Lecture #19

CS 170

Spring 2021

Search Problems, P and NP

- Last time: Reductions $A \to B$
  - $A \to B$ means can solve A using subroutine for B
  - B "easy" (poly-time) $\Rightarrow$ A easy
  - A "hard" (no poly-time alg known) $\Rightarrow$ B hard
- Goal - try to classify problems as easy or hard
- Def: A Binary Relation is a subset $R \subseteq \{0,1\}^* \times \{0,1\}^*$ of pairs of finite bit strings, $(x,w) =$ (instance, witness)
- Def: decide(R) = given instance x, decide if $\exists w$ such that $(x,w) \in R$ (output = yes/no)
- Def: search(R) = given instance x, find a witness w such that $(x,w) \in R$ if it exists, else "no"
- decide $\to$ search, other way too

Search Problem - Example

- Def: A Binary Relation is a subset $R \subseteq \{0,1\}^* \times \{0,1\}^*$ of pairs of finite bit strings, $(x,w) = $ (instance, witness)
- Def: decide(R) = given instance x, decide if $\exists w$ such that $(x,w) \in R$ (output = yes/no)
- Def: search(R) = given instance x, find a witness w such that $(x,w) \in R$ if it exists, else "no"

- Ex: Max Flow
  - Instance: $x = (G, s, t)$, G = network, s = source, t = sink
  - Witness: w = max-flow
  - Decide(R) = yes (nothing to do)
  - Search(R) = solve, using Ford-Fulkerson

Does decide(R) always exist?

- No: Halting Problem
  - instance x = computer program, no witness
  - $R(x, \text{null}) = 1$ if x halts, else 0
  - Undecidable (no algorithm exists, CS70)
- Focus on binary relations R that are efficiently verifiable:
  - Given (x, w) = (instance, witness) there exists an algorithm $V_R(x, w) = R(x, w) \in \{0, 1\}$ with runtime $O(\text{poly}(|x|))$, where $|x| = \text{size}(x)$
- New question: given V_R, how hard is decide(R)?
  - Cost(decide(R)) at most $2^{\text{poly}(|x|)}$

    For all $w \in \{0,1\}^{\text{poly}(|x|)}$, if $V_R(x,w)=1$ output yes

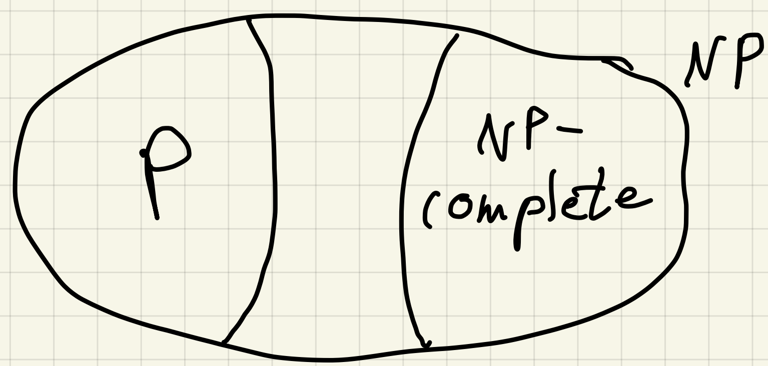
    Output no

Defining P and NP

- P = "complexity class" of all relations R such that decide(R) costs poly(|x|) (P = "polynomial")
- NP = all relations R such that given x, ∃ w of size |w| = poly(|x|), so $V_R(x, w)$ costs poly(|x|) when R(x, w) = 1 for some w
    - Ex: if $V_R(x, w)$ costs poly(|x|)
- $P \subseteq NP$
    - Does P = NP? Win \$1M Millennium Prize!

Defining NP-hard and NP-complete

- P = "complexity class" of all relations R such that decide(R) costs poly($|x|$) (P = "polynomial")
- NP = all relations R such that given x, $\exists\, w$ of size $|w| = \text{poly}(|x|)$, so $V_R(x, w)$ costs poly($|x|$) when $R(x, w) = 1$ for some w
  - Ex: if $V_R(x, w)$ costs poly($|x|$)
- Def: problem A is NP-hard if $B \to A$ for all $B \in NP$
- Def: problem A is NP-complete if A NP-hard and in NP
  - NP-complete problems exist!



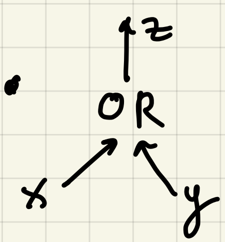
CSAT is NP-complete

- Def: CSAT is binary relation R_{CSAT} where
  $(C = \text{circuit}, w) \in R_{CSAT}$ if $C(w) = 1$

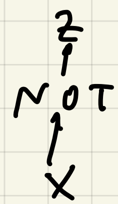
- Claim CSAT is NP-complete
  CSAT in NP because $C(w)$ efficienty
  computable (just evaluate circuit)
  CSAT NP-hard because everything in NP
  can be reduced to it:

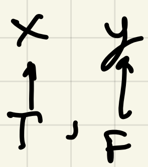
  $B \in NP \Rightarrow \exists$ efficient verifier $V_B(x_B, w_B)$
  Reduction: $x_B \rightarrow$ Preprocess to make output
  circuit for $V_B(x_B, *) \rightarrow CSAT \rightarrow$
  w that makes $C(w) = 1 \longrightarrow x_B$ solvable or not

Reducing CSAT to simpler problems: SAT

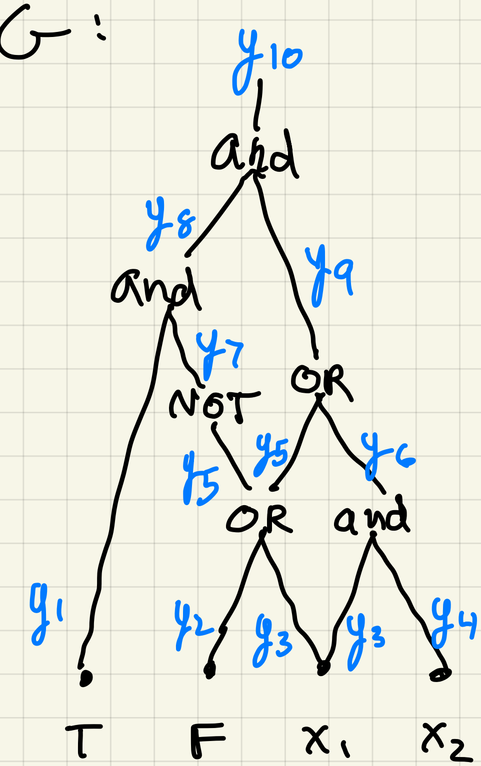
- Recall what a circuit is: DAG of gates
- Convert circuit to CNF = conjunctive normal form = and of clauses like $(x_1 \vee \bar{x}_2 \vee x_3)$
  - One variable per gate in DAG:
    - 

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

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

      becomes $(z \vee x) \wedge (\bar{z} \vee \bar{x})$
    - 

      become (x), $(\bar{y})$



SAT NP-complete

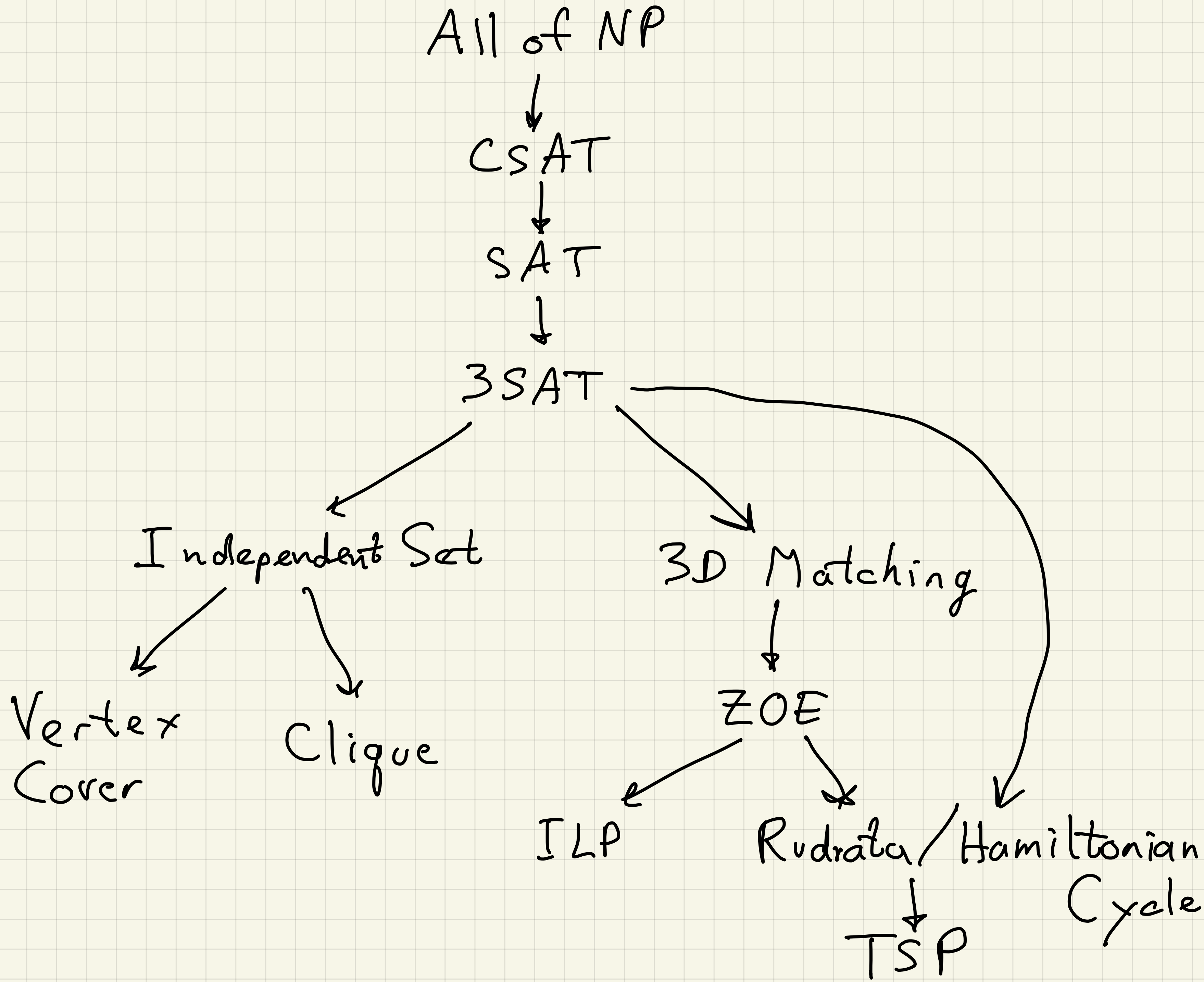
Reducing SAT to simpler case: 3SAT

- Want to show "simple" problems are NP-complete, to make them easier to use to show others are
- 3SAT: SAT with ≤ 3 variables per clause
  - Ex: $(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_4 \vee \bar{x}_5) \wedge (x_2 \vee x_6) \wedge \cdots$
- Trick to convert $(a_1 \vee a_2 \vee a_3 \cdots \vee a_k)$ to 3SAT
  - Introduce new variables $y_1, \ldots, y_{k-3}$
  - Convert to

$$(a_1 \vee a_2 \vee y_1) \wedge (\bar{y}_1 \vee a_3 \vee y_2) \wedge (\bar{y}_2 \vee a_4 \vee y_3) \ldots (\bar{y}_{i-2} \vee a_i \vee y_{i-1}) \ldots (\bar{y}_{k-3} \vee a_{k-1} \vee a_k)$$

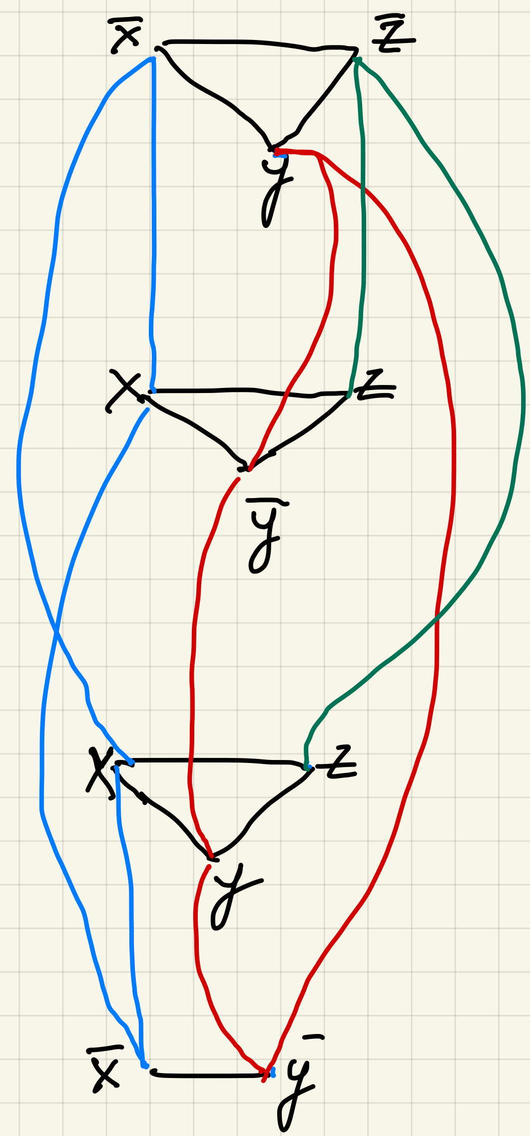
  - If all $a_i = F$, making above expression $= T \Rightarrow$ $y_1 = T \Rightarrow y_2 = T \Rightarrow \cdots \bar{y}_{k-3} = F \Rightarrow$ expression $= F$
  - If $a_i = T$, set $y_1 = y_2 = \cdots = y_{i-2} = T$, $y_{i-1} = y_i = \cdots = F$ to make all clauses $= T$

More NP-complete problems



Reducing 3SAT to Independent Set (IS)

- IS: Does graph G have $\geq g$ unconnected vertices?
- Ex: $(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$
- Transform to graph where
  - each variable is a vertex
  - each clause is a clique
    $\Rightarrow$ pick at most 1 vertex per clause for IS
  - add edge between every $(v, \bar{v})$
    $\Rightarrow$ can choose either v or $\bar{v}$ for IS



- Is there an IS of size #clauses?

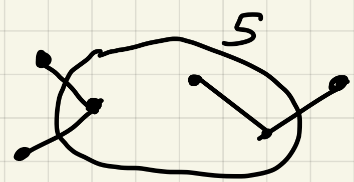
Yes $\Rightarrow$ one vertex per clause, set = T
$\Rightarrow$ each clause = T

- Is expression satisfiable?

Yes $\Rightarrow$ ≥ 1 vertex per clause = T
$\Rightarrow$ choose 1 from each clause, get IS

Reducing Independent Set (IS) to...

- Vertex Cover (VC): Subset $S \subseteq V$ that touch every edge
  - Fact: S is a VC iff $V-S$ is an IS



  - $\exists$ VC of size k iff $\exists$ IS of size $|V|-k$

- Clique (Cl): Subset $S \subseteq V$ that is fully connected
  - Fact: S is a clique in $G=(V,E)$ iff S is an IS in $G'=(V,\bar{E})$, $\bar{E}$ = all edges not in E

Did I forget to prove anything?

- Need to confirm all NP-complete problems are in NP, not just NP-hard
- Easy to confirm a witness w is correct for an instance x:
  - (3)SAT: plug values into formula, evaluate it
  - IS: given a list of vertices, confirm no edges connect them
  - VC: given a list of vertices, confirm all edges touch one of them
  - Clique: given a list of vertices confirm each pair connected