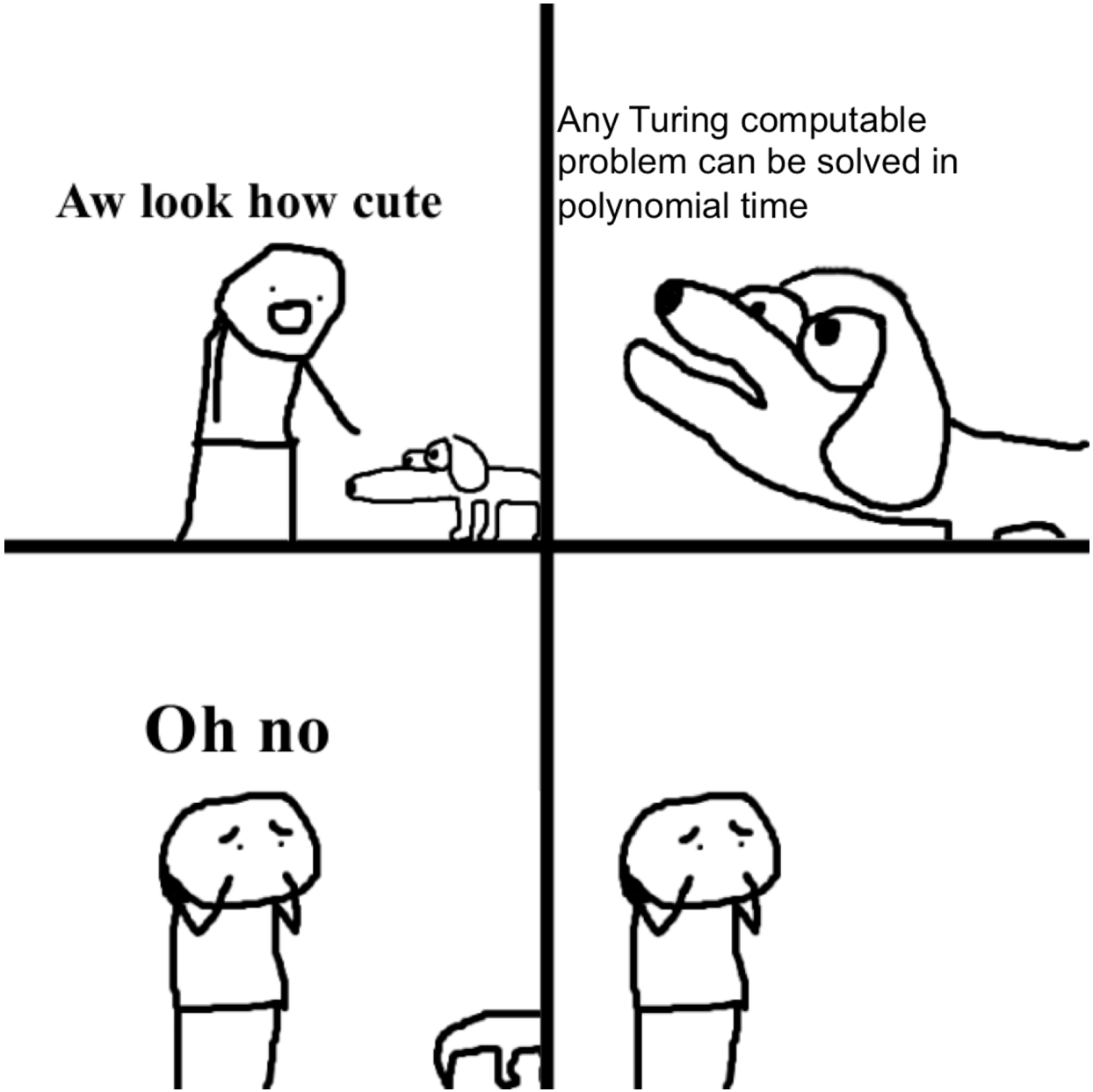
Lecture 7: More NP-Complete Problems



Lecture 7: NP-completeness

Today we will cover:

Lecture 7: NP-completeness

Today we will cover:

- The 3-SAT problem.

Lecture 7: NP-completeness

Today we will cover:

- The 3-SAT problem.
- Its NP-hardness as $3\text{-}SAT \leq_p SAT$.

Lecture 7: NP-completeness

Today we will cover:

- The 3-SAT problem.
- Its NP-hardness as $3\text{-}SAT \leq_p SAT$.
- k-SAT.

Lecture 7: NP-completeness

Today we will cover:

- The 3-SAT problem.
- Its NP-hardness as $3\text{-}SAT \leq_p SAT$.
- k-SAT.
- Independent Set.

Lecture 7: NP-completeness

Today we will cover:

- The 3-SAT problem.
- Its NP-hardness as $3\text{-}SAT \leq_p SAT$.
- k-SAT.
- Independent Set.
- (D)HCP.

Lecture 7: NP-completeness

Today we will cover:

- The 3-SAT problem.
- Its NP-hardness as $3\text{-}SAT \leq_p SAT$.
- k-SAT.
- Independent Set.
- (D)HCP.
- Vertex Cover.

Let's recap what we saw

Which of the following are correct?

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$.

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$.
- SAT has the highest time complexity among all NP problems.

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$.
- SAT has the highest time complexity among all NP problems.
- If SAT can be solved in polynomial time so is TSDP.

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$.
- SAT has the highest time complexity among all NP problems.
- If SAT can be solved in polynomial time so is TSDP.
- SAT is EXPTIME-complete.

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$.
- SAT has the highest time complexity among all NP problems.
- If SAT can be solved in polynomial time so is TSDP.
- SAT is EXPTIME-complete.
- If we can decide if a number is prime in p-time then we can solve SAT in p-time.

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$.
- SAT has the highest time complexity among all NP problems.
- If SAT can be solved in polynomial time so is TSDP.
- SAT is EXPTIME-complete.
- If we can decide if a number is prime in p-time then we can solve SAT in p-time.
- It is known that $SAT \notin P$.

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$.
- SAT has the highest time complexity among all NP problems.
- If SAT can be solved in polynomial time so is TSDP.
- SAT is EXPTIME-complete.
- If we can decide if a number is prime in p-time then we can solve SAT in p-time.
- It is known that $SAT \notin P$.
- All problems are either in P or NP-complete.

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$.
- SAT has the highest time complexity among all NP problems.
- If SAT can be solved in polynomial time so is TSDP.
- SAT is EXPTIME-complete.
- If we can decide if a number is prime in p-time then we can solve SAT in p-time.
- It is known that $SAT \notin P$.
- All problems are either in P or NP-complete.
- All problems are either in P or NP-hard.

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$.
- SAT has the highest time complexity among all NP problems.
- If SAT can be solved in polynomial time so is TSDP.
- SAT is EXPTIME-complete.
- If we can decide if a number is prime in p-time then we can solve SAT in p-time.
- It is known that $SAT \notin P$.
- All problems are either in P or NP-complete.
- All problems are either in P or NP-hard.

Answer on Mentimeter:  .

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$.

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$. No (e.g., some problems require super-exponential runtime.)
- SAT has the highest time complexity among all NP problems.

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$. No (e.g., some problems require super-exponential runtime.)
- SAT has the highest time complexity among all NP problems. Probably not. NP-hardness does not dictate the problem's actual complexity.
- If SAT can be solved in polynomial time so is TSDP.

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$. No (e.g., some problems require super-exponential runtime.)
- SAT has the highest time complexity among all NP problems. Probably not. NP-hardness does not dictate the problem's actual complexity.
- If SAT can be solved in polynomial time so is TSDP. Yes, as we've seen that TSDP$\in NP$.
- SAT is EXPTIME-complete.

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$. No (e.g., some problems require super-exponential runtime.)
- SAT has the highest time complexity among all NP problems. Probably not. NP-hardness does not dictate the problem's actual complexity.
- If SAT can be solved in polynomial time so is TSDP. Yes, as we've seen that TSDP$\in NP$.
- SAT is EXPTIME-complete. Probably not. This would imply that NP=EXPTIME which would be surprising.
- If we can decide if a number is prime in p-time then we can solve SAT in p-time.

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$. No (e.g., some problems require super-exponential runtime.)
- SAT has the highest time complexity among all NP problems. Probably not. NP-hardness does not dictate the problem's actual complexity.
- If SAT can be solved in polynomial time so is TSDP. Yes, as we've seen that TSDP$\in NP$.
- SAT is EXPTIME-complete. Probably not. This would imply that NP=EXPTIME which would be surprising.
- If we can decide if a number is prime in p-time then we can solve SAT in p-time. Probably not. Primality checking is known to be in P, so SAT$\in P$ is equivalent to $P = NP$.
- It is known that $SAT \notin P$.

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$. No (e.g., some problems require super-exponential runtime.)
- SAT has the highest time complexity among all NP problems. Probably not. NP-hardness does not dictate the problem's actual complexity.
- If SAT can be solved in polynomial time so is TSDP. Yes, as we've seen that TSDP$\in NP$.
- SAT is EXPTIME-complete. Probably not. This would imply that NP=EXPTIME which would be surprising.
- If we can decide if a number is prime in p-time then we can solve SAT in p-time. Probably not. Primality checking is known to be in P, so SAT$\in P$ is equivalent to $P = NP$.
- It is known that $SAT \notin P$. No. This would imply $P \neq NP$ which is an open question.
- All decidable problems are either in P or NP-complete.

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$. No (e.g., some problems require super-exponential runtime.)
- SAT has the highest time complexity among all NP problems. Probably not. NP-hardness does not dictate the problem's actual complexity.
- If SAT can be solved in polynomial time so is TSDP. Yes, as we've seen that TSDP$\in NP$.
- SAT is EXPTIME-complete. Probably not. This would imply that NP=EXPTIME which would be surprising.
- If we can decide if a number is prime in p-time then we can solve SAT in p-time. Probably not. Primality checking is known to be in P, so SAT$\in P$ is equivalent to $P = NP$.
- It is known that $SAT \notin P$. No. This would imply $P \neq NP$ which is an open question.
- All decidable problems are either in P or NP-complete. No, e.g., some problems are not in NP.
- All decidable problems are either in P or NP-hard.

Let's recap what we saw

Which of the following are correct?

- For any decidable $L \subseteq \Sigma^*$, $L \leq_p SAT$. No (e.g., some problems require super-exponential runtime.)
- SAT has the highest time complexity among all NP problems. Probably not. NP-hardness does not dictate the problem's actual complexity.
- If SAT can be solved in polynomial time so is TSDP. Yes, as we've seen that TSDP$\in NP$.
- SAT is EXPTIME-complete. Probably not. This would imply that NP=EXPTIME which would be surprising.
- If we can decide if a number is prime in p-time then we can solve SAT in p-time. Probably not. Primality checking is known to be in P, so SAT$\in P$ is equivalent to $P = NP$.
- It is known that $SAT \notin P$. No. This would imply $P \neq NP$ which is an open question.
- All decidable problems are either in P or NP-complete. No, e.g., some problems are not in NP.
- All decidable problems are either in P or NP-hard. Probably not. Some problems (e.g., graph isomorphism) are conjectured to be in $NP \setminus P$ but not NP-hard.

k-SAT

k-SAT is similar to SAT, but each clause may have at most k literals. E.g.,

$$(p \vee \overline{p} \vee q) \wedge (p \vee q \vee r) \wedge (\overline{p} \vee \overline{q} \vee \overline{r}) \wedge (p \vee \overline{q} \vee r)$$

is a 3-SAT formula.

k-SAT

k-SAT is similar to SAT, but each clause may have at most k literals. E.g.,

$$(p \vee \overline{p} \vee q) \wedge (p \vee q \vee r) \wedge (\overline{p} \vee \overline{q} \vee \overline{r}) \wedge (p \vee \overline{q} \vee r)$$

is a 3-SAT formula.

Yes-instance: if it is satisfiable.
No-instance: otherwise.

SAT $\leq_p$ 3SAT

Idea

$$(l_1 \vee l_2 \vee l_3 \vee l_4)$$

$\mathbf{SAT} \leq_p \mathbf{3SAT}$

Idea

$$(l_1 \vee l_2 \vee l_3 \vee l_4) \mapsto (l_1 \vee l_2 \vee x) \wedge (\overline{x} \vee l_3 \vee l_4)$$

SAT $\leq_p$ **3SAT**, More formally

Let C be a clause of ϕ.

SAT $\leq_p$ **3SAT**, More formally

Let C be a clause of ϕ.

SAT $\leq_p$ 3SAT, More formally

Let C be a clause of ϕ.

- If C has at most three literals let $g(C) = C$.

Suppose C has $m > 3$ literals,

$$C = (l_1 \vee l_2 \vee \ldots \vee l_m)$$

SAT $\leq_p$ 3SAT, More formally

Let C be a clause of ϕ.

- If C has at most three literals let $g(C) = C$.

Suppose C has $m > 3$ literals,

$$C = (l_1 \vee l_2 \vee \ldots \vee l_m)$$

Add new variables (unique to C) $p_1, p_2, \ldots, p_{m-1}$.

SAT $\leq_p$ 3SAT, More formally

Let C be a clause of ϕ.

- If C has at most three literals let $g(C) = C$.

Suppose C has $m > 3$ literals,

$$C = (l_1 \vee l_2 \vee \ldots \vee l_m)$$

Add new variables (unique to C) $p_1, p_2, \ldots, p_{m-1}$.
Replace C by

$$g(C) = (l_1 \vee p_1) \wedge (\overline{p_1} \vee l_2 \vee p_2) \wedge (\overline{p_2} \vee l_3 \vee p_3) \wedge$$
$$\ldots \wedge (\overline{p_{i-1}} \vee l_i \vee p_i) \wedge$$
$$\ldots \wedge (\overline{p_{m-2}} \vee l_{m-1} \vee p_{m-1}) \wedge (\overline{p_{m-1}} \vee l_m)$$

$\mathbf{SAT} \leq_p \mathbf{3SAT}$, More formally

Let C be a clause of ϕ.

- If C has at most three literals let $g(C) = C$.

Suppose C has $m > 3$ literals,

$$C = (l_1 \vee l_2 \vee \ldots \vee l_m)$$

Add new variables (unique to C) $p_1, p_2, \ldots, p_{m-1}$.
Replace C by

$$\begin{gathered} g(C) = (l_1 \vee p_1) \wedge (\overline{p_1} \vee l_2 \vee p_2) \wedge (\overline{p_2} \vee l_3 \vee p_3) \wedge \\ \ldots \wedge (\overline{p_{i-1}} \vee l_i \vee p_i) \wedge \\ \ldots \wedge (\overline{p_{m-2}} \vee l_{m-1} \vee p_{m-1}) \wedge (\overline{p_{m-1}} \vee l_m) \end{gathered}$$

Intuitively, if $l_i = T$ we can set $p_1 = \ldots = p_{i-1} = T$ and $p_i = \ldots = p_{m-1} = F$.

$\mathbf{SAT} \leq_p \mathbf{3SAT}$, More formally

Let C be a clause of ϕ.

- If C has at most three literals let $g(C) = C$.

Suppose C has $m > 3$ literals,

$$C = (l_1 \vee l_2 \vee \ldots \vee l_m)$$

Add new variables (unique to C) $p_1, p_2, \ldots, p_{m-1}$.
Replace C by

$$\begin{gathered} g(C) = (l_1 \vee p_1) \wedge (\overline{p_1} \vee l_2 \vee p_2) \wedge (\overline{p_2} \vee l_3 \vee p_3) \wedge \\ \ldots \wedge (\overline{p_{i-1}} \vee l_i \vee p_i) \wedge \\ \ldots \wedge (\overline{p_{m-2}} \vee l_{m-1} \vee p_{m-1}) \wedge (\overline{p_{m-1}} \vee l_m) \end{gathered}$$

Intuitively, if $l_i = T$ we can set $p_1 = \ldots = p_{i-1} = T$ and $p_i = \ldots = p_{m-1} = F$. Time taken is $O(m)$.

Correctness

Rewrite each clause and let result be $g(\phi)$.

Correctness

Rewrite each clause and let result be $g(\phi)$. (This takes $O(|\phi|)$ time.)

Correctness

Rewrite each clause and let result be $g(\phi)$. (This takes $O(|\phi|)$ time.)

If ϕ is a yes-instance of SAT then there is an assignment v with $v(\phi) = T$.

Correctness

Rewrite each clause and let result be $g(\phi)$. (This takes $O(|\phi|)$ time.)

If ϕ is a yes-instance of SAT then there is an assignment v with $v(\phi) = T$.

We need to show that there exists an assignment v' that satisfies $g(\phi)$.

Correctness

Rewrite each clause and let result be $g(\phi)$. (This takes $O(|\phi|)$ time.)

If ϕ is a yes-instance of SAT then there is an assignment v with $v(\phi) = T$.

We need to show that there exists an assignment v' that satisfies $g(\phi)$.

We have $v(C) = T$ for each clause of ϕ. If

$$C = (l_1 \vee l_2 \vee \ldots \vee l_m)$$

then $v(l_i) = T$, for some i with $1 \leq i \leq m$.

Correctness

Rewrite each clause and let result be $g(\phi)$. (This takes $O(|\phi|)$ time.)

If ϕ is a yes-instance of SAT then there is an assignment v with $v(\phi) = T$.

We need to show that there exists an assignment v' that satisfies $g(\phi)$.

We have $v(C) = T$ for each clause of ϕ. If

$$C = (l_1 \vee l_2 \vee \ldots \vee l_m)$$

then $v(l_i) = T$, for some i with $1 \leq i \leq m$.

Set $v'(p_j) = T$ for $j < i - 1$ and $v'(p_j) = F$ for $i - 1 \leq j \leq m$.

Correctness

Rewrite each clause and let result be $g(\phi)$. (This takes $O(|\phi|)$ time.)

If ϕ is a yes-instance of SAT then there is an assignment v with $v(\phi) = T$.

We need to show that there exists an assignment v' that satisfies $g(\phi)$.

We have $v(C) = T$ for each clause of ϕ. If

$$C = (l_1 \vee l_2 \vee \ldots \vee l_m)$$

then $v(l_i) = T$, for some i with $1 \leq i \leq m$.

Set $v'(p_j) = T$ for $j < i - 1$ and $v'(p_j) = F$ for $i - 1 \leq j \leq m$.

Then $v(\overline{p_{i-1}} \vee l_i \vee p_i) = T$, and all other 3-clauses are true under v'.

Hence $v(g(C)) = T$.

Correctness

Rewrite each clause and let result be $g(\phi)$. (This takes $O(|\phi|)$ time.)

If ϕ is a yes-instance of SAT then there is an assignment v with $v(\phi) = T$.

We need to show that there exists an assignment v' that satisfies $g(\phi)$.

We have $v(C) = T$ for each clause of ϕ. If

$$C = (l_1 \vee l_2 \vee \ldots \vee l_m)$$

then $v(l_i) = T$, for some i with $1 \leq i \leq m$.

Set $v'(p_j) = T$ for $j < i - 1$ and $v'(p_j) = F$ for $i - 1 \leq j \leq m$.

Then $v(\overline{p_{i-1}} \vee l_i \vee p_i) = T$, and all other 3-clauses are true under v'.

Hence $v(g(C)) = T$.

$v(g(\phi)) = T$ holds for each clause,

Correctness

Rewrite each clause and let result be $g(\phi)$. (This takes $O(|\phi|)$ time.)

If ϕ is a yes-instance of SAT then there is an assignment v with $v(\phi) = T$.

We need to show that there exists an assignment v' that satisfies $g(\phi)$.

We have $v(C) = T$ for each clause of ϕ. If

$$C = (l_1 \vee l_2 \vee \ldots \vee l_m)$$

then $v(l_i) = T$, for some i with $1 \leq i \leq m$.

Set $v'(p_j) = T$ for $j < i - 1$ and $v'(p_j) = F$ for $i - 1 \leq j \leq m$.

Then $v(\overline{p_{i-1}} \vee l_i \vee p_i) = T$, and all other 3-clauses are true under v'.

Hence $v(g(C)) = T$.

$v(g(\phi)) = T$ holds for each clause, thus $g(\phi)$ is a yes-instance of 3-SAT.

Correctness (Cont.)

Conversely, if $g(\phi)$ is a yes-instance of 3-SAT then there is a assignment v' with $v'(g(\phi)) = T$.

Correctness (Cont.)

Conversely, if $g(\phi)$ is a yes-instance of 3-SAT then there is a assignment v' with $v'(g(\phi)) = T$.

Easy to check that $v(\phi) = T$, so ϕ is a yes-instance of SAT.

k-SAT for other values of k

k-SAT for other values of k

It's easy to generalize our reduction to any $k \geq 3$.

k-SAT for other values of k

It's easy to generalize our reduction to any $k \geq 3$.
Also easy: 1-SAT (basically, a conjunction) is in P.

k-SAT for other values of k

It's easy to generalize our reduction to any $k \geq 3$.
Also easy: 1-SAT (basically, a conjunction) is in P.
What about 2-SAT?

k-SAT for other values of k

It's easy to generalize our reduction to any $k \geq 3$.
Also easy: 1-SAT (basically, a conjunction) is in P.
What about 2-SAT?

Theorem

$2-SAT \in P$.

k-SAT for other values of k

It's easy to generalize our reduction to any $k \geq 3$.
Also easy: 1-SAT (basically, a conjunction) is in P.
What about 2-SAT?

Theorem

$2-SAT \in P$.

Proof sketch (Krom 1967).

If each variable appears only in positive (x_i) or only in negative ($\overline{x_i}$) form, the formula is satisfiable (why?).

k-SAT for other values of k

It's easy to generalize our reduction to any $k \geq 3$.
Also easy: 1-SAT (basically, a conjunction) is in P.
What about 2-SAT?

Theorem

$2-SAT \in P$.

Proof sketch (Krom 1967).

If each variable appears only in positive (x_i) or only in negative ($\overline{x_i}$) form, the formula is satisfiable (why?).
Assume that ϕ has two clauses $(a \vee b), (\overline{b} \vee c)$.

k-SAT for other values of k

It's easy to generalize our reduction to any $k \geq 3$.
Also easy: 1-SAT (basically, a conjunction) is in P.
What about 2-SAT?

Theorem

$2-SAT \in P$.

Proof sketch (Krom 1967).

If each variable appears only in positive (x_i) or only in negative ($\overline{x_i}$) form, the formula is satisfiable (why?).
Assume that ϕ has two clauses $(a \vee b), (\overline{b} \vee c)$.
Replace both clauses with $(a \vee c)$.

k-SAT for other values of k

It's easy to generalize our reduction to any $k \geq 3$.
Also easy: 1-SAT (basically, a conjunction) is in P.
What about 2-SAT?

Theorem

$2-SAT \in P$.

Proof sketch (Krom 1967).

If each variable appears only in positive (x_i) or only in negative ($\overline{x_i}$) form, the formula is satisfiable (why?).
Assume that ϕ has two clauses $(a \vee b), (\overline{b} \vee c)$.
Replace both clauses with $(a \vee c)$.
Now we have a clause with one fewer "conflict", and can repeat.

k-SAT for other values of k

It's easy to generalize our reduction to any $k \geq 3$.
Also easy: 1-SAT (basically, a conjunction) is in P.
What about 2-SAT?

Theorem

$2-SAT \in P$.

Proof sketch (Krom 1967).

If each variable appears only in positive (x_i) or only in negative ($\overline{x_i}$) form, the formula is satisfiable (why?).
Assume that ϕ has two clauses $(a \vee b), (\overline{b} \vee c)$.
Replace both clauses with $(a \vee c)$.
Now we have a clause with one fewer "conflict", and can repeat.
Runs in p-time (although much faster solutions are known!).

Why do we care about 3-SAT?

Why do we care about 3-SAT?

By definition, for any NP-hard problem A there exists a reduction $SAT \leq_p A$.

Why do we care about 3-SAT?

By definition, for any NP-hard problem A there exists a reduction $SAT \leq_p A$.

However, finding a reduction from B to A is "easier" if B is simpler.

Why do we care about 3-SAT?

By definition, for any NP-hard problem A there exists a reduction $SAT \leq_p A$.

However, finding a reduction from B to A is "easier" if B is simpler.

For example, we can prove hardness with $3-SAT \leq_p A$, which allows us to assume that every clause has at most three literals.

Why do we care about 3-SAT?

By definition, for any NP-hard problem A there exists a reduction $SAT \leq_p A$.

However, finding a reduction from B to A is "easier" if B is simpler.

For example, we can prove hardness with $3-SAT \leq_p A$, which allows us to assume that every clause has at most three literals.

We will use this to show that Independent Set (IS) and HCP are NP-complete.

Independent Set

Independent Set

Input: An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.

Independent Set

Input: An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.

Yes-instance: if there is a set of vertices $I \subseteq V$ of size $|I| = k$ without edges between them (i.e., $(I \times I) \cap E = \emptyset$).

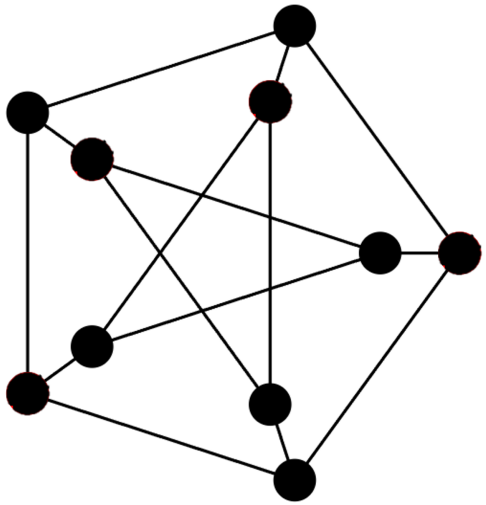
No-instance: otherwise.

Independent Set

Input: An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.

Yes-instance: if there is a set of vertices $I \subseteq V$ of size $|I| = k$ without edges between them (i.e., $(I \times I) \cap E = \emptyset$).

No-instance: otherwise.



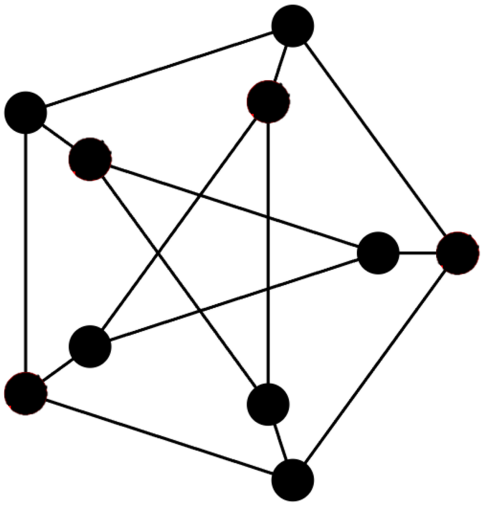
Is there a $k = 4$-sized IS?

Independent Set

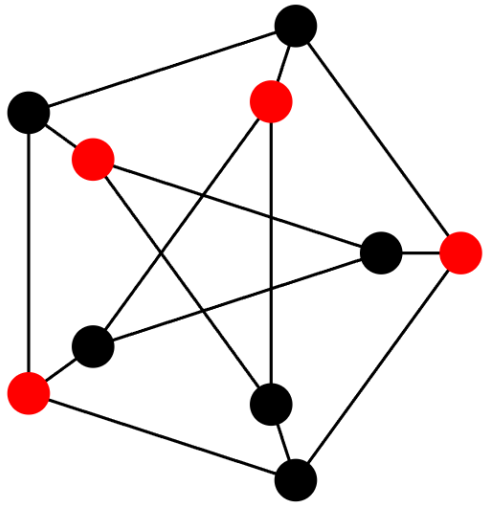
Input: An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.

Yes-instance: if there is a set of vertices $I \subseteq V$ of size $|I| = k$ without edges between them (i.e., $(I \times I) \cap E = \emptyset$).

No-instance: otherwise.



Is there a $k = 4$-sized IS?



Independent Set is *NPC* Part I: $IS \in NP$

A very simple algorithm:

Independent Set is *NPC* Part I: $IS \in NP$

A very simple algorithm:

(1) Guess a set of k vertices.

Independent Set is *NPC* Part I: $IS \in NP$

A very simple algorithm:

(1) Guess a set of k vertices.

(2) Accept if there are no edges between the vertices.

Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

We are given a 3-CNF formula ϕ, and need to construct an *IS* instance.

Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

We are given a 3-CNF formula ϕ, and need to construct an *IS* instance.

Intuitively, instead of thinking of assignments to each of the variables, for each clause we want to choose a "representative" literal that satisfies it.

Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

We are given a 3-CNF formula ϕ, and need to construct an *IS* instance.

Intuitively, instead of thinking of assignments to each of the variables, for each clause we want to choose a "representative" literal that satisfies it.

For example, given

$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \quad ,$$

Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

We are given a 3-CNF formula ϕ, and need to construct an *IS* instance.

Intuitively, instead of thinking of assignments to each of the variables, for each clause we want to choose a "representative" literal that satisfies it.

For example, given

$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \quad ,$$

which is satisfiable using $x_1 = T, x_2 = x_3 = F$,

Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

We are given a 3-CNF formula ϕ, and need to construct an *IS* instance.

Intuitively, instead of thinking of assignments to each of the variables, for each clause we want to choose a "representative" literal that satisfies it.

For example, given

$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \quad ,$$

which is satisfiable using $x_1 = T, x_2 = x_3 = F$, we can choose x_1 from the first clause, $\overline{x_2}$ from the second, and $\overline{x_3}$ from the third and fourth.

Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

We are given a 3-CNF formula ϕ, and need to construct an *IS* instance.

Intuitively, instead of thinking of assignments to each of the variables, for each clause we want to choose a "representative" literal that satisfies it.

For example, given

$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \quad ,$$

which is satisfiable using $x_1 = T, x_2 = x_3 = F$, we can choose x_1 from the first clause, $\overline{x_2}$ from the second, and $\overline{x_3}$ from the third and fourth.

If we are able to choose literals without picking a variable and its negation, we can satisfy the formula (e.g., using $x_1 = T, x_2 = x_3 = F$ in this example).

Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

We need to translate this into a graph.

Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

We need to translate this into a graph.
Assume that the input is:

$$\phi = \bigwedge_{1 \leq i \leq m} (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}),$$

where each $\ell_{i,j}$ is a literal (i.e., $\ell_{i,j} \in \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$).

Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

We need to translate this into a graph.
Assume that the input is:

$$\phi = \bigwedge_{1 \leq i \leq m} (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}),$$

where each $\ell_{i,j}$ is a literal (i.e., $\ell_{i,j} \in \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$).

We build the graph over the vertices
$V = \{\ell_{i,j} \mid i \in \{1, \ldots, m\}, j \in \{1, 2, 3\}\}$.

Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

We need to translate this into a graph.
Assume that the input is:

$$\phi = \bigwedge_{1 \leq i \leq m} (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}),$$

where each $\ell_{i,j}$ is a literal (i.e., $\ell_{i,j} \in \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$).

We build the graph over the vertices
$V = \{\ell_{i,j} \mid i \in \{1, \ldots, m\}, j \in \{1, 2, 3\}\}$.

Each clause would be a triangle $\{\{\ell_{i,j}, \ell_{i,(j+1) \mod 3} \mid j \in \{1, 2, 3\}\}\}$.

Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

We need to translate this into a graph.
Assume that the input is:

$$\phi = \bigwedge_{1 \leq i \leq m} (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}),$$

where each $\ell_{i,j}$ is a literal (i.e., $\ell_{i,j} \in \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$).

We build the graph over the vertices
$V = \{\ell_{i,j} \mid i \in \{1, \ldots, m\}, j \in \{1, 2, 3\}\}$.

Each clause would be a triangle $\{\{\ell_{i,j}, \ell_{i,(j+1) \mod 3} \mid j \in \{1, 2, 3\}\}\}$.

In addition, for each variable x that appears both as is and negated:
$\ell_{i,j} = x, \ell_{i',j'} = \overline{x}$

Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

We need to translate this into a graph.
Assume that the input is:

$$\phi = \bigwedge_{1 \leq i \leq m} (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}),$$

where each $\ell_{i,j}$ is a literal (i.e., $\ell_{i,j} \in \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$).

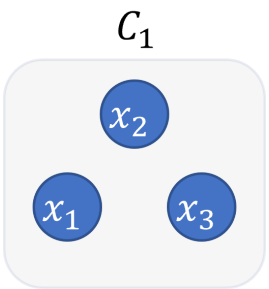
We build the graph over the vertices
$V = \{\ell_{i,j} \mid i \in \{1, \ldots, m\}, j \in \{1, 2, 3\}\}$.

Each clause would be a triangle $\{\{\ell_{i,j}, \ell_{i,(j+1) \mod 3} \mid j \in \{1, 2, 3\}\}\}$.

In addition, for each variable x that appears both as is and negated:
$\ell_{i,j} = x, \ell_{i',j'} = \overline{x}$ we add an edge $\{\ell_{i,j}, \ell_{i',j'}\}$.

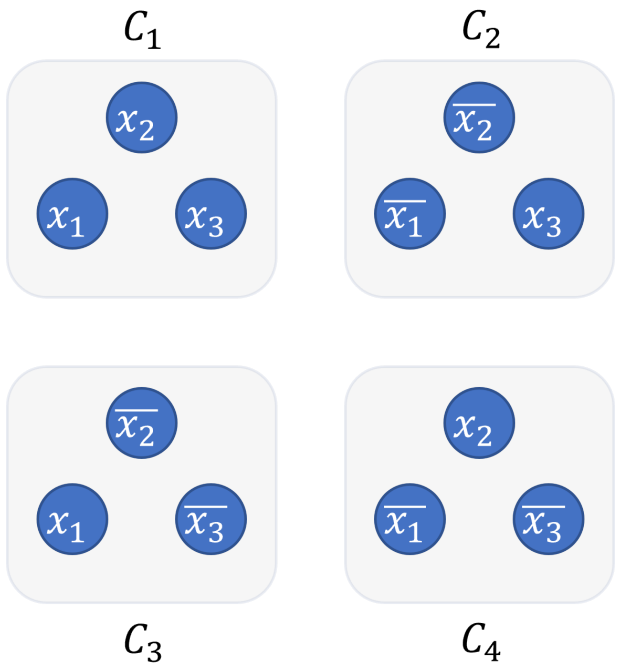
Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

$$\overbrace{(x_1 \vee x_2 \vee x_3)}^{C_1} \wedge \overbrace{(\overline{x_1} \vee \overline{x_2} \vee x_3)}^{C_2} \wedge \overbrace{(x_1 \vee \overline{x_2} \vee \overline{x_3})}^{C_3} \wedge \overbrace{(\overline{x_1} \vee x_2 \vee \overline{x_3})}^{C_4} \quad ,$$



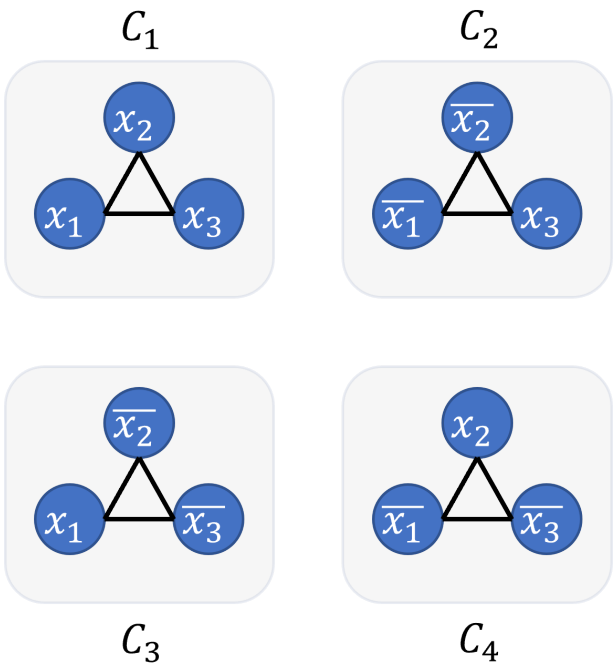
Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

$$\overbrace{(x_1 \vee x_2 \vee x_3)}^{C_1} \wedge \overbrace{(\overline{x_1} \vee \overline{x_2} \vee x_3)}^{C_2} \wedge \overbrace{(x_1 \vee \overline{x_2} \vee \overline{x_3})}^{C_3} \wedge \overbrace{(\overline{x_1} \vee x_2 \vee \overline{x_3})}^{C_4} \quad ,$$



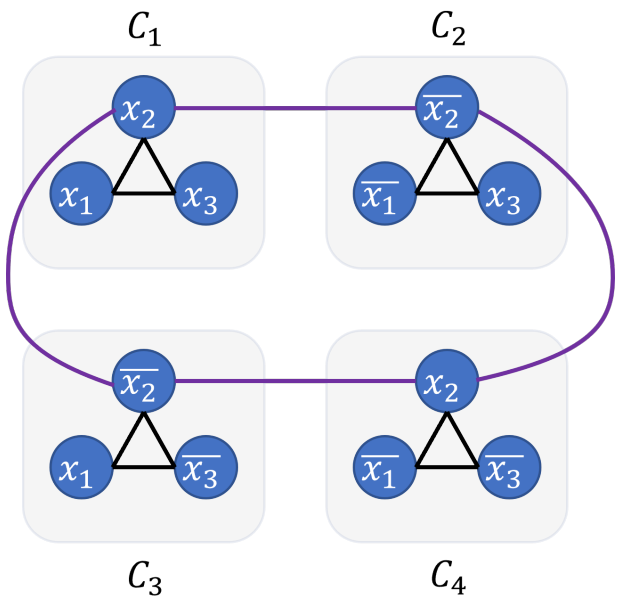
Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

$$\overbrace{(x_1 \vee x_2 \vee x_3)}^{C_1} \wedge \overbrace{(\overline{x_1} \vee \overline{x_2} \vee x_3)}^{C_2} \wedge \overbrace{(x_1 \vee \overline{x_2} \vee \overline{x_3})}^{C_3} \wedge \overbrace{(\overline{x_1} \vee x_2 \vee \overline{x_3})}^{C_4} \quad ,$$



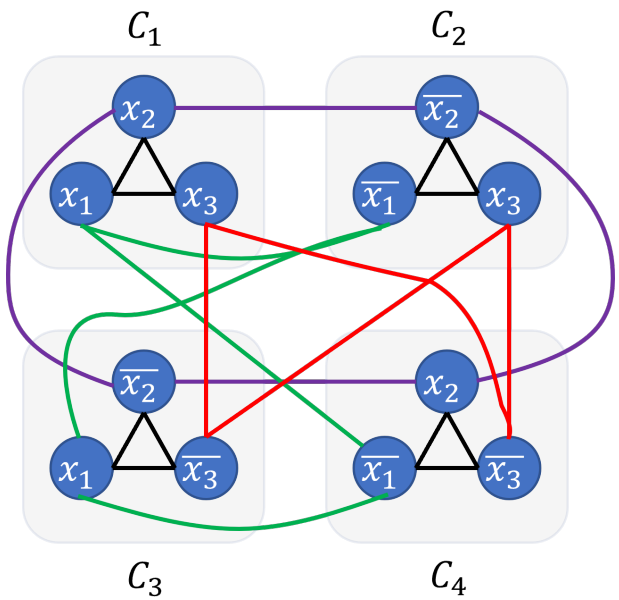
Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

$$\overbrace{(x_1 \vee x_2 \vee x_3)}^{C_1} \wedge \overbrace{(\overline{x_1} \vee \overline{x_2} \vee x_3)}^{C_2} \wedge \overbrace{(x_1 \vee \overline{x_2} \vee \overline{x_3})}^{C_3} \wedge \overbrace{(\overline{x_1} \vee x_2 \vee \overline{x_3})}^{C_4} \quad ,$$



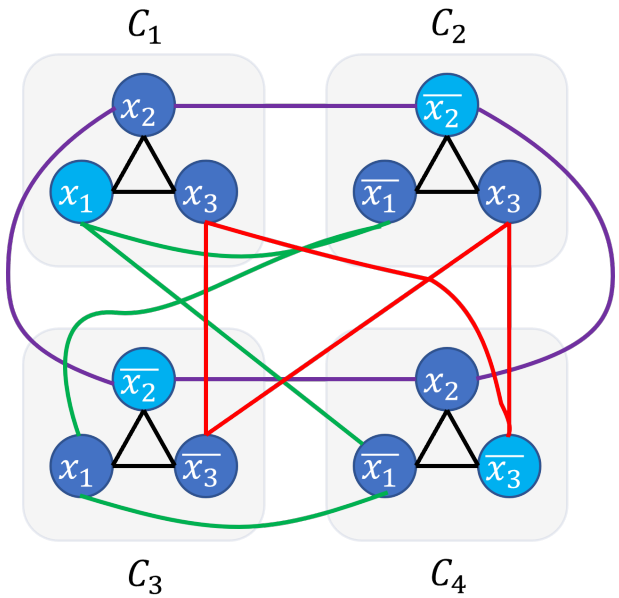
Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

$$\overbrace{(x_1 \vee x_2 \vee x_3)}^{C_1} \wedge \overbrace{(\overline{x_1} \vee \overline{x_2} \vee x_3)}^{C_2} \wedge \overbrace{(x_1 \vee \overline{x_2} \vee \overline{x_3})}^{C_3} \wedge \overbrace{(\overline{x_1} \vee x_2 \vee \overline{x_3})}^{C_4} \quad ,$$



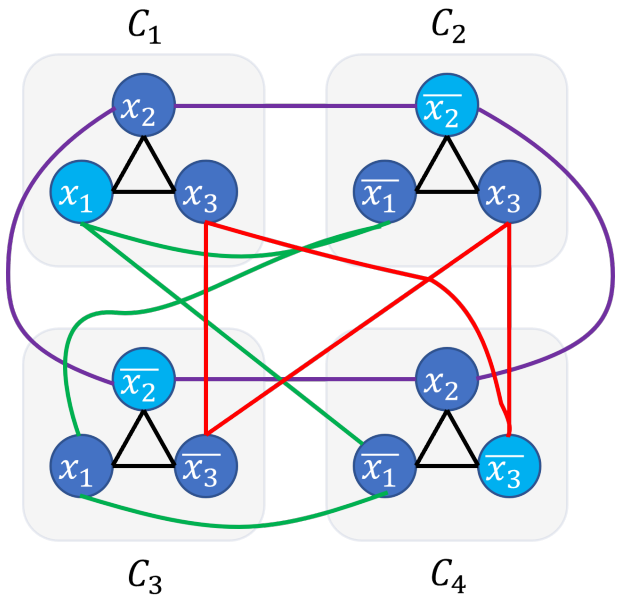
Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

$$\overbrace{(x_1 \vee x_2 \vee x_3)}^{C_1} \wedge \overbrace{(\overline{x_1} \vee \overline{x_2} \vee x_3)}^{C_2} \wedge \overbrace{(x_1 \vee \overline{x_2} \vee \overline{x_3})}^{C_3} \wedge \overbrace{(\overline{x_1} \vee x_2 \vee \overline{x_3})}^{C_4} \quad ,$$



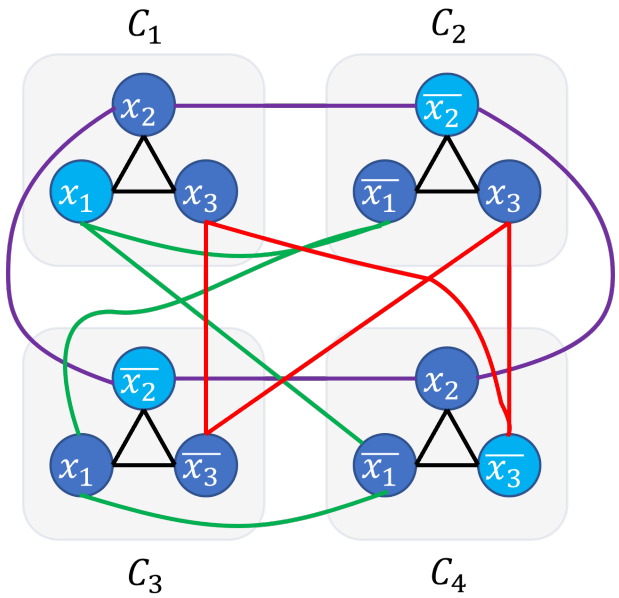
Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

$$\overbrace{(x_1 \vee x_2 \vee x_3)}^{C_1} \wedge \overbrace{(\overline{x_1} \vee \overline{x_2} \vee x_3)}^{C_2} \wedge \overbrace{(x_1 \vee \overline{x_2} \vee \overline{x_3})}^{C_3} \wedge \overbrace{(\overline{x_1} \vee x_2 \vee \overline{x_3})}^{C_4} \quad ,$$



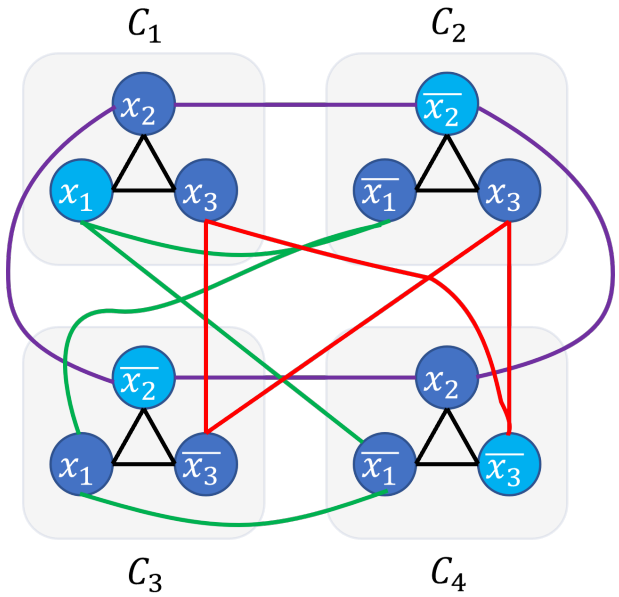
We found an independent set with m vertices, giving us the assignment $v(x_1) = T, v(x_2) = v(x_3) = F$.

Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*



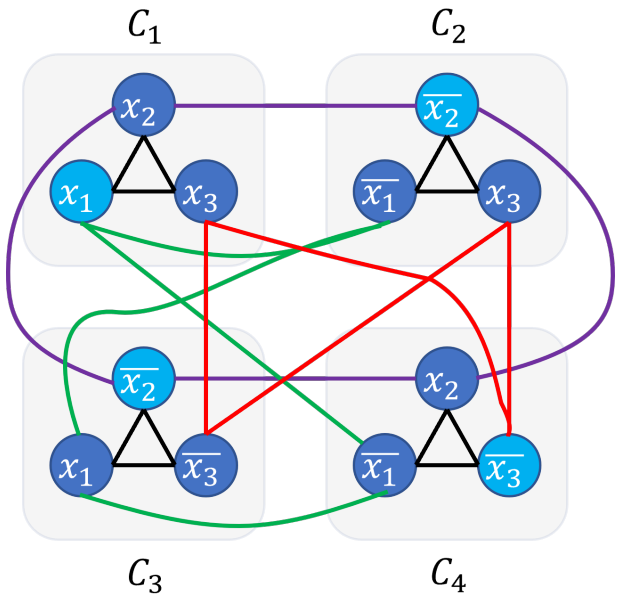
(Not a formal proof!) We have that an IS of size m corresponds to choosing a T representative literal for each clause and thus to a valid assignment for the variables.

Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*



(Not a formal proof!) We have that an IS of size m corresponds to choosing a T representative literal for each clause and thus to a valid assignment for the variables.
We need to analyze the complexity.

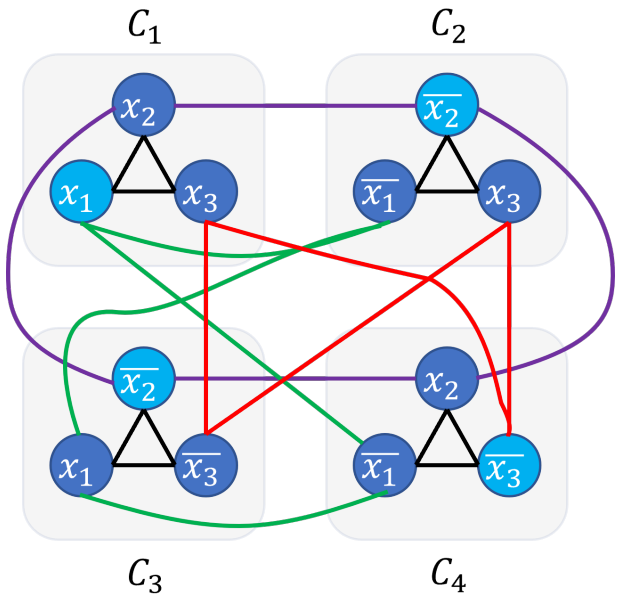
Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*



(Not a formal proof!) We have that an IS of size m corresponds to choosing a T representative literal for each clause and thus to a valid assignment for the variables.

We need to analyze the complexity. What is the resulting graph size?

Independent Set is *NPC* Part II: 3-*SAT* $\leq_p$ *IS*

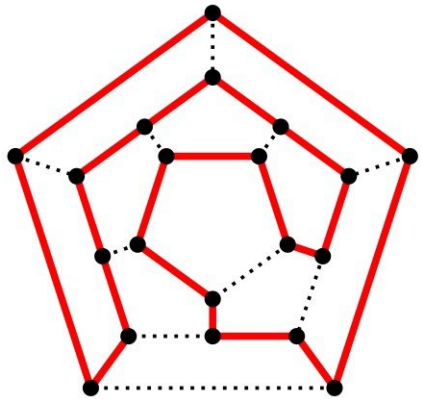


(Not a formal proof!) We have that an IS of size m corresponds to choosing a T representative literal for each clause and thus to a valid assignment for the variables.

We need to analyze the complexity. What is the resulting graph size? We get $3m$ vertices, and $O(m^2)$ edges, so the reduction is polynomial.

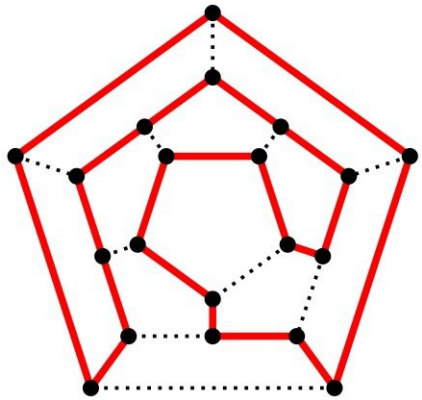
HCP is *NPC* Part I: 3-*SAT* $\leq_p$ *DHCP*

Reminder: in the HCP problem we get an (undirected) graph and need to determine if it has a Hamiltonian circuit.



HCP is *NPC* Part I: 3-$SAT \leq_p$ *DHCP*

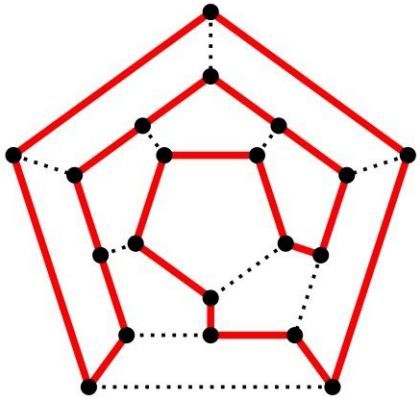
Reminder: in the HCP problem we get an (undirected) graph and need to determine if it has a Hamiltonian circuit.



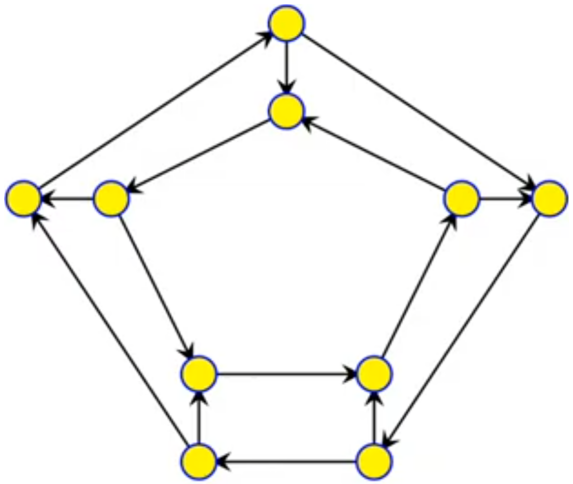
Instead of directly showing that 3-$SAT \leq_p$ *HCP*, we will go through DHCP (Directed HCP) and show (I) 3-$SAT \leq_p$ *DHCP* and (II) *DHCP* $\leq_p$ *HCP*.

HCP is *NPC* Part I: 3-$SAT \leq_p$ *DHCP*

Reminder: in the HCP problem we get an (undirected) graph and need to determine if it has a Hamiltonian circuit.

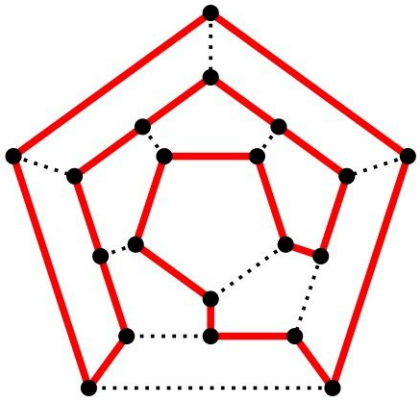


Instead of directly showing that 3-$SAT \leq_p$ *HCP*, we will go through DHCP (Directed HCP) and show (I) 3-$SAT \leq_p$ *DHCP* and (II) *DHCP* $\leq_p$ *HCP*.

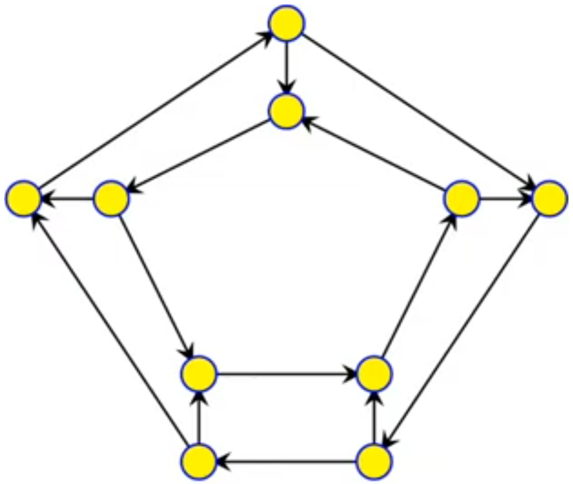


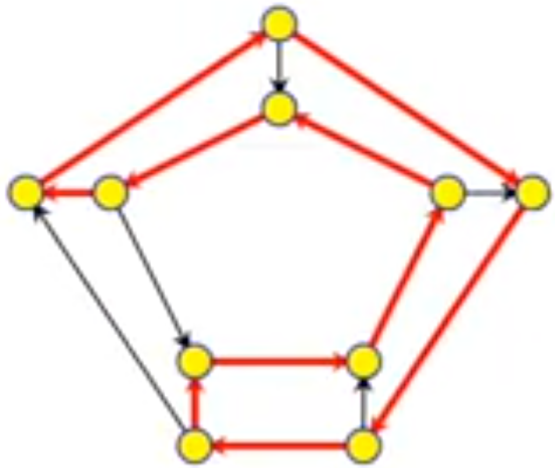
HCP is *NPC* Part I: 3-$SAT \leq_p$ *DHCP*

Reminder: in the HCP problem we get an (undirected) graph and need to determine if it has a Hamiltonian circuit.



Instead of directly showing that 3-$SAT \leq_p$ *HCP*, we will go through DHCP (Directed HCP) and show (I) 3-$SAT \leq_p$ *DHCP* and (II) *DHCP* $\leq_p$ *HCP*.





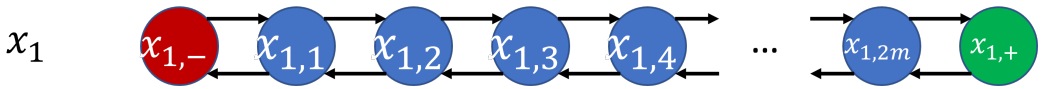
HCP is *NPC* Part I: 3-*SAT* $\leq_p$ *DHCP*

We start from a 3-CNF formula ϕ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$ and construct a graph.

HCP is *NPC* Part I: 3-*SAT* $\leq_p$ *DHCP*

We start from a 3-CNF formula ϕ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$ and construct a graph.

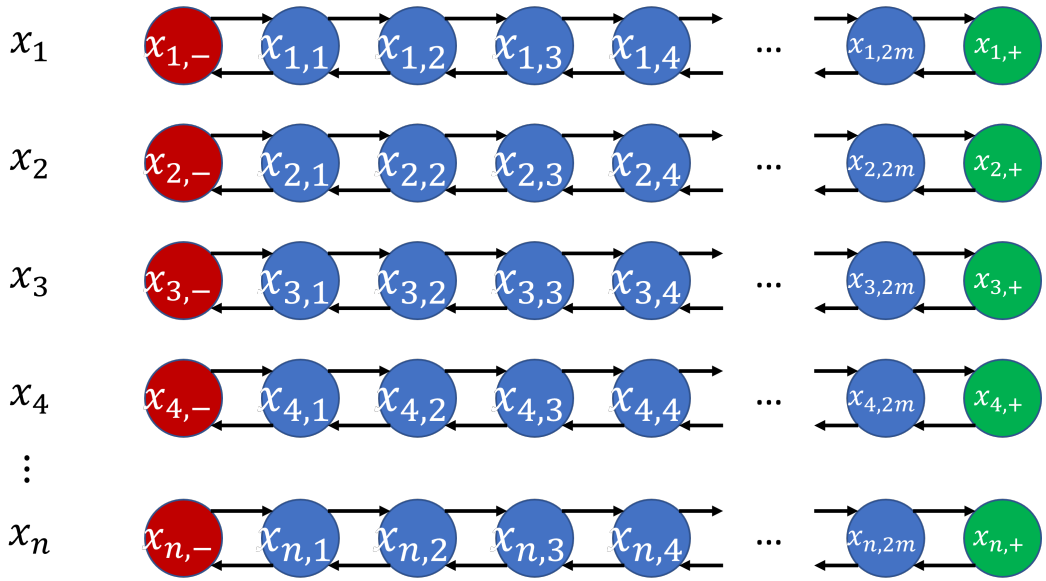
Basic graph: (1) A bi-directional path of length $2m+2$ for each variable.



HCP is *NPC* Part I: 3-*SAT* $\leq_p$ *DHCP*

We start from a 3-CNF formula ϕ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$ and construct a graph.

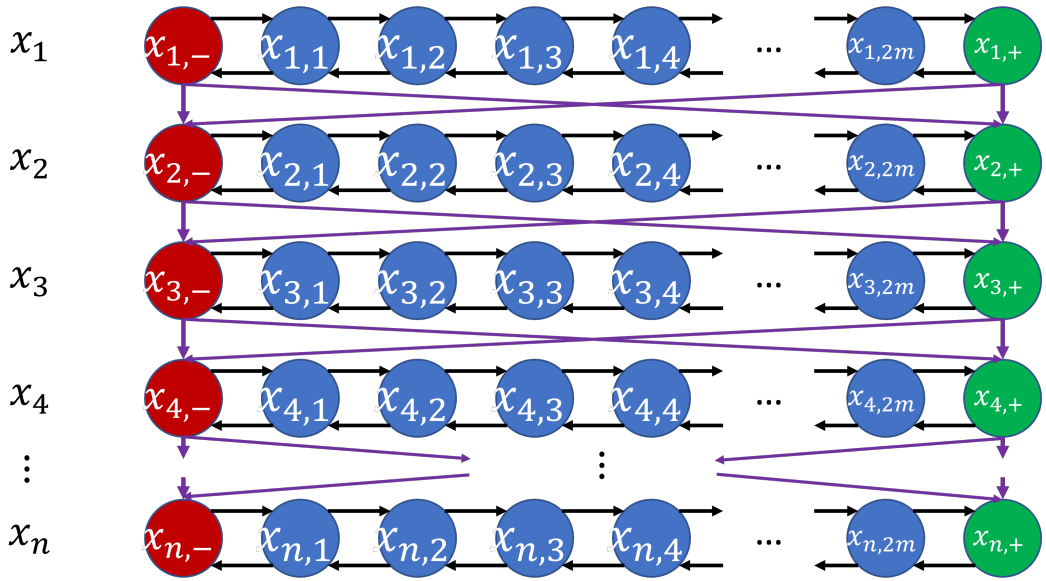
Basic graph: (1) A bi-directional path of length $2m+2$ for each variable.



HCP is *NPC* Part I: 3-*SAT* $\leq_p$ *DHCP*

We start from a 3-CNF formula ϕ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$ and construct a graph.

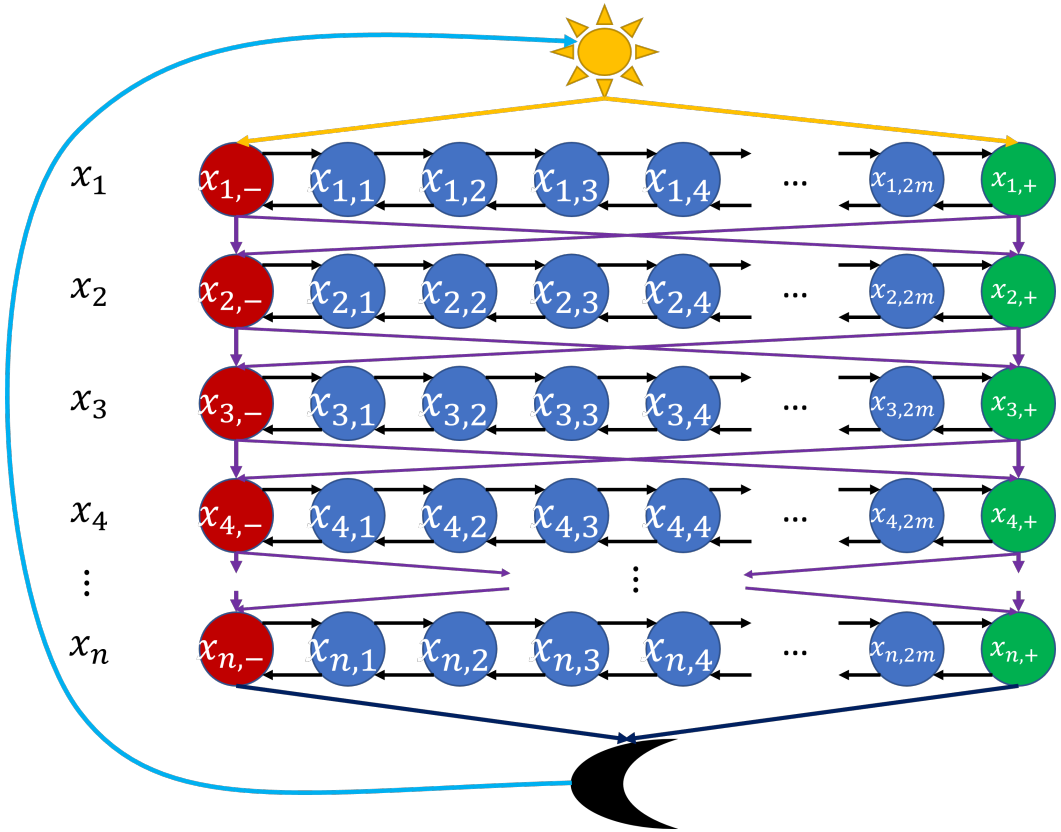
Basic graph: (2) Connect consecutive path ends.



HCP is *NPC* Part I: 3-*SAT* $\leq_p$ *DHCP*

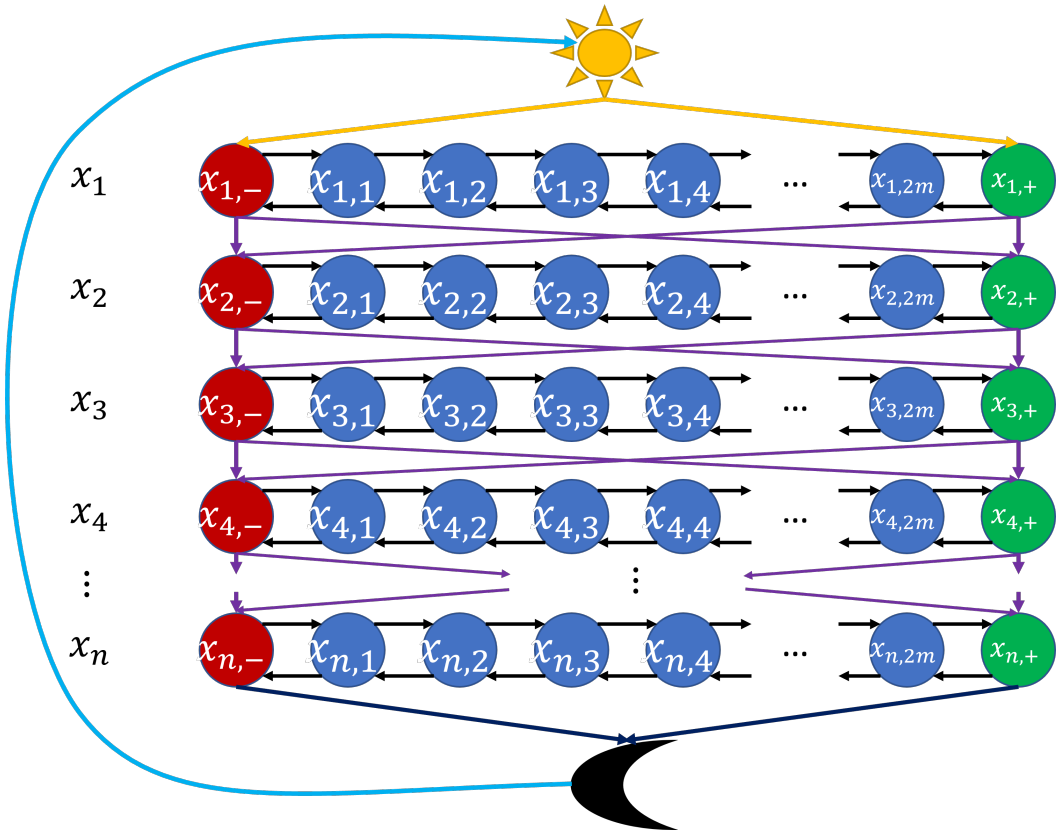
We start from a 3-CNF formula ϕ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$ and construct a graph.

Basic graph: (3) Add a "sun" and "moon" vertices.



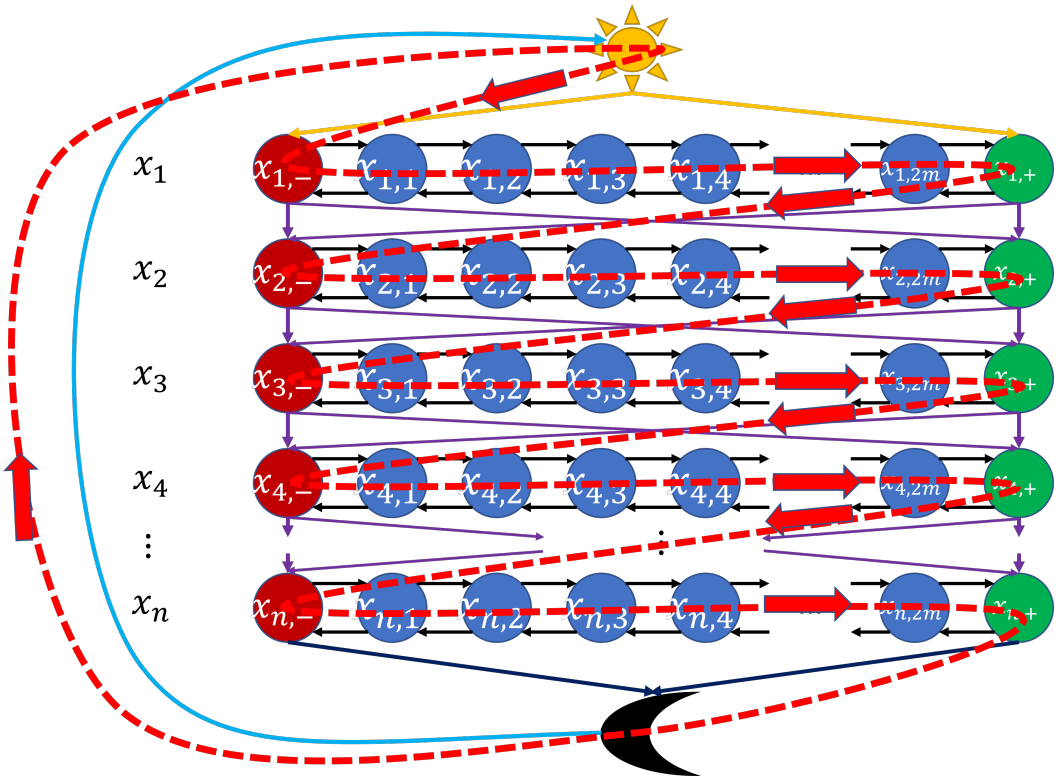
HCP is *NPC* Part I: 3-*SAT* $\leq_p$ *DHCP*

We start from a 3-CNF formula ϕ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$ and construct a graph.
Intuitively, every circuit in the graph correspond to an assignment (we have 2^n circuits and 2^n satisfying assignments before we modelled the clauses.)



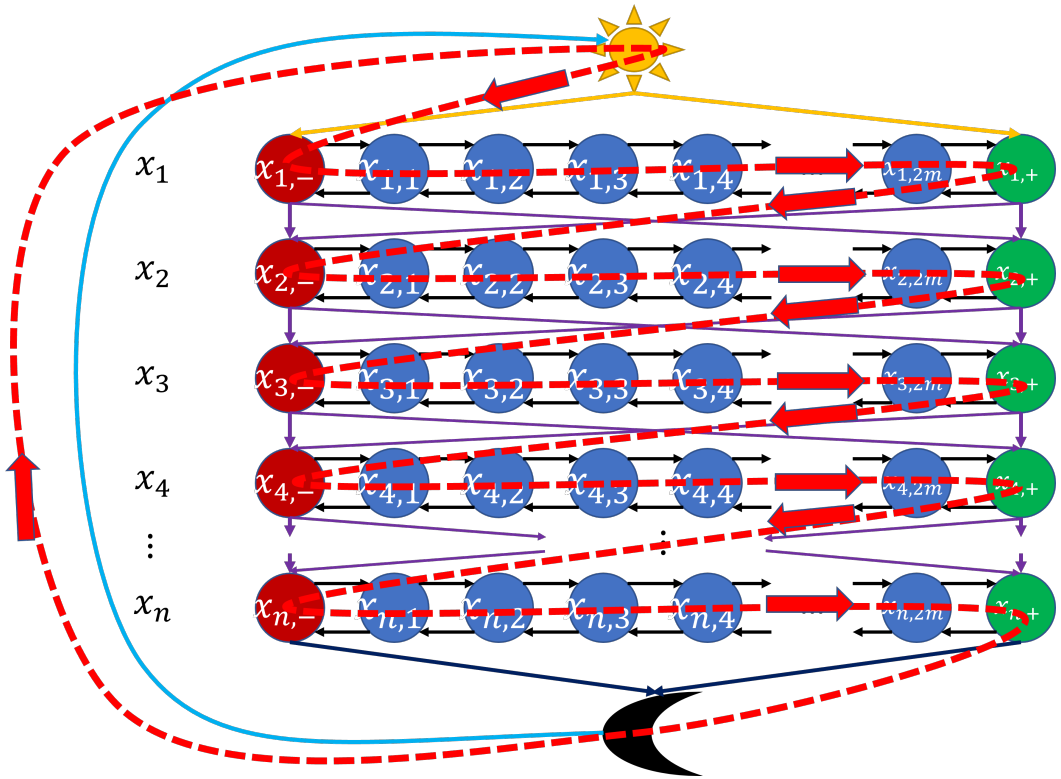
HCP is *NPC* Part I: 3-*SAT* $\leq_p$ *DHCP*

We start from a 3-CNF formula ϕ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$ and construct a graph.
Intuitively, every circuit in the graph correspond to an assignment (we have 2^n circuits and 2^n satisfying assignments before we modelled the clauses.)



HCP is *NPC* Part I: 3-*SAT* $\leq_p$ *DHCP*

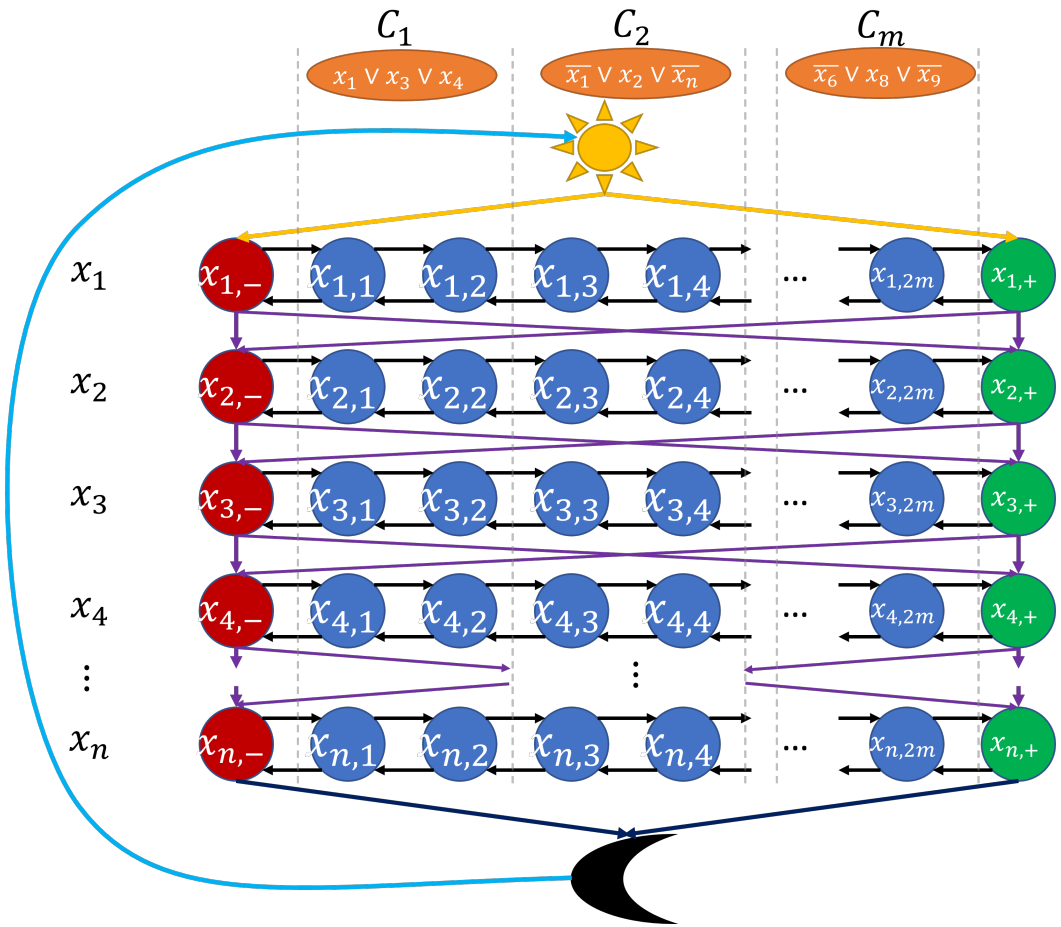
We start from a 3-CNF formula ϕ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$ and construct a graph.
Intuitively, every circuit in the graph correspond to an assignment (we have 2^n circuits and 2^n satisfying assignments before we modelled the clauses.)



Convention: The last x_i vertex on the circuit is $x_{i,+} \iff v(x_i) = T$.

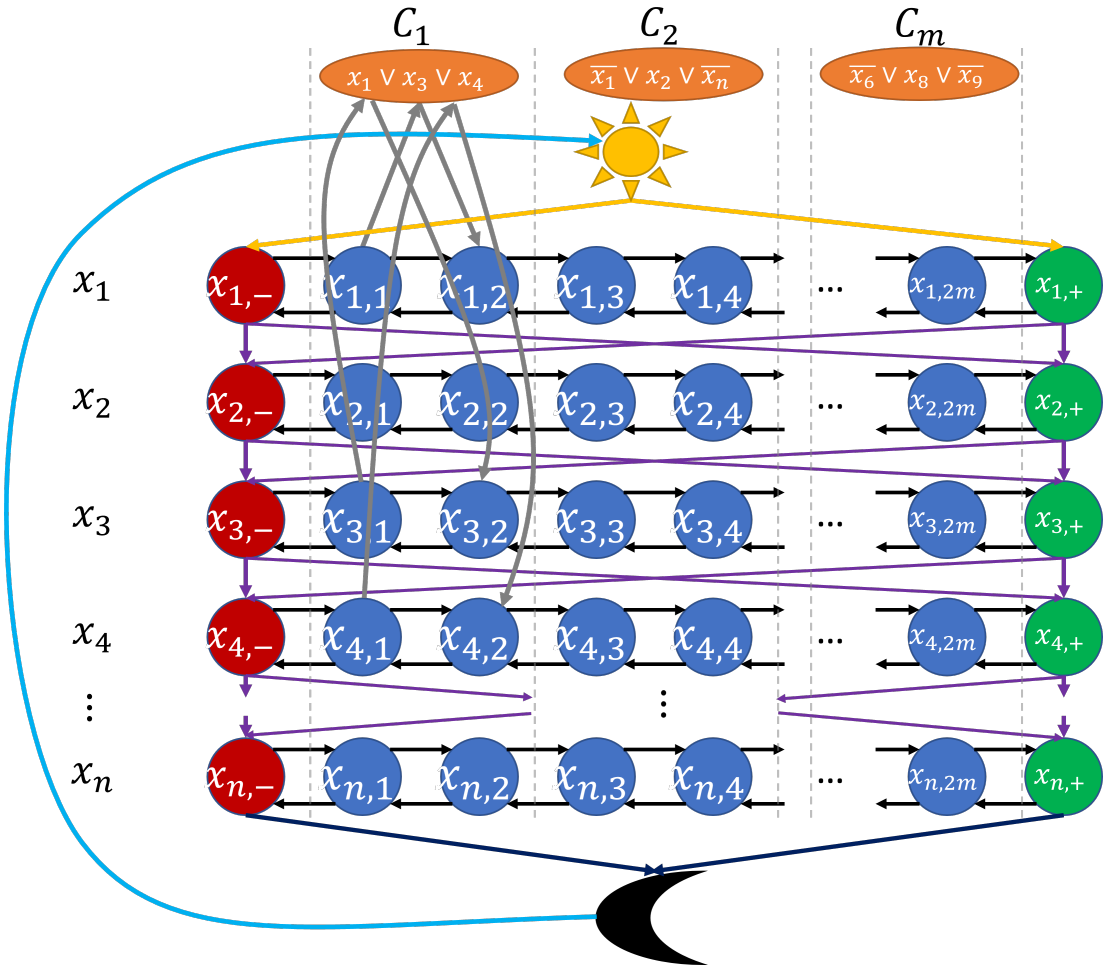
HCP is *NPC* Part I: 3-*SAT* $\leq_p$ *DHCP*

Each clause C_j has a vertex and two columns $\{x_{i,2j}, x_{i,2j+1} \mid i \in \{1, \ldots, n\}\}$.



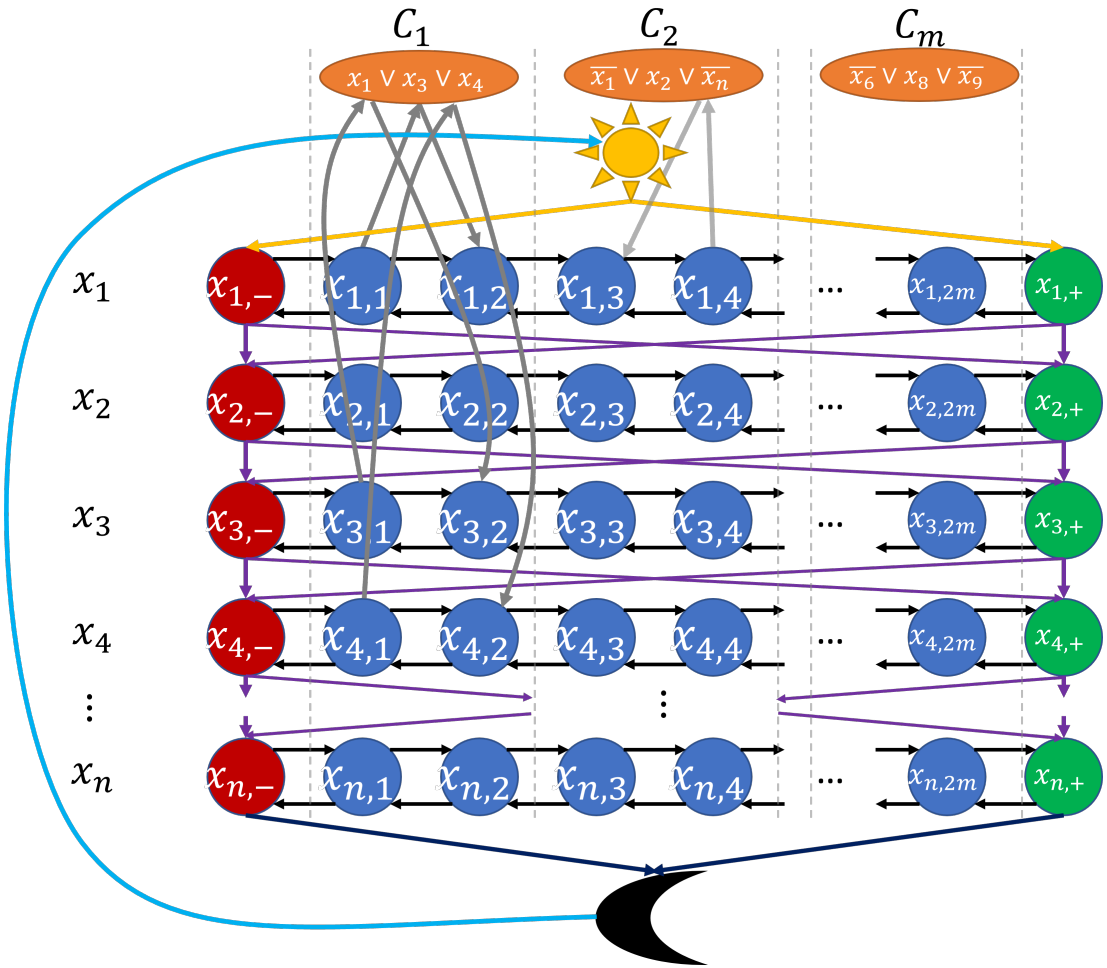
HCP is *NPC* Part I: 3-*SAT* $\leq_p$ *DHCP*

We connect positive literals from left to right and vice-versa.



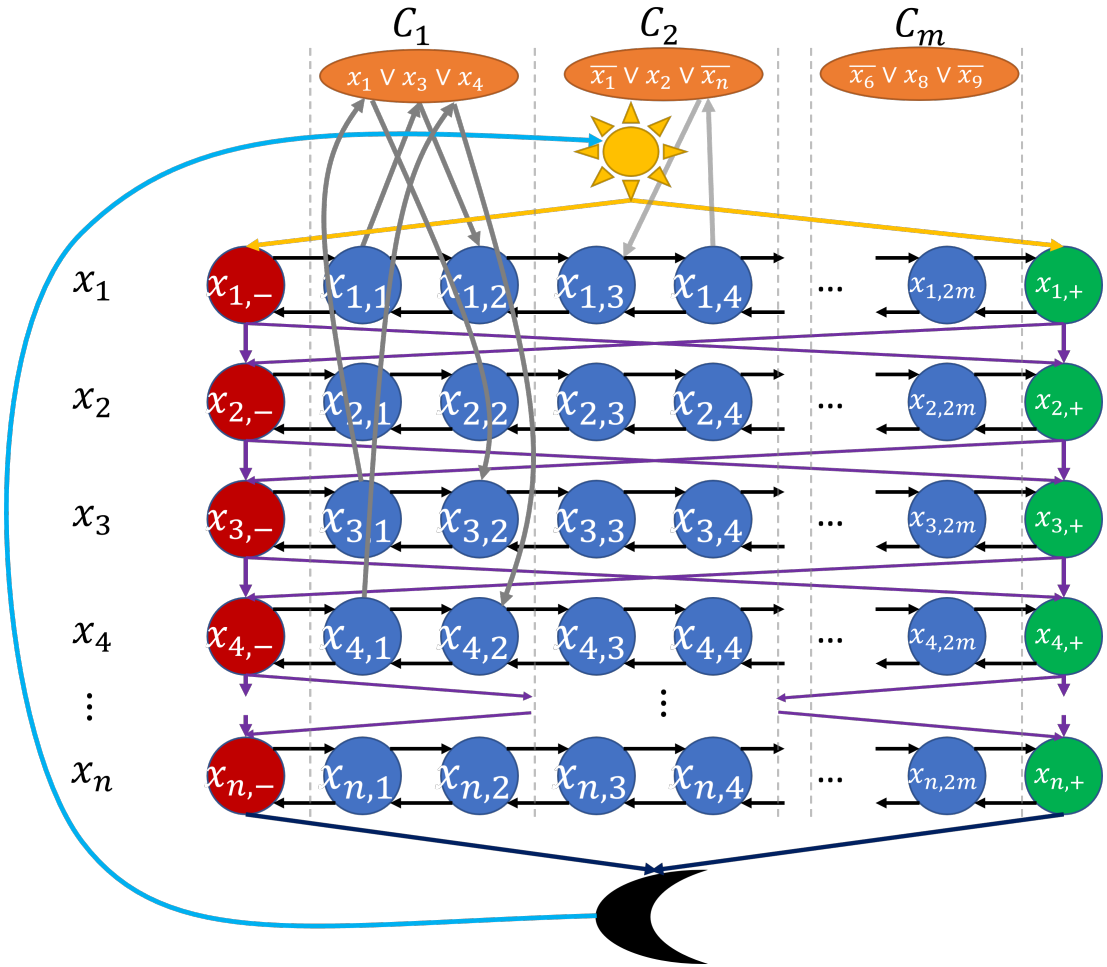
HCP is *NPC* Part I: 3-*SAT* $\leq_p$ *DHCP*

We connect positive literals from left to right and vice-versa.
Observe that assignment for x_1 can satisfy C_1 or C_2, but not both!



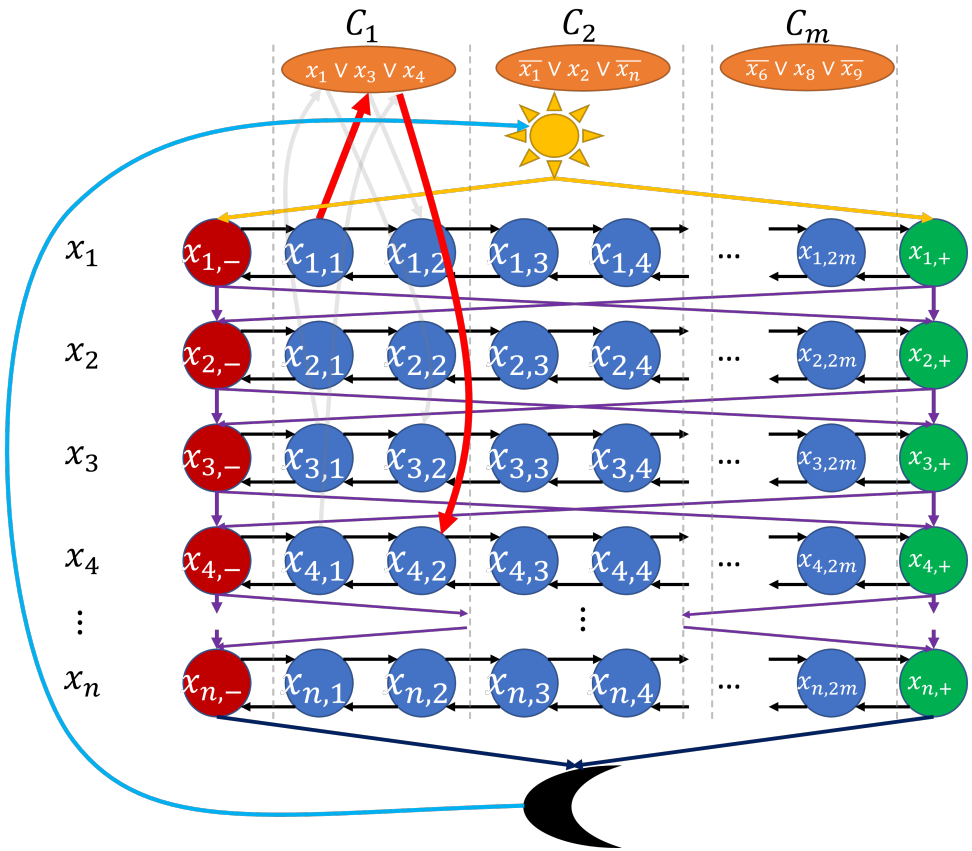
HCP is *NPC* Part I: 3-*SAT* $\leq_p$ *DHCP*

For correctness, showing that a satisfying assignment corresponds to a circuit is easy (why?).



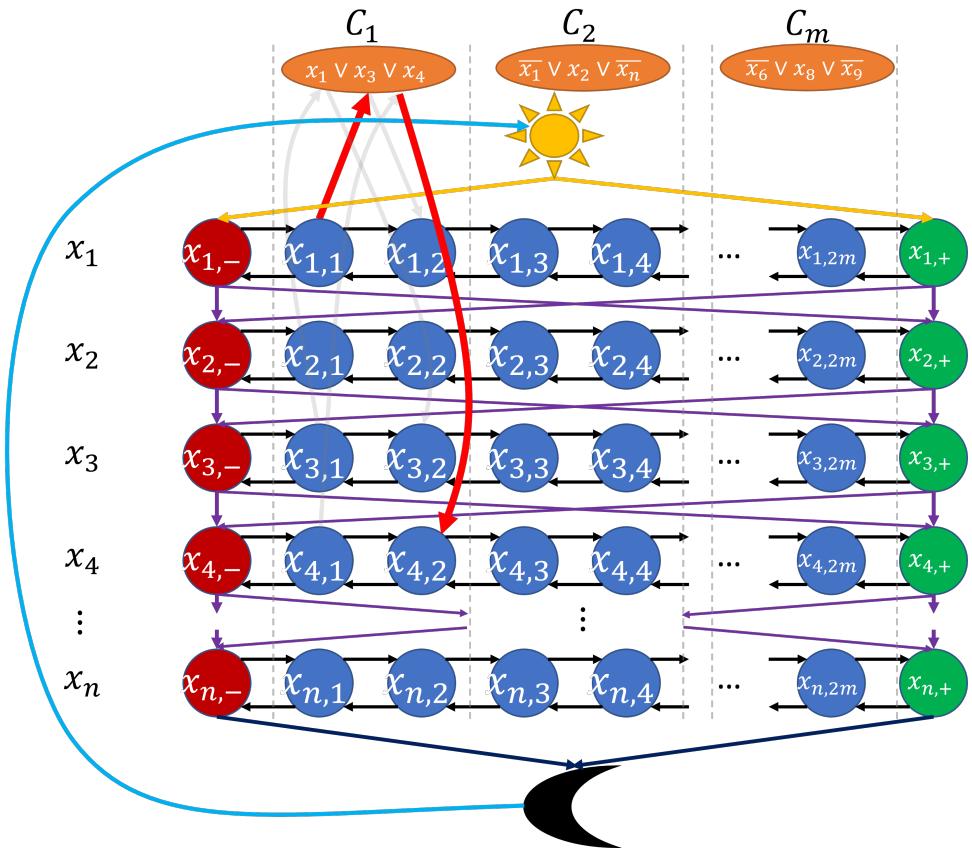
HCP is *NPC* Part I: 3-*SAT* $\leq_p$ *DHCP*

But why must a circuit correspond to a satisfying assignment?



HCP is *NPC* Part I: 3-*SAT* $\leq_p$ *DHCP*

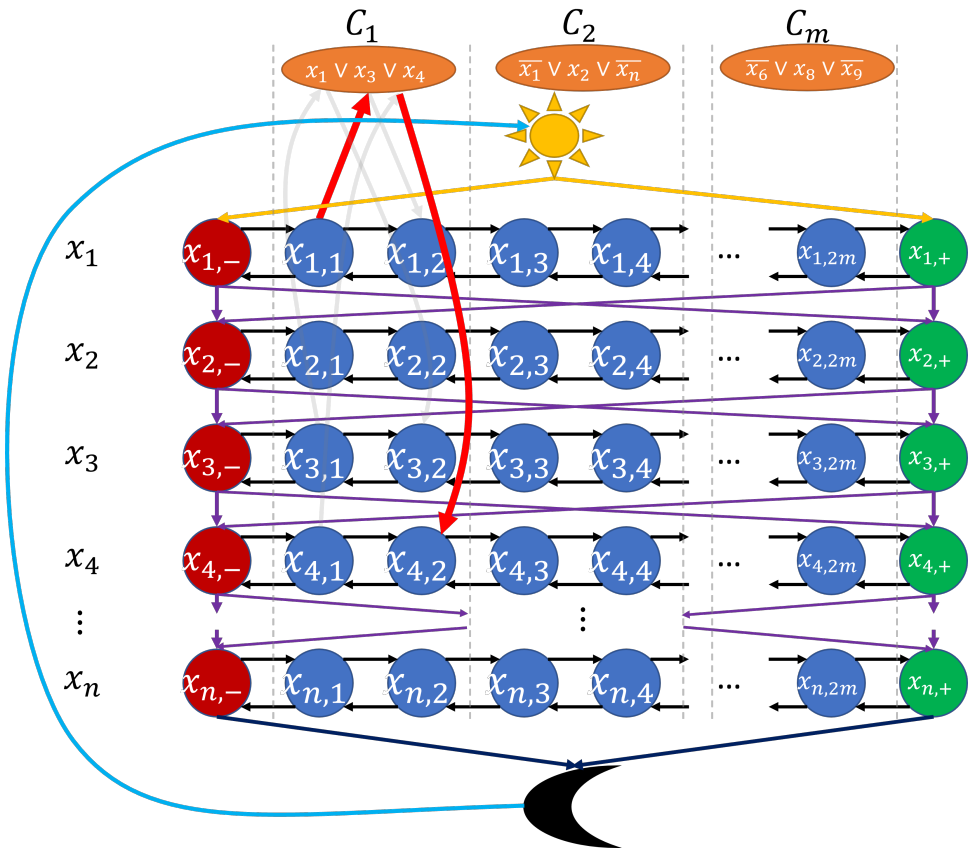
But why must a circuit correspond to a satisfying assignment?



If the circuit "cheats" like that, how can it visit $x_{1,2}$?

HCP is *NPC* Part I: 3-*SAT* $\leq_p$ *DHCP*

But why must a circuit correspond to a satisfying assignment?



If the circuit "cheats" like that, how can it visit $x_{1,2}$?
How big is the resulting graph? $O(n \cdot m)$, which is polynomial in the formula size.

HCP is *NPC* Part II: $DHCP \leq_p HCP$

We saw that 3-$SAT \leq_p DHCP$, and have that $DHCP \in NP$ (exercise!).

HCP is *NPC* Part II: $DHCP \leq_p HCP$

We saw that 3-$SAT \leq_p DHCP$, and have that $DHCP \in NP$ (exercise!). This implies that $DHCP \in NPC$.

HCP is *NPC* Part II: $DHCP \leq_p HCP$

We saw that 3-$SAT \leq_p DHCP$, and have that $DHCP \in NP$ (exercise!). This implies that $DHCP \in NPC$. How about (undirected) HCP?

HCP is *NPC* Part II: $DHCP \leq_p HCP$

We saw that 3-$SAT \leq_p DHCP$, and have that $DHCP \in NP$ (exercise!). This implies that $DHCP \in NPC$. How about (undirected) HCP?

Theorem

$$DHCP \leq_p HCP.$$

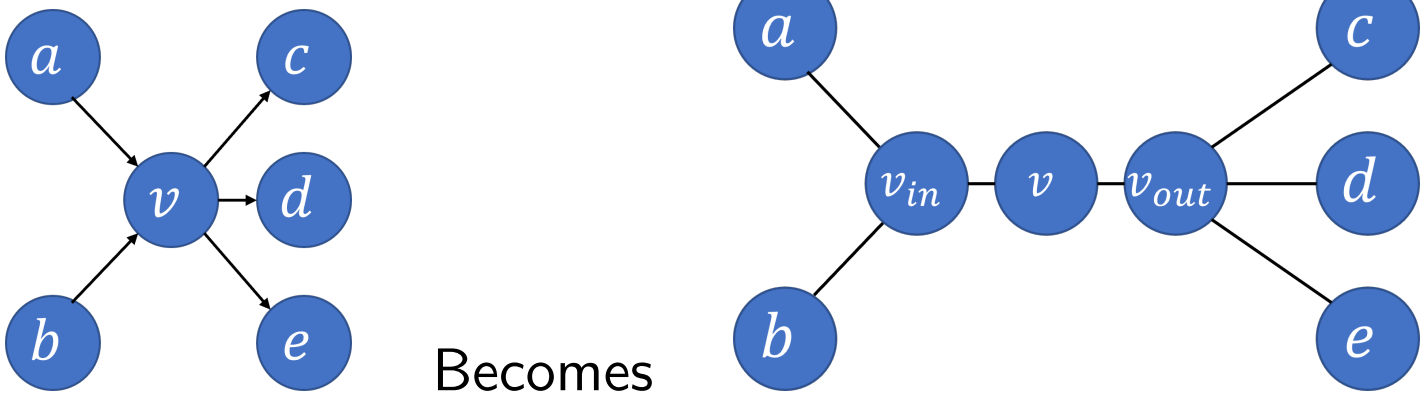
HCP is *NPC* Part II: $DHCP \leq_p HCP$

We saw that 3-$SAT \leq_p DHCP$, and have that $DHCP \in NP$ (exercise!). This implies that $DHCP \in NPC$. How about (undirected) HCP?

Theorem

$$DHCP \leq_p HCP.$$

Idea:



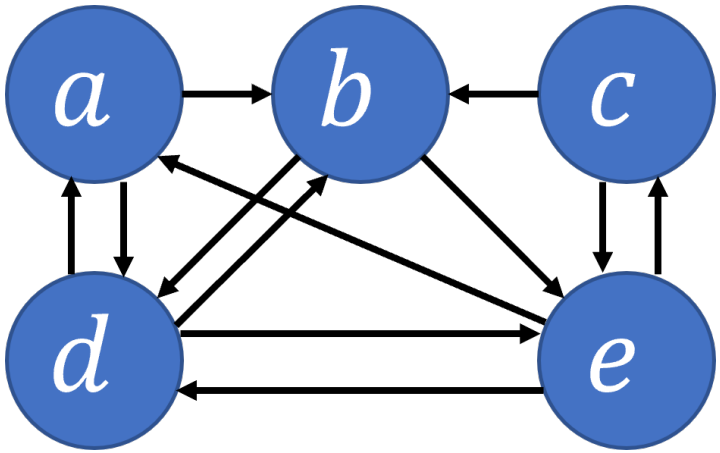
HCP is *NPC* Part II: $DHCP \leq_p HCP$

We saw that 3-$SAT \leq_p DHCP$, and have that $DHCP \in NP$ (exercise!). This implies that $DHCP \in NPC$. How about (undirected) HCP?

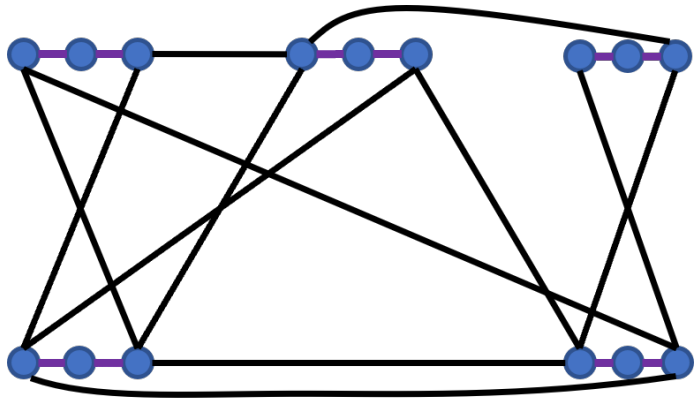
Theorem

$$DHCP \leq_p HCP.$$

Example:



Becomes



Correctness proof is left as an exercise.

NP-complete problems we saw so far, a summary

What we've seen so far:

$$SAT \leq_p 3-SAT \leq_p \{IS, DHCP\} \leq_p HCP \leq_p TSDP.$$

NP-complete problems we saw so far, a summary

What we've seen so far:

$$SAT \leq_p 3-SAT \leq_p \{IS, DHCP\} \leq_p HCP \leq_p TSDP.$$

This (together with being in NP) implies that all these problems are NP-complete!

NP-complete problems we saw so far, a summary

What we've seen so far:

$$SAT \leq_p 3-SAT \leq_p \{IS, DHCP\} \leq_p HCP \leq_p TSDP.$$

This (together with being in NP) implies that all these problems are NP-complete!
Let's see one more, which is a bit different.

Vertex Cover is NPC

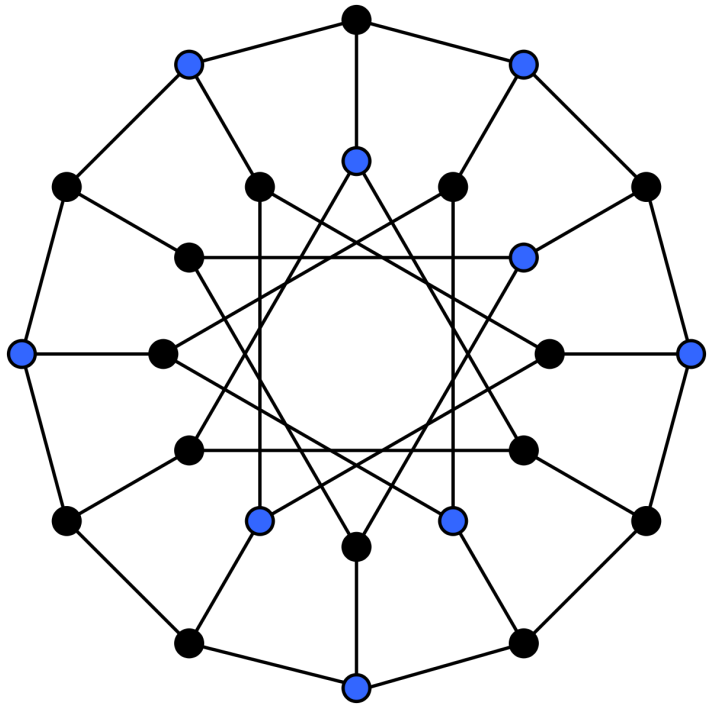
A vertex cover $C \subseteq V$ of a graph (V, E) is a set such that each edge has an endpoint in C.

Vertex Cover is NPC

A vertex cover $C \subseteq V$ of a graph (V, E) is a set such that each edge has an endpoint in C.
Instance: A graph (V, E) and an integer k.
Yes-instance: there exists a cover of size k.
No-instance: otherwise.

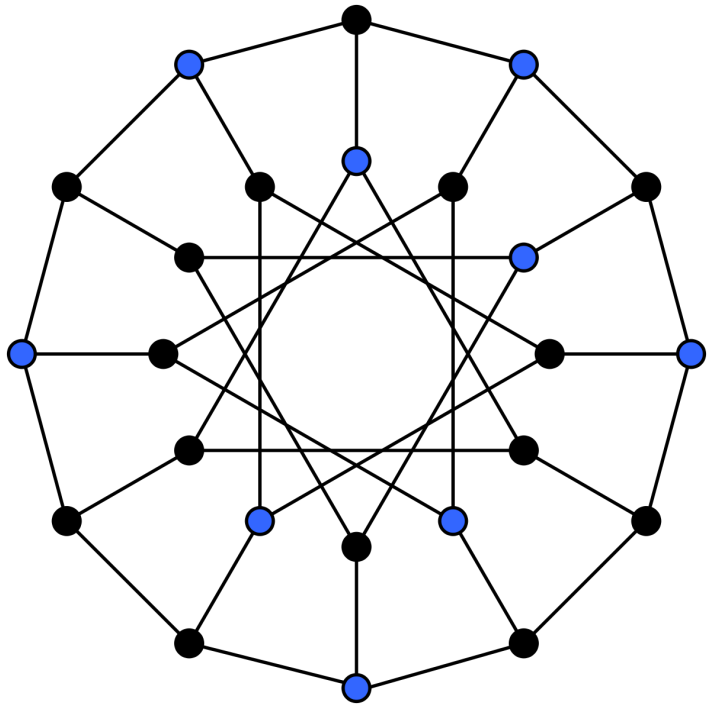
Vertex Cover is NPC

A vertex cover $C \subseteq V$ of a graph (V, E) is a set such that each edge has an endpoint in C.
Instance: A graph (V, E) and an integer k.
Yes-instance: there exists a cover of size k.
No-instance: otherwise.



Vertex Cover is NPC

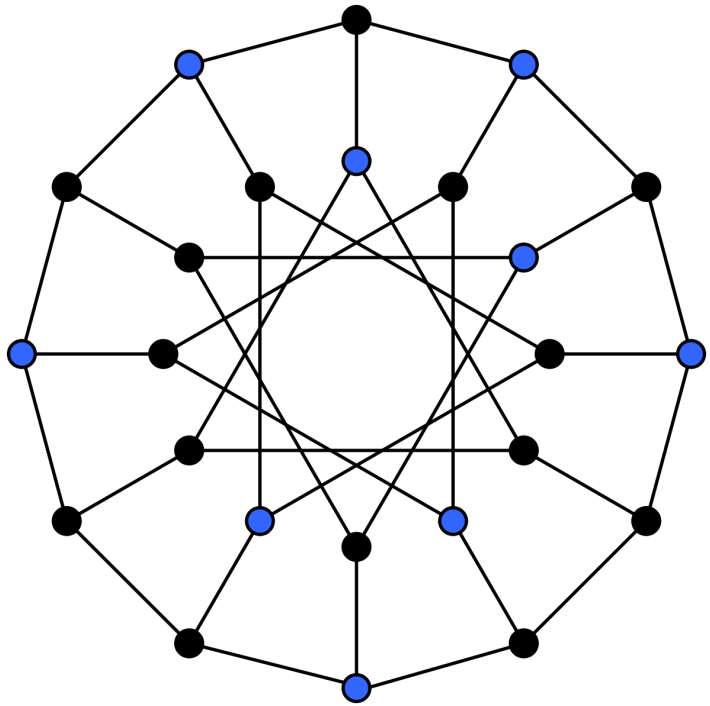
A vertex cover $C \subseteq V$ of a graph (V, E) is a set such that each edge has an endpoint in C.
Instance: A graph (V, E) and an integer k.
Yes-instance: there exists a cover of size k.
No-instance: otherwise.



Any ideas?

Vertex Cover is NPC

A vertex cover $C \subseteq V$ of a graph (V, E) is a set such that each edge has an endpoint in C.
Instance: A graph (V, E) and an integer k.
Yes-instance: there exists a cover of size k.
No-instance: otherwise.



Any ideas? G has a VC of size $k \iff$ it has an IS of size $|V| - k$.

Let's recap

Which of the following are correct?

Let's recap

Which of the following are correct?

- The 2-SAT problem is in NP.

Let's recap

Which of the following are correct?

- The 2-SAT problem is in NP.
- Since all our reductions started from SAT, it has the highest time complexity.

Let's recap

Which of the following are correct?

- The 2-SAT problem is in NP.
- Since all our reductions started from SAT, it has the highest time complexity.
- Independent set is a set of vertices that don't share neighbors.

Let's recap

Which of the following are correct?

- The 2-SAT problem is in NP.
- Since all our reductions started from SAT, it has the highest time complexity.
- Independent set is a set of vertices that don't share neighbors.
- 3-SAT can be solved in p-time if all literals appear just once.

Let's recap

Which of the following are correct?

- The 2-SAT problem is in NP.
- Since all our reductions started from SAT, it has the highest time complexity.
- Independent set is a set of vertices that don't share neighbors.
- 3-SAT can be solved in p-time if all literals appear just once.
- 3-SAT can be solved in p-time if all variables appear at most twice.

Let's recap

Which of the following are correct?

- The 2-SAT problem is in NP.
- Since all our reductions started from SAT, it has the highest time complexity.
- Independent set is a set of vertices that don't share neighbors.
- 3-SAT can be solved in p-time if all literals appear just once.
- 3-SAT can be solved in p-time if all variables appear at most twice.
- Max-2-SAT, which asks to satisfy the maximal number of clauses, can be solved in p-time.

Let's recap

Which of the following are correct?

- The 2-SAT problem is in NP.
- Since all our reductions started from SAT, it has the highest time complexity.
- Independent set is a set of vertices that don't share neighbors.
- 3-SAT can be solved in p-time if all literals appear just once.
- 3-SAT can be solved in p-time if all variables appear at most twice.
- Max-2-SAT, which asks to satisfy the maximal number of clauses, can be solved in p-time.
- *Weighted* vertex cover is NP-complete.

Let's recap

Which of the following are correct?

- The 2-SAT problem is in NP.
- Since all our reductions started from SAT, it has the highest time complexity.
- Independent set is a set of vertices that don't share neighbors.
- 3-SAT can be solved in p-time if all literals appear just once.
- 3-SAT can be solved in p-time if all variables appear at most twice.
- Max-2-SAT, which asks to satisfy the maximal number of clauses, can be solved in p-time.
- *Weighted* vertex cover is NP-complete.

Answer on Mentimeter:  .

Let's recap

Which of the following are correct?

Let's recap

Which of the following are correct?

- The 2-SAT problem is in NP.

Let's recap

Which of the following are correct?

- The 2-SAT problem is in NP. Yes. And also in P.
- Since all our reductions started from SAT, it has the highest time complexity.

Let's recap

Which of the following are correct?

- The 2-SAT problem is in NP. Yes. And also in P.
- Since all our reductions started from SAT, it has the highest time complexity. That would be unlikely.
- Independent set is a set of vertices that don't share neighbors.

Let's recap

Which of the following are correct?

- The 2-SAT problem is in NP. Yes. And also in P.
- Since all our reductions started from SAT, it has the highest time complexity. That would be unlikely.
- Independent set is a set of vertices that don't share neighbors. No, they don't share edges.
- 3-SAT can be solved in p-time if all literals appear just once.

Let's recap

Which of the following are correct?

- The 2-SAT problem is in NP. Yes. And also in P.
- Since all our reductions started from SAT, it has the highest time complexity. That would be unlikely.
- Independent set is a set of vertices that don't share neighbors. No, they don't share edges.
- 3-SAT can be solved in p-time if all literals appear just once. Yes.
- 3-SAT can be solved in p-time if all variables appear at most twice.

Let's recap

Which of the following are correct?

- The 2-SAT problem is in NP. Yes. And also in P.
- Since all our reductions started from SAT, it has the highest time complexity. That would be unlikely.
- Independent set is a set of vertices that don't share neighbors. No, they don't share edges.
- 3-SAT can be solved in p-time if all literals appear just once. Yes.
- 3-SAT can be solved in p-time if all variables appear at most twice. Yes!
- Max-2-SAT, which asks to satisfy the maximal number of clauses, can be solved in p-time.

Let's recap

Which of the following are correct?

- The 2-SAT problem is in NP. Yes. And also in P.
- Since all our reductions started from SAT, it has the highest time complexity. That would be unlikely.
- Independent set is a set of vertices that don't share neighbors. No, they don't share edges.
- 3-SAT can be solved in p-time if all literals appear just once. Yes.
- 3-SAT can be solved in p-time if all variables appear at most twice. Yes!
- Max-2-SAT, which asks to satisfy the maximal number of clauses, can be solved in p-time. No! One can show $3-SAT \leq_p Max-2-SAT$!
- *Weighted* vertex cover is NP-complete.

Let's recap

Which of the following are correct?

- The 2-SAT problem is in NP. Yes. And also in P.
- Since all our reductions started from SAT, it has the highest time complexity. That would be unlikely.
- Independent set is a set of vertices that don't share neighbors. No, they don't share edges.
- 3-SAT can be solved in p-time if all literals appear just once. Yes.
- 3-SAT can be solved in p-time if all variables appear at most twice. Yes!
- Max-2-SAT, which asks to satisfy the maximal number of clauses, can be solved in p-time. No! One can show $3\text{-}SAT \leq_p Max\text{-}2\text{-}SAT$!
- *Weighted* vertex cover is NP-complete. Yes.