Wrapping Up Intractability

Reminders and Overview

- Assignment 7 is due coming Wednesday
  - Extra TA hours **today** for this assignment: **2-4 pm** and **5-6 pm**!
- Complete a reductions: see textbook readings!
- **Colloquium today 3.15 pm:** Data Structure Design for Skewed Dynamic Graph Processing by Helen Xu (MIT)
- Ten more lectures (overview):
  - Today we wrap up intractability
  - Next: Probability and randomized algorithms
  - Last: Approximation algorithms

Helen Xu (MIT)

Class Exercise:

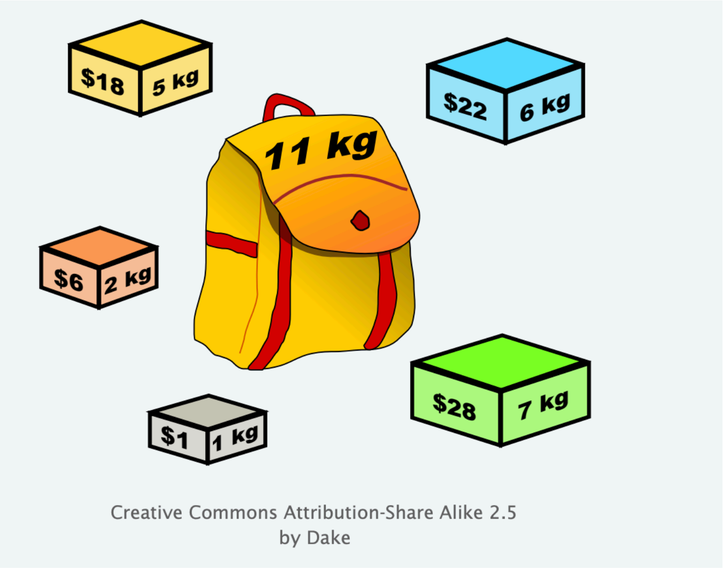
SUBSET-SUM $\leq_p$ Knapsack

Recap: Subset Sum

- **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
- We proved **SUBSET-SUM** is NP hard by a reduction from vertex cover
- Since it has a pseudo-polynomial $O(nT)$-time algorithm it is a weakly NP hard problem
- Today: use subset sum to prove that Knapsack is also NP hard

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

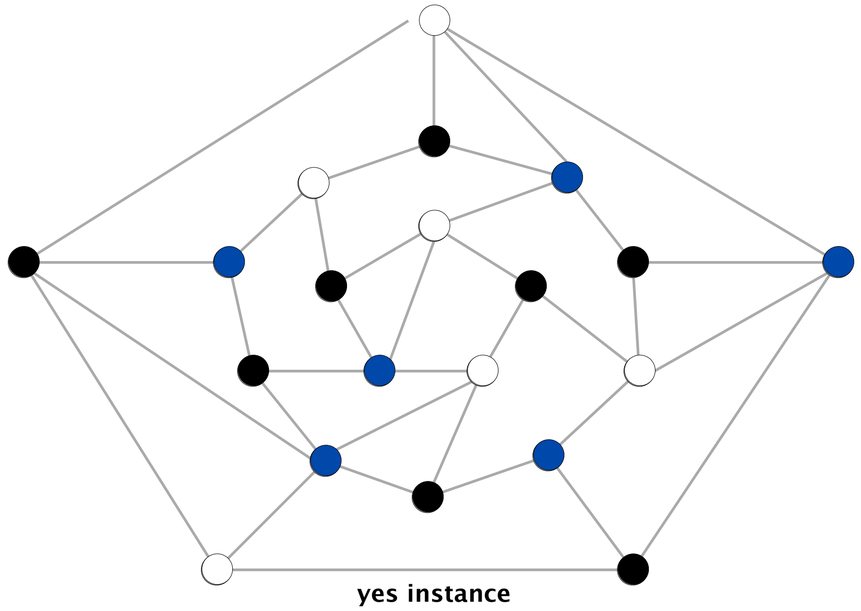
- **Knapsack.** Given n elements $a_1, \ldots, a_n$ where each element has a weight $w_i \geq 0$ and a value $v_i \geq 0$ and target weight W and value K. Does there exist a subset X of numbers such that
  - $\sum_{a_i \in X} w_i \leq W$ and $\sum_{a_i \in X} v_i \geq K$
- **Subset-Sum $\leq_p$ Knapsack Proof idea:**
  - $K = W = T$ and $w_i = v_i = a_i$ for all i
- If ∃ subset S s.t. $\sum_{i \in S} a_i = T$, then pick those S to be in Knapsack
- If ∃ a subset X s.t. total weight of items $\leq W = T$ and the total value $\geq K = T$, then X is the subset of items that sums to exactly T

Graph-3-Color is NP Complete:

3-SAT $\leq_p$ Graph 3-Color

Graph 3-Color Problem

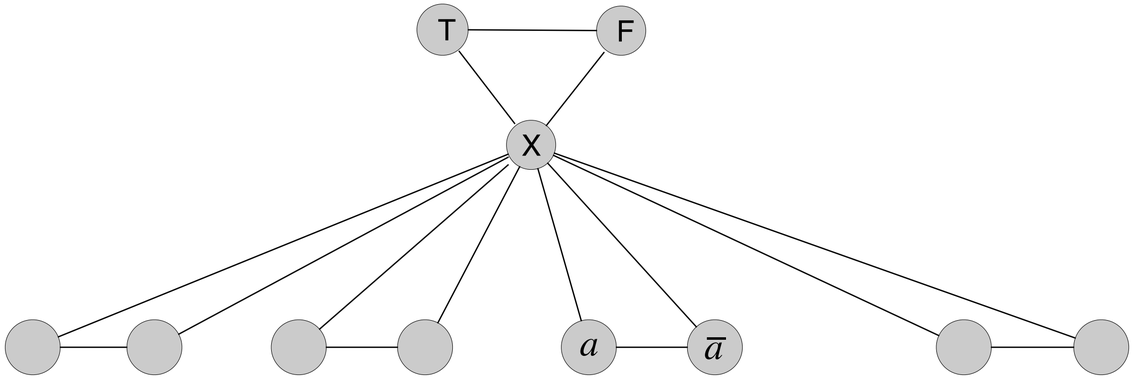
- **3-COLOR.** Given an undirected graph $G = (V, E)$, is it possible to color the vertices with 3 colors s.t. no adjacent nodes have the same color.
- We have shown in the past that **3-COLOR** $\in$ NP.



yes instance

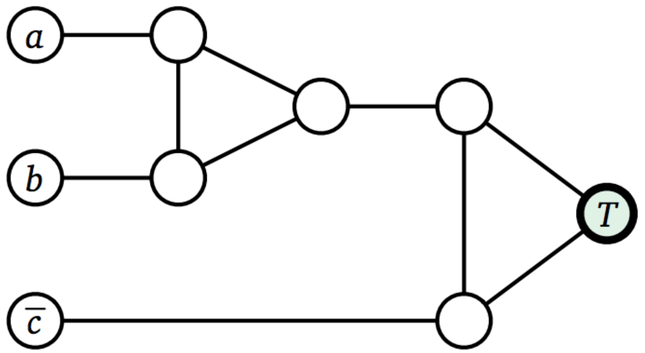
3-SAT to 3-Color Problem

- **Theorem.** **3-SAT** $\leq_p$ **3-COLOR**
- **Proof.** Given a **3-SAT** instance Φ, 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:** a 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

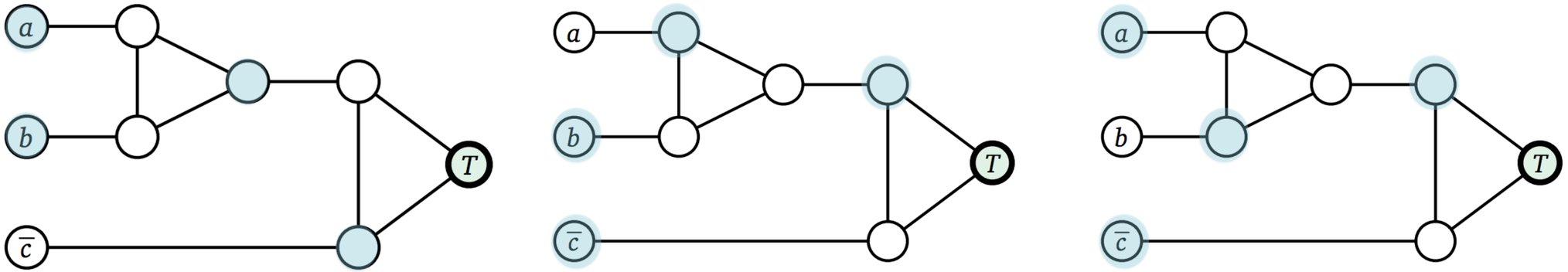
- Given a **3-SAT** instance Φ, define G as follows
  - **Truth gadget**: a triangle with three nodes $T, F,$ and X (for true, false and other) — they must get different colors (say true, false, other)
  - **Variable gadget:** triangle made up of variable a, its negation $\overline{a}$ and the X node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

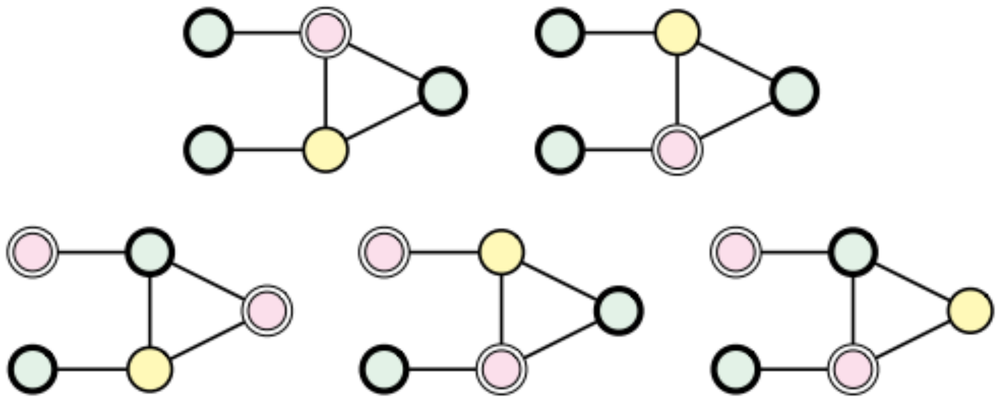
- **Observation**. A clause gadget has a valid 3-coloring, if and only if, **at least one** literal is colored True
  - If a, b (or $b, \overline{c}$) or ($a, \overline{c}$) get the same color (say, FALSE) then the right-end-point of the triangle must be colored the same (shown in blue)
  - The remaining literal must be colored True!

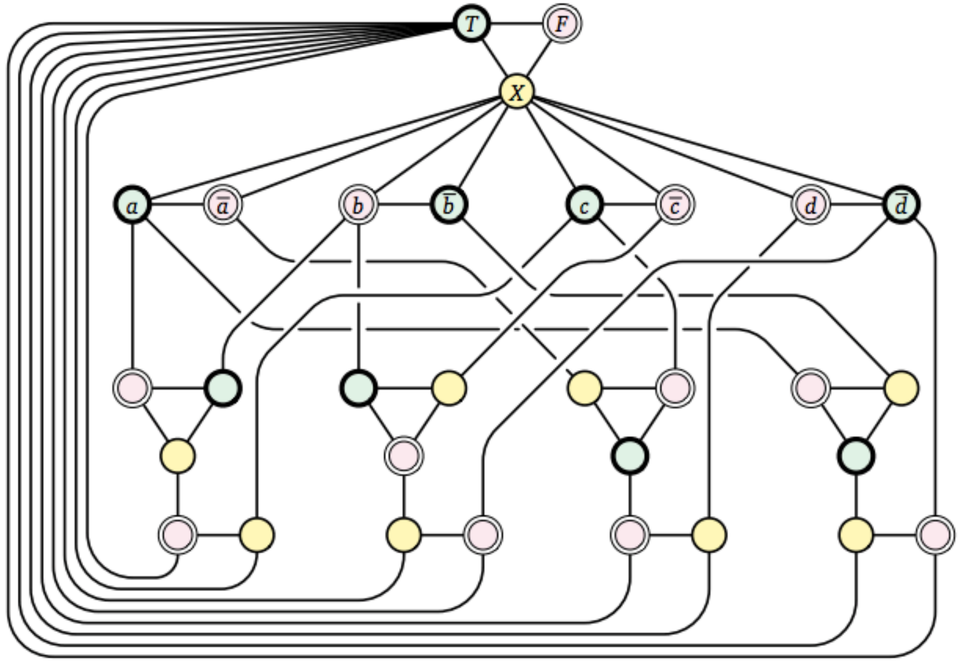


$(a \vee b \vee \overline{c})$

3-SAT to 3-Color Problem

- **Example.** All valid 3-colorings of the “half-gadget" of the clause on the left
- Overall G for example instance on the right





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

3-SAT to 3-Color Problem

Correctness of reduction

- ($\Rightarrow$) If Φ is satisfiable, color the variable gadget based on the satisfying assignment
- Why can we extend this coloring to a valid 3-coloring of the clause gadgets?
    - Because at least one literal in each clause must be True
- ($\Leftarrow$) If G is 3-colorable, then the variable nodes must be colored T or F (because of the variable gadget), we can assign truth values based on the colors
- Why is this a satisfying assignment?
    - At least one of the literals in each clause must be colored true and thus the resulting assignment must satisfy Φ

3-SAT to 3-Color Problem

Running time of reduction

- If ϕ has n variables and k clauses, then our resulting graph G has at most $2n + 5k + 3$ nodes
- Thus, we can construct G in $O(n + k)$ time
- Reduction is polynomial time!

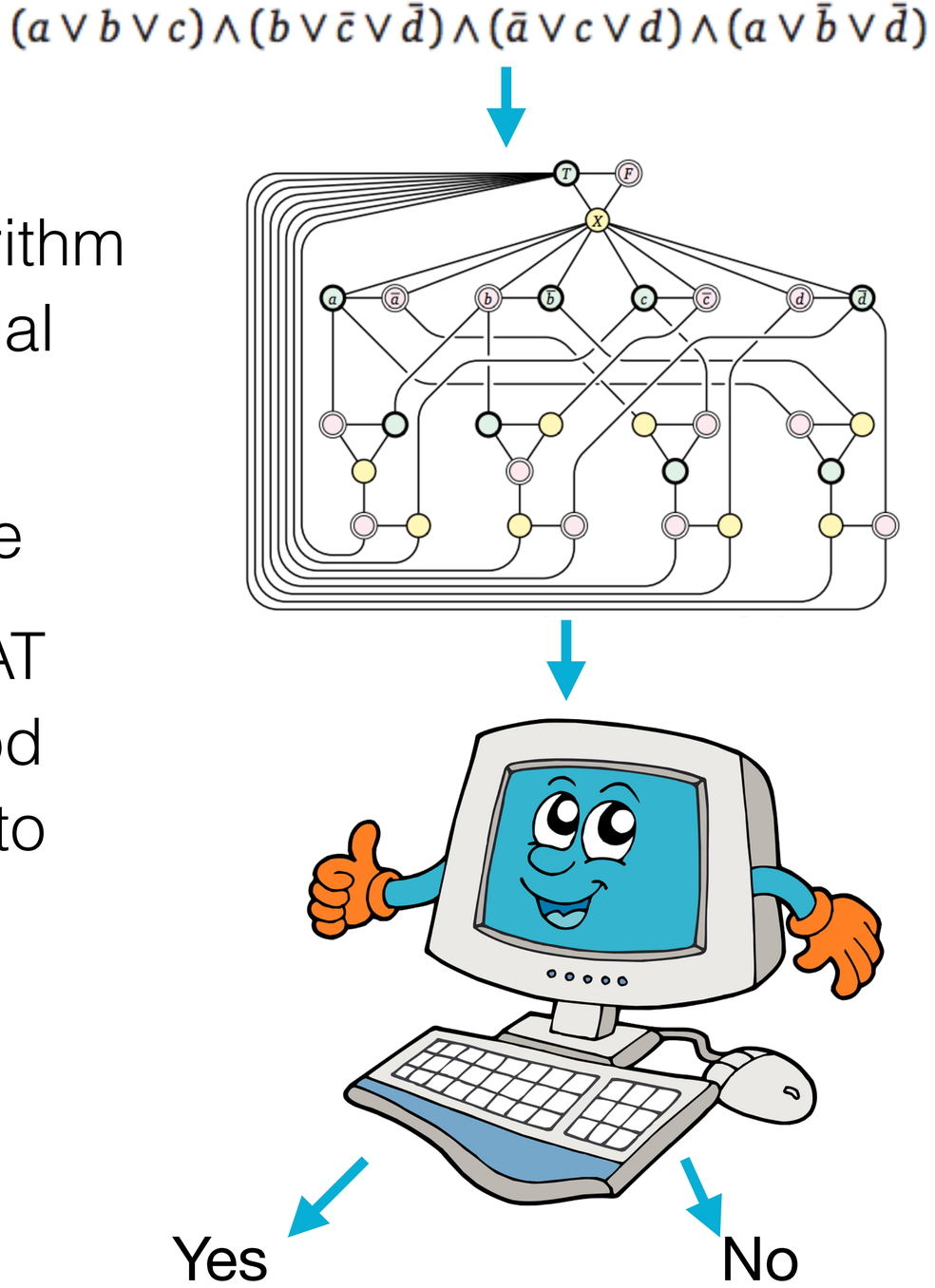
NP Hardness

- Another explanation about what we're proving and why

    - What do you start with an instance of?
    - What information do you have?
    - What do you need to prove?

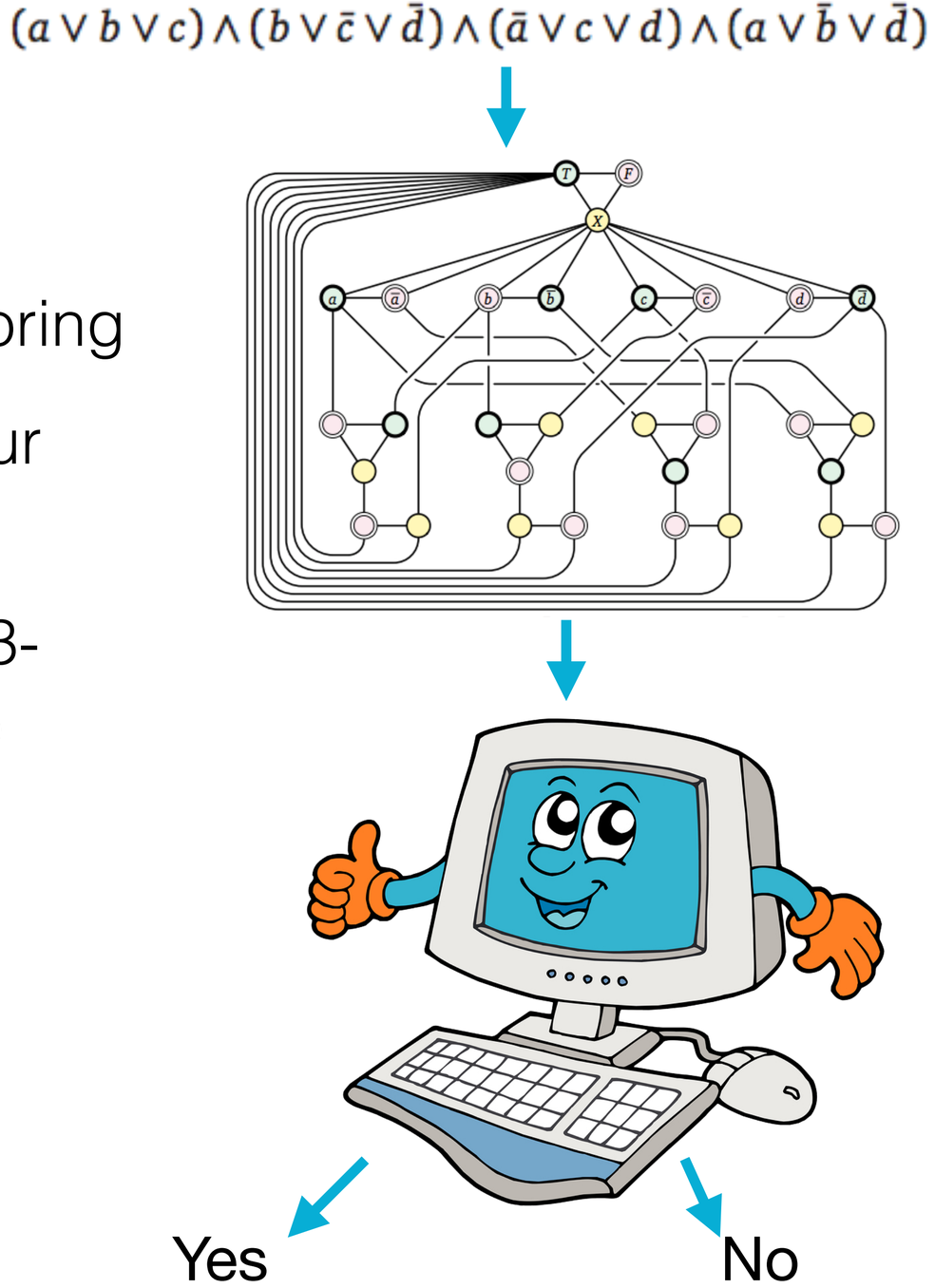
NP Hardness

- Let's say you told me you had an algorithm that could solve 3-coloring in polynomial time
- Then I come to you with a SAT instance
- In polynomial time we transform the SAT instance into a graph (using the method from last slide), and feed that graph into your algorithm



NP Hardness

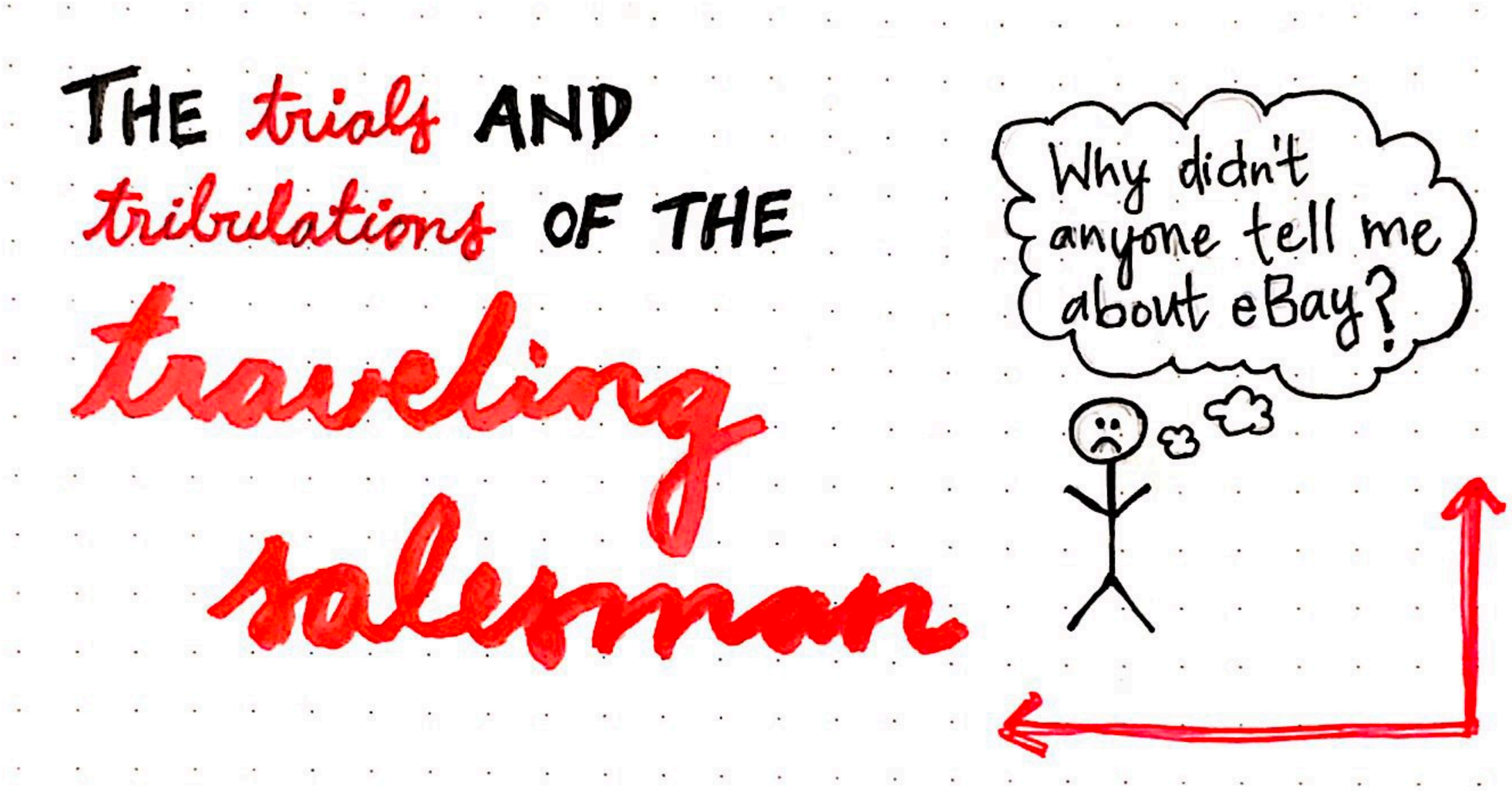
- (We proved) If my SAT instance is satisfiable, your algorithm finds a 3-coloring
- If my SAT instance is not satisfiable, your algorithm does not find a 3-coloring
    - Same as: if your algorithm finds a 3-coloring, my SAT instance must be satisfiable
- Contradiction! SAT (probably) can't be solved in polynomial time



List of NPC Problems So Far

- **SAT/ 3-SAT**
- **INDEPENDENT SET**
- Covering problems: **VERTEX COVER, SET COVER**
- **CLIQUE**
- **3-COLOR**
- Packing problems: **Subset-Sum, Knapsack**
- Next (Sequencing problems):
    - **Traveling salesman problem**
    - **Hamiltonian cycle / path**

Traveling Salesman Problem



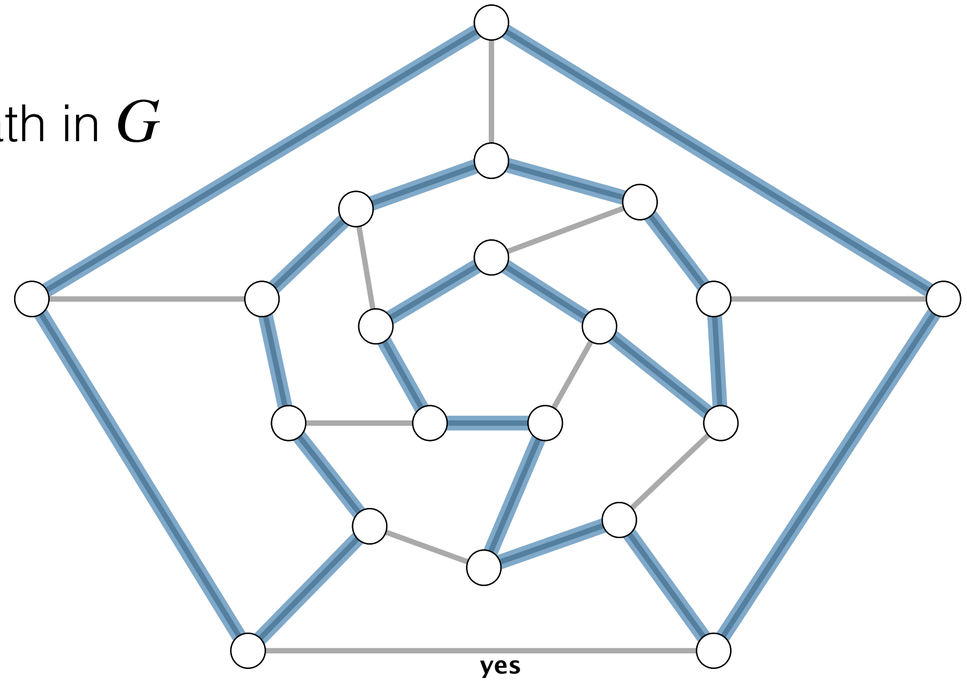
Vaidehi Joshi
https://medium.com/basecs/the-trials-and-tribulations-of-the-traveling-salesman-56048d6709d

Traveling Salesman Problem

- 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)
- **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?
- Famous problem with **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**
    - We will reduce **3SAT** to it (time permitting)
- Let us assume it is NP hard for now



Class Exercise:

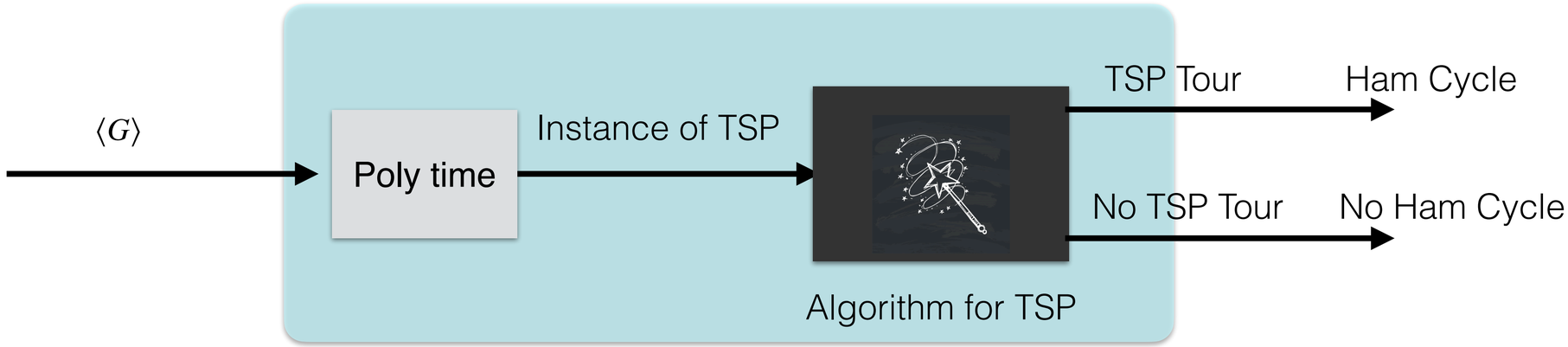
Hamiltonian-Cycle $\leq_p$ TSP

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

- **Step 1**. TSP $\in$ NP
  - Certificate: tour of cities
  - Polynomial-time verifier: check if tour starts and ends at c_1 and visits all c_i for $2 \leq i \leq n$ and is the tour length is at most D (linear time verifier)
- **Step 2. Reduction** from Hamiltonian cycle to TSP
  - Hamiltonian cycle input: directed graph G
  - What describes an instance of TSP?
    - Cities $c_1, \ldots, c_n$
    - Distance function d_{ij} between cities and target D

TSP is NP Complete

- **Step 2.** **[Reduction]** Given directed graph $G = (V, E)$, define instance of TSP as:
  - City v_i' for each node v_i
  - $d(i', j') = 1$ if $(v_i, v_j) \in E$ and $d(i', j') = 2$ if $(v_i, v_j) \notin E$
  - Target $D = n = |V|$
- **Step 3.** Prove reduction is correct:
  - If G has a Hamiltonian cycle, then TSP instance has a valid tour of length at most D
  - If G does not have a Hamiltonian cycle, then TSP instance does not have a valid tour of length at most D
    - Same as showing if TSP instance has a valid tour of length at most D then G has a Hamiltonian cycle

TSP is NP Complete

Proof of correctness:

- ($\Rightarrow$) If G has a Hamiltonian cycle, then TSP instance has a valid tour of length at most D
- Let $C = u_1, \ldots, u_n, u_1$ be the Hamiltonian cycle in G
- What is the corresponding TSP tour in our reduced instance?
  - Same tour: $u'_1, \ldots, u'_n, u'_1$
- Why is a valid tour of TSP?
  - Visits all n cities and starts and ends at the same city
  - Total distance: $\sum_{i=1}^{n} d(i, i+1) = \sum_{i=1}^{n} 1 = n \leq D$ (because each $(u_i, u_{i+1}) \in E$)

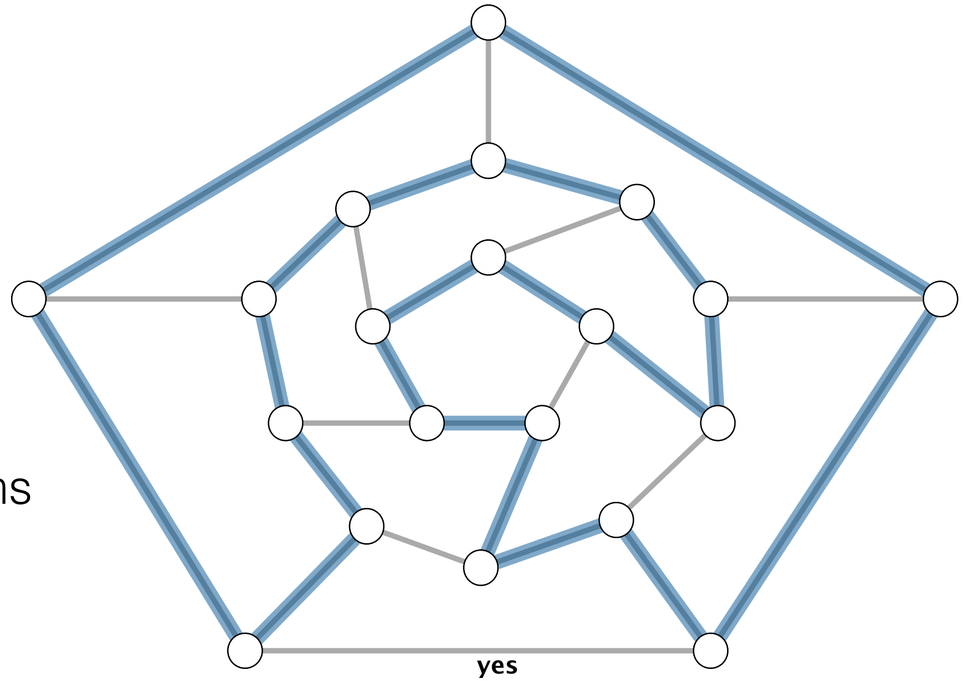
TSP is NP Complete

Proof of correctness:

- ($\Leftarrow$) If the TSP instance has a valid tour of length at most $D = n$, then G has a Hamiltonian cycle
- Let $C = u'_1, \ldots, u'_n, u'_1$ be the TSP tour of length at most n
- What can we conclude about each adjacent city (u_i, u_{i+1}) in tour?
  - $d_{i,i+1} = 1$ for all $1 \leq i \leq n - 1$ which means $(u_i, u_{i+1}) \in E$
- What is the corresponding Hamiltonian cycle in G?
  - C is the Hamiltonian cycle: it visits each node exactly once, why?(
    - Since tour visits each city and has length exactly n

(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



Final Reduction (High Level):

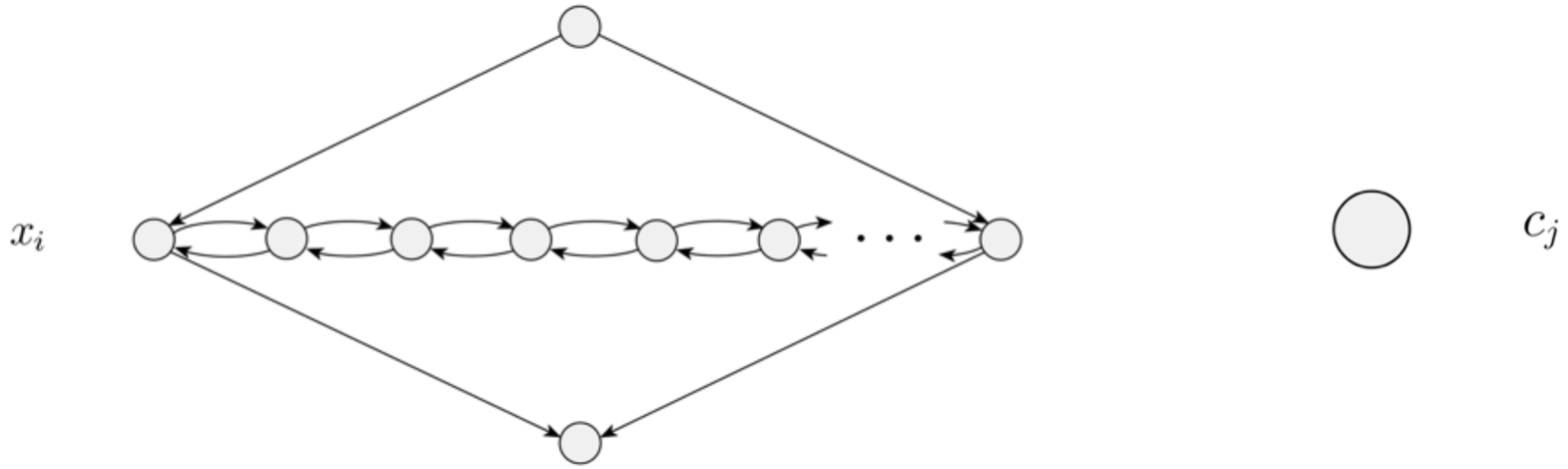
3SAT $\leq_p$ Hamiltonian Cycle

3SAT $\leq_p$ Hamiltonian Cycle

- Given 3SAT instance Φ, transform it to directed graph G s.t. Φ is satisfiable iff G has a hamiltonian cycle
- Essential ingredients of a input assignments of Φ
  - 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
  - Hook them up together appropriately

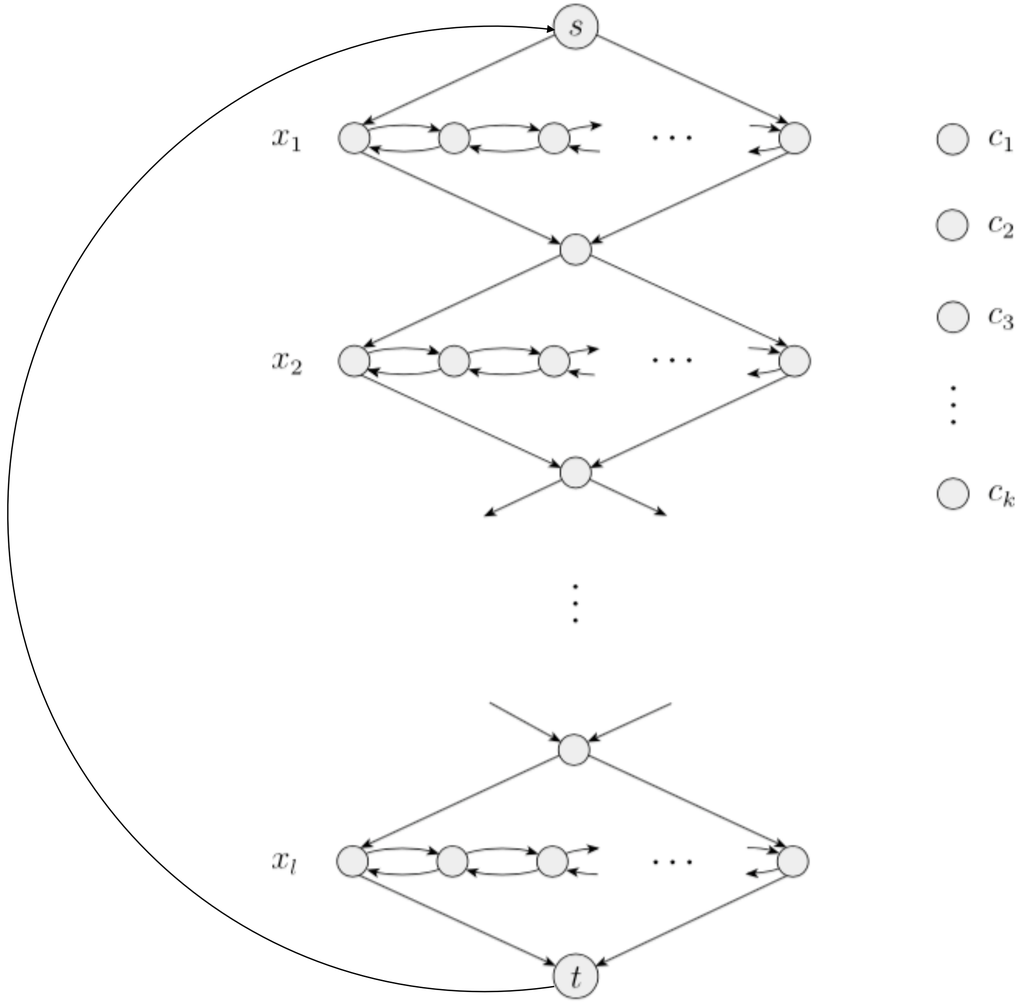
3SAT $\leq_p$ Hamiltonian Cycle

- Let Φ contain k clauses and ℓ variables
- Let $x_1, \ldots, x_\ell$ denote the ℓ variable in Φ
- 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



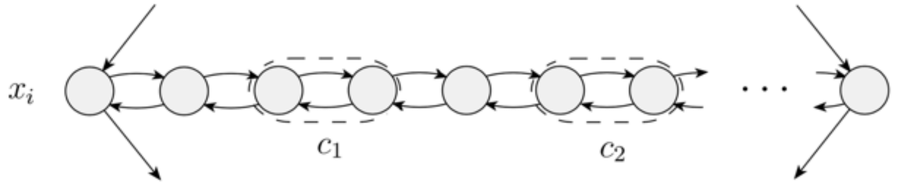
3SAT $\leq_p$ Hamiltonian Cycle

- Global structure

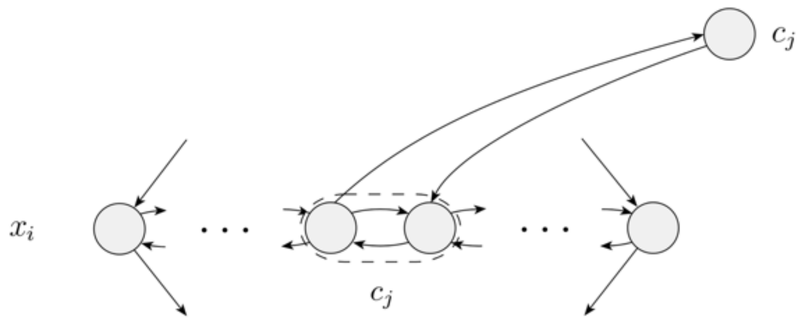


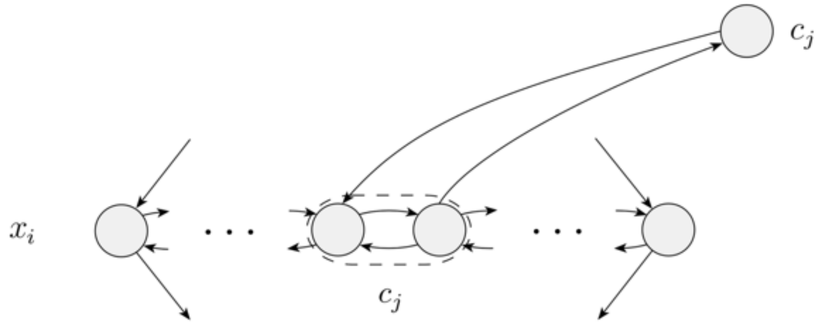
3SAT $\leq_p$ Hamiltonian Cycle

- **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 jth pair in the ith diamond to the jth clause as on the left. If $\overline{x}_i$ appears in c_j, connect it as on the right.

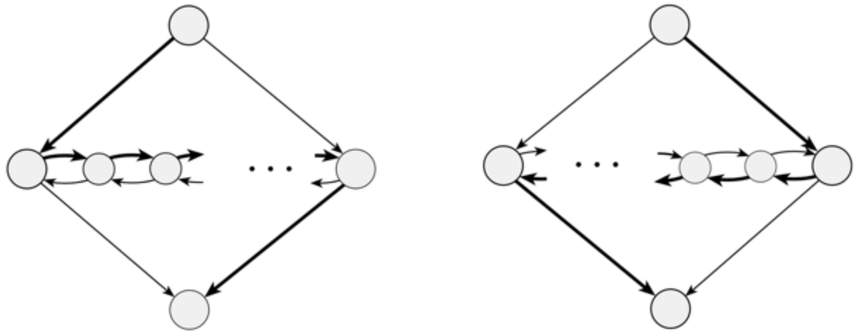




Correctness

$(\Rightarrow)$ Suppose Φ 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

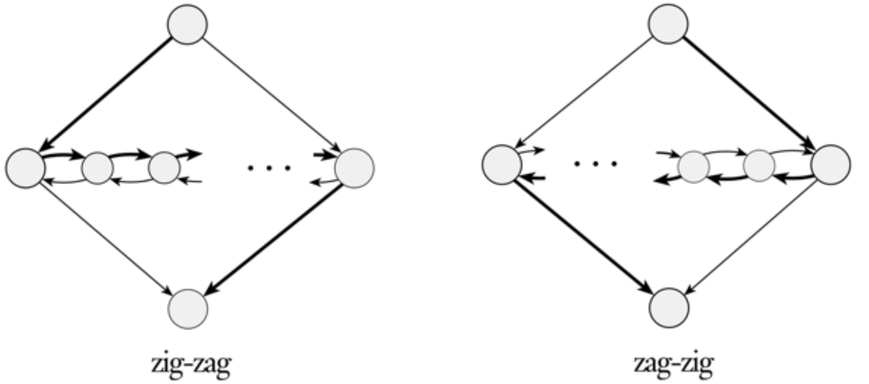


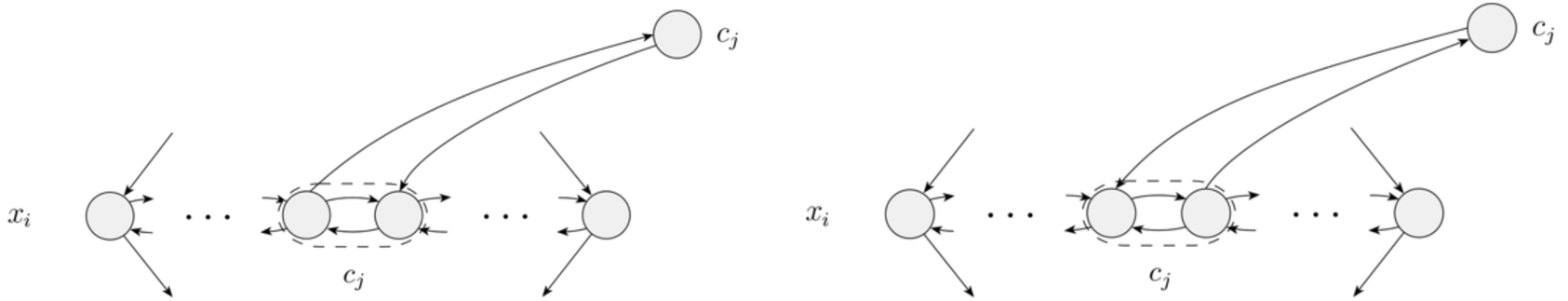
- This path hits each node exactly once except the clause nodes

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

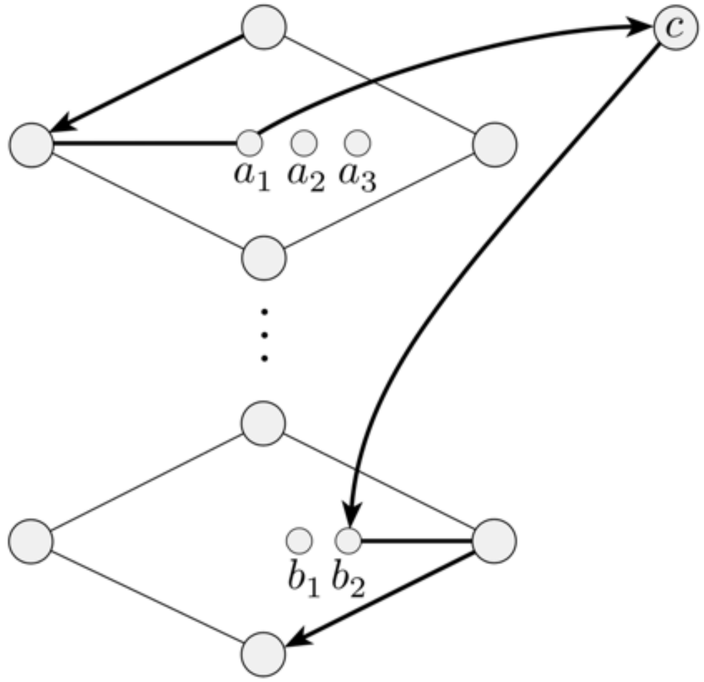




Correctness II

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

- 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
- If a diamond is traverses 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

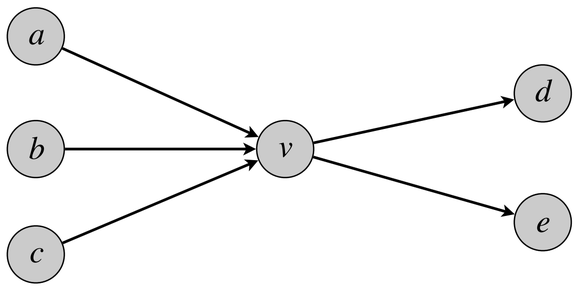
Hamiltonian Variants:
Undirected, Paths, Cycles

(Directed) Hamiltonian Path

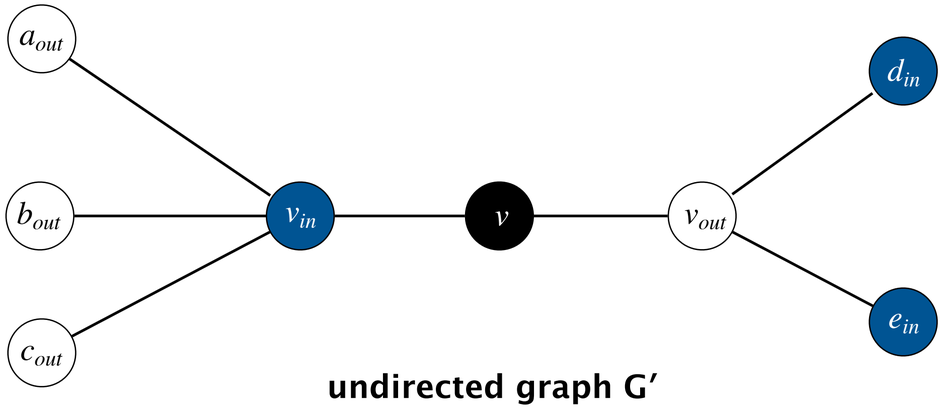
- **HAMILTONIAN-PATH.** Given a directed graph $G = (V, E)$ does there exists a path P that visits every vertex exactly once? Such a path is called a hamiltonian path
- Note: path is allowed to start and end anywhere as long as it visits every node exactly once
- **HAMILTONIAN-PATH** $\in$ NP
  - Certificate: 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:



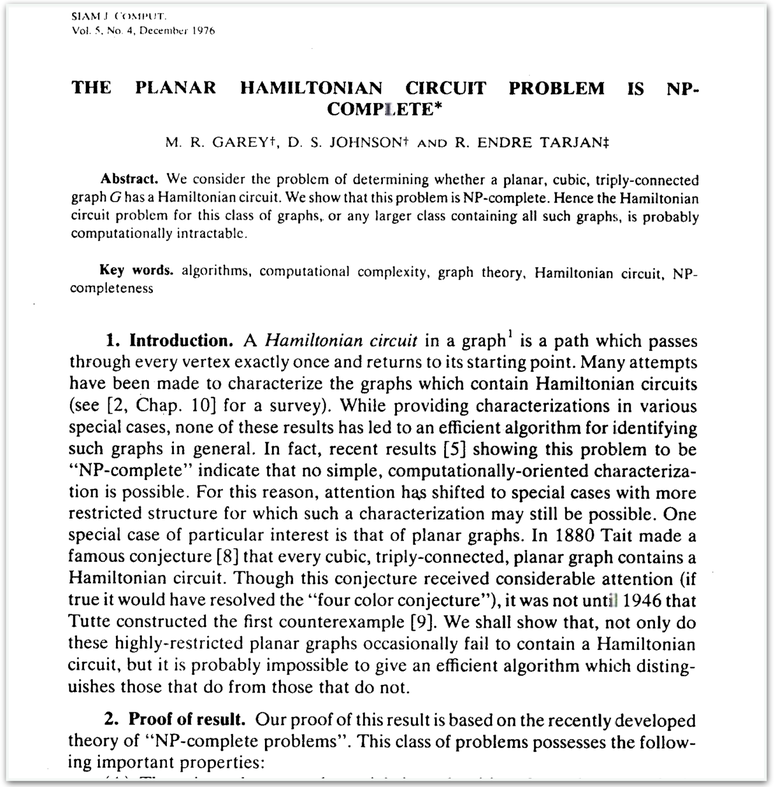
directed graph G



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*

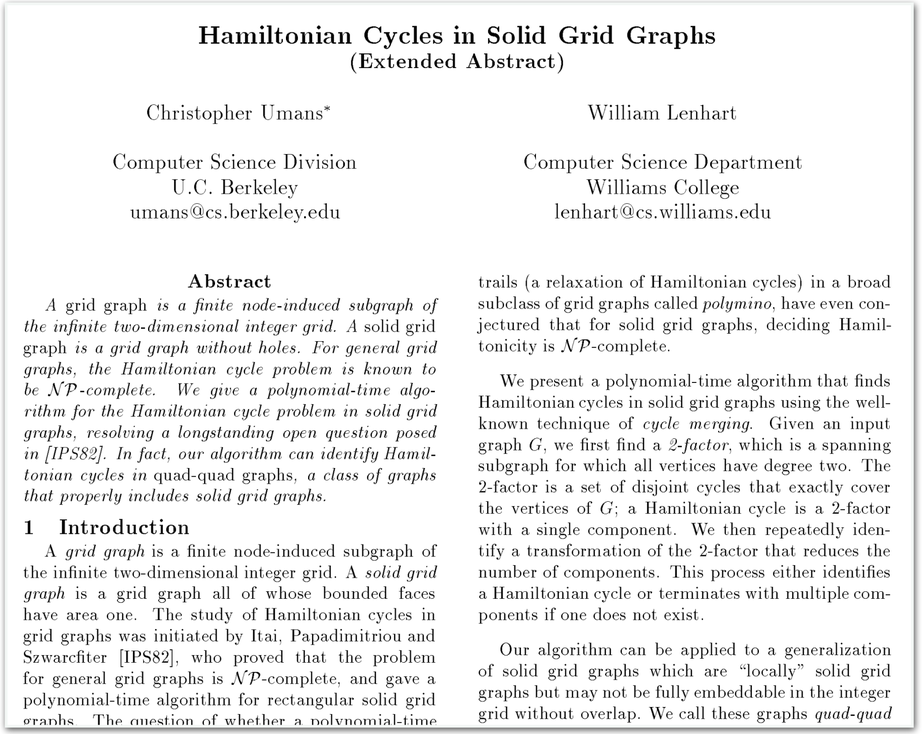
M. R. GAREY†, D. S. JOHNSON† AND R. ENDRE TARJAN‡

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

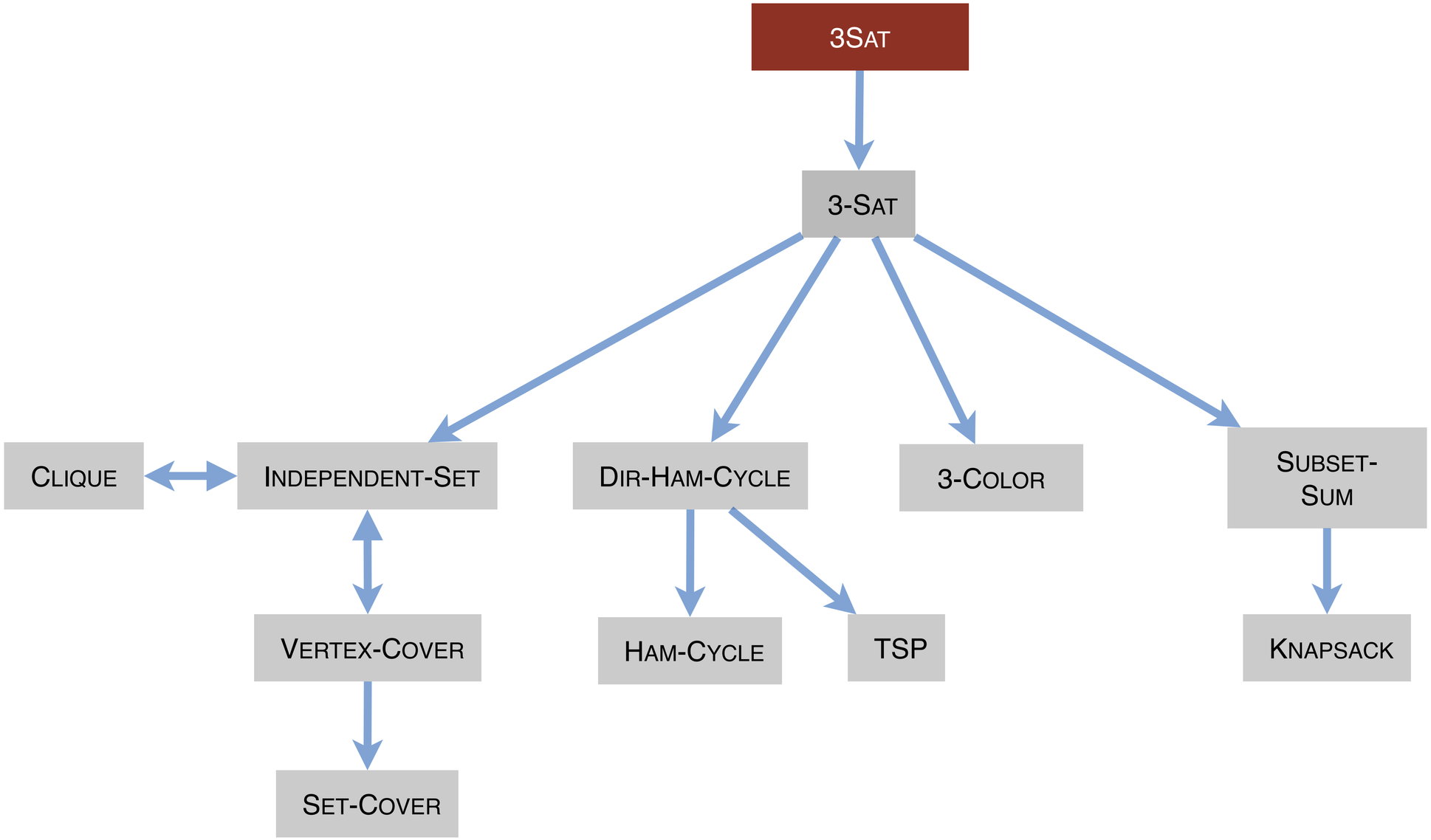
A grid graph *is a finite node-induced subgraph of the infinite two-dimensional integer grid. A* solid grid graph *is a grid graph without holes. For general grid graphs, the Hamiltonian cycle problem is known to be $\mathcal{NP}$-complete. We give a polynomial-time algorithm for the Hamiltonian cycle 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*


3SAT
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?

Many 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.

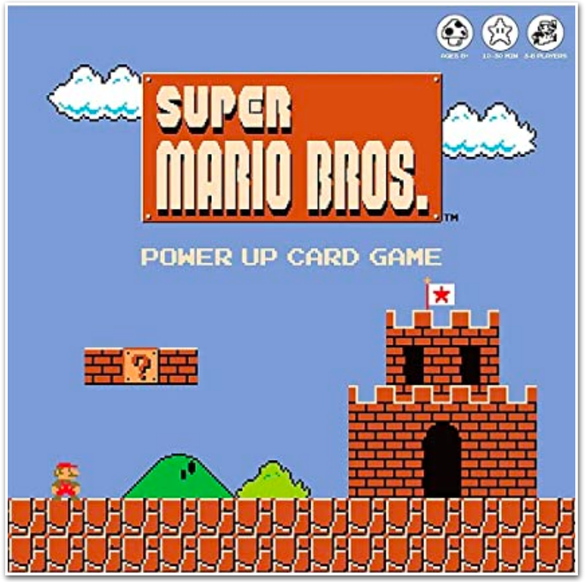
Politics. Shapley–Shubik voting power.

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)
- **RUBIC'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