# NP Hardness Reductions II

# Admin

- Problem 2 on Assignment 7 is extra credit
- Assignment 7 due tomorrow
- I just found out you can't move the boxes we make while we're grading—-we'll keep that in mind
- Thanksgiving is soon!
- In the meantime, let's enjoy some more NP hardness proofs.  A good mix of difficulty today…

# SET-COVER

# Set Cover

- **Set-Cover.** Given a set $U$ of elements, a collection $\mathcal{S}$ of subsets of $U$ and an integer $k$, are there at most $k$ subsets $S_1, \ldots, S_k$ whose union covers $U$, that is, $U \subseteq \cup_{i=1}^{k} S_i$

$U = \{ 1, 2, 3, 4, 5, 6, 7 \}$

$S_a = \{ 3, 7 \}$ $S_b = \{ 2, 4 \}$

$S_c = \{ 3, 4, 5, 6 \}$ $S_d = \{ 5 \}$

$S_e = \{ 1 \}$ $S_f = \{ 1, 2, 6, 7 \}$

$k = 2$

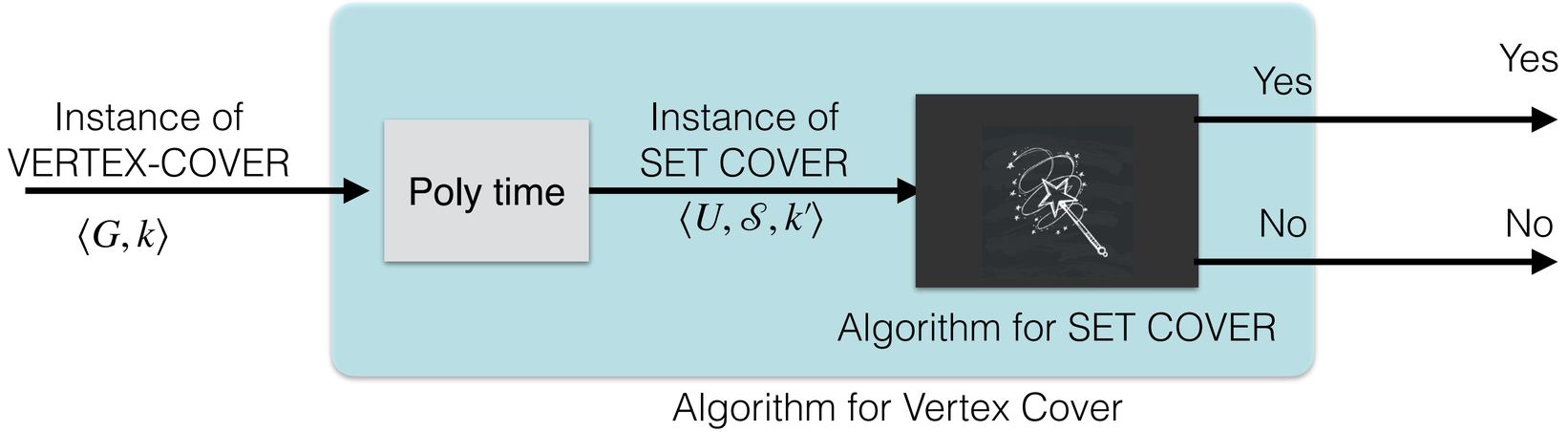
**a set cover instance**

# List of NPC Problems So Far

- SAT/ 3-SAT
- INDEPENDENT SET
- VERTEX COVER
- CLIQUE

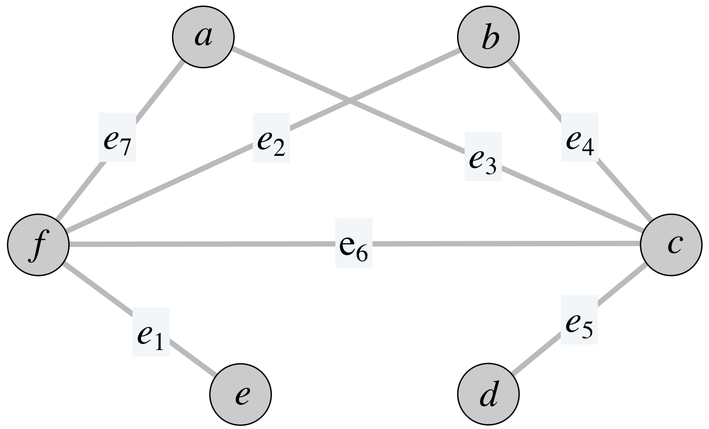
# Vertex Cover $\leq_p$ Set Cover

- **Theorem.** VERTEX-COVER $\leq_p$ SET-COVER
- **Proof.** Given instance $\langle G, k \rangle$ of vertex cover, construct an instance $\langle U, \mathcal{S}, k' \rangle$ of set cover problem such that $G$ has a vertex cover of size at most $k$ if and only if $\langle U, \mathcal{S}, k' \rangle$ has a set cover of size at most $k$.



# Vertex Cover $\leq_p$ Set Cover

- **Theorem.** VERTEX-COVER $\leq_p$ SET-COVER
- **Proof.** Given instance $\langle G, k \rangle$ of vertex cover, construct an instance $\langle U, \mathcal{S}, k \rangle$ of set cover problem that has a set cover of size $k$ iff $G$ has a vertex cover of size $k$.
- **Reduction.** $U = E$, for each node $v \in V$, let
  $S_v = \{ e \in E \mid e \text{ incident to } v \}$




vertex cover instance
(k = 2)

$U = \{ e_1, e_2, \ldots, e_7 \}$

$S_a = \{ e_3, e_7 \}$ $S_b = \{ e_2, e_4 \}$

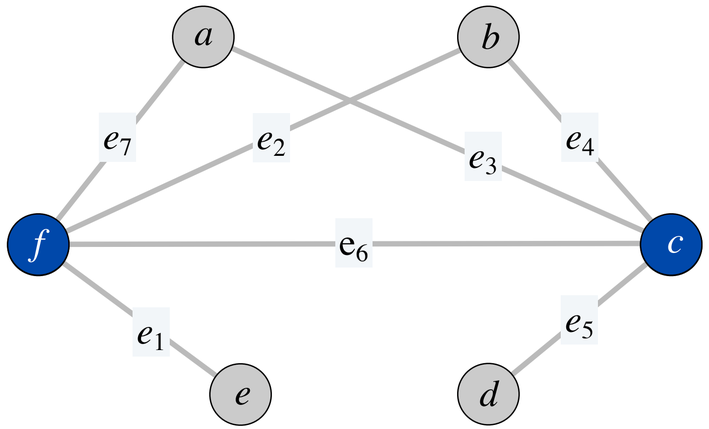
$S_c = \{ e_3, e_4, e_5, e_6 \}$ $S_d = \{ e_5 \}$

$S_e = \{ e_1 \}$ $S_f = \{ e_1, e_2, e_6, e_7 \}$

set cover instance
(k = 2)

# Correctness

- **Claim.** ( $\Rightarrow$ ) If $G$ has a vertex cover of size at most $k$, then $U$ can be covered using at most $k$ subsets.
- **Proof.** Let $X \subseteq V$ be a vertex cover in $G$
  - Then, $Y = \{S_v \mid v \in X\}$ is a set cover of $U$ of the same size




vertex cover instance
(k = 2)

$U = \{ e_1, e_2, \ldots, e_7 \}$

$S_a = \{ e_3, e_7 \}$ $S_b = \{ e_2, e_4 \}$

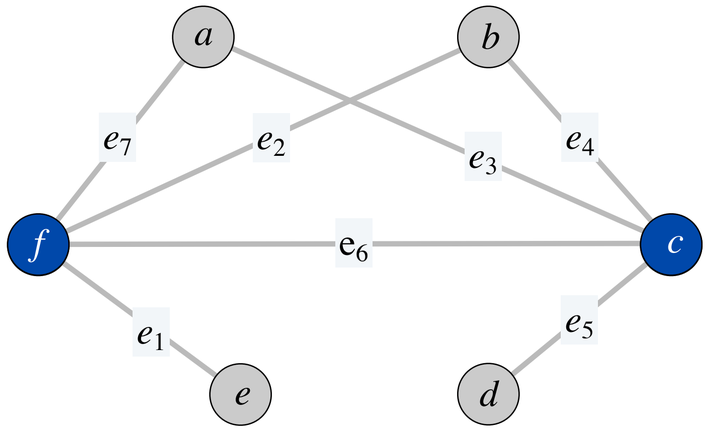
$S_c = \{ e_3, e_4, e_5, e_6 \}$ $S_d = \{ e_5 \}$

$S_e = \{ e_1 \}$ $S_f = \{ e_1, e_2, e_6, e_7 \}$

set cover instance
(k = 2)

# Correctness

- **Claim.** ( $\Leftarrow$ ) If $U$ can be covered using at most $k$ subsets then $G$ has a vertex cover of size at most $k$.
- **Proof.** Let $Y \subseteq \mathcal{S}$ be a set cover of size $k$
  - Then, $X = \{v \mid S_v \in Y\}$ is a vertex cover of size $k$




vertex cover instance
(k = 2)

$U = \{ e_1, e_2, \ldots, e_7 \}$

$S_a = \{ e_3, e_7 \}$

$S_b = \{ e_2, e_4 \}$

$S_c = \{ e_3, e_4, e_5, e_6 \}$

$S_d = \{ e_5 \}$

$S_e = \{ e_1 \}$

$S_f = \{ e_1, e_2, e_6, e_7 \}$

set cover instance
(k = 2)

# SUBSET-SUM is NP Complete:

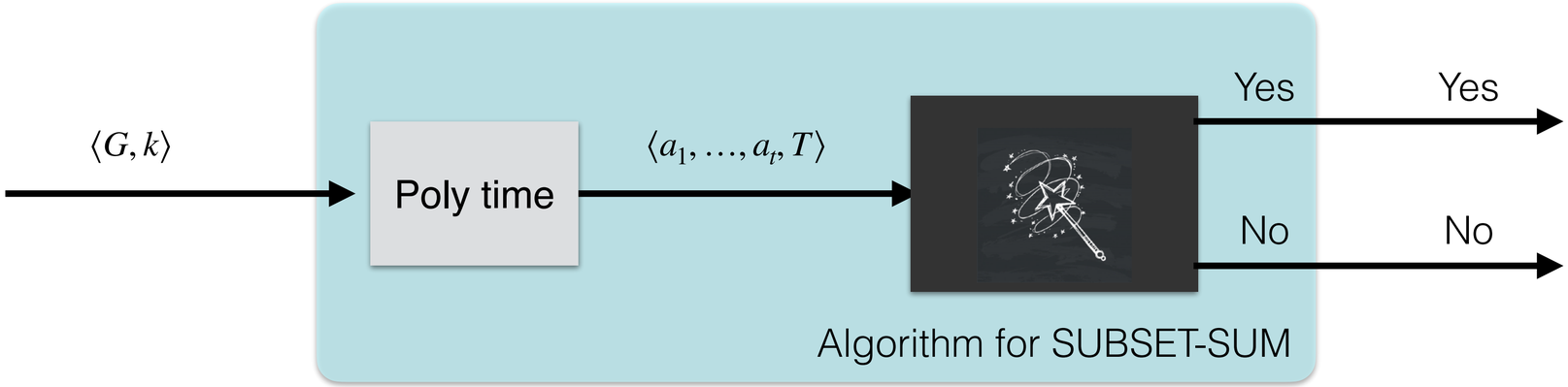
Vertex-Cover $\leq_p$ SUBSET-SUM

# Subset Sum Problem

- **SUBSET-SUM.**
  Given $n$ positive integers $a_1, \ldots, a_n$ and a target integer $T$, is there a subset of numbers that adds up to exactly $T$
- **SUBSET-SUM** $\in$ NP
  - Certificate: a subset of numbers
  - Poly-time verifier: checks if subset is from the given set and sums exactly to $T$
- Problem has a pseudo-polynomial $O(nT)$-time dynamic programming algorithm similar to Knapsack
- Will prove **SUBSET-SUM** is **NP hard:** reduction from vertex cover
- NP hard problems that have pseudo-polynomial algorithms are called *weakly NP hard*

# Vertex Cover to Subset Sum

- **Theorem.** VERTEX-COVER $\leq_p$ SUBSET-SUM
- Proof. Given a graph $G$ with $n$ vertices and $m$ edges and a number $k$, we construct a set of numbers $a_1, \ldots, a_t$ and a target sum $T$ such that $G$ has a vertex cover of size $k$ iff there is a subset of numbers that sum to $T$

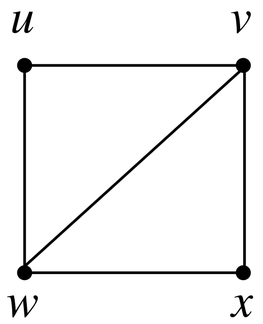


# Vertex Cover to Subset Sum

- **Theorem.** VERTEX-COVER $\leq_p$ SUBSET-SUM
- **Proof.** Label the edges of $G$ as $0, 1, \ldots, m-1$.
- **Reduction**. Create $n + m$ integers and a target value $T$ as follows
- Each integer is a $m + 1$-bit number in base four
- Integers representing vertices and edges:
  - Vertex integer $a_v$ : $m$th (most significant) bit is $1$ and for $i < m$, the $i$th bit is 1 if $i$th edge is incident to vertex $v$
  - Edge integer $b_{uv}$ : $m$th digit is $0$ and for $i < m$, the $i$th bit is 1 if this integer represents an edge $i = (u, v)$
  - Target value $T = k \cdot 4^m + \sum_{i=0}^{m-1} 2 \cdot 4^i$

# Vertex Cover to Subset Sum

- Example: consider the graph $G = (V, E)$ where $V = \{u, v, w, x\}$ and $E = \{(u, v),\ (u, w),\ (v, w),\ (v, x),\ (w, x)\}$




| | 5th | 4th : (uv) | 3rd : (uw) | 2nd : (vw) | 1st : (vx) | 0th: (wx) |
|---|---|---|---|---|---|---|
| $a_u$ | 1 | 1 | 1 | 0 | 0 | 0 |
| $a_v$ | 1 | 1 | 0 | 1 | 1 | 0 |
| $a_w$ | 1 | 0 | 1 | 1 | 0 | 1 |
| $a_x$ | 1 | 0 | 0 | 0 | 1 | 1 |
| $b_{uv}$ | 0 | 1 | 0 | 0 | 0 | 0 |
| $b_{uw}$ | 0 | 0 | 1 | 0 | 0 | 0 |
| $b_{vw}$ | 0 | 0 | 0 | 1 | 0 | 0 |
| $b_{vx}$ | 0 | 0 | 0 | 0 | 1 | 0 |
| $b_{wx}$ | 0 | 0 | 0 | 0 | 0 | 1 |

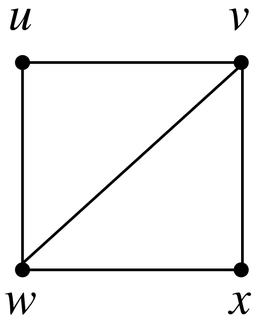
$$a_u := 111000_4 = 1344$$
$$a_v := 110110_4 = 1300$$
$$a_w := 101101_4 = 1105$$
$$a_x := 100011_4 = 1029$$
$$b_{uv} := 010000_4 = 256$$
$$b_{uw} := 001000_4 = 64$$
$$b_{vw} := 000100_4 = 16$$
$$b_{vx} := 000010_4 = 4$$
$$b_{wx} := 000001_4 = 1$$

- If $k = 2$ then $T = 222222_4 = 2730$

# Correctness

- **Claim.** $G$ has a vertex cover of size $k$ if and only there is a subset $X$ of corresponding integers that sums to value $T$
- ( $\Rightarrow$ ) Let $C$ be a vertex cover of size $k$ in $G$, define $X$ as

  $X := \{a_v \mid v \in C\} \cup \{b_i \mid \text{edge } i \text{ has exactly one endpoint in } C\}$
- Sum of the most significant bits of $X$ is $k$ and all other bits sum to $2$
- Thus the elements of $X$ sum to exactly $T$




| | $5^{th}$ | $4^{th}$ : (uv) | $3^{rd}$ : (uw) | $2^{nd}$ : (vw) | $1^{st}$ : (vx) | $0^{th}$: (wx) |
|---|---|---|---|---|---|---|
| $a_v$ | 1 | 1 | 0 | 1 | 1 | 0 |
| $a_w$ | 1 | 0 | 1 | 1 | 0 | 1 |
| $b_{uv}$ | 0 | 1 | 0 | 0 | 0 | 0 |
| $b_{uw}$ | 0 | 0 | 1 | 0 | 0 | 0 |
| $b_{vx}$ | 0 | 0 | 0 | 0 | 1 | 0 |
| $b_{wx}$ | 0 | 0 | 0 | 0 | 0 | 1 |

$$C = \{v, w\}$$

$$T = k \cdot 4^m + \sum_{i=0}^{m-1} 2 \cdot 4^i$$

$$T = 222222_4 = 2730$$

# Vertex Cover to Subset Sum

- **Claim.** $G$ has a vertex cover of size $k$ if and only there is a subset $X$ of corresponding integers that sums to value $T$
- $(\Leftarrow)$ Let $X$ be the subset of numbers that sum to $T$
- That is, there is $V' \subseteq V, E' \subseteq E$ s.t.

$$X := \sum_{v \in V'} a_v + \sum_{i \in E'} b_i = T = k \cdot 4^m + \sum_{i=0}^{m-1} 2 \cdot 4^i$$

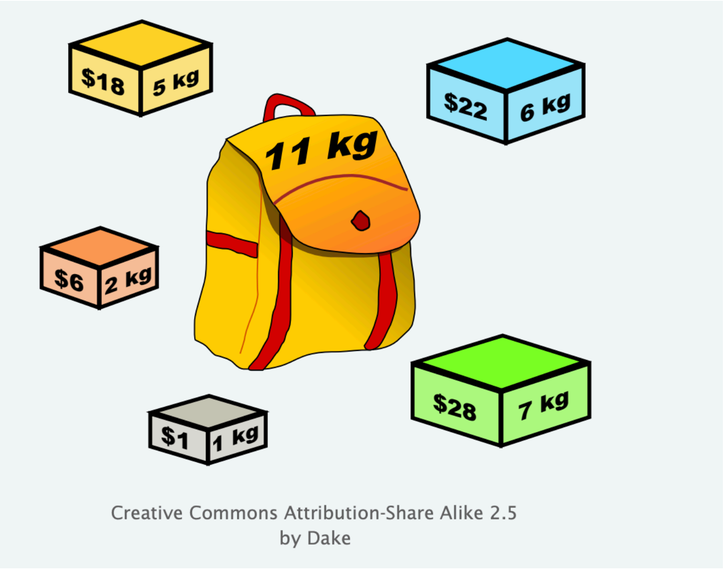
- These numbers are base 4 and there are no carries
- Each $b_i$ only contributes $1$ to the $i$th digit, which is 2
- Thus, for each edge $i$, at least one of its endpoints must be in $V'$
  - $V'$ is a vertex cover
- Size of $V'$ is $k$: only vertex-numbers have a $1$ in the $m$th position

# Class Exercise:

SUBSET-SUM $\leq_p$ Knapsack

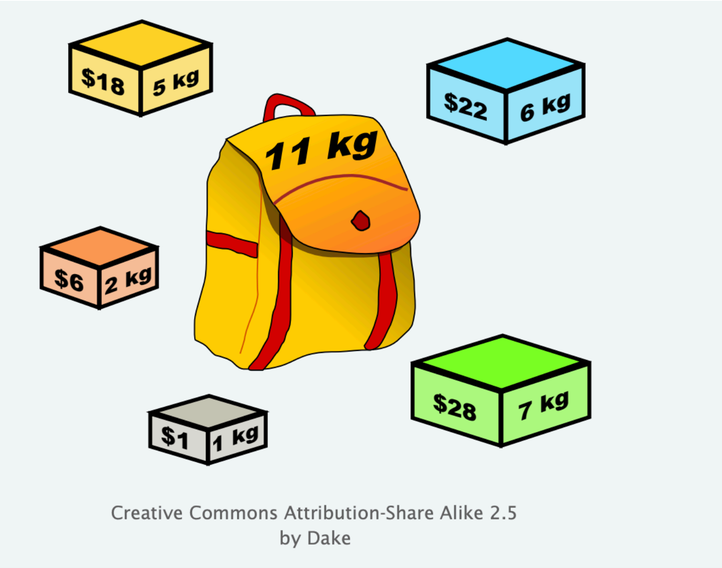
# Subset Sum to Knapsack

- **Knapsack**. Given $n$ elements $a_1, \ldots, a_n$ where each element has a weight $w_i \geq 0$ and a value $v_i \geq 0$ and target weight $W$ and value $K$. Does there exist a subset $X$ of numbers such that
  - $\sum_{a_i \in X} w_i \leq W$
  - $\sum_{a_i \in X} v_i \geq K$
- **Knapsack** $\in$ NP
  - Can check if given subset satisfies the above conditions
- Show **Subset-Sum** $\leq_p$ **Knapsack**.



# Subset Sum to Knapsack

- **Knapsack.** Given $n$ elements $a_1, \ldots, a_n$ where each element has a weight $w_i \geq 0$ and a value $v_i \geq 0$ and target weight $W$ and value $K$. Does there exist a subset $X$ of numbers such that
  - $\sum_{a_i \in X} w_i \leq W$
  - $\sum_{a_i \in X} v_i \geq K$
- **Knapsack** $\in$ NP
  - Can check if given subset satisfies the above conditions
- **Subset-Sum** $\leq_p$ **Knapsack Proof idea:**
  - $K = W = T$ and $w_i = v_i = a_i$ for all $i$

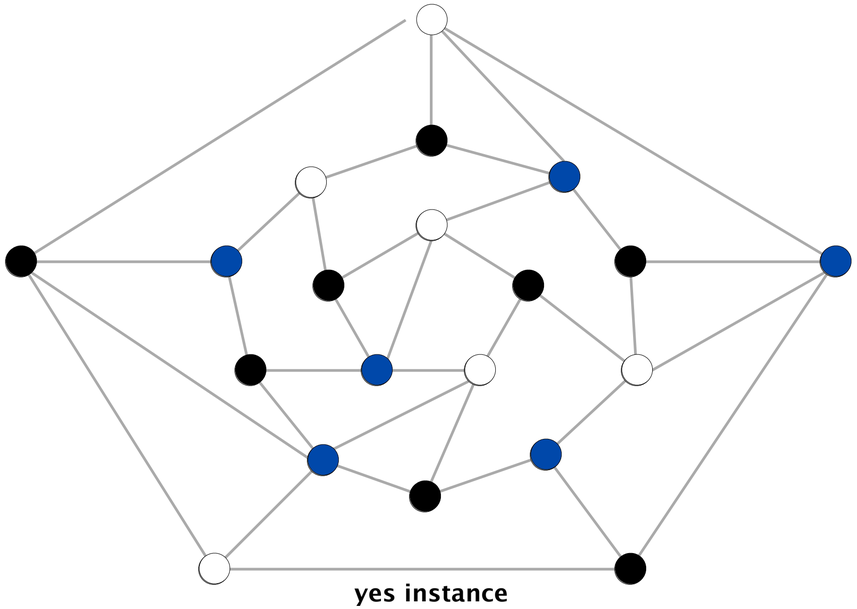


# Graph-3-Color is NP Complete:

3-SAT $\leq_p$ Graph 3-Color

# Graph 3-Color Problem

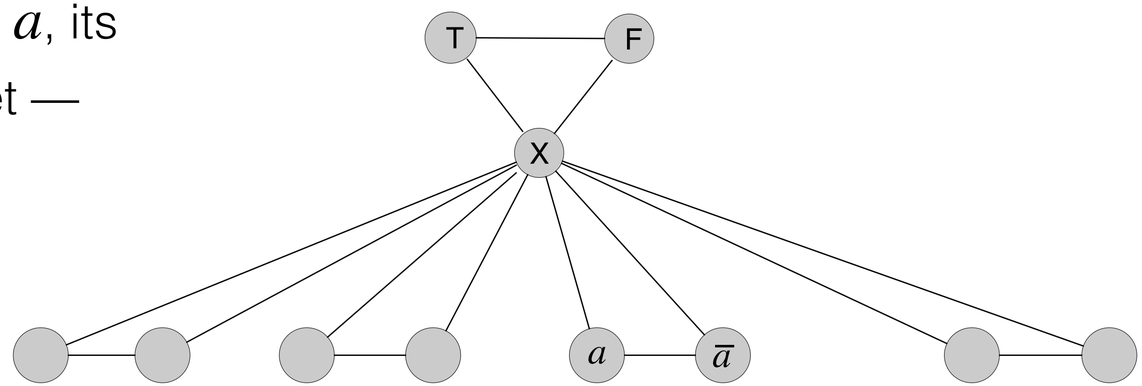
- **3-COLOR**. Given an undirected graph $G = (V, E)$, is it possible to color the vertices with 3 colors s.t. no adjacent nodes have the same color.
- We argued previously that **3-COLOR** $\in$ NP.



yes instance

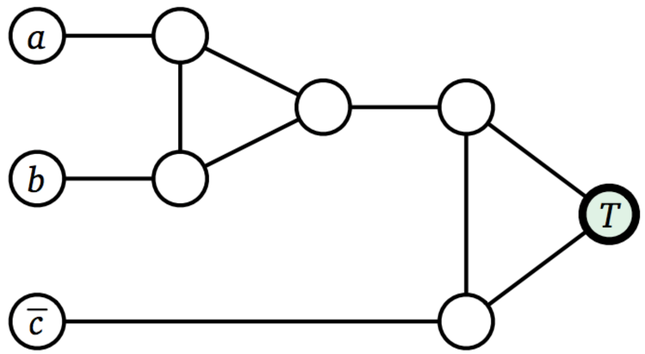
# 3-SAT to 3-Color Problem

- **Theorem.** **3-SAT** $\leq_p$ **3-COLOR**
- **Proof.** Given a 3-SAT instance $\Phi$, define $G$ as follows
  - Truth gadget: a triangle with three nodes $T, F,$ and $X$ (for true, false and other) — they must get different colors (say *true*, *false*, *other*)
  - Variable gadget: triangle made up of variable $a$, its negation $\overline{a}$ and the $X$ node of the truth gadget — enforces $a, \overline{a}$ are colored true/false



# 3-SAT to 3-Color Problem

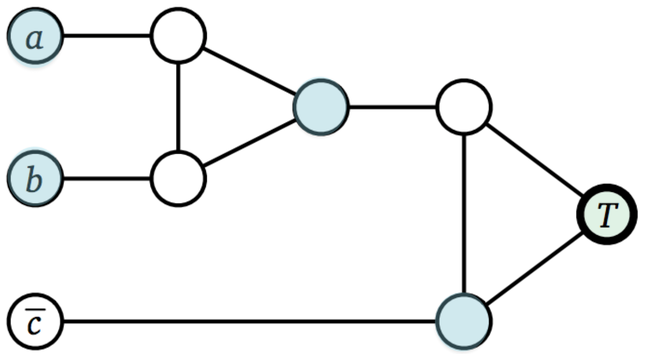
- **Theorem.** **3-SAT $\leq_p$ 3-COLOR**
- **Proof.** Given a 3-SAT instance $\Phi$, define $G$ as follows
  - Truth gadget: a triangle with three nodes $T, F,$ and $X$ (for true, false and other) — they must get different colors (say *true, false, other*)
  - Variable gadget: triangle made up of variable $a$, its negation $\overline{a}$ and the $X$ node of the truth gadget — enforces $a, \overline{a}$ are colored true/false
  - Clause gadget: joins three literal nodes (from the variable gadget) to node $T$ in the truth gadget using a subgraph as shown below




$(a \vee b \vee \overline{c})$

# 3-SAT to 3-Color Problem

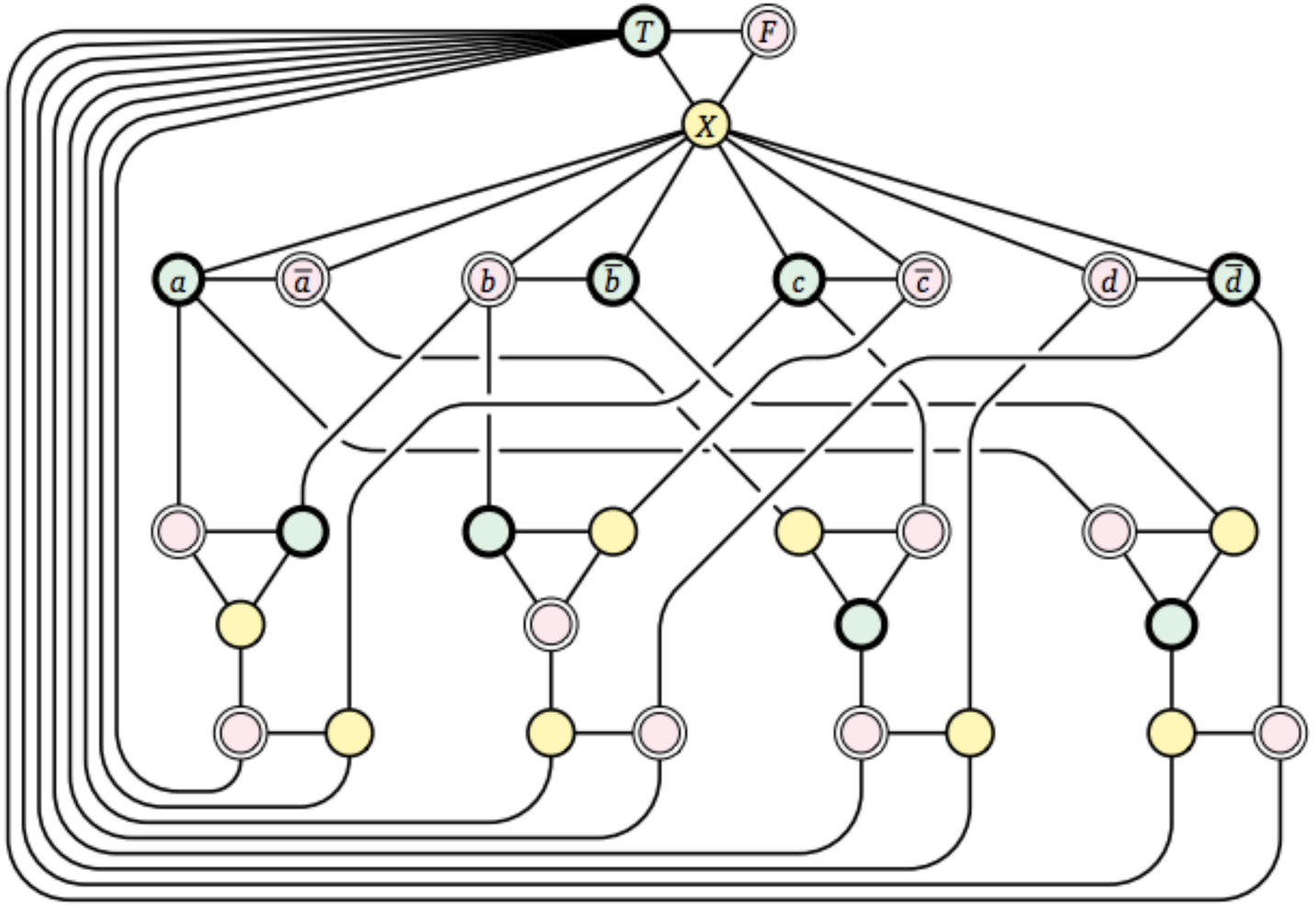
- **Observation**.
  - Clause gadget enforces that in a valid 3-coloring, not all three literals can be colored FALSE
  - If $a, b$ (or $b, \overline{c}$) or ($a, \overline{c}$) get the same color (say, FALSE) then the right-end-point of the triangle must be colored the same (shown in blue)
    - The remaining literal cannot be colored false!




$(a \vee b \vee \overline{c})$

# 3-SAT to 3-Color Problem

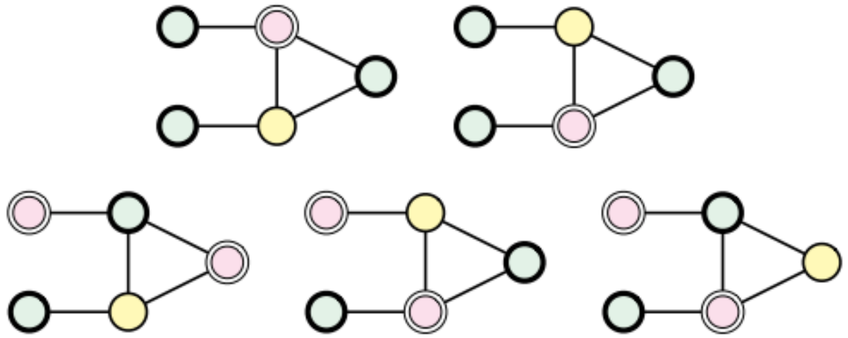
- **Theorem.** 3-SAT $\leq_p$ 3-COLOR
- Overall $G$ example
- (Yes, this is a complicated graph. Complicated graphs are going to be the hard graphs for problems like 3-color!)




$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$

# 3-SAT to 3-Color Problem

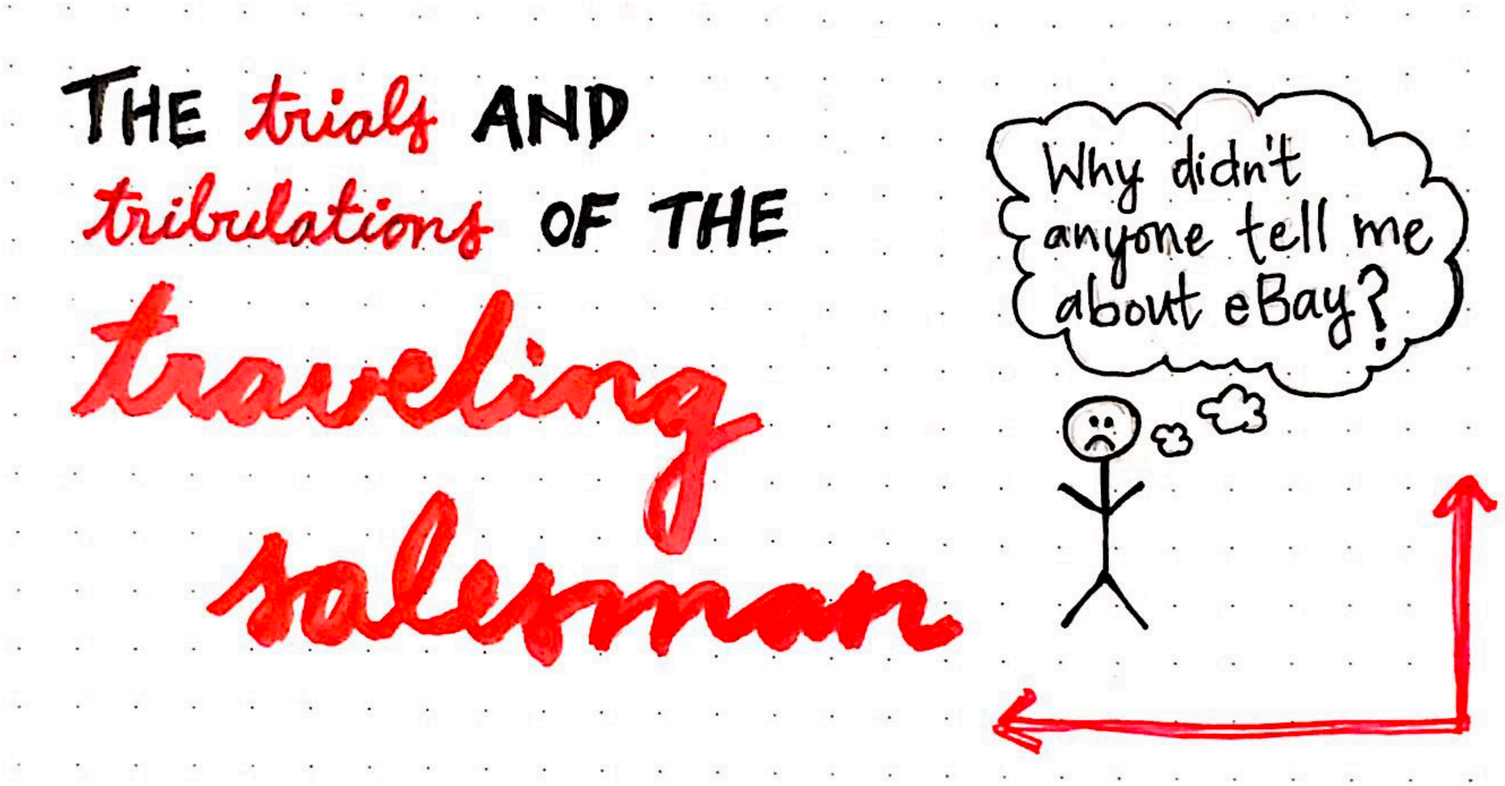
- **Theorem.** 3-SAT $\leq_p$ 3-COLOR
- **Proof Sketch**.
  - ( $\Rightarrow$ ) If $\Phi$ is satisfiable, color the variables based on the satisfying assignment (and because each clause is satisfied) extend the coloring to the clause gadgets
  - ( $\Leftarrow$ ) If $G$ is 3-colorable, then we can assign truth values based on the colors (at least one of the literals in each clause must be colored true) and thus the resulting assignment must satisfy $\Phi$
- Note this problem extends to $k$-coloring of graphs for $k \geq 3$ and the generalized problem is also hard.



# List of NPC Problems So Far

- SAT/ 3-SAT
- INDEPENDENT SET
- VERTEX COVER
- SET COVER
- CLIQUE
- 3-COLOR
- Subset-Sum
- Knapsack
- Next:
  - Traveling salesman problem
  - Hamiltonian cycle / path

# Traveling Salesman Problem



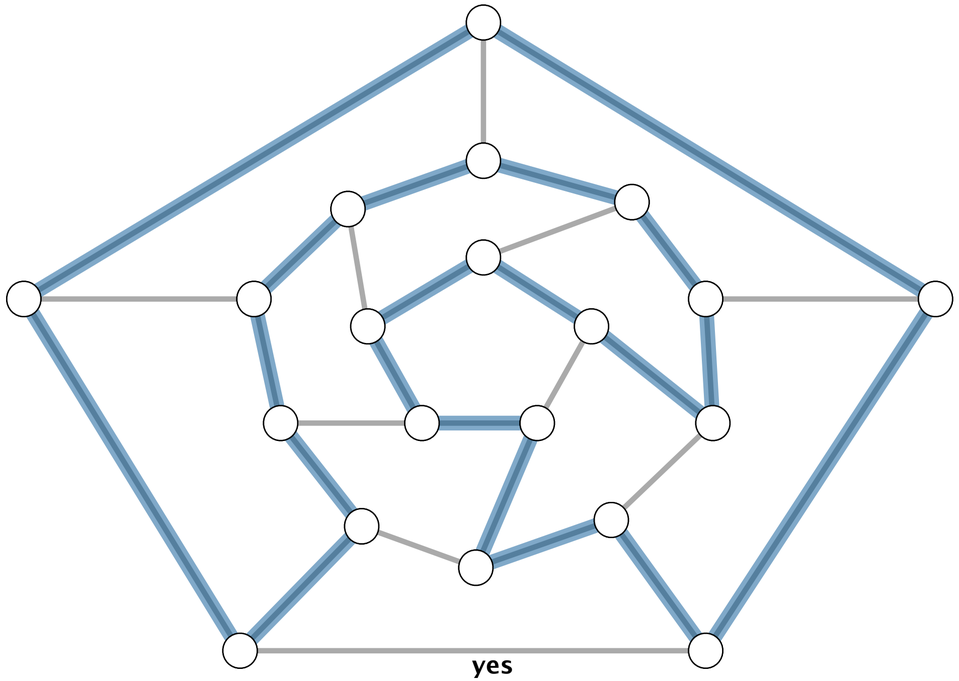

Vaidehi Joshi
https://medium.com/basecs/the-trials-and-tribulations-of-the-traveling-salesman-56048d6709d

# Traveling Salesman Problem

- Extremely famous NP complete problem
- Consider a salesman who visits $n$ cities labeled $v_1, \ldots, v_n$
- Salesman starts at $v_1$ and wants to find a *tour,* an order in which to visit all the other cities and return home
- **Goal**. Travel as little distance as possible
- Formally, let $d(i, j)$ be the distance from city $v_i$ to city $v_j$ (not necessarily symmetric or triangle inequality (e.g. airplane prices))
- **TSP.** Decision version: given a set of distances on $n$ cities and a bound $D$, is there a tour (of all the cities) of length at most $D$?
- Many applications: VLSI design, robotics, cache-efficiency
- Will prove **TSP** is NP hard using a similar problem: **HAMILTONIAN CYCLE/PATH**

# (Directed) Hamiltonian Cycle

- **HAMILTONIAN-CYCLE.** Given a directed graph $G = (V, E)$ does there exists a cycle $T$ that visits every vertex exactly once?
- We want to prove **HAMILTONIAN-CYCLE** is NP complete
  - **HAMILTONIAN-CYCLE** $\in$ NP
    - Certificate: sequence of vertices in the graph
    - Poly-time verifier
      - Check if sequence is a valid path in $G$
      - Check if path visits every vertex exactly once
  - **HAMILTONIAN-CYCLE is NP hard**
    - Sufficiently different from other NP hard graph problems
    - We reduce 3SAT to it

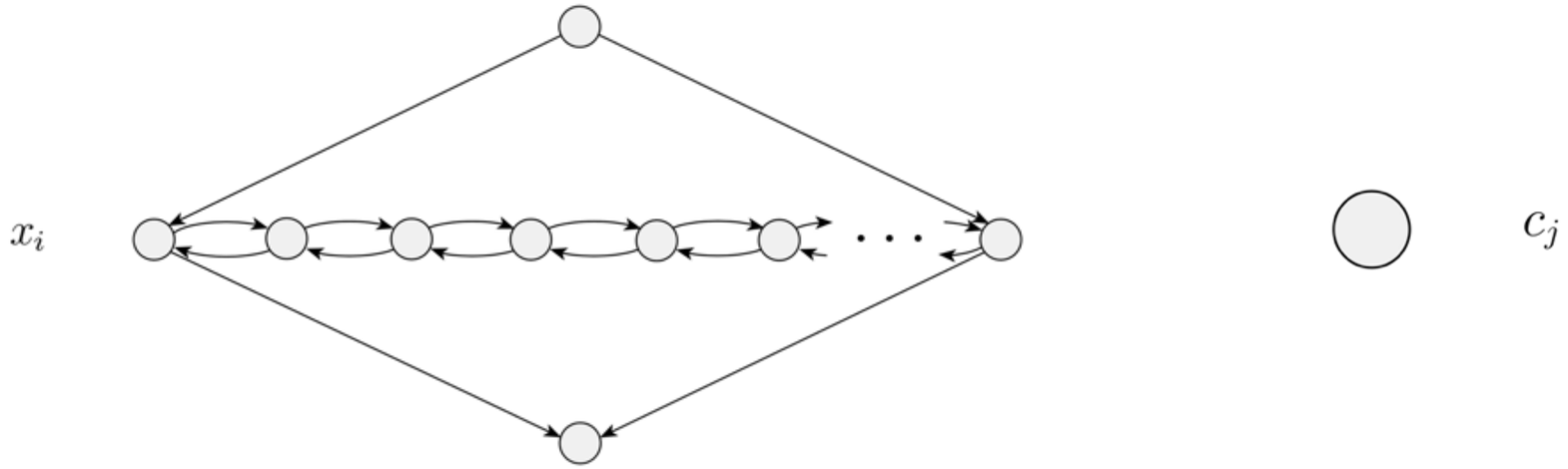


# 3SAT $\leq_p$ Hamiltonian Cycle

- Given 3SAT instance $\Phi$, transform it to directed graph $G$ s.t. $\Phi$ is satisfiable iff $G$ has a hamiltonian cycle
- Essential ingredients of a input assignments of $\Phi$
  - Each variable can be set to true or false (need to encode these settings in the graph in our variable gadget)
  - For a clause to be satisfied at least one literal is set to true
- High-level reduction idea
  - Variable gadgets that encode true/false assignment
  - Clause gadget that is set to true iff hamiltonian cycle exists
  - Tie them together appropriately

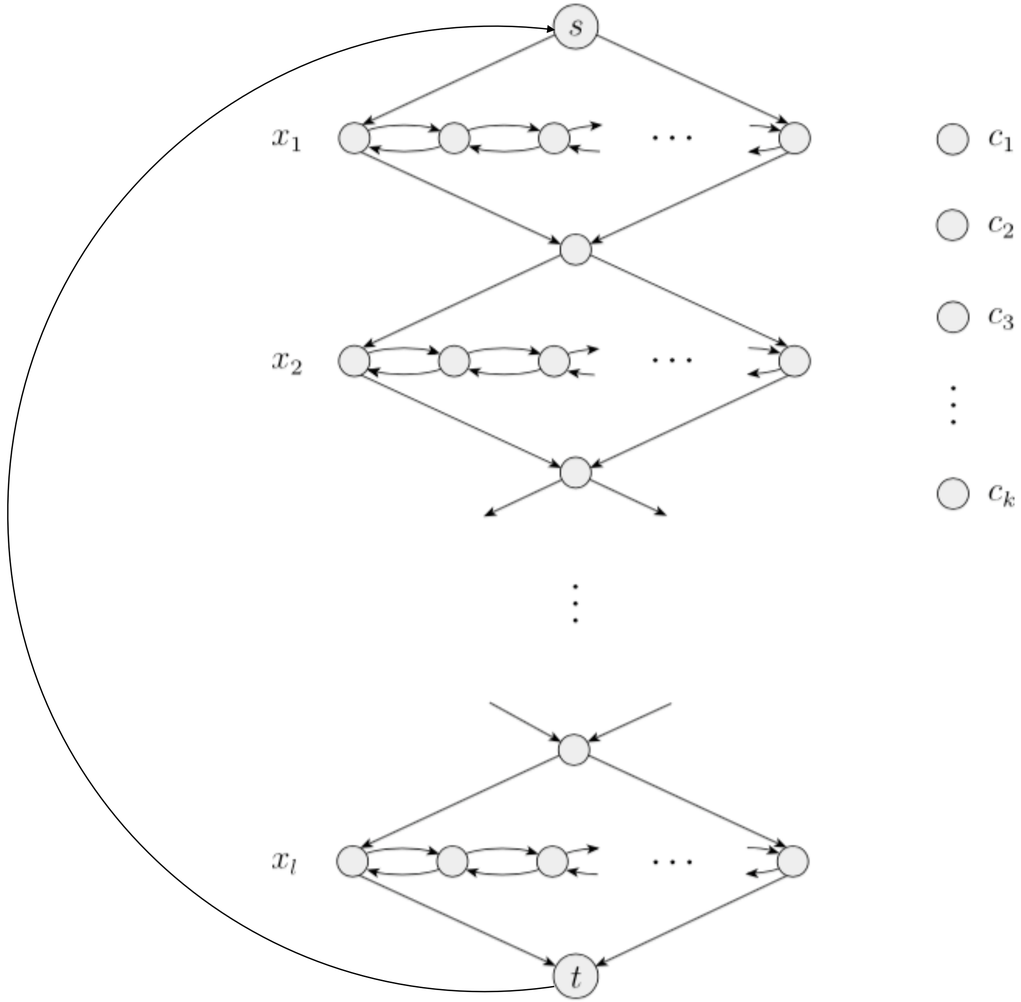
# 3SAT $\leq_p$ Hamiltonian Cycle

- Let $\Phi$ contain $k$ clauses and $\ell$ variables
- Let $x_1, \ldots, x_\ell$ denote the $\ell$ variable in $\Phi$
- Variable gadget: for each variable $x_i$ create a diamond shape structure with a horizontal row of nodes
- Clause gadget: for each clause $c_j$ we create a single node



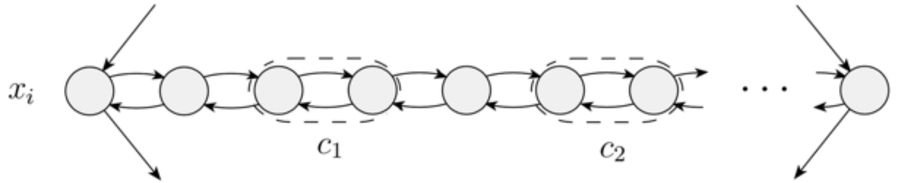
# 3SAT $\leq_p$ Hamiltonian Cycle

- Global structure

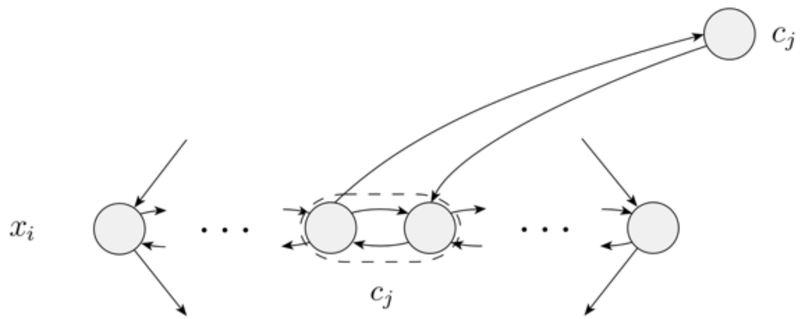


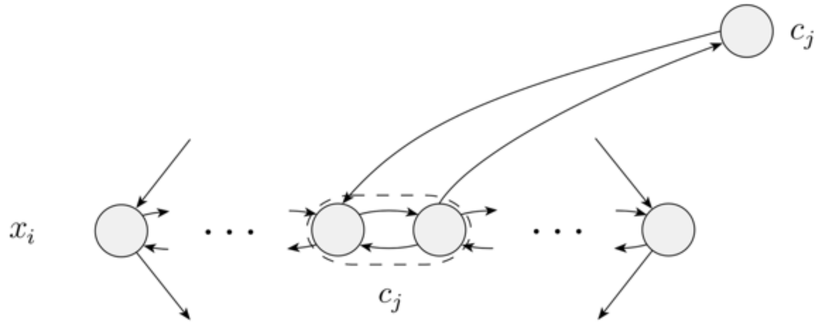
# 3SAT $\leq_p$ Hamiltonian Cycle

- **Variable gadget.** Horizontal row has $3k + 1$ internal nodes, adjacent pairs for each clause, with a separator node in between




- If $x_i$ appears in $c_j$, connect $j$th pair in the $i$th diamond to the $j$th clause as on the left. If $\overline{x_i}$ appears in $c_j$, connect it as on the right.

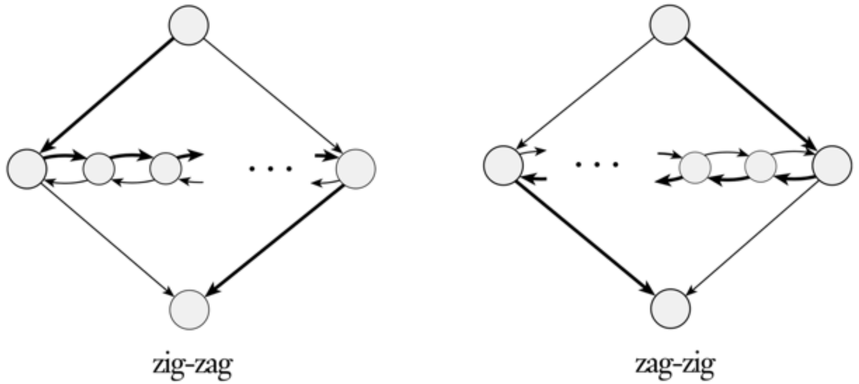


# Correctness

( $\Rightarrow$ ) Suppose $\Phi$ has a satisfying assignment, we show that $G$ has a hamiltonian cycle

- Consider cycle starting with edge $t \rightarrow s$, traversing the diamond gadgets (ignoring clauses for now) and ending up at $t$
- If $x_i$ is set to true in satisfying assignment, traverse the corresponding diamond in a zig-zag fashion, otherwise zag-zig as shown below

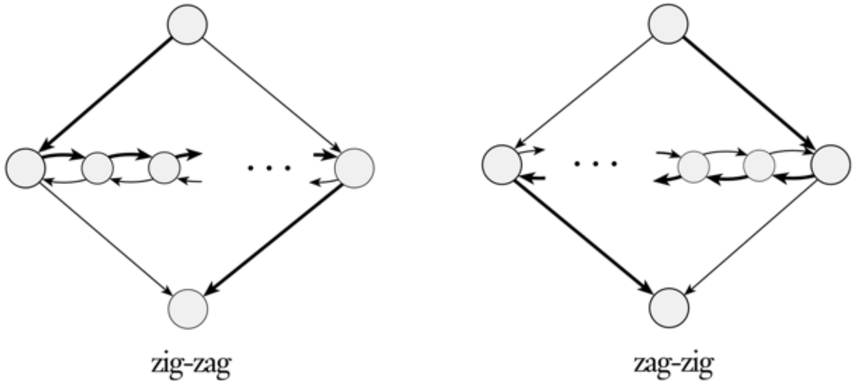


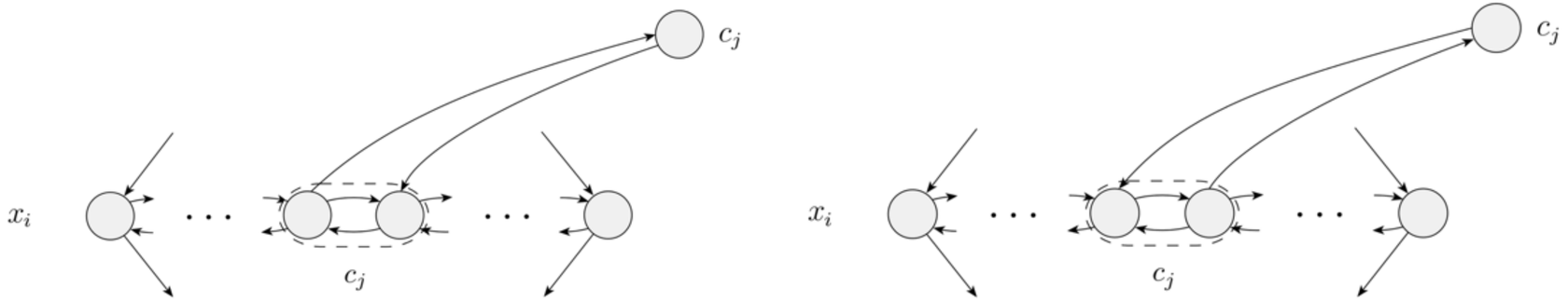

- This path hits each node exactly once except the clause nodes

# Correctness I

( $\Rightarrow$ ) For each clause select one true literal (must contain one)

- Add detours to visit each clause node $c_j$ from the selected literal $x_i$ or $\overline{x_i}$:
  - If we selected $x_i$ the path zig-zags and thus thus visit $c_j$
  - If we selected $\overline{x_i}$, the path zag-zigs and thus can visit $c_j$

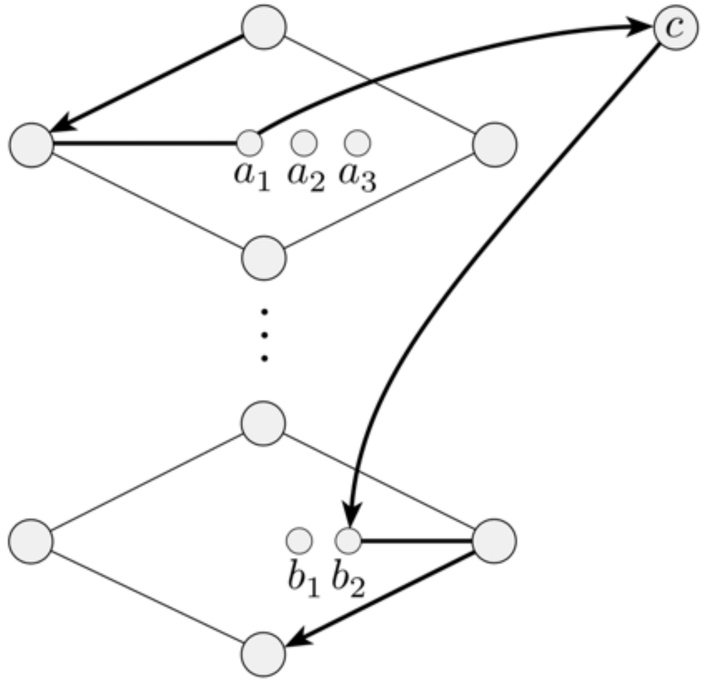


# Correctness II

( $\Leftarrow$ ) Suppose $G$ has a hamiltonian cycle, we need to construct a satisfying assignment to $\Phi$:

- Note that this hamiltonian cycle must visit each diamond from top-down with clause detours (either zig-zagging or zag-zigging)
- Situation that cannot occur: clause entered from one diamond but exited to a different (why?)
- If a diamond is traversed zig-zag: set variable to true
- Else, set it to false
- Must be a satisfying assignment, why?
  - Cycle is able to visit clause nodes
  - At least one literal set to true per clause




Such a cycle would never visit node $a_2$

# Class Exercise:

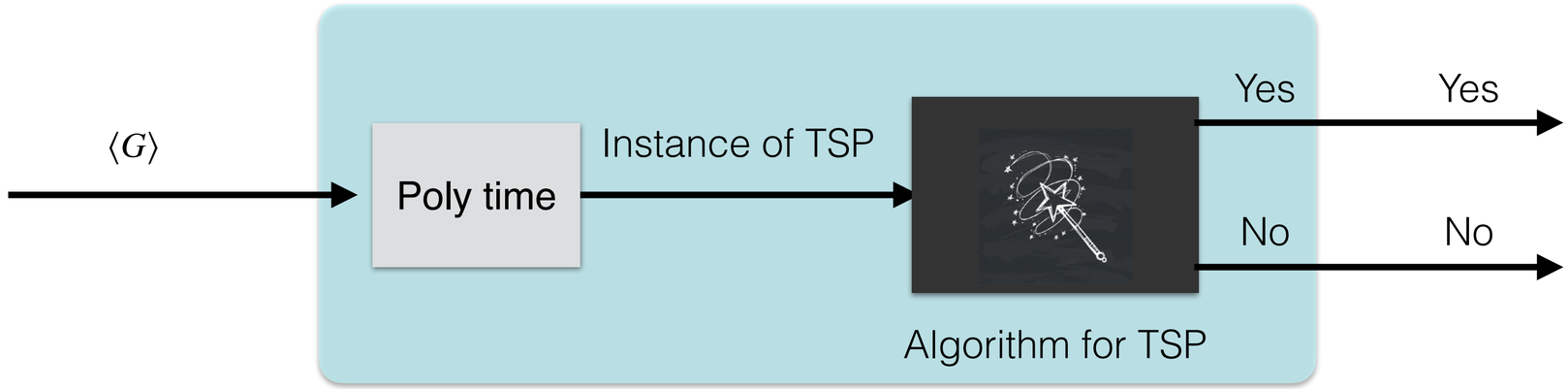
Hamiltonian-Cycle $\leq_p$ TSP

# Hamiltonian Cycle to TSP

**In Class Exercise.** **HAMILTONIAN-CYCLE $\leq_p$ TSP**

Given a directed graph $G$, convert it to an instance of TSP: that is,

- Cities $c_1, \ldots, c_n$
- $d(i, j)$: distance from city $i$ to city $j$
- Target $D$ such that $G$ has a hamiltonian cycle iff there exists a tour of $n$ cities of length at most $D$




Algorithm for HAM CYCLE

# TSP is NP Complete

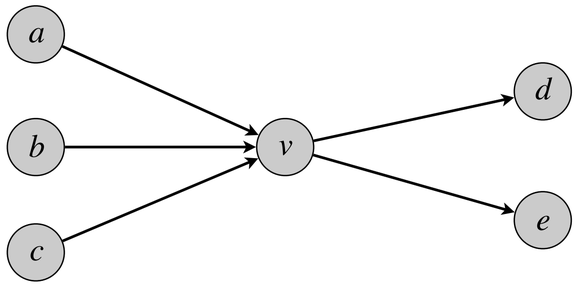
- **Claim.** **TSP** $\in$ NP
- **Claim.** **HAMILTONIAN-CYCLE** $\leq_p$ **TSP**
- **Proof.** Given directed graph $G = (V, E)$, define instance of TSP as:
  - City $v_i'$ for each node $v_i$
  - $d(i', j') = 1$ if $(v_i, v_j) \in E$
  - $d(i', j') = 2$ if $(v_i, v_j) \notin E$
- $G$ has a Hamiltonian cycle iff there is a tour of length at most $n$
- $(\Rightarrow)$ If $G$ has a hamiltonian cycle, then it defines a tour of length $n$
- $(\Leftarrow)$ Suppose there is a tour of length at most $n$, why does this ordering correspond to a hamiltonian cycle?

# (Directed) Hamiltonian Path

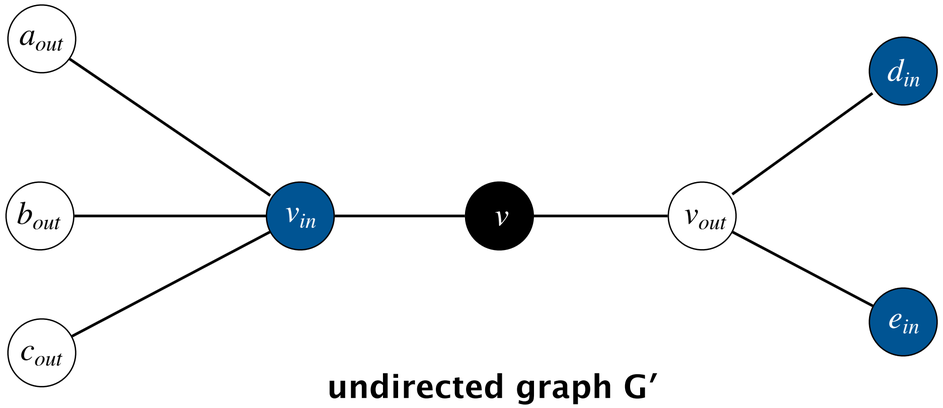
- **HAMILTONIAN-PATH.** Given a directed graph $G = (V, E)$ does there exists a path $P$ that visits every vertex exactly once? Such a path is called a hamiltonian path
- Note: path is allowed to start and end anywhere as long as it visits every node exactly once
- **HAMILTONIAN-PATH** $\in$ NP
  - Certificate: path in $G$
  - Verifier: check if path visits each node exactly once
- To prove **HAMILTONIAN PATH** is NP hard, we can either
  - We can modify our hamiltonian cycle reduction (delete $t \rightarrow s$)
  - More fun: (exercise) Directly reduce from **HAMILTONIAN CYCLE**

# Undirected Ham Path/Cycle

- Undirected version of Hamiltonian path/cycle are also NP complete
- Can reduce from directed version
- **Reduction idea:** Given a directed graph $G = (V, E)$, construct an undirected graph $G'$ with $3n$ nodes as follows:



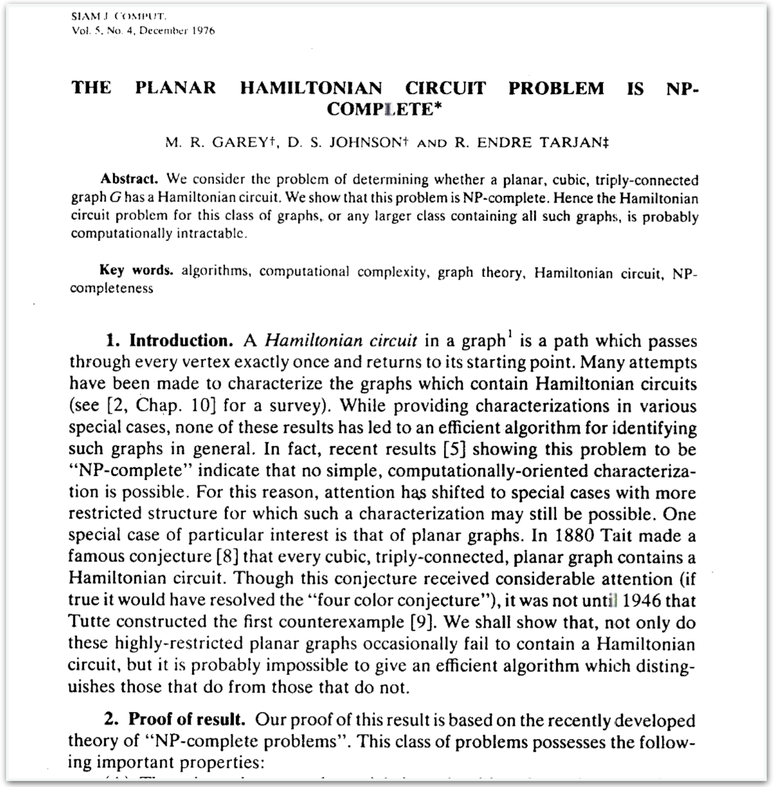

**directed graph G**




**undirected graph G′**

# Fun Facts

- Hamiltonian path problem says NP complete even on two connected, cubic and planar graphs!
- Still NP complete on general grid graphs, but poly-time solvable on “solid grid graphs” (a Williams undergrad thesis by Chris Umans)



SIAM J. COMPUT.
Vol. 5, No. 4, December 1976

**THE PLANAR HAMILTONIAN CIRCUIT PROBLEM IS NP-COMPLETE***

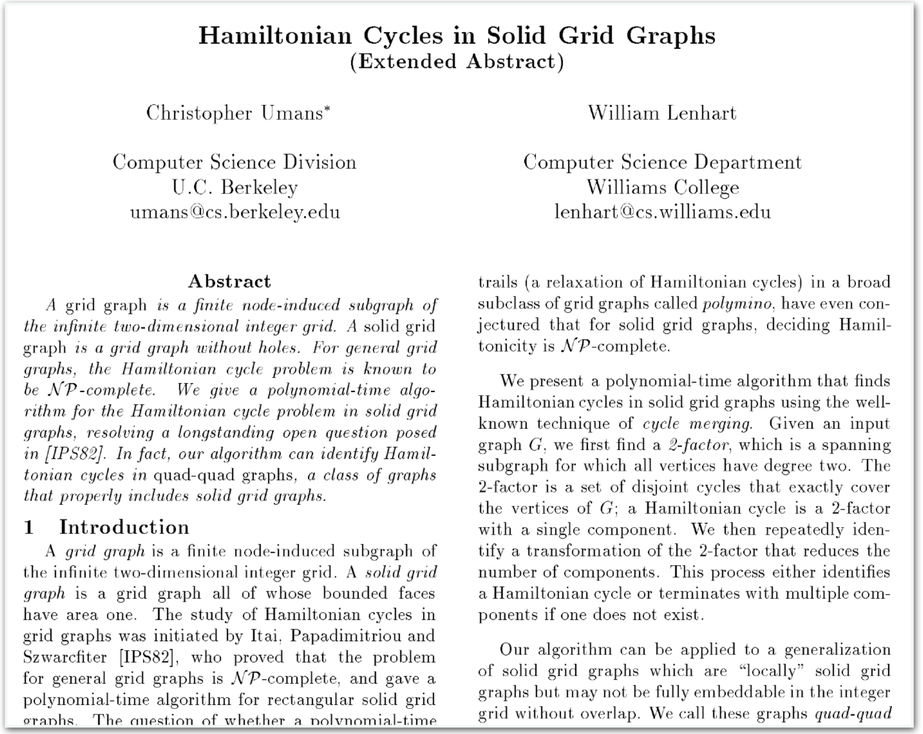
M. R. GAREY†, D. S. JOHNSON† AND R. ENDRE TARJAN‡

**Abstract.** We consider the problem of determining whether a planar, cubic, triply-connected graph $G$ has a Hamiltonian circuit. We show that this problem is NP-complete. Hence the Hamiltonian circuit problem for this class of graphs, or any larger class containing all such graphs, is probably computationally intractable.

**Key words.** algorithms, computational complexity, graph theory, Hamiltonian circuit, NP-completeness

**1. Introduction.** A *Hamiltonian circuit* in a graph[1] is a path which passes through every vertex exactly once and returns to its starting point. Many attempts have been made to characterize the graphs which contain Hamiltonian circuits (see [2, Chap. 10] for a survey). While providing characterizations in various special cases, none of these results has led to an efficient algorithm for identifying such graphs in general. In fact, recent results [5] showing this problem to be "NP-complete" indicate that no simple, computationally-oriented characterization is possible. For this reason, attention has shifted to special cases with more restricted structure for which such a characterization may still be possible. One special case of particular interest is that of planar graphs. In 1880 Tait made a famous conjecture [8] that every cubic, triply-connected, planar graph contains a Hamiltonian circuit. Though this conjecture received considerable attention (if true it would have resolved the "four color conjecture"), it was not until 1946 that Tutte constructed the first counterexample [9]. We shall show that, not only do these highly-restricted planar graphs occasionally fail to contain a Hamiltonian circuit, but it is probably impossible to give an efficient algorithm which distinguishes those that do from those that do not.

**2. Proof of result.** Our proof of this result is based on the recently developed theory of "NP-complete problems". This class of problems possesses the following important properties:



**Hamiltonian Cycles in Solid Grid Graphs**
**(Extended Abstract)**

Christopher Umans*

Computer Science Division
U.C. Berkeley
umans@cs.berkeley.edu

William Lenhart

Computer Science Department
Williams College
lenhart@cs.williams.edu

**Abstract**

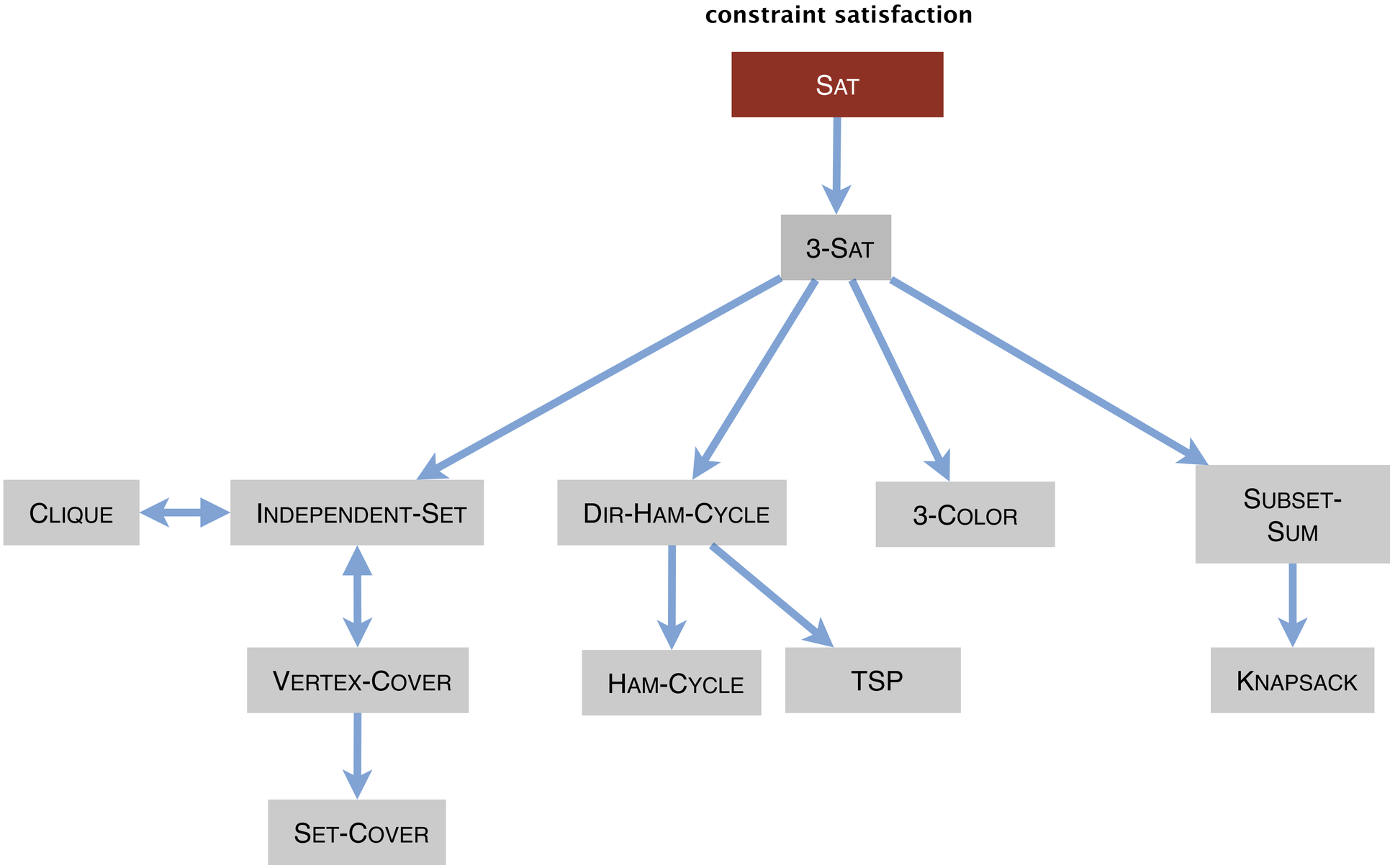
*A* grid graph *is a finite node-induced subgraph of the infinite two-dimensional integer grid. A* solid grid graph *is a grid graph without holes. For general grid graphs, the Hamiltonian cycle problem is known to be $\mathcal{NP}$-complete. We give a polynomial-time algorithm for the Hamiltonian cycle problem in solid grid graphs, resolving a longstanding open question posed in [IPS82]. In fact, our algorithm can identify Hamiltonian cycles in* quad-quad graphs, *a class of graphs that properly includes solid grid graphs.*

## 1 Introduction

A *grid graph* is a finite node-induced subgraph of the infinite two-dimensional integer grid. A *solid grid graph* is a grid graph all of whose bounded faces have area one. The study of Hamiltonian cycles in grid graphs was initiated by Itai, Papadimitriou and Szwarcfiter [IPS82], who proved that the problem for general grid graphs is $\mathcal{NP}$-complete, and gave a polynomial-time algorithm for rectangular solid grid graphs. The question of whether a polynomial-time trails (a relaxation of Hamiltonian cycles) in a broad subclass of grid graphs called *polymino*, have even conjectured that for solid grid graphs, deciding Hamiltonicity is $\mathcal{NP}$-complete.

We present a polynomial-time algorithm that finds Hamiltonian cycles in solid grid graphs using the well-known technique of *cycle merging*. Given an input graph $G$, we first find a *2-factor*, which is a spanning subgraph for which all vertices have degree two. The 2-factor is a set of disjoint cycles that exactly cover the vertices of $G$; a Hamiltonian cycle is a 2-factor with a single component. We then repeatedly identify a transformation of the 2-factor that reduces the number of components. This process either identifies a Hamiltonian cycle or terminates with multiple components if one does not exist.

Our algorithm can be applied to a generalization of solid grid graphs which are "locally" solid grid graphs but may not be fully embeddable in the integer grid without overlap. We call these graphs *quad-quad*


constraint satisfaction
SAT
3-SAT
CLIQUE
INDEPENDENT-SET
DIR-HAM-CYCLE
3-COLOR
SUBSET-SUM
VERTEX-COVER
HAM-CYCLE
TSP
KNAPSACK
SET-COVER

# Useful NP-hard Problems

- **BIN-PACKING.** Given a set of items $I = \{1, \ldots, n\}$ where item $i$ has size $s_i \in (0,1]$, bins of capacity $c$, find an assignment of items to bins that minimizes the number of bins used?
- **PARTITION.** Given a set $S$ of $n$ integers, are there subsets $A$ and $B$ such that $A \cup B = S$, $A \cap B = \emptyset$ and $\sum_{a \in A} a = \sum_{b \in B} b$
- **MAXCUT.** Given an undirected graph $G = (V, E)$, find a subset $S \subset V$ that maximizes the number of edges with exactly one endpoint in $S$.
- **MAX-2-SAT.** Given a Boolean formula in CNF, with exactly two literals per clause, find a variable assignment that maximizes the number of clauses with at least one true literal. (**2-SAT** on the other hand is in **P**)
- **3D-MATCHING.** Given $n$ instructors, $n$ courses, and $n$ times, and a list of the possible courses and times each instructor is willing to teach, is it possible to make an assignment so that all courses are taught at different times?

# More hard computational problems

Aerospace engineering. Optimal mesh partitioning for finite elements.

Biology. Phylogeny reconstruction.

Chemical engineering. Heat exchanger network synthesis.

Chemistry. Protein folding.

Civil engineering. Equilibrium of urban traffic flow.

Economics. Computation of arbitrage in financial markets with friction.

Electrical engineering. VLSI layout.

Environmental engineering. Optimal placement of contaminant sensors.

Financial engineering. Minimum risk portfolio of given return.

Game theory. Nash equilibrium that maximizes social welfare.

Mathematics. Given integer $a_1, \ldots, a_n$, compute

Mechanical engineering. Structure of turbulence in sheared flows.

Medicine. Reconstructing 3d shape from biplane angiocardiogram.

Operations research. Traveling salesperson problem.

Physics. Partition function of 3d Ising model.

Politics. Shapley–Shubik voting power.

Recreation. Versions of Sudoku, Checkers, Minesweeper, Tetris, Rubik's Cube.

Statistics. Optimal experimental design.

# Fun NP-hard Games

- **MINESWEEPER** (from CIRCUIT-SAT)
- **SODUKO** (from 3-SAT)
- **TETRIS** (from 3PARTITION)
- **SOLITAIRE** (from 3PARTITION)
- **SUPER MARIO BROTHERS** (from 3-SAT)
- **CANDY CRUSH SAGA** (from 3-SAT variant)
- **PAC-MAN** (from Hamiltonian Cycle)
- **RUBIK's CUBE** (recent 2017 result, from Hamiltonian Cycle)
- **TRAINYARD** (from Dominating Set)

# Acknowledgments


- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)
  - Hamiltonian cycle reduction images from Michael Sipser's Theory of Computation Book