CS 6363.005.19S Lecture 23—April 23, 2019

Main topics are #NP-hardness.

Prelude

- Homework 5 is due Tuesday April 30th.
- I'm holding office hours today from 2pm to 3pm.

Vertex Cover

- We're doing more NP-hardness proofs today. Here's our list of known problems so far…
  - CircuitSAT
  - SAT
  - 3SAT (so why would you ever reduce from SAT?)
  - MaxIndSet
  - MaxClique
- A *vertex cover* is a set of vertices that touch every edge in the graph. *MinVertexCover* asks for the size of the smallest vertex cover in the graph.

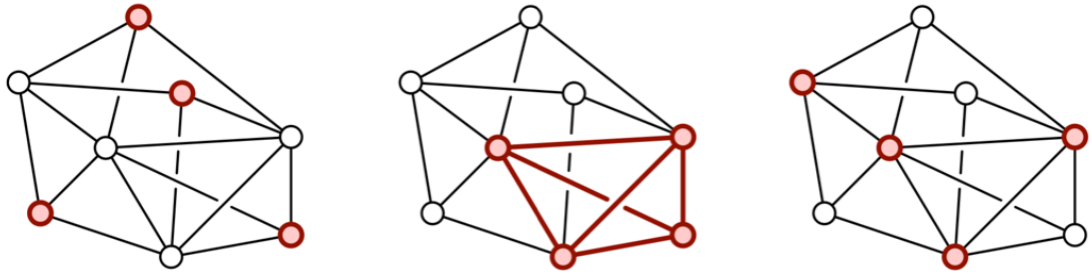
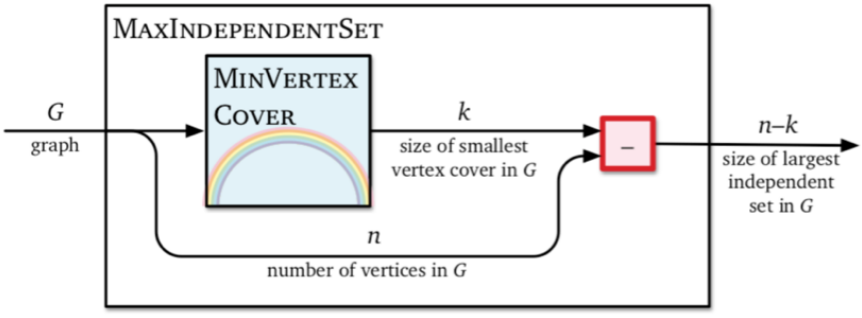


Figure 12.9. A graph whose largest independent set, largest clique, and smallest vertex cover all have size 4.

- Claim: MinVertexCover is NP-hard.
- Observe that I is an independent set in G = (V, E) if and only if V \ I is a vertex cover. So the largest independent set in G is the complement of the smallest vertex cover. If the smallest vertex cover has size k, the largest independent set has size n - k.

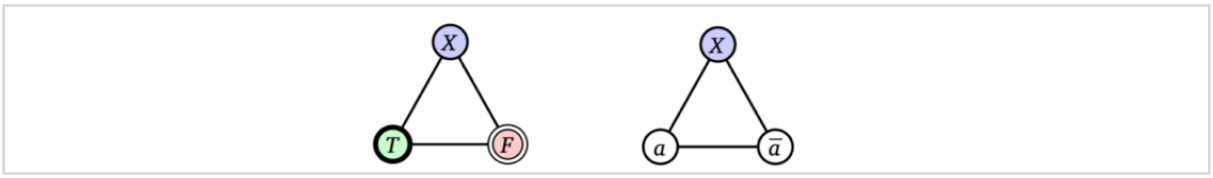


- One thing to point out, the decision version of these optimization problems asks if you can find a set of interest of size k where k is part of the input. That's what we're really asking for

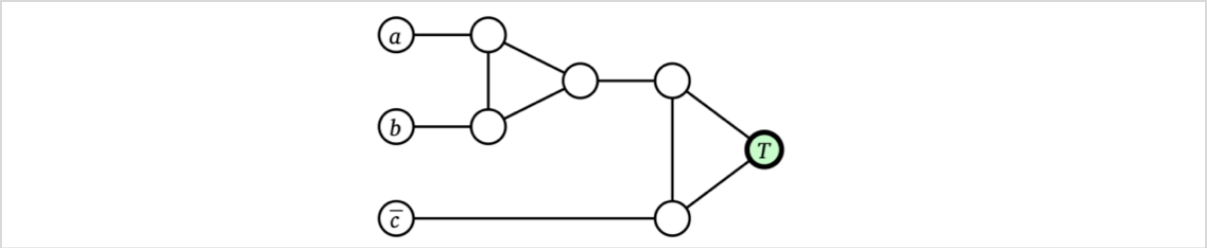
in these reductions, so the decision problems are NP-hard as well.

Graph Coloring

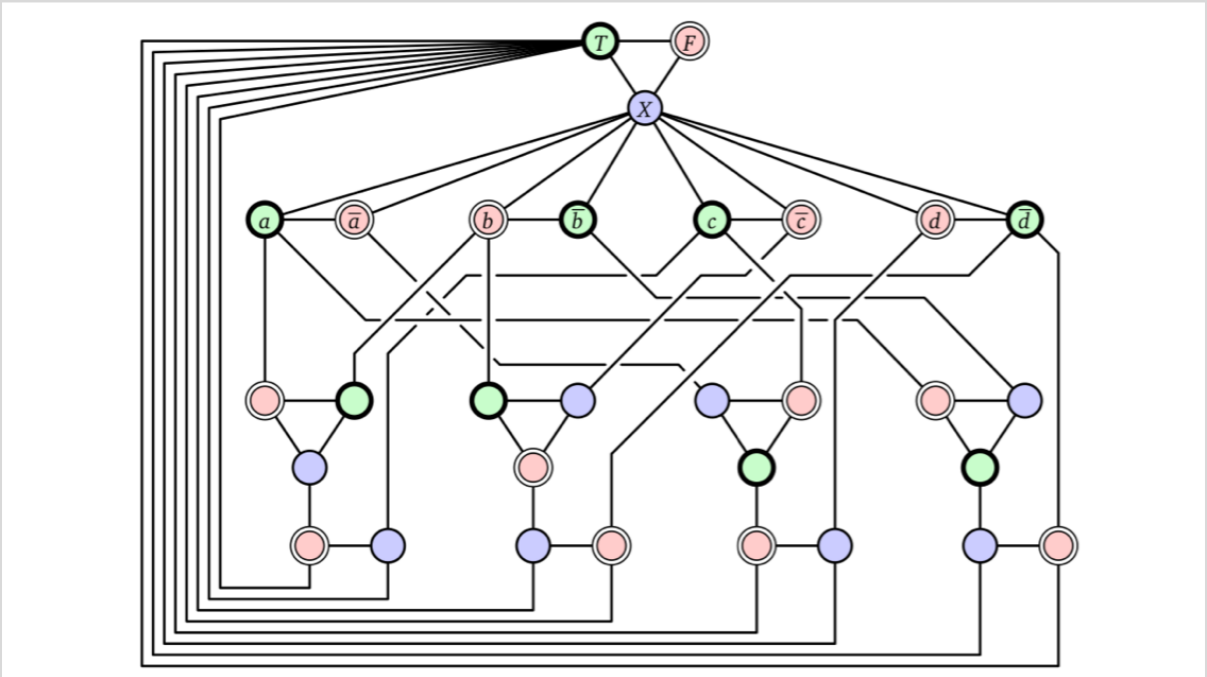
- Let's look at yet another graph problem that's a bit different.
- A *proper k-coloring* of G = (V, E) is a function C : V ➜ {1, 2, …, k} assigning one of k "colors" to each vertex so each edge has distinct colors at its endpoints.
- The graph coloring problem is to find the smallest possible number of colors to get a proper k-coloring.
- It's directly used for certain applications like compiler design. Can I store all my local variables for this function using only a few registers?
- 3Color is the "easier" problem where we simply ask, given a graph, does it have a 3-coloring?
- Claim: 3Color is NP-complete.
- It's in NP, because you can just tell me the colors and I can verify its a proper coloring in polynomial time.
- We'll do a reduction from 3SAT. If all else fails, we can maybe use 3SAT.
- Suppose we're given a 3CNF formula Phi. Like many reductions, we'll build a graph G by combining together a collection of useful subgraphs called *gadgets*. There are three types for this reduction:
  - A *truth gadget*: A triangle with vertices T, F, and X standing for True, False, and Other. These three vertices have to have different colors in a proper 3-coloring. For convenience, we'll refer to the colors they get as True, False, and Other respectively.
  - For each variable a, a *variable gadget*: a triangle joining two nodes a and ~a to *the node* X used in the truth gadget. Node a must be colored True or False in proper 3-coloring, implying ~a gets False or True, respectively.



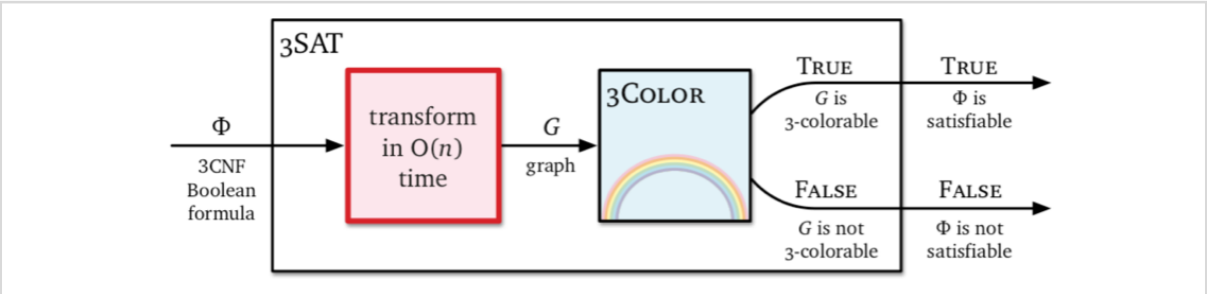
  - For each clause in Phi, a *clause gadget*: We join the three literal nodes for the clause to node T using five new nodes and ten new edges. If all three literals are colored False, then we'll have a monochromatic edge in the clause gadget when using three colors. However, if at least one literal is colored True, we can get a proper coloring of the clause gadget. The proof of this claim is just a tedious examination of all the cases.



- So here's the whole graph for our earlier formula (a ∨ b ∨ c) ∧ (b ∨ ¯c ∨ d¯) ∧ (a¯ ∨ c ∨ d) ∧ (a ∨ ¯b ∨ d¯).



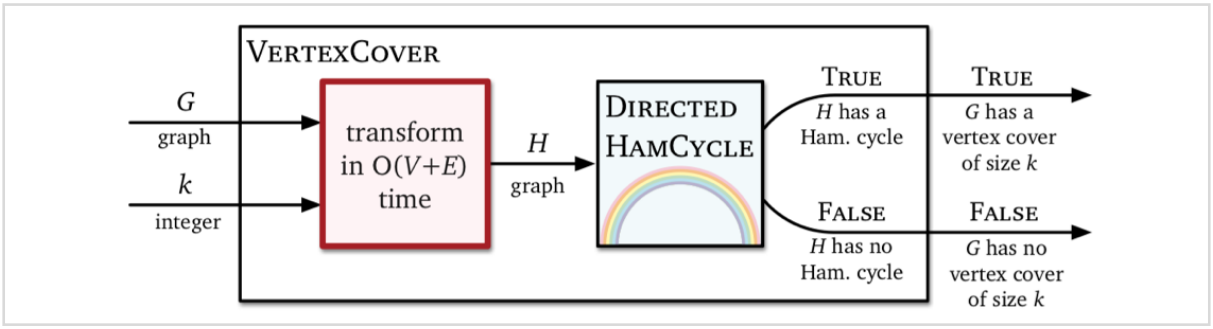
- So now I claim the graph is 3-colorable if and only if Phi is satisfiable.
  - If the graph is 3-colorable, exactly one of a or ~a is assigned True for each variable. And as I already argued, at least one literal per clause is assigned true so we can use the True literals to satisfy Phi.
  - If Phi is satisfiable, then we can use the literal assignments to get their colors and we can assign colors to the rest of the clause gadget vertices to properly color the whole graph.
- Here's the whole algorithm:



- 3Color is NP-hard since we got a reduction, so it's NP-complete.
- And the more general optimization version of graph coloring "how many colors do I need" must be NP-hard as well, since it naturally solves 3Color.

Hamiltonian Cycle

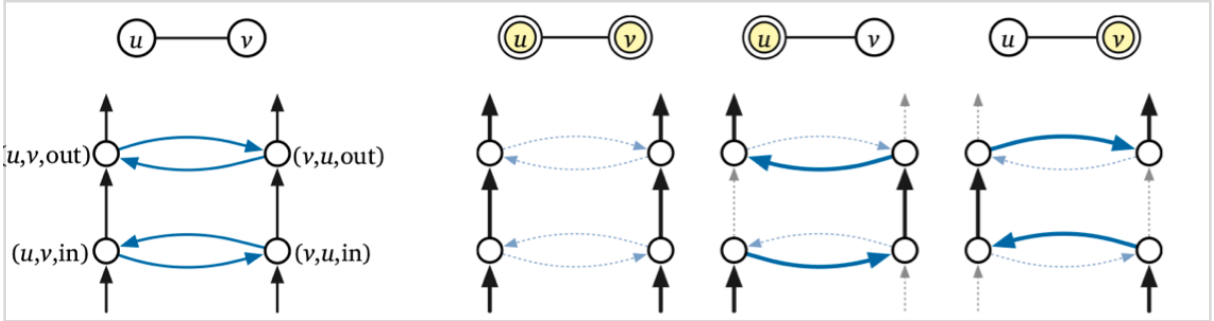
- A Hamiltonian cycle is a cycle in a graph that visits every *vertex* exactly once. (Visiting every edge once is an Eulerian tour.)
- The Hamiltonian cycle problem is given a directed graph H = (V, E), does H contain a Hamiltonian cycle?
- Claim: Hamiltonian cycle is NP-complete.
- Again, I can tell you the order of vertices in the cycle so its in NP.
- We'll reduce from the decision version of VertexCover: is there a vertex cover of size k?
- I'll be a bit lighter on the details here so we can get to one last example.
- Let's start with the box algorithm this time. We want to know if undirected graph G has a vertex cover of size k, so we'll build a directed graph H and ask if it has a Hamiltonian cycle.



- Now let's build some gadgets.
- Each edge uv in G becomes four vertices (u, v, in), (u, v, out), (v, u, in), and (v, u, out) in H plus six directed edges

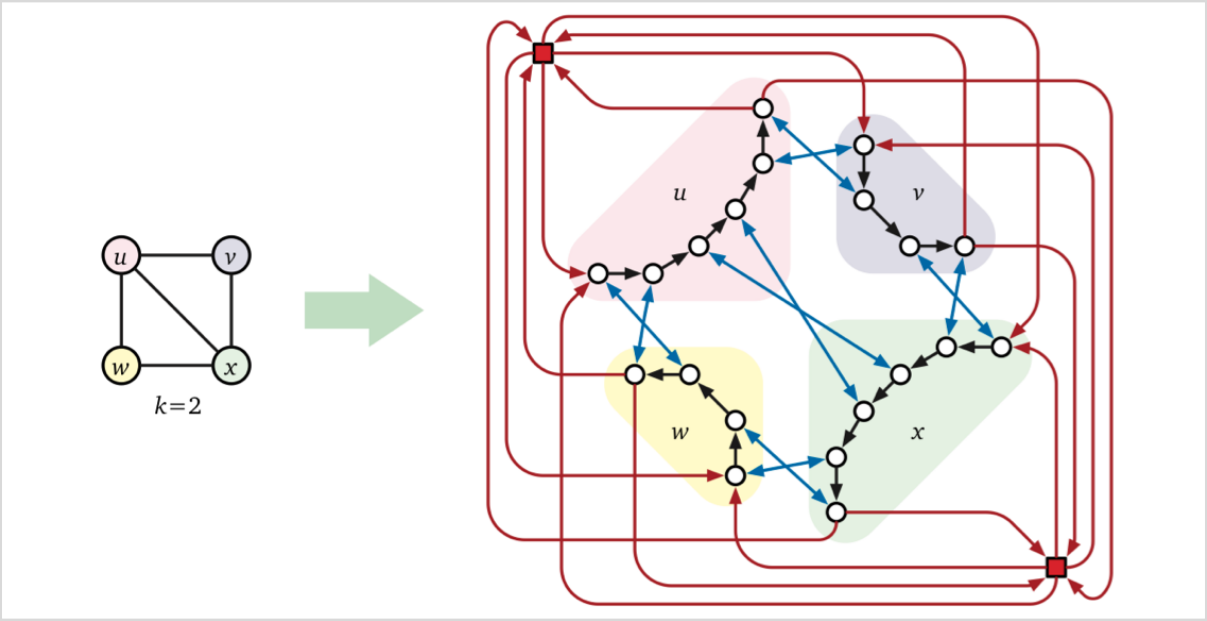
$$(u,v,\text{in})\to(u,v,\text{out}) \quad (u,v,\text{in})\to(v,u,\text{in}) \quad (v,u,\text{in})\to(u,v,\text{in})$$
$$(v,u,\text{in})\to(v,u,\text{out}) \quad (u,v,\text{out})\to(v,u,\text{out}) \quad (v,u,\text{out})\to(u,v,\text{out})$$

- Eventually, we'll have one edge going out from each out vertex and one edge going in to each in vertex as shown below. There's exactly three routes a Hamiltonian cycle can take through the four vertices. Each corresponds to a choice of u and v, only u, or only v being used in a vertex cover of G.



- For each vertex u in G, we connect all the edge gadgets for edges uv in a directed path called the *vertex chain*. Specifically, if u has d neighbors v_1, v_2, …, v_d, we add d - 1 edges (u, v_i, out) ➜ (u, v_{i+1}, in) for each i from 1 to d - 1.

- And we add k additional *cover vertices* x_1, x_2, …, x_k. Each has an edge to the first in vertex in each chain and an edge from the last out vertex in each chain. Here's an example:



- Now, suppose there is a vertex cover u_1, u_2, …, u_k in G. We can find a Hamiltonian cycle in H. For each i from 1 to k, we go from x_i, through u_i's chain, and to x_{i+1}. We go through each of u_i's edge gadgets as described above, detouring through the (v, u_i) vertices for any edge uv where v is not in the cover.
- Conversely, suppose there is a Hamiltonian cycle. It must contain an edge from every cover vertex to the start of some vertex chain. If you start at some edge (u, v, in) you must leave (u, v, out), so we'll touch every edge gadget in the chain. Every edge gadget for uv must be entered in at least one of its two entrances (u, v, in) or (v, u, in), so we'll go through enough vertex chains to cover all edges in G. And we go through exactly k vertex chains, so we have a vertex cover of size k.
- There are many variants of HamiltonianCycle. HamiltonianPath asks if there is a path containing each vertex exactly once. It's also NP-hard. They problems remain NP-hard in undirected graphs as well.

Subset Sum

- Let's finish with one last problem that doesn't involve booleans or graphs.
- SubsetSum: Given a set X of positive integers and an integer T, does X have a subset whose elements sum to T?
- We'll again reduce from VertexCover. Suppose we have a graph G = (V, E) and an integer k. Is there a vertex cover of size k?
- We need to make a set X of integers and a target value T so there's a subset sum of T if and only if there's a vertex cover of size k.
- We're still going to use gadgets, but now the gadgets will be very large numbers.
- Number the edges of G from 0 to E - 1. X contains b_i := 4^i for each edge i.

- For each vertex v, it also contains

$$a_v := 4^E + \sum_{i \in \Delta(v)} 4^i$$

where Delta(v) is the set of edges incident to v.

- We can also think of each integer as a (E + 1)-digit number written in base 4. The Eth digit is 1 if the integer represents a vertex and is 0 otherwise.
- For each i < E, the ith digit is 1 if the integer represents edge i or one of its endpoints, and is 0 otherwise.
- Finally, set

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

- It only takes polynomial time to write out these numbers in base 4 or even binary.
- Here's the SubsetSum instance for that graph we were using for HamiltonianCycle.

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

- Now, suppose there is a vertex cover C of size k.
  - Let X_C be the subset of integers that contains a_v for every vertex v in C and b_i for every edge i covered *exactly once*.
  - The sum of the integers, written in base 4, has a 2 in each of the first E digits.
  - And we're summing k 4^E's for our k vertices.
  - So the sum is exactly T.
- Suppose there is a subset X' of X that sums to T.
  - Let V' be the subset of vertices whose vertex numbers we chose.
  - There are no carries in the first E digits, because for each i there are only three numbers whose ith digit is 1. Each edge number b_i contributes a single 1 to the ith digit, so we need one of the vertex numbers for its endpoints. In other words, each edge is covered by V'.
  - And every vertex number is at least 4^m, so we can only afford to use k of them. |V'| ≤ k.
- Now, if you were reading carefully, you might have noticed there's an O(nT) time algorithm for SubsetSum earlier in Erickson. Doesn't that imply P = NP?
- That's an example of a *pseudo-polynomial time* algorithm. It's running time is exponential in the actual input size, because we only used O(log T) bits to write out these numbers.
- NP-hard problems with such algorithms are called *weakly NP-hard*. If the problem is still

NP-hard even after writing down numbers in unary (so in space proportional to their value), then the problem is *strongly NP-hard*.

- And that's it! On Thursday we'll do one last review day, and next week you can do some QE practice if you'd like.