Even More NP Completeness

Lecture 26

May 1, 2025

Part I

Wrap Up 3-Coloring

Last Time: 3COLOR

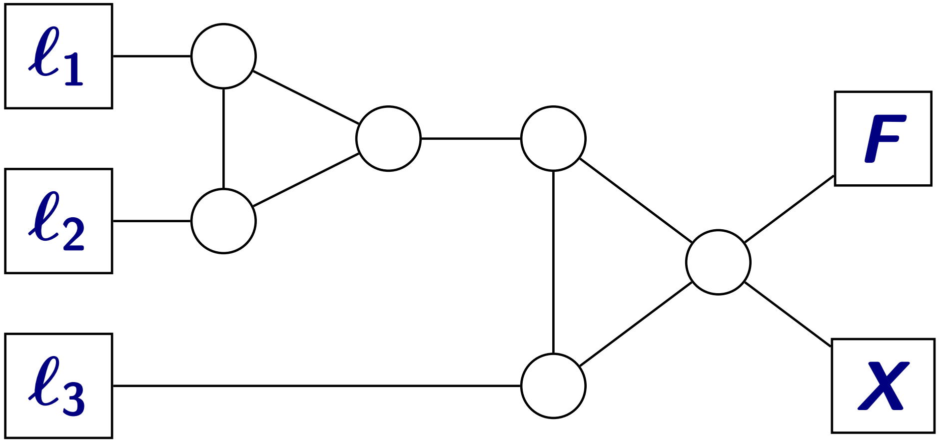
Recall: last time, wanted to prove that $\mathbf{3COLOR}$ is $\mathbf{NP}$-complete.
Need a function f such that $\varphi \in \mathbf{3SAT}$ iff $f(\varphi) \in \mathbf{3COLOR}$.

Last Time: 3COLOR

Recall: last time, wanted to prove that $\mathbf{3COLOR}$ is $\mathbf{NP}$-complete.
Need a function $\boldsymbol{f}$ such that $\varphi \in \mathbf{3SAT}$ iff $\boldsymbol{f(\varphi)} \in \mathbf{3COLOR}$.

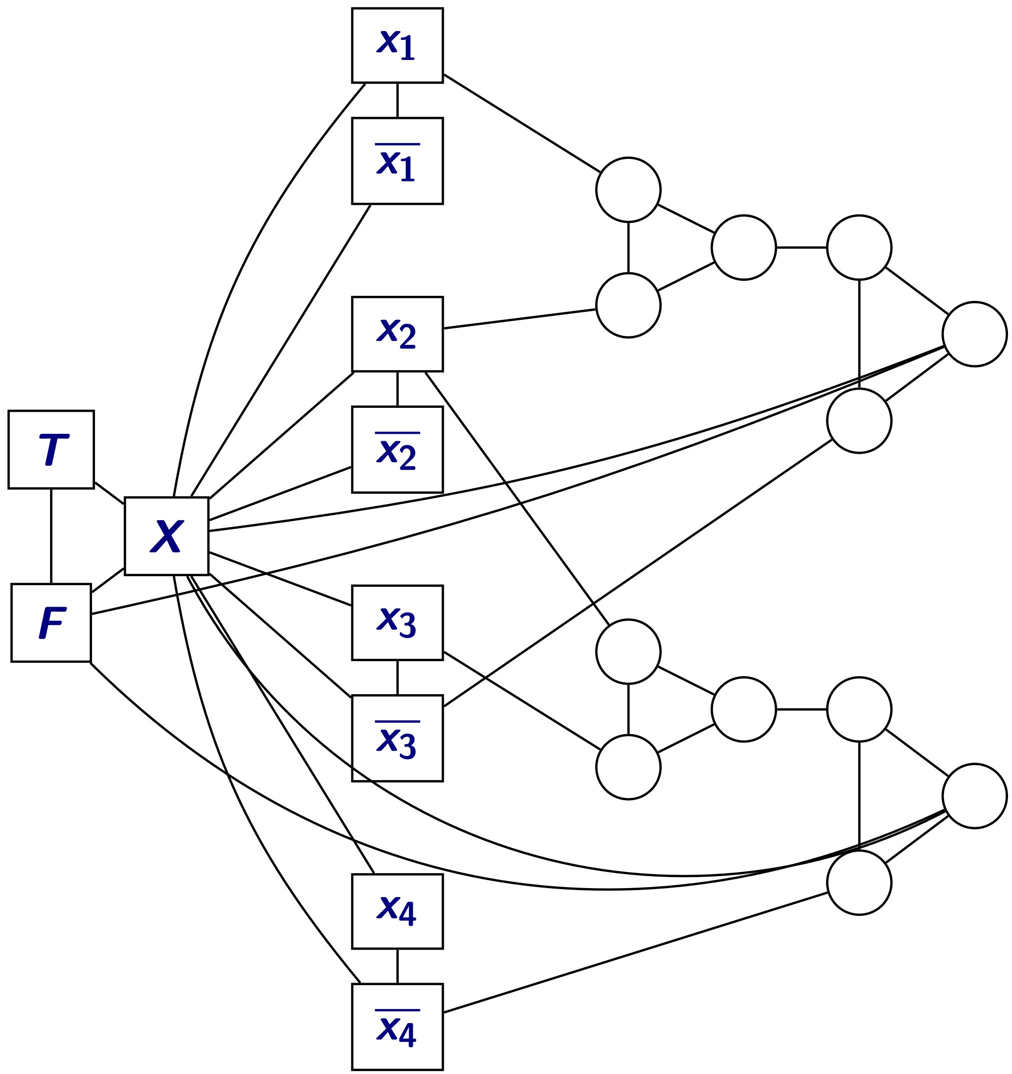
Let $\boldsymbol{f(\varphi) = G}$, where:

- We add vertices $\boldsymbol{T}$, $\boldsymbol{F}$, and $\boldsymbol{X}$ to $\boldsymbol{G}$, all connected.
- For each variable $\boldsymbol{x_i}$ in φ, we add vertices $\boldsymbol{x_i}$ and $\overline{\boldsymbol{x_i}}$, connected to each other and to $\boldsymbol{X}$.
- For each clause $\boldsymbol{C = (\ell_1 \vee \ell_2 \vee \ell_3)}$, we add the following "gadget" to $\boldsymbol{G}$: (Note: square vertices already exist in $\boldsymbol{G}$.)



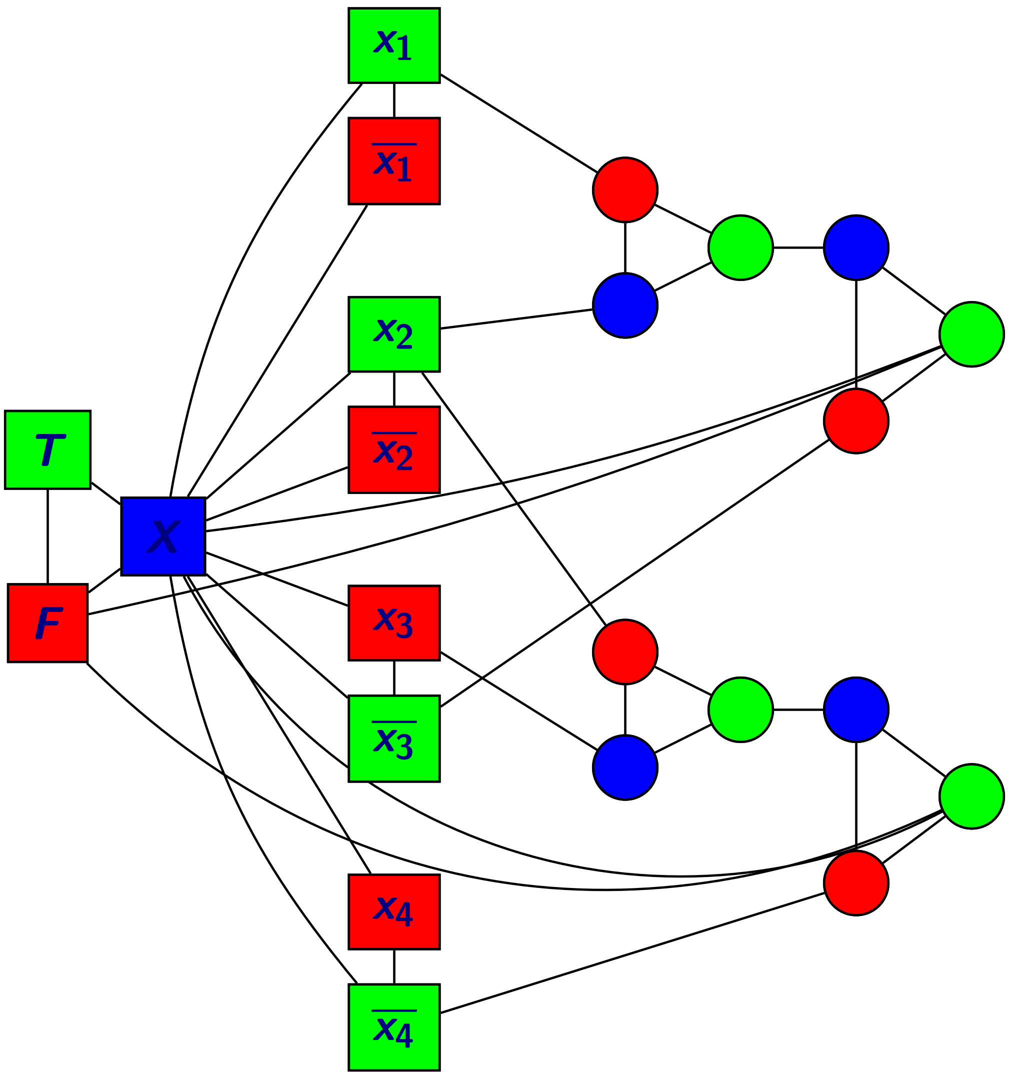
3SAT to 3COLOR: Picture

Say $\varphi = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$



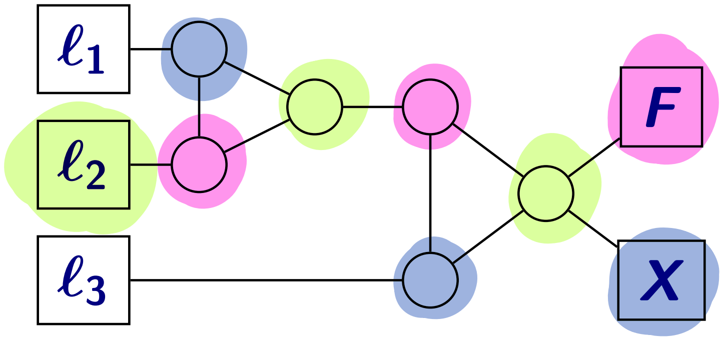
3SAT to 3COLOR: Picture

Say $\varphi = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$



3SAT to 3COLOR: Only-If

Let $f(\varphi) = G$, where for each clause $C = (\ell_1 \vee \ell_2 \vee \ell_3)$, we include:



color T = green
F = red
X = blue

Claim: if φ is satisfiable, G is 3-colorable.

Given a satisfying assignment to φ

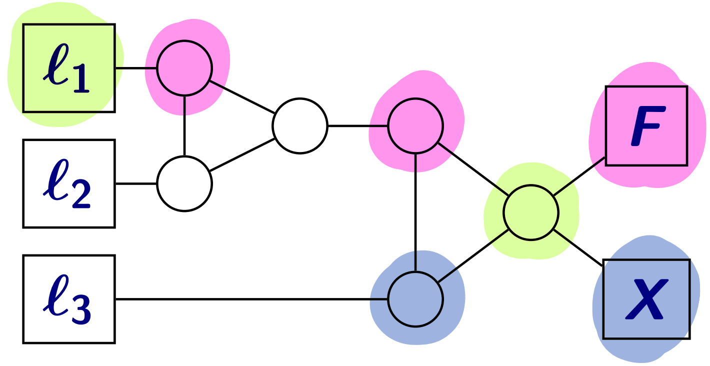
for each x_i: if $x_i = T$, color x_i = green, $\bar{x}_i$ = red

else, color x_i = red, $\bar{x}_i$ green

assignment is satisfying, each clause has ≥ 1 true literal

3SAT to 3COLOR: If

Let $f(\varphi) = G$, where for each clause $C = (\ell_1 \vee \ell_2 \vee \ell_3)$, we include:



Claim: if G is 3-colorable, φ is satisfiable.

Given a valid 3-coloring of G. WLOG, T is colored green, F red, and X blue.

For every x_i: if x_i is colored green, set $x_i := T$
if x_i is colored red, set $x_i := F$

Claim: every clause gadget has ≥ 1 literal vertex green

Part II

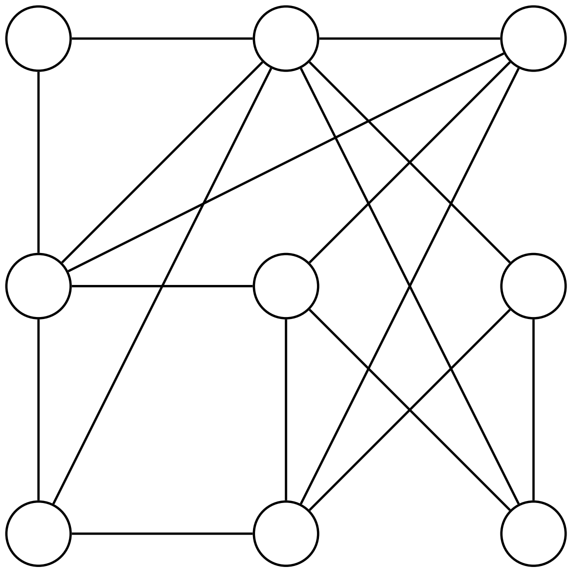
Hamiltonian Cycle

Hamiltonian Cycle

A ***Hamiltonian cycle*** is a cycle that visits every vertex.

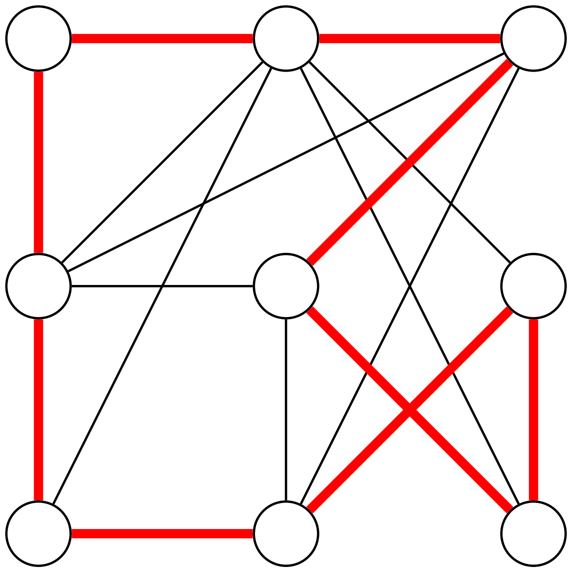
Hamiltonian Cycle

A ***Hamiltonian cycle*** is a cycle that visits every vertex.



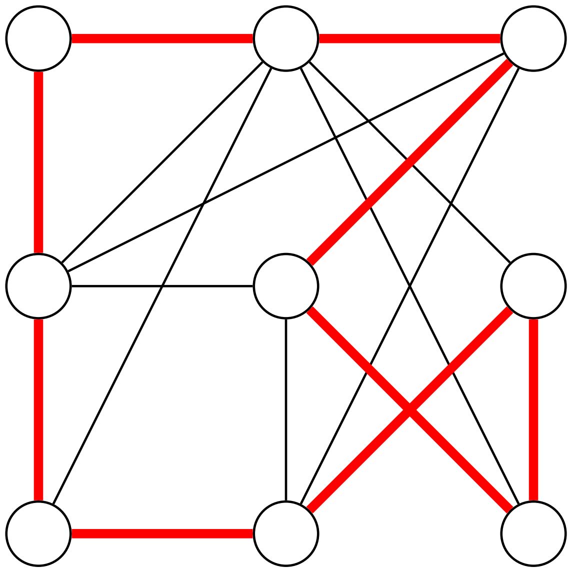
Hamiltonian Cycle

A ***Hamiltonian cycle*** is a cycle that visits every vertex.



Hamiltonian Cycle

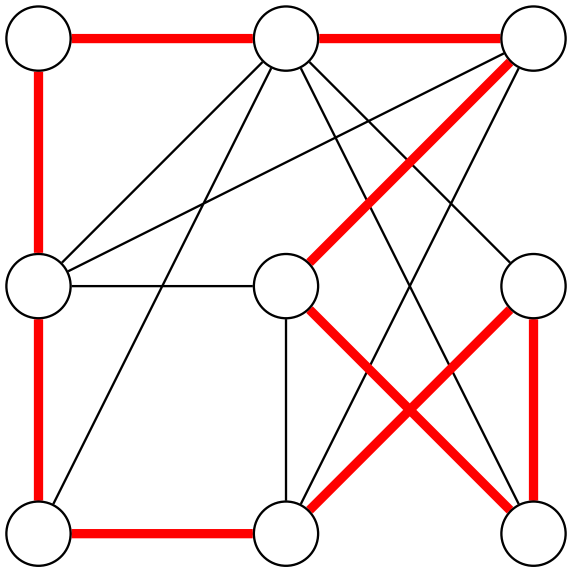
A ***Hamiltonian cycle*** is a cycle that visits every vertex.



Key question: does G have a Hamiltonian cycle?

Hamiltonian Cycle

A ***Hamiltonian cycle*** is a cycle that visits every vertex.



Key question: does G have a Hamiltonian cycle?

For ease of reduction, we will focus on the case where G is *directed*.

DIRHAM

Claim

$\textbf{DIRHAM} = \{G \mid G \text{ has a directed Ham cycle}\}$ *is* **NP**-*complete.*

DIRHAM

Claim

$\textbf{\textit{DIRHAM}} = \{\boldsymbol{G} \mid \boldsymbol{G}$ *has a directed Ham cycle*$\}$ *is* $\textbf{\textit{NP}}$*-complete.*

DIRHAM is in ***NP***: $\boldsymbol{w}$ is the description of a Hamiltonian cycle.

DIRHAM

Claim

DIRHAM $=$ {***G*** | ***G*** *has a directed Ham cycle*} *is* ***NP****-complete.*

DIRHAM is in ***NP***: ***w*** is the description of a Hamiltonian cycle.

What problem should we reduce to ***DIRHAM*** in order to prove hardness?

3SAT   IS/VC/Clique   3Color

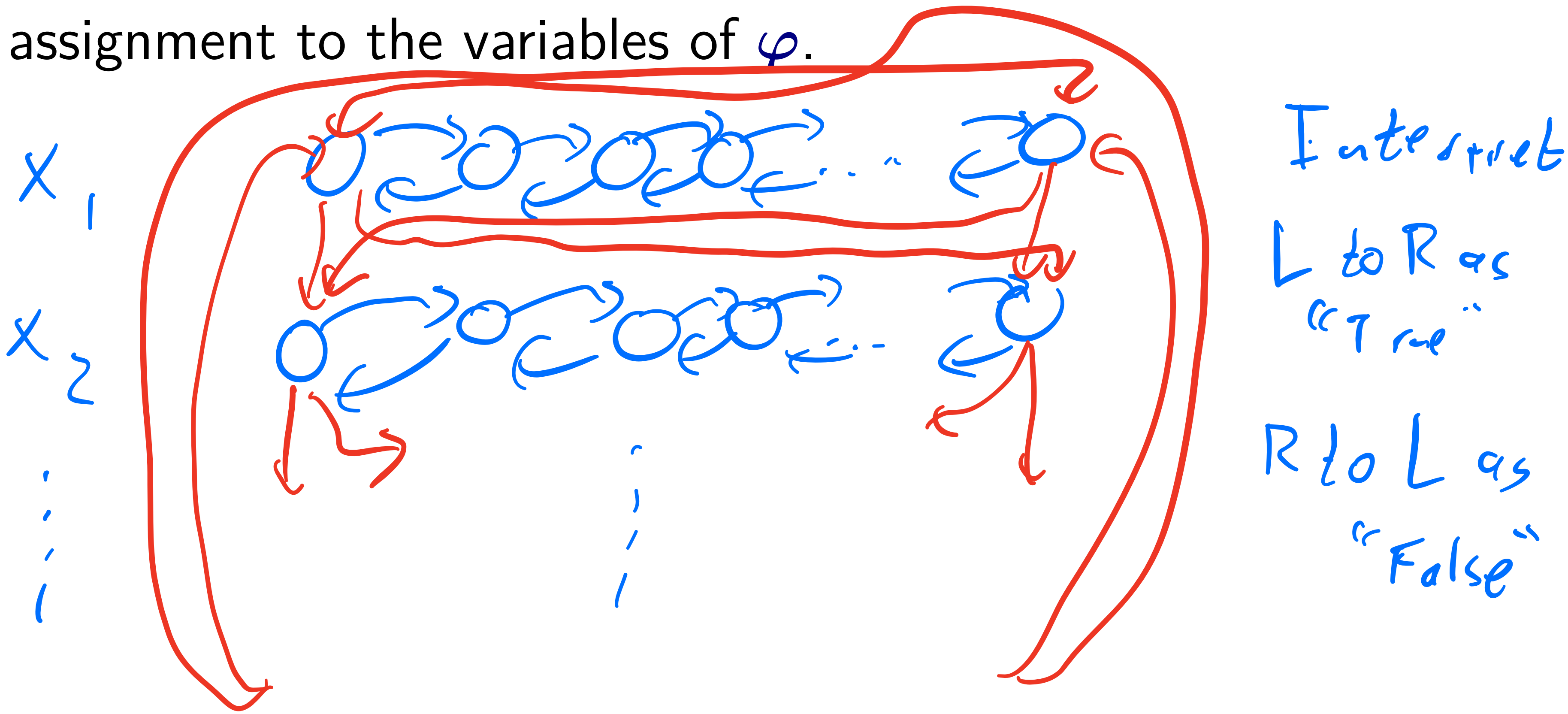
3SAT to DIRHAM: Intuition

We have a $\mathbf{3SAT}$ formula φ. We want to construct a (directed) graph $\boldsymbol{G}$ such that φ is satisfiable iff $\boldsymbol{G}$ has a Hamiltonian cycle.

3SAT to DIRHAM: Intuition

We have a ***3SAT*** formula φ. We want to construct a (directed) graph G such that φ is satisfiable iff G has a Hamiltonian cycle.

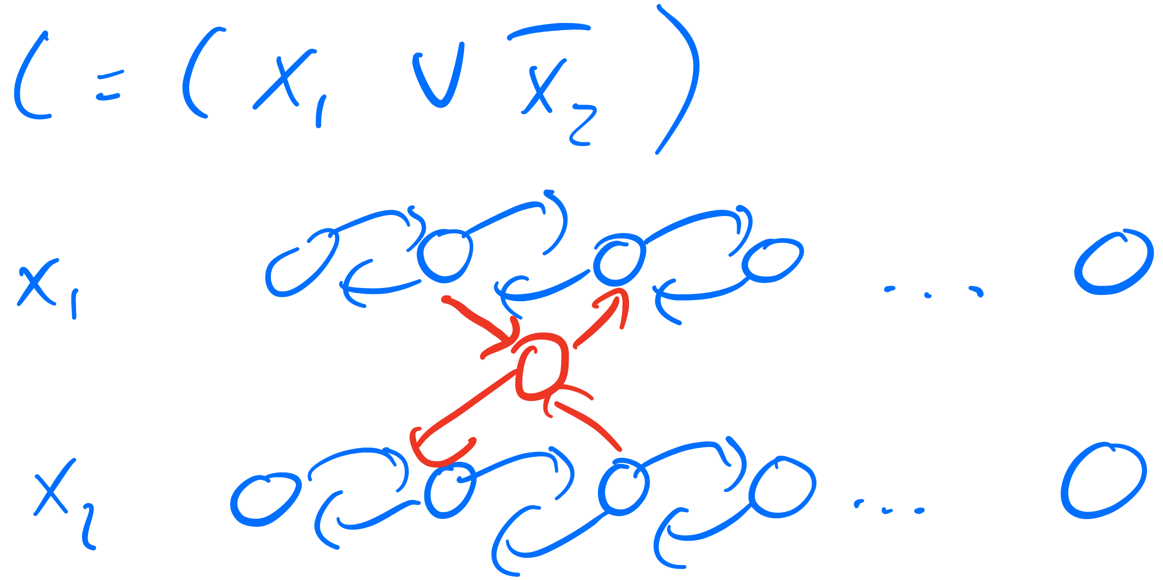
Step 1: construct G such that each Hamiltonian cycle corresponds to *some* assignment to the variables of φ.



3SAT to DIRHAM: Intuition

We have a ***3SAT*** formula φ. We want to construct a (directed) graph $\boldsymbol{G}$ such that φ is satisfiable iff $\boldsymbol{G}$ has a Hamiltonian cycle.

Step 1: construct $\boldsymbol{G}$ such that each Hamiltonian cycle corresponds to *some* assignment to the variables of φ.
Step 2: ensure that every clause is satisfied.



3SAT to DIRHAM: Reduction

Let $f(\varphi) = G$, where:

3SAT to DIRHAM: Reduction

Let $f(\varphi) = G$, where:

- If φ has n variables and k clauses, G has vertices (i, j) for $1 \leq i \leq n$ and $1 \leq j \leq 3k + 3$.

3SAT to DIRHAM: Reduction

Let $f(\varphi) = G$, where:

- If φ has n variables and k clauses, G has vertices (i, j) for $1 \leq i \leq n$ and $1 \leq j \leq 3k + 3$.
- We add edges $(i, j) \to (i, j + 1)$ and $(i, j) \to (i, j - 1)$.

3SAT to DIRHAM: Reduction

Let $f(\varphi) = G$, where:

- If φ has n variables and k clauses, G has vertices (i, j) for $1 \leq i \leq n$ and $1 \leq j \leq 3k + 3$.
- We add edges $(i, j) \to (i, j + 1)$ and $(i, j) \to (i, j - 1)$.
- We add edges $(i, 1) \to (i + 1, 1)$, $(i, 1) \to (i + 1, 3k + 3)$, $(i, 3k + 3) \to (i + 1, 1)$, and $(i, 3k + 3) \to (i + 1, 3k + 3)$.

  (We treat $n + 1$ as 1 for this step.)

3SAT to DIRHAM: Reduction

Let $f(\varphi) = G$, where:

- If φ has n variables and k clauses, G has vertices (i, j) for $1 \leq i \leq n$ and $1 \leq j \leq 3k + 3$.
- We add edges $(i, j) \to (i, j + 1)$ and $(i, j) \to (i, j - 1)$.
- We add edges $(i, 1) \to (i + 1, 1)$, $(i, 1) \to (i + 1, 3k + 3)$, $(i, 3k + 3) \to (i + 1, 1)$, and $(i, 3k + 3) \to (i + 1, 3k + 3)$.

  (We treat $n + 1$ as 1 for this step.)
- For each clause C_j, we add a vertex.

3SAT to DIRHAM: Reduction

Let $f(\varphi) = G$, where:

- If φ has n variables and k clauses, G has vertices (i, j) for $1 \leq i \leq n$ and $1 \leq j \leq 3k + 3$.
- We add edges $(i, j) \to (i, j + 1)$ and $(i, j) \to (i, j - 1)$.
- We add edges $(i, 1) \to (i + 1, 1)$, $(i, 1) \to (i + 1, 3k + 3)$, $(i, 3k + 3) \to (i + 1, 1)$, and $(i, 3k + 3) \to (i + 1, 3k + 3)$.

  (We treat $n + 1$ as 1 for this step.)
- For each clause C_j, we add a vertex.
- If x_i appears in C_j, we add edges $(i, 3j) \to C_j$ and $C_j \to (i, 3j + 1)$. If $\overline{x_i}$ appears in C_j, we add edges $(i, 3j + 1) \to C_j$ and $C_j \to (i, 3j)$.

3SAT to DIRHAM: Reduction

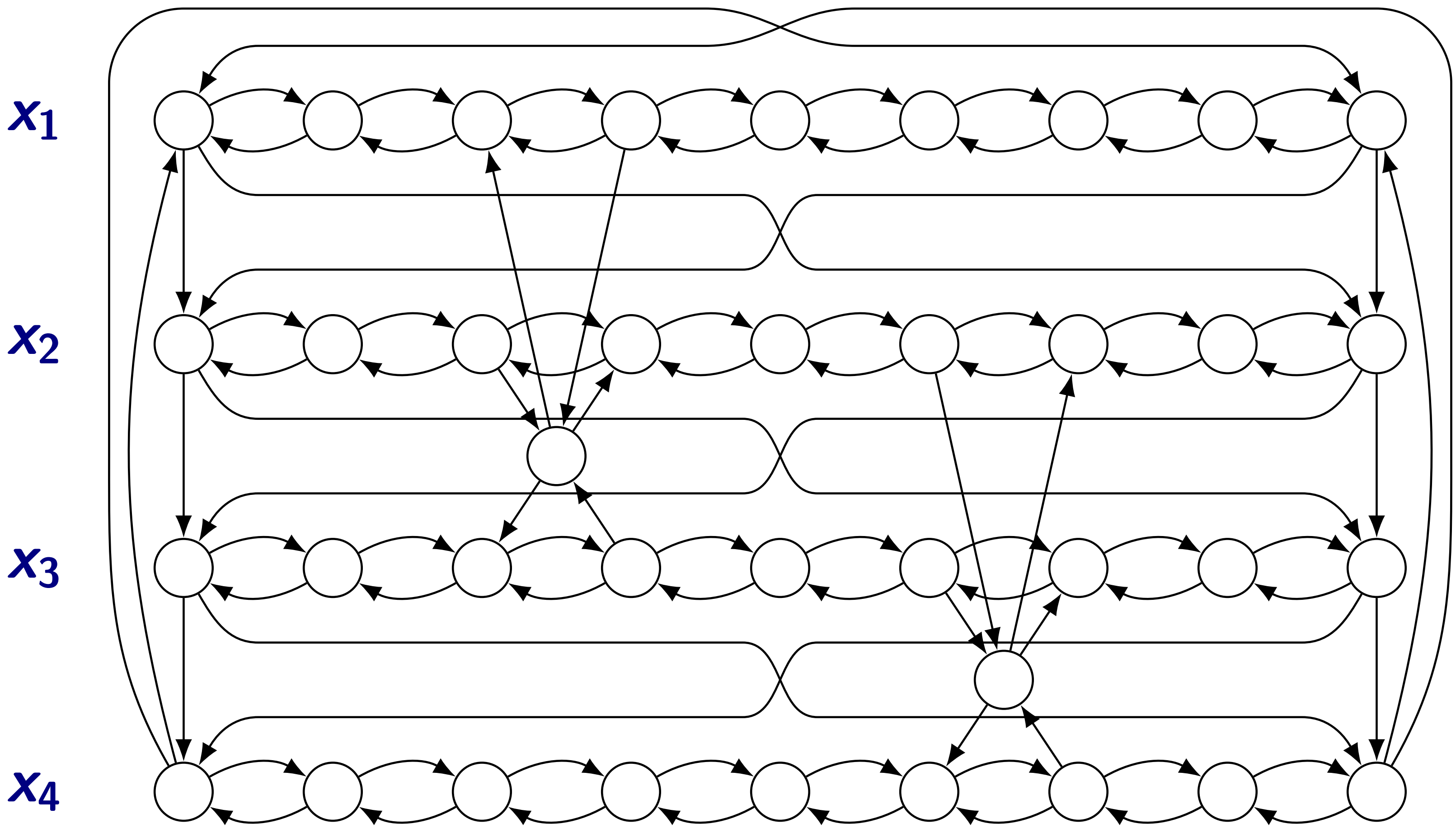
Let $f(\varphi) = G$, where:

- If φ has n variables and k clauses, G has vertices (i, j) for $1 \leq i \leq n$ and $1 \leq j \leq 3k + 3$.
- We add edges $(i, j) \to (i, j + 1)$ and $(i, j) \to (i, j - 1)$.
- We add edges $(i, 1) \to (i + 1, 1)$, $(i, 1) \to (i + 1, 3k + 3)$, $(i, 3k + 3) \to (i + 1, 1)$, and $(i, 3k + 3) \to (i + 1, 3k + 3)$.
  (We treat $n + 1$ as 1 for this step.)
- For each clause C_j, we add a vertex.
- If x_i appears in C_j, we add edges $(i, 3j) \to C_j$ and $C_j \to (i, 3j + 1)$. If $\overline{x_i}$ appears in C_j, we add edges $(i, 3j + 1) \to C_j$ and $C_j \to (i, 3j)$.

This reduction clearly runs in polynomial time. (In fact, quadratic.)
Just need to show that $\varphi \in \mathit{3SAT}$ iff $G \in \mathit{DIRHAM}$.

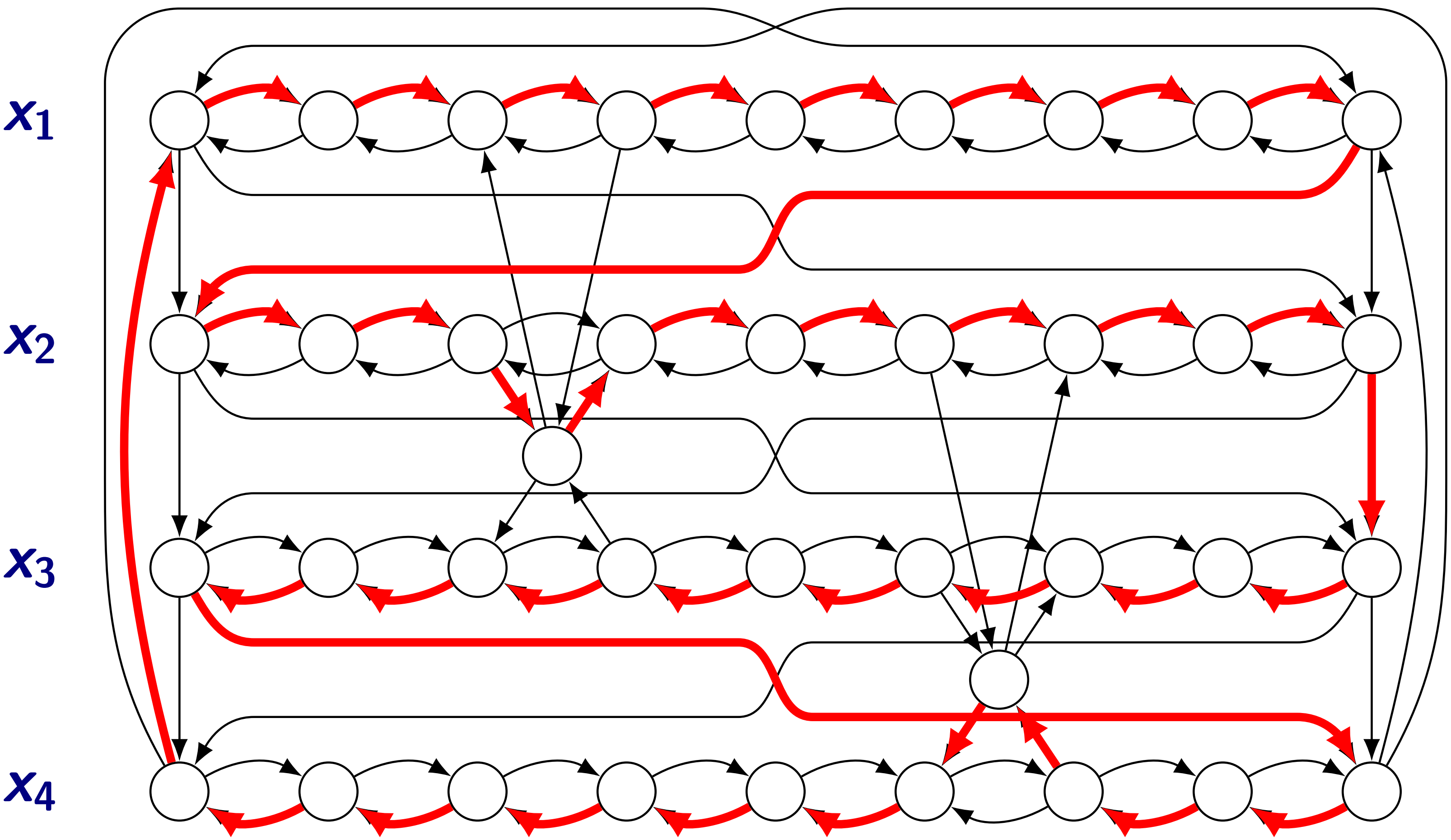
3SAT to DIRHAM: Picture

Say $\varphi = (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$



3SAT to DIRHAM: Picture

Say $\varphi = (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$



3SAT to DIRHAM: Only-If

Let $f(\varphi) = G$, where:

- For each variable x_i we have a bidirectional path of $3k+3$ vertices (i,j).
- Path end points for x_i can go to path end points for x_{i+1} (wrapping n to 1).
- For each clause C_j, we add edges $(i,3j) \to C_j$ and $C_j \to (i,3j+1)$ if x_i is in C_j, or $(i,3j+1) \to C_j$ and $C_j \to (i,3j)$ if $\overline{x_i}$ is in C_j.

Claim: if φ is satisfiable, G has a Hamiltonian cycle.

Given a satisfying assignment to φ

for each x_i: if $x_i = T$, traverse path i L to R

else, " " R to L

for each C_j: C_j must have ≥ 1 true literal

for that literal, detour to C_j between $(i,3j)$ and $(i,3j+1)$

3SAT to DIRHAM: If

Let $f(\varphi) = G$, where:

- For each variable x_i we have a bidirectional path of $3k + 3$ vertices (i, j).
- Path end points for x_i can go to path end points for x_{i+1} (wrapping n to 1).
- For each clause C_j, we add edges $(i, 3j) \rightarrow C_j$ and $C_j \rightarrow (i, 3j+1)$ if x_i is in C_j, or $(i, 3j+1) \rightarrow C_j$ and $C_j \rightarrow (i, 3j)$ if $\overline{x_i}$ is in C_j.

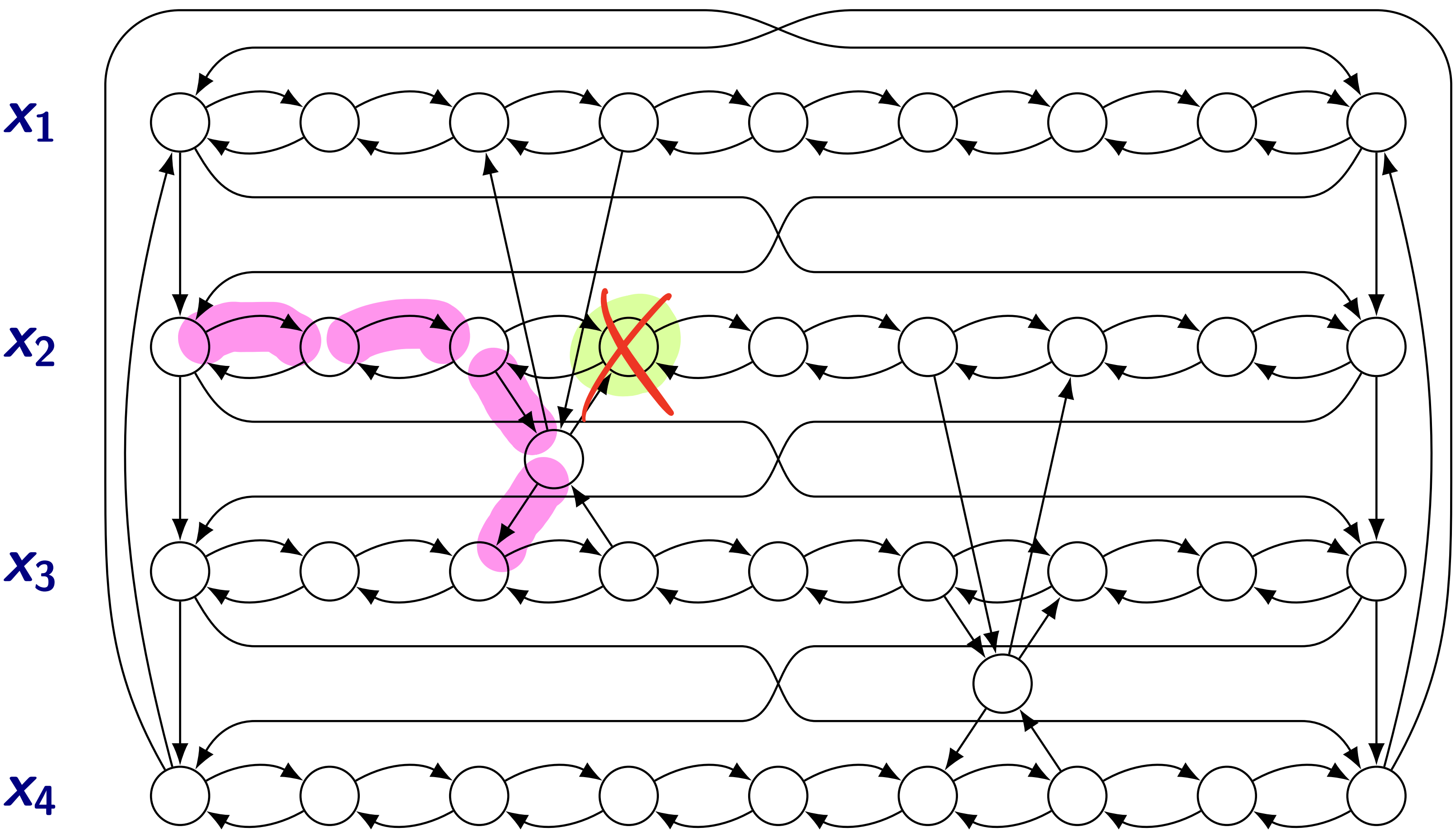
Claim: if G has a Hamiltonian cycle, φ is satisfiable.

(1) if Ham Cycle uses $(i, 3j) \rightarrow C_j$, it must also use $C_j \rightarrow (i, 3j+1)$ and vice versa

(2) if I "ignore detours", each path is either traversed L to R ($x_i = T$) or R to L ($x_i = F$)

(3) every clause has to be reached by some detour
detour is only possible if that path is traversed in the right direction

3SAT to DIRHAM: If Visualization



Related Problems

This shows that (assuming $\boldsymbol{P} \neq \boldsymbol{NP}$), there is no polynomial-time algorithm to find a Hamiltonian cycle in a *directed* graph.

- Exercise: reduce finding a Hamiltonian cycle in a directed graph to finding a Hamiltonian cycle in an undirected graph.
- From PrairieLearn: Hamiltonian *path* is also $\boldsymbol{NP}$-hard.

Related Problems

This shows that (assuming $\boldsymbol{P} \neq \boldsymbol{NP}$), there is no polynomial-time algorithm to find a Hamiltonian cycle in a *directed* graph.

- Exercise: reduce finding a Hamiltonian cycle in a directed graph to finding a Hamiltonian cycle in an undirected graph.
- From PrairieLearn: Hamiltonian *path* is also **NP**-hard.

Claim

Finding the single-source shortest (simple) paths in a graph with no constraints on the edge weights is **NP***-hard.*

Reduce from Ham Path:

Set all edge weights to -1, run SSSP from every vertex

G has Ham Path iff any vertex has distance $-(V-1)$

Part III

Subset Sum

Subset Sum

Subset Sum problem: given a set S of n positive integers, is there a subset of S that adds up to some target integer T?

Subset Sum

Subset Sum problem: given a set S of n positive integers, is there a subset of S that adds up to some target integer T?

Say $S = [1, 3, 7, 12, 374]$. Can we make:

- $T = 11$?
- $T = 17$?
- $T = 397$?
- $T = 398$?

SUBSUM

Claim

*$\textbf{SUBSUM} = \{\textbf{S}, \textbf{T} \mid \text{a subset of } \textbf{S} \text{ sums to } \textbf{T}\}$ is **NP**-complete.*

SUBSUM

Claim

$\boldsymbol{SUBSUM} = \{\boldsymbol{S}, \boldsymbol{T} \mid$ *a subset of* $\boldsymbol{S}$ *sums to* $\boldsymbol{T}\}$ *is* $\boldsymbol{NP}$*-complete.*

$\boldsymbol{SUBSUM}$ is in $\boldsymbol{NP}$: $\boldsymbol{w}$ describes a subset of $\boldsymbol{S}$ that sums to $\boldsymbol{T}$.

SUBSUM

Claim

$\textbf{SUBSUM} = \{S, T \mid \textit{a subset of } S \textit{ sums to } T\}$ *is* **NP***-complete.*

SUBSUM is in ***NP***: w describes a subset of S that sums to T.

What problem should we reduce to ***SUBSUM*** in order to prove hardness?

VC to SUBSUM: Intuition

We have a graph G and a number k. We want to construct a set S and target T such that G has a vertex cover of size k iff S has a subset that sums to T.

VC to SUBSUM: Intuition

We have a graph G and a number k. We want to construct a set S and target T such that G has a vertex cover of size k iff S has a subset that sums to T.

Key idea: make an integer per vertex, representing edges "covered".

VC to SUBSUM: Reduction

Let $f(G, k) = (S, T)$ where:

VC to SUBSUM: Reduction

Let $f(G, k) = (S, T)$ where:

- We number the edges of G arbitrarily from 0 to $E - 1$

VC to SUBSUM: Reduction

Let $f(G, k) = (S, T)$ where:

- We number the edges of G arbitrarily from 0 to $E - 1$
- For each vertex v, we include $a_v = 4^E + \sum_{i \in \Delta(v)} 4^i$ in S.

  ($\Delta(v)$ is the set of edges with v as one endpoint.)

VC to SUBSUM: Reduction

Let $f(G, k) = (S, T)$ where:

- We number the edges of G arbitrarily from 0 to $E - 1$
- For each vertex v, we include $a_v = 4^E + \sum_{i \in \Delta(v)} 4^i$ in S.

  ($\Delta(v)$ is the set of edges with v as one endpoint.)
- For each edge i, we include $b_i = 4^i$ in S.

VC to SUBSUM: Reduction

Let $f(G, k) = (S, T)$ where:

- We number the edges of G arbitrarily from 0 to $E - 1$
- For each vertex v, we include $a_v = 4^E + \sum_{i \in \Delta(v)} 4^i$ in S.
  ($\Delta(v)$ is the set of edges with v as one endpoint.)
- For each edge i, we include $b_i = 4^i$ in S.
- We set $T = k \cdot 4^E + \sum_{i=0}^{E-1} 2 \cdot 4^i$.

VC to SUBSUM: Reduction

Let $f(G, k) = (S, T)$ where:

- We number the edges of G arbitrarily from 0 to $E - 1$
- For each vertex v, we include $a_v = 4^E + \sum_{i \in \Delta(v)} 4^i$ in S.

  ($\Delta(v)$ is the set of edges with v as one endpoint.)
- For each edge i, we include $b_i = 4^i$ in S.
- We set $T = k \cdot 4^E + \sum_{i=0}^{E-1} 2 \cdot 4^i$.

This reduction clearly* runs in polynomial time. (In fact, linear.)
Just need to show that $(G, k) \in VC$ iff $(S, T) \in SUBSUM$.

Aside: Representing Integers

Our reduction uses integers that are exponentially large—how can it run in polynomial time?

Aside: Representing Integers

Our reduction uses integers that are exponentially large—how can it run in polynomial time?

To represent an integer x, we only need $\log_2(x)$ bits!

Aside: Representing Integers

Our reduction uses integers that are exponentially large—how can it run in polynomial time?

To represent an integer x, we only need $\log_2(x)$ bits!

- In our reduction, T is at most $(k+1) \cdot 4^E$, and so takes at most $\log_2(k+1) + 2E$ bits.

Aside: Representing Integers

Our reduction uses integers that are exponentially large—how can it run in polynomial time?

To represent an integer x, we only need $\log_2(x)$ bits!

- In our reduction, T is at most $(k+1) \cdot 4^E$, and so takes at most $\log_2(k+1) + 2E$ bits.

The "size" of (S, T) is $|S| + \log T$, so showing ***SUBSUM*** is ***NP***-complete just means that no algorithm can solve it in time polynomial in $|S|$ and $\log T$. (We call such problems "weakly ***NP***-hard".)

Aside: Representing Integers

Our reduction uses integers that are exponentially large—how can it run in polynomial time?

To represent an integer x, we only need $\log_2(x)$ bits!

- In our reduction, T is at most $(k+1) \cdot 4^E$, and so takes at most $\log_2(k+1) + 2E$ bits.

The "size" of (S, T) is $|S| + \log T$, so showing ***SUBSUM*** is ***NP***-complete just means that no algorithm can solve it in time polynomial in $|S|$ and $\log T$. (We call such problems "weakly ***NP***-hard".)

- There is a dynamic programming algorithm that runs in time $O(nT)$, so our reduction did in fact *need* to use large integers.

VC to SUBSUM: Only-If

Let $f(G, k) = (S, T)$, where:

- For each vertex v, we add $a_v = 4^E + \sum_{i \in \Delta(v)} 4^i$ to S.
- For each edge i, we add $b_i = 4^i$ to S.
- We set $T = k \cdot 4^E + \sum_{i=0}^{E-1} 2 \cdot 4^i$.

Claim: if G has a vertex cover of size k, S has a subset sum to T.

VC to SUBSUM: If

Let $f(G, k) = (S, T)$, where:

- For each vertex v, we add $a_v = 4^E + \sum_{i \in \Delta(v)} 4^i$ to S.
- For each edge i, we add $b_i = 4^i$ to S.
- We set $T = k \cdot 4^E + \sum_{i=0}^{E-1} 2 \cdot 4^i$.

Claim: if S has a subset sum to T, G has a vertex cover of size k.

Takeaway Points

Known ***NP***-complete languages

- SAT (from Cook-Levin)
- CNF-SAT (from Cook-Levin)
- 3SAT (from CNF-SAT)
- Independent Set (from 3SAT)
- Clique (from Independent Set)
- Vertex Cover (from Independent Set)
- 3-coloring (from 3SAT)
- Hamiltonian path / cycle (directed or undirected) (from 3SAT)
- Subset Sum (from Vertex Cover)
- And many others we don't have time for! (See Jeff's book.)