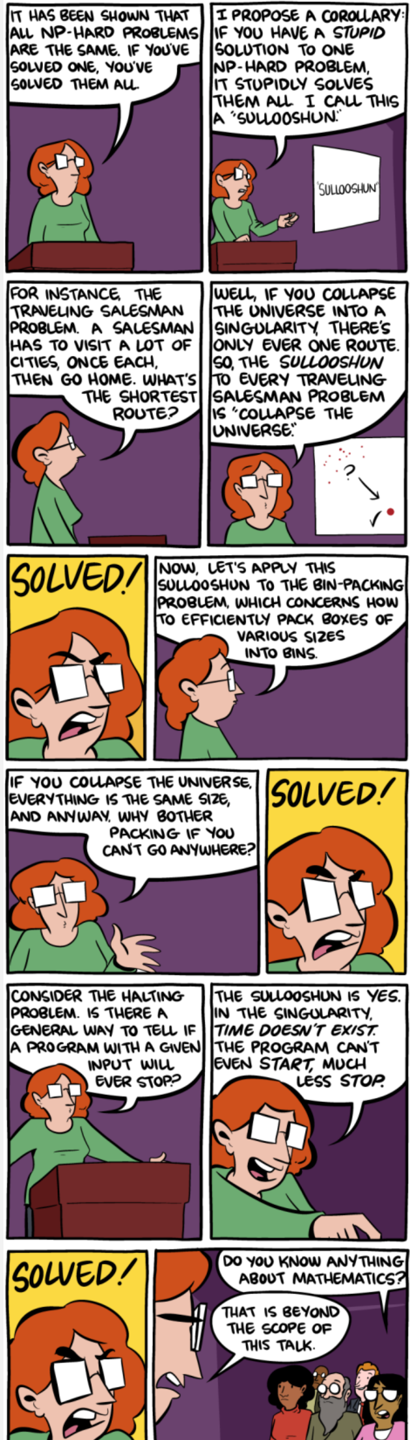
More NP-Complete Problems

Monday, November 22, 2021



Announcements

- HW 9 due Sun 11:59pm EST
  - (after break)

Last Time: **NP**-Completeness

DEFINITION

A language B is ***NP-complete*** if it satisfies two conditions:

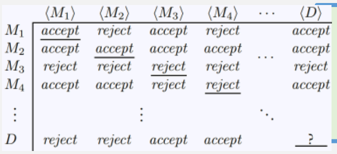
1. B is in NP, and
2. every A in NP is polynomial time reducible to B.

Must prove for <u>all</u> langs, not just a single language

THEOREM

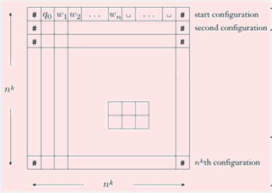
SAT is NP-complete.

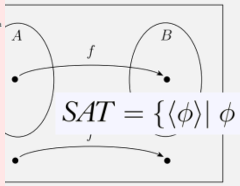
It's difficult to prove <u>the first</u> **NP**-complete problem!



	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\cdots$	$\langle D\rangle$
M_1	*accept*	*reject*	*accept*	*reject*		*accept*
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*
M_3	*reject*	*reject*	*reject*	*reject*		*reject*
M_4	*accept*	*accept*	*reject*	*reject*		*accept*
$\vdots$		$\vdots$			$\ddots$	
D	*reject*	*reject*	*accept*	*accept*		*?*

(Just like finding <u>the first</u> **undecidable** problem was hard!)





$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$

But each **NP**-complete problem we prove makes it **easier to prove the next one!**

THEOREM

known

unknown

Last Time: If B is NP-complete and $B \leq_{\mathrm{P}} C$ for C in NP, then C is NP-complete.

If you're not Stephen Cook or Leonid Levin, **use this theorem to prove a language is NP-complete**

THEOREM

Last Time: If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

3 steps to prove a language C is **NP**-complete:

1. Show C is in **NP**
2. Choose B, the **NP**-complete problem to reduce from
3. Show a poly time mapping reduction from B to C

To show poly time mapping reducibility:
1. create **computable fn**,
2. show that it **runs in poly time**,
3. then show **forward direction** of mapping red.,
4. and **reverse direction**
   (or **contrapositive** of **forward direction**)

THEOREM

Last Time: If B is NP-complete and $B \leq_{\mathrm{P}} C$ for C in NP, then C is NP-complete.

3 steps to prove a language C is **NP**-complete:

1. Show C is in **NP**
2. Choose B, the **NP**-complete problem to reduce from
3. Show a poly time mapping reduction from B to C

Example:

Let $C = 3SAT$, to prove $3SAT$ is **NP**-Complete:

1. Show $3SAT$ is in **NP**

THEOREM

Last Time: If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

<u>3 steps</u> to prove a language is **NP**-complete:

1. Show *C* is in **NP**
2. Choose *B*, the **NP**-complete problem to reduce from
3. Show a poly time mapping reduction from *B* to *C*

To show poly time mapping <u>mapping reducibility</u>:
1. create **computable fn**,
2. show that it **runs in poly time**,
3. then show **forward direction** of mapping red.,
4. and **reverse direction** (or **contrapositive** of **forward direction**)

<u>Example</u>:

Let *C* = *3SAT*, to prove *3SAT* is **NP**-Complete:

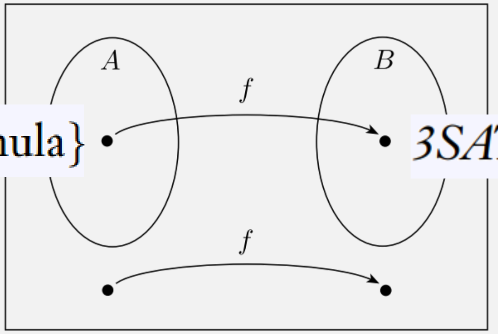
☑ 1. Show *3SAT* is in **NP**

☑ 2. Choose *B*, the **NP**-complete problem to reduce from: *SAT*

3. Show a poly time mapping reduction from *SAT* to *3SAT*

Flashback: *SAT* is Poly Time Reducible to *3SAT*

$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$



$3SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable 3cnf-formula}\}$

Need: poly time computable fn converting a Boolean formula ϕ to 3CNF:

1. Convert ϕ to CNF (an AND of OR clauses)
   a) Use DeMorgan's Law to push negations onto literals

      $\neg(P \vee Q) \iff (\neg P) \wedge (\neg Q)$ $\quad$ $\neg(P \wedge Q) \iff (\neg P) \vee (\neg Q)$ $\quad$ $O(n)$

   b) Distribute ORs to get ANDs outside of parens

      $(P \vee (Q \wedge R)) \Leftrightarrow ((P \vee Q) \wedge (P \vee R))$ $\quad$ $O(n)$

2. Convert to 3CNF by adding new variables

   $(a_1 \vee a_2 \vee a_3 \vee a_4) \Leftrightarrow (a_1 \vee a_2 \vee z) \wedge (\overline{z} \vee a_3 \vee a_4)$ $\quad$ $O(n)$

Remaining step: show iff relation holds …

… easy for formula conversion: each step is already a known "law"

THEOREM

Last Time: If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

<u>**3 steps**</u> to prove a language is **NP**-complete:

1. Show *C* is in **NP**
2. Choose *B,* the **NP**-complete problem to reduce from
3. Show a poly time mapping reduction from *B* to *C*

<u>**Theorem**</u>: *3SAT* is NP-complete

Let *C* = *3SAT*, to prove *3SAT* is **NP**-Complete:

☑ 1. Show *3SAT* is in **NP**

☑ 2. Choose *B,* the **NP**-complete problem to reduce from: *SAT*

☑ 3. Show a poly time mapping reduction from *SAT* to *3SAT*

■

Now have <u>2</u> **NP**-Complete languages to use:

- *SAT*
- *3SAT*

THEOREM

Last Time: If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

<u>3 steps</u> to prove a language is **NP**-complete:

1. Show C is in **NP**
2. Choose B, the **NP**-complete problem to reduce from
3. Show a poly time mapping reduction from B to C

<u>**Theorem**</u>: $CLIQUE$ is NP-complete

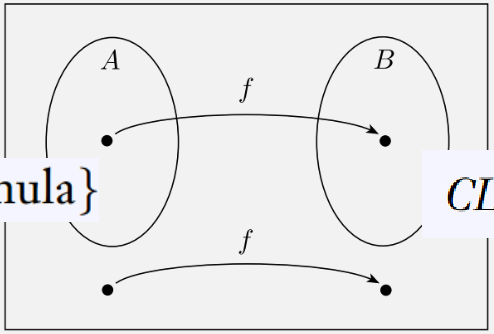
Let C = ~~*3SAT*~~ ***CLIQUE***, to prove ~~*3SAT*~~ ***CLIQUE*** is **NP**-Complete:

?1. Show ~~*3SAT*~~ ***CLIQUE*** is in **NP**

?2. Choose B, the **NP**-complete problem to reduce from: ~~*SAT*~~ ***3SAT***

?3. Show a poly time mapping reduction from B to C

Flashback: *3SAT* is polynomial time reducible to *CLIQUE*.



$3SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable 3cnf-formula}\}$

$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

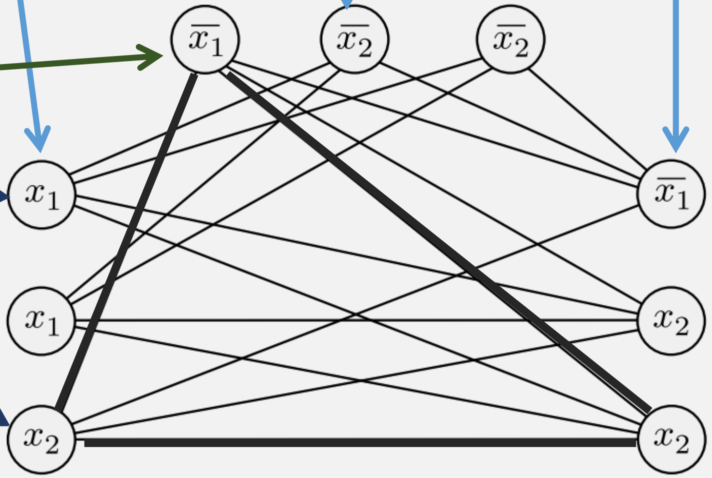
<u>Need</u>: poly time <u>computable fn</u> converting a 3cnf-formula …

Example:

$$\phi = (x_1 \vee x_1 \vee \boxed{x_2}) \wedge (\boxed{\overline{x_1}} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee \boxed{x_2})$$

- … to a graph containing a clique:
  - Each clause maps to a group of 3 nodes
  - Connect all nodes <u>except</u>:
    - Contradictory nodes
    - Nodes in the same group

Don't forget iff



Runs in **poly time:**
- # literals = # nodes $O(\boldsymbol{n})$
- # edges poly in # nodes $O(\boldsymbol{n}^2)$

⇒ If $\phi \in 3SAT$
- Then each clause has a TRUE literal
  - Those are nodes in the clique!
  - E.g., $x_1 = 0, x_2 = 1$

⇐ If $\phi \notin 3SAT$
- For any assignment, some clause must have a contradiction with another clause
- Then in the graph, some clause's group of nodes won't be connected to another group, preventing the clique

THEOREM

Last Time: If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

3 steps to prove a language is **NP**-complete:

1. Show C is in **NP**
2. Choose B, the **NP**-complete problem to reduce from
3. Show a poly time mapping reduction from B to C

Now have 3 **NP**-Complete languages to use:

- *SAT*
- *3SAT*
- *CLIQUE*

Theorem: *CLIQUE* is NP-complete

Let C = ~~*3SAT*~~ ***CLIQUE***, to prove ~~*3SAT*~~ ***CLIQUE*** is **NP**-Complete:

☑ 1. Show ~~*3SAT*~~ ***CLIQUE*** is in **NP**

☑ 2. Choose B, the **NP**-complete problem to reduce from: ~~SAT~~ ***3SAT***

☑ 3. Show a poly time mapping reduction from B to C

■

Last Time: **NP**-Complete problems, so far

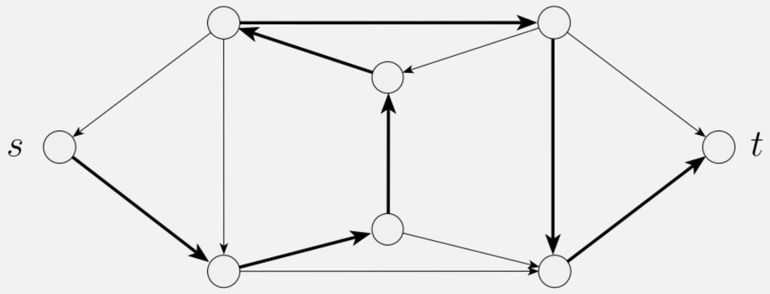
- $SAT = \{\langle \phi \rangle | \ \phi \text{ is a satisfiable Boolean formula}\}$ (Cook-Levin Theorem)
- $3SAT = \{\langle \phi \rangle | \ \phi \text{ is a satisfiable 3cnf-formula}\}$ (reduced *SAT* to *3SAT*)
- $CLIQUE = \{\langle G, k \rangle | \ G \text{ is an undirected graph with a } k\text{-clique}\}$ (reduced *3SAT* to *CLIQUE*)

We now have 3 options to choose from when proving the next **NP**-complete problem

Flashback: The *HAMPATH* Problem

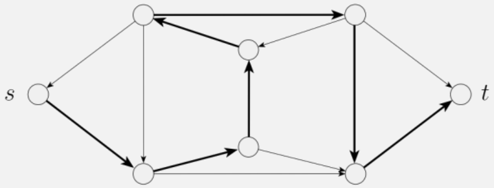
$$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$$

- A Hamiltonian path goes through every node in the graph



- The **Search** problem:
  - Exponential time (brute force) algorithm:
    - Check all possible paths of length $\boldsymbol{n}$
    - See if any connects s and t: $O(\boldsymbol{n}!) = O(2^n)$
  - Polynomial time algorithm:
    - Unknown!!!
- The **Verification** problem:
  - Still $O(\boldsymbol{n}^2)$, just like *PATH*!
- So *HAMPATH* is in **NP** but not known to be in **P**

Theorem: *HAMPATH* is NP-complete



$$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$$

THEOREM

Using: If B is NP-complete and $B \leq_{\mathrm{P}} C$ for C in NP, then C is NP-complete.

3 steps to prove a language is **NP**-complete:

1. Show C is in **NP**
2. Choose B, the **NP**-complete problem to reduce from
3. Show a poly time mapping reduction from B to C

Theorem: *HAMPATH* is NP-complete

$HAMPATH = \{\langle G, s, t\rangle \mid G$ is a directed graph with a Hamiltonian path from s to $t\}$

To prove *HAMPATH* is **NP**-complete:

☑ 1. Show *HAMPATH* is in **NP** (in HW9)

☑ 2. Choose *B*, the **NP**-complete problem to reduce from *3SAT*

3. Show a poly time mapping reduction from *B* to *HAMPATH*

Theorem: *HAMPATH* is NP-complete

$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$

To prove *HAMPATH* is **NP**-complete:

☑1. Show *HAMPATH* is in **NP** (in HW9)

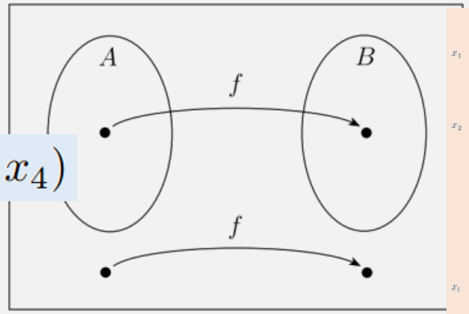
☑2. Choose *B*, the **NP**-complete problem to reduce from *3SAT*

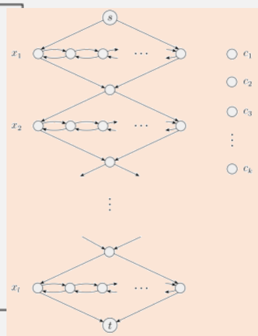
? 3. Show a poly time mapping reduction from *3SAT* to *HAMPATH*

To show poly time mapping reducibility:
1. create **computable fn**,
2. show that it **runs in poly time**,
3. then show **forward direction** of mapping red.,
4. and **reverse direction**
   (or **contrapositive** of **forward direction**)

???

$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$





Theorem: *HAMPATH* is NP-complete

$HAMPATH = \{\langle G, s, t\rangle \mid G$ is a directed graph with a Hamiltonian path from s to $t\}$

To prove *HAMPATH* is **NP**-complete:

☑ 1. Show *HAMPATH* is in **NP** (in HW9)

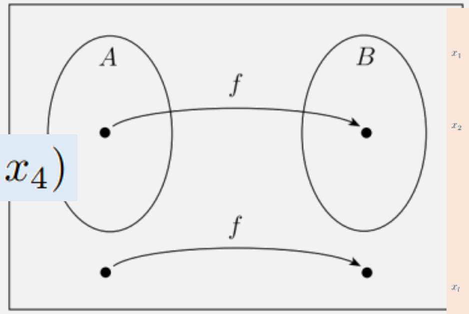
☑ 2. Choose *B*, the **NP**-complete problem to reduce from *3SAT*

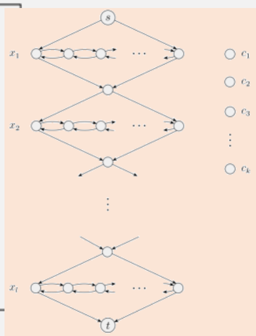
? 3. Show a poly time mapping reduction from *3SAT* to *HAMPATH*

To show poly time mapping reducibility:
1. create **computable fn**,
2. show that it **runs in poly time**,
3. then show **forward direction** of mapping red.,
4. and **reverse direction** (or **contrapositive** of **forward direction**)

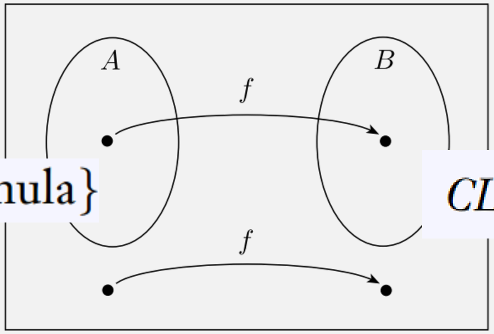
???

$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$





Flashback: $3SAT$ is polynomial time reducible to $CLIQUE$.



$3SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable 3cnf-formula}\}$

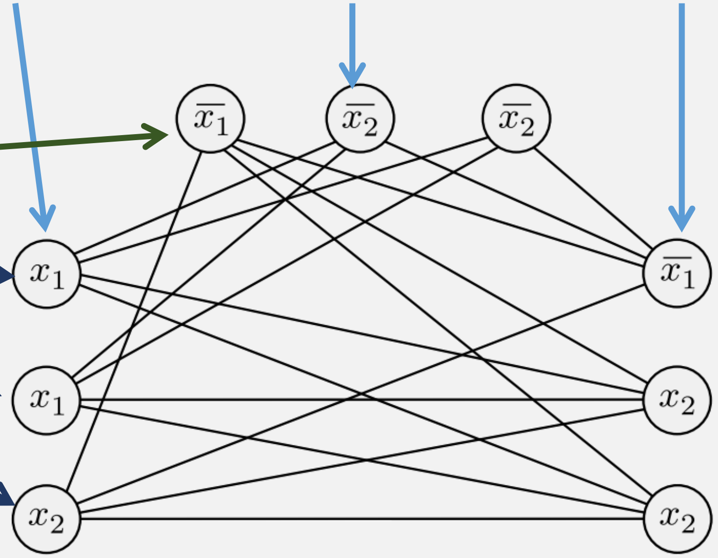
$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

Need: poly time computable fn converting a 3cnf-formula …

Example:

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

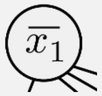
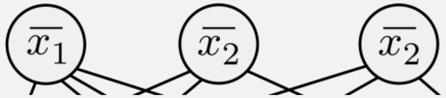
- … to a graph containing a clique:
  - Each clause maps to a group of 3 nodes
  - Connect all nodes except:
    - Contradictory nodes
    - Nodes in the same group



General Strategy: Reducing from *3SAT*

NOTE: "gadgets" are not always graphs

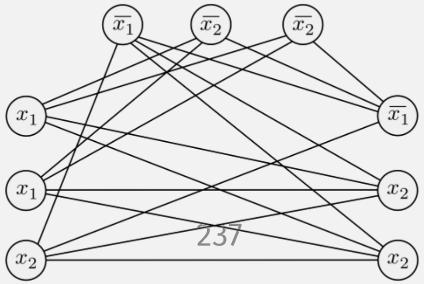
Create a **computable function** mapping formula to "gadgets":

- Variable → another "gadget", e.g., 
- Clause → some "gadget", e.g., 

  Gadget is typically "used" in two "opposite" ways:
  - "something" when var is assigned TRUE, or
  - "something else" when var is assigned FALSE

Then connect variable and clause "gadgets":

- Literal x_i in clause c_j → gadget x_i "connects to" gadget c_j
- Literal $\overline{x_i}$ in clause c_j → gadget x_i "connects to" gadget c_j
- E.g., connect each node to node not in clause



Theorem: *HAMPATH* is NP-complete

$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$

To prove *HAMPATH* is **NP**-complete:

☑ 1. Show *HAMPATH* is in **NP** (in HW9)

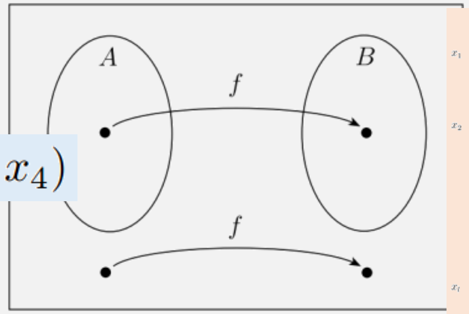
☑ 2. Choose *B*, the **NP**-complete problem to reduce from *3SAT*

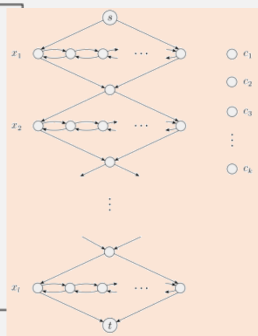
? 3. Show a poly time mapping reduction from *3SAT* to *HAMPATH*

To show poly time mapping reducibility:
1. create **computable fn**,
2. show that it **runs in poly time**,
3. then show **forward direction** of mapping red.,
4. and **reverse direction** (or **contrapositive** of **forward direction**)

???

$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$





Computable Fn: Formula (blue) → Graph (orange)

variable   clause

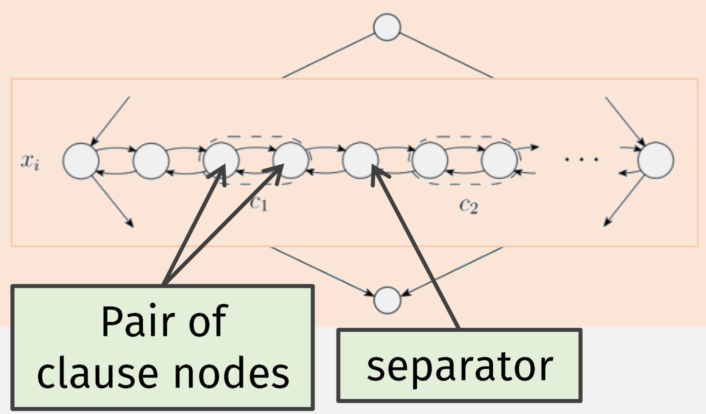
Example input: $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_k \vee b_k \vee c_k)$

k = # clauses

- Clause → (extra) single nodes, Total = k

(extra)

- Variable → diamond-shaped graph "gadget"
  - Clause → 2 "connector" nodes + separator
  - Total = $3k$+1 "connector" nodes per "gadget"



Computable Fn: Formula (blue) → Graph (orange)

Example input: $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_k \vee b_k \vee c_k)$

k = # clauses

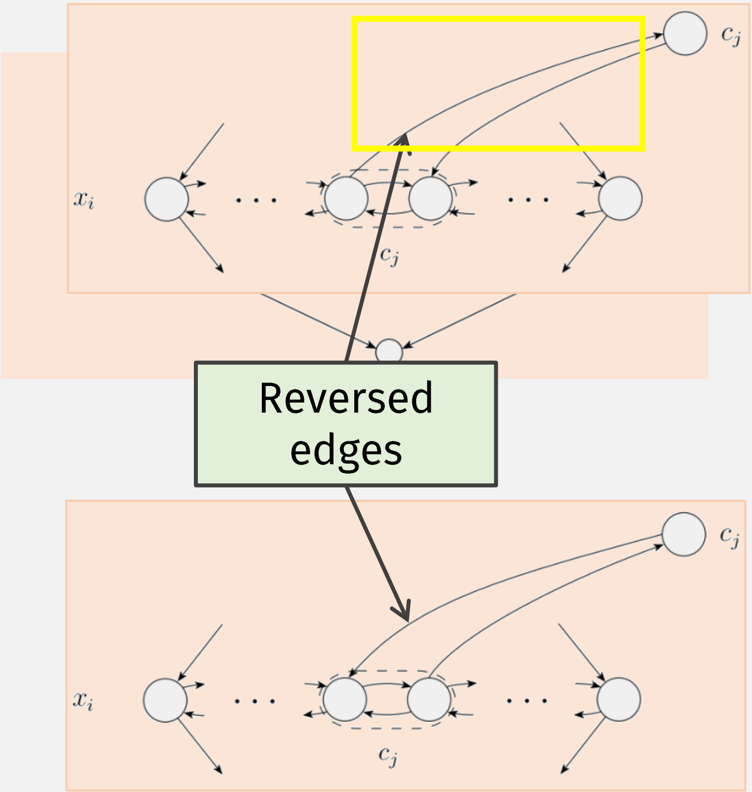
- Clause → (extra) single nodes, Total = k
- Variable → diamond-shaped graph "gadget"
  - Clause → 2 "connector" nodes + separator
  - Total = $3k$+1 "connector" nodes per "gadget"

Literal = variable or negated variable

- Lit x_i in clause c_j → c_j node edges in gadget x_i

Each extra c_j node has 6 edges

- Lit $\overline{x_i}$ in clause c_j → c_j edges in gadget x_i (rev)



Theorem: *HAMPATH* is NP-complete

$HAMPATH = \{\langle G, s, t\rangle \mid G$ is a directed graph with a Hamiltonian path from s to $t\}$

To prove *HAMPATH* is **NP**-complete:

☑ 1. Show *HAMPATH* is in **NP**

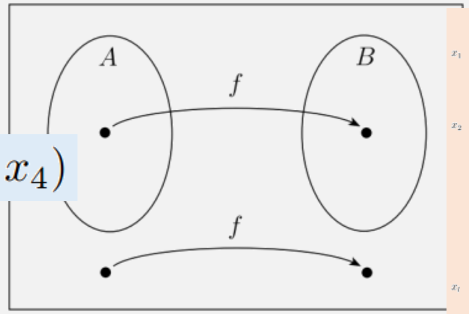
☑ 2. Choose *B*, the **NP**-complete problem to reduce from *3SAT*

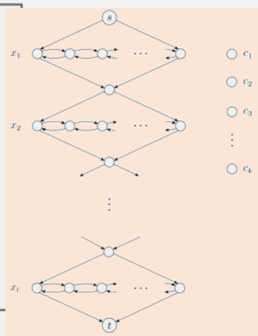
? 3. Show a poly time mapping reduction from *3SAT* to *HAMPATH*

To show poly time mapping reducibility:

☑ 1. create **computable fn**,
2. show that it **runs in poly time**,
3. then show **forward direction** of mapping red.,
4. and **reverse direction**
   (or **contrapositive** of **forward direction**)

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$$





Polynomial Time?

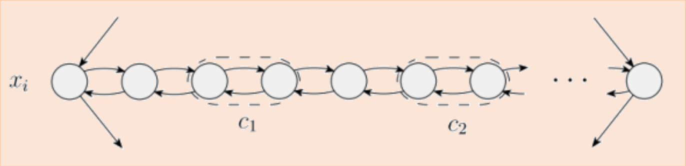
TOTAL: $O(k^2)$

Example input: $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_k \vee b_k \vee c_k)$

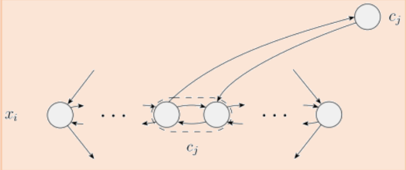
k = # clauses = at most $\mathbf{3k}$ variables

- Clause → (extra) single nodes  $O(k)$

- Variable → diamond-shaped graph "gadget" $O(k^2)$
  - Clause → 2 "connector" nodes + separator
  - Total = $3k+1$ "connector" nodes per "gadget"

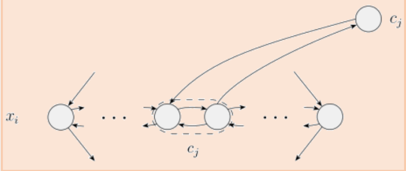


- Lit x_i in clause c_j → c_j node edges in gadget x_i



$O(k)$

- Lit $\overline{x_i}$ in clause c_j → c_j edges in gadget x_i (rev)



$O(k)$

Theorem: *HAMPATH* is NP-complete

$HAMPATH = \{\langle G, s, t \rangle | \ G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$

To prove *HAMPATH* is **NP**-complete:

☑ 1. Show *HAMPATH* is in **NP**

☑ 2. Choose *B*, the **NP**-complete problem to reduce from *3SAT*

? 3. Show a poly time mapping reduction from *3SAT* to *HAMPATH*

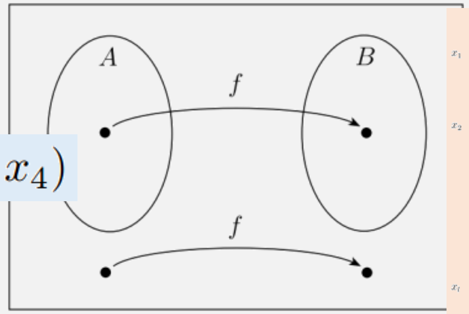
To show poly time mapping reducibility:

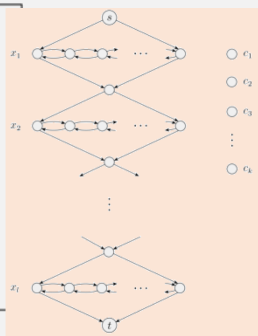
☑ 1. create **computable fn**,

☑ 2. show that it **runs in poly time**,

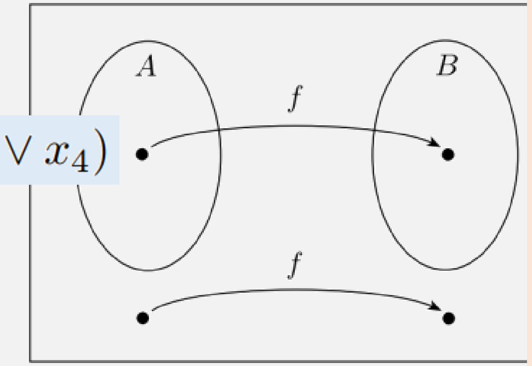
3. then show **forward direction** of mapping red.,
4. and **reverse direction**
   (or **contrapositive** of **forward direction**)

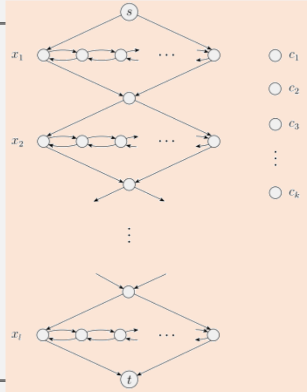
$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$





$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$



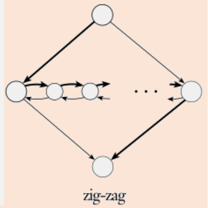


Want: Satisfiable 3cnf formula ⇔ graph with Hamiltonian path

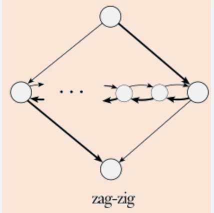
⇒ If there is satisfying assignment, then Hamiltonian path exists

These hit all nodes except extra c_js

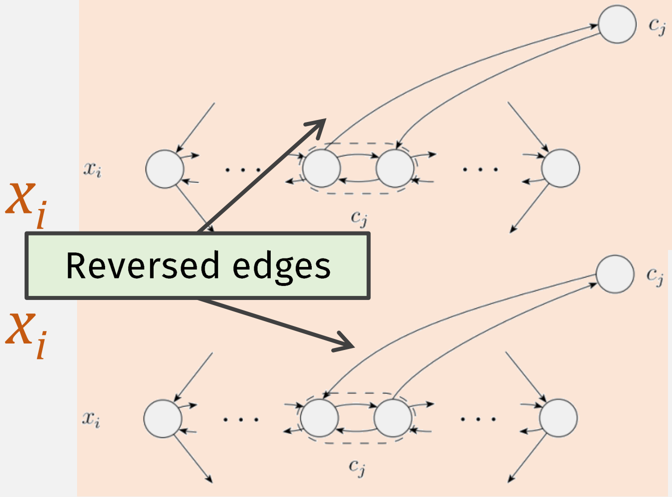
- x_i = TRUE → Hampath "zig-zags" gadget x_i



- x_i = FALSE → Hampath "zag-zigs" gadget x_i



- Lit x_i makes clause c_j TRUE → "detour" to c_j in gadget x_i
- Lit $\overline{x_i}$ makes clause c_j TRUE → "detour" to c_j in gadget x_i

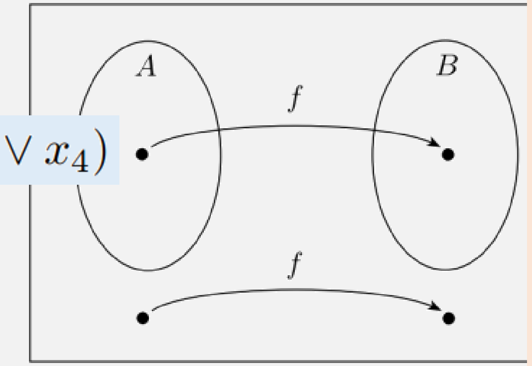


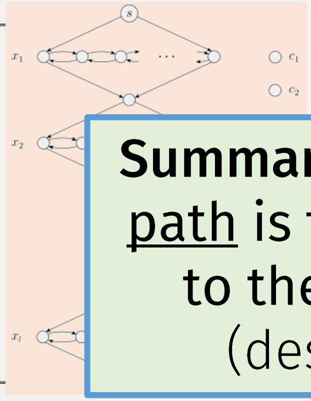
Now path goes through every node

Every clause must be TRUE so path hits all c_j nodes

- And edge directions align with TRUE/FALSE assignments

$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$

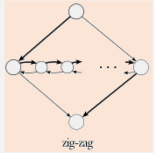




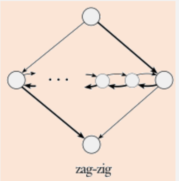
Summary: the only possible Ham. path is the one that corresponds to the satisfying assignment (described on prev slide)

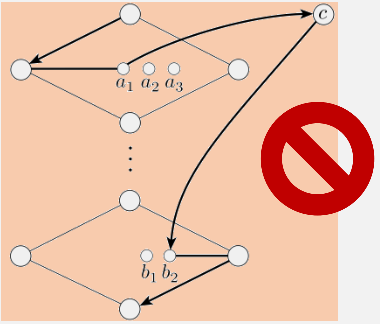
Want: Satisfiable 3cnf formula ⇔ graph with Hamiltonian path

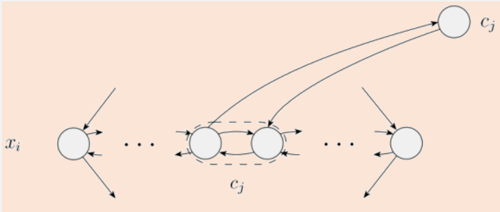
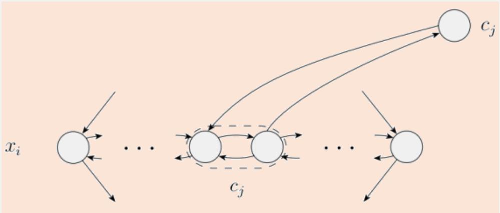
⇐ if output has Ham. path, then input had Satisfying assignment



- A Hamiltonian path must choose to either zig-zag or zag-zig gadgets
- Ham path can only hit "detour" c_j nodes by coming right back
- Otherwise, it will miss some nodes





-  gadget x_i "detours" from left to right → x_i = TRUE
-  gadget x_i "detours" from right to left → x_i = FALSE

Theorem: *HAMPATH* is NP-complete

$HAMPATH = \{\langle G, s, t\rangle \mid G$ is a directed graph with a Hamiltonian path from s to $t\}$

To prove *HAMPATH* is **NP**-complete:

☑ 1. Show *HAMPATH* is in **NP**

☑ 2. Choose *B*, the **NP**-complete problem to reduce from *3SAT*

☑ 3. Show a poly time mapping reduction from *3SAT* to *HAMPATH*

To show poly time mapping reducibility:

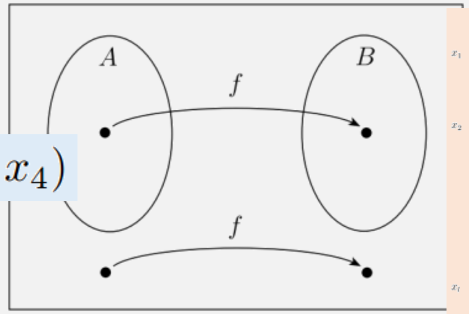
☑ 1. create **computable fn**,

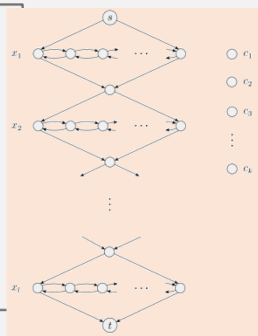
☑ 2. show that it **runs in poly time**,

☑ 3. then show **forward direction** of mapping red.,

☑ 4. and **reverse direction** (or **contrapositive** of **forward direction**)

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$$





■

Theorem: *UHAMPATH* is NP-complete

UHAMPATH = $\{\langle G, s, t\rangle \mid G$ is a ~~directed~~ undirected graph with a Hamiltonian path from s to $t\}$

To prove *UHAMPATH* is **NP**-complete:

☑ 1. Show *UHAMPATH* is in **NP**

➡ 2. Choose the **NP**-complete problem to reduce from *HAMPATH*

3. Show a poly time mapping reduction from *???* to *UHAMPATH*

Theorem: *UHAMPATH* is NP-complete

$UHAMPATH = \{\langle G, s, t \rangle \mid G$ is a ~~directed~~ undirected graph with a Hamiltonian path from s to $t\}$

To prove *UHAMPATH* is **NP**-complete:

☑ 1. Show *UHAMPATH* is in **NP**

☑ 2. Choose the **NP**-complete problem to reduce from *HAMPATH*

➡ 3. Show a poly time mapping reduction from *HAMPATH* to *UHAMPATH*

Theorem: *UHAMPATH* is NP-complete

un

$$UHAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$$

Need: Computable function from *HAMPATH* to *UHAMPATH*

Naïve Idea: Make all directed edges undirected?

- But we would create some paths that didn't exist before

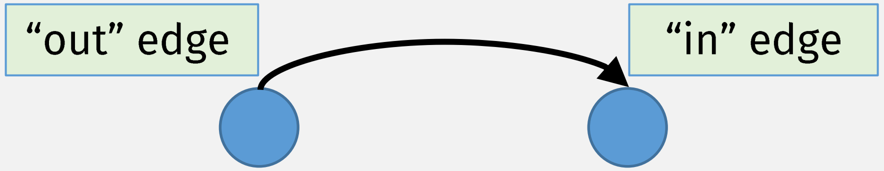
- Doesn't work!

Theorem: *UHAMPATH* is NP-complete

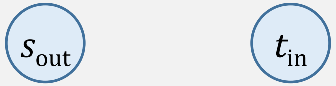
$UHAMPATH = \{\langle G, s, t \rangle |\ G$ is an undirected graph with a Hamiltonian path from s to $t\}$

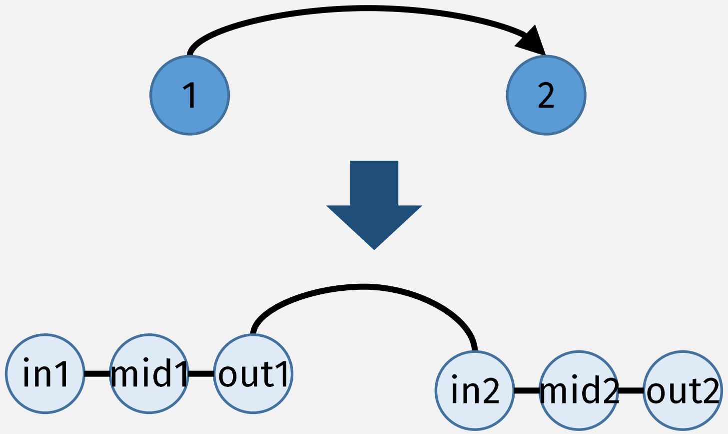
Need: Computable function from *HAMPATH* to *UHAMPATH*

Better Idea:



- Distinguish “in” vs “out” edges
- Nodes (directed) → 3 Nodes (undirected): **in/mid/out**
  - Connect in/mid/out with edges
  - Directed edge (u, v) → $(u_{\text{out}}, v_{\text{in}})$
- Except: s → s_{out}, t → t_{in} **only!**



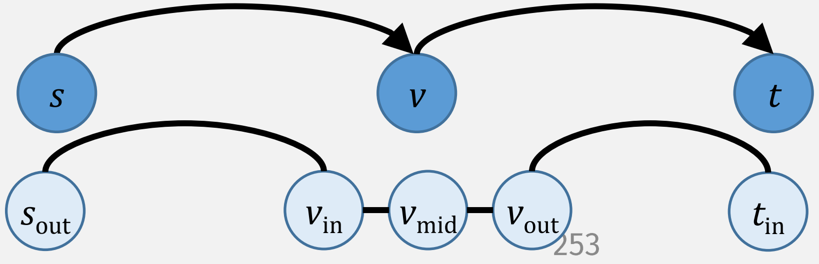


Theorem: *UHAMPATH* is NP-complete

$UHAMPATH = \{\langle G, s, t\rangle \mid G$ is a undirected graph with a Hamiltonian path from s to $t\}$

Need: Computable function from *HAMPATH* to *UHAMPATH*

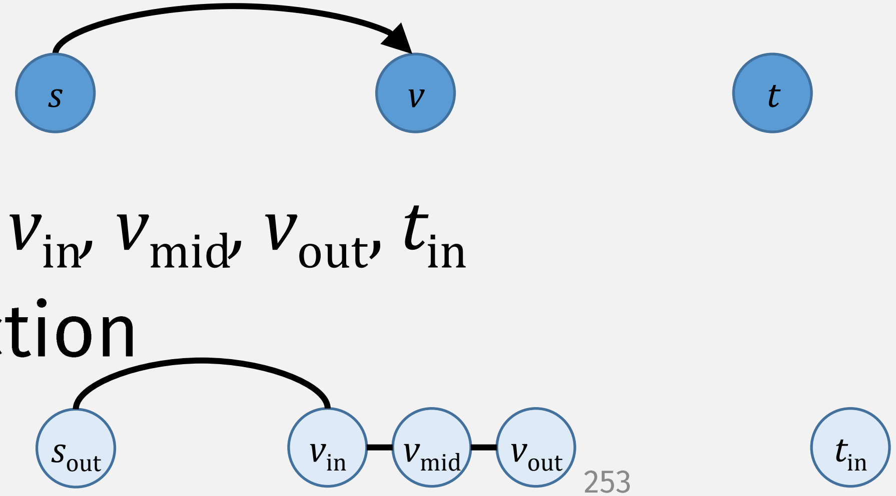
$\Rightarrow$



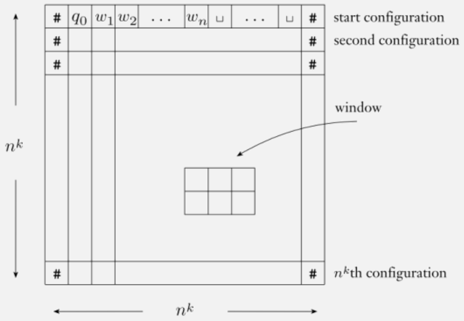
- If there was a directed path s, v, t ...
- ... then there is an undirected path $s_{out}, v_{in}, v_{mid}, v_{out}, t_{in}$

$\Leftarrow$

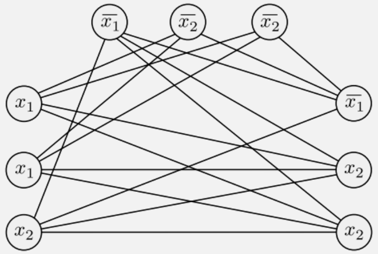
- If there was no directed path s, v, t ...
- ... then there is no undirected path $s_{out}, v_{in}, v_{mid}, v_{out}, t_{in}$
- Because there will be a missing connection



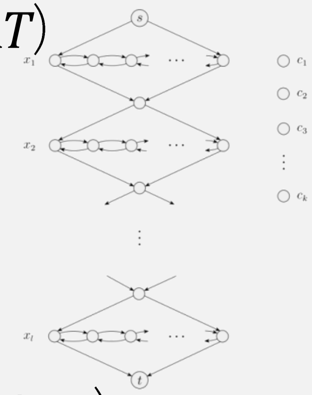
NP-Complete problems, so far



- $SAT = \{\langle\phi\rangle \mid \phi$ is a satisfiable Boolean formula$\}$ (Cook-Levin Theorem)



- $3SAT = \{\langle\phi\rangle \mid \phi$ is a satisfiable 3cnf-formula$\}$ (reduce from *SAT*)
- $CLIQUE = \{\langle G, k\rangle \mid G$ is an undirected graph with a k-clique$\}$ (reduce from 3*SAT*)



- $HAMPATH = \{\langle G, s, t\rangle \mid G$ is a directed graph with a Hamiltonian path from s to $t\}$ (reduce from 3*SAT*)
- $UHAMPATH = \{\langle G, s, t\rangle \mid G$ is a undirected graph with a Hamiltonian path from s to $t\}$ (reduce from *HAMPATH*)

More **NP**-Complete problems

- $$\textit{SUBSET-SUM} = \{\langle S, t\rangle |\ S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \Sigma y_i = t\}$$
  - (reduce from *3SAT*)

- $$\textit{VERTEX-COVER} = \{\langle G, k\rangle |\ G \text{ is an undirected graph that has a } k\text{-node vertex cover}\}$$
  - (reduce from *3SAT*)

Theorem: *SUBSET-SUM* is NP-complete

$$SUBSET\text{-}SUM = \{\langle S,t\rangle \mid S = \{x_1,\ldots,x_k\}, \text{ and for some } \{y_1,\ldots,y_l\} \subseteq \{x_1,\ldots,x_k\}, \text{ we have } \Sigma y_i = t\}$$

50 kg??

THEOREM

Using: If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

3 steps to prove a language is **NP**-complete:

1. Show C is in **NP**
2. Choose B, the **NP**-complete problem to reduce from
3. Show a poly time mapping reduction from B to C

Theorem: *SUBSET-SUM* is NP-complete

$$SUBSET\text{-}SUM = \{\langle S,t\rangle \mid S = \{x_1,\ldots,x_k\}, \text{ and for some } \{y_1,\ldots,y_l\} \subseteq \{x_1,\ldots,x_k\}, \text{ we have } \Sigma y_i = t\}$$

3 steps to prove *SUBSET-SUM* is **NP**-complete:

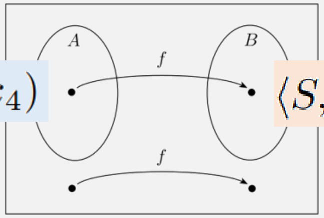
☑ 1. Show *SUBSET-SUM* is in **NP**

☑ 2. Choose the **NP**-complete problem to reduce from: *3SAT*

3. Show a poly time mapping reduction from *3SAT* to *SUBSET-SUM*

To show poly time mapping reducibility:
1. create **computable fn**,
2. show that it **runs in poly time**,
3. then show **forward direction** of mapping red.,
4. and **reverse direction** (or **contrapositive** of **forward direction**)

$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$

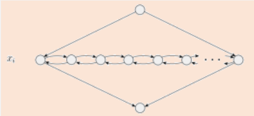


$\langle S,t\rangle \mid S = \{x_1,\ldots,x_k\}$

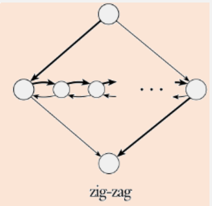
Review: Reducing from *3SAT*

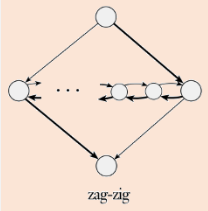
Create a **computable function** mapping formula to "gadgets":

- Clause → some "gadget", e.g., 

- Variable → another "gadget", e.g., 

  Gadget is typically used in two "opposite" ways:
  - ZIG when var is assigned TRUE, or
  - ZAG when var is assigned FALSE

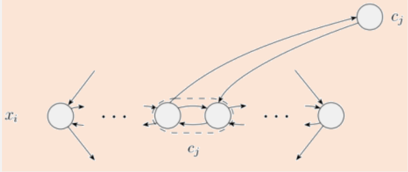


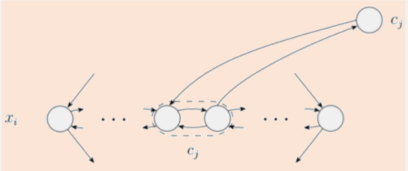


NOTE: "gadgets" are not always graphs

Then connect "gadgets" according to clause literals:

- Literal x_i in clause c_j → gadget x_i "detours" to c_j
- Literal $\overline{x_i}$ in clause c_j → gadget x_i "reverse detours" to c_j





Computable Fn: 3cnf → $\langle S, t \rangle$

E.g., $(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee x_3 \vee \cdots) \wedge \cdots \wedge (\overline{x_3} \vee \cdots \vee \cdots)$ ➡

- Assume formula has:
  - ***l*** variables $x_1, \ldots, x_l$
  - ***k*** clauses $c_1, \ldots, c_k$
- Computable function *f* maps:
  - Variable x_i →
  - Clause c_j →
  - arranged in a table …
- Each number has max ***l*+*k*** digits:
  - Literal x_i in clause c_j →
  - Literal $\overline{x_i}$ in clause c_j →
- Sum is ***l*** 1s followed by ***k*** 3s

	1	2	3	4	$\cdots$	l	c_1	c_2	$\cdots$	c_k
y_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
z_1	1	0	0	0	$\cdots$	0	0	0	$\cdots$	0
y_2		1	0	0	$\cdots$	0	0	1	$\cdots$	0
z_2		1	0	0	$\cdots$	0	1	0	$\cdots$	0
y_3			1	0	$\cdots$	0	1	1	$\cdots$	0
z_3			1	0	$\cdots$	0	0	0	$\cdots$	1
$\vdots$					$\ddots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
y_l						1	0	0	$\cdots$	0
z_l						1	0	0	$\cdots$	0
g_1							1	0	$\cdots$	0
h_1							1	0	$\cdots$	0
g_2								1	$\cdots$	0
h_2								1	$\cdots$	0
$\vdots$									$\ddots$	$\vdots$
g_k										1
h_k										1
t	1	1	1	1	$\cdots$	1	3	3	$\cdots$	3

y_i and z_i: i^{th} digit = 1

y_i: ***l***+j^{th} digit = 1 if c_j has x_i

z_i: ***l***+j^{th} digit = 1 if c_j has $\overline{x_i}$

g_j and h_j: ***l***+j^{th} digit = 1, To help get the right sum

The sum

<u>Theorem</u>: *SUBSET-SUM* is NP-complete

$SUBSET\text{-}SUM = \{\langle S,t\rangle \mid S = \{x_1,\ldots,x_k\}, \text{ and for some } \{y_1,\ldots,y_l\} \subseteq \{x_1,\ldots,x_k\}, \text{ we have } \Sigma y_i = t\}$

<u>3 steps</u> to prove *SUBSET-SUM* is **NP**-complete:

☑ 1. Show *SUBSET-SUM* is in **NP**

☑ 2. Choose the **NP**-complete problem to reduce from: *3SAT*

3. Show a poly time mapping reduction from *3SAT* to *SUBSET-SUM*

To show poly time <u>mapping reducibility</u>:

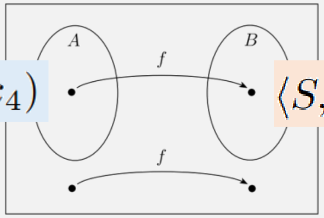
☑ 1. create **computable fn**,

2. show that it **runs in poly time**,

3. then show **forward direction** of mapping red.,

4. and **reverse direction**
   (or **contrapositive** of **forward direction**)

$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$



$\langle S,t\rangle \mid S = \{x_1,\ldots,x_k\}$

Polynomial Time?

E.g., $(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee x_3 \vee \cdots) \wedge \cdots \wedge (\overline{x_3} \vee \cdots \vee \cdots)$ ➡

- Assume formula has:
  - ***l*** variables $x_1, \ldots, x_l$
  - ***k*** clauses $c_1, \ldots, c_k$
- Table size: (***l*** + ***k***) * (2***l*** + 2***k***)
  - Creating it requires constant number of passes over the table
  - Num variables ***l*** = at most 3***k***
- Total: $O(k^2)$

	1	2	3	4	$\cdots$	l	c_1	c_2	$\cdots$	c_k
y_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
z_1	1	0	0	0	$\cdots$	0	0	0	$\cdots$	0
y_2		1	0	0	$\cdots$	0	0	1	$\cdots$	0
z_2		1	0	0	$\cdots$	0	1	0	$\cdots$	0
y_3			1	0	$\cdots$	0	1	1	$\cdots$	0
z_3			1	0	$\cdots$	0	0	0	$\cdots$	1
$\vdots$					$\ddots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
y_l						1	0	0	$\cdots$	0
z_l						1	0	0	$\cdots$	0
g_1							1	0	$\cdots$	0
h_1							1	0	$\cdots$	0
g_2								1	$\cdots$	0
h_2								1	$\cdots$	0
$\vdots$									$\ddots$	$\vdots$
g_k										1
h_k										1
t	1	1	1	1	$\cdots$	1	3	3	$\cdots$	3

Theorem: *SUBSET-SUM* is NP-complete

$SUBSET\text{-}SUM = \{\langle S,t\rangle \mid S=\{x_1,\ldots,x_k\}, \text{ and for some } \{y_1,\ldots,y_l\}\subseteq\{x_1,\ldots,x_k\}, \text{ we have } \Sigma y_i = t\}$

3 steps to prove *SUBSET-SUM* is **NP**-complete:

☑ 1. Show *SUBSET-SUM* is in **NP**

☑ 2. Choose the **NP**-complete problem to reduce from: *3SAT*

3. Show a poly time mapping reduction from *3SAT* to *SUBSET-SUM*

To show poly time mapping reducibility:

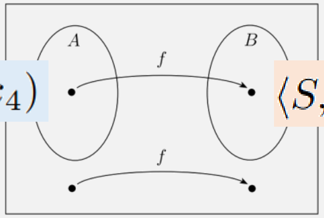
☑ 1. create **computable fn**,

☑ 2. show that it **runs in poly time**,

➡ 3. then show **forward direction** of mapping red.,

4. and **reverse direction** (or **contrapositive** of **forward direction**)

$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$



$\langle S,t\rangle \mid S=\{x_1,\ldots,x_k\}$

ϕ is a satisfiable 3cnf-formula $\iff$ $f(\langle\phi\rangle) = \langle S, t\rangle$ where some subset of S sums to t

⇒ **<u>If</u>** formula is satisfiable …

- Sum t = l 1s followed by k 3s
- Choose for the subset …
  - y_i if x_i = TRUE
  - z_i if x_i = FALSE
  - and some of g_i and h_i to make the sum t
- … **<u>Then</u>** this subset of S must sum to t bc:
  - <u>Left digits</u>:
    - only one of y_i or z_i is in S
  - <u>Right digits</u>:
    - <u>Top right</u>: Each column sums to 1, 2, or 3
      - Because each clause has 3 literals
    - <u>Bottom right</u>:
      - Can always use g_i and/or h_i to make column sum to 3

Each column:
- At least one 1
- At most 3 1s

S only includes one

	1	2	3	4	$\cdots$	l	c_1	c_2	$\cdots$	c_k
y_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
z_1	1	0	0	0	$\cdots$	0	0	0	$\cdots$	0
y_2		1	0	0	$\cdots$	0	0	1	$\cdots$	0
z_2		1	0	0	$\cdots$	0	1	0	$\cdots$	0
y_3			1	0	$\cdots$	0	1	1	$\cdots$	0
z_3			1	0	$\cdots$	0	0	0	$\cdots$	1
$\vdots$					$\ddots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
y_l						1	0	0	$\cdots$	0
z_l						1	0	0	$\cdots$	0
g_1							1	0	$\cdots$	0
h_1							1	0	$\cdots$	0
g_2								1	$\cdots$	0
h_2								1	$\cdots$	0
$\vdots$									$\ddots$	$\vdots$
g_k										1
h_k										1
t	1	1	1	1	$\cdots$	1	3	3	$\cdots$	3

g_j and h_j: help get the correct sum

So each column sum (for left digits) is 1

ϕ is a satisfiable 3cnf-formula $\iff$ $f(\langle\phi\rangle) = \langle S, t\rangle$ where some sub… o t

Subset must have some number with 1 in each right column

$\Leftarrow$ **If** a subset of S sums to t …

The only way to do it is as prev described:

- It can only include either y_i or z_i
  - Because each left digit column must sum to 1
  - And no carrying is possible
- Also, since each right digit column must sum to 3:
  - And only 2 can come from g_i and h_i
  - Then for every right column, some y_i or z_i in the subset has a 1 in that column
- … **Then** table must have been created from a sat. ϕ:
  - x_i = TRUE if y_i in the subset
  - x_i = FALSE if z_i in the subset
- This is satisfying because:
  - Table was constructed so 1 in column c_j for y_i or z_i means that variable x_i satisfies clause c_j
  - We already determined, for every right column, some number in the subset has a 1 in the column
  - So all clauses are satisfied

S only includes y_i or z_i

	1	2	3	4	$\cdots$	l	c_1	c_2	$\cdots$	c_k
y_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
z_1	1	0	0	0	$\cdots$	0	0	0	$\cdots$	0
y_2		1	0	0	$\cdots$	0	0	1	$\cdots$	0
z_2		1	0	0	$\cdots$	0	1	0	$\cdots$	0
y_3			1	0	$\cdots$	0	1	1	$\cdots$	0
z_3			1	0	$\cdots$	0	0	0	$\cdots$	1
$\vdots$					$\ddots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
						1	0	0	$\cdots$	0
						1	0	0	$\cdots$	0
							1	0	$\cdots$	0
							1	0	$\cdots$	0
								1	$\cdots$	0
h_2								1	$\cdots$	0
$\vdots$									$\ddots$	$\vdots$
g_k										1
h_k										1
t	1	1	1	1	$\cdots$	1	3	3	$\cdots$	3

In each right column, g_i and h_i can account for at most 2

Because each column sum (for left digits) is 1

More **NP**-Complete problems

☑ • $SUBSET\text{-}SUM = \{\langle S, t\rangle |\ S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \Sigma y_i = t\}$

  - (reduce from *3SAT*)

- $VERTEX\text{-}COVER = \{\langle G, k\rangle |\ G \text{ is an undirected graph that has a } k\text{-node vertex cover}\}$

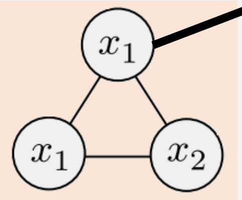
  - (reduce from *3SAT*)

Theorem: *VERTEX-COVER* is NP-complete

$VERTEX\text{-}COVER = \{\langle G, k\rangle \mid G$ is an undirected graph that has a k-node vertex cover$\}$

- A vertex cover of a graph is …
  - … a subset of its nodes where every edge touches one of those nodes
- Proof Sketch: Reduce *3SAT* to *VERTEX-COVER*
- The reduction maps:
- Variable x_i → 2 connected nodes
  - corresponding to the var and its negation, e.g.,

- Clause → 3 connected nodes
  - corresponding to its literals, e.g.,

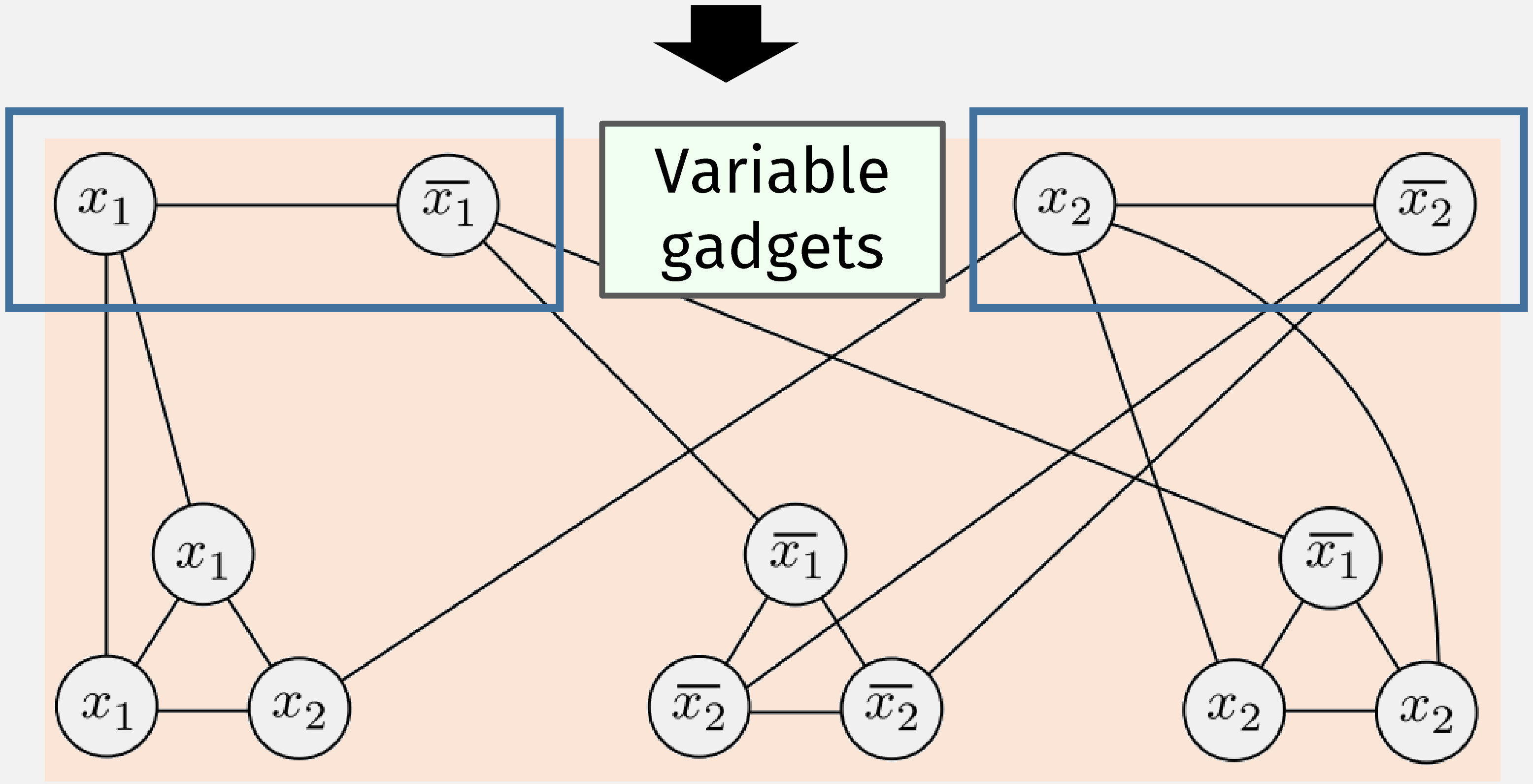


- Additionally,
  - connect var and clause gadgets by …
  - … connecting nodes that correspond to the same literal

VERTEX-COVER example

VERTEX-COVER = $\{\langle G, k\rangle \mid$ G is an undirected graph that has a k-node vertex cover$\}$

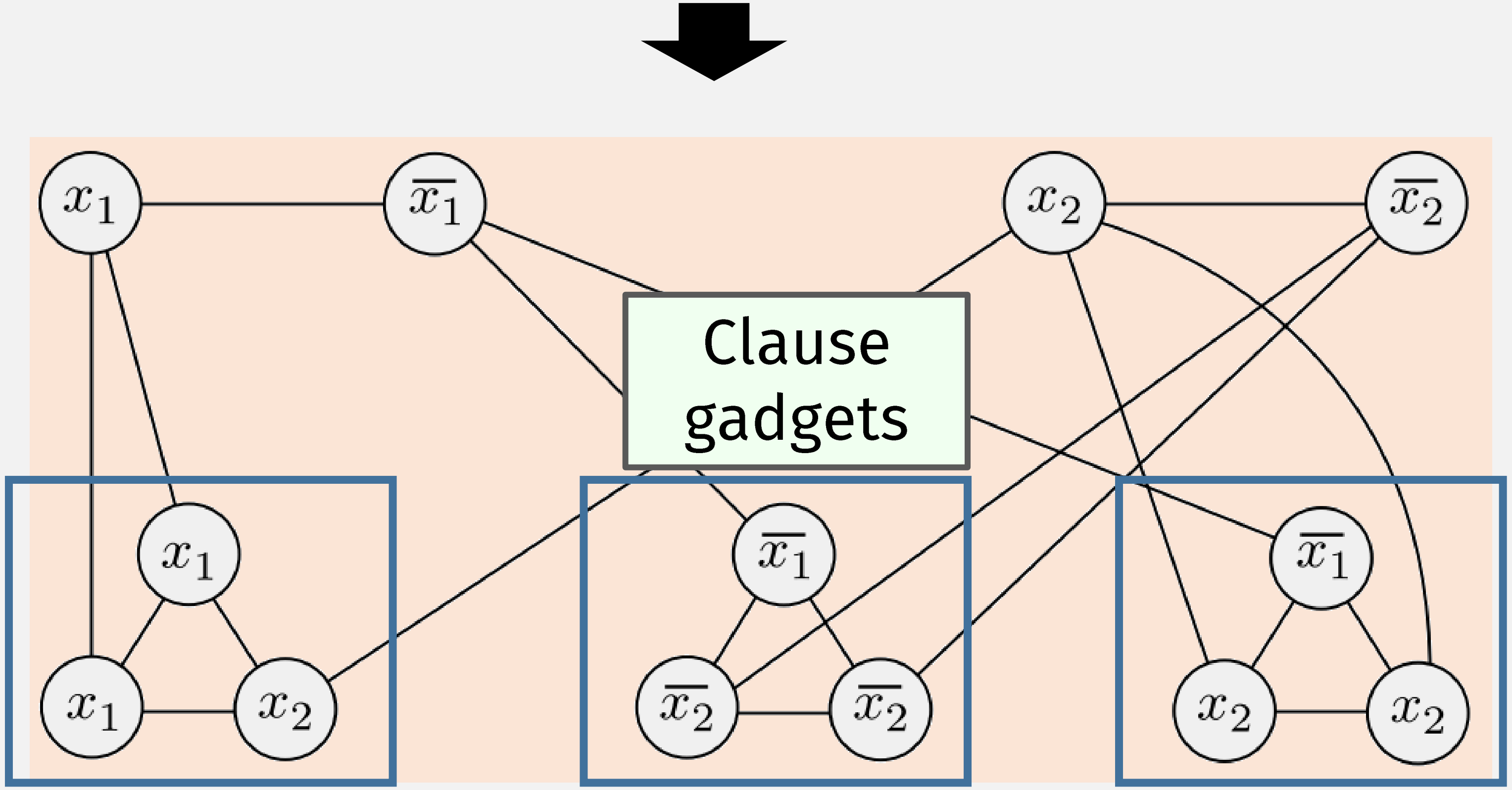
$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



VERTEX-COVER example

VERTEX-COVER $= \{\langle G, k\rangle \mid G$ is an undirected graph that has a k-node vertex cover$\}$

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

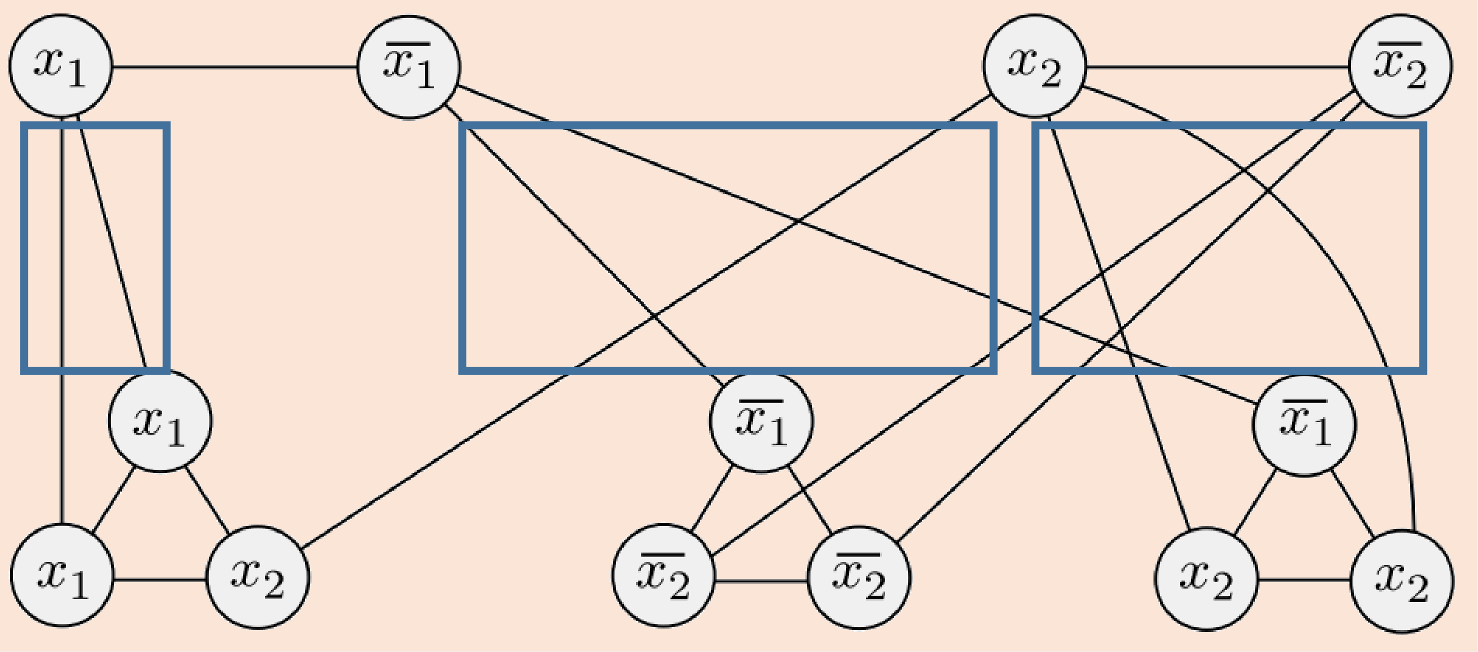


VERTEX-COVER example

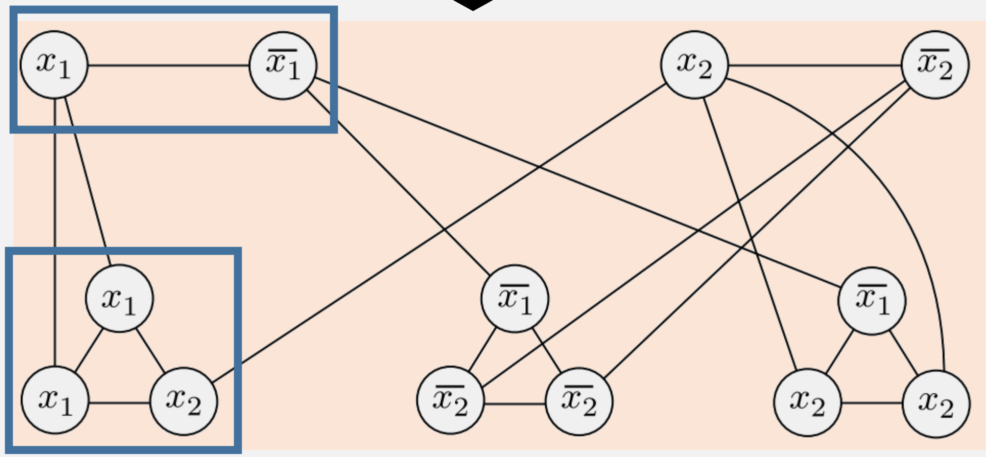
VERTEX-COVER = $\{\langle G, k\rangle \mid G$ is an undirected graph that has a k-node vertex cover$\}$

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

Extra edges connecting variable and clause gadgets together



$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



VERTEX-COVER example

- If formula has …
  - ***m*** = # variables
  - ***l*** = # clauses
- Then graph has …
  - # nodes = 2***m*** + 3***l***

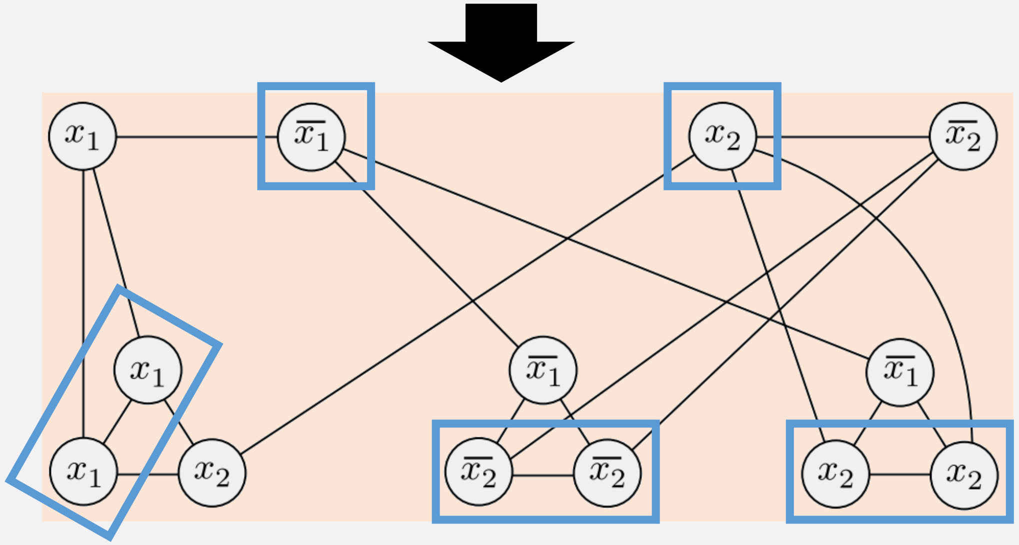
⇒ If satisfying assignment, then there is a ***k***-cover, where ***k*** = ***m*** + 2***l***

- Nodes in the cover:
  - In each of ***m*** var gadgets, choose 1 node corresponding to TRUE literal
  - For each of ***l*** clause gadgets, ignore 1 TRUE literal and choose other 2
  - Since there is satisfying assignment, each clause has a TRUE literal
  - Total = ***m*** + 2***l***

VERTEX-COVER = $\{\langle G, k\rangle \mid G$ is an undirected graph that has a k-node vertex cover$\}$

$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

VERTEX-COVER example



- If formula has …
  - m = # variables
  - l = # clauses
- Then graph has …
  - # nodes = $2m + 3l$

Example:
x_1 = FALSE
x_2 = TRUE

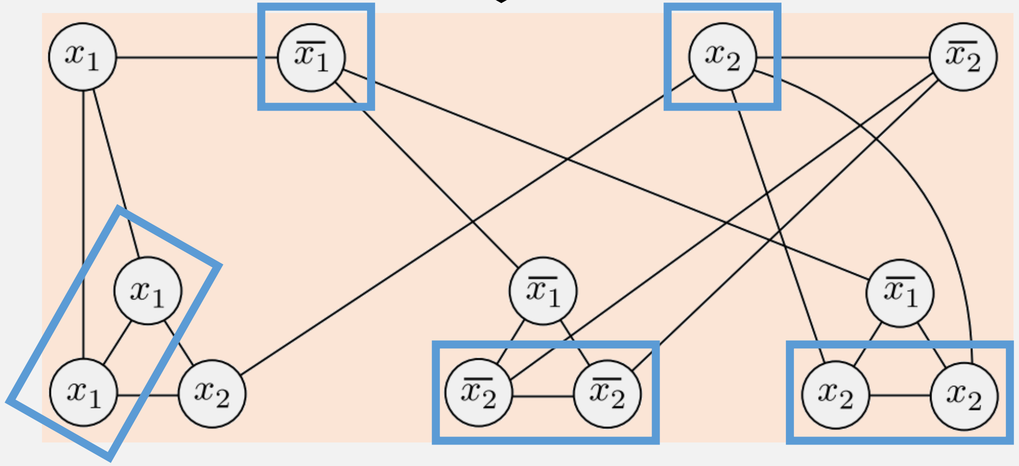
⇒ If satisfying assignment, then there is a k-cover, where $k = m + 2l$

- Nodes in the cover:
  - In each of m var gadgets, choose 1 node corresponding to TRUE literal
  - For each of l clause gadgets, ignore 1 TRUE literal and choose other 2
  - Since there is satisfying assignment, each clause has a TRUE literal
  - Total = $m + 2l$

VERTEX-COVER = $\{\langle G, k\rangle |$ G is an undirected graph that has a k-node vertex cover$\}$

$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

VERTEX-COVER example



- If formula has …
  - $\boldsymbol{m}$ = # variables
  - $\boldsymbol{l}$ = # clauses
- Then graph has …
  - # nodes = $2\boldsymbol{m} + 3\boldsymbol{l}$

Example:
x_1 = FALSE
x_2 = TRUE

⇐ If there is a $\boldsymbol{k} = \boldsymbol{m} + 2\boldsymbol{l}$ cover,

- Then it can only be a $\boldsymbol{k}$-cover as described on the last slide …
  - 1 node from each of “var” gadgets
  - 2 nodes from each “clause” gadget
- Which means that input has satisfying assignment:
  - x_i = TRUE if node x_i from x_i gadget is in cover set
  - Else x_i = FALSE

VERTEX-COVER = $\{\langle G, k\rangle \mid G$ is an undirected graph that has a k-node vertex cover$\}$

Quiz 11/22