP problems called NP-complete (NPC) problems that are the "hardest" NP problems.

- To determine whether P=NP, it suffices to know whether P=NPC.
  - If the hardest problems can be solved in polytime, then all NP problems can be solved in polytime. i.e. P=NP.

- So the study of P vs NP focuses on NPC problems.

NP-completeness

- Def A problem A is NP-complete (NPC) if the following are true.
  - $A \in NP$.
  - Given any other problem $B \in NP$, $B \leq_P A$.

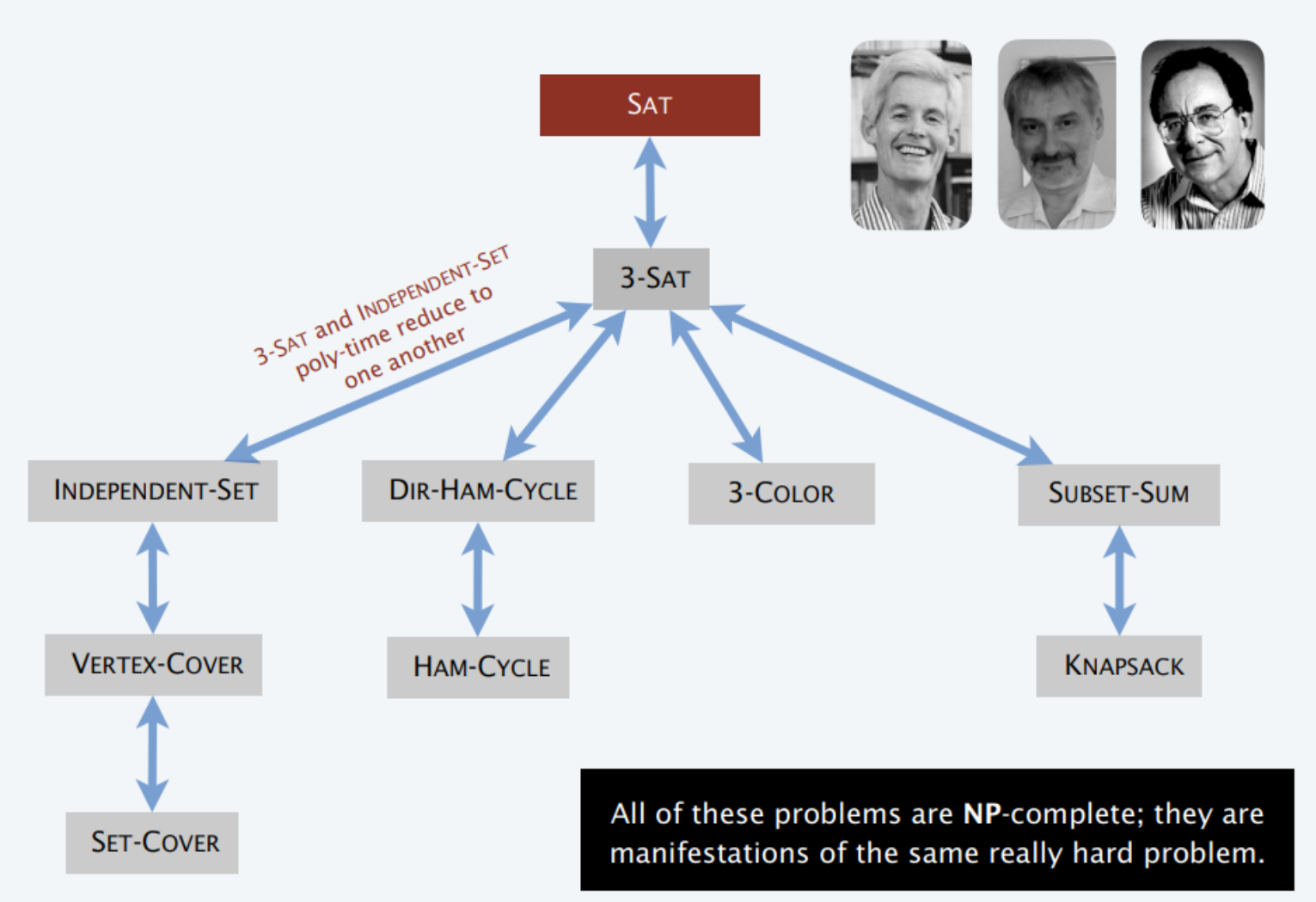
- Thus, a NP-complete problem is an NP problem that can be used to solve any other NP problem.
  - It's a "hardest" NP problem.

NP-completeness and SAT

- Do NP-complete problems really exist?
- Yes! Steve Cook and Leonid Levin proved around 1970 that SAT is NP-complete.
- SAT = satisfiable Boolean formulas.
  - Not, And, Or Gates. Proof is not required.
  - Ex $(A \vee B \vee \neg C) \wedge (A \vee \neg B \vee C) \wedge (\neg A \vee \neg B \vee \neg D) \in SAT$.
    - Setting A=B=C=true, D=false makes the formula true.

The web of NP-completeness

- You can use it in exam directly.



How to prove NPC?

- Thm 1 Given two NP problems A and B, suppose A is NP-complete, and $A \leq_P B$. Then B is also NP-complete.
- Proof Let C be any NP problem. Then $C \leq_P A$, since A is NP-complete.
  - Since $A \leq_P B$, then by Theorem 1, we have $C \leq_P A \leq_P B$.
  - Since also $B \in NP$, then B is NPC.

- To prove a problem B is NP-complete
  - First, you have to prove $B \in NP$, but that's usually not hard.
  - Second, take a problem A you know is NPC, and prove $A \leq_P B$.
    - To prove $A \leq_P B$, you need to give a polytime reduction from A to B.
    - $\Rightarrow$: To say B is harder than A, which has been known as a NPC problem. If A has a Yes-instance, then B also has a Yes-instance.
    - $\Leftarrow$: To say if B has a Yes-instance, then A also has a Yes-instance.

NP-completeness and P vs NP

- Thm 2 Suppose a problem A is NP-complete, and $A \in P$. Then P=NP.
- Proof Consider any other NP problem B. We'll show $B \in P$.
  - Since A is NPC, there's a polytime mapping f from B to A.
  - Given an instance X of B, run f on X to get an instance Y of A.
  - Since $A \in P$, there's a polytime algorithm g to solve A.
  - Run $g(Y)$, and return the same answer for X.
  - By the definition of $\leq_P$, $g(Y)$ is true $\Leftrightarrow$ X is true.
  - Running f and g both take polytime. So we can solve B in polytime.

NP-completeness and P vs NP

- Cor Suppose a problem A is NP-complete, and $A \notin P$. Then for any NP-complete problem B, $B \notin P$.
  - If $B \in P$, then since B is NPC, we have P = NP by Theorem 2. So since $A \in NP$, we have $A \in P$, a contradiction.

- To prove $P \neq NP$ (which is what most people think), it's enough to show one NPC problem is not solvable in polytime, by the corollary.
  - Nor has anyone shown a polytime algorithm for any NPC problem.

Exercise 1

- The problem of determining whether an undirected graph contains as a subgraph a complete graph on k vertices is in NP.

- True. Providing a subset of k vertices is a $O(k)$ polynomial size certificate which can be checked for completeness in $O(k^2)$ polynomial time.

Exercise 2

- You have a magic machine that can solve instances of an NP-Complete problem in constant time. If you can design an algorithm that uses the machine to solve another decision problem A in polynomial time, then this implies that problem A is also NP-Complete.

- False, the reduction is the wrong direction; reducing A to an NP-Complete problem only proves that the A is no harder than the NP-Complete problem, i.e. A is in NP.

Exercise 3

- Let X be a problem that belongs to the class NP. Then which one of the following is FALSE?
- A. There is no polynomial time algorithm for X.
- B. If X can be solved deterministically in polynomial time, then P = NP.
- C. If there is another problem Y, Y is NP-complete, and X reduces to Y in polynomial time, then X is NP-complete.

- (A) is False because set NP includes both P(Polynomial time solvable) and NP-Complete.
- (B) is False because X may belong to P (same reason as (A))
- (C) is False. You can only know X is in NP.

Exercise 4

- 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.

- 1. We can either use BFS or DFS to find whether there is a cycle in an undirected graph. For example, see DFS based implementation to detect cycle in an undirected graph. The time complexity is O(V+E) which is polynomial.
- 2. If a problem is in P, then it is definitely in NP (can be verified in polynomial time). See NP-Completeness
- 3. True. See See NP-Completeness

NPC Prove Example 1:
3-coloring and 4-coloring

4-coloring is NP-complete: First Part

- Consider the 4-coloring problem. We have a graph $G = (V, E)$ and 4 different colors as set C. The question is whether we can find an assignment $a: V \to C$ such that for each edge $e = (i, j)$, $a(i) \neq a(j)$. Show that 4-coloring is an NP-complete problem.

- First, we prove 4-coloring$\in$NP.
  - The certificate is a given color assignment. (In polynomial size of $|V|$)
  - The certifier just checks for each edge in $e = (i, j) \in E$ if the value of $a(i)$ is equal to $a(j)$.
  - This can be done in polynomial time $O(|V|^2)$.

4-coloring is NP-complete: Second Part

- Second, show some NP-complete problem reduces to 4-coloring.
  - The problem you reduce from has to be NP-complete, not just in NP.
  - Note, you're reducing from the NPC problem to your problem.
  - You can choose any NP-complete problem to reduce from. Decide on the right problem can make the task a lot easier.

4-coloring is NP-complete: Second Part

- Assume we've already proven 3-coloring is NP-complete.
- We show 3-coloring $\leq_P$ 4-coloring.
- The reduction says that given a 3-coloring Graph $G = (V, E)$, we can create in polytime a graph $G' = (V', E')$, V' is V with an additional vertex $\tilde{v}$ and E' is E with edges $(v, \tilde{v})$ for every $v \in V$.
- This means that G is satisfiable if and only if G' has an 4-color assignment.

Proving the reduction works

- We first need to show the reduction runs in polytime.
  - Yes. If there are n vertices, the reduction takes $O(n^2)$ time.
- Show $G \in$ 3-coloring $\Leftrightarrow$ $G' \in$ 4-coloring.
  - ($\Rightarrow$) If G has a satisfying assignment, then G' has an 4-color assignment.
  - ($\Leftarrow$) If G' has an 4-color assignment, then G has a satisfying assignment.

$\exists$ 3-color assignment $\Rightarrow \exists$ 4-color assignment

- If G is a Yes-instance of 3-coloring problem with assignment a, we can just use the assignment for vertices in $V' \cap V$ of G' and color the last vertex $\tilde{v}$ with the fourth color which is unused.
- Then the assignment a' is valid for G' which means the constructed 4-coloring is also a Yes-instance.

$\exists$ 4-color assignment $\Rightarrow \exists$ 3-color assignment

- If G is a Yes-instance of constructed 4-coloring problem with assignment a', then since the vertex $\tilde{v}$ directly connected to every other vertices, the set $V' \cap V$ can only use three different colors.
- Then $G = G' - \{\tilde{v}\}$ is also a Yes-instance.

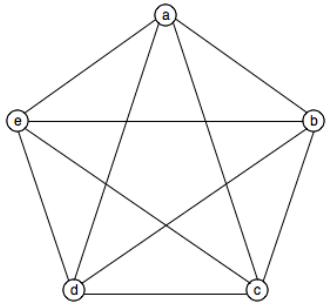
4-coloring is NP-complete

- We've shown 4-coloring $\in$ NP.
- We've shown 3-coloring $\leq_P$ 4-coloring.
  - We found a polytime reduction, constructing a graph G'
    - ($\Rightarrow$) If G has a satisfying assignment, then G' has an 4-color assignment.
    - ($\Leftarrow$) If G' has an 4-color assignment, then G has a satisfying assignment.
- So 4-coloring is NP-complete.

NPC Prove Example 2: CLIQUE and 3-SAT

CLIQUE is NP-complete: First Part

- Given a graph with n nodes, is there a clique with $\frac{n}{3}$ nodes?
  - Actually, the CLIQUE problem asks if there's a k-clique, for an arbitrary k. But we consider $k = \frac{n}{3}$ for simplicity.



- First, we prove CLIQUE$\in$NP.
  - The certificate is a purported $\frac{n}{3}$-clique.
  - The certifier just checks there are $\frac{n}{3}$ nodes, and they're are all connected.

CLIQUE is NP-complete: Second Part

- Second, show some NP-complete problem reduces to CLIQUE.
  - The problem you reduce from has to be NP-complete, not just in NP.
  - Note, you're reducing from the NPC problem to your problem.
  - You can choose any NP-complete problem to reduce from. Decide on the right problem can make the task a lot easier.

CLIQUE is NP-complete: Second Part

- Assume we've already proven 3-CNF-SAT is NP-complete.
- We show 3-CNF-SAT $\leq_P$ CLIQUE.
- The reduction says that given a 3-CNF-SAT formula ϕ, we can create in polytime a graph G, such that ϕ is satisfiable if and only if G has an $\frac{n}{3}$-clique.
  - This is actually quite remarkable. We will see how to construct a special graph that captures the satisfiability of a 3-CNF formula.

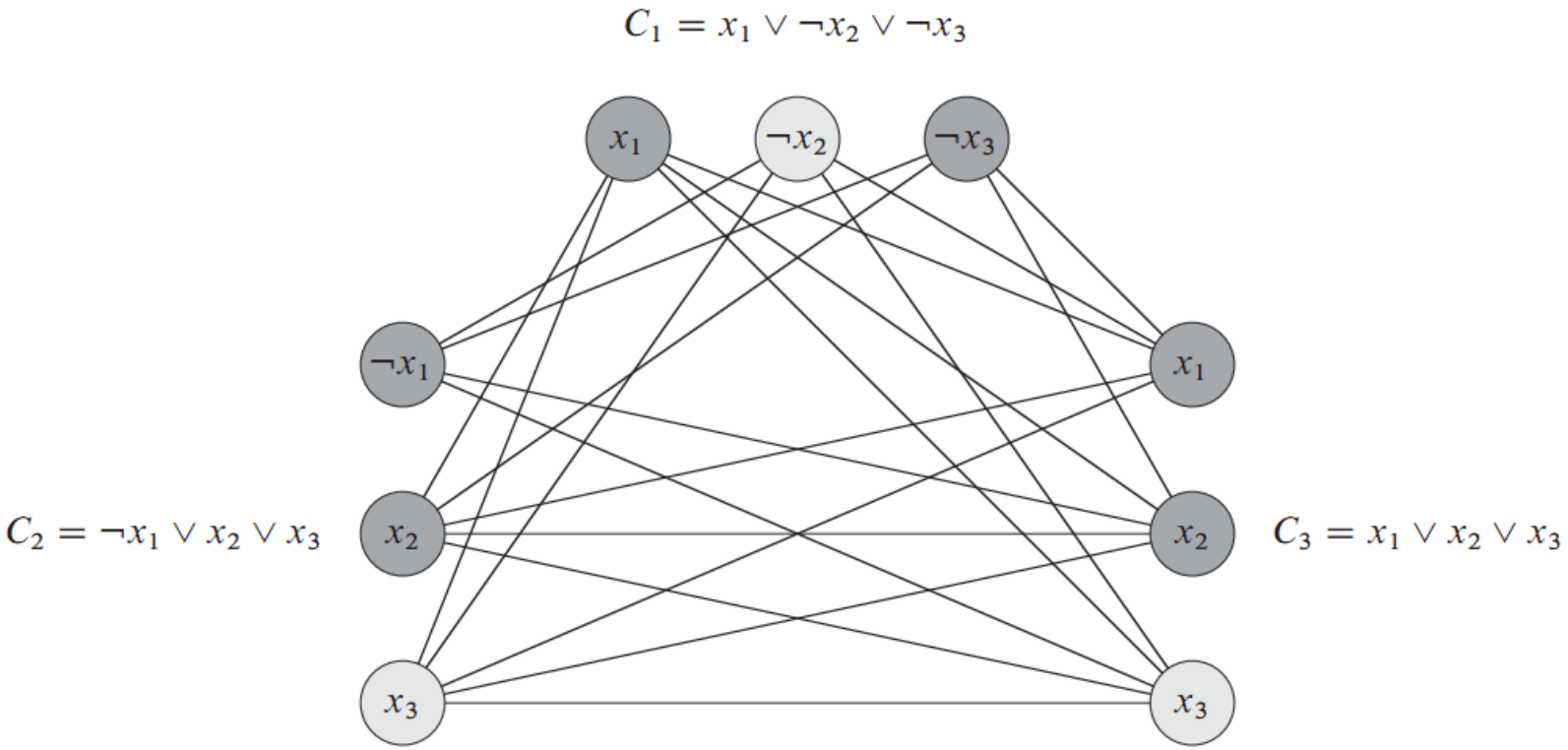
Reducing 3-CNF-SAT to CLIQUE

- Let ϕ be a 3-CNF formula with m clauses.
- Let C be a clause in ϕ. Then C has 3 literals.
  - Make 3 vertices in G corresponding to the literals.
    - So G has $3m$ vertices total.
    - Let n be the number of nodes in G. Then $m = \frac{n}{3}$.
- Now, add in an edge between two vertices u, v if both conditions below hold.
  - u, v correspond to literals from different clauses of ϕ.
  - The literals corresponding to u and v are not negations of each other.
    - We say u and v are consistent.

Reducing 3-CNF-SAT to CLIQUE

- 3 vertices for each clause.
- For vertices u,v, add edge (u,v) if u,v are from different clauses, and are consistent (not negations of each other).

$$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$



Proving the reduction works

- We first need to show the reduction runs in polytime.
  - Yes. If there are n clauses, the reduction takes $O(n^2)$ time.
- Recall the graph has $n = 3m$ nodes, so $m = \frac{n}{3}$.
- Show $\phi \in$ 3-CNF-SAT $\Leftrightarrow G \in$ m-CLIQUE.
  - ($\Rightarrow$) If ϕ has a satisfying assignment, then G has an m clique.
  - ($\Leftarrow$) If G has an m clique, then ϕ is satisfiable.

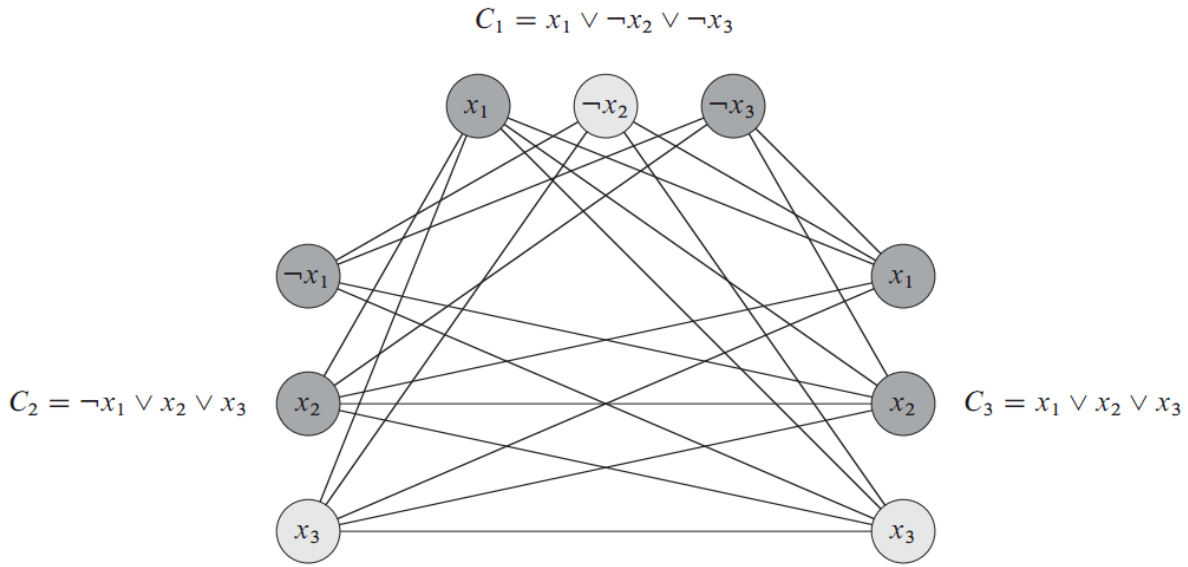
$\exists$ sat. assignment $\Rightarrow \exists$ m clique

- In the satisfying assignment, every clause has to be true, since we AND them.
- In each clause, at least one literal has to be true, since we OR them.
- So for each clause, pick a true literal.
  - We pick $m = \frac{n}{3}$ literals.
- The true literal corresponds to a vertex in the graph.
  - Pick m vertices corresponding to the m literals we picked.
- Claim The selected vertices form an m-clique.
- Proof Consider any 2 vertices u, v we selected.
  - u, v come from different triples. Because they come from literals from different clauses.

$\exists$ sat. assignment $\Rightarrow \exists$ m clique

- Proof ctd u, v are consistent. I.e. they don't correspond to a literal in one clause, and its negation in another clause.
  - Because we only picked true literals.
  - So there's an edge (u, v), by construction.
  - So any 2 of the m selected vertices are connected. So the vertices are an m-clique.
- Ex ϕ has a satisfying assignment $x_1 = x_2 = x_3 = T$.
  - The corresponding nodes form a 3-clique.

$$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$



$\exists$ m clique $\Rightarrow$ $\exists$ sat. assignment

- Consider the m vertices in the clique.
- None of the vertices come from literals in the same clause in formula.
  - For any pair of vertices, they're connected.
  - There are no edges between vertices from the same clause.
- None of the vertices correspond to a literal and its negation in formula.
  - We don't add edges between such vertices.
- For the literals corresponding to the clique vertices, set all of them to be true in the formula.
  - This is a valid assignment, since we never set a literal and its negation both to true.
  - We have one true literal per clause.
  - So every clause is true.
  - So the formula is true.

CLIQUE is NP-complete

- We've shown CLIQUE$\in$NP.
- We've shown 3-CNF-SAT $\leq_P$ CLIQUE.
  - We found a polytime reduction, constructing a graph G s.t. for every 3-CNF-SAT formula ϕ
    - If ϕ is satisfiable, G has an $\frac{n}{3}$-clique.
    - If G has an $\frac{n}{3}$-clique, then ϕ is satisfiable.
- So CLIQUE is NP-complete.