NP Completeness

COMP 455 – 002, Spring 2019

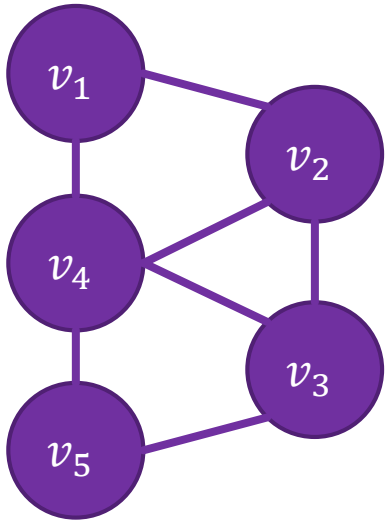
Time Complexity (Revisited)

- A **problem** is a yes/no question.
- Example: The **clique problem**: Given a graph G and integer k, does G contain a k-clique?
  - (A ***k*-clique** is a fully-connected group of k nodes.)
- (G, k) is an *instance* of this problem (for some specific graph G).
- A problem is *decidable* if an algorithm exists that will take any instance of this problem and always produce the correct yes/no answer.
- An *algorithm* is a TM that always halts.

We will only be concerned with decidable problems when discussing NP completeness.

Another Example Problem

Consider this undirected graph:



Let $k = 3$. We can encode an instance of the clique problem as a string w:

- $w = 3(1,2)(1,4)(2,3)(2,4)(3,4)(3,5)(4,5)$

The language corresponding to the clique problem could be stated as follows:

- $L = \{$all strings $k(i_1, i_2) \ldots (i_m, j_m)$ | the corresponding graph has a k-clique$\}$.
- If M is a TM that solves the clique problem, then $L(M) = L$.

Additional Constraints on Encoding

Many different ways exist to encode problems. For these problems, the *choice of encoding matters*!

- The **standard encoding** for these problems is binary or something "polynomially related" to binary.
  - For example, decimal is OK. Unary is *not OK*.

There are two reasons for this:

- With unary encodings, polynomial-time solutions exist to some NP-complete problems. This is because time complexity is measured based on input *length*! Using artificially long inputs is "cheating".
- Real computers use binary encodings.

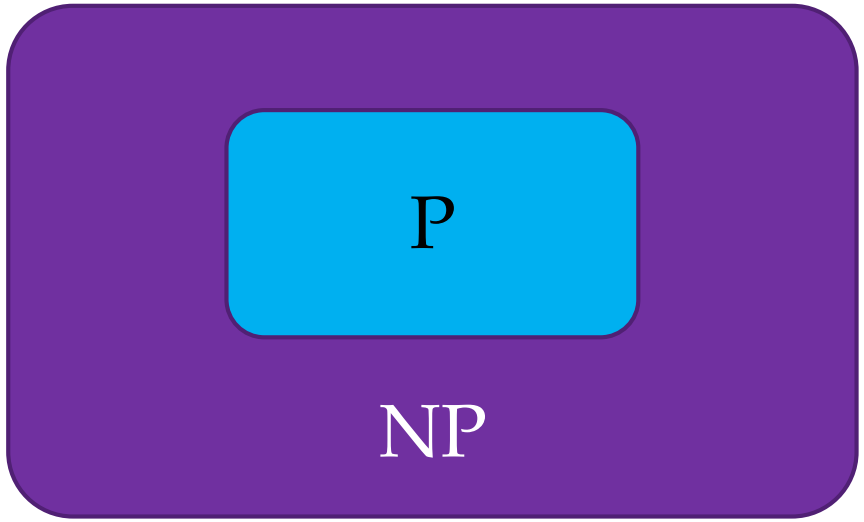
Two Basic Complexity Classes

(There are lots of other classes—we're just focusing on these two for now.)

- **P-Time**: Any language *L* accepted in polynomial time by some *deterministic* TM.
- **NP-Time**: Any language *L* accepted in polynomial time by some *nondeterministic* TM.
  - Recall that the time complexity of a NDTM is based on the smallest number of moves needed to accept a string of a given length.

We usually abbreviate P-Time as "P" and NP-Time as "NP."

The Relationship Between P and NP



- Every DTM is also a NDTM, so P is in NP.
- It's not known whether $P = NP$ or $P \subset NP$.

P vs. NP

The P = NP question is the *most* important open question in theoretical computer science. Here's why:

- Lots of interesting problems are in NP.
- Problems in P have "efficient" solutions. (Meaning deterministic, polynomial-time solutions.)
- For many important problems in NP (e.g. yes/no counterparts to optimization problems or graph problems) we haven't found "efficient" solutions despite decades of research.
  - The suggests that $P \neq NP$.
  - *But*, if $P = NP$, then these problems *do* have "efficient" solutions.

Hard and Complete Problems

We want to identify the "hardest" problems in NP.

- If the "hard" problems have deterministic polynomial-time solutions, then so do all problems in NP.
- This reduces the P=NP question to that of whether one of these "hard" problems is in P.

Hard and Complete Problems

In other words, we want a DTM that converts instances of one problem to instances of another in polynomial time.

What do we specifically mean by "hard" and "complete"? We need some more definitions:

- L is **polynomially transformable** to L' if there exists a polynomial-time-bounded DTM M that can convert each string w in the alphabet of L into a string w' in the alphabet of L' such that $w \in L$ if and only if $w' \in L'$.
- Let C be a class of languages. L is **hard for C** if every language in C is polynomially transformable to L.
- L is **complete for C** if L is in C and hard for C.

NP-Complete Problems

Using the previous definitions, *L* is **NP-complete** if it is in NP and NP-hard.

- Note: Showing that *L* is in NP is usually straightforward: show that a NDTM can accept *L* by "guessing" an answer and then verifying the guess is correct in polynomial time.

Our goal is to first identify some NP-complete problems.

NP-Complete Problems

- Given the unlikelihood of P=NP, if a language *L* is NP-complete, then *L* is *probably* an intractable problem.
  - Note: "NP-complete" is *not* (currently) synonymous with "exponential time."
- We can show P=NP if we're able to find a single NP-Complete problem that's in P. Nobody's been able to do so, but it hasn't been proven impossible yet, either.

"Guess-and-Verify" Example

Let a NDTM accept strings of the form $10^{i_1}10^{i_2} \ldots 10^{i_k}$ such that there is some $I \subseteq \{1, \ldots, k\}$ for which $\sum_{j \in I} i_j = \sum_{j \notin I} i_j$.

(This is called the **partition problem**.)

Here's how our NDTM can work:

- The TM has three tapes, the input is on tape 1.
- The TM scans the input and copies each group of 0s to either tape 2 or tape 3, the choice being nondeterministic.
- The TM accepts if an equal number of 0s were copied to both tapes after scanning the entire input string.

This technically isn't a "legal" encoding (it encodes i_n using a unary # of 0s). We're just using it here to simplify the example.

Proving a Problem is NP-Complete

There are two ways to show *L* is NP complete:

- Prove it from scratch (very difficult!)
- Use *reduction*:
  I. Show that *L* is in NP (*i.e.* design a "guess-and-verify" polynomial-time NDTM for *L*), and
  II. Show that a known NP-complete problem can be polynomially transformed to *L*.

So, if we want to use reduction, we first need a known NP-complete problem.

Cook's Theorem

The **satisfiability problem** is the following: given a Boolean expression E, is it possible to assign values to the variables in such a way that makes E true?

Example:

- $(x \vee y) \wedge z$ is satisfiable.
- $(x \vee y) \wedge \bar{x} \wedge \bar{y}$ is not satisfiable.

Cook's Theorem: The satisfiability problem is NP-complete.

The proof is too long to cover in this class, so we will need to take it on faith.

Variations on Satisfiability

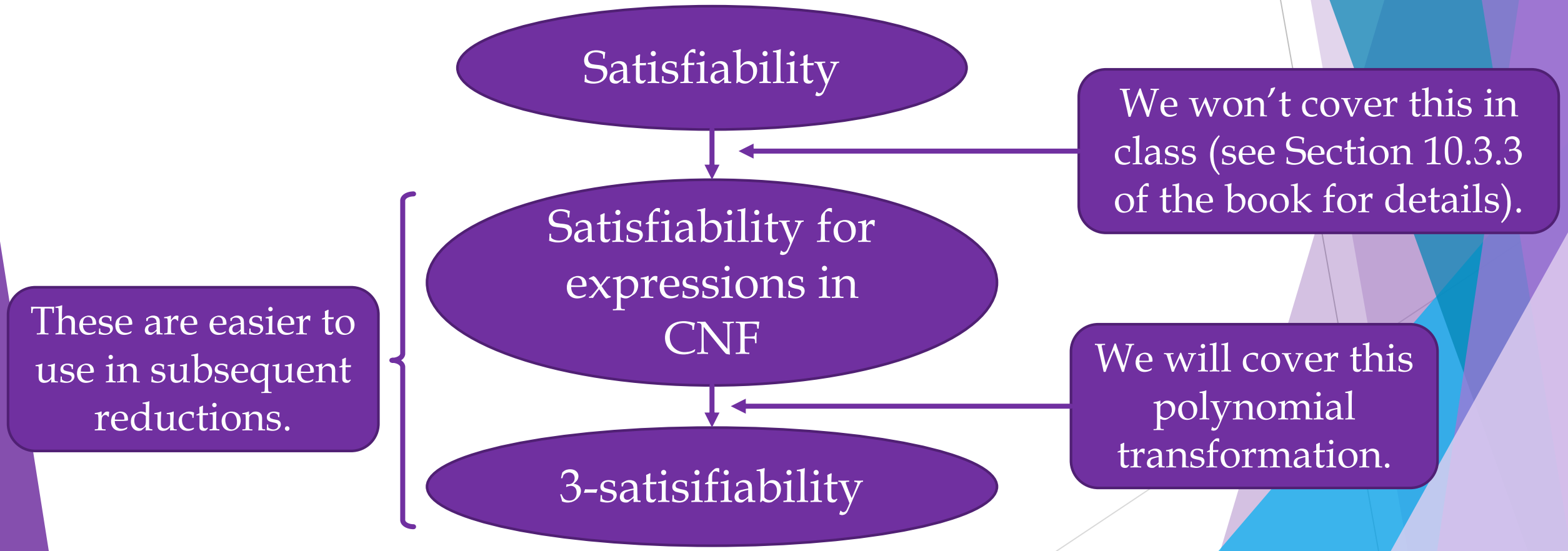
- A **literal** is either x or $\neg x$, where x is a Boolean variable.
- A Boolean expression is in **conjunctive normal form** (CNF) if it is a conjunct of "clauses" of literals:
  - Example: $(a \vee b) \wedge (c) \wedge (\neg a \vee b \vee \neg c)$
- **k-CNF**: A conjunct of clauses containing k literals.
- **k-satisfiability**: Satisfiability for expressions in k-CNF.

A *conjunct* is a group of literals or clauses that are "and"-ed together.

A *clause* is a group of literals that are "or"-d together.

Variations on Satisfiability

We will show the following:



3-Sat

- The satisfiability problem for CNF expressions with exactly three literals per clause is called *3-satisifiability* ("3-Sat").
  - Example: Is $(a \vee b \vee c) \wedge (\neg c \vee d \vee e)$ satisfiable?

3-Sat

Theorem 10.15: 3-satisfiability is NP-complete.

Proof:

We must do two things:

- Show 3-Sat is in NP.
  - This is easy: just nondeterministically generate an assignment for each variable, then, in polynomial time verify that each clause evaluates to true.
- Show 3-Sat is NP-hard.
  - To do this we will show that if we can solve 3-Sat efficiently, then we can solve CNF-Sat efficiently.

This is called a **reduction** from CNF-Sat.

Reducing from CNF-Sat to 3-Sat

- Our goal is to take a Boolean expression in CNF and, *in polynomial time*, generate a new expression in 3-CNF that is satisfiable if and only if the original CNF expression is satisfiable.
- We want to show that if a polynomial time solution to 3-CNF exists, then the following algorithm would be possible:
  1. Convert a CNF-Sat problem to a 3-Sat problem in polynomial time.
  2. Use the polynomial-time 3-Sat algorithm to determine if the converted expression is satisfiable.

Reducing from CNF-Sat to 3-Sat

Converting from CNF-Sat to 3-Sat:

- We can convert each clause containing n literals $x_1 \ldots x_n$ to a conjunct of $n - 2$ new clauses, each of which contain exactly three literals.
  - This requires adding $n - 3$ new literals $y_1 \ldots y_{n-3}$ per clause.

Example:

$(x_1 \vee x_2 \vee \cdots \vee x_n)$ becomes:

$(x_1 \vee x_2 \vee y_1) \wedge (\overline{y_1} \vee x_3 \vee y_2) \wedge (\overline{y_2} \vee x_4 \vee y_3) \wedge \cdots$
$\wedge (\overline{y_{n-4}} \vee x_{n-2} \vee y_{n-3}) \wedge (\overline{y_{n-3}} \vee x_{n-1} \vee x_n)$

Reducing from CNF-Sat to 3-Sat

- We want to show that the old expression is satisfiable if and only if the new expression is satisfiable.
- First: The old expression is satisfiable ⇒ the new expression is satisfiable.
  - Assume the old expression is satisfiable
  - ⇒ ∃ an assignment s.t. the old expression is true
  - ⇒ some x_i is true in the assignment
  - ⇒ Setting $y_j := \text{true}$ for $j \leq i-2$ and $y_j := \text{false}$ for $j < i-2$ makes the new expression true.
  - ⇒ The new expression is satisfiable.

Old: $(x_1 \vee x_2 \vee \cdots \vee x_n)$

New:

$(x_1 \vee x_2 \vee y_1) \wedge$
$(\overline{y_1} \vee x_3 \vee y_2) \wedge$
$(\overline{y_2} \vee x_4 \vee y_3) \wedge \cdots \wedge$
$(\overline{y_{n-3}} \vee x_{n-1} \vee x_n)$

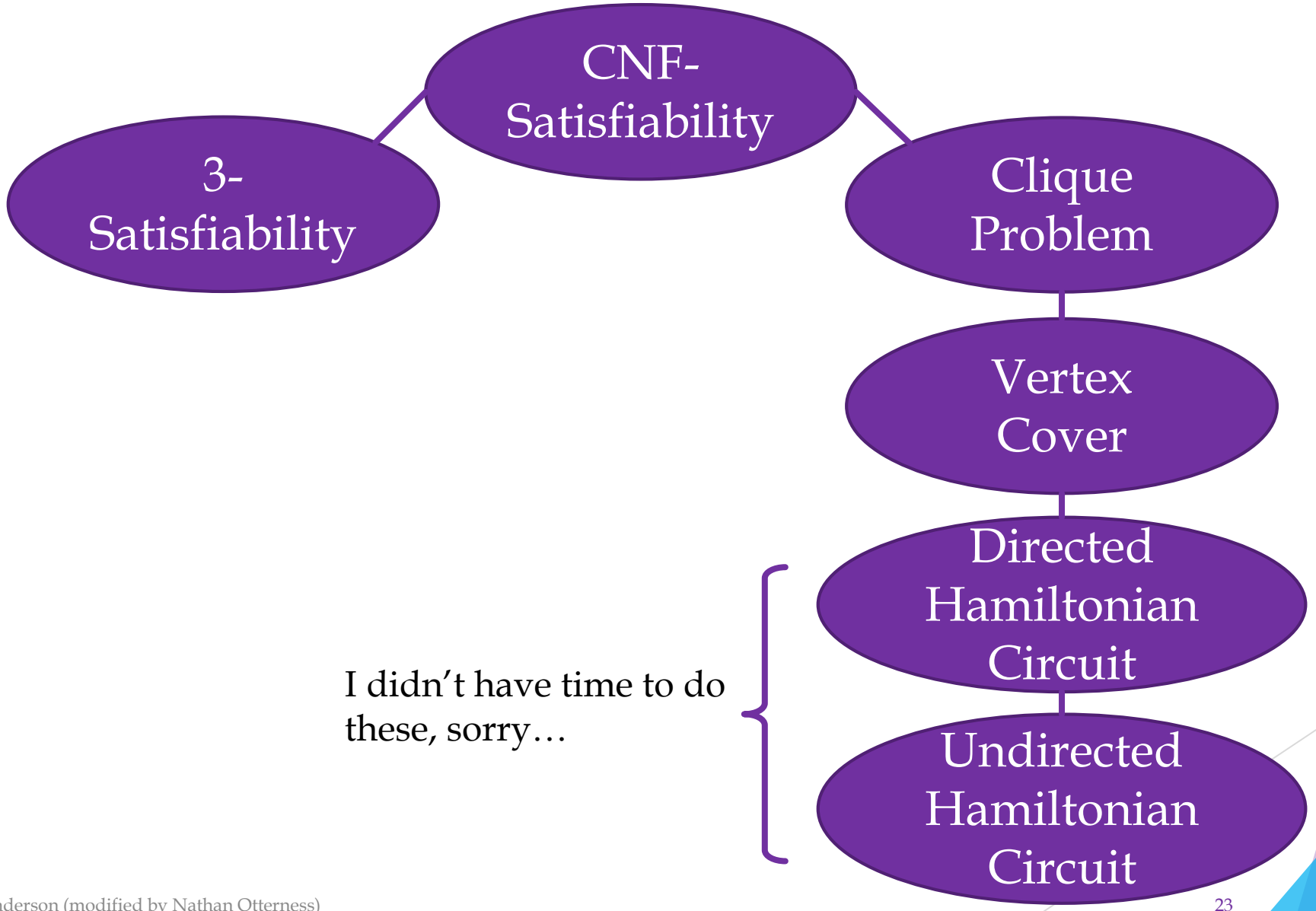
Reducing from CNF-Sat to 3-Sat

Old: $(x_1 \vee x_2 \vee \cdots \vee x_n)$

New:
$(x_1 \vee x_2 \vee y_1) \wedge$
$(\overline{y_1} \vee x_3 \vee y_2) \wedge$
$(\overline{y_2} \vee x_4 \vee y_3) \wedge \cdots \wedge$
$(\overline{y_{n-3}} \vee x_{n-1} \vee x_n)$

- Second: The new expression is satisfiable ⇒ the old expression is satisfiable.
- Assume an assignment exists such that the new expression is true (i.e. it's satisfiable). One of the following must be the case:
  - If y_1 is false, then x_1 or x_2 is true.
  - If y_{n-3} is true, then x_{n-1} or x_n is true.
  - If y_1 is true and y_{n-3} is false, then some i exists such that y_i is true and y_{i+1} is false. This implies that x_{i+2} is true.
- For all of the above cases, the old expression also had to be true because some x_i was true.

Problems We Will Cover



The Clique Problem

Theorem: The clique problem is NP-complete.

Proof:

- The clique problem is in NP.
- To show the clique problem is NP-hard we will show that CNF-Sat reduces to the clique problem.
- Consider a CNF expression $F = F_1 \wedge F_2 \wedge \cdots \wedge F_q$, where $F_i = (x_{i1} \vee x_{i2} \vee \cdots \vee x_{ik_i})$.
- We will construct a graph G that has a q-clique if and only if F is satisfiable.

To show it's in NP:
1. Nondeterministically "guess" a selection of k nodes in the graph.
2. In polynomial time, check whether the k nodes are fully connected.

The Clique Problem

- $F = F_1 \wedge F_2 \wedge \cdots \wedge F_q$, where $F_i = (x_{i1} \vee x_{i2} \vee \cdots \vee x_{ik_i})$.
- Construct the graph $G = (V, E)$ as follows:
  - Each pair $[i, j]$ is a vertex, where $1 \leq i \leq q$ and $1 \leq j \leq k_i$.
  - Each $([i, j], [k, l])$ is an edge, where $i \neq k$ and $\overline{x_{ij}} \neq x_{kl}$.
  - Given F, you can construct G in polynomial time.

The Clique Problem

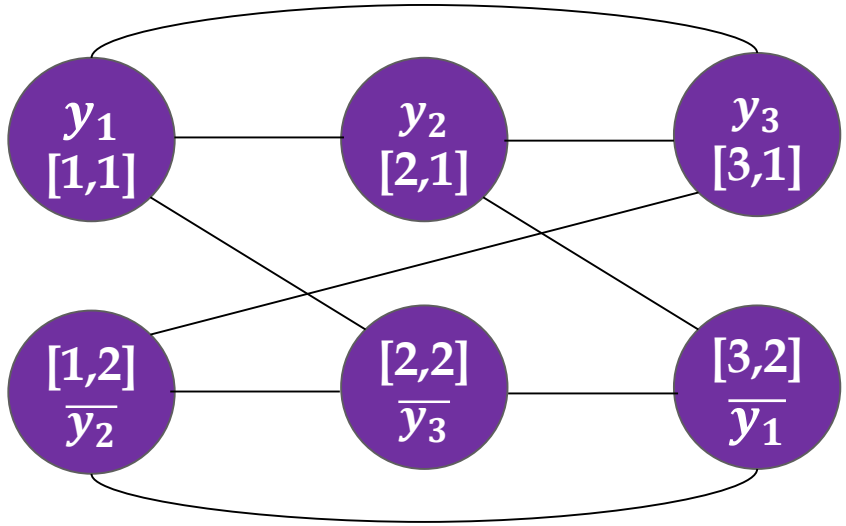
Example: $F = (y_1 \vee \overline{y_2}) \wedge (y_2 \vee \overline{y_3}) \wedge (y_3 \vee \overline{y_1})$

The literals are:

$x_{11} = y_1$ $\quad$ $x_{21} = y_2$ $\quad$ $x_{31} = y_3$

$x_{12} = \overline{y_2}$ $\quad$ $x_{22} = \overline{y_3}$ $\quad$ $x_{32} = \overline{y_1}$

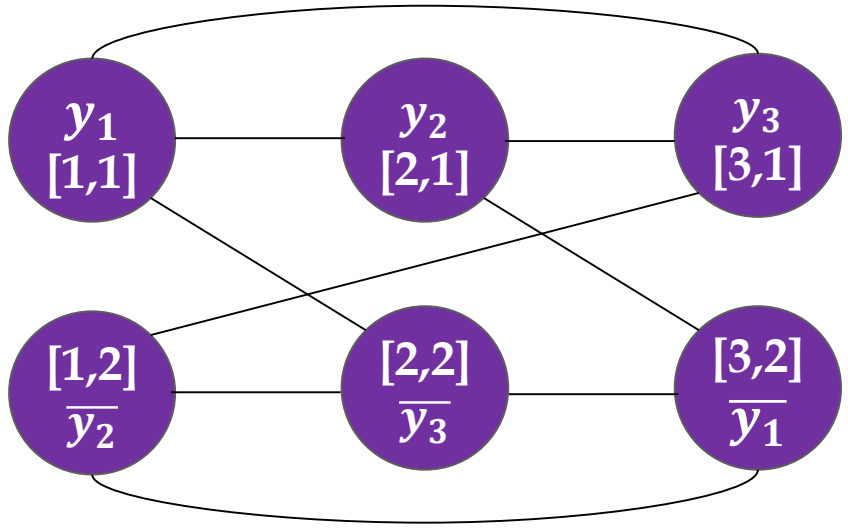
G:



The Clique Problem

Example: $F = (y_1 \vee \overline{y_2}) \wedge (y_2 \vee \overline{y_3}) \wedge (y_3 \vee \overline{y_1})$

G:



G has a clique of size q if and only if F is satisfiable.

G has two 3-cliques:

- {[1,1], [2,1], [3,1]} corresponding to $y_1 = y_2 = y_3 =$ true.
- {[1,2], [2,2], [3,2]} corresponding to $y_1 = y_2 = y_3 =$ false.

The Clique Problem

Proof that G has a q-clique if and only if F is satisfiable:

- Assume F is satisfiable.
  - Then, ∃ an assignment to variables s.t. F is true.
  - Each F_i has ≥ 1 literal that is true.
  - Let x_{im_i} in F_i be true.
  - Consider $\{[i, m_i] \mid 1 \leq i \leq q\}$.
  - Consider $[i, m_i]$ and $[j, m_j]$ where $i \neq j$.
  - $x_{im_i} = x_{jm_j} =$ true implies $x_{im_i} \neq \overline{x_{jm_j}}$.
  - Since $i \neq j$, $([i, m_i], [j, m_j])$ is an edge.
  - So, $\{[i, m_i] \mid 1 \leq i \leq q\}$ is a q-clique.

The Clique Problem

- Assume G has a q-clique.
  - Let the vertices in the clique be $\{[i, m_i] \mid 1 \leq i \leq q\}$
  - Let $S_{true} = \{y \mid x_{im_i} = y \text{ for some } i\}$ and $S_{false} = \{y \mid x_{im_i} = \bar{y} \text{ for some } i\}$.
  - By setting each variable in S_{true} to true and each variable in S_{false} to false, each clause F_i is made true. So, F is satisfiable.

Vertex Cover

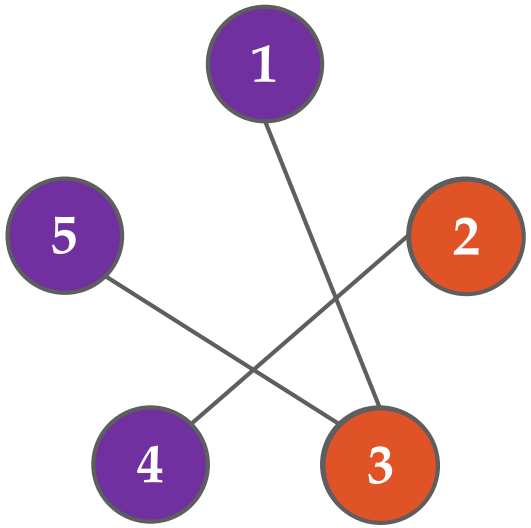
Theorem 10.20: The Vertex Cover problem is NP-Complete.

Vertex Cover: Given a graph $G = (V, E)$, find a subset S of the vertices V such that each edge in E is incident upon some vertex in S. (This is also called node cover.)

The **Vertex Cover Problem**: *Does an undirected graph G have a vertex cover of size k?*

Vertex Cover

Vertex Cover example: Does this graph G contain a vertex cover of size 2?



Yes: $\{2, 3\}$ is a vertex cover.

Vertex Cover

To prove that Vertex Cover is in NP:

1. Nondeterministically "Guess" a set of k vertices in G.
2. In polynomial time, check to see if this set is a vertex cover.

To prove that Vertex Cover is NP-hard, we will reduce from the clique problem

- We show that a polynomial-time solution to vertex cover will let us solve the clique problem in polynomial time.

Vertex Cover

Here's the transformation that we'll make:

1. Start with an instance of the clique problem: (G, k)
   - Let $G = (V, E)$
2. In polynomial time, compute $\bar{G}$
   - Let $\bar{G} = (V, \bar{E})$, where $\bar{E} = \{(v, w) \mid v \neq w \wedge (v, w) \notin E\}$
3. Construct the following instance of the vertex cover problem: $(\bar{G}, |V| - k)$.

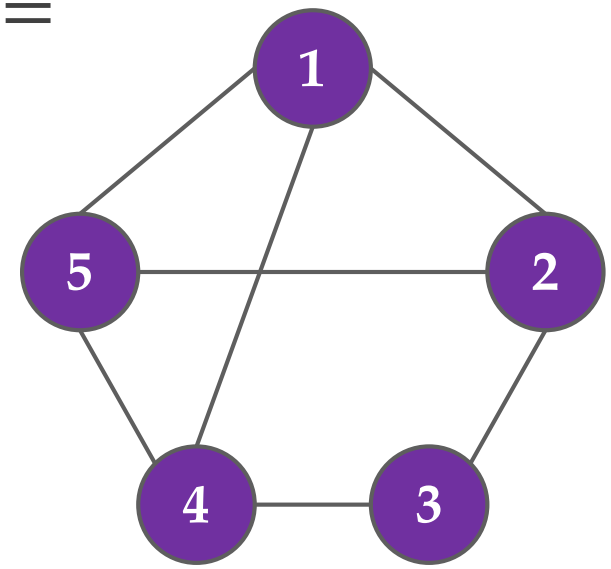
$\bar{G}$ has a vertex cover of size $|V| - k$ if and only if G had a clique of size k. (We'll prove this on the following slides.)

Vertex Cover

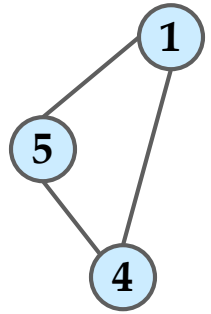
Transformation example:

- Input to the clique problem: Does G have a 3-clique?

$G =$



(Yes, it does)



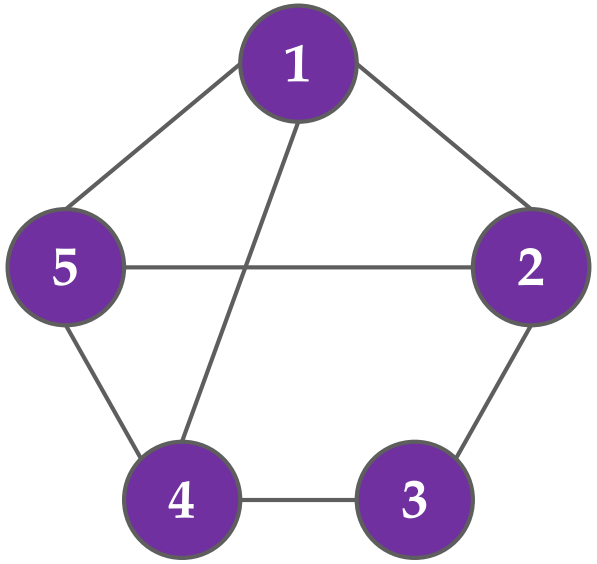
1. Start with an instance of the clique problem: (G, k)
   - $G = (V, E)$
2. In polynomial time, compute $\bar{G}$
   - $\bar{G} = (V, \bar{E})$, where $\bar{E} = \{(v, w) \mid v \neq w \wedge (v, w) \notin E\}$
3. Construct the following instance of the vertex cover problem: $(\bar{G}, |V| - k)$.

Vertex Cover

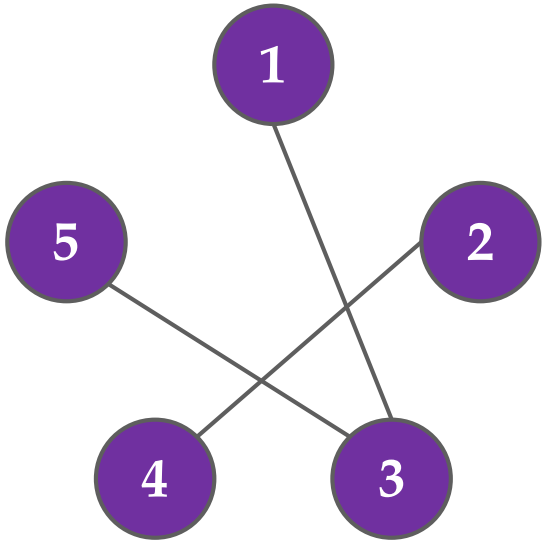
Transformation example:

- Next, construct $\bar{G}$:

G:



$\bar{G}$:



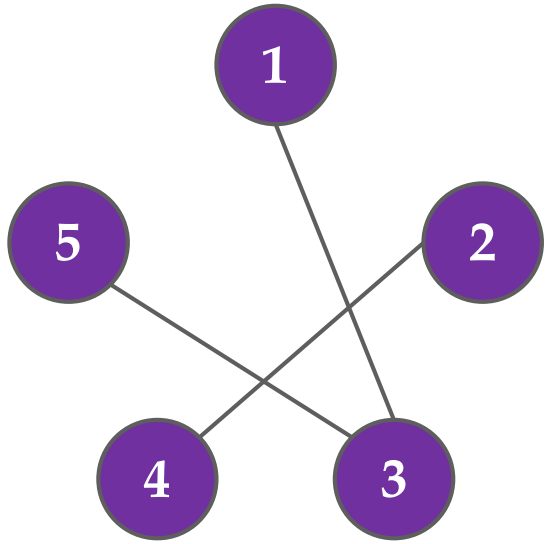
1. Start with an instance of the clique problem: (G, k)
   - $G = (V, E)$
2. In polynomial time, compute $\bar{G}$
   - $\bar{G} = (V, \bar{E})$, where $\bar{E} = \{(v, w) \mid v \neq w \wedge (v, w) \notin E\}$
3. Construct the following instance of the vertex cover problem: $(\bar{G}, |V| - k)$.

Vertex Cover

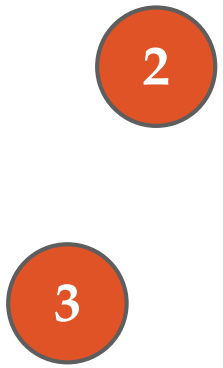
Transformation example:

- Does $\bar{G}$ contain a vertex cover of size $|V| - k$?
  - (k was from the size of the clique in the clique problem, so $|V| - k$ is 2.)

$\bar{G}$:



(Yes, it does)



1. Start with an instance of the clique problem: (G, k)
   - $G = (V, E)$
2. In polynomial time, compute $\bar{G}$
   - $\bar{G} = (V, \bar{E})$, where $\bar{E} = \{(v, w) \mid v \neq w \wedge (v, w) \notin E\}$
3. Construct the following instance of the vertex cover problem: $(\bar{G}, |V| - k)$.

Vertex Cover

Now, we need to prove that $\bar{G}$ has a vertex cover of size $|V| - k$ if and only if G has a clique of size k:

- Assume S is a clique in G.
  - No edge in $\bar{G}$ connects two vertices in S, and
  - Every edge in $\bar{G}$ is incident upon at least one vertex in $V - S$, so
  - $V - S$ is a vertex cover of $\bar{G}$.

Vertex Cover

Now, we need to prove that $\bar{G}$ has a vertex cover of size $|V| - k$ if and only if G has a clique of size k:

- Assume $V - S$ is a vertex cover of $\bar{G}$
  - This means every edge in $\bar{G}$ is incident upon at least one vertex in $V - S$, and
  - No edge in $\bar{G}$ connects two vertices in S, so
  - Every pair of vertices in S is connected in G
    - This is the definition of a clique, so S is a clique in G.