CS4102 Algorithms

Spring 2021 – Floryan and Horton

Module 4, Day 4: Recorded Lecture

Topics: NP, NP-Hard, NP Complete

Readings: CLRS Chapter 34 (be selective)

Some Preliminaries

Before we go further on this topic….

- This is a complex (and interesting!) topic in CS theory
- In our few lectures, we may approach things from a simpler viewpoint than you'd get in a CS theory course

- The math and theory related to NP-complete problems starts with ***decision problems***
  - What's that? Let's use vertex cover as an example
  - What's described next applies to any optimization problems we've seen

Forms of the Vertex Cover Problem

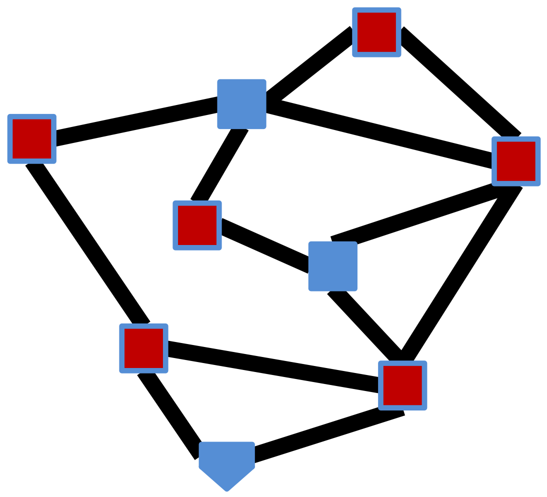
Vertex Cover: $C \subseteq V$ is a vertex cover if every edge in E has one of its endpoints in C

- **Minimum Vertex Cover Problem:** Given a graph $G = (V, E)$ find the minimum vertex cover C
  - Result is C, a set of vertices
- $\boldsymbol{k}$ **Vertex Cover Problem:** Given a graph $G = (V, E)$ and an integer k, **determine if there is a vertex cover $\boldsymbol{C}$ of size $\boldsymbol{k}$**
  - Result is True or False
  - This is the ***decision problem form*** of Vertex Cover

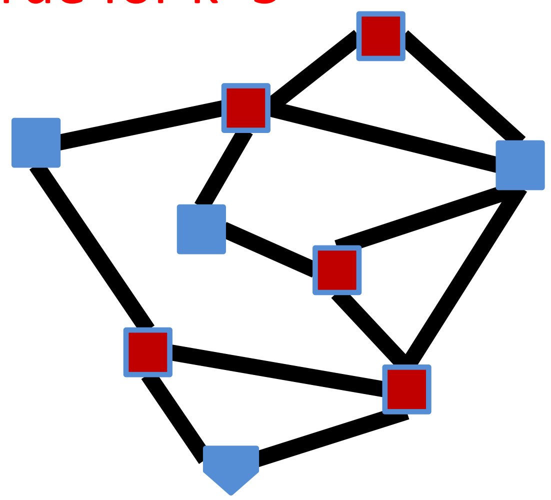
k Vertex Cover

- **k Vertex Cover Problem:** Given a graph $G = (V, E)$ and an integer k, **determine if there is a vertex cover C of size k**

True for k=6



True for k=5



Is 5 the smallest?

True for k=4?

Problem Types

- **Decision Problems:**
  - Is there a solution?
    - Result is True/False
  - E.g. Is there a vertex cover of size k?
- **Optimal Value Problems:**
  - E.g. What's the min k for k-vertex cover decision problem?
- **Search Problems:**
  - Find a solution
    - Result more complex than T/F or a k
  - E.g. Find a vertex cover of size k
- **Verification Problems:**
  - Given a potential solution for an input, is that input valid?
    - Result is True/False
  - E.g. Is **set of vertices** a vertex cover of size k**?**

If we can solve this…

…Then we can solve this,…

…and also this

Looking ahead:
We'll use this to define a problem classes P and NP

Looking ahead:
We'll use this to define a problem class called NP

Using a k-VertexCover decider to build a searcher

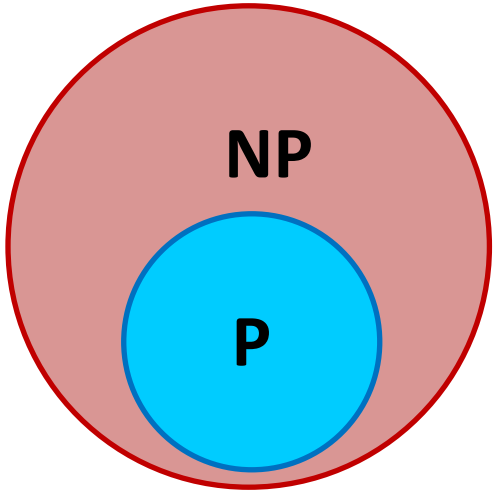
Note this is a reduction!
kVC-search $\le_p$ kVC-decider

- Set $i = k - 1$
- Remove nodes (and incident edges) one at a time
- Check if there is a vertex cover of size i (i.e. use the “decider”)
  - If so, then that removed node was part of the k vertex cover, set $i = i - 1$
  - Else, it wasn’t

Did I need this node to cover its edges to have a vertex cover of size k?

Classes of Problems: P vs NP

- P
  - Deterministic Polynomial Time
  - P is the set of problems solvable in polynomial time
    - $O(n^c)$ for some number c
- NP
  - Non-Deterministic Polynomial Time
  - NP is the set of problems ***verifiable*** in polynomial time
    - Verify a proposed solution (not find one) in $O(n^c)$ for some number c
- Open Problem: Does P=NP?
  - Certainly $P \subseteq NP$



k-Independent Set is NP

To show: Given a potential solution S, can we **verify** it in $O(n^c)$? $[n = V + E]$

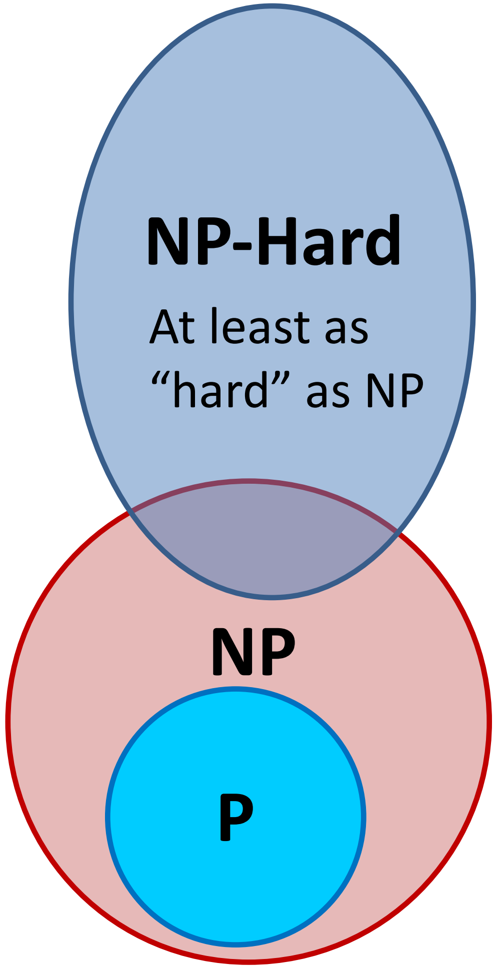
How can we verify it?

1. Check that S is of size k? Takes $O(V)$
2. Check that S is an independent set? Takes $O(V^2)$

Therefore, *k-IndSet* $\subseteq NP$

NP-Hard

- How can we try to figure out if P=NP?
- Identify problems at least as "hard" as any NP
  - If any of these "hard" problems can be solved in polynomial time, then all NP problems can be solved in polynomial time.
- Definition: NP-Hard:
  - B is NP-Hard if $\forall A \in NP, A \leq_p B$
  - $A \leq_p B$ means A reduces to B in polynomial time
  - Remember: $A \leq_p B$ implies A is not harder than B



Polynomial Reduction & Relative Hardness

$\boldsymbol{A \leq_p B}$

A problem we **don't** know how to solve

A

Then A could be solved in polynomial time

Solution for $\boldsymbol{A}$

R_A

Polynomial Reduction

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

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

A problem we **do** know how to solve

B

Using any Algorithm for $\boldsymbol{B}$

If B could be solved in polynomial time

Solution for $\boldsymbol{B}$

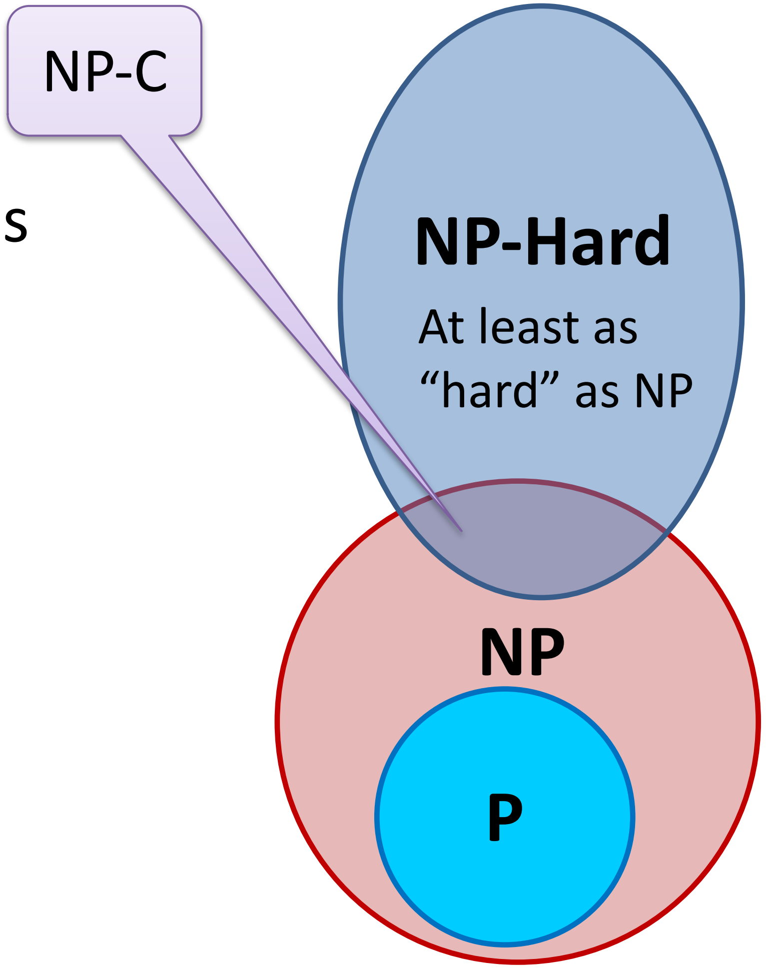
R_B

NP-Complete

NP-Complete = NP ∩ NP-Hard

- The "hardest" of all the problems in NP
- An NP-C problem is polynomial iff all NP problems are polynomial. I.e. P=NP
- If P=NP, then all NP-C problems are polynomial
- "Together they stand, together they fall"

- **How to show a problem C is NP-Complete?**
  - Show C belongs to NP
    - Show we can verify a solution in polynomial time
  - Show C is NP-Hard
    - $\forall A \in NP, A \leq_p C$ (That sounds really hard to do!)
    - Or, show a reduction from another NP-Hard problem. (But we need to have a proven NP-Hard problem.)



"Stand and Fall Together"

$\boldsymbol{A \leq_p B}$

Any NP-C problem

A

Polynomial

Reduction

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

Any other NP-C problem

B

Using any Algorithm for $\boldsymbol{B}$

Then A could be solved in polynomial time

If B could be solved in polynomial time

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

Solution for $\boldsymbol{A}$

Solution for $\boldsymbol{B}$

R_A

R_B

"Stand and Fall Together"

$\boldsymbol{A \leq_p B}$

Any NP-C problem

A

Polynomial

Reduction

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

Any other NP-C problem

B

Using any Algorithm for $\boldsymbol{B}$

If A <u>cannot</u> be solved in polynomial time

Then B cannot be solved in polynomial time

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

Solution for $\boldsymbol{A}$

R_A

Solution for $\boldsymbol{B}$

R_B

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 on ”certificates” for more info.
- **P** $\subseteq$ **NP**
- Open question: Does **P = NP**?

More Reminders and Some Consequences

- **Definition of NP-Hard and NP-Complete:**
  - If all problems A $\in$ **NP** are reducible to B , then B is *NP-Hard*
  - We say B is *NP-Complete* if:
    - B is NP-Hard
    - <u>and</u> B $\in$ **NP**

- Any NP-C must reduce to any other NP-C. Can you see why?

- If B $\leq_p$ C and B is NP-Complete, C is also NP-Complete
  - Don't see why? We'll show details in two more slides
  - As long as C $\in$ **NP**. Otherwise can only say C $\in$ **NP-hard.**

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 $\in$ **NP**

Order of the Reduction When Proving NP-Completeness

- To prove B is **NP-C**, show $A \leq_p B$ where $A \in$ **NP-Hard**
  - Why have the known NP-Hard problem "on the left"? Shouldn't it be the other way around? (No!)
- If $A \in$ **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 $A \in$ NP-C and $A \leq_p B$, then $B \in$ NP-c
  - As long as both $\in$ **NP**

What's Next?

- **Where we want to go next:**
  - Are there any NP-Hard problems? Are there any NP-C problems?

- Reminder: why do we care?
  - We know $\boldsymbol{P} \subseteq \boldsymbol{NP}$
  - 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!)

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 **<u>any</u> NP problem can be expressed this way**

Conjunctive Normal Form (CNF)

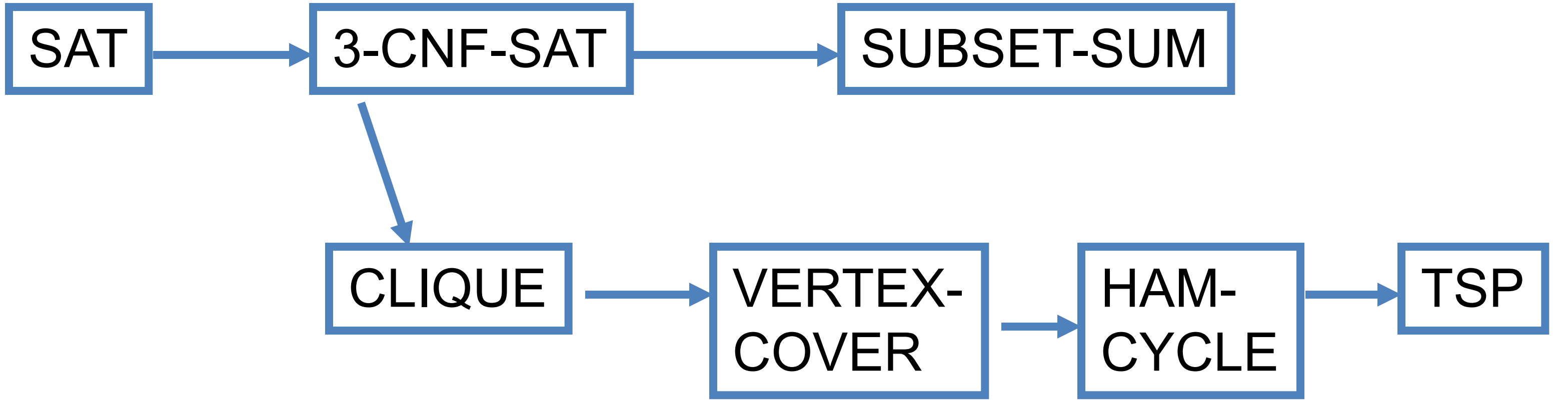
- Even if the form of the Boolean expression is simplified, the problem may be NP-Complete
  - *Literal*: an occurrence of a Boolean or its negation
  - A Boolean formula is in *conjunctive normal form*, or *CNF*, if it is an AND of clauses, each of which is an OR of literals
    - Ex: $(x_1 \vee \neg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

“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!

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

- So, find a reduction and then try to prove B $\in$ **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.