# More NP-Complete Problems

Lecture 23

April 21, 2011

# Recap

**NP**: languages that have polynomial time certifiers/verifiers

A language **L** is **NP-Complete** iff

- **L** is in **NP**
- for every **L′** in **NP**, $\mathbf{L'} \leq_P \mathbf{L}$

**L** is **NP-Hard** if for every **L′** in NP, $\mathbf{L'} \leq_P \mathbf{L}$.

**Theorem (Cook-Levin)**

***Circuit-SAT*** *and* ***SAT*** *are* ***NP-Complete****.*

# Recap

**NP**: languages that have polynomial time certifiers/verifiers

A language $\mathbf{L}$ is **NP-Complete** iff

- $\mathbf{L}$ is in **NP**
- for every $\mathbf{L'}$ in **NP**, $\mathbf{L'} \leq_P \mathbf{L}$

$\mathbf{L}$ is **NP-Hard** if for every $\mathbf{L'}$ in NP, $\mathbf{L'} \leq_P \mathbf{L}$.

**Theorem (Cook-Levin)**

**Circuit-SAT** *and* **SAT** *are* **NP-Complete**.

# Recap

**NP**: languages that have polynomial time certifiers/verifiers

A language **L** is **NP-Complete** iff

- **L** is in **NP**
- for every $\mathbf{L'}$ in **NP**, $\mathbf{L'} \leq_{\mathbf{P}} \mathbf{L}$

**L** is **NP-Hard** if for every $\mathbf{L'}$ in NP, $\mathbf{L'} \leq_{\mathbf{P}} \mathbf{L}$.

**Theorem (Cook-Levin)**

**Circuit-SAT** *and* **SAT** *are* **NP-Complete**.

# Recap

**NP**: languages that have polynomial time certifiers/verifiers

A language **L** is **NP-Complete** iff

- **L** is in **NP**
- for every $\mathbf{L'}$ in **NP**, $\mathbf{L'} \leq_{\mathbf{P}} \mathbf{L}$

**L** is **NP-Hard** if for every $\mathbf{L'}$ in NP, $\mathbf{L'} \leq_{\mathbf{P}} \mathbf{L}$.

**Theorem (Cook-Levin)**

**Circuit-SAT** *and* **SAT** *are* **NP-Complete**.

# Recap contd

**Theorem (Cook-Levin)**

*Circuit-SAT and SAT are* **NP-Complete**.

Establish **NP-Complete**ness via reductions:

- SAT $\leq_P$ 3-SAT and hence 3-SAT is **NP**-complete
- 3-SAT $\leq_P$ Independent Set (which is in **NP**) and hence Independent Set is **NP-Complete**
- Vertex Cover is **NP-Complete**
- Clique is **NP-Complete**
- Set Cover is **NP-Complete**

# Today

Prove

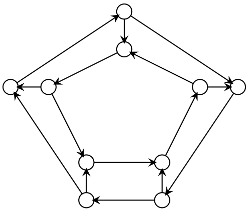
- Hamiltonian Cycle Problem is **NP-Complete**
- 3-Coloring is **NP-Complete**

# Directed Hamiltonian Cycle

**Input** Given a directed graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with $\mathbf{n}$ vertices

**Goal** Does $\mathbf{G}$ have a Hamiltonian cycle?

- A Hamiltonian cycle is a cycle in the graph that visits every vertex in $\mathbf{G}$ exactly once

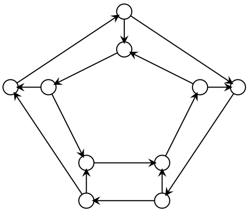


# Directed Hamiltonian Cycle

**Input** Given a directed graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with $\mathbf{n}$ vertices

**Goal** Does $\mathbf{G}$ have a Hamiltonian cycle?

- A Hamiltonian cycle is a cycle in the graph that visits every vertex in $\mathbf{G}$ exactly once

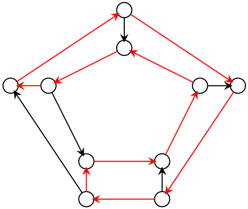


# Directed Hamiltonian Cycle

**Input** Given a directed graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with $\mathbf{n}$ vertices

**Goal** Does $\mathbf{G}$ have a Hamiltonian cycle?

- A Hamiltonian cycle is a cycle in the graph that visits every vertex in $\mathbf{G}$ exactly once



# Directed Hamiltonian Cycle is NP-complete

- Directed Hamiltonian Cycle is in **NP**
  - Certificate: Sequence of vertices
  - Certifier: Check if every vertex (except the first) appears exactly once, and that consecutive vertices are connected by a directed edge
- Hardness: We will show
  **3-SAT $\leq_P$ Directed Hamiltonian Cycle**

# Reduction

Given 3-SAT formula $\varphi$ create a graph $\mathbf{G}_\varphi$ such that

- $\mathbf{G}_\varphi$ has a Hamiltonian cycle if and only if $\varphi$ is satisfiable
- $\mathbf{G}_\varphi$ should be constructible from $\varphi$ by a polynomial time algorithm $\mathcal{A}$

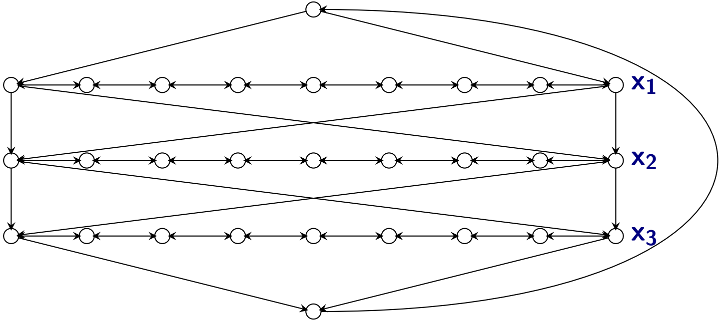
Notation: $\varphi$ has $\mathbf{n}$ variables $\mathbf{x_1, x_2, \ldots, x_n}$ and $\mathbf{m}$ clauses $\mathbf{C_1, C_2, \ldots, C_m}$.

# Reduction: First Ideas

- Viewing SAT: Assign values to $\mathbf{n}$ variables, and each clauses has 3 ways in which it can be satisfied
- Construct graph with $\mathbf{2^n}$ Hamiltonian cycles, where each cycle corresponds to some boolean assignment
- Then add more graph structure to encode constraints on assignments imposed by the clauses

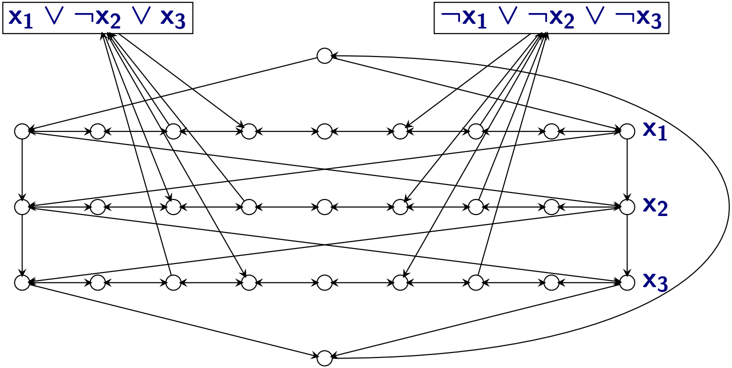
# The Reduction: Phase I

- Traverse path $\mathbf{i}$ from left to right iff $\mathbf{x_i}$ is set to true
- Each path has $\mathbf{3(m+1)}$ nodes where $\mathbf{m}$ is number of clauses in $\varphi$; nodes numbered from left to right ($\mathbf{1}$ to $\mathbf{3m+3}$)



# The Reduction: Phase II

- Add vertex $\mathbf{c_j}$ for clause $\mathbf{C_j}$. $\mathbf{c_j}$ has edge *from* vertex $\mathbf{3j}$ and *to* vertex $\mathbf{3j+1}$ on path $\mathbf{i}$ if $\mathbf{x_i}$ appears in clause $\mathbf{C_j}$, and has edge *from* vertex $\mathbf{3j+1}$ and *to* vertex $\mathbf{3j}$ if $\neg\mathbf{x_i}$ appears in $\mathbf{C_j}$.



# Correctness Proof

## Proposition

*$\varphi$ has a satisfying assignment iff $\mathbf{G}_\varphi$ has a Hamiltonian cycle*

## Proof.

$\Rightarrow$ Let $\mathbf{a}$ be the satisfying assignment for $\varphi$. Define Hamiltonian cycle as follows

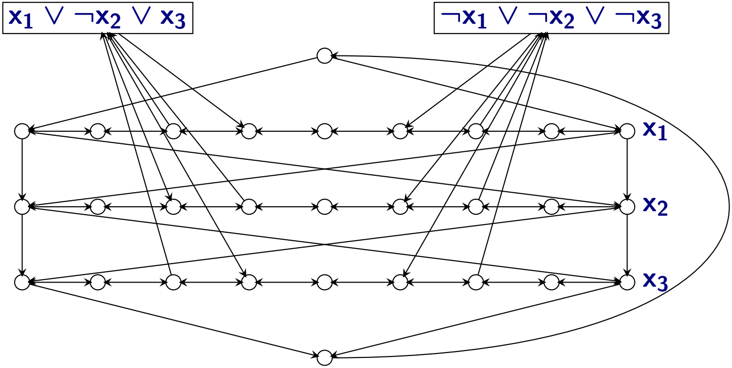
- If $\mathbf{a(x_i) = 1}$ then traverse path $\mathbf{i}$ from left to right
- If $\mathbf{a(x_i) = 0}$ then traverse path $\mathbf{i}$ from right to left
- For each clause, path of at least one variable is in the "right" direction to splice in the node corresponding to clause □

# Hamiltonian Cycle $\Rightarrow$ Satisfying assignment

Suppose $\Pi$ is a Hamiltonian cycle in $G_\varphi$

- If $\Pi$ enters $c_j$ (vertex for clause $C_j$) from vertex $3j$ on path $i$ then it must leave the clause vertex on edge to $3j+1$ on the *same path* $i$
  - If not, then only unvisited neighbor of $3j+1$ on path $i$ is $3j+2$
  - Thus, we don't have two unvisited neighbors (one to enter from, and the other to leave) to have a Hamiltonian Cycle
- Similarly, if $\Pi$ enters $c_j$ from vertex $3j+1$ on path $i$ then it must leave the clause vertex $c_j$ on edge to $3j$ on path $i$

# Example



# Hamiltonian Cycle $\implies$ Satisfying assignment (contd)

- Thus, vertices visited immediately before and after $C_i$ are connected by an edge
- We can remove $c_j$ from cycle, and get Hamiltonian cycle in $G - c_j$
- Consider Hamiltonian cycle in $G - \{c_1, \ldots c_m\}$; it traverses each path in only one direction, which determines the truth assignment

# Hamiltonian Cycle

## Problem

Input *Given undirected graph* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$

Goal *Does* $\mathbf{G}$ *have a Hamiltonian cycle? That is, is there a cycle that visits every vertex exactly one (except start and end vertex)?*

# **NP**-Completeness

## Theorem

**Hamiltonian cycle** *problem for undirected graphs is* **NP-Complete***.*

## Proof.

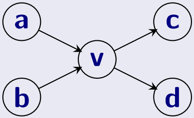
- The problem is in **NP**; proof left as exercise
- Hardness proved by reducing Directed Hamiltonian Cycle to this problem □

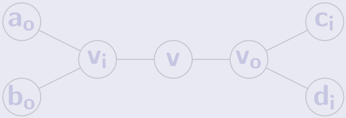
# Reduction Sketch

Goal: Given directed graph **G**, need to construct undirected graph **G′** such that **G** has Hamiltonian Path iff **G′** has Hamiltonian path

## Reduction

- Replace each vertex **v** by 3 vertices: $\mathbf{v_{in}}$, **v**, and $\mathbf{v_{out}}$
- A directed edge $(\mathbf{a}, \mathbf{b})$ is replaced by edge $(\mathbf{a_{out}}, \mathbf{b_{in}})$



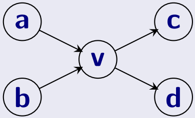



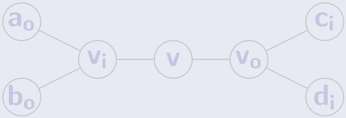
# Reduction Sketch

Goal: Given directed graph $G$, need to construct undirected graph $G'$ such that $G$ has Hamiltonian Path iff $G'$ has Hamiltonian path

## Reduction

- Replace each vertex $v$ by 3 vertices: $v_{in}$, $v$, and $v_{out}$
- A directed edge $(a, b)$ is replaced by edge $(a_{out}, b_{in})$



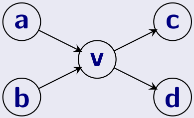



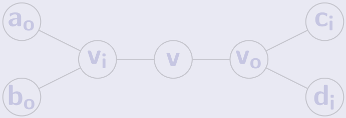
# Reduction Sketch

Goal: Given directed graph $\mathbf{G}$, need to construct undirected graph $\mathbf{G}'$ such that $\mathbf{G}$ has Hamiltonian Path iff $\mathbf{G}'$ has Hamiltonian path

## Reduction

- Replace each vertex $\mathbf{v}$ by 3 vertices: $\mathbf{v_{in}}, \mathbf{v}$, and $\mathbf{v_{out}}$
- A directed edge $(\mathbf{a}, \mathbf{b})$ is replaced by edge $(\mathbf{a_{out}}, \mathbf{b_{in}})$



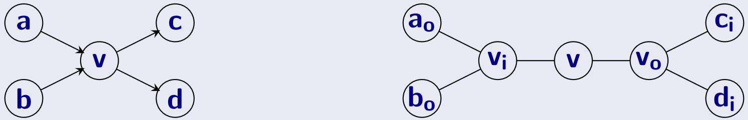



# Reduction Sketch

Goal: Given directed graph $G$, need to construct undirected graph $G'$ such that $G$ has Hamiltonian Path iff $G'$ has Hamiltonian path

## Reduction

- Replace each vertex $v$ by 3 vertices: $v_{in}$, $v$, and $v_{out}$
- A directed edge $(a, b)$ is replaced by edge $(a_{out}, b_{in})$



# Reduction: Wrapup

- The reduction is polynomial time (exercise)
- The reduction is correct (exercise)

# Graph Coloring

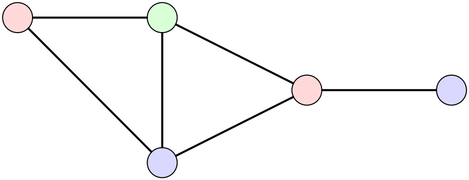
Input Given an undirected graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and integer $\mathbf{k}$

Goal Can the vertices of the graph be colored using $\mathbf{k}$ colors so that vertices connected by an edge do not get the same color?

# Graph 3-Coloring

**Input** Given an undirected graph $G = (V, E)$

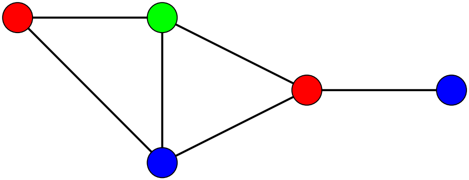
**Goal** Can the vertices of the graph be colored using $3$ colors so that vertices connected by an edge do not get the same color?



# Graph 3-Coloring

Input Given an undirected graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$

Goal Can the vertices of the graph be colored using $\mathbf{3}$ colors so that vertices connected by an edge do not get the same color?



# Graph Coloring

Observation: If $G$ is colored with $k$ colors then each color class (nodes of same color) form an independent set in $G$. Thus, $G$ can be partitioned into $k$ independent sets iff $G$ is $k$-colorable.

Graph $2$-Coloring can be decided in polynomial time.

$G$ is $2$-colorable iff $G$ is bipartite! There is a linear time algorithm to check if $G$ is bipartite using **BFS** (see book).

# Graph Coloring

Observation: If $G$ is colored with $k$ colors then each color class (nodes of same color) form an independent set in $G$. Thus, $G$ can be partitioned into $k$ independent sets iff $G$ is $k$-colorable.

Graph $2$-Coloring can be decided in polynomial time.

$G$ is $2$-colorable iff $G$ is bipartite! There is a linear time algorithm to check if $G$ is bipartite using **BFS** (see book).

# Graph Coloring

Observation: If **G** is colored with **k** colors then each color class (nodes of same color) form an independent set in **G**. Thus, **G** can be partitioned into **k** independent sets iff **G** is **k**-colorable.

Graph **2**-Coloring can be decided in polynomial time.

**G** is **2**-colorable iff **G** is bipartite! There is a linear time algorithm to check if **G** is bipartite using **BFS** (see book).

# Graph Coloring

Observation: If $G$ is colored with $k$ colors then each color class (nodes of same color) form an independent set in $G$. Thus, $G$ can be partitioned into $k$ independent sets iff $G$ is $k$-colorable.

Graph $2$-Coloring can be decided in polynomial time.

$G$ is $2$-colorable iff $G$ is bipartite! There is a linear time algorithm to check if $G$ is bipartite using **BFS** (see book).

# Graph Coloring and Register Allocation

## Register Allocation

Assign variables to (at most) **k** registers such that variables needed at the same time are not assigned to the same register

## Interference Graph

Vertices are variables, and there is an edge between two vertices, if the two variables are "live" at the same time.

## Observations

- [Chaitin] Register allocation problem is equivalent to coloring the interference graph with **k** colors
- Moreover, **3-COLOR** $\leq_P$ **k-Register Allocation**, for any $k \geq 3$

# Class Room Scheduling

Given **n** classes and their meeting times, are **k** rooms sufficient?

Reduce to Graph **k**-Coloring problem

Create graph **G**

- a node $v_i$ for each class **i**
- an edge between $v_i$ and $v_j$ if classes **i** and **j** *conflict*

Exercise: **G** is **k**-colorable iff **k** rooms are sufficient

# Class Room Scheduling

Given $\mathbf{n}$ classes and their meeting times, are $\mathbf{k}$ rooms sufficient?

Reduce to Graph $\mathbf{k}$-Coloring problem

Create graph $\mathbf{G}$

- a node $\mathbf{v_i}$ for each class $\mathbf{i}$
- an edge between $\mathbf{v_i}$ and $\mathbf{v_j}$ if classes $\mathbf{i}$ and $\mathbf{j}$ *conflict*

Exercise: $\mathbf{G}$ is $\mathbf{k}$-colorable iff $\mathbf{k}$ rooms are sufficient

# Class Room Scheduling

Given $\mathbf{n}$ classes and their meeting times, are $\mathbf{k}$ rooms sufficient?

Reduce to Graph $\mathbf{k}$-Coloring problem

Create graph $\mathbf{G}$

- a node $\mathbf{v_i}$ for each class $\mathbf{i}$
- an edge between $\mathbf{v_i}$ and $\mathbf{v_j}$ if classes $\mathbf{i}$ and $\mathbf{j}$ *conflict*

Exercise: $\mathbf{G}$ is $\mathbf{k}$-colorable iff $\mathbf{k}$ rooms are sufficient

# Frequency Assignments in Cellular Networks

Cellular telephone systems that use Frequency Division Multiple Access (FDMA) (example: GSM in Europe and Asia and AT&T in USA)

- Breakup a frequency range $[a, b]$ into disjoint *bands* of frequencies $[a_0, b_0], [a_1, b_1], \ldots, [a_k, b_k]$
- Each cell phone tower (simplifying) gets one band
- Constraint: nearby towers cannot be assigned same band, otherwise signals will interference

Problem: given $k$ bands and some region with $n$ towers, is there a way to assign the bands to avoid interference?

Can reduce to $k$-coloring by creating intereference/conflict graph on towers

# Frequency Assignments in Cellular Networks

Cellular telephone systems that use Frequency Division Multiple Access (FDMA) (example: GSM in Europe and Asia and AT&T in USA)

- Breakup a frequency range $[a, b]$ into disjoint *bands* of frequencies $[a_0, b_0], [a_1, b_1], \ldots, [a_k, b_k]$
- Each cell phone tower (simplifying) gets one band
- Constraint: nearby towers cannot be assigned same band, otherwise signals will interference

Problem: given $k$ bands and some region with $n$ towers, is there a way to assign the bands to avoid interference?

Can reduce to $k$-coloring by creating intereference/conflict graph on towers

# 3-Coloring is NP-Complete

- 3-Coloring is in **NP**
  - Certificate: for each node a color from $\{1, 2, 3\}$
  - Certifier: Check if for each edge $(u, v)$, the color of $u$ is different from that of $v$
- Hardness: We will show **3-SAT** $\leq_P$ **3-Coloring**

# Reduction Idea

Start with **3SAT** formula (i.e., $3\mathrm{CNF}$ formula) $\varphi$ with $n$ variables $x_1, \ldots, x_n$ and $m$ clauses $C_1, \ldots, C_m$. Create graph $G_\varphi$ such that $G_\varphi$ is 3-colorable iff $\varphi$ is satisfiable

- need to establish truth assignment for $x_1, \ldots, x_n$ via colors for some nodes in $G_\varphi$.
- create triangle with node True, False, Base
- for each variable $x_i$ two nodes $v_i$ and $\bar{v}_i$ connected in a triangle with common Base
- If graph is 3-colored, either $v_i$ or $\bar{v}_i$ gets the same color as True. Interpret this as a truth assignment to $v_i$
- Need to add constraints to ensure clauses are satisfied (next phase)

# Reduction Idea

Start with **3SAT** formula (i.e., **3CNF** formula) $\varphi$ with $\mathbf{n}$ variables $\mathbf{x_1, \ldots, x_n}$ and $\mathbf{m}$ clauses $\mathbf{C_1, \ldots, C_m}$. Create graph $\mathbf{G}_\varphi$ such that $\mathbf{G}_\varphi$ is 3-colorable iff $\varphi$ is satisfiable

- need to establish truth assignment for $\mathbf{x_1, \ldots, x_n}$ via colors for some nodes in $\mathbf{G}_\varphi$.
- create triangle with node True, False, Base
- for each variable $\mathbf{x_i}$ two nodes $\mathbf{v_i}$ and $\mathbf{\bar{v}_i}$ connected in a triangle with common Base
- If graph is 3-colored, either $\mathbf{v_i}$ or $\mathbf{\bar{v}_i}$ gets the same color as True. Interpret this as a truth assignment to $\mathbf{v_i}$
- Need to add constraints to ensure clauses are satisfied (next phase)

# Reduction Idea

Start with **3SAT** formula (i.e., $\mathbf{3}\mathrm{CNF}$ formula) $\varphi$ with $\mathbf{n}$ variables $\mathbf{x_1, \ldots, x_n}$ and $\mathbf{m}$ clauses $\mathbf{C_1, \ldots, C_m}$. Create graph $\mathbf{G}_\varphi$ such that $\mathbf{G}_\varphi$ is 3-colorable iff $\varphi$ is satisfiable

- need to establish truth assignment for $\mathbf{x_1, \ldots, x_n}$ via colors for some nodes in $\mathbf{G}_\varphi$.
- create triangle with node True, False, Base
- for each variable $\mathbf{x_i}$ two nodes $\mathbf{v_i}$ and $\bar{\mathbf{v}}_\mathbf{i}$ connected in a triangle with common Base
- If graph is 3-colored, either $\mathbf{v_i}$ or $\bar{\mathbf{v}}_\mathbf{i}$ gets the same color as True. Interpret this as a truth assignment to $\mathbf{v_i}$
- Need to add constraints to ensure clauses are satisfied (next phase)

# Reduction Idea

Start with **3SAT** formula (i.e., $\mathbf{3}\mathrm{CNF}$ formula) $\varphi$ with $\mathbf{n}$ variables $\mathbf{x_1, \ldots, x_n}$ and $\mathbf{m}$ clauses $\mathbf{C_1, \ldots, C_m}$. Create graph $\mathbf{G}_\varphi$ such that $\mathbf{G}_\varphi$ is 3-colorable iff $\varphi$ is satisfiable

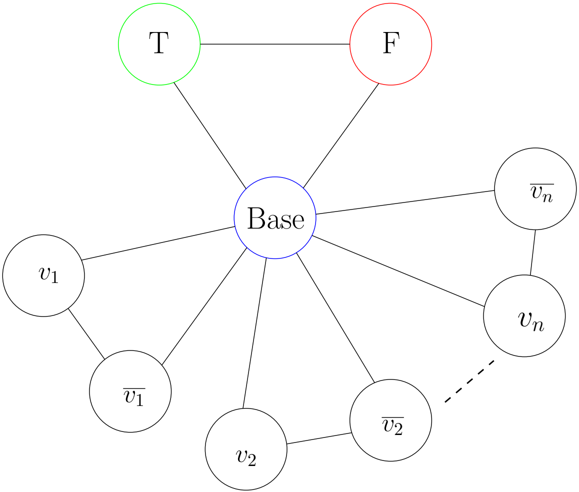
- need to establish truth assignment for $\mathbf{x_1, \ldots, x_n}$ via colors for some nodes in $\mathbf{G}_\varphi$.
- create triangle with node True, False, Base
- for each variable $\mathbf{x_i}$ two nodes $\mathbf{v_i}$ and $\mathbf{\bar{v}_i}$ connected in a triangle with common Base
- If graph is 3-colored, either $\mathbf{v_i}$ or $\mathbf{\bar{v}_i}$ gets the same color as True. Interpret this as a truth assignment to $\mathbf{v_i}$
- Need to add constraints to ensure clauses are satisfied (next phase)

# Reduction Idea

Start with **3SAT** formula (i.e., $3\mathrm{CNF}$ formula) $\varphi$ with $n$ variables $x_1, \ldots, x_n$ and $m$ clauses $C_1, \ldots, C_m$. Create graph $G_\varphi$ such that $G_\varphi$ is 3-colorable iff $\varphi$ is satisfiable

- need to establish truth assignment for $x_1, \ldots, x_n$ via colors for some nodes in $G_\varphi$.
- create triangle with node True, False, Base
- for each variable $x_i$ two nodes $v_i$ and $\bar{v}_i$ connected in a triangle with common Base
- If graph is 3-colored, either $v_i$ or $\bar{v}_i$ gets the same color as True. Interpret this as a truth assignment to $v_i$
- Need to add constraints to ensure clauses are satisfied (next phase)

# Figure

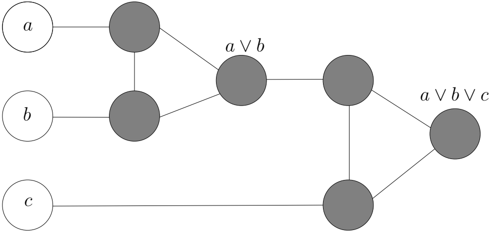


# Clause Satisfiability Gadget

For each clause $\mathbf{C_j = (a \lor b \lor c)}$, create a small gadget graph

- gadget graph connects to nodes corresponding to **a, b, c**
- needs to implement OR

OR-gadget-graph:



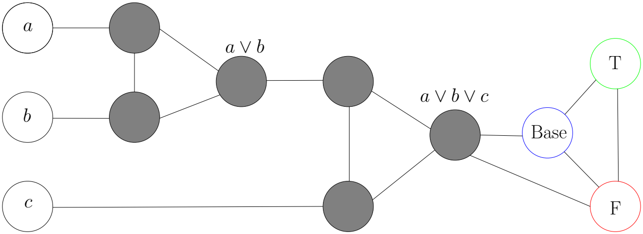
# OR-Gadget Graph

Property: if $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are colored False in a 3-coloring then output node of OR-gadget has to be colored False.

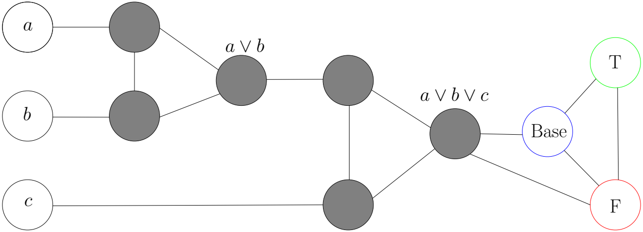
Property: if one of $\mathbf{a}, \mathbf{b}, \mathbf{c}$ is colored True then OR-gadget can be 3-colored such that output node of OR-gadget is colored True.

# Reduction

- create triangle with nodes True, False, Base
- for each variable $\mathbf{x_i}$ two nodes $\mathbf{v_i}$ and $\bar{\mathbf{v}}_\mathbf{i}$ connected in a triangle with common Base
- for each clause $\mathbf{C_j = (a \vee b \vee c)}$, add OR-gadget graph with input nodes $\mathbf{a, b, c}$ and connect output node of gadget to both False and Base



# Reduction



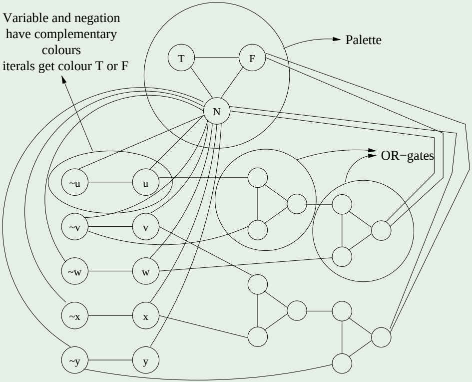

## Claim

*No legal* **3***-coloring of above graph (with coloring of nodes* **T, F, B** *fixed) in which* **a, b, c** *are colored False. If any of* **a, b, c** *are colored True then there is a legal* **3***-coloring of above graph.*

# Reduction Outline

## Example

$\varphi = (\mathbf{u} \vee \neg\mathbf{v} \vee \mathbf{w}) \wedge (\mathbf{v} \vee \mathbf{x} \vee \neg\mathbf{y})$



# Correctness of Reduction

$\varphi$ is satisfiable implies $\mathbf{G}_\varphi$ is 3-colorable

- if $\mathbf{x_i}$ is assigned True, color $\mathbf{v_i}$ True and $\mathbf{\bar{v}_i}$ False
- for each clause $\mathbf{C_j} = (\mathbf{a} \vee \mathbf{b} \vee \mathbf{c})$ at least one of $\mathbf{a}, \mathbf{b}, \mathbf{c}$ is colored True. OR-gadget for $\mathbf{C_j}$ can be 3-colored such that output is True.

$\mathbf{G}_\varphi$ is 3-colorable implies $\varphi$ is satisfiable

- if $\mathbf{v_i}$ is colored True then set $\mathbf{x_i}$ to be True, this is a legal truth assignment
- consider any clause $\mathbf{C_j} = (\mathbf{a} \vee \mathbf{b} \vee \mathbf{c})$. it cannot be that all $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are False. If so, output of OR-gadget for $\mathbf{C_j}$ has to be colored False but output is connected to Base and False!

# Correctness of Reduction

$\varphi$ is satisfiable implies $\mathbf{G}_\varphi$ is 3-colorable

- if $\mathbf{x_i}$ is assigned True, color $\mathbf{v_i}$ True and $\bar{\mathbf{v}}_\mathbf{i}$ False
- for each clause $\mathbf{C_j = (a \vee b \vee c)}$ at least one of $\mathbf{a, b, c}$ is colored True. OR-gadget for $\mathbf{C_j}$ can be 3-colored such that output is True.

$\mathbf{G}_\varphi$ is 3-colorable implies $\varphi$ is satisfiable

- if $\mathbf{v_i}$ is colored True then set $\mathbf{x_i}$ to be True, this is a legal truth assignment
- consider any clause $\mathbf{C_j = (a \vee b \vee c)}$. it cannot be that all $\mathbf{a, b, c}$ are False. If so, output of OR-gadget for $\mathbf{C_j}$ has to be colored False but output is connected to Base and False!

# Correctness of Reduction

$\varphi$ is satisfiable implies $\mathbf{G}_\varphi$ is 3-colorable

- if $\mathbf{x_i}$ is assigned True, color $\mathbf{v_i}$ True and $\bar{\mathbf{v}}_\mathbf{i}$ False
- for each clause $\mathbf{C_j = (a \lor b \lor c)}$ at least one of $\mathbf{a, b, c}$ is colored True. OR-gadget for $\mathbf{C_j}$ can be 3-colored such that output is True.

$\mathbf{G}_\varphi$ is 3-colorable implies $\varphi$ is satisfiable

- if $\mathbf{v_i}$ is colored True then set $\mathbf{x_i}$ to be True, this is a legal truth assignment
- consider any clause $\mathbf{C_j = (a \lor b \lor c)}$. it cannot be that all $\mathbf{a, b, c}$ are False. If so, output of OR-gadget for $\mathbf{C_j}$ has to be colored False but output is connected to Base and False!

# Correctness of Reduction

$\varphi$ is satisfiable implies $\mathbf{G}_\varphi$ is 3-colorable

- if $\mathbf{x_i}$ is assigned True, color $\mathbf{v_i}$ True and $\bar{\mathbf{v}}_\mathbf{i}$ False
- for each clause $\mathbf{C_j} = (\mathbf{a} \vee \mathbf{b} \vee \mathbf{c})$ at least one of $\mathbf{a}, \mathbf{b}, \mathbf{c}$ is colored True. OR-gadget for $\mathbf{C_j}$ can be 3-colored such that output is True.

$\mathbf{G}_\varphi$ is 3-colorable implies $\varphi$ is satisfiable

- if $\mathbf{v_i}$ is colored True then set $\mathbf{x_i}$ to be True, this is a legal truth assignment
- consider any clause $\mathbf{C_j} = (\mathbf{a} \vee \mathbf{b} \vee \mathbf{c})$. it cannot be that all $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are False. If so, output of OR-gadget for $\mathbf{C_j}$ has to be colored False but output is connected to Base and False!

# Correctness of Reduction

$\varphi$ is satisfiable implies $\mathbf{G}_\varphi$ is 3-colorable

- if $\mathbf{x_i}$ is assigned True, color $\mathbf{v_i}$ True and $\bar{\mathbf{v}}_\mathbf{i}$ False
- for each clause $\mathbf{C_j = (a \vee b \vee c)}$ at least one of $\mathbf{a, b, c}$ is colored True. OR-gadget for $\mathbf{C_j}$ can be 3-colored such that output is True.

$\mathbf{G}_\varphi$ is 3-colorable implies $\varphi$ is satisfiable

- if $\mathbf{v_i}$ is colored True then set $\mathbf{x_i}$ to be True, this is a legal truth assignment
- consider any clause $\mathbf{C_j = (a \vee b \vee c)}$. it cannot be that all $\mathbf{a, b, c}$ are False. If so, output of OR-gadget for $\mathbf{C_j}$ has to be colored False but output is connected to Base and False!

# Other NP-Complete Problems

- 3-Dimensional Matching
- Subset Sum

Read book.

# Need to Know **NP-Complete** Problems

- 3-SAT
- Circuit-SAT
- Independent Set
- Vertex Cover
- Clique
- Set Cover
- Hamiltonian Cycle in Directed/Undirected Graphs
- 3-Coloring
- 3-D Matching
- Subset Sum

# Subset Sum and Knapsack

Subset Sum Problem: Given $\mathbf{n}$ integers $\mathbf{a_1, a_2, \ldots, a_n}$ and a target $\mathbf{B}$, is there a subset of $\mathbf{S}$ of $\{\mathbf{a_1, \ldots, a_n}\}$ such that the numbers in $\mathbf{S}$ add up *precisely* to $\mathbf{B}$?

Subset Sum is **NP-Complete**— see book.

Knapsack: Given $\mathbf{n}$ items with item $\mathbf{i}$ having size $\mathbf{s_i}$ and profit $\mathbf{p_i}$, a knapsack of capacity $\mathbf{B}$, and a target profit $\mathbf{P}$, is there a subset $\mathbf{S}$ of items that can be packed in the knapsack and the profit of $\mathbf{S}$ is at least $\mathbf{P}$?

Show Knapsack problem is **NP-Complete** via reduction from Subset Sum (exercise).

# Subset Sum and Knapsack

Subset Sum Problem: Given $\mathbf{n}$ integers $\mathbf{a_1, a_2, \ldots, a_n}$ and a target $\mathbf{B}$, is there a subset of $\mathbf{S}$ of $\{\mathbf{a_1, \ldots, a_n}\}$ such that the numbers in $\mathbf{S}$ add up *precisely* to $\mathbf{B}$?

Subset Sum is **NP-Complete**— see book.

Knapsack: Given $\mathbf{n}$ items with item $\mathbf{i}$ having size $\mathbf{s_i}$ and profit $\mathbf{p_i}$, a knapsack of capacity $\mathbf{B}$, and a target profit $\mathbf{P}$, is there a subset $\mathbf{S}$ of items that can be packed in the knapsack and the profit of $\mathbf{S}$ is at least $\mathbf{P}$?

Show Knapsack problem is **NP-Complete** via reduction from Subset Sum (exercise).

# Subset Sum and Knapsack

Subset Sum Problem: Given $\mathbf{n}$ integers $\mathbf{a_1, a_2, \ldots, a_n}$ and a target $\mathbf{B}$, is there a subset of $\mathbf{S}$ of $\{\mathbf{a_1, \ldots, a_n}\}$ such that the numbers in $\mathbf{S}$ add up *precisely* to $\mathbf{B}$?

Subset Sum is **NP-Complete**— see book.

Knapsack: Given $\mathbf{n}$ items with item $\mathbf{i}$ having size $\mathbf{s_i}$ and profit $\mathbf{p_i}$, a knapsack of capacity $\mathbf{B}$, and a target profit $\mathbf{P}$, is there a subset $\mathbf{S}$ of items that can be packed in the knapsack and the profit of $\mathbf{S}$ is at least $\mathbf{P}$?

Show Knapsack problem is **NP-Complete** via reduction from Subset Sum (exercise).

# Subset Sum and Knapsack

Subset Sum Problem: Given $n$ integers $a_1, a_2, \ldots, a_n$ and a target $B$, is there a subset of $S$ of $\{a_1, \ldots, a_n\}$ such that the numbers in $S$ add up *precisely* to $B$?

Subset Sum is **NP-Complete**— see book.

Knapsack: Given $n$ items with item $i$ having size $s_i$ and profit $p_i$, a knapsack of capacity $B$, and a target profit $P$, is there a subset $S$ of items that can be packed in the knapsack and the profit of $S$ is at least $P$?

Show Knapsack problem is **NP-Complete** via reduction from Subset Sum (exercise).

# Subset Sum and Knapsack

Subset Sum can be solved in $O(nB)$ time using dynamic programming (exercise).

Implies that problem is hard only when numbers $a_1, a_2, \ldots, a_n$ are exponentially large compared to $n$. That is, each $a_i$ requires polynomial in $n$ bits.

*Number problems* of the above type are said to be **weakly NP-Complete**.

# Subset Sum and Knapsack

Subset Sum can be solved in $O(nB)$ time using dynamic programming (exercise).

Implies that problem is hard only when numbers $a_1, a_2, \ldots, a_n$ are exponentially large compared to $n$. That is, each $a_i$ requires polynomial in $n$ bits.

*Number problems* of the above type are said to be **weakly NP-Complete**.

# Subset Sum and Knapsack

Subset Sum can be solved in $O(nB)$ time using dynamic programming (exercise).

Implies that problem is hard only when numbers $a_1, a_2, \ldots, a_n$ are exponentially large compared to $n$. That is, each $a_i$ requires polynomial in $n$ bits.

*Number problems* of the above type are said to be **weakly NP-Complete**.

# Notes

# Notes

# Notes

# Notes