

# 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**,  $L' \leq_P L$

$L$  is **NP-HARD** if for every  $L'$  in **NP**,  $L' \leq_P L$ .

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  $L'$  in **NP**,  $L' \leq_P L$

$L$  is **NP-HARD** if for every  $L'$  in **NP**,  $L' \leq_P L$ .

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  $L'$  in **NP**,  $L' \leq_P L$

$L$  is **NP-HARD** if for every  $L'$  in **NP**,  $L' \leq_P L$ .

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  $L'$  in **NP**,  $L' \leq_P L$

$L$  is **NP-HARD** if for every  $L'$  in **NP**,  $L' \leq_P L$ .

## Theorem (Cook-Levin)

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

## Theorem (Cook-Levin)

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

Establish NP-COMPLETEness 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

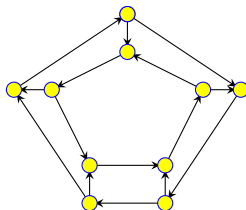
- Hamiltonian Cycle Problem is NP-COMPLETE
- 3-Coloring is NP-COMPLETE

# Directed Hamiltonian Cycle

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

**Goal** Does  $G$  have a **Hamiltonian cycle**?

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



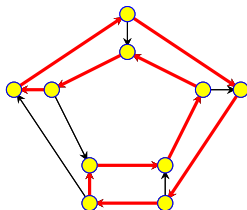


# Directed Hamiltonian Cycle

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

**Goal** Does  $G$  have a **Hamiltonian cycle**?

- A Hamiltonian cycle is a cycle in the graph that visits every vertex in  $G$  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  
 **$3\text{-SAT} \leq_P \text{Directed Hamiltonian Cycle}$**

# Reduction

Given 3-SAT formula  $\varphi$  create a graph  $G_\varphi$  such that

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

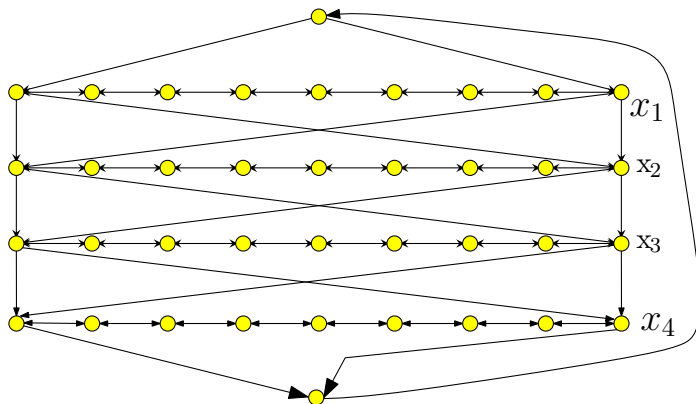
**Notation:**  $\varphi$  has  $n$  variables  $x_1, x_2, \dots, x_n$  and  $m$  clauses  $C_1, C_2, \dots, C_m$ .

# Reduction: First Ideas

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

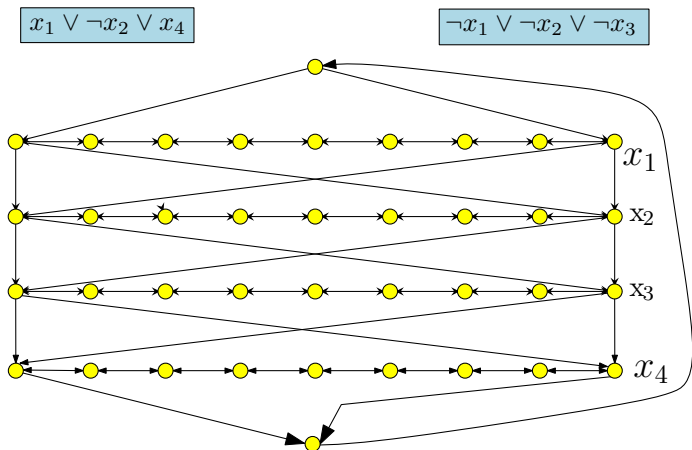
# The Reduction: Phase I

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



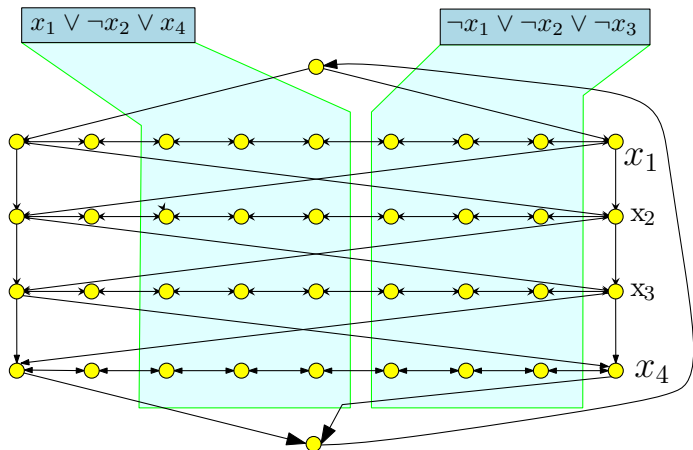
# The Reduction: Phase II

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



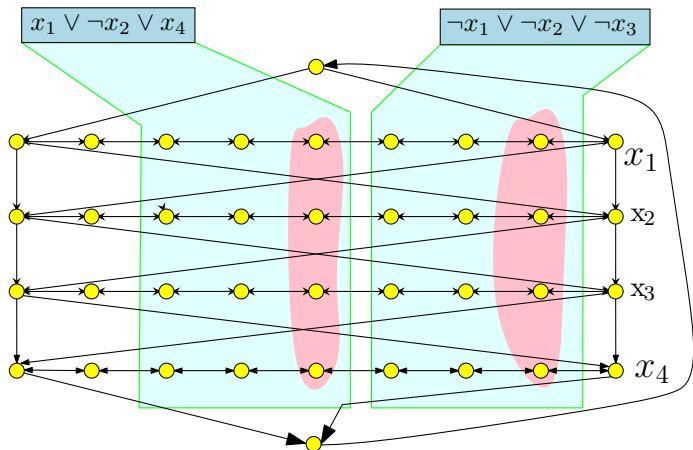
# The Reduction: Phase II

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



# The Reduction: Phase II

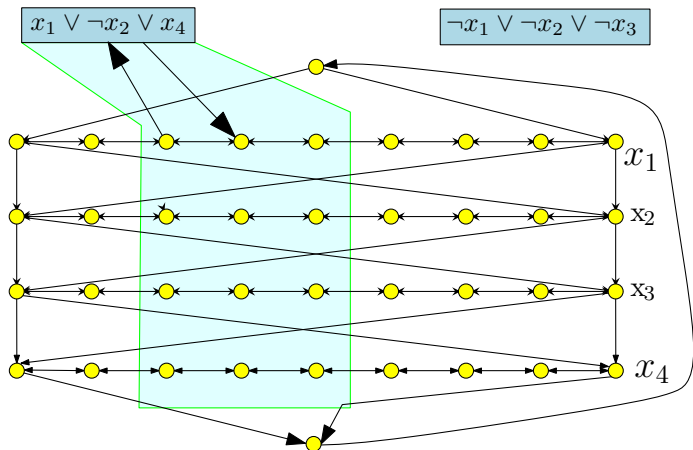
- Add vertex  $c_j$  for clause  $C_j$ .  $c_j$  has edge from vertex  $3j$  and to vertex  $3j + 1$  on path  $i$  if  $x_i$  appears in clause  $C_j$ , and has edge from vertex  $3j + 1$  and to vertex  $3j$  if  $\neg x_i$  appears in  $C_j$ .





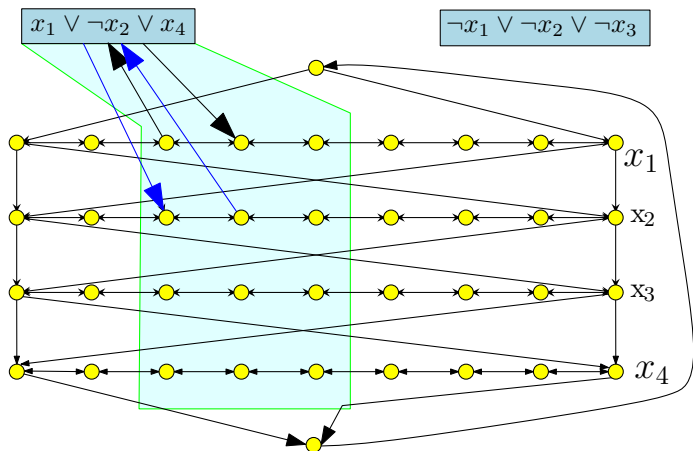
# The Reduction: Phase II

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



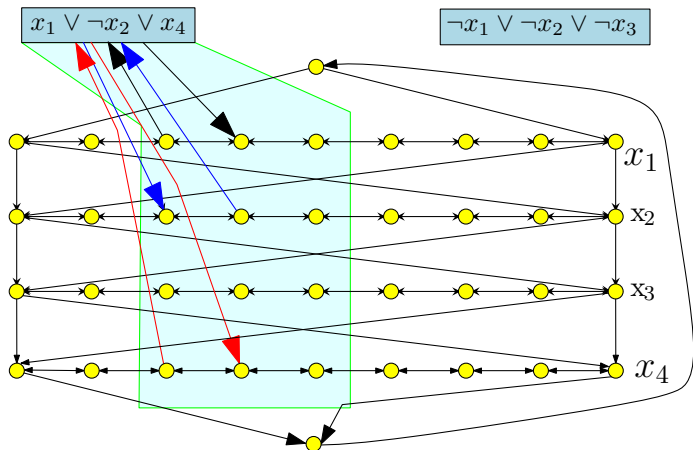
# The Reduction: Phase II

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



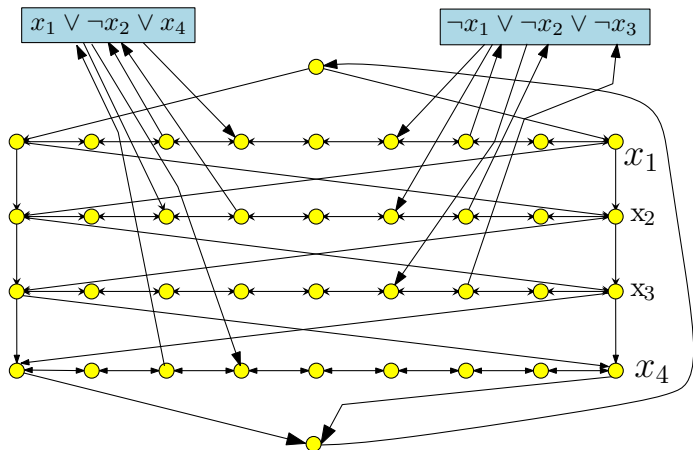
# The Reduction: Phase II

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



# The Reduction: Phase II

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



# Correctness Proof

## Proposition

$\varphi$  has a satisfying assignment iff  $G_\varphi$  has a Hamiltonian cycle.

## Proof.

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

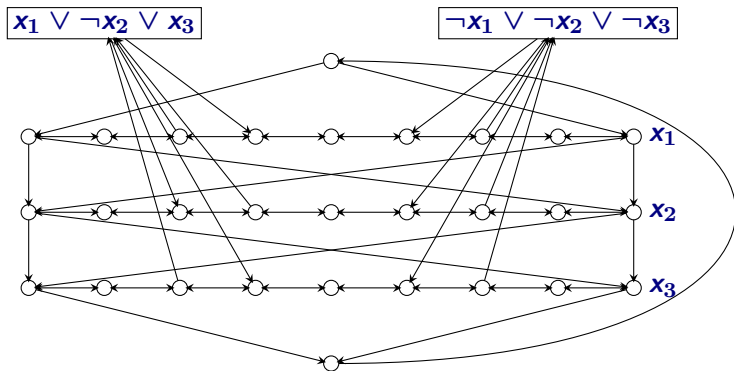
- If  $\mathbf{a}(x_i) = 1$  then traverse path  $i$  from left to right
- If  $\mathbf{a}(x_i) = 0$  then traverse path  $i$  from right to left
- For each clause, path of at least one variable is in the “right” direction to splice in the node corresponding to clause □

# 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  $G - \{C_1, \dots, C_m\}$ ; it traverses each path in only one direction, which determines the truth assignment



# Hamiltonian Cycle

## Problem

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

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

## Theorem

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

## Proof.

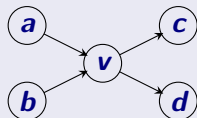
- The problem is in *NP*; proof left as exercise
- Hardness proved by reducing Directed Hamiltonian Cycle to this problem □

# 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:  $v_{in}$ ,  $v$ , and  $v_{out}$
- A directed edge  $(a, b)$  is replaced by edge  $(a_{out}, b_{in})$

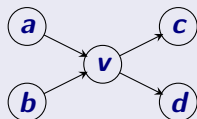


# Reduction Sketch

**Goal:** Given directed graph  $G$ , 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  $(a_{out}, b_{in})$

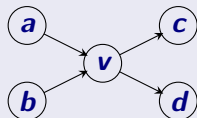


# Reduction Sketch

**Goal:** Given directed graph  $G$ , 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  $(a_{out}, b_{in})$

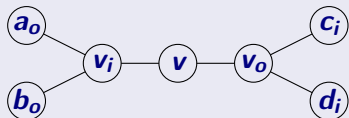
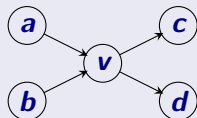


# Reduction Sketch

**Goal:** Given directed graph  $G$ , 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  $G = (V, E)$  and integer  $k$

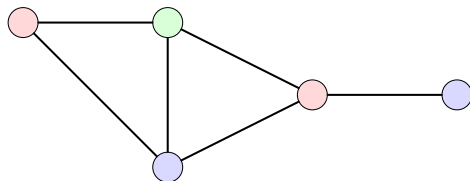
**Goal** Can the vertices of the graph be colored using  $k$  colors so that vertices connected by an edge do not get the same color?



# Graph 3-Coloring

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

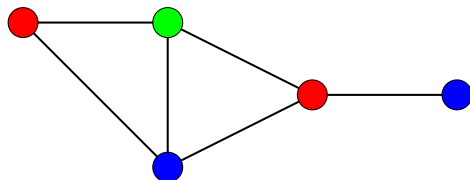
**Goal** Can the vertices of the graph be colored using **3** colors so that vertices connected by an edge do not get the same color?



# Graph 3-Coloring

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

**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 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,  $3\text{-COLOR} \leq_P k\text{-Register Allocation}$ , 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  $n$  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  $n$  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

# 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  $[a_0, b_0], [a_1, b_1], \dots, [a_k, b_k]$
- 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 interference/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  $[a_0, b_0], [a_1, b_1], \dots, [a_k, b_k]$
- 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 interference/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  $3\text{-SAT} \leq_P 3\text{-Coloring}$

# Reduction Idea

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

- need to establish truth assignment for  $x_1, \dots, x_n$  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  $n$  variables  $x_1, \dots, x_n$  and  $m$  clauses  $C_1, \dots, C_m$ . Create graph  $G_\varphi$  such that  $G_\varphi$  is 3-colorable iff  $\varphi$  is satisfiable

- need to establish truth assignment for  $x_1, \dots, x_n$  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  $n$  variables  $x_1, \dots, x_n$  and  $m$  clauses  $C_1, \dots, C_m$ . Create graph  $G_\varphi$  such that  $G_\varphi$  is 3-colorable iff  $\varphi$  is satisfiable

- need to establish truth assignment for  $x_1, \dots, x_n$  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  $n$  variables  $x_1, \dots, x_n$  and  $m$  clauses  $C_1, \dots, C_m$ . Create graph  $G_\varphi$  such that  $G_\varphi$  is 3-colorable iff  $\varphi$  is satisfiable

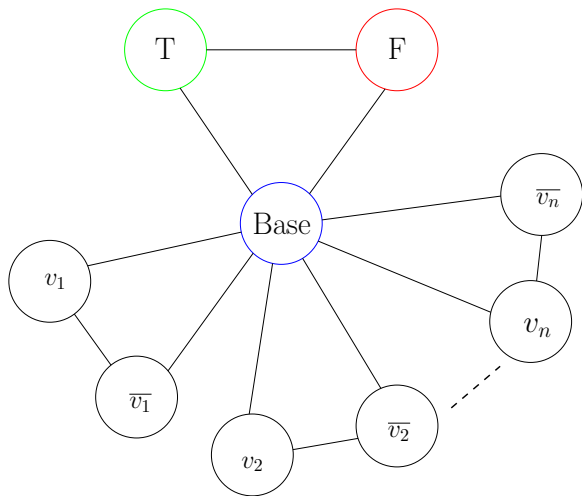
- need to establish truth assignment for  $x_1, \dots, x_n$  via colors for some nodes in  $G_\varphi$ .
- create triangle with node True, False, Base
- for each variable  $x_i$  two nodes  $v_i$  and  $\bar{v}_i$  connected in a triangle with common Base
- If graph is 3-colored, either  $v_i$  or  $\bar{v}_i$  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  $n$  variables  $x_1, \dots, x_n$  and  $m$  clauses  $C_1, \dots, C_m$ . Create graph  $G_\varphi$  such that  $G_\varphi$  is 3-colorable iff  $\varphi$  is satisfiable

- need to establish truth assignment for  $x_1, \dots, x_n$  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

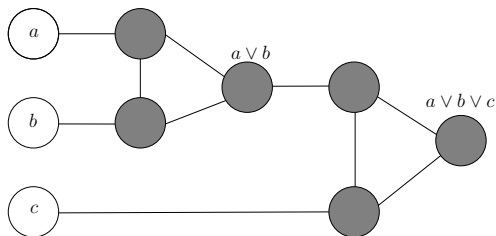


# Clause Satisfiability Gadget

For each clause  $C_j = (a \vee b \vee c)$ , create a small gadget graph

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

OR-gadget-graph:



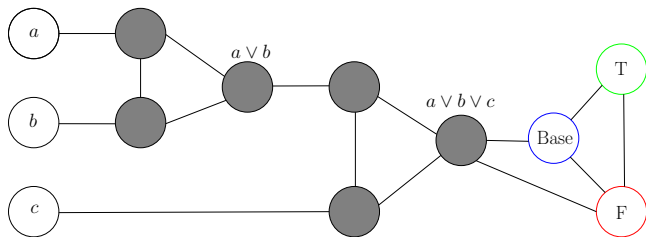
# OR-Gadget Graph

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

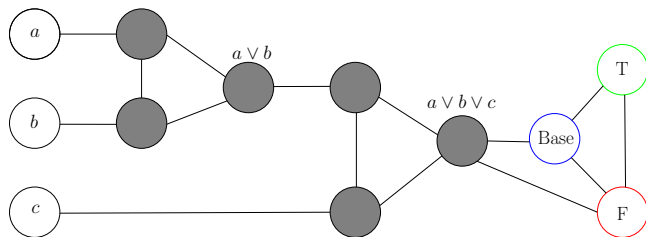
**Property:** if one of  $a, b, c$  is colored True then OR-gadget can be 3-colored such that output node of OR-gadget is colored True.

# Reduction

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



# Reduction



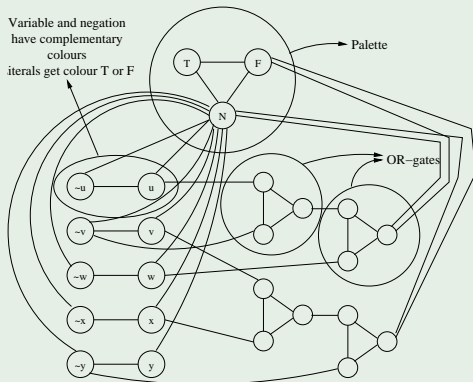
## Claim

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

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





# Correctness of Reduction

$\varphi$  is satisfiable implies  $G_\varphi$  is 3-colorable

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

$G_\varphi$  is 3-colorable implies  $\varphi$  is satisfiable

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

# Correctness of Reduction

$\varphi$  is satisfiable implies  $G_\varphi$  is 3-colorable

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

$G_\varphi$  is 3-colorable implies  $\varphi$  is satisfiable

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

# Correctness of Reduction

$\varphi$  is satisfiable implies  $G_\varphi$  is 3-colorable

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

$G_\varphi$  is 3-colorable implies  $\varphi$  is satisfiable

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

# Correctness of Reduction

$\varphi$  is satisfiable implies  $G_\varphi$  is 3-colorable

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

$G_\varphi$  is 3-colorable implies  $\varphi$  is satisfiable

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

# Correctness of Reduction

$\varphi$  is satisfiable implies  $G_\varphi$  is 3-colorable

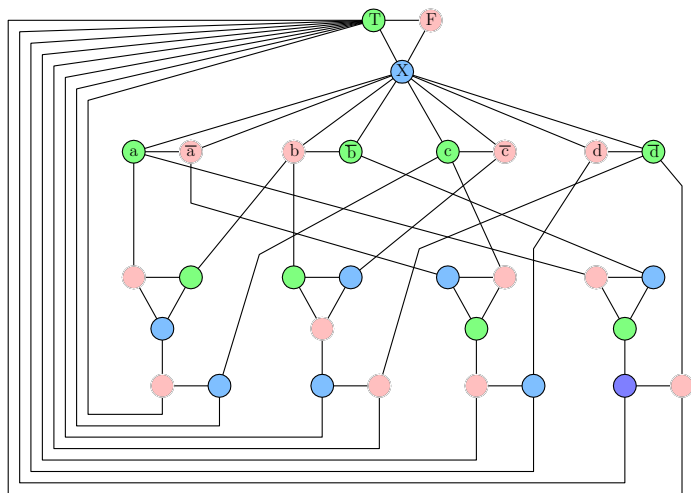
- if  $x_i$  is assigned True, color  $v_i$  True and  $\bar{v}_i$  False
- for each clause  $C_j = (a \vee b \vee c)$  at least one of  $a, b, c$  is colored True. OR-gadget for  $C_j$  can be 3-colored such that output is True.

$G_\varphi$  is 3-colorable implies  $\varphi$  is satisfiable

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

# Graph generated in reduction...

... from **3SAT** to **3COLOR**



# Subset Sum

## Problem: **Subset Sum**

**Instance:**  $S$  - set of positive integers,  $t$ : - an integer number (Target)

**Question:** Is there a subset  $X \subseteq S$  such that  $\sum_{x \in X} x = t$ ?

## Claim

**Subset Sum** is NP-COMPLETE.

# Vec Subset Sum

We will prove following problem is NP-COMplete...

## Problem: Vec Subset Sum

**Instance:**  $S$  - set of  $n$  vectors of dimension  $k$ , each vector has non-negative numbers for its coordinates, and a target vector  $\vec{t}$ .

**Question:** Is there a subset  $X \subseteq S$  such that  $\sum_{\vec{x} \in X} \vec{x} = \vec{t}$ ?

Reduction from 3SAT.



# Vec Subset Sum

## Handling a single clause

Think about vectors as being lines in a table.

### First gadget

Selecting between two lines.

Target	??	??	01	???
$a_1$	??	??	01	??
$a_2$	??	??	01	??

Two rows for every variable  $x$ : selecting either  $x = 0$  or  $x = 1$ .

# Handling a clause...

We will have a column for every clause...

numbers	...	$C \equiv a \vee b \vee \bar{c}$	...
$a$	...	01	...
$\bar{a}$	...	00	...
$b$	...	01	...
$\bar{b}$	...	00	...
$c$	...	00	...
$\bar{c}$	...	01	...
$C$ fix-up 1	000	07	000
$C$ fix-up 2	000	08	000
$C$ fix-up 3	000	09	000
TARGET		10	

# 3SAT to Vec Subset Sum

numbers	$a \vee \bar{a}$	$b \vee \bar{b}$	$c \vee \bar{c}$	$d \vee \bar{d}$	$D \equiv \bar{b} \vee c \vee \bar{d}$	$C \equiv a \vee b \vee \bar{c}$
$a$	1	0	0	0	00	01
$\bar{a}$	1	0	0	0	00	00
$b$	0	1	0	0	00	01
$\bar{b}$	0	1	0	0	01	00
$c$	0	0	1	0	01	00
$\bar{c}$	0	0	1	0	00	01
$d$	0	0	0	1	00	00
$\bar{d}$	0	0	0	1	01	01
$C$ fix-up 1	0	0	0	0	00	07
$C$ fix-up 2	0	0	0	0	00	08
$C$ fix-up 3	0	0	0	0	00	09
$D$ fix-up 1	0	0	0	0	07	00
$D$ fix-up 2	0	0	0	0	08	00
$D$ fix-up 3	0	0	0	0	09	00
<b>TARGET</b>	1	1	1	1	10	10

# Vec Subset Sum to Subset Sum

numbers
010000000001
010000000000
000100000001
000100000100
000001000100
000001000001
000000010000
000000010101
000000000007
000000000008
000000000009
000000000700
000000000800
000000000900

# 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  $n$  integers  $a_1, a_2, \dots, a_n$  and a target  $B$ , is there a subset  $S$  of  $\{a_1, \dots, a_n\}$  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 Problem:** Given  $n$  integers  $a_1, a_2, \dots, a_n$  and a target  $B$ , is there a subset  $S$  of  $\{a_1, \dots, a_n\}$  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 Problem:** Given  $n$  integers  $a_1, a_2, \dots, a_n$  and a target  $B$ , is there a subset  $S$  of  $\{a_1, \dots, a_n\}$  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 Problem:** Given  $n$  integers  $a_1, a_2, \dots, a_n$  and a target  $B$ , is there a subset  $S$  of  $\{a_1, \dots, a_n\}$  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  $a_1, a_2, \dots, a_n$  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  $a_1, a_2, \dots, a_n$  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  $a_1, a_2, \dots, a_n$  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**.









