

CS4102 Algorithms

Spring 2020

Just Kidding!

**But you'd get an A and not have to take
the final if you could. 😊**

**Also, you'd be famous, and you'd win
\$1M US for solving one of the
Millennium Prize problems.**

Today's Keywords

- P vs NP
- NP Hard, NP Completeness
- 3SAT
- k-Independent Set
- k-Vertex Cover
- k-Clique

- Readings: CLRS Chapter 34

Summary of Where We Are

- Focusing on “hard” problems, those that seem to be exponential
- Reductions used to show “hardness” relationships between problems
- Starting to define “classes” of problems based on complexity issues
 - P are problems that can be solved in polynomial time
 - NP are problems where a solution can be verified in polynomial time
 - NP-hard are problems that are at least as hard as anything in NP
 - NP-complete are NP-hard problems that “stand or fall together”

Review: P And NP Summary

- **P** = set of problems that can be solved in polynomial time
- **NP** = set of problems for which a solution can be verified in polynomial time
 - Note: this is a more “informal” definition, but it’s fine for CS4102
 - See later slide for more info.
- **$P \subseteq NP$**
- Open question: Does **$P = NP$** ?

Review: Reduction

- A problem A can be *reduced* to another problem B if
 - any instance of A can be “rephrased” to an instance of B, such that...
 - the solution to which provides a solution to the instance of A
 - (We sometimes call this rephrasing a *transformation*)
- Intuitively: If A reduces *in polynomial time* to B, A is “no harder to solve” than B
 - I.e. if B is polynomial, A is not exponential

Review: NP-Hard and NP-Complete

- If A is *polynomial-time reducible* to B, we denote this $A \leq_p B$
- **Definition of NP-Hard and NP-Complete:**
 - If all problems $A \in \mathbf{NP}$ are reducible to B, then B is *NP-Hard*
 - We say B is *NP-Complete* if B is NP-Hard and $B \in \mathbf{NP}$
- If $B \leq_p C$ and B is NP-Complete, C is also NP-Complete
 - Can you explain why?

Before We Move On...

- **Where we want to go next:**
Are there any NP-Hard problems? Any NP-C problems? How do we show a given problem is NP-C?
- But first, a moment to comment on some “subtleties”
 - Pseudo-polynomial
 - The “real” (i.e. formal) definition of NP

Encodings, Input Sizes

- Our CLRS text takes a formal CS approach to these topics
 - Formal languages, encodings, etc.
 - pp. 1055-1061
- Here in CS4102 we're OK being less formal
 - So we've tried to “translate” or simplify a few things
 - So we're talking about this without those using approach shown on those pages!
- But one point about on encoding and input size...

Subtlety #1: Input Size and P

- **Sometimes a problem seems to be in P but really isn't**
- Example: finding if value n is a prime
 - Just loop and do a mod. That's just $\Theta(n)$, isn't it?
- Note that here “ n ” is not the count or number of data items.
 - There's just one input item.
 - But “ n ” is a value with a size that affects the execution time.
 - The size is the number of bits, which is $\log(n)$
 - $T(\text{size}) = n$ but size is $\log(n)$.
 - $T(\text{size}) = T(\log n) = n = 10^{\log n} = 10^{\text{size}}$ This is really an exponential algorithm!
- Be careful when “ n ” is not a count of data items but a value to be processed
 - E.g. Dynamic programming problems (e.g. a table's dimension)
 - We talked about *pseudo-polynomial run-times* in our lectures on DP

Subtlety #2: NP and Decision Problems

- We've said all this theory is based on reasoning about decision problems
- We've said NP are problems where you can check a solution in polynomial time
- But a solution to a decision problem is yes/no or true/false
 - E.g. $k\text{-vertex-cover}(G, k) \rightarrow \text{yes/no}$
 - If we're only given "yes", how can we verify that solution against G and k without solving the problem?

A Glimpse of Formal Definition of NP

- **NP** (*nondeterministic polynomial time*) is the set of decision problems that can be solved in polynomial time by a *nondeterministic* computer
 - Think of a non-deterministic computer as a computer that magically “guesses” a what we’ve thought of as solution, then verifies it is correct
 - Solution for original decision problem is yes/no, but this **witness** or **certificate** is information that allows us to say “yes” or “no”
 - If answer should be “yes” for an input, computer always guesses right witness
 - Or, you can think of it as a parallel machine that generates all possible witnesses and says “yes” if any of those verifies
- For more, see Wikipedia or other source about these topics

What's Next?

- **Where we want to go next:**
 - Are there any NP-Hard problems? Are there any NP-C problems?
 - How do we show a given problem is NP-C?
- Reminder: why do we care?
 - We know $P \subseteq NP$
 - But are they equal or is it a proper subset?
 - In other words, is there a problem in **NP** that cannot be directly solved in polynomial time?
Do some problems in NP have an exponential lower bound?
 - Is $P = NP$? Or not? (The big question!)

Reminder (again)

- **Definition of NP-Hard and NP-Complete:**
 - If all problems $A \in \mathbf{NP}$ are reducible to B , then B is *NP-Hard*
 - We say B is *NP-Complete* if:
 - B is NP-Hard
 - and $B \in \mathbf{NP}$
- If $B \leq_p C$ and B is NP-Complete, C is also NP-Complete
 - We'll see why in two more slides

Proving NP-Completeness

- *What steps do we have to take to prove a problem B is NP-Complete?*
 - Pick a known NP-Hard (or NP-Complete) problem A
 - Assuming there is one! (More later.)
 - Reduce A to B
 - Describe a transformation that maps instances of A to instances of B, such that “yes” for instance of B = “yes” for instance of A
 - Prove the transformation works
 - Prove it runs in polynomial time
 - Oh yeah, prove $B \in \mathbf{NP}$

Order of the Reduction When Proving NP-Completeness

- To prove B is **NP-c**, show $A \leq_p B$ where $A \in \mathbf{NP-Hard}$
 - Why have the known NP-Hard problem “on the left”? Shouldn’t it be the other way around? (No!)
- If $A \in \mathbf{NP-Hard}$, then: all NP problems $\leq_p A$
- If you show $A \leq_p B$, then:
any-NP-problem $\leq_p A \leq_p B$
- Thus any problem in NP can be reduced to B if the two transformations are applied in sequence
 - And both are polynomial
- NP-c are “complete” because: if $A \in \mathbf{NP-c}$ and $A \leq_p B$, then $B \in \mathbf{NP-c}$

“Consequences” of NP-Completeness

- So NP-Complete is a subset of the “hardest” problems in NP, with these important properties:
 - If any *one* NP-Complete problem can be solved in polynomial time...
 - ...then *every* NP-Complete problem can be solved in polynomial time...
 - ...and in fact *every* problem in **NP** can be solved in polynomial time (which would show **P = NP**)
 - Or, prove an exponential lower-bound for *any single* **NP-hard** problem, then *every* **NP-hard** problem (including **NP-C**) is exponential

Therefore: solve (say) traveling salesperson problem in $O(n^{100})$ time, you've proved that **P = NP**. Retire rich & famous!

Can a Problem be NP-Hard but not NP-C?

- So, find a reduction and then try to prove $B \in \mathbf{NP}$
 - *What if you can't?*
- Are there any problems B that are NP-hard but not NP-complete? This means:
 - All problems in NP reduce to B . (A known NP-Hard problem can be reduced to B.)
 - But, B cannot be proved to be in NP
- Yes! Some examples:
 - Non-decision forms of known NP-Cs (e.g. TSP)
 - The halting problem. (Transform a SAT expression to a Turing machine.)
 - Others.

But You Need One NP-Hard First...

- If you have one NP-Hard problem, you can use the technique just described to prove other problems are NP-Hard and NP-c
 - We need an NP-Hard problem to start this off
- The definition of NP-Hard was created to prove a point
 - There *might be* problems that are at least as hard as “anything” (i.e. all NP problems)
- Are there really NP-complete problems?
- **Cook-Levin Theorem: The satisfiability problem (SAT) is NP-Complete.**
 - Stephen Cook proved this “directly”, from first principles, in 1971
 - Proven independently by Leonid Levin (USSR)
 - Showed that any problem that meets the definition of NP can be transformed in polynomial time to a CNF formula.
 - Proof outside the scope of this course (lucky you)

More About The SAT Problem

- The first problem 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: $((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$
- You might imagine that lots of decision problems could be expressed as a complex logical expression
 - And Cook and Levin proved you were right!
 - Proved the general result that any NP problem can be expressed this way

Conjunctive Normal Form (CNF)

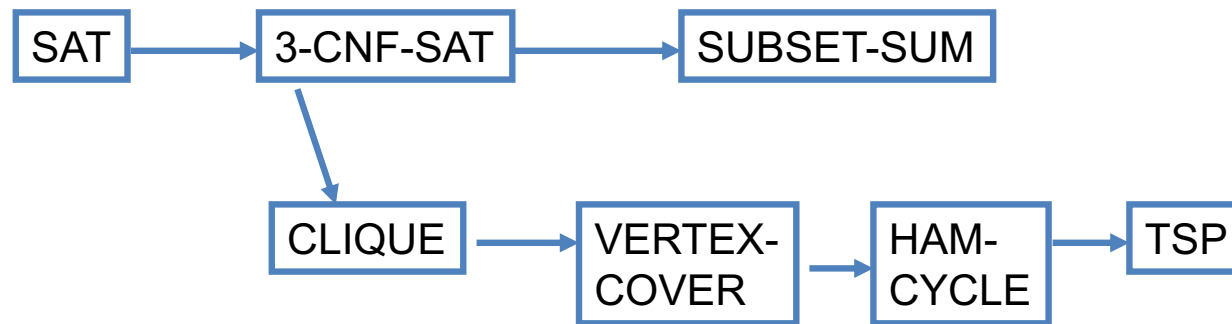
- Even if the form of the Boolean expression is simplified, the problem may be NP-Complete
 - *Literal*: an occurrence of a Boolean or its negation
 - A Boolean formula is in *conjunctive normal form*, or *CNF*, if it is an AND of clauses, each of which is an OR of literals
 - Ex: $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_5)$
 - *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
 - Note: Arbitrary SAT expressions can be translated into CNF forms by introducing intermediate variables etc.

The 3-CNF Problem

- Satisfiability of Boolean formulas in 3-CNF form (the *3-CNF Problem*) is NP-Complete
 - Proof: Also done by Cook (“part 2” of Cook’s theorem)
 - But it’s not that hard to show $\text{SAT} \leq_p \text{3-CNF}$
- The reason we care about the 3-CNF problem is that it is relatively easy to reduce to others
 - Thus by proving 3-CNF is NP-Complete we can prove many seemingly unrelated problems are NP-Complete

Joining the Club

- Given one NP-c problem, others can join the club
 - Prove that SAT reduces to another problem, and so on...



- Membership in NP-c grows...
- Classic textbook: Garey, M. and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1979.

Reductions to Prove NP-C

- Next:
 - A tour of how to prove some problems are NP-C
 - 3-SAT is a good starting point!
 - k -Independent Set
 - k -Vertex Cover
 - k -Clique

Reminder about 3-SAT

- Shown to be NP-hard by Cook
- Given a 3-CNF formula (logical AND of **clauses**, each an OR of 3 **variables**), is there an **assignment** of true/false to each variable to make the formula true (i.e., satisfy the formula)?

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$

Clause

Variables

$x = \text{true}$
 $y = \text{false}$
 $z = \text{false}$
 $u = \text{true}$

k -Independent Set is NP-Complete

1. Show that it belongs to NP
2. Show it is NP-Hard
 - Show $3\text{-SAT} \leq_p k\text{-Independent Set}$

k -Independent Set is in NP

- **Show:** For any graph G :
 - There is a short **witness** (“solution” for search problem) that G has a k -independent set
 - The witness can be checked efficiently (in polynomial time)

Witness for G : $S = \{A, C, E, G, H, J\}$
(nodes in the k -independent set)

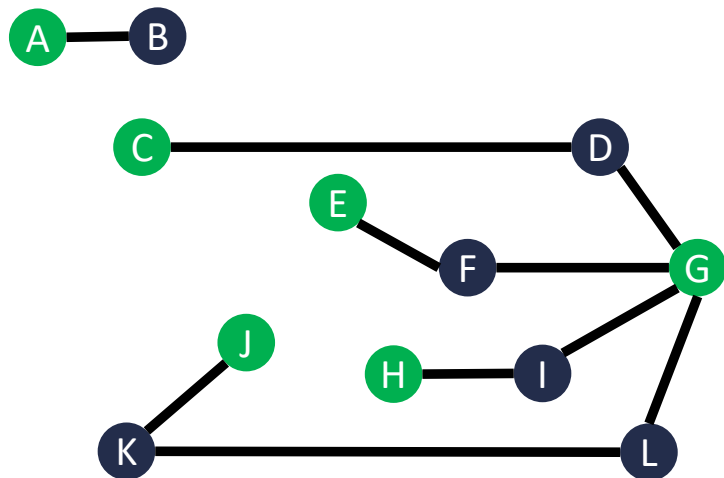
Checking the witness:

- Check that $|S| = k$
- Check that every edge is incident on at most one node in S

$$O(k) = O(|V|)$$

$$O(|V| + |E|)$$

Total time: $O(|E| + |V|) = \text{poly}(|V| + |E|)$



Graph G

k -Independent Set is NP-Complete

1. Show that it belongs to NP
2. Show it is NP-Hard
 - Show $3\text{-SAT} \leq_p k\text{-Independent Set}$



3-SAT \leq_p k -Independent Set

3-SAT

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z})$$

$x = \text{true}$
 $y = \text{false}$
 $z = \text{false}$
 $u = \text{true}$

Map instances of problem
A to instances of **B**

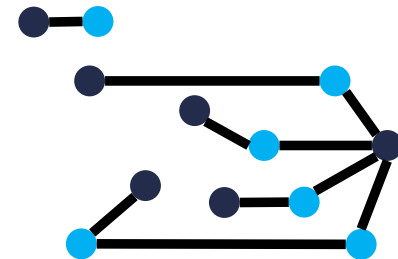
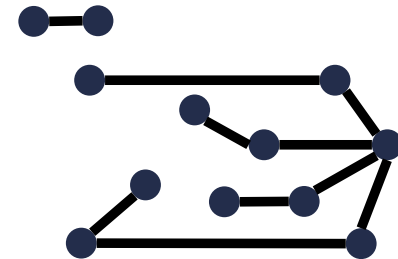
polynomial time

Map solutions of problem
B to solutions of **A**

polynomial time

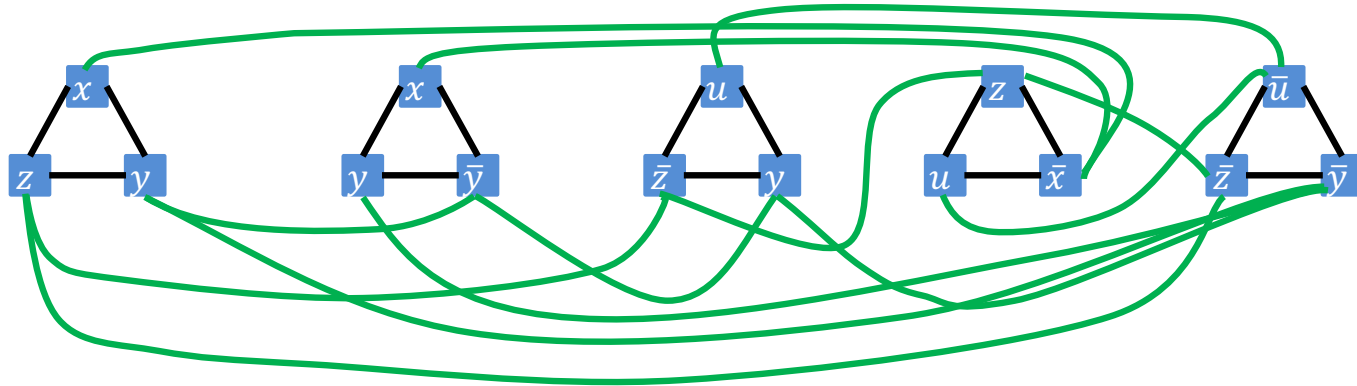
polynomial-time reduction

k -independent set



3-SAT \leq_p k -Independent Set

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$



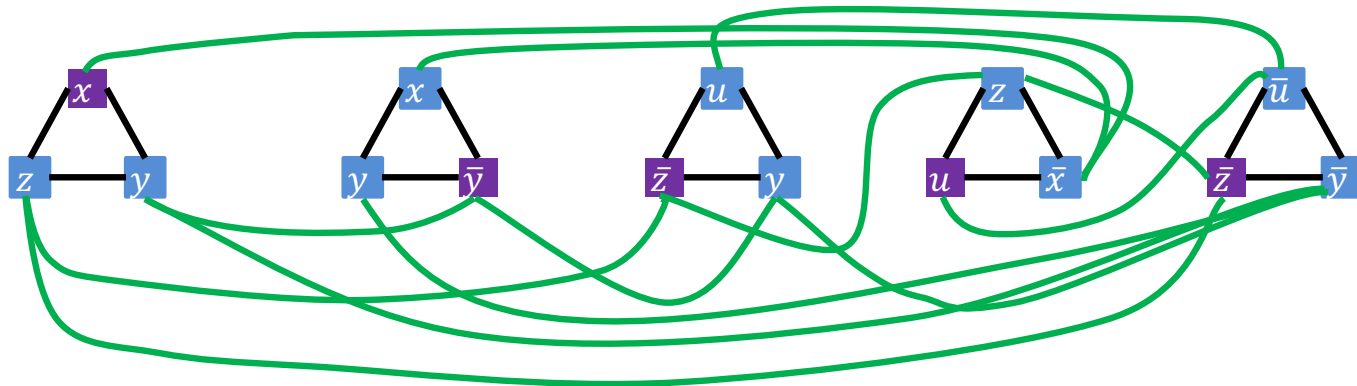
For each clause, construct a triangle graph with its three variables as nodes
 Add an edge between each node and its negation

Let k = number of clauses

Claim. There is a k -independent set in this graph if and only if there is a satisfying assignment

3-SAT \leq_p k -Independent Set

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$



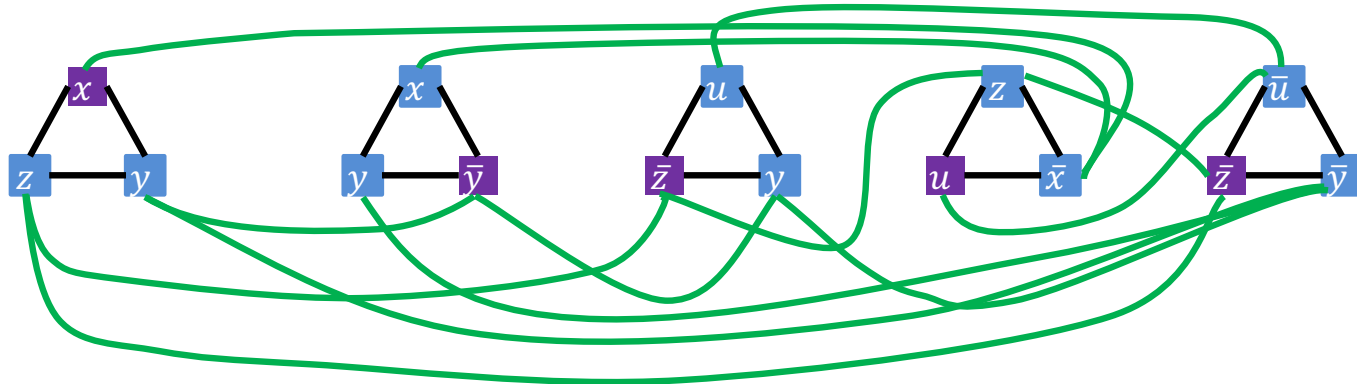
$x = \text{true}$
 $y = \text{false}$
 $z = \text{false}$
 $u = \text{true}$

Suppose there is a k -independent set S in this graph G

- By construction of G , at most one node from each triangle is in S
- Since $|S| = k$ and there are k triangles, each triangle contributes one node
- If a variable x is selected in one triangle, then \bar{x} is never selected in another triangle (since each variable is connected to its negation)
- There are no contradicting assignments, so can set variable chosen in each triangle to “true”; satisfying assignment by construction

3-SAT \leq_p k -Independent Set

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$



$x = \text{true}$
 $y = \text{false}$
 $z = \text{false}$
 $u = \text{true}$

Suppose there is a **satisfying assignment** to the formula

- At least one variable in each clause must be true
- Add the node to that variable to the set S
- There are k clauses, so set S has exactly k nodes
- If we use x in any clause, we will never use \bar{x} , so there are no edges among the nodes in S

3-SAT \leq_p k -Independent Set

3-SAT

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z})$$

$x = \text{true}$
 $y = \text{false}$
 $z = \text{false}$
 $u = \text{true}$

Map instances of problem
A to instances of **B**

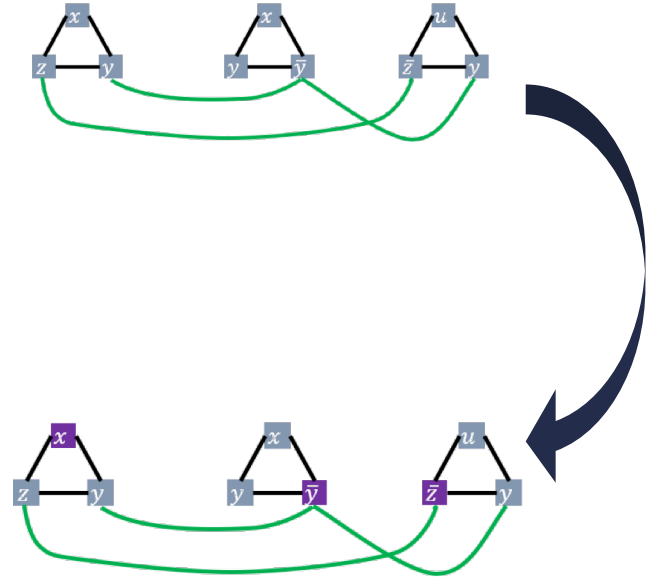
polynomial time

Map solutions of problem
B to solutions of **A**

polynomial time

polynomial-time reduction

k -independent set



k -Independent Set is NP-Complete

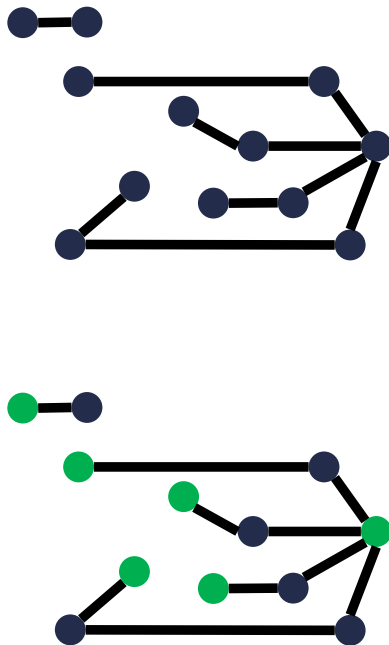
1. Show that it belongs to NP
2. Show it is NP-Hard
 - Show $3\text{-SAT} \leq_p k\text{-independent set}$



- Next example: k -Vertex Cover
- Remember?
 - We did the following reduction in an earlier slide set!
 k -Independent Set $\leq_p k$ -Vertex Cover
 - We just showed k -Independent Set is NP-C
 - Therefore.... (you know, right?)

Max Independent Set \leq_p k -Vertex Cover

k -independent set



Map instances of problem A to instances of B

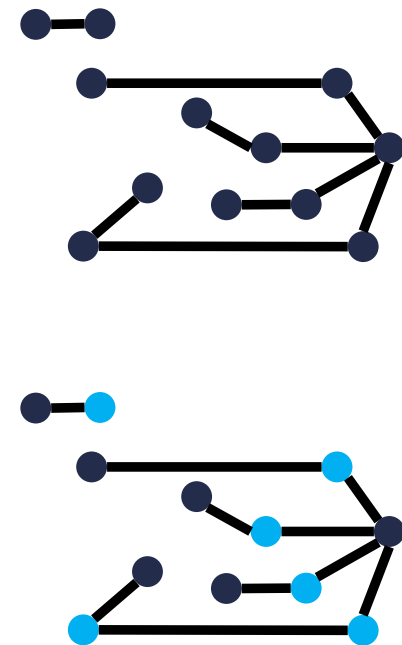
$O(1)$ time

Map solutions of problem B to solutions of A



$O(|V|)$ time

Reduction

k -vertex cover



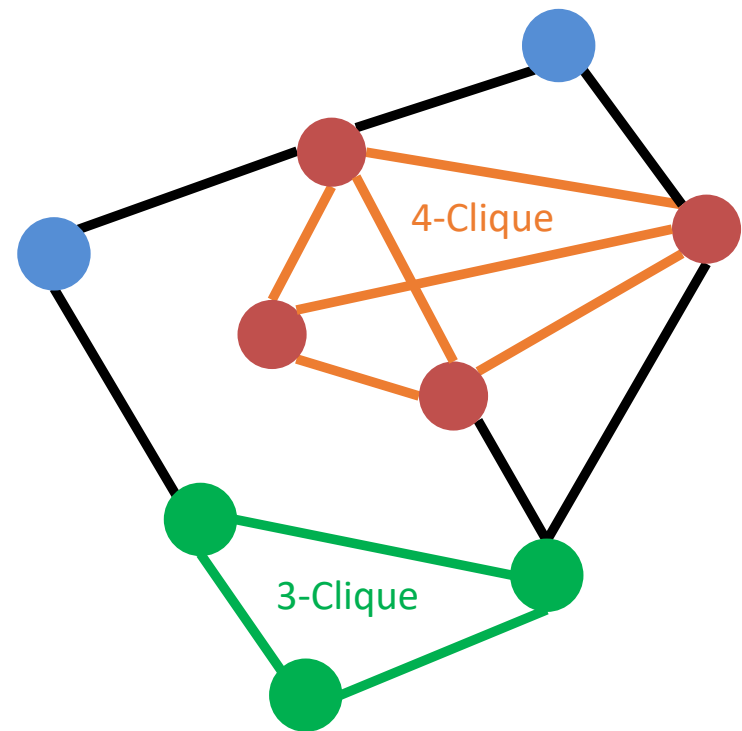
k -Vertex Cover is NP-Complete

1. Show that it belongs to NP 
 - Given a candidate cover, check that every edge is covered
2. Show it is NP-Hard 
 - Show k -independent set $\leq_p k$ -vertex cover

- 
- Next example: k -Clique

k -Clique Problem

- **Clique:** A complete subgraph
- **k -Clique problem:** given a graph G and a number k , is there a clique of size k ?



k -Clique is NP-Complete

1. Show that it belongs to NP
 - Give a polynomial time verifier
2. Show it is NP-Hard
 - Give a reduction from a known NP-Hard problem
 - We will show $3\text{-SAT} \leq_p k\text{-clique}$

k -Clique is in NP

- **Show:** For any graph G :
 - There is a short witness (“solution”) that G has a k -clique
 - The witness can be checked efficiently (in polynomial time)

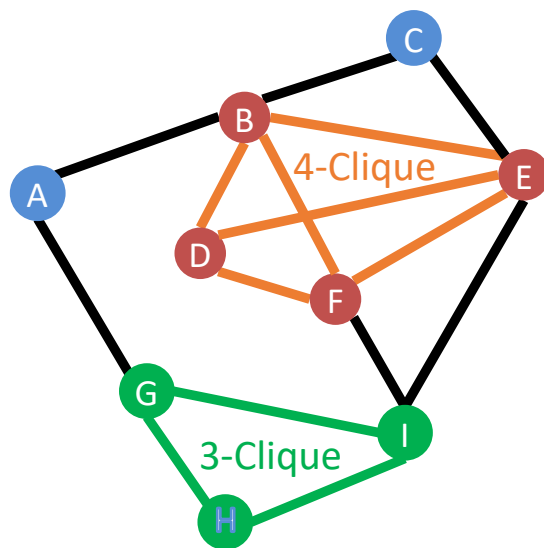
Suppose $k = 4$

Witness for G : $S = \{B, D, E, F\}$
(nodes in the k -clique)

Checking the witness:

- Check that $|S| = k$ $O(k) = O(|V|)$
- Check that every pair of nodes in S share an edge $O(k^2) = O(|V|^2)$

Total time: $O(|V|^2) = \text{poly}(|V| + |E|)$



k -Clique is NP-Complete

1. Show that it belongs to NP



- Give a polynomial time verifier

2. Show it is NP-Hard

- Give a reduction from a known NP-Hard problem
- We will show $3\text{-SAT} \leq_p k\text{-clique}$

3-SAT \leq_p k -Clique

3-SAT

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z})$$

$x = \text{true}$
 $y = \text{false}$
 $z = \text{false}$
 $u = \text{true}$

Map instances of problem
A to instances of **B**

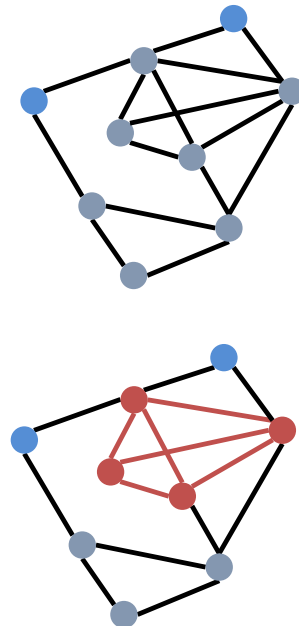
polynomial time

Map solutions of problem
B to solutions of **A**

polynomial time

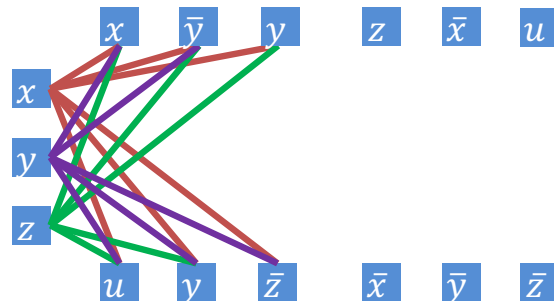
polynomial-time reduction

k -clique



3-SAT \leq_p k -Clique

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$



(also do this for the other clauses,
omitted due to clutter)

For each clause, introduce a node for each of its three variables

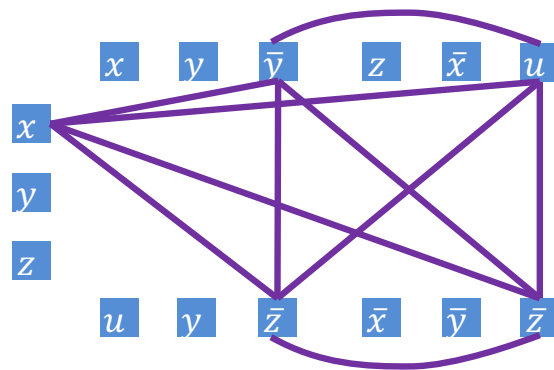
Add an edge from each node to all non-contradictory nodes in the other clauses (i.e., to all nodes that is not the negation of its own variable)

Let k = number of clauses

Claim. There is a k -clique in this graph if and only if there is a satisfying assignment

$$3\text{-SAT} \leq_p k\text{-Clique}$$

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$

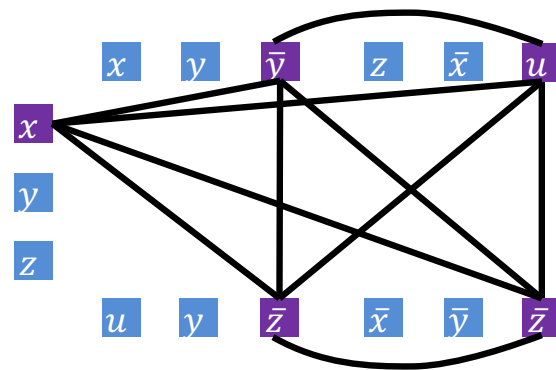


Suppose there is a *k*-clique in this graph

- There are no edges between nodes for variables in the same clause, so *k*-clique must contain one node from each clause
- Nodes in clique cannot contain variable and its negation
- Nodes in clique must then correspond to a satisfying assignment

$$3\text{-SAT} \leq_p k\text{-Clique}$$

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$



Suppose there is a **satisfying assignment** to the formula

- For each clause, choose one node whose value is true
- There are k clauses, so this yields a collection of k nodes
- Since the assignment is consistent, there is an edge between every pair of nodes, so this constitutes a k -clique

3-SAT \leq_p k -Clique

3-SAT

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z})$$

$x = \text{true}$
 $y = \text{false}$
 $z = \text{false}$
 $u = \text{true}$

Map instances of problem **A** to instances of **B**

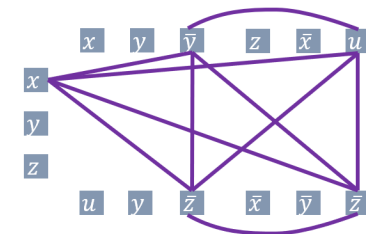
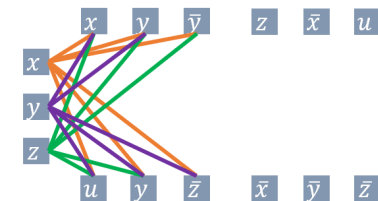
polynomial time

Map solutions of problem **B** to solutions of **A**

polynomial time

polynomial-time reduction

k -clique



k -Clique is NP-Complete

1. Show that it belongs to NP



- Give a polynomial time verifier

2. Show it is NP-Hard



- Give a reduction from a known NP-Hard problem
- We will show $3\text{-SAT} \leq_p k\text{-clique}$

Wrap Up and Reminders