CS 362, Lecture 23

Jared Saia
University of New Mexico

Today's Outline

- Review
- NP-Hardness and three more reductions

Classes of Problems

We can characterize many problems into three classes:

- **P** is the set of yes/no problems that can be solved in polynomial time. Intuitively P is the set of problems that can be solved "quickly"
- **NP** is the set of yes/no problems with the following property: If the answer is yes, then there is a *proof* of this fact that can be checked in polynomial time
- **co-NP** is the set of yes/no problems with the following property: If the answer is no, then there is a *proof* of this fact that can be checked in polynomial time

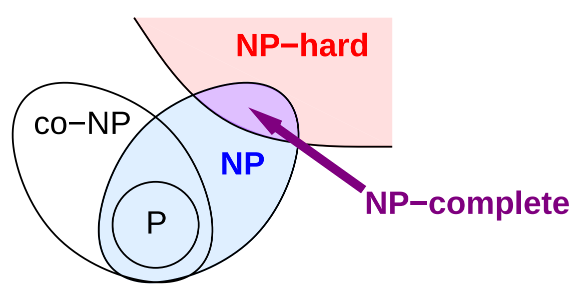
NP-Hard

- A problem Π is **NP-hard** if a polynomial-time algorithm for Π would imply a polynomial-time algorithm for *every problem in NP*
- In other words: **Π is NP-hard iff If Π can be solved in polynomial time, then P=NP**
- In other words: if we can solve one particular NP-hard problem quickly, then we can quickly solve *any* problem whose solution is quick to check (using the solution to that one special problem as a subroutine)
- If you tell your boss that a problem is NP-hard, it's like saying: "Not only can't I find an efficient solution to this problem but neither can all these other very famous people." (you could then seek to find an approximation algorithm for your problem)

NP-Complete

- A problem is *NP-Easy* if it is in NP
- A problem is *NP-Complete* if it is NP-Hard and NP-Easy
- In other words, a problem is NP-Complete if it is in NP but is at least as hard as all other problems in NP.
- If anyone finds a polynomial-time algorithm for even one NP-complete problem, then that would imply a polynomial-time algorithm for *every* NP-Complete problem
- *Thousands* of problems have been shown to be NP-Complete, so a polynomial-time algorithm for one (i.e. all) of them is incredibly unlikely

Example



A detailed picture of what we *think* the world looks like.

Independent Set

- Independent Set is the following problem: "Does there exist a set of k vertices in a graph G with no edges between them?"
- In the hw, you'll show that independent set is NP-Hard by a reduction from CLIQUE
- Thus we can now use Independent Set to show that other problems are NP-Hard

Vertex Cover

- A *vertex cover* of a graph is a set of vertices that touches every edge in the graph
- The problem *Vertex Cover* is: "Does there exist a vertex cover of size k in a graph G?"
- We can prove this problem is NP-Hard by an easy reduction from Independent Set

Key Observation

- Key Observation: If I is an independent set in a graph $G = (V, E)$, then $V - I$ is a vertex cover.
- Thus, there is an independent set of size k iff there is a vertex cover of size $|V| - k$.
- For the reduction, we want to show that a polynomial time algorithm for Vertex Cover can give a polynomial time algorithm for Independent Set

The Reduction

- We are given a graph $G = (V, E)$ and a value k and we must determine if there is an independent set of size k in G.
- To do this, we ask if there is a vertex cover of size $|V| - k$ in G.
- If so then we return that there *is* an independent set of size k in G
- If not, we return that there *is not* an independent set of size k in G

The Reduction

$$\begin{tikzcd} \boxed{\text{graph } G = (V,E),\ k} \arrow[r, "\text{trivial}"] & \boxed{\text{graph } G = (V,E),\ |V| - k} \arrow[d, Rightarrow, "\text{VertexCover}"] \\ \boxed{\text{True or False}} & \boxed{\text{True or False}} \arrow[l, "O(1)"'] \end{tikzcd}$$

Graph Coloring

- A c-coloring of a graph G is a map $C : V \to \{1, 2, \ldots, c\}$ that assigns one of c "colors" to each vertex so that every edge has two different colors at its endpoints
- The graph coloring problem is: "Does there exist a c-coloring for the graph G?"
- Even when $c = 3$, this problem is hard. We call this problem *3Colorable* i.e. "Does there exist a 3-coloring for the graph G?"

3Colorable

- To show that 3Colorable is NP-hard, we will reduce from 3Sat
- This means that we want to show that a polynomial time algorithm for 3Colorable can give a polynomial time algorithm for 3Sat
- Recall that the 3-SAT problem is just: "Is there any assignment of variables to a 3CNF formula that makes the formula evaluate to true?"
- And a 3CNF formula is just a conjunct of a bunch of clauses, each of which contains exactly 3 variables e.g.

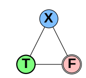
$$\overbrace{(a \vee b \vee c)}^{\text{clause}} \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee d)$$

Reduction

- We are given a 3-CNF formula, F, and we must determine if it has a satisfying assignment
- To do this, we produce a graph as follows
- The graph contains one *truth* gadget, one *variable* gadget for each variable in the formula, and one *clause* gadget for each clause in the formula

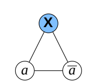
The Truth Gadget

- The truth gadget is just a triangle with three vertices T, F and X, which intuitively stand for **True**, **False**, and **other**
- Since these vertices are all connected, they must have different colors in any 3-coloring
- For the sake of convenience, we will name those colors **True**, **False**, and **Other**
- Thus when we say a node is colored "True", we just mean that it's colored the same color as the node T



The Variable Gadgets

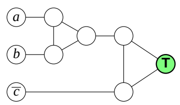
- The variable gadget for a variable a is also a triangle joining two new nodes labeled a and $\overline{a}$ to node X in the truth gadget
- Node a must be colored either "True" or "False", and so node $\overline{a}$ must be colored either "False" or "True", respectively.



- The variable gadget ensures that each of the literals is colored either "True" or "False"

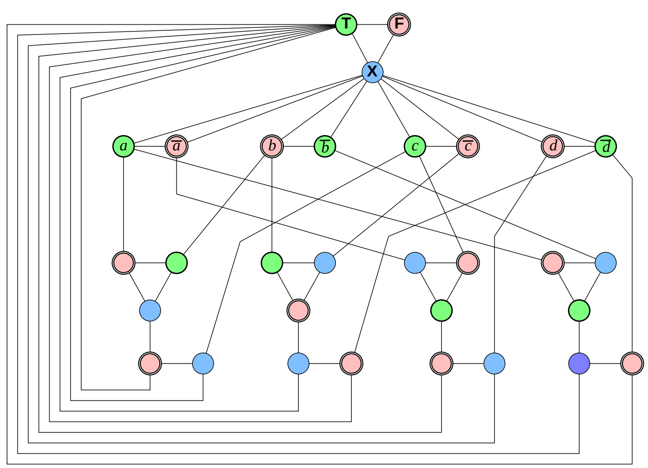
The Clause Gadgets

- Each clause gadget joins three literal nodes to node T in the truth gadget using five new unlabelled nodes and ten edges (as in the figure)
- This clause gadget ensures that at least one of the three literal nodes in each clause is colored "True"



Example

Consider the formula $(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$.
Following is the graph created by the reduction:



Example

- Note that the 3-coloring of this example graph corresponds to a satisfying assignment of the formula
- Namely, $a = c =$ True, $b = d =$ False.
- Note that the final graph contains only *one* node T, only *one* node F, only *one* node $\bar{a}$ for each variable a and so on

Correctness

- The proof of correctness for this reduction is direct
- If the graph is 3-colorable, then we can extract a satisfying assignment from any 3-coloring, since at least one of the three literal nodes in every clause gadget is colored "True"
- Conversely, if the formula is satisfiable, then we can color the graph according to any satisfying assignment

Reduction Picture

$$
\begin{tikzcd}
\boxed{\text{3CNF formula}} \arrow[r, "O(n)"] & \boxed{\text{graph}} \arrow[d, Rightarrow, "\text{3Colorable}"] \\
\boxed{\text{True or False}} & \boxed{\text{True or False}} \arrow[l, "\text{trivial}"']
\end{tikzcd}
$$

Wrap Up

- We've just shown that if 3Colorable can be solved in polynomial time then 3-SAT can be solved in polynomial time
- This shows that 3Colorable is NP-Hard
- To show that 3Colorable is in NP, we just need to note that we can easily verify that a graph has been correctly 3-colored in linear time: just compare the endpoints of every edge
- Thus, 3Coloring is NP-Complete.
- This implies that the more general graph coloring problem is also NP-Complete

In-Class Exercise

Consider the problem *4Colorable*: "Does there exist a 4-coloring for a graph G?"

- Q1: Show this problem is in NP by showing that there exists an efficiently verifiable proof of the fact that a graph is 4 colorable.
- Q2: Show the problem is NP-Hard by a reduction from the problem 3Colorable. In particular, show the following:
  - Given a graph G, you can create a graph G' such that G' is 4-colorable iff G is 3-colorable.
  - Creating G' from G takes polynomial time

Note: You've now shown that 4Colorable is NP-Complete!

Hamiltonian Cycle

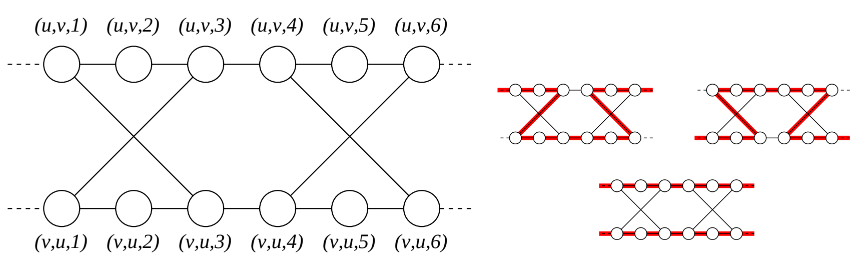
- A *Hamiltonian Cycle* in a graph is a cycle that visits every vertex exactly once (note that this is very different from an *Eulerian cycle* which visits every *edge* exactly once)
- The Hamiltonian Cycle problem is to determine if a given graph G has a Hamiltonian Cycle
- We will show that this problem is NP-Hard by a reduction from the vertex cover problem.

The Reduction

- To do the reduction, we need to show that we can solve Vertex Cover in polynomial time if we have a polynomial time solution to Hamiltonian Cycle.
- Given a graph G and an integer k, we will create another graph G' such that G' has a Hamiltonian cycle iff G has a vertex cover of size k
- As for the last reduction, our transformation will consist of putting together several "gadgets"

Edge Gadget and Cover Vertices

- For each edge (u, v) in G, we have an *edge gadget* in G' consisting of twelve vertices and fourteen edges, as shown below



An edge gadget for (u, v) and the only possible Hamiltonian paths through it.

Edge Gadget

- The four corner vertices $(u, v, 1)$, $(u, v, 6)$, $(v, u, 1)$, and $(v, u, 6)$ each have an edge leaving the gadget
- A Hamiltonian cycle can only pass through an edge gadget in one of the three ways shown in the figure
- These paths through the edge gadget will correspond to one or both of the vertices u and v being in the vertex cover.

Cover Vertices

- G' also contains k *cover vertices*, simply numbered 1 through k

Vertex Chains

- For each vertex u in G, we string together all the edge gadgets for edges (u, v) into a single *vertex chain* and then connect the ends of the chain to all the cover vertices
- Specifically, suppose u has d neighbors $v_1, v_2, \ldots, v_d$. Then G' has the following edges:
  - $d - 1$ edges between $(u, v_i, 6)$ and $(u, v_{i+1}, 1)$ (for all i between 1 and $d - 1$)
  - k edges between the cover vertices and $(u, v_1, 1)$
  - k edges between the cover vertices and $(u, v_d, 6)$

The Reduction

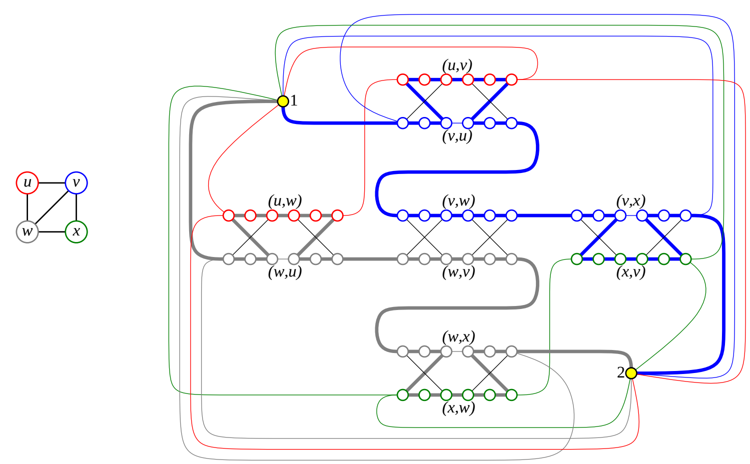
- It's not hard to prove that if $\{v_1, v_2, \ldots, v_k\}$ is a vertex cover of G, then G' has a Hamiltonian cycle
- To get this Hamiltonian cycle, we start at cover vertex 1, traverse through the vertex chain for v_1, then visit cover vertex 2, then traverse the vertex chain for v_2 and so forth, until we eventually return to cover vertex 1
- Conversely, one can prove that any Hamiltonian cycle in G' alternates between cover vertices and vertex chains, and that the vertex chains correspond to the k vertices in a vertex cover of G

Thus, G has a vertex cover of size k iff G' has a Hamiltonian cycle

The Reduction

- The transformation from G to G' takes at most $O(|V|^2)$ time, so the Hamiltonian cycle problem is NP-Hard
- Moreover we can easily verify a Hamiltonian cycle in linear time, thus Hamiltonian cycle is also in NP
- Thus Hamiltonian Cycle is NP-Complete

Example



The original graph G with vertex cover $\{v, w\}$, and the transformed graph G' with a corresponding Hamiltonian cycle (bold edges).
Vertex chains are colored to match their corresponding vertices.

The Reduction

$$\begin{array}{ccc} \boxed{\text{graph } G=(V,E),\ k} & \xrightarrow{O(|V|^2)} & \boxed{\text{graph } G'} \\ & & \Big\Downarrow \text{Hamiltonian Cycle} \\ \boxed{\text{True or False}} & \xleftarrow{O(1)} & \boxed{\text{True or False}} \end{array}$$

Traveling Sales Person

- A problem closely related to Hamiltonian cycles is the famous *Traveling Salesperson Problem(TSP)*
- The TSP problem is: "Given a weighted graph G, find the shortest cycle that visits every vertex.
- Finding the shortest cycle is obviously harder than determining if a cycle exists at all, so since Hamiltonian Path is NP-hard, TSP is also NP-hard!

NP-Hard Games

- In 1999, Richard Kaye proved that the solitaire game Minesweeper is NP-Hard, using a reduction from Circuit Satifiability.
- Also in the last few years, Eric Demaine, et. al., proved that the game Tetris is NP-Hard

Challenge Problem

- Consider the *optimization* version of, say, the graph coloring problem: "Given a graph G, what is the smallest number of colors needed to color the graph?" (Note that unlike the *decision* version of this problem, this is not a yes/no question)
- Show that the optimization version of graph coloring is also NP-Hard by a reduction from the decision version of graph coloring.
- Is the optimization version of graph coloring also NP-Complete?

Challenge Problem

- Consider the problem 4Sat which is: "Is there any assignment of variables to a 4CNF formula that makes the formula evaluate to true?"
- Is this problem NP-Hard? If so, give a reduction from 3Sat that shows this. If not, give a polynomial time algorithm which solves it.

Challenge Problem

- Consider the following problem: "Does there exist a clique of size 5 in some input graph G?"
- Is this problem NP-Hard? If so, prove it by giving a reduction from some known NP-Hard problem. If not, give a polynomial time algorithm which solves it.