More NP-Complete Problems

Lecture 23
April 17, 2018

Most slides are courtesy Prof. Chekuri
Recap

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

A problem $X$ is **NP-Complete** iff
- $X$ is in **NP**
- $X$ is **NP-Hard**.

$X$ is **NP-Hard** if for every $Y$ in **NP**, $Y \leq_P X$.

**Theorem (Cook-Levin)**

**SAT** is **NP-Complete**.
Recap contd

Theorem (Cook-Levin)

SAT is NP-Complete.

Establish NP-Completeness via reductions:

1. SAT is NP-Complete.
2. SAT \(\leq_P\) 3-SAT and hence 3-SAT is NP-Complete.
3. 3-SAT \(\leq_P\) Independent Set (which is in NP) and hence Independent Set is NP-Complete.
4. Clique is NP-Complete
5. Vertex Cover is NP-Complete
6. Set Cover is NP-Complete
7. Hamilton Cycle and Hamiltonian Path are NP-Complete
8. 3-Color is NP-Complete
NP vs co-NP

Prove

- Hamiltonian Cycle is NP-Complete
- 3-Coloring is NP-Complete
- Subset Sum is NP-Complete

All via reductions from 3-SAT
**NP vs co-NP**

**NP**: Problems with polynomial time verifier for a “yes” instance.
**NP vs co-NP**

**NP**: Problems with polynomial time verifier for a “yes” instance.

**SAT**: Given a CNF formula $\phi$, does there exist a satisfying assignment?
- Poly-time verification (proof) for “yes” instances.

**co-NP**: Complements of decision problems in NP.
- No-Hamiltonian-Cycle, Is-Prime, No-Subset-Sum...
**NP vs co-NP**

**NP**: Problems with polynomial time verifier for a “yes” instance.

**SAT**: Given a CNF formula $\phi$, does there exists a satisfying assignment?
- Poly-time verification (proof) for “yes” instances.

**Definition**

Given a decision problem $X$, its complement $\bar{X}$ is the same problem with “yes” and “no” answers reversed.
**NP vs co-NP**

**NP**: Problems with polynomial time verifier for a “yes” instance.

**SAT**: Given a CNF formula $\phi$, does there exists a satisfying assignment?
- Poly-time verification (proof) for “yes” instances.

**Definition**

Given a decision problem $X$, its **complement** $\overline{X}$ is the same problem with “yes” and “no” answers reversed.

**complement-SAT**: Is $\phi$ always false?
NP vs co-NP

NP: Problems with polynomial time verifier for a “yes” instance.

SAT: Given a CNF formula $\phi$, does there exists a satisfying assignment?
- Poly-time verification (proof) for “yes” instances.

Definition

Given a decision problem $X$, its complement $\bar{X}$ is the same problem with “yes” and “no” answers reversed.

complement-SAT: Is $\phi$ always false?
- Poly-time verification (proof) for “no” instances.
NP vs co-NP

**NP**: Problems with polynomial time verifier for a “yes” instance.

**SAT**: Given a CNF formula $\phi$, does there exists a satisfying assignment? 
- Poly-time verification (proof) for “yes” instances.

**Definition**

Given a decision problem $X$, its complement $\bar{X}$ is the same problem with “yes” and “no” answers reversed.

**Complement-SAT**: Is $\phi$ always false? 
- Poly-time verification (proof) for “no” instances.

**co-NP**: Complements of decision problems in NP.
- No-Hamiltonian-Cycle, Is-Prime, No-Subset-Sum.
**NP vs co-NP**

**NP**: Problems with polynomial time verifier for a “yes” instance.

**SAT**: Given a CNF formula $\phi$, does there exists a satisfying assignment?
- Poly-time verification (proof) for “yes” instances.

**Definition**

Given a decision problem $X$, its **complement** $\bar{X}$ is the same problem with “yes” and “no” answers reversed.

**complement-SAT**: Is $\phi$ always false?
- Poly-time verification (proof) for “no” instances.

**co-NP**: Complements of decision problems in NP.
  - **No-Hamiltonian-Cycle**, **Is-Prime**, **No-Subset-Sum**.
  - Poly-time verification for “no” instances
  - “no” instances can be solved in non-deterministic polynomial time.
Integer Factorization

Given integers $q$ and $n$, is there a prime factor of $q$ larger than $n$?

Input size: $\log(q) + \log(n)$
Integer Factorization

Given integers $q$ and $n$, is there a prime factor of $q$ larger than $n$?

Input size: $\log(q) + \log(n)$

Verifier for a “yes” instance?
Given integers $q$ and $n$, is there a prime factor of $q$ larger than $n$?

Input size: $\log(q) + \log(n)$

Verifier for a “yes” instance?

Verifier for a “no” instance?
Given integers $q$ and $n$, is there a prime factor of $q$ larger than $n$?

Input size: $\log(q) + \log(n)$

Verifier for a “yes” instance?

Verifier for a “no” instance?

Int-Factorization $\in$ NP $\cap$ co-NP.
Given integers \( q \) and \( n \), is there a prime factor of \( q \) larger than \( n \)?

Input size: \( \log(q) + \log(n) \)

Verifier for a “yes” instance?

Verifier for a “no” instance?

\( \text{Int-Factorization} \in \text{NP} \cap \text{co-NP} \). But not known to be in \( \text{P} \).
Landscape of Containment

EXP

NP

co-NP

P
Part I

NP-Completeness of Hamiltonian Cycle
Directed Hamiltonian Cycle

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

\textbf{Goal}  Does \( G \) have a Hamiltonian cycle?
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
Is the following graph Hamiltonian? 

(A) Yes.  
(B) No.
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 $G_\varphi$ 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 $A$

Notation: $\varphi$ has $n$ variables $x_1, x_2, \ldots, x_n$ and $m$ clauses $C_1, C_2, \ldots, C_m$. 
Viewing SAT: Assign values to $n$ variables, and each clauses 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 $3m + 3$)
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 \( C_j \).
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 $C_j$. 
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 $C_j$. 
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 $C_j$. 

$x_1 \lor \neg x_2 \lor x_4$

$\neg x_1 \lor \neg x_2 \lor \neg x_3$
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 $C_j$. 
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 $C_j$. 
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 $C_j$. 
Proposition

ϕ has a satisfying assignment iff $G_ϕ$ has a Hamiltonian cycle.

Proof.

⇒ Let $a$ be the satisfying assignment for ϕ. Define Hamiltonian cycle as follows

- If $a(x_i) = 1$ then traverse path $i$ from left to right
- If $a(x_i) = 0$ 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
Suppose $\Pi$ is a Hamiltonian cycle in $G_\phi$

- 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 $i$
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
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
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.
Is covering by cycles hard?

Given a directed graph $G$, deciding if $G$ can be covered by vertex disjoint cycles (each of length at least two) is

(A) NP-Hard.
(B) NP-Complete.
(C) P.
(D) IDK.
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
Goal: Given directed graph $G$, need to construct undirected graph $G'$ such that $G$ has Hamiltonian cycle iff $G'$ has Hamiltonian cycle.

Reduction:

Replace each vertex $v$ by 3 vertices: $v_{in}$, $v$, and $v_{out}$. A directed edge $(u, v)$ is replaced by edge $(u_{out}, v_{in})$. 

Diagram:

```
  a -- v -- c
  \  /    /  \
   b     v     d
```

Reduction Sketch
Goal: Given directed graph $G$, need to construct undirected graph $G'$ such that $G$ has Hamiltonian cycle iff $G'$ has Hamiltonian cycle

Reduction

- Replace each vertex $v$ by 3 vertices: $v_{in}$, $v$, and $v_{out}$

\[ a \rightarrow v \rightarrow c \]
\[ b \rightarrow v \rightarrow d \]
**Reduction Sketch**

**Goal:** Given directed graph $G$, need to construct undirected graph $G'$ such that $G$ has Hamiltonian cycle iff $G'$ has Hamiltonian cycle.

**Reduction**

- Replace each vertex $v$ by 3 vertices: $v_{in}$, $v$, and $v_{out}$.
- A directed edge $(u, v)$ is replaced by edge $(u_{out}, v_{in})$.

![Diagram](image)
Goal: Given directed graph $G$, need to construct undirected graph $G'$ such that $G$ has Hamiltonian cycle iff $G'$ has Hamiltonian cycle

**Reduction**
- Replace each vertex $v$ by 3 vertices: $v_{in}$, $v$, and $v_{out}$
- A directed edge $(u, v)$ is replaced by edge $(u_{out}, v_{in})$
Reduction: Wrapup

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

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

Goal  Does $G$ have a Hamiltonian path?
- A Hamiltonian path is a path in the graph that visits every vertex in $G$ exactly once
Hamiltonian Path

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

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

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

**Exercise:** Modify the reduction from 3-SAT to Hamilton cycle to prove that 3-SAT reduces to Hamilton path.
Input  Given a directed graph $G = (V, E)$ with $n$ vertices

Goal  Does $G$ have a Hamiltonian path?

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

Exercise: Modify the reduction from 3-SAT to Hamilton cycle to prove that 3-SAT reduces to Hamilton path.

Exercise: Also prove that Hamilton path in undirected graphs is NP-Complete.
Part II

NP-Completeness of Graph Coloring
Problem: **Graph Coloring**

**Instance:** $G = (V, E)$: Undirected graph, integer $k$.

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

Instance: $G = (V, E)$: Undirected graph.

Question: Can the vertices of the graph be colored using 3 colors so that vertices connected by an edge do not get the same color?
Problem: 3 Coloring

Instance: $G = (V, E)$: Undirected graph.

Question: 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.
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.
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!
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.
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.
**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?
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
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 $[a_0, b_0], [a_1, b_1], \ldots, [a_k, b_k]$
- Each cell phone tower (simplifying) gets one band
- Constraint: nearby towers cannot be assigned same band, otherwise signals will interference
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 interfere 

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 color this gadget.

You are given three colors: red, green and blue. Can the following graph be three colored in a valid way (assuming the four nodes are already colored as indicated).

(A) Yes.
(B) No.
You are given three colors: red, green and blue. Can the following graph be three colored in a valid way (assuming the four nodes are already colored as indicated).

(A) Yes.
(B) No.
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 \( x_1, \ldots, x_n \) 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 \( G_\varphi \).
Reduction Idea

Start with \textbf{3SAT} formula (i.e., \textbf{3CNF} 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.
Reduction Idea

Start with \textbf{3SAT} formula (i.e., \textbf{3CNF} 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 \( \overline{v}_i \) connected in a triangle with common Base
Reduction Idea

Start with **3SAT** formula (i.e., **3CNF** 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 $\overline{v_i}$ connected in a triangle with common Base
- If graph is 3-colored, either $v_i$ or $\overline{v_i}$ gets the same color as True. Interpret this as a truth assignment to $v_i$
Reduction Idea

Start with **3SAT** formula (i.e., **3CNF** 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 $\overline{v}_i$ connected in a triangle with common Base
- If graph is 3-colored, either $v_i$ or $\overline{v}_i$ 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

- **T**
- **F**
- **Base**
- **\(v_1\)**
- **\(\overline{v}_1\)**
- **\(v_2\)**
- **\(\overline{v}_2\)**
- **\(v_n\)**
- **\(\overline{v}_n\)**
Clause Satisfiability Gadget

For each clause $C_j = (a \lor b \lor c)$, 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 $a, b, c$ 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.
3 coloring of the clause gadget
3 coloring of the clause gadget
3 coloring of the clause gadget

<table>
<thead>
<tr>
<th>FFF - BAD</th>
<th>FFT</th>
<th>FTF</th>
</tr>
</thead>
</table>

- FFF - BAD
- FFT
- FTF
3 coloring of the clause gadget

FFF - BAD

FFT

FTF

FTT

TFF

TFT

TTF

TTT
Reduction

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

\[ \varphi = (u \lor \neg v \lor w) \land (v \lor x \lor \neg y) \]
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
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 $C_j = (a \lor b \lor c)$ at least one of $a$, $b$, $c$ is colored True. OR-gadget for $C_j$ can be 3-colored such that output is True.
Correctness of Reduction

\( \varphi \) is satisfiable implies \( G_\varphi \) is 3-colorable

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

\( \varphi \) is satisfiable implies \( G_\varphi \) is 3-colorable

- if \( x_i \) is assigned True, color \( v_i \) True and \( \overline{v_i} \) False
- for each clause \( C_j = (a \lor b \lor c) \) 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 \( C_j = (a \lor b \lor c) \). 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
Part III

Hardness of 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.
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 $\overrightarrow{t}$.

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

Reduction from **3SAT**.
Think about vectors as being lines in a table.

How to “select” exactly one of $x = 0$ and $x = 1$.

First gadget

Selecting between two lines.

<table>
<thead>
<tr>
<th>Target</th>
<th>??</th>
<th>??</th>
<th>01</th>
<th>??</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>??</td>
<td>??</td>
<td>01</td>
<td>??</td>
</tr>
<tr>
<td>$a_2$</td>
<td>??</td>
<td>??</td>
<td>01</td>
<td>??</td>
</tr>
</tbody>
</table>
Think about vectors as being lines in a table.

**How to “select” exactly one of \( x = 0 \) and \( x = 1 \).**

### First gadget

Selecting between two lines.

<table>
<thead>
<tr>
<th>Target</th>
<th>??</th>
<th>??</th>
<th>01</th>
<th>???</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>??</td>
<td>??</td>
<td>01</td>
<td>??</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>??</td>
<td>??</td>
<td>01</td>
<td>??</td>
</tr>
</tbody>
</table>

Two rows for every variable \( x \): selecting either \( x = 0 \) or \( x = 1 \).
Handling a clause...

We will have a column for every clause...

<table>
<thead>
<tr>
<th>numbers</th>
<th>...</th>
<th>$C \equiv a \lor b \lor \overline{c}$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>...</td>
<td>01</td>
<td>...</td>
</tr>
<tr>
<td>$\overline{a}$</td>
<td>...</td>
<td>00</td>
<td>...</td>
</tr>
<tr>
<td>$b$</td>
<td>...</td>
<td>01</td>
<td>...</td>
</tr>
<tr>
<td>$\overline{b}$</td>
<td>...</td>
<td>00</td>
<td>...</td>
</tr>
<tr>
<td>$c$</td>
<td>...</td>
<td>00</td>
<td>...</td>
</tr>
<tr>
<td>$\overline{c}$</td>
<td>...</td>
<td>01</td>
<td>...</td>
</tr>
<tr>
<td>$C$ fix-up 1</td>
<td>000</td>
<td>07</td>
<td>000</td>
</tr>
<tr>
<td>$C$ fix-up 2</td>
<td>000</td>
<td>08</td>
<td>000</td>
</tr>
<tr>
<td>$C$ fix-up 3</td>
<td>000</td>
<td>09</td>
<td>000</td>
</tr>
<tr>
<td>TARGET</td>
<td></td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>
### 3SAT to Vec Subset Sum

<table>
<thead>
<tr>
<th>numbers</th>
<th>$a \lor \overline{a}$</th>
<th>$b \lor \overline{b}$</th>
<th>$c \lor \overline{c}$</th>
<th>$d \lor \overline{d}$</th>
<th>$D \equiv \overline{b} \lor c \lor \overline{d}$</th>
<th>$C \equiv a \lor b \lor \overline{c}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>$\overline{a}$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$b$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>$\overline{b}$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>$c$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>$\overline{c}$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>$d$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$\overline{d}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>$C$ fix-up 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>07</td>
</tr>
<tr>
<td>$C$ fix-up 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>08</td>
</tr>
<tr>
<td>$C$ fix-up 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>09</td>
</tr>
<tr>
<td>$D$ fix-up 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>07</td>
<td>00</td>
</tr>
<tr>
<td>$D$ fix-up 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>08</td>
<td>00</td>
</tr>
<tr>
<td>$D$ fix-up 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>09</td>
<td>00</td>
</tr>
<tr>
<td>TARGET</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>
# Vec Subset Sum to Subset Sum

<table>
<thead>
<tr>
<th>Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>010000000001</td>
</tr>
<tr>
<td>010000000000</td>
</tr>
<tr>
<td>000100000001</td>
</tr>
<tr>
<td>000100000100</td>
</tr>
<tr>
<td>000001000100</td>
</tr>
<tr>
<td>000001000001</td>
</tr>
<tr>
<td>000000100000</td>
</tr>
<tr>
<td>000000101010</td>
</tr>
<tr>
<td>000000000007</td>
</tr>
<tr>
<td>000000000008</td>
</tr>
<tr>
<td>000000000009</td>
</tr>
<tr>
<td>000000000070</td>
</tr>
<tr>
<td>000000000080</td>
</tr>
<tr>
<td>000000000090</td>
</tr>
<tr>
<td>010101011010</td>
</tr>
</tbody>
</table>
Other **NP-Complete** Problems

- 3-Dimensional Matching
- 3-Partition

Read book.
Subset Sum and Knapsack

**Knapsack**: Given $n$ items with item $i$ having non-negative integer size $s_i$ and non-negative integer 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 $P$?

**Exercise**: Show **Knapsack** is **NP-Complete** via reduction from **Subset Sum**
**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**.

Number problems which are NP-Complete even when the numbers are written in unary are **strongly 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**.

*Number problems* which are **NP-Complete** even when the numbers are written in unary are **strongly NP-Complete**.
A Strongly NP-Complete Number Problem

**3-Partition**: Given $3n$ numbers $a_1, a_2, \ldots, a_{3n}$ and target $B$ can the numbers be partitioned into $n$ groups of $3$ each such that the sum of numbers in each group is exactly $B$?

Can further assume that each number $a_i$ is between $B/3$ and $2B/3$.

Can reduce **3-D-Matching** to **3-Partition** in polynomial time such that each number $a_i$ can be written in unary.
Need to Know **NP-Complete** Problems

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