NP-Completeness

CS 4102: Algorithms

Fall 2021

Mark Floryan and Tom Horton

Topics

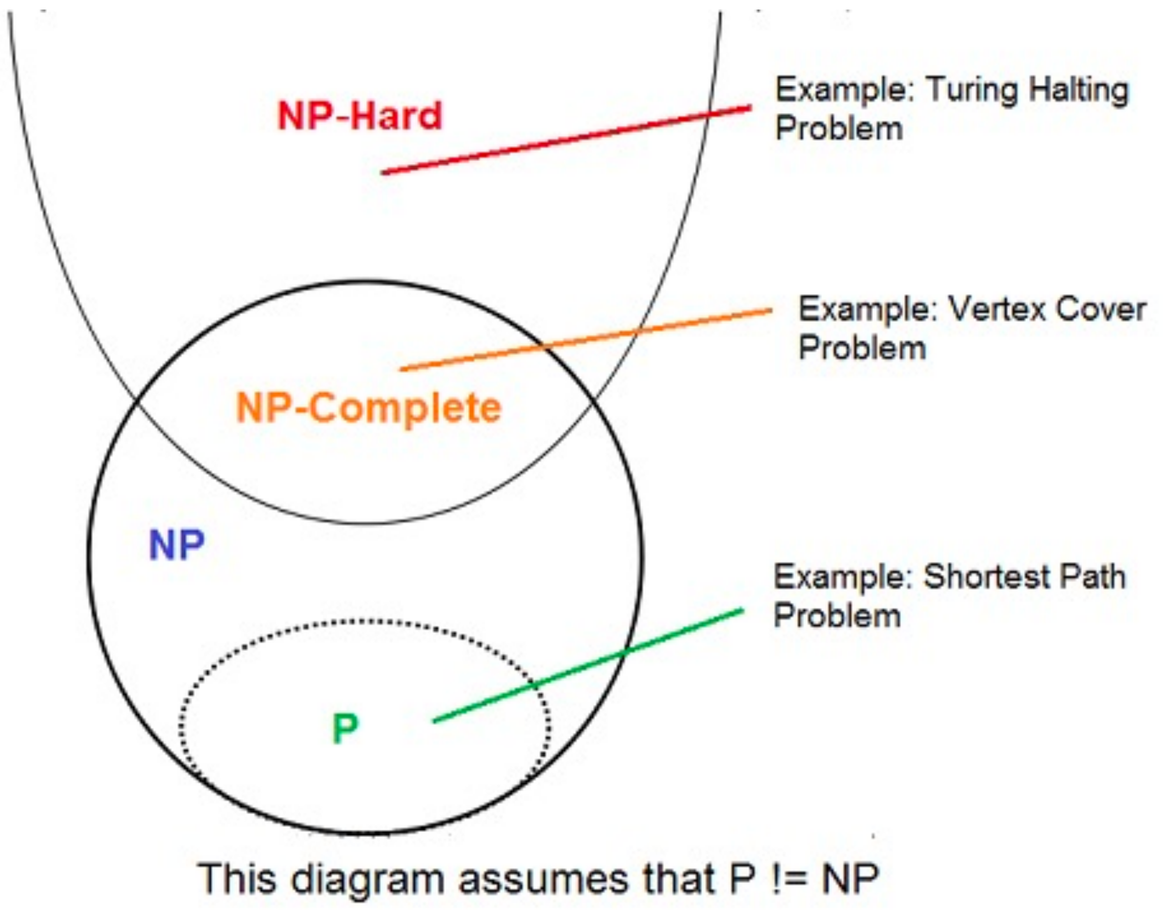
Why Study NP-Completeness

- All semester, we've studied ***finding algorithms*** to solve problems using various tools.
- Sometimes we instead need to prove that a problem is ***extremely hard***, so as not to waste time on it!
  - NP-Complete Problems are hard
  - Let's go over a few of them quickly
  - Let's show how to prove a new problem is NP-Complete

NP-Completeness

Quick Background!

- ***P***: Set of problems solved in polynomial time (e.g., sorting a list)
- ***NP***: Set of problems that can be:
  - 1) Solved in non-deterministic polynomial time
  - 2) Verified in polynomial time
- ***NP-Hard***: Set of problems that are as hard as (or harder) than the hardest problems in NP
- ***NP-Complete***: Set of problems that are both NP and NP-Hard (i.e., the equally hardest problems in NP)



NP-Completeness

- So…a problem is NP-Complete if you can do the following:

- 1) Show how to verify it in polynomial time
  - Given a solution to the problem, verify it is correct
  - That algorithm's runtime needs to be a polynomial (usually easy)

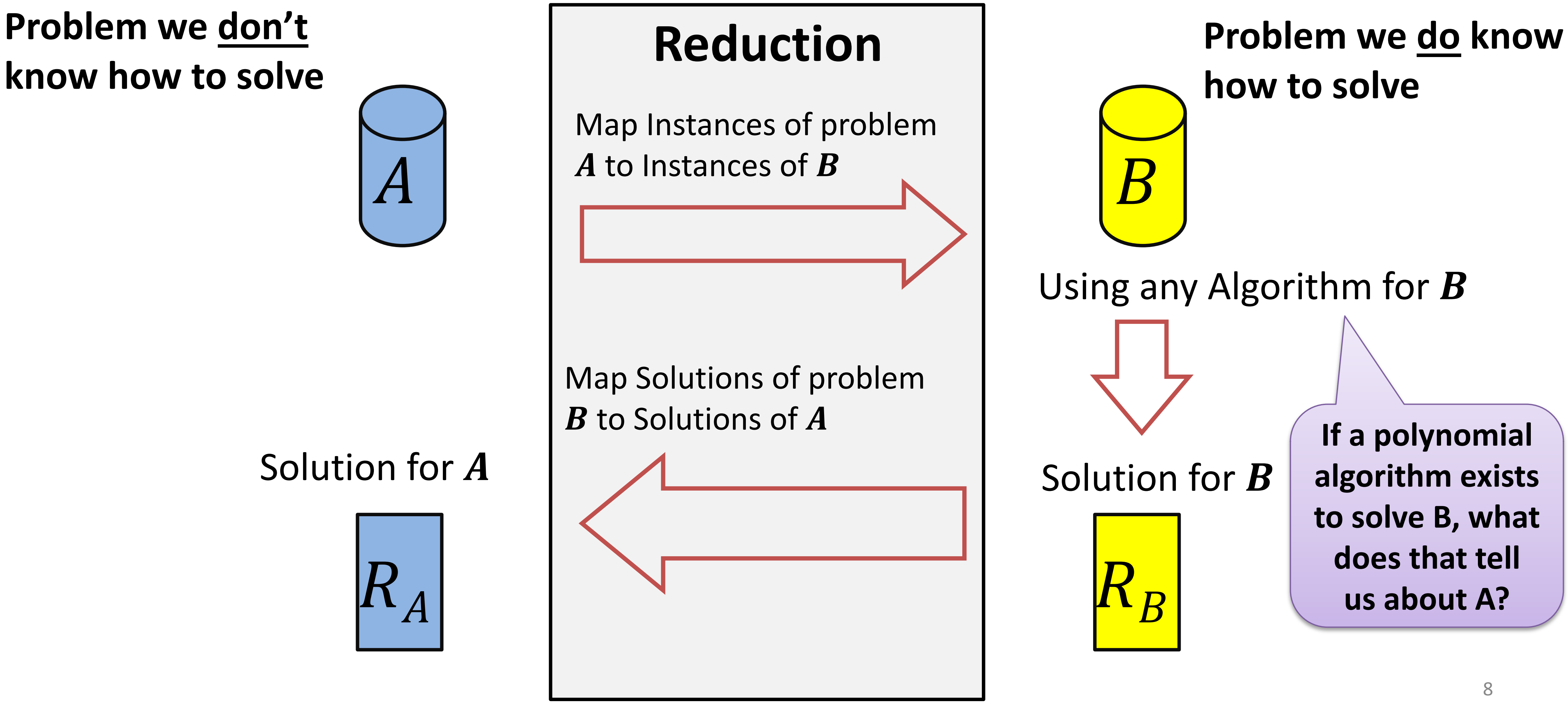
- 2) Show the problem is NP-Hard (harder than a known NP-C Problem)
  - Take a currently known NP-C problem (let's call it A)
  - Show that $A \leq_p X$ //where X is your problem
  - Why? If A is NP-Hard, then: *any NP problem* $\leq_p A$
  - Transitivity: *any NP problem* $\leq_p A \leq_p X$
  - So X satisfies definition of NP-Hard

“Consequences” of NP-Completeness

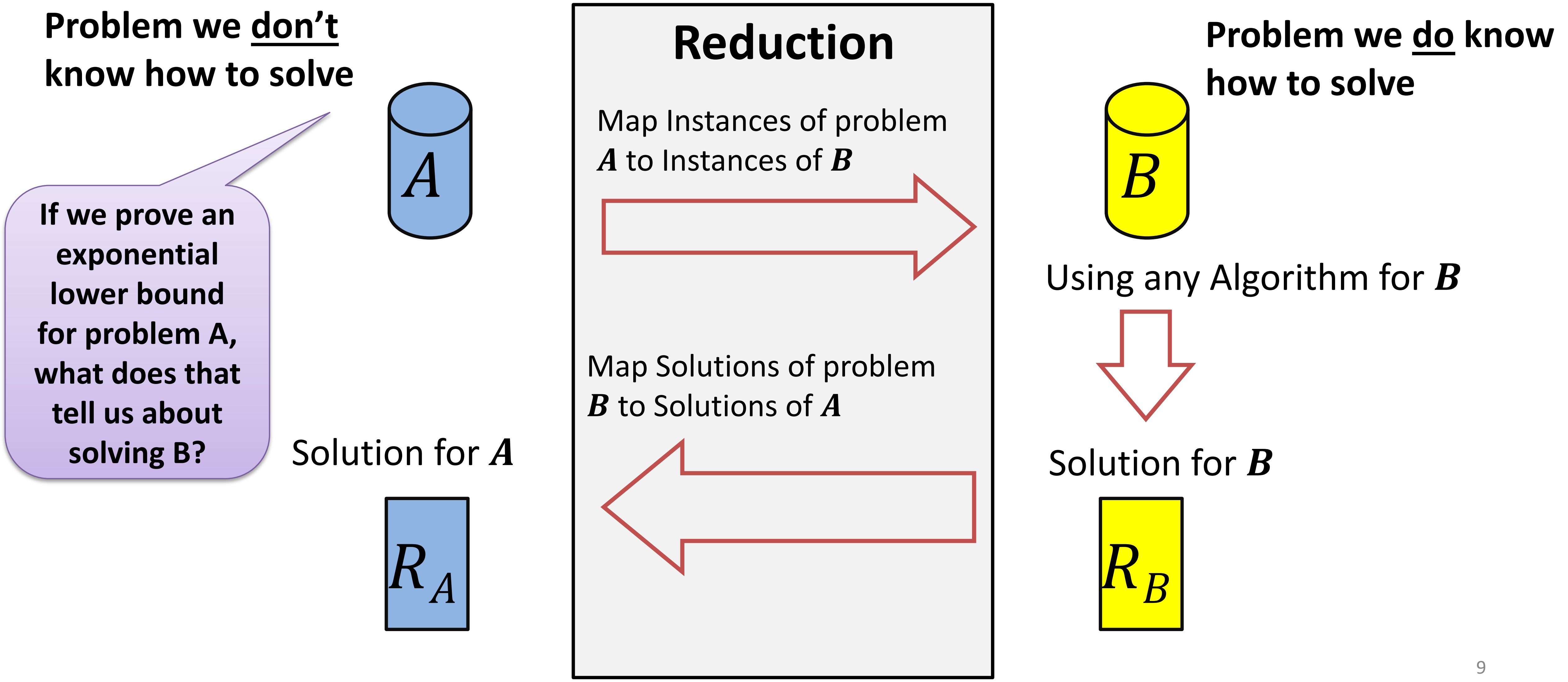
- NP-Complete is the set of “hardest” problems in NP, with these important properties:
  - If any *one* NP-Complete problem can be solved in polynomial time…
  - …then *every* NP-Complete problem can be solved in polynomial time…
  - …and in fact *every* problem in **NP** can be solved in polynomial 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!

$A \le_p B$ and B in P



$A \leq_p B$ and we prove A <u>not</u> in P



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-C 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)

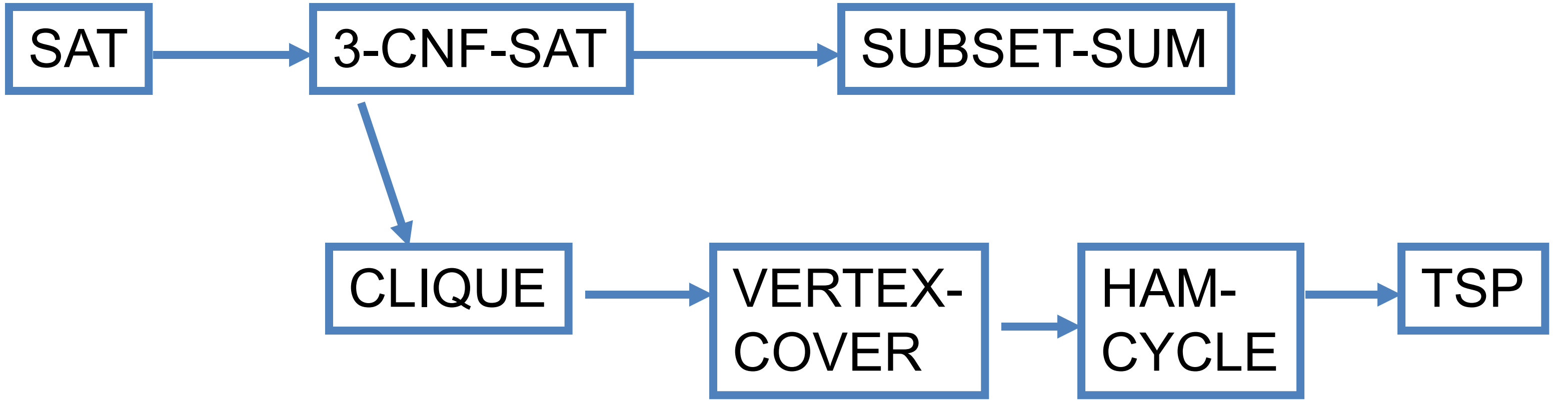
- Even if the form of the Boolean expression is simplified, the problem may be NP-Complete
  - *Literal*: an occurrence of a Boolean or its negation
  - A Boolean formula is in *conjunctive normal form*, or *CNF*, if it is an AND of clauses, each of which is an OR of literals
    - Ex: $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_5)$
  - *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 SAT $\leq_p$ 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-Clique (We will probably only have time for this one)
  - k-Independent Set
  - k-Vertex Cover

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., <u>satisfy</u> the formula)?

$$\underbrace{(x \vee y \vee z)}_{\text{Clause}} \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$

Variables

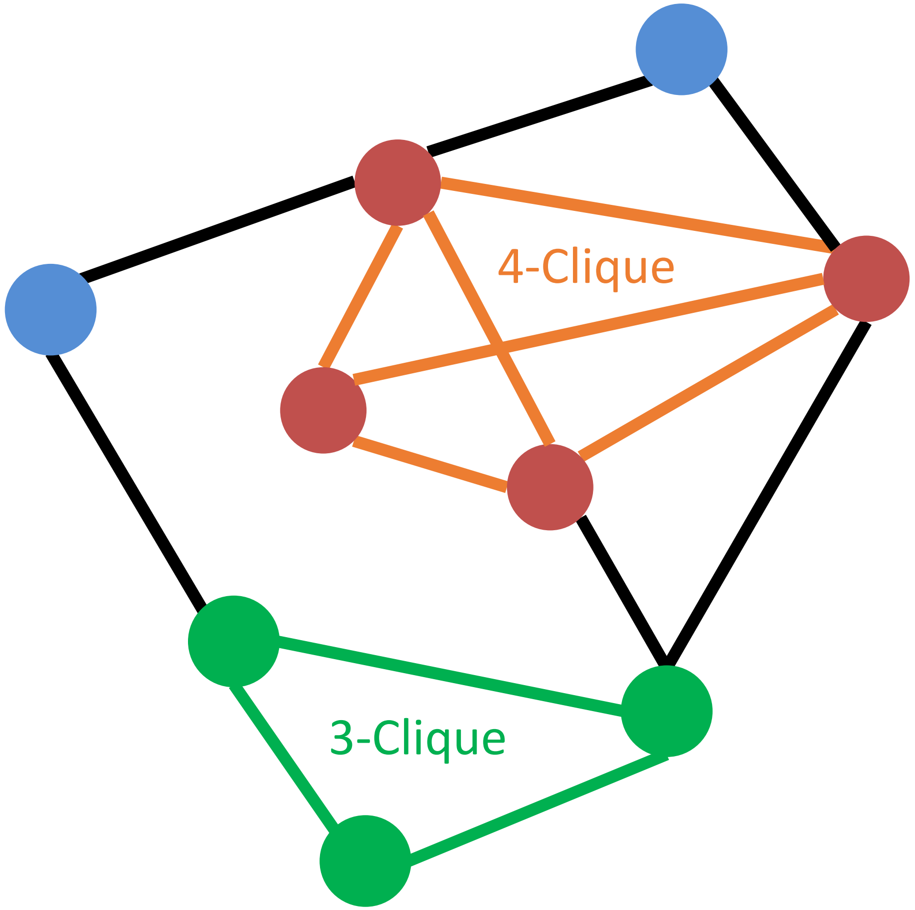
$x = \text{true}$
$y = \text{false}$
$z = \text{false}$
$u = \text{true}$

- Next example: *k*-Clique

- Let’s show that k-Clique is NP-Complete!

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?

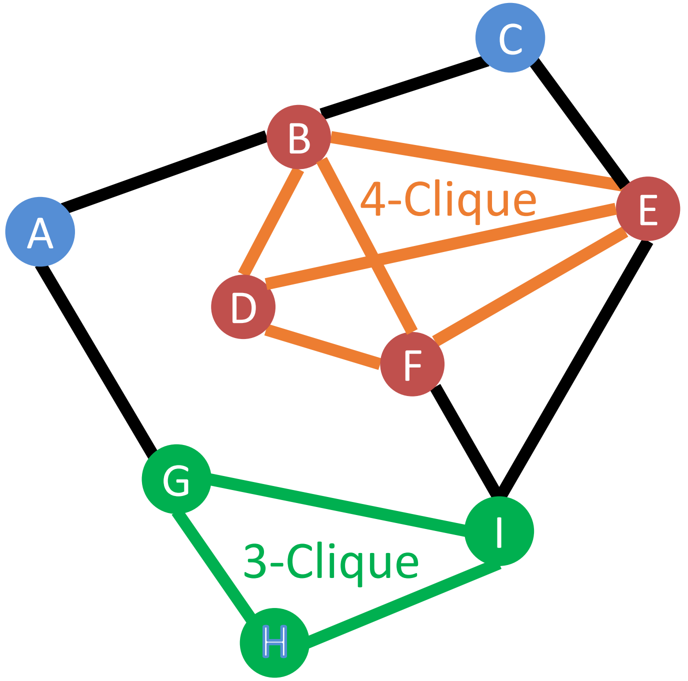


k-Clique is NP-Complete

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-SAT $\leq_p$ k-clique

k-Clique is in NP

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



Graph G

Suppose $k = 4$

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

Checking the certificate:

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

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-SAT $\leq_p$ k-clique

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

3-SAT

$(x \lor y \lor z) \land (x \lor \bar{y} \lor y) \land (u \lor y \lor \bar{z}) \land$
$(z \lor \bar{x} \lor u) \land (\bar{x} \lor \bar{y} \lor \bar{z})$

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

Map instances of problem $\boldsymbol{A}$ to instances of $\boldsymbol{B}$

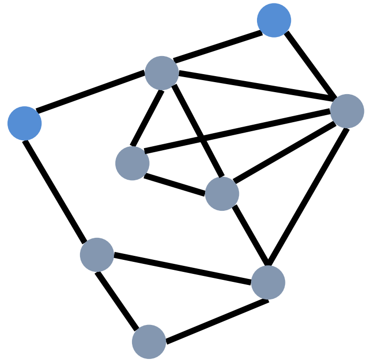
polynomial time

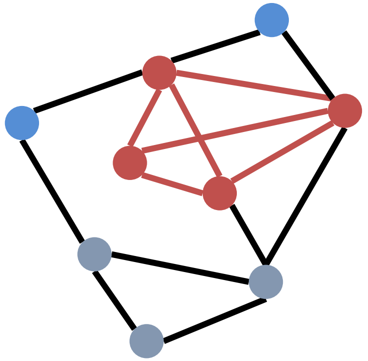
Map solutions of problem $\boldsymbol{B}$ to solutions of $\boldsymbol{A}$

polynomial time

polynomial-time reduction

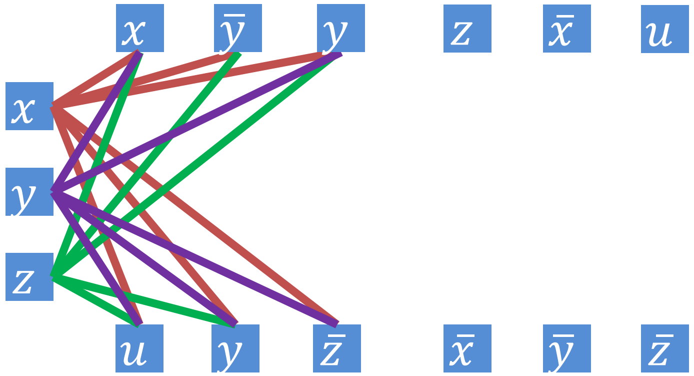
k-clique





3-SAT $\leq_p$ $\boldsymbol{k}$-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

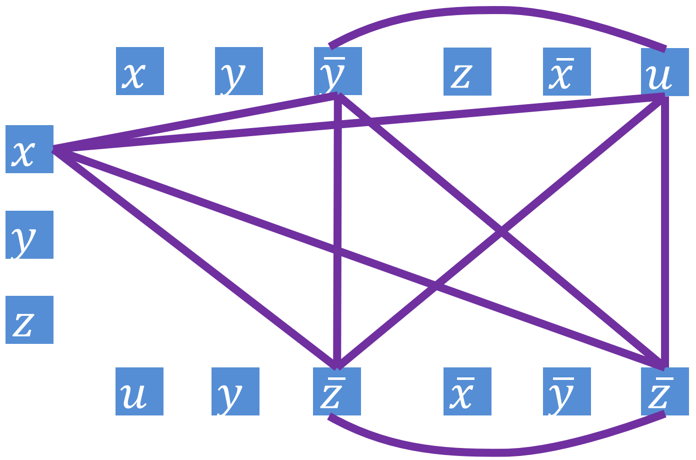
Add an edge from each node to all non-contradictory nodes in the other clauses (i.e., to all nodes that is not the negation of its own variable)

Let k = number of clauses

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

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

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

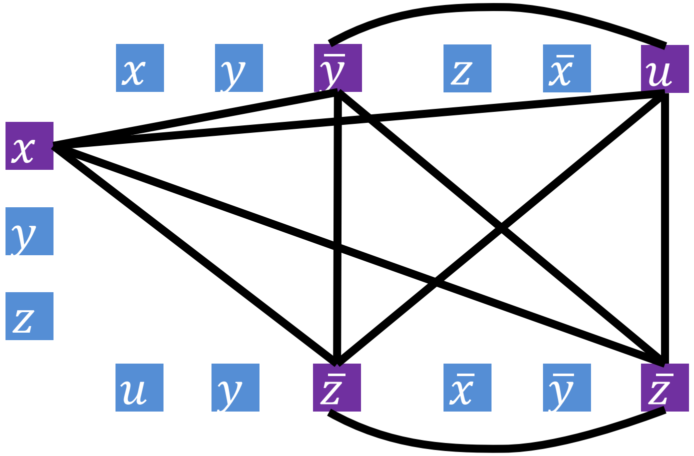


Suppose there is a k-clique in this graph

- 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-SAT $\leq_p$ $\boldsymbol{k}$-Clique

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



x = true
y = false
z = false
u = true

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 $\leq_p$ $\boldsymbol{k}$-Clique

3-SAT

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

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

Map instances of problem $\boldsymbol{A}$ to instances of $\boldsymbol{B}$

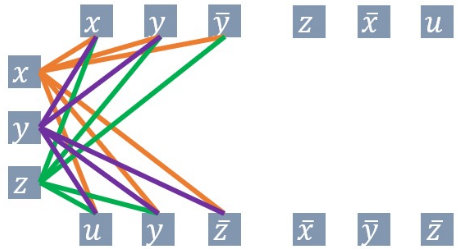
polynomial time

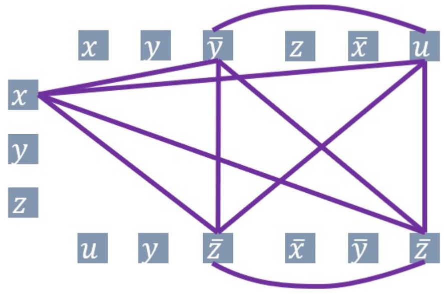
Map solutions of problem $\boldsymbol{B}$ to solutions of $\boldsymbol{A}$

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-SAT $\leq_p$ k-clique

END HERE

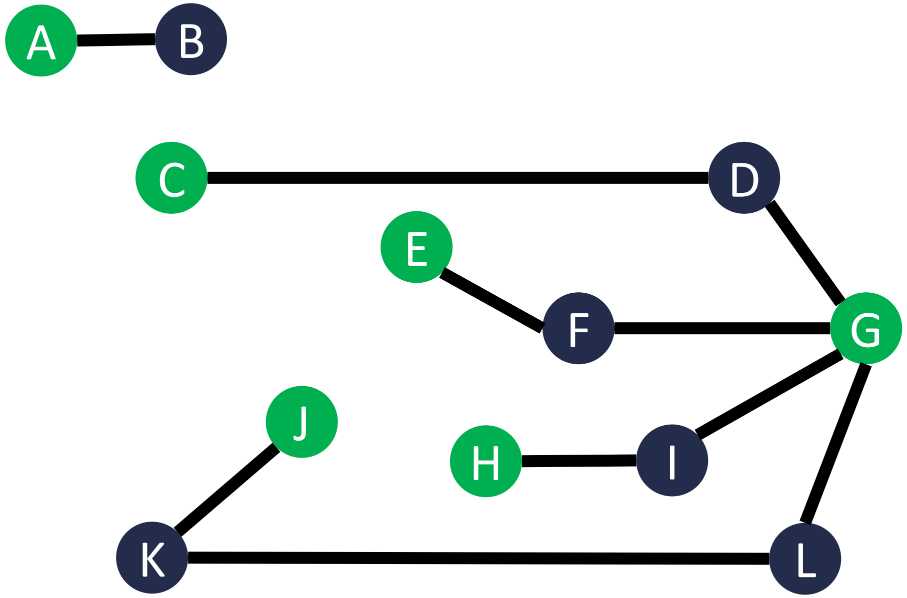
More below if you are interested

k-Independent Set is NP-Complete

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

k-Independent Set is in NP

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



Graph G

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

Checking the certificate:

- Check that $|S| = k$ — $O(k) = O(|V|)$
- Check that every edge is incident on at most one node in S — $O(|V| + |E|)$

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

k-Independent Set is NP-Complete

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

3-SAT $\le_p$ $\boldsymbol{k}$-Independent Set

3-SAT

$(x \lor y \lor z) \land (x \lor \bar{y} \lor y) \land (u \lor y \lor \bar{z})$

x = true
y = false
z = false
u = true

Map instances of problem $\boldsymbol{A}$ to instances of $\boldsymbol{B}$

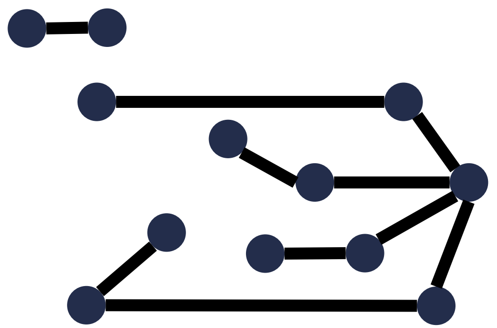
polynomial time

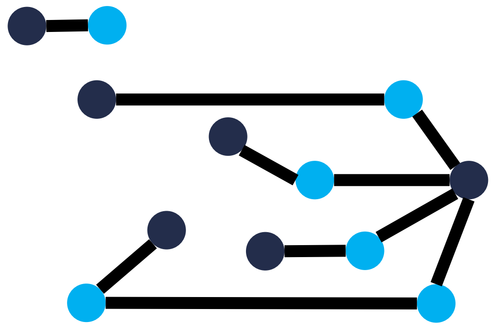
Map solutions of problem $\boldsymbol{B}$ to solutions of $\boldsymbol{A}$

polynomial time

polynomial-time reduction

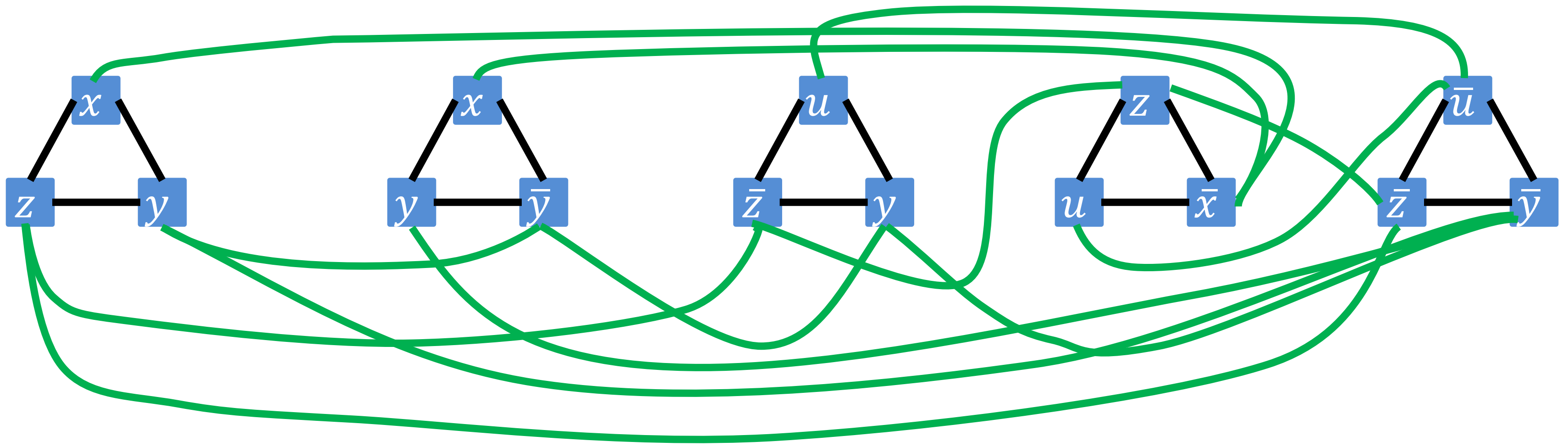
k-independent set





3-SAT $\leq_p$ $\boldsymbol{k}$-Independent Set

$$(x \lor y \lor z) \land (x \lor \bar{y} \lor y) \land (u \lor y \lor \bar{z}) \land (z \lor \bar{x} \lor u) \land (\bar{x} \lor \bar{y} \lor \bar{z})$$



For each clause, construct a <u>triangle graph</u> with its three variables as nodes

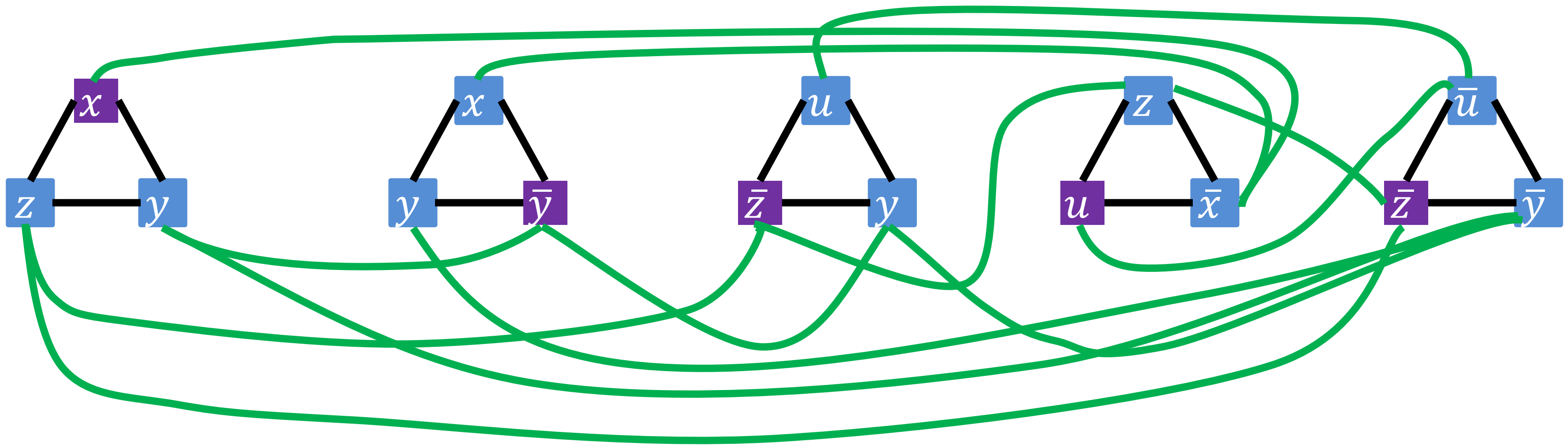
Add an edge between each node and its negation

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 $\leq_p$ $\boldsymbol{k}$-Independent Set

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



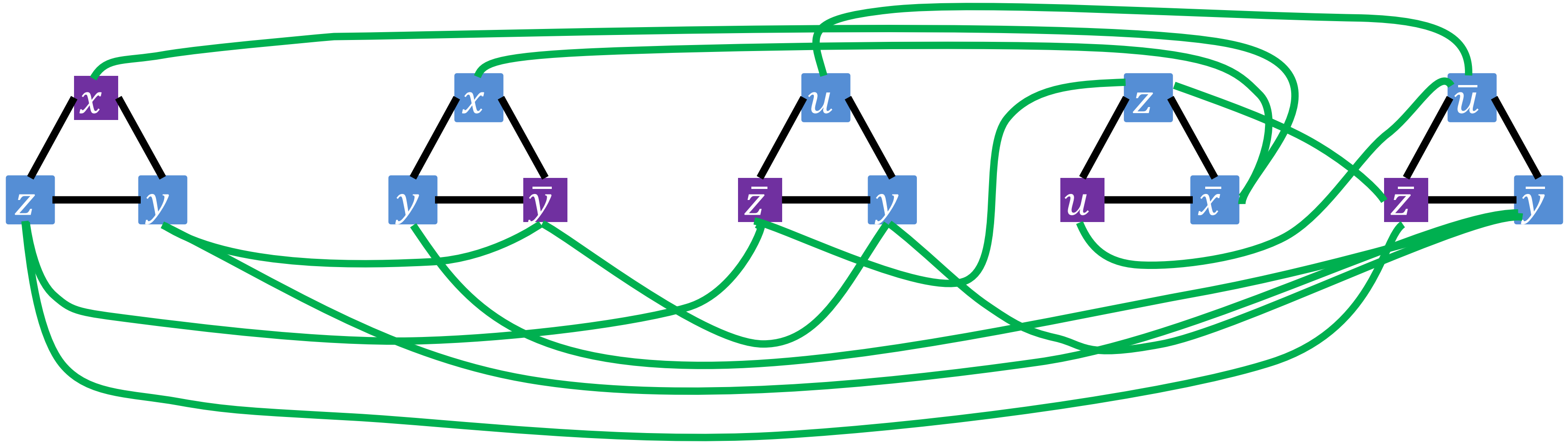
$x = \text{true}$
$y = \text{false}$
$z = \text{false}$
$u = \text{true}$

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 $\leq_p$ $\boldsymbol{k}$-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 $\leq_p$ $\boldsymbol{k}$-Independent Set

3-SAT

$(x \lor y \lor z) \land (x \lor \bar{y} \lor y) \land (u \lor y \lor \bar{z})$

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

Map instances of problem $\boldsymbol{A}$ to instances of $\boldsymbol{B}$

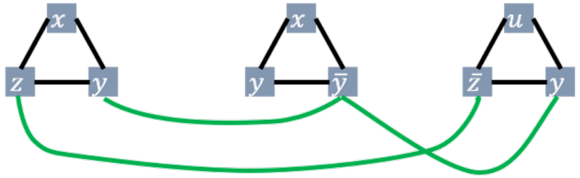
polynomial time

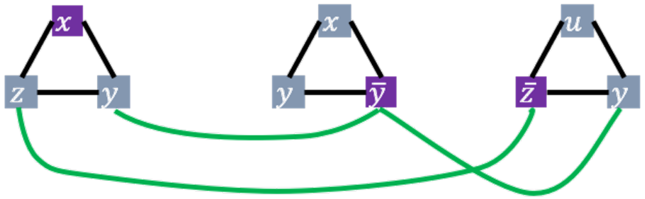
Map solutions of problem $\boldsymbol{B}$ to solutions of $\boldsymbol{A}$

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-SAT $\leq_p$ k-independent set

- Next example: *k*-Vertex Cover

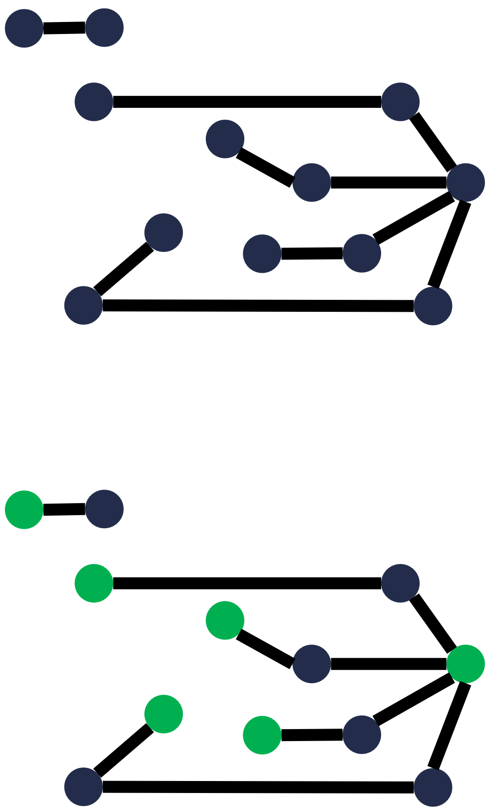
- Remember?
  - We did the following reduction in an earlier slide set!

    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 $\boldsymbol{A}$ to instances of $\boldsymbol{B}$

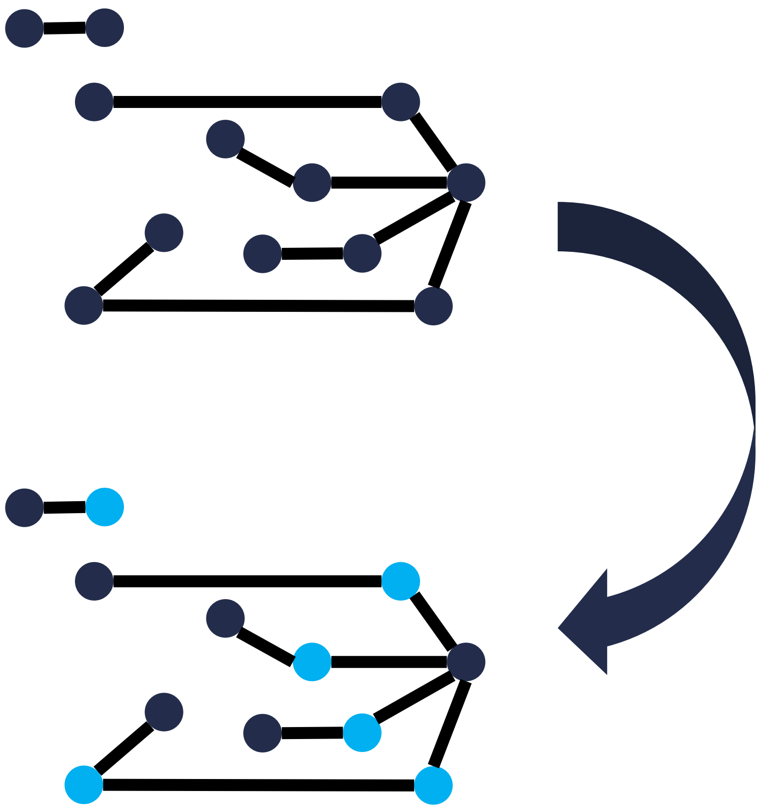
$O(1)$ time

Map solutions of problem $\boldsymbol{B}$ to solutions of $\boldsymbol{A}$

$O(|V|)$ time

Reduction

k-vertex cover



k-Vertex Cover is NP-Complete

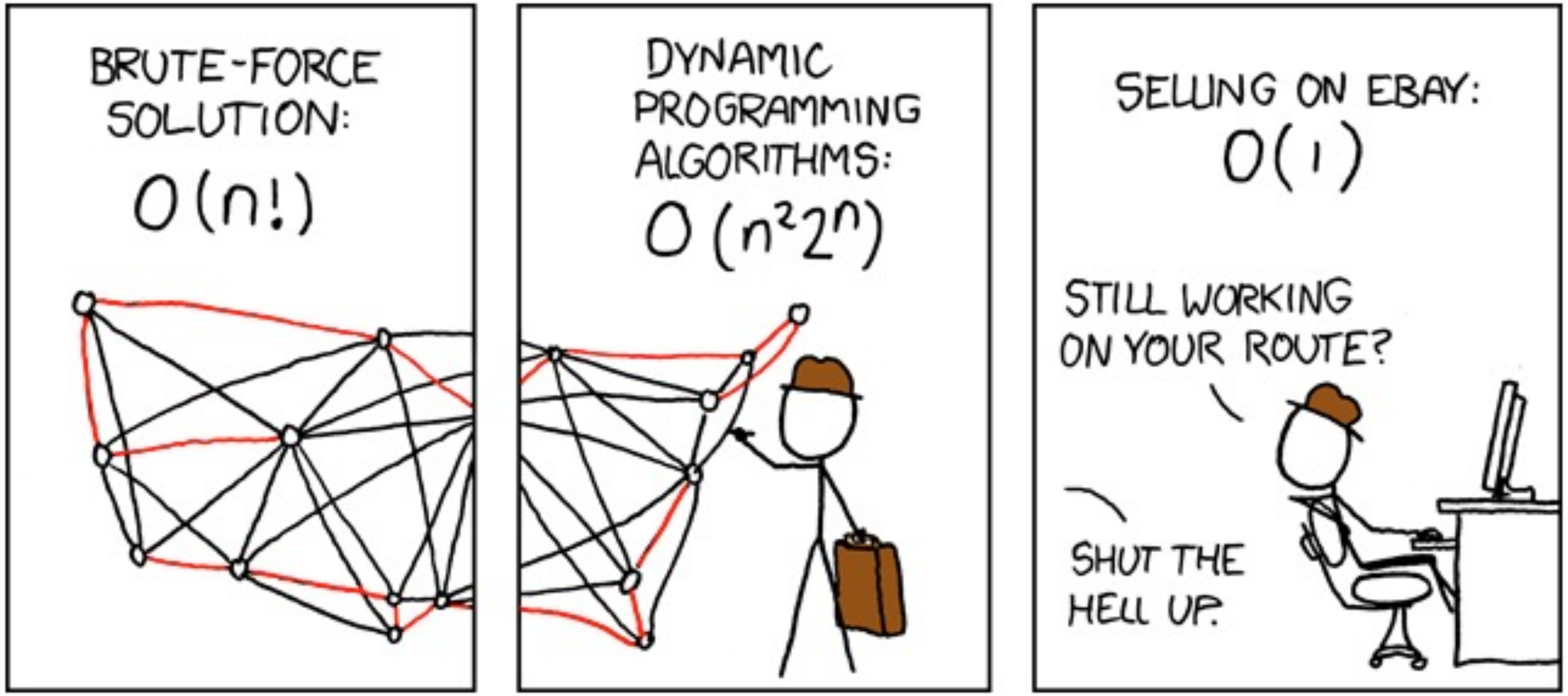
1. Show that it belongs to NP
   - Given a candidate cover, check that every edge is covered
2. Show it is NP-Hard
   - Show k-independent set $\leq_p$ k-vertex cover

Wrap Up and Reminders

Why Prove NP-Completeness?

- Though nobody has proven that **P ≠ NP**, if you prove a problem NP-Complete, most people accept that it is probably exponential
- Therefore it can be important for you to prove that a problem is NP-Complete
  - Don't need to try to come up perfect non-exponential algorithm
  - Can instead work on *approximation algorithms*

What's a poor salesperson to do?



http://xkcd.com/399/

Approximation Algorithms

- Look at first 3 pages of Ch. 35 of CLRS textbook
- Can we find an algorithm for problem A $\in$ **NP-C** that:
  - Runs in polynomial time
  - Gets "near optimal" results
- Prove some bound on the algorithm's correctness in terms of the true optimal result
  - No worse that (some factor) of optimal
  - "It's not always right (best), but it's guaranteed to be this close."

General Comments

- At least 3000 problems have been shown to be NP-Complete
  - That number is from a non-recent report, so we might say that counts is a weak lower-bound on the true number found
  - https://en.wikipedia.org/wiki/List_of_NP-complete_problems including some popular games
- Some reductions are profound, some are comparatively easy, many are easy once the key insight is given

Other NP-Complete Problems

- *Hamilton Path/Cycle, Traveling Salesperson*
- *Subset-sum*: Given a set of integers, does there exist a subset that adds up to some target T ?
- *0-1 knapsack*: when weights not just integers
- *Graph coloring*: can a given graph be colored with k colors such that no adjacent vertices are the same color?
- Etc…

Review (Again)

- A problem B is *NP-complete*
  - if it is in NP **and** it is NP-hard.
- A problem B is *NP-hard*
  - if *every* problem in NP is reducible to **B**.
- A problem A is *reducible* to a problem B if
  - there exists a polynomial reduction function T such that
    - For every string x,
    - if x is a yes input for A, then T(x) is a yes input for B
    - if x is a no input for A, then T(x) is a no input for B.
    - T can be computed in polynomially bounded time.

“Consequences” of NP-Completeness

- NP-Complete the set 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-C** problem, then *every* **NP-C** problem is exponential

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

What We <u>Don't</u> Know: Open Questions

– Is it **impossible** to solve an NP-c problem in polynomial time?
  - No one has proved an exponential lower bound for any problem in NP.
  - But, most computer scientists <u>believe</u> such a lower bound exists for NP-c problems.

– Are all problems in NP tractable or intractable? I.e., does P=NP or not?
  - If someone found a polynomial solution to any NP-c problem, we'd know P = NP.
  - But, most computer scientists <u>believe</u> P≠ NP.