17 March 2025

NP-Hard Problems

Plan

* NP-Hardness
* Announcements
* NP-Hard Problems
    * 3SAT
    * Independent Set

Complexity Theory

Categorize problems based on how EASY / HARD they are to solve.

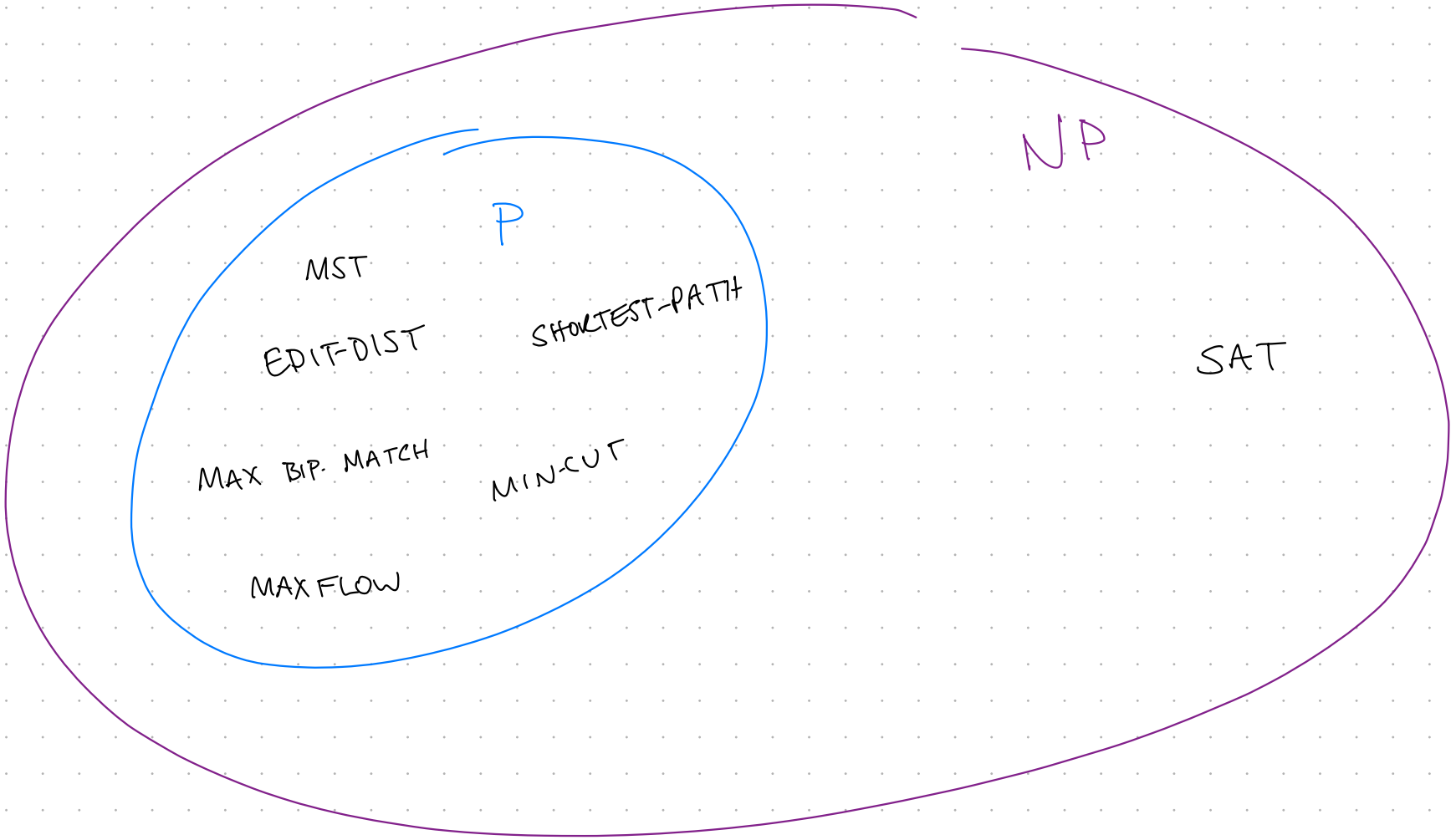
Complexity Theory

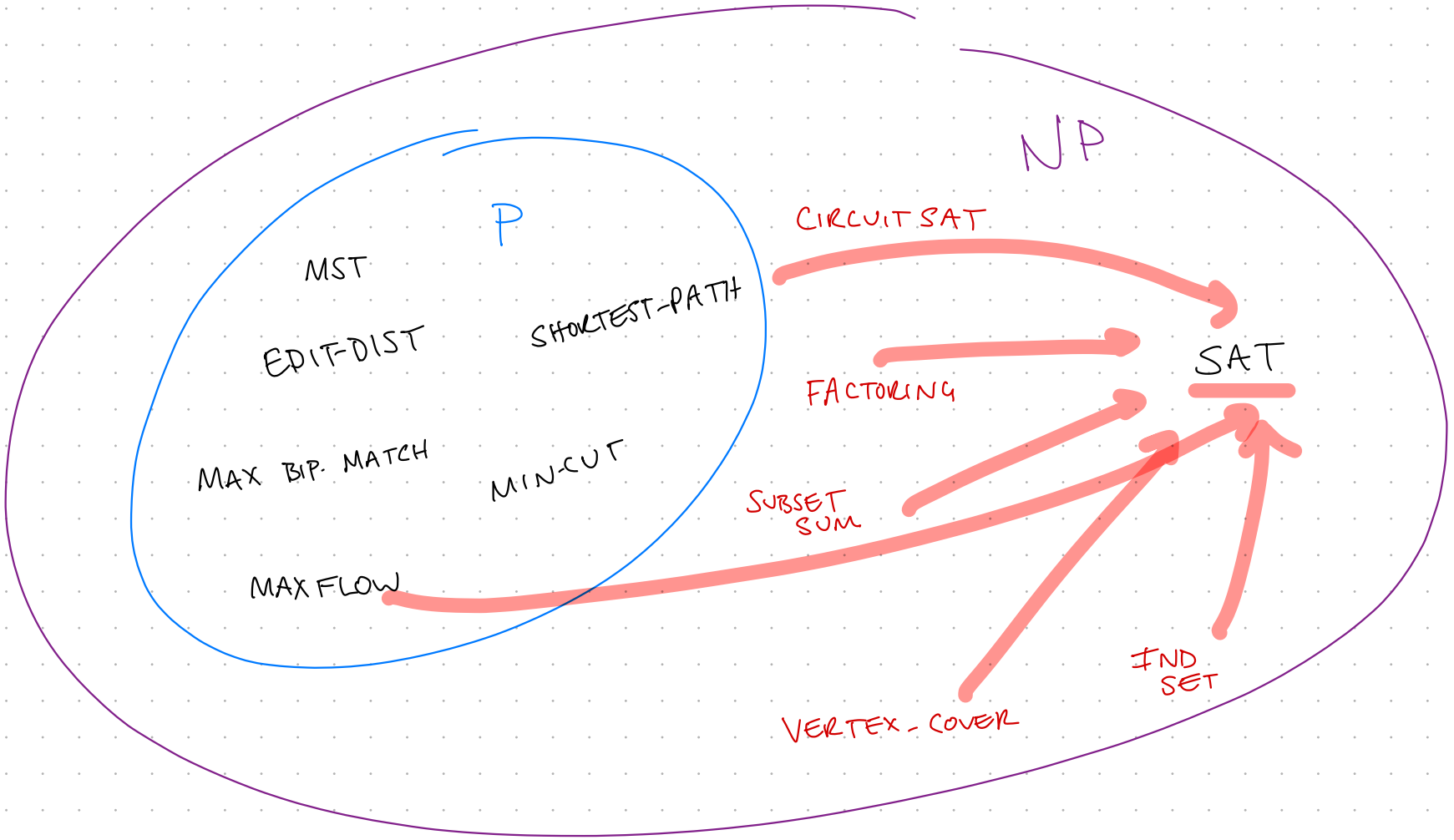
Categorize problems based on how EASY / HARD they are to solve.

P ≡ Problems that can be solved in polynomial time.

NP ≡ Problems that can be verified in polynomial time.

NP-Hard ≡ Q is NP-Hard if every problem in NP reduces to Q.


NP
P
MST
EDIT-DIST
SHORTEST-PATH
SAT
MAX BIP. MATCH
MIN-CUT
MAX FLOW



Theorem. Every problem $Q \in NP$ reduces to SAT!

If any efficiently-verifiable problem is hard to solve, then SAT is hard to solve.

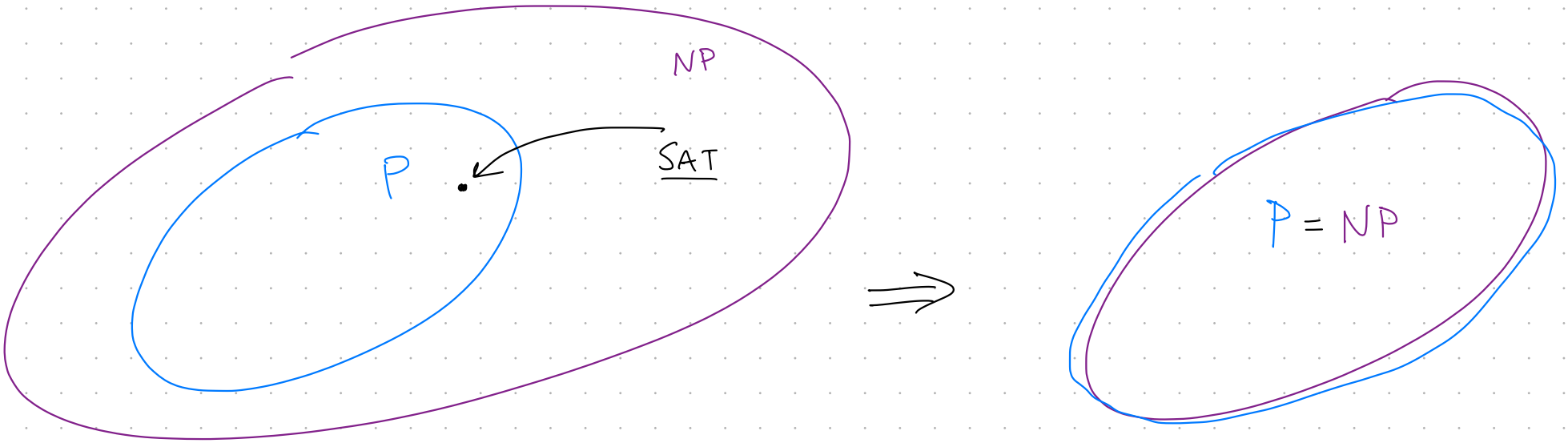
SAT. Given a boolean formula ϕ, does there exist a satisfying assignment?

$\exists\, \vec{a} \in \{0,1\}^n$ s.t. $\phi(\vec{a}) = 1$?

$$\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_5 \vee x_1 \vee x_6) \wedge (x_3 \vee \neg x_6 \vee \neg x_5)$$

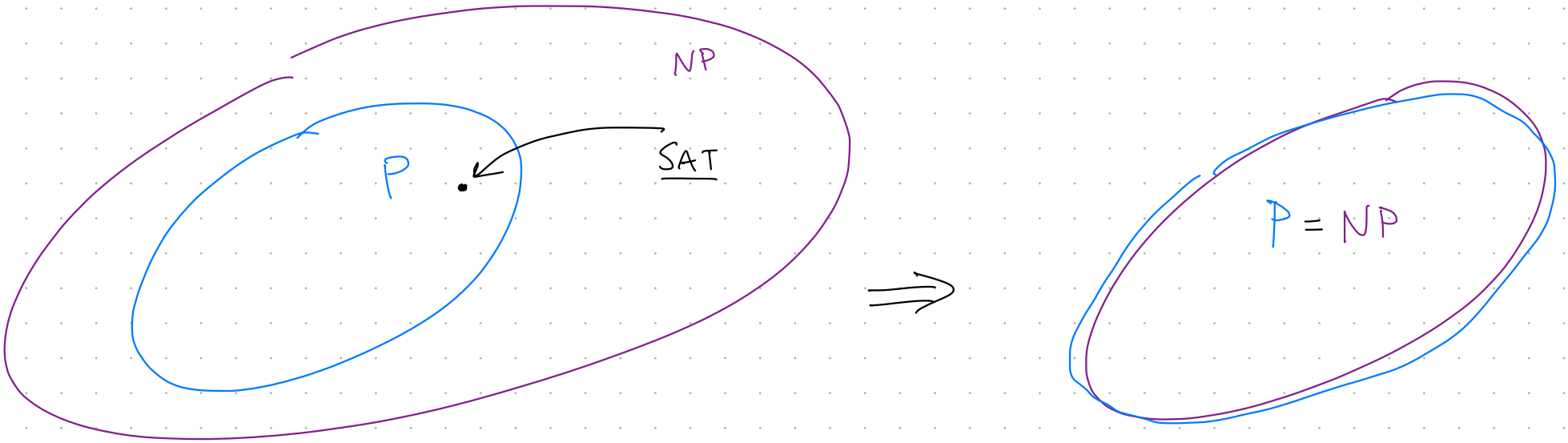
SAT captures P vs. NP.

Theorem. If there is a polynomial-time algorithm for SAT, then P = NP.



SAT captures P vs. NP.

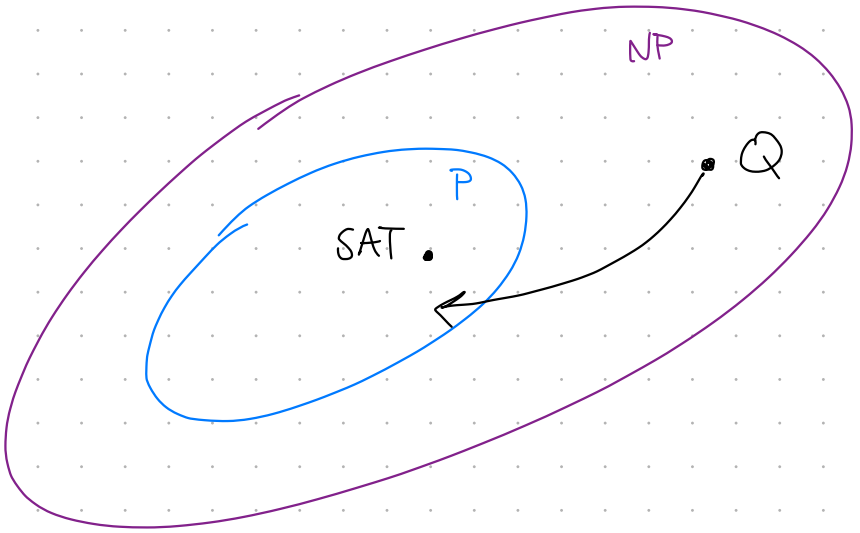
Theorem. If there is a polynomial-time algorithm for ~~SAT~~, any NP-Hard problem then P = NP.



Theorem. If there is a polynomial-time algorithm for SAT, then P = NP.

Proof. Suppose A is a poly-time algo that solves SAT.

* Consider any problem $Q \in NP$.
* We show that $Q \in P \implies NP \subseteq P \implies P = NP$.

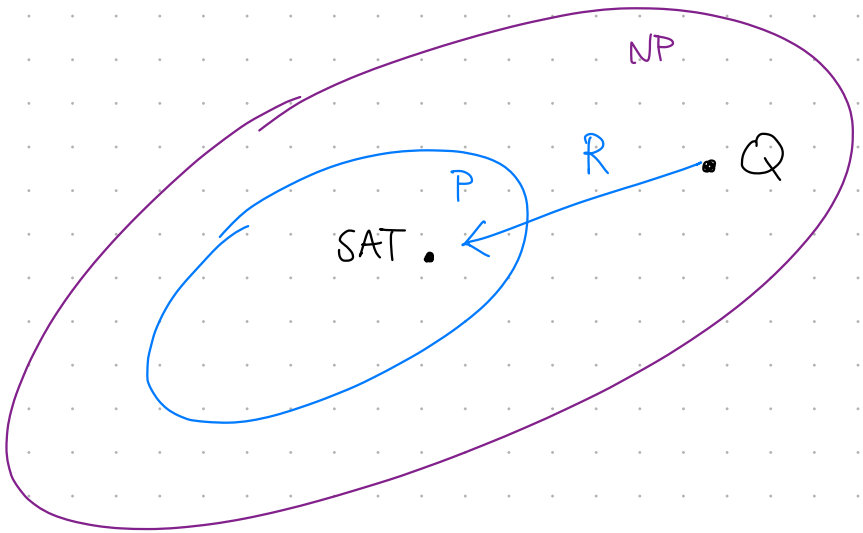


Theorem. If there is a polynomial-time algorithm for SAT, then P = NP.

Proof. Suppose A is a poly-time algo that solves SAT.

* Consider any problem $Q \in NP$.

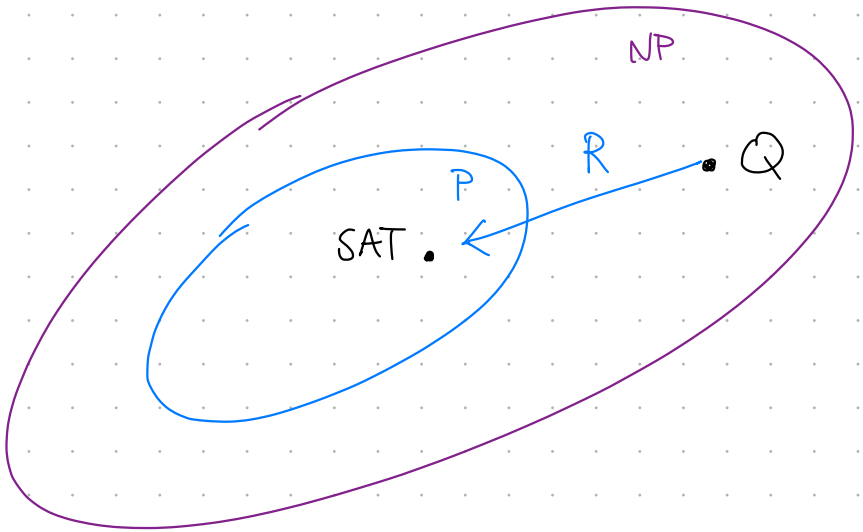
* SAT NP-Hard $\Rightarrow$ $\exists$ poly-time reduction R from Q to SAT



Theorem. If there is a polynomial-time algorithm for SAT, then P = NP.

Proof. Suppose A is a poly-time algo that solves SAT.

* Consider any problem $Q \in NP$.
* SAT NP-Hard $\Rightarrow$ $\exists$ poly-time reduction R from Q to SAT



Algo Q. On input x to Q,

Run R on x

Run A on R(x)

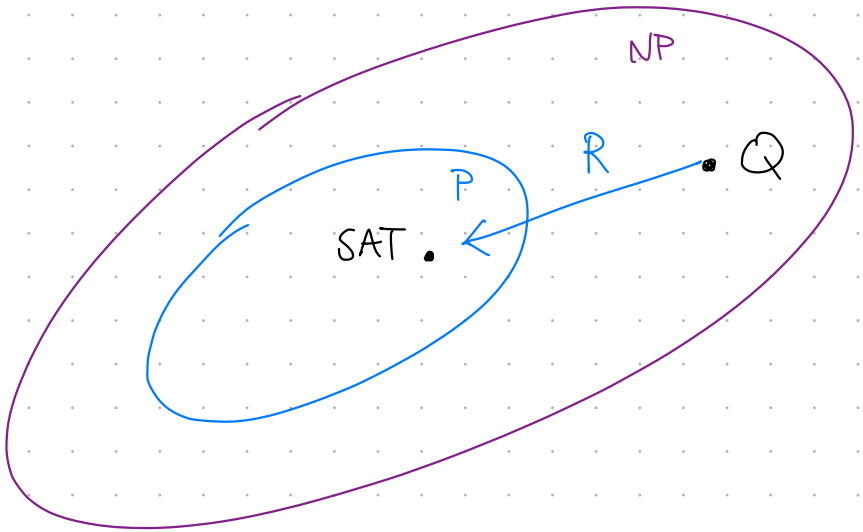
Return A(R(x))

Theorem. If there is a polynomial-time algorithm for SAT, then P = NP.

Proof. Suppose A is a poly-time algo that solves SAT.

* Consider any problem $Q \in NP$.

* SAT NP-Hard $\Rightarrow$ $\exists$ poly-time reduction R from Q to SAT



Algo Q. On input x to Q,

Run R on x

Run A on $R(x)$

Return $A(R(x))$

Claim. Algo Q solves Q in poly-time! $\Rightarrow Q \in P$.

Most Computer Scientists Agree

$$P \neq NP$$

By previous theorem (contrapositive)

If $P \neq NP$, then SAT does NOT have a polynomial-time algorithm.

Today Leverage SAT to show other problems must be HARD

To show problem Q is NP-HARD, show

SAT reduces to Q in poly-time

$$SAT \leq_p Q$$

Today Leverage SAT to show other problems must be HARD

To show problem Q is NP-HARD, show

SAT reduces to Q in poly-time

$$SAT \leq_p Q$$

Conclude.

No polynomial-time algorithm for Q (unless P=NP).

Announcements

* HW5 ongoing
* HW6
  - Released Wednesday 19 March
  - Solutions Wednesday 26 March
* Prelim 2. 27 March, 7:30 - 9p
  - ↳ Location: Same as Prelim 1.
  - ↳ Coverage: through HW6
    - Stable Matching
    - Max Flow / Min Cut
    - Mathematical Algos
    - NP-Completeness
  - ↳ Practice Exam released Wednesday.

SAT. Given a boolean formula ϕ, does there exist a satisfying assignment?

$\exists\, \vec{a} \in \{0,1\}^n$ s.t. $\phi(\vec{a}) = 1$?

$$\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_5 \vee x_1 \vee x_6) \wedge (x_3 \vee \neg x_6 \vee \neg x_5)$$

SAT. Given a boolean formula ϕ, does there exist a satisfying assignment?

$$\exists\, \vec{a} \in \{0,1\}^n \text{ s.t. } \phi(\vec{a}) = 1 \;?$$

$$\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_5 \vee x_1 \vee x_6) \wedge (x_3 \vee \neg x_6 \vee \neg x_5)$$

k-CNF formula ϕ

→ Conjunction (AND) of m clauses

→ Disjunction (OR) of k literals

→ one of n variables or negation

3SAT Given a 3-CNF formula ϕ, does $\exists \vec{a}$ s.t. $\phi(\vec{a}) = 1$?

Every clause has ≤ 3 literals

3SAT Given a 3-CNF formula ϕ, does $\exists \vec{a}$ s.t. $\phi(\vec{a}) = 1$?

Every clause has ≤ 3 literals

Theorem (Cook-Levin) 3SAT is NP-Hard.

$3SAT = \{ \phi : \phi$ is a satisfiable 3CNF formula $\}$

↳ Represent YES/NO Decision Problems as subsets of valid inputs

(i.e. the set of YES instances)

$3SAT = \{ \phi : \phi \text{ is a satisfiable 3CNF formula} \}$

↳ Represent YES/NO Decision Problems as subsets of valid inputs

(i.e. the set of YES instances)

Reduction maps YES instances $\longrightarrow$ YES instances

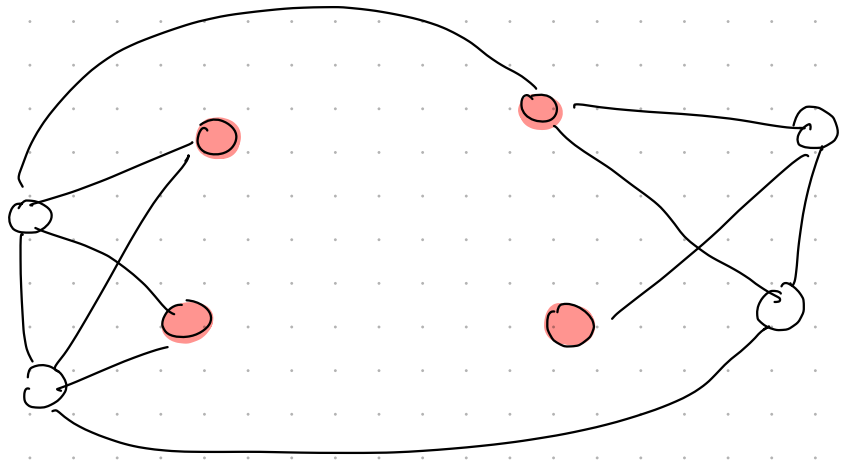
NO instances $\longrightarrow$ NO instances

Maximum Independent Set

Given a graph $G = (V, E)$,

$S \subseteq V$ is an independent set if for all

$$u \in S, v \in S \Rightarrow (u, v) \notin E$$



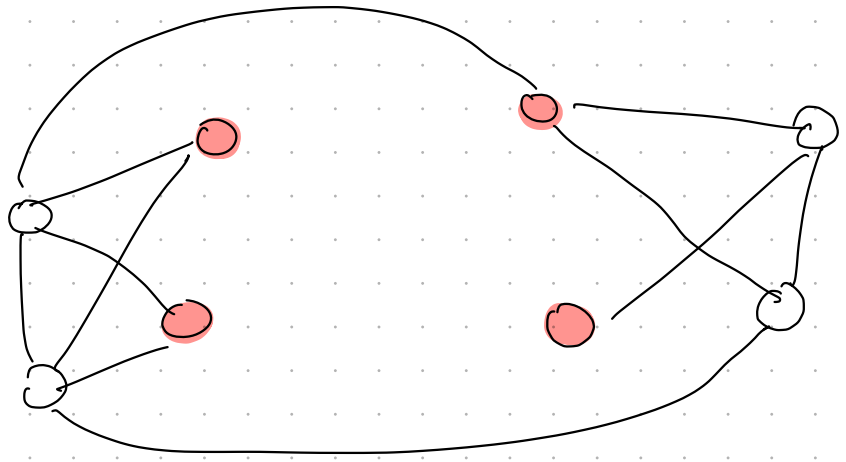
$$\text{INDSET} = \{ \langle G, k \rangle : \begin{array}{l} G \text{ contains an independent set} \\ \text{of cardinality} \geq k \end{array} \}$$

Maximum Independent Set

Given a graph $G = (V, E)$,

$S \subseteq V$ is an independent set if for all

$$u \in S, v \in S \Rightarrow (u, v) \notin E$$



$$\text{IndSet} = \{ \langle G, k \rangle : \begin{array}{l} G \text{ contains an independent set} \\ \text{of cardinality } \geq k \end{array} \}$$

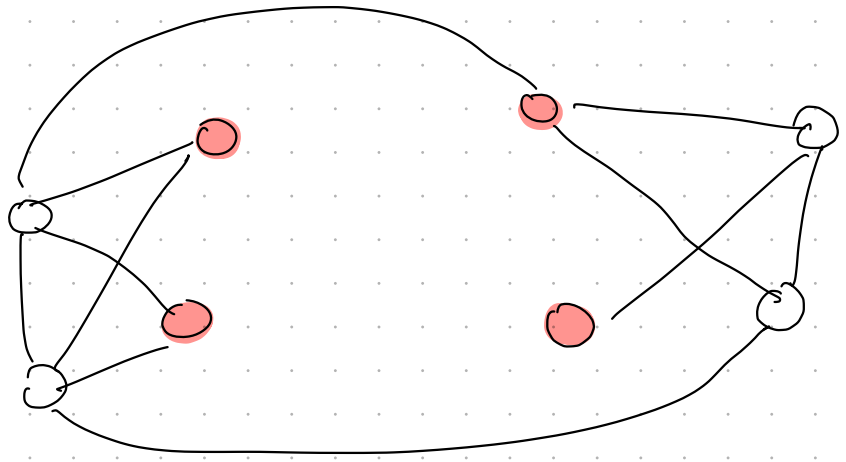
Fact. $\text{IndSet} \in NP$.

Maximum Independent Set

Given a graph $G = (V, E)$,

$S \subseteq V$ is an independent set if for all

$$u \in S, v \in S \Rightarrow (u, v) \notin E$$



$\exists S \in NP \Rightarrow$ poly-time verifier

Verifier G, k, S

Checks $|S| \geq k$.

$\forall u, v \in S$ check $(u, v) \notin E$.

$$\text{IndSet} = \{ \langle G, k \rangle : G \text{ contains an independent set of cardinality } \geq k \}$$

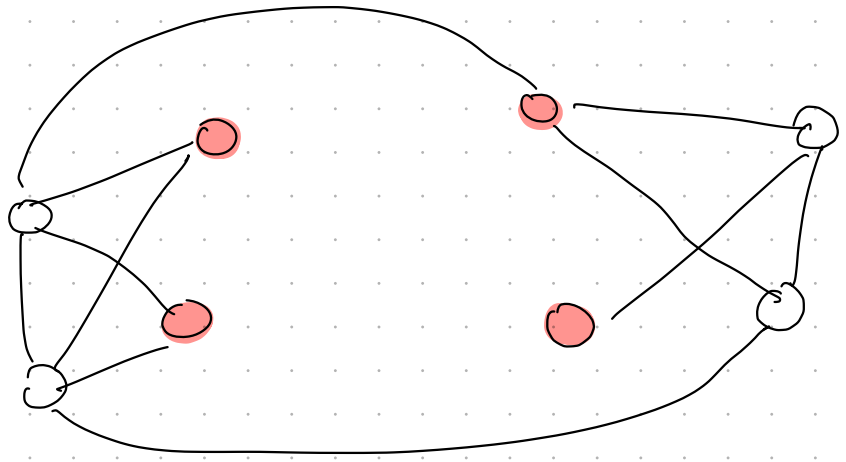
Fact. $\text{IndSet} \in NP$.

Maximum Independent Set

Given a graph $G = (V, E)$,

$S \subseteq V$ is an independent set if for all

$$u \in S, v \in S \Rightarrow (u, v) \notin E$$



$$\text{INDSET} = \{ \langle G, k \rangle : \begin{array}{l} G \text{ contains an independent set} \\ \text{of cardinality} \geq k \end{array} \}$$

Theorem. $3SAT \leq_p \text{INDSET}$.

Corollary. INDSET is NP-HARD.

Reduction from 3SAT to INDSET.

Design polynomial-time algorithm:

* Given ϕ
* Returns $\langle G, k \rangle$

s.t.

ϕ satisfiable $\iff$ G has IS of cardinality $\geq k$.

Reduction from 3SAT to INDSET.

Design polynomial-time algorithm:

* Given ϕ
* Returns $\langle G, k \rangle$

s.t.

ϕ satisfiable $\iff$ G has IS of cardinality $\geq k$.

$\phi \in$ 3SAT $\iff$ $\langle G, k \rangle \in$ INDSET

Reduction from 3SAT to INDSET.

Design polynomial-time algorithm:

* Given ϕ
* Returns $\langle G, k \rangle$

s.t.

ϕ satisfiable $\iff$ G has IS of cardinality $\geq k$.

$\phi \in$ 3SAT $\iff$ $\langle G, k \rangle \in$ INDSET

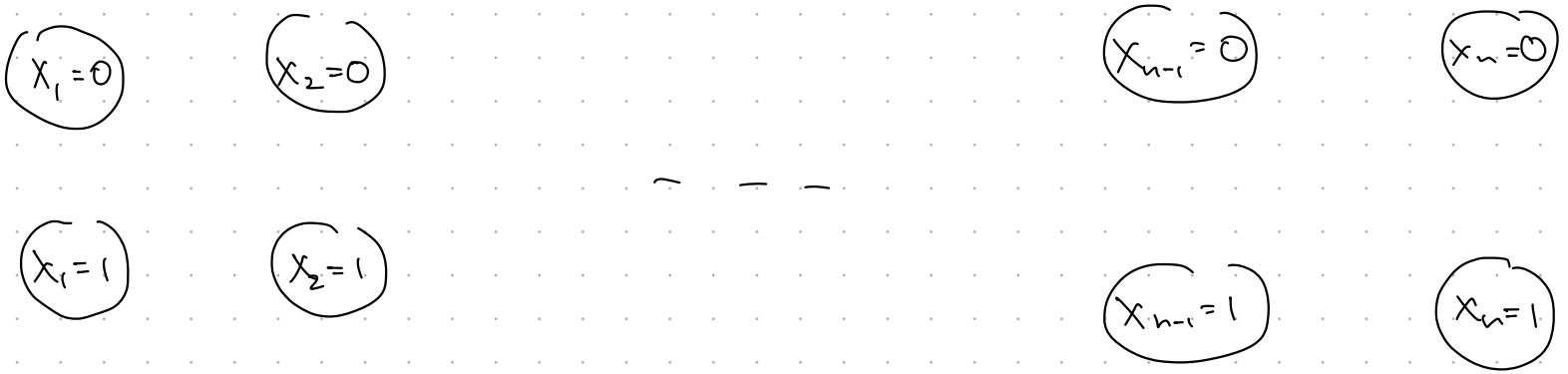
Want: produce a graph where

large IS "selects" assignment that satisfies all clauses.

Variable Assignment Gadgets

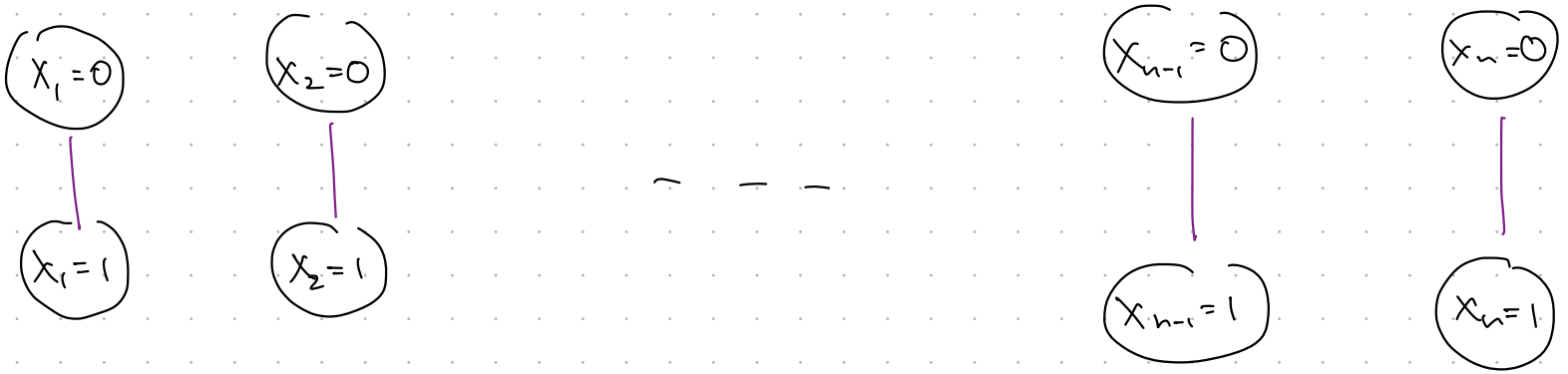
For each $i = 1 \cdots n$, $x_i = a_i \in \{0, 1\}$

assigned a single value.



Variable Assignment Gadgets

For each $i = 1, \dots, n$, $x_i = a_i \in \{0, 1\}$ assigned a single value.



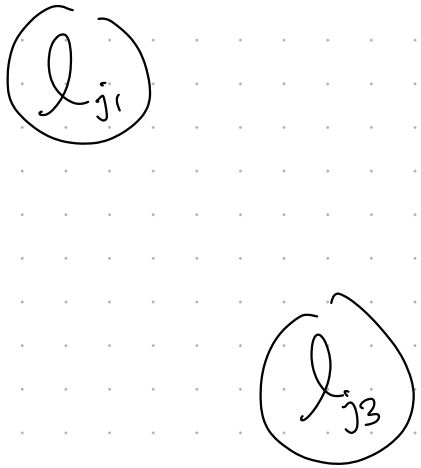
$\forall i = 1, \dots, n$

- Create two vertices $x_{i0} = (x_i = 0)$ $\in V$.
  $x_{i1} = (x_i = 1)$
- Add $(v_{i0}, v_{i1}) \in E$

Clause Satisfaction Gadgets

For each $j = 1, \ldots, m$ some literal in C_j evaluates to 1 (else, unsatisfiable)

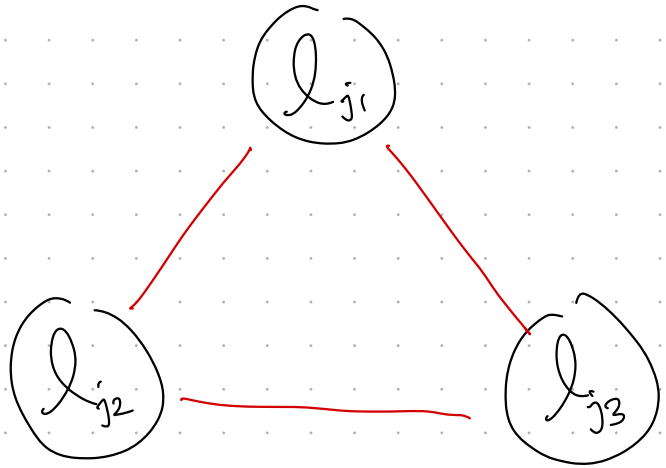
$$C_j = (\ell_{j1} \vee \ell_{j2} \vee \ell_{j3})$$



Clause Satisfaction Gadgets

For each $j = 1, \dots, m$ some literal in C_j evaluates to 1 (else, unsatisfiable)

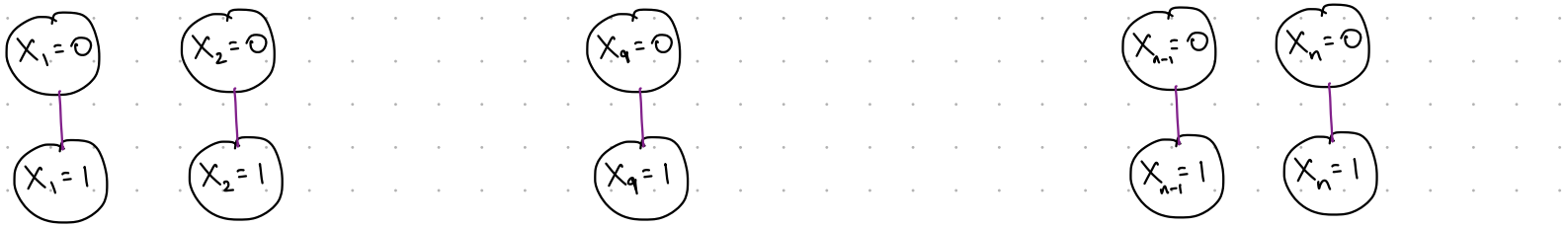
$$C_j = (\ell_{j1} \vee \ell_{j2} \vee \ell_{j3})$$

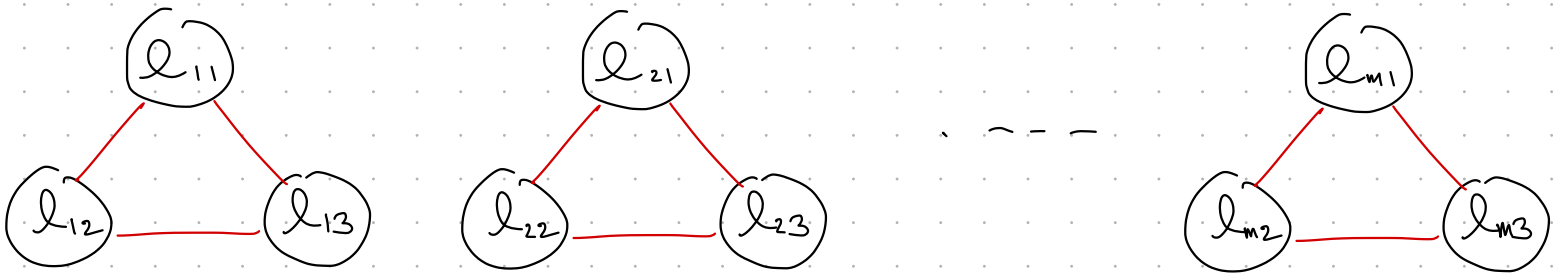


$\forall j = 1 \dots m$

- Create 3 vertices $\ell_{j1}, \ell_{j2}, \ell_{j3} \in V$
- Add edges between each $(\ell_{j1}, \ell_{j2}), (\ell_{j2}, \ell_{j3}), (\ell_{j3}, \ell_{j1}) \in E$

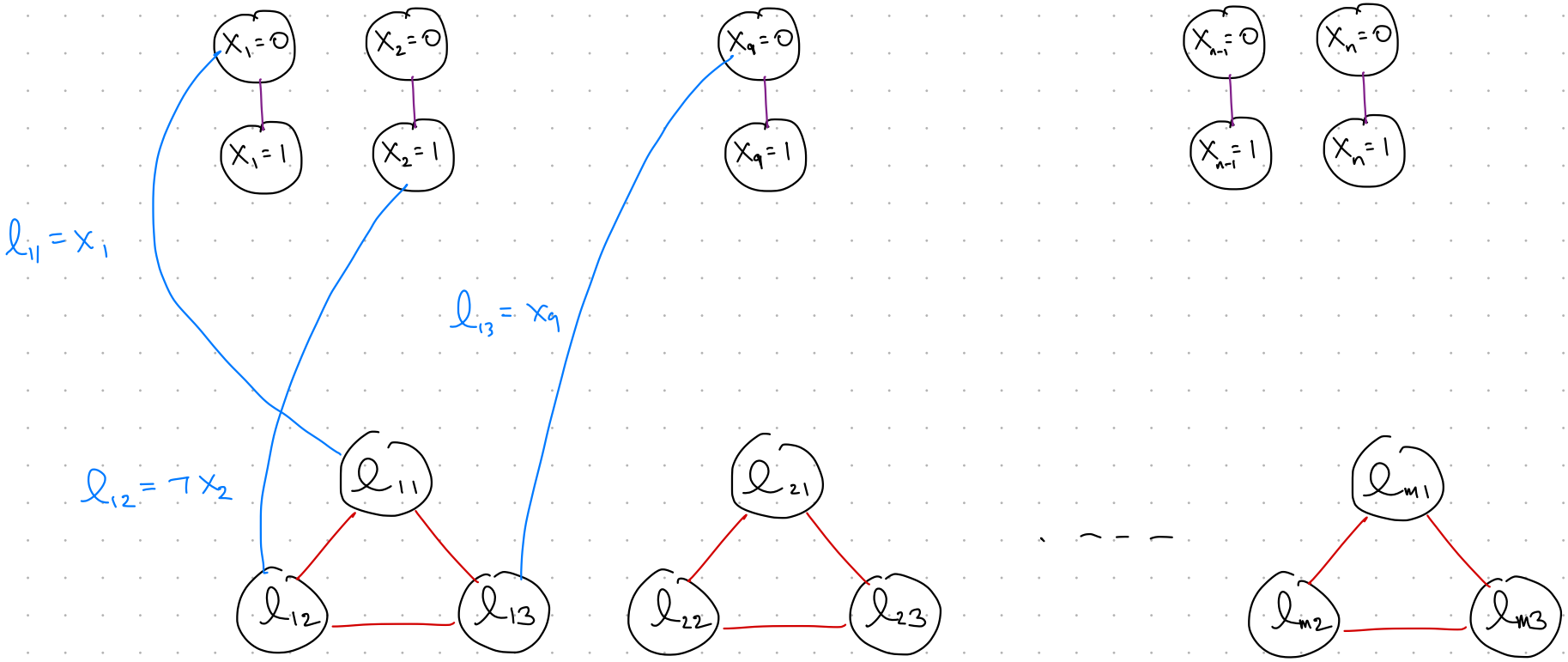
Clause / Assignment Consistency





$$\phi = (x_1 \vee \neg x_2 \vee x_q) \wedge (\neg x_q \vee x_{n-1} \vee \neg x_n) \wedge \cdots$$

Clause / Assignment Consistency

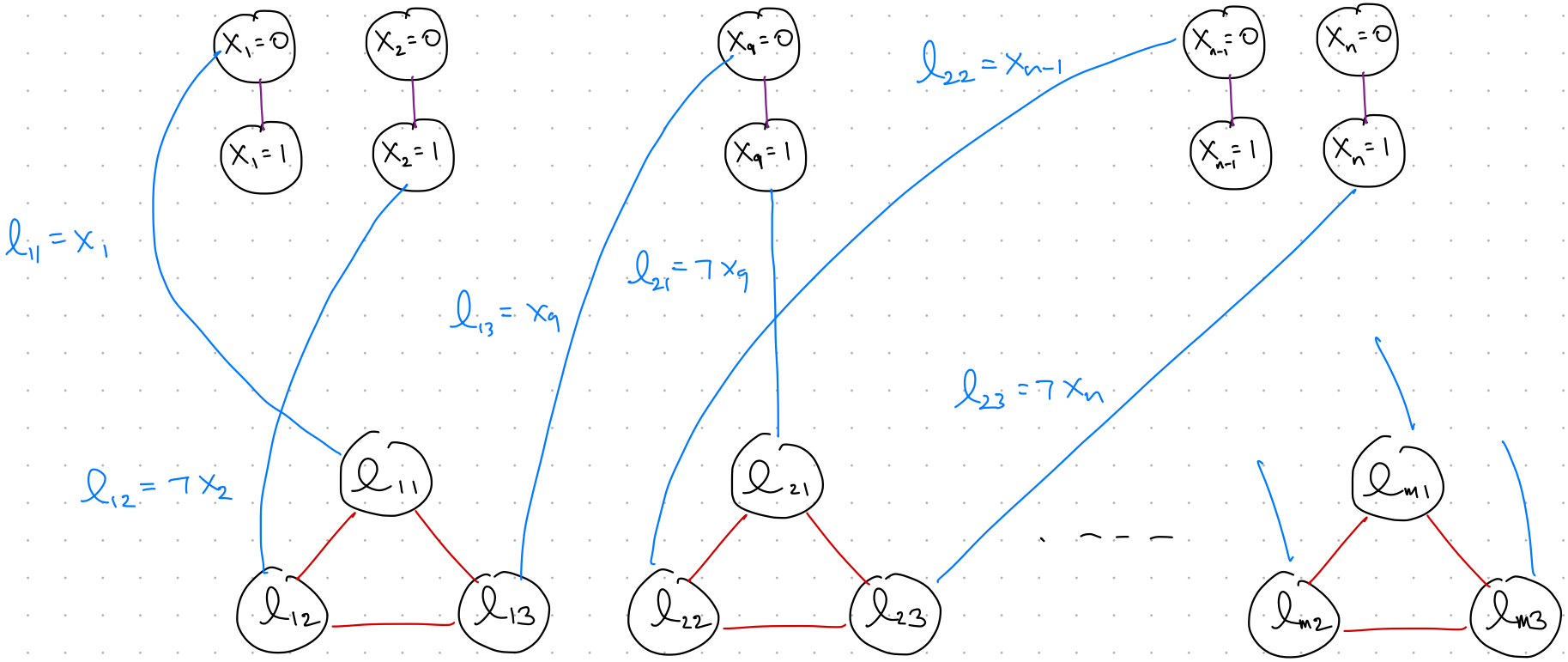


$$\phi = (x_1 \vee \neg x_2 \vee x_q) \wedge (\neg x_q \vee x_{n-1} \vee \neg x_n) \wedge \cdots$$

If literal $\ell_{j+} = x_i$, add edge $(\ell_{j+}, x_{i0}) \in E$

If literal $\ell_{j+} = \neg x_i$, add edge $(\ell_{j+}, x_{i1}) \in E$

Clause / Assignment Consistency



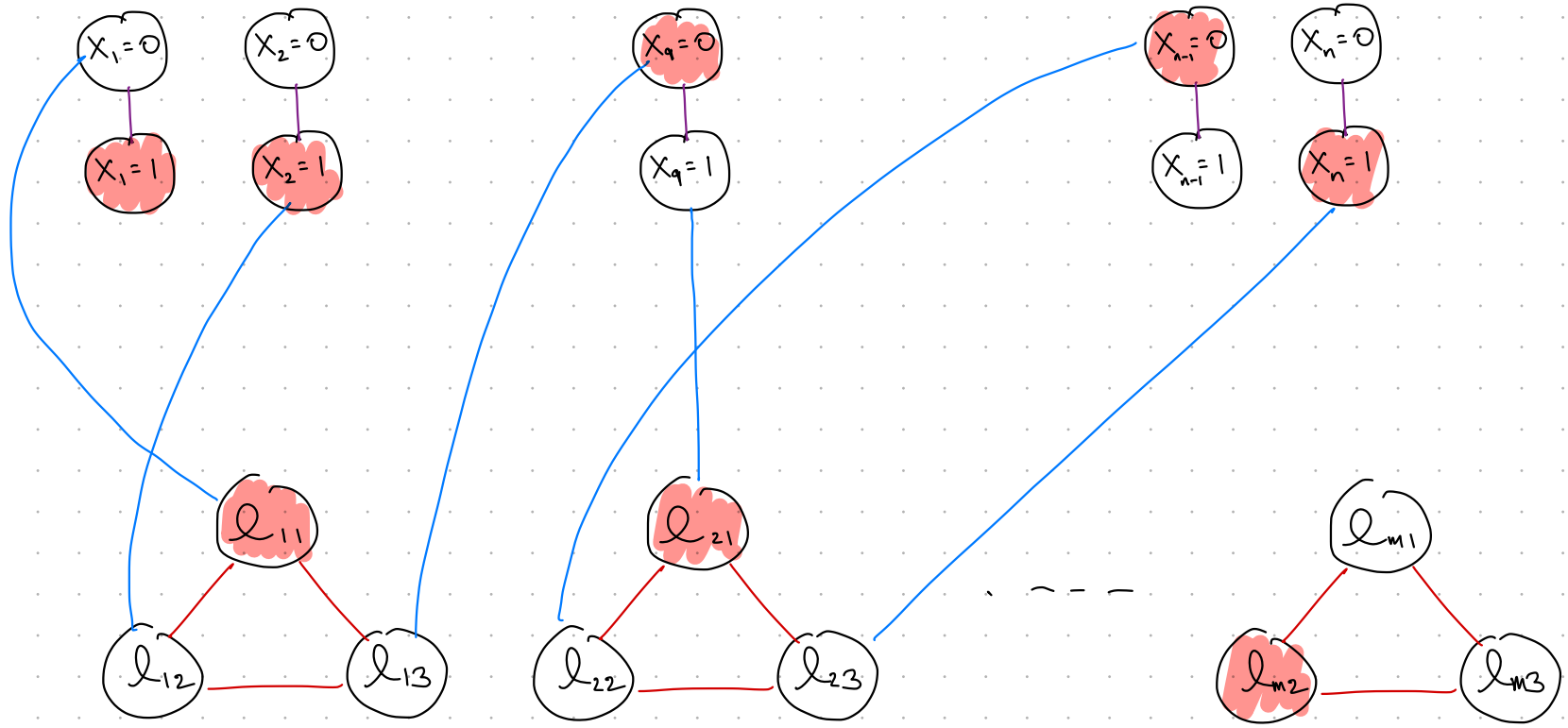
$$\phi = (x_1 \vee \neg x_2 \vee x_q) \wedge (\neg x_q \vee x_{n-1} \vee \neg x_n) \wedge \cdots$$

For each clause C_j, for each literal l_{jt} for $t=1,2,3$

If literal $l_{jt} = x_i$, add edge $(l_{jt}, x_{i0}) \in E$

If literal $l_{jt} = \neg x_i$, add edge $(l_{jt}, x_{i1}) \in E$

Clause / Assignment Consistency



$$\phi = (x_1 \vee \neg x_2 \vee x_q) \wedge (\neg x_q \vee x_{n-1} \vee \neg x_n) \wedge \cdots$$

Claim. ϕ is satisfiable $\iff$ G has an IS of cardinality $\geq n+m$.

ϕ satisfiable $\Rightarrow$ IS of size $n + m$.

* Consider a satisfying assignment $\vec{a} \in \{0,1\}^n$.
* For each $i = 1 \dots n$, add x_{ib} to S for $a_i = b$.
* For each $j = 1 \dots m$, add literal ℓ_{jt} to S for some $\ell_{jt} = 1$ under $\vec{a}$.

Claim 0. $|S| = n + m$

Claim 1. S is an independent set.

ϕ satisfiable $\Rightarrow$ IS of size $n+m$.

* Consider a satisfying assignment $\vec{a} \in \{0,1\}^n$.
* For each $i = 1 \dots n$, add x_{ib} to S for $a_i = b$.
* For each $j = 1 \dots m$, add literal ℓ_{jt} to S for some $\ell_{jt} = 1$ under $\vec{a}$.

Claim ①. $|S| = n+m$

Claim 1. S is an independent set.

* only select 1 vertex per variable gadget
  1 vertex per clause gadget.
* $(\ell_{jt}, x_{ib}) \in E$ only if setting $x_i = b$ causes $\ell_{jt} = 0$.
  $\Rightarrow (\ell_{jt}, x_{ib}) \notin E$.

ϕ unsatisfiable $\Longrightarrow$ Every IS has cardinality $< n+m$

---

* Every IS has $\leq n$ vertices from variable gadgets
  & $\leq m$ vertices from clause gadgets

* Every assignment $\vec{a} \in \{0,1\}^n$ results in some clause C_j with every literal $\ell_{j1} = \ell_{j2} = \ell_{j3} = 0$.

ϕ unsatisfiable $\Rightarrow$ Every IS has cardinality $< n+m$

---

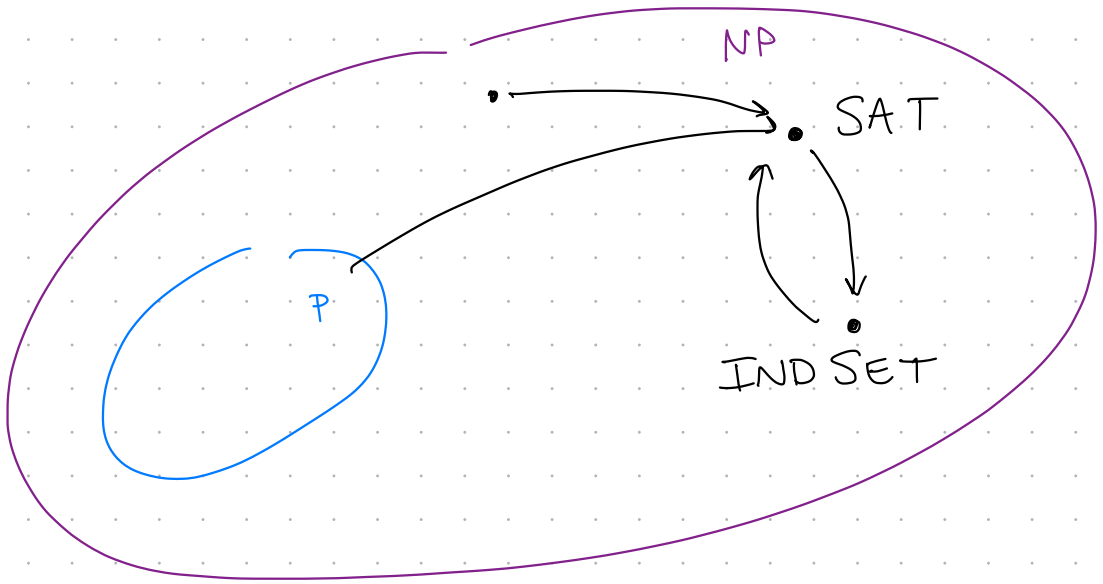
* Every IS has $\leq n$ vertices from variable gadgets
  & $\leq m$ vertices from clause gadgets

* Every assignment $\vec{a} \in \{0,1\}^n$ results in some clause C_j with every literal $\ell_{j1} = \ell_{j2} = \ell_{j3} = 0$.

* Consider the set $S_{\vec{a}}$ of n vertices associated w/ $\vec{x} = \vec{a}$.
  ↳ For each vertex $\ell_{j1}, \ell_{j2}, \ell_{j3}$,
  there is some neighboring vertex $x_{ib} \in S_{\vec{a}}$

Thus, IS cannot have 1 vertex per variable
and 1 vertex per clause.

$\Rightarrow$ IS $< n+m$.



INDSET is NP-Hard

$\Rightarrow$ IND SET $\in$ NP-Complete.

IndSet $\in$ NP.