Lecture 36: NP, Completed

COSC 311 *Algorithms*, Fall 2022

Announcements

1. Final Exam: Friday, Dec. 16 9:00–12:00
   - same format as midterms
   - ~8 questions
2. Final Guide: + OH schedule
   - posted this weekend
3. Grading:
   - assignments 2, 3 this weekend
   - assignments 4, 5 next week

Previously

Two Classes of Problems:

P: decision problems solvable in polynomial time

NP: decision problems with a polynomial time verifier

A decision problem A is **NP complete** if

1. $A \in \mathrm{NP}$
2. For every $B \in \mathrm{NP}$, $B \leq_P A$.

Theorem [Cook, Levin]. Boolean Satisfiability (SAT) is NP complete.

Today

1. More NP Complete Problems
2. Coping with NP Completeness

Simpler Boolean Formulae

Terminology:

- a **literal** is a variable or its negation: $x, \bar{x}$
- a **clause** is an expression of the from
  1. $(z_1 \wedge z_2 \wedge \cdots \wedge z_k)$ (conjuctive clause) where each z_i is a literal, or
  2. $(z_1 \vee z_2 \vee \cdots \vee z_k)$ (disjunctive clause) where each z_i is a literal
- a **conjunctive normal form** (CNF) expression is an expression of the form $\boxed{C_1} \wedge \boxed{C_2} \wedge \cdots \wedge C_\ell$ where each C_i is a disjunctive clause

Observation: a CNF formula evaluates to true $\iff$ *all* clauses evaluate to true

3-SAT

Definition. A **3-CNF formula** is a Boolean formula in conjunctive normal form such that every clause contains 3 literals.

Example.

$$\varphi(w, x, y, z) = \overset{C_1}{\underbrace{(x \vee y \vee z)}_{3}} \wedge \overset{C_2}{\underbrace{(y \vee \bar{z} \vee w)}_{3}} \wedge \overset{C_3}{\underbrace{(\bar{x} \vee \bar{y} \vee \bar{w})}_{3}} \quad *$$

3-SAT:

- Input: a 3-CNF formula φ
- Output: "yes" $\iff$ φ is satisfiable

3-SAT is NP-Complete

Theorem (Tseytin 1970). Any Boolean formula φ can be efficiently (in polynomial time) transformed into a 3-CNF formula ψ such that:

1. if φ is satisfiable, then so is ψ
2. if φ is not satisfiable, then neither is ψ

3-SAT is NP-Complete

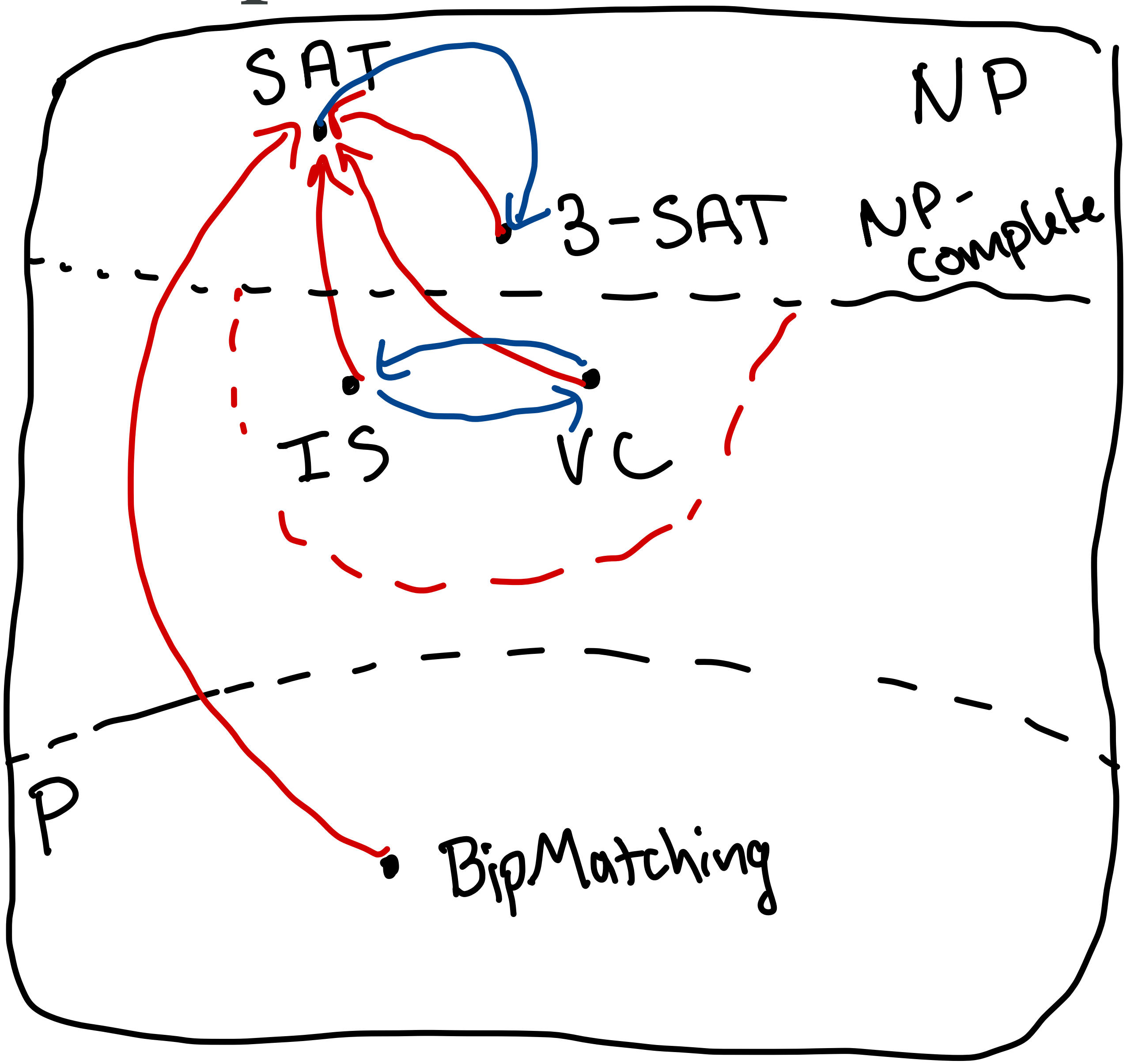
Theorem (Tseytin 1970). Any Boolean formula φ can be efficiently (in polynomial time) transformed into a 3-CNF formula ψ such that:

1. if φ is satisfiable, then so is ψ
2. if φ is not satisfiable, then neither is ψ

Consequences.

1. SAT $\leq_P$ 3-SAT
2. 3-SAT is NP complete

Relationships



Showing NP Completeness

In order to show a problem A is NP complete, show:

1. $A \in NP$
   - describe a polynomial time verifier for A
2. $B \leq_P A$ for *any* NP complete problem B
   - describe a polynomial time reduction from B to A

IS is NP Complete

Theorem. IS in NP Complete.

Question. What do we need to show?

Input: G, k

Out: "yes" $\iff$ G has indep. set of size k

- Already $IS \in NP$, gave verifier for IS in lect. 34
- To show: reduction from NP-complete problem to IS

IS is NP Complete

Theorem. IS in NP Complete.

Question. What do we need to show?

Strategy. Reduction from 3-SAT

- show 3-SAT $\leq_P$ IS

Question. How to transform a 3-CNF φ into a graph G such that solving IS on G tells us whether φ is satisfiable?

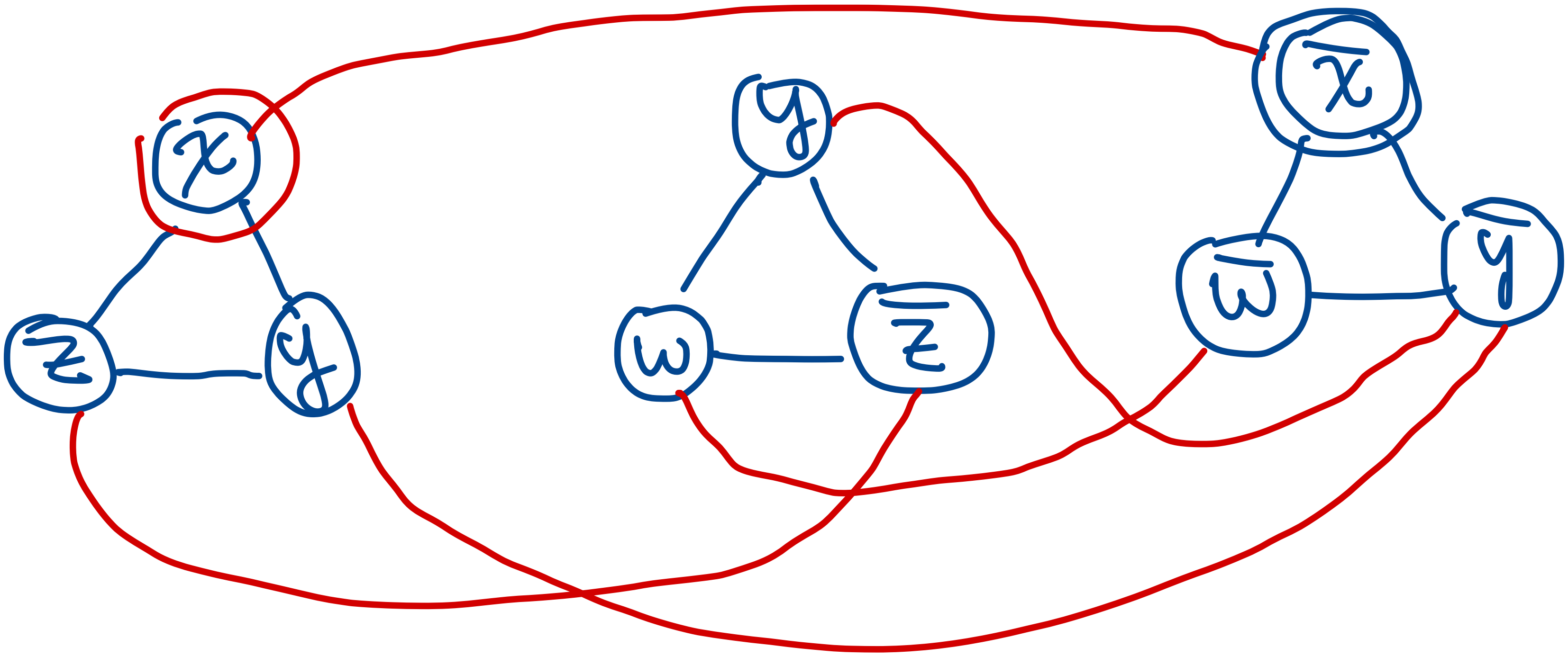
k : # Clauses

$k = 3$

Example

C_1 C_2 C_3

$$\varphi(w,x,y,z) = (x \vee y \vee z) \wedge (y \vee \bar{z} \vee w) \wedge (\bar{x} \vee \bar{y} \vee \bar{w})$$



Note: MaxIS $\leq k$

Construction, Formalized

Input:

- 3-SAT formula $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_k$
  - clause $C_i = (x_i \vee y_i \vee z_i)$ with x_i, y_i, z_i literals (variables or negated variables)

Output:

- graph $G = (V, E)$ on $n = 3k$ vertices
  - $V = \{x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_k, y_k, z_k\}$
  - edges:
    - for each i, x_i, y_i, z_i form a triangle
    - if $x_i = \neg x_j$, add edge (x_i, x_j) (sim. for other variables)

Claim 1

Suppose φ a 3-SAT formula with k clauses, G corresponding graph. If φ is satisfiable, then G has an independent set of size k.

given: satisfying assgt for φ

U = set of vertices labelled w/ "true" literals

For each triangle, if mult. vertices are in U, choose 1

Call resulting set U'

<u>Show</u>: U' is an indep. set of size k.

Claim 2

Suppose φ a 3-SAT formula with k clauses, G corresponding graph. If G has an independent set of size k, then φ is satisfiable.

U = indep set

$\hookrightarrow$ $u_1, u_2, \ldots, u_k$ literals

<u>Show</u>. set $x_1, x_2, \ldots, x_n$ variables in φ s.t. all $u_1, \ldots, u_k$ are "true" then we get satisfying assignment.

Conclusion

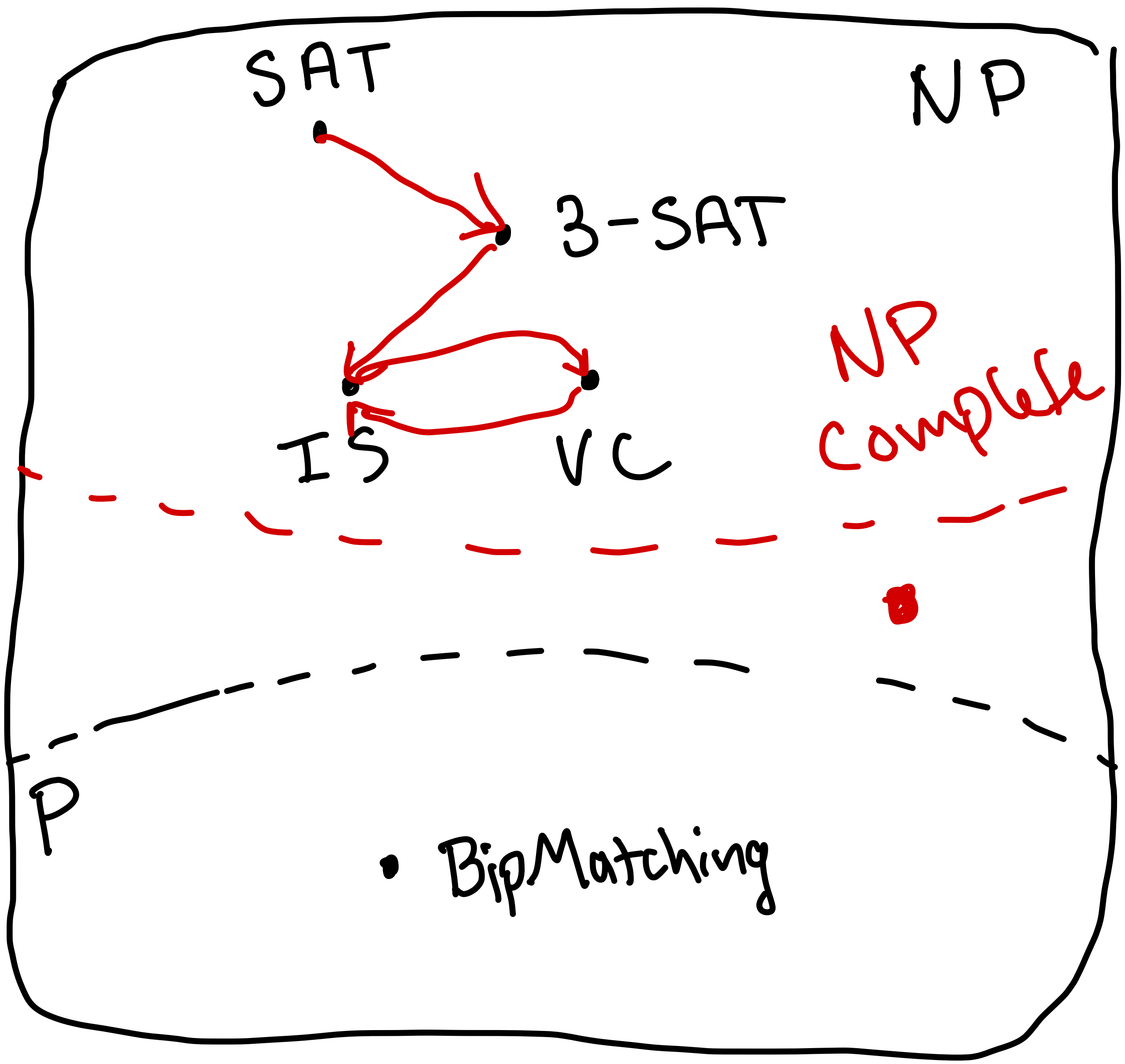
The correspondence $\varphi \to G$ is a polynomial time reduction from 3-SAT to IS.

- $\implies$ 3-SAT $\leq_P$ IS.
- $\implies$ IS is NP complete

Previously. Showed Vertex Cover (VC) satisfies IS $\leq_P$ VC

- $\implies$ VC is NP complete

More Relationships



NP Hard Problems

A problem A is **NP Hard** if $B \leq_P A$ for some NP-complete problem B.

NP Hard Problems

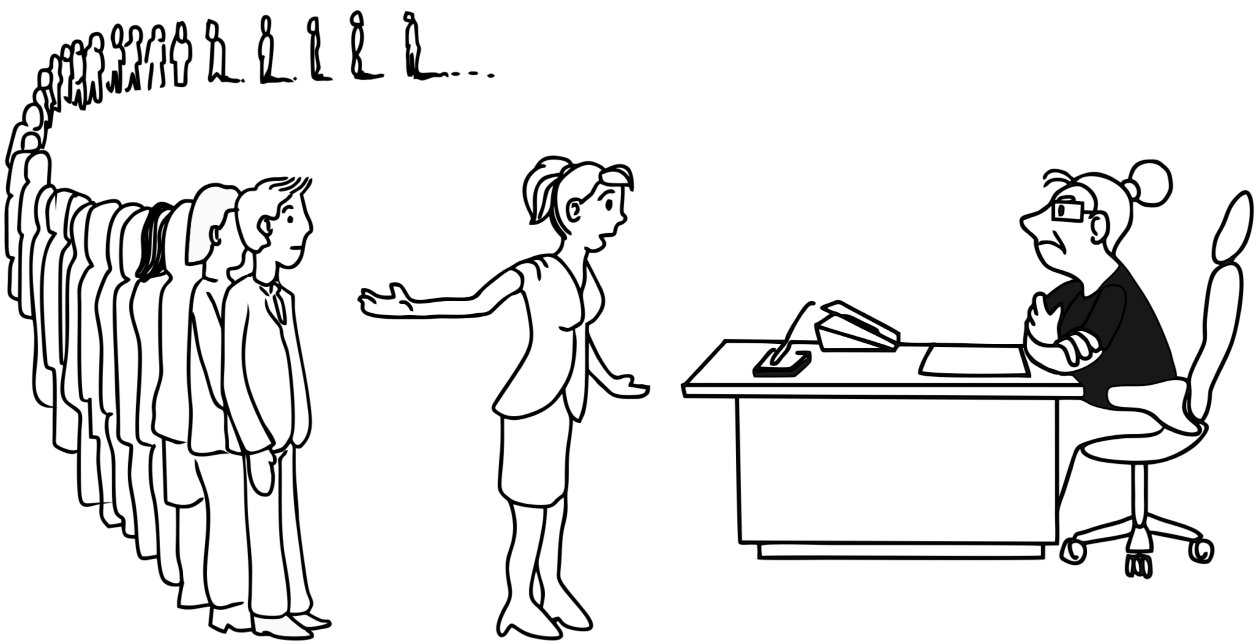
A problem A is **NP Hard** if $B \leq_P A$ for some NP-complete problem B.

Examples.

1. MaxIS and MVC
2. Traveling Salesperson (TSP)
   - *input:* weighted graph G, set U of vertices
   - *output:* minimum weight cycle containing all vertices of U
3. Subset Sum
   - *input:* numbers $w_1, w_2, \ldots, w_n$, target s
   - *output:* subest of numbers that sum to s

Coping with NP Hardness

Fact of Life. Many important practical problems are NP-Hard.



"I can't find an efficient algorithm, but neither can all these famous people."

Question. So what do we do about it?

Coping Strategies

What if we need to solve an NP hard problem?

Coping Strategies

What if we need to solve an NP hard problem?

- deal with it: exact (exponential time) algorithms

Coping Strategies

What if we need to solve an NP hard problem?

- deal with it: exact (exponential time) algorithms
- heuristics: no running time or correctness *guarantee*
  - local search
  - machine learning

Coping Strategies

What if we need to solve an NP hard problem?

- deal with it: exact (exponential time) algorithms
- heuristics: no running time or correctness *guarantee*
  - local search
  - machine learning
- approximation algorithms: efficient algorithms with guaranteed approximation to optimal

Coping Strategies

What if we need to solve an NP hard problem?

- deal with it: exact (exponential time) algorithms
- heuristics: no running time or correctness *guarantee*
  - local search
  - machine learning
- approximation algorithms: efficient algorithms with guaranteed approximation to optimal
- parameterized algorithms: classify *instances* that can be solved efficiently

Where to go from Here?

1. More algorithms!
   - parallel & distributed algorithms (COSC 273, 373)
   - computational geometry (COSC 225)
   - randomized algorithms
   - streaming and sublinear algorithms
   - approximation algorithms
2. More complexity!
   - automata/computability theory (COSC 401)
   - computational complexity
   - cryptography
   - models of computation

Thank You!