CS 6363.005.19S Lecture 22—April 18, 2019

Main topics are #NP-hardness.

Prelude

- Homework 5 is due Tuesday April 30th.
- Here's the schedule for the rest of the semester:
  - Today and next Tuesday the 23rd, we'll continue to look at NP-hardness with more details on how to prove NP-hardness and lots more examples of NP-hard problems.
  - Thursday the 25th will be another review day where we can discuss EVERYTHING except Homework 5 of course.
  - Like I said earlier in the semester, I have to attend a workshop the last week. We'll use this as an opportunity to get special preparation for the QE. I'll draft some practice exams that are half the length of the real one for you to take in class. The first will cover divide-and-conquer and dynamic programming. The second will cover graphs and NP-hardness. Half an exam in half the time is more difficult than a full exam in full time, but it should give you some good practice I hope.
  - I can't promise I'll have time to grade anything, but please come by office hours afterward to discuss your solutions and work on any other practice you desire.

NP-hardness

- Last time, we discussed P, NP, and NP-hardness.
- In short, P is the set of polynomial time solvable decision problems. NP are those that you can verify the answer is yes in polynomial time, assuming somebody gives you a proof.
- Finally, a problem is NP-hard if there is a polynomial time reduction *from* any problem in NP *to* the NP-hard problem.
- Equivalently, a polynomial time algorithm for any NP-hard problem would imply P = NP.
- A problem is NP-complete if it is in NP and is NP-hard.
- We saw three NP-complete problems last time.
  - CircuitSAT: Given a boolean circuit, is there a set of inputs to make it output true?
  - SAT: Given a boolean formula, is there a setting of the variables to make it true?

3SAT

- Let's look at another NP-complete problem. This one is a special case of SAT called 3SAT or sometimes 3CNF-SAT.
- First, some definitions you may have seen.
- A *literal* is a boolean variable or its negation (a or not(a)).

- A *clause* is a disjunction (OR) of several literals (b or not(c) or not(d))
- A boolean formula is in *conjunctive normal form* (CNF) if it is a conjunction (AND) of several clauses.

$$\overbrace{(a \lor b \lor c \lor d)}^{\text{clause}} \land (b \lor \bar{c} \lor \bar{d}) \land (\bar{a} \lor c \lor d) \land (a \lor \bar{b})$$

- A 3CNF formula is a CNF formula with exactly three literals per clause. So this example is not a 3CNF formula since the first and last clauses have the wrong number of literals.
- 3SAT: Given a 3CNF formula, is there an assignment of the variables that makes the formula evaluate to True?
- 3SAT looks like it should be easier than general SAT since I'm restricting what types of inputs you get, but it turns out the problem is still NP-hard.
- Remember: To prove NP-hardness, you need to reduce *from* a known NP-hard problem *to* your new problem.
- We'll use a reduction directly from CircuitSAT to show 3SAT is NP-hard. This should be the last time we reduce directly from CircuitSAT.
- Given a boolean circuit:
    1. Change it so every AND and OR gate has only two inputs. If a gate has k > 2 inputs, replace it with a binary tree of k - 1 two-input gates.
    2. Write down the circuit as a formula with one clause per gate. Just like in the reduction to SAT.
    3. Change every gate clause into a CNF formula.

$$\begin{aligned} a = b \land c &\quad\longmapsto\quad (a \lor \bar{b} \lor \bar{c}) \land (\bar{a} \lor b) \land (\bar{a} \lor c) \\ a = b \lor c &\quad\longmapsto\quad (\bar{a} \lor b \lor c) \land (a \lor \bar{b}) \land (a \lor \bar{c}) \\ a = \bar{b} &\quad\longmapsto\quad (a \lor b) \land (\bar{a} \lor \bar{b}) \end{aligned}$$

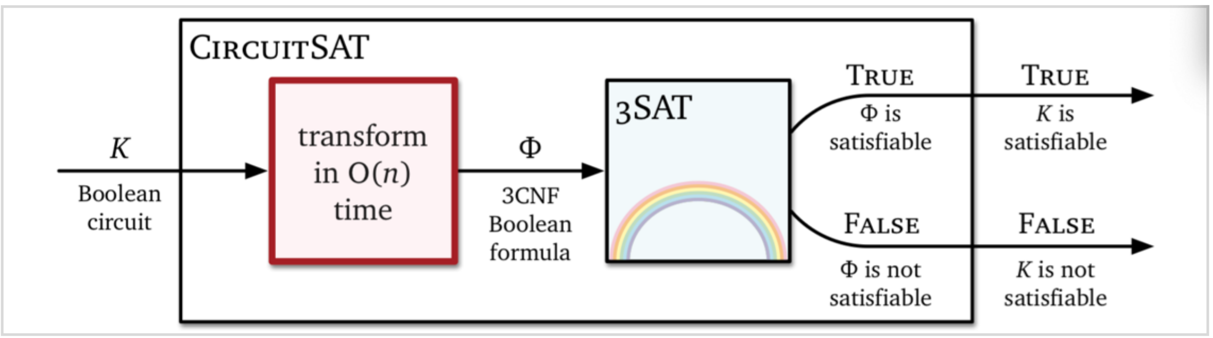
    4. Make sure every clause has exactly three literals by introducing new literals for every one and two-literal clause and expanding them into new clauses.

$$\begin{aligned} a \lor b &\longmapsto (a \lor b \lor x) \land (a \lor b \lor \bar{x}) \\ a &\longmapsto (a \lor x \lor y) \land (a \lor \bar{x} \lor y) \land (a \lor x \lor \bar{y}) \land (a \lor \bar{x} \lor \bar{y}) \end{aligned}$$

- Here's the 3CNF formula you get from our favorite example circuit from last week.

$$\begin{gathered}
(y_1 \vee \overline{x_1} \vee \overline{x_4}) \wedge (\overline{y_1} \vee x_1 \vee z_1) \wedge (\overline{y_1} \vee x_1 \vee \overline{z_1}) \wedge (\overline{y_1} \vee x_4 \vee z_2) \wedge (\overline{y_1} \vee x_4 \vee \overline{z_2}) \\
\wedge (y_2 \vee x_4 \vee z_3) \wedge (y_2 \vee x_4 \vee \overline{z_3}) \wedge (\overline{y_2} \vee \overline{x_4} \vee z_4) \wedge (\overline{y_2} \vee \overline{x_4} \vee \overline{z_4}) \\
\wedge (y_3 \vee \overline{x_3} \vee \overline{y_2}) \wedge (\overline{y_3} \vee x_3 \vee z_5) \wedge (\overline{y_3} \vee x_3 \vee \overline{z_5}) \wedge (\overline{y_3} \vee y_2 \vee z_6) \wedge (\overline{y_3} \vee y_2 \vee \overline{z_6}) \\
\wedge (\overline{y_4} \vee y_1 \vee x_2) \wedge (y_4 \vee \overline{x_2} \vee z_7) \wedge (y_4 \vee \overline{x_2} \vee \overline{z_7}) \wedge (y_4 \vee \overline{y_1} \vee z_8) \wedge (y_4 \vee \overline{y_1} \vee \overline{z_8}) \\
\wedge (y_5 \vee x_2 \vee z_9) \wedge (y_5 \vee x_2 \vee \overline{z_9}) \wedge (\overline{y_5} \vee \overline{x_2} \vee z_{10}) \wedge (\overline{y_5} \vee \overline{x_2} \vee \overline{z_{10}}) \\
\wedge (y_6 \vee x_5 \vee z_{11}) \wedge (y_6 \vee x_5 \vee \overline{z_{11}}) \wedge (\overline{y_6} \vee \overline{x_5} \vee z_{12}) \wedge (\overline{y_6} \vee \overline{x_5} \vee \overline{z_{12}}) \\
\wedge (\overline{y_7} \vee y_3 \vee y_5) \wedge (y_7 \vee \overline{y_3} \vee z_{13}) \wedge (y_7 \vee \overline{y_3} \vee \overline{z_{13}}) \wedge (y_7 \vee \overline{y_5} \vee z_{14}) \wedge (y_7 \vee \overline{y_5} \vee \overline{z_{14}}) \\
\wedge (y_8 \vee \overline{y_4} \vee \overline{y_7}) \wedge (\overline{y_8} \vee y_4 \vee z_{15}) \wedge (\overline{y_8} \vee y_4 \vee \overline{z_{15}}) \wedge (\overline{y_8} \vee y_7 \vee z_{16}) \wedge (\overline{y_8} \vee y_7 \vee \overline{z_{16}}) \\
\wedge (y_9 \vee \overline{y_8} \vee \overline{y_6}) \wedge (\overline{y_9} \vee y_8 \vee z_{17}) \wedge (\overline{y_9} \vee y_6 \vee z_{18}) \wedge (\overline{y_9} \vee y_6 \vee \overline{z_{18}}) \wedge (\overline{y_9} \vee y_8 \vee \overline{z_{17}}) \\
\wedge (y_9 \vee z_{19} \vee z_{20}) \wedge (y_9 \vee \overline{z_{19}} \vee z_{20}) \wedge (y_9 \vee z_{19} \vee \overline{z_{20}}) \wedge (y_9 \vee \overline{z_{19}} \vee \overline{z_{20}})
\end{gathered}$$

- Yeah, that's gross, but it's only a constant factor larger than the original circuit, and you can compute it in polynomial time.
- If we have a satisfying input for the circuit, it immediately corresponds to a satisfying assignment for the formula. Any satisfying assignment for the formula can be translated into a satisfying input for the circuit. In other words, the circuit is satisfiable if and only if the formula is.
- In summary, here is what our reduction looked like:



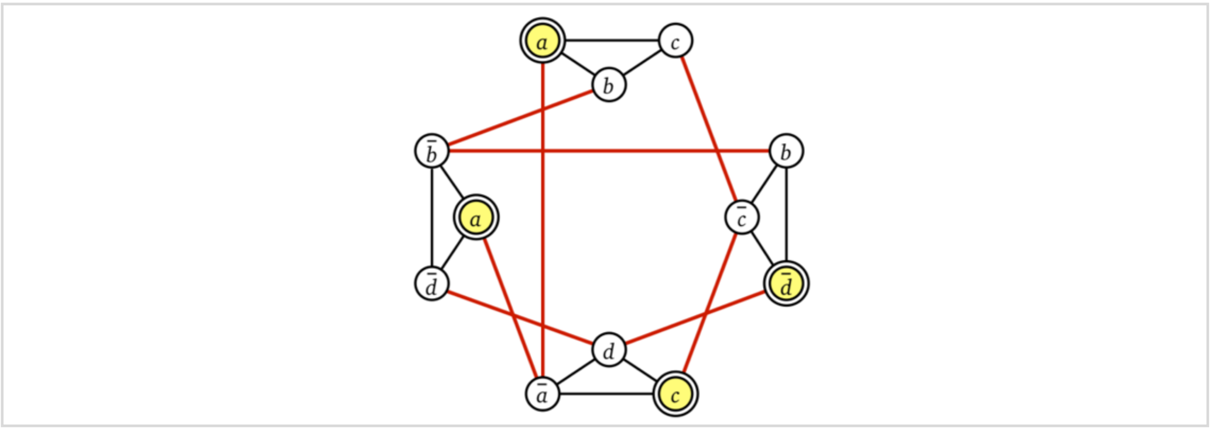
- So a polynomial time algorithm for 3SAT gives a polynomial time algorithm for CircuitSAT and therefore any problem in NP. 3SAT is NP-hard.
- It is also in NP, so 3SAT is NP-complete.

Maximum Independent Set

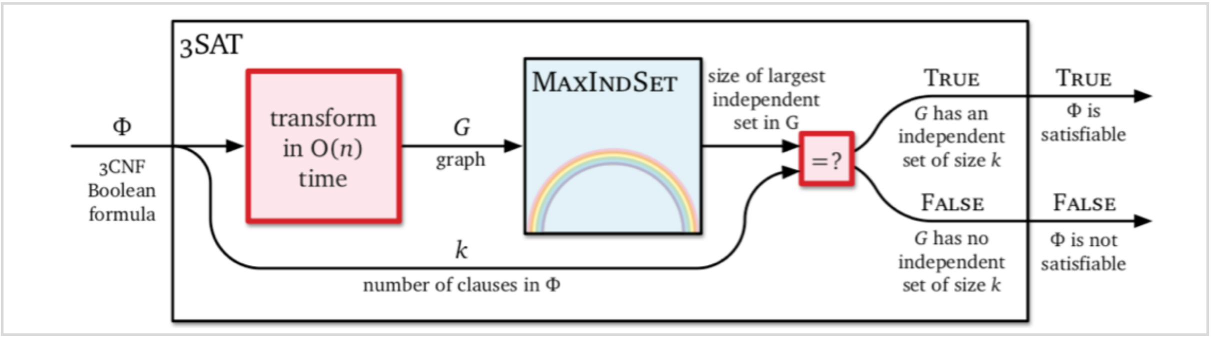
- But not every NP-hard problem is based on circuits.
- Suppose we're given a simple, unweighted graph G.
- An *independent set* in G is a subset of vertices with no edges between them.
- The *maximum independent set* problem (MaxIndSet) asks for the largest independent set in the graph.
- Claim: MaxIndSet is NP-hard.
- We'll do a reduction *from* 3SAT.
- Suppose we're given a 3CNF formula Phi. Let k be the number of clauses in Phi.
- We'll make a graph G with 3k vertices, one for each literal in Phi.
- Any two literals in the same clause get a "triangle" edge. Also, any two literals representing

a variable and its inverse get a "negation" edge.

- For example, $(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$ becomes



- I claim G contains an independent set of size exactly k if and only if Phi is satisfiable.
    - If Phi is satisfiable, fix a satisfying assignment. Each clause contains at least one true literal, so arbitrarily choose one per clause to make vertex set S. There's one per clause so no triangle edge has both sides chosen. And we only chose True literals, so no negation edge has both sides chosen. So S is an independent set of size k.
    - Suppose there is an independent set S of size k. Make each chosen literal True and assign arbitrary values to variables that weren't represented by S. Any independent set contains at most one vertex per clause triangle, so we chose one true literal per clause. And there are no contradictions, because of the negation edges.
- The transformation itself takes O(n) time, so its a polynomial time reduction.
- Here's our overall algorithm: Do the transformation, and return True if and only if the maximum independent set has size k.



- So to solve any problem in NP, we can reduce to CircuitSAT and then reduce to 3SAT and then reduce to MaxIndSet, so MaxIndSet is NP-hard. In other words, a polynomial time algorithm for MaxIndSet implies P = NP, so there probably isn't one!

The General Pattern

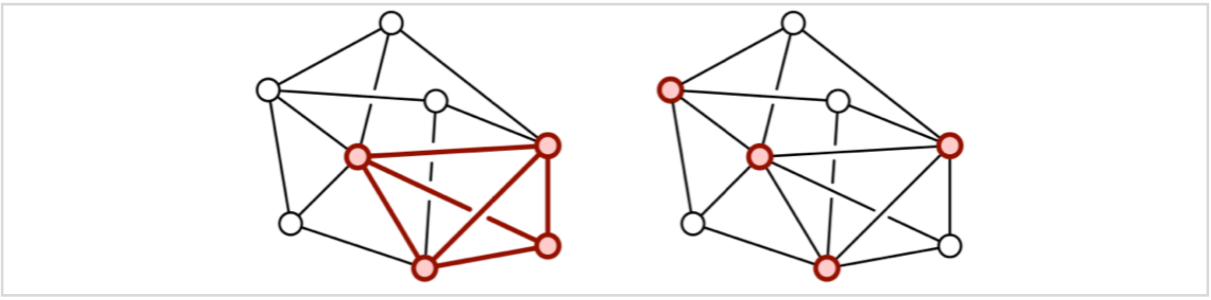
- All NP-hardness proofs follow the same general pattern. To prove problem Y is NP-hard, we reduce NP-hard problem X to problem Y. We usually need to do three things:
    1. Describe a polynomial time algorithm to transform an *arbitrary* instance x of X to a *special* instance y of Y.

2. Prove that if x is a "good" instance of X, then y is a "good" instance of Y.
3. Prove that if y is a "good" instance of Y, then x is a "good" instance of X.

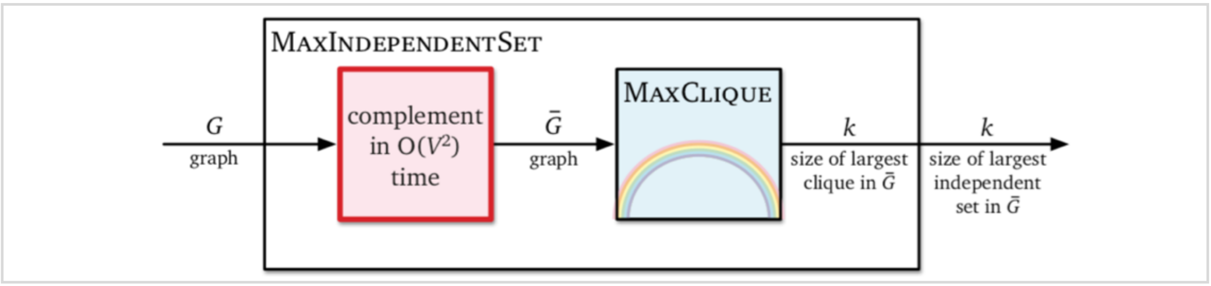
- You have to do both directions for the proof!
- And you normally think about all three things at once so they're all coherent.
- To address one point of possible confusion: We only have to do the reduction itself in one direction, from X to Y. But the correctness proof itself it in both directions, between the *instances* of x and y.
- It may help to think about writing these proofs as algorithms themselves. Being in NP means you have some kind of *certificate* proving the answer is Yes or True. e.g., what inputs to the circuits, what settings of the variables, which vertices to make a large independent set.
- Step 1 above is an algorithm to transform instances of X to instances of Y in polynomial time.
- Step 2 is transforming an arbitrary certificate for x to a certificate for y.
- Step 3 is transforming an arbitrary certificate for y to a certificate for x.
- For example, in CircuitSAT to 3SAT:
    - We transformed an *arbitrary* boolean circuit into a special 3CNF formula in polynomial time.
    - We can transform a satisfying input to the circuit into a satisfying assignment to the formula by tracing through the circuit, transferring values from each wire to the corresponding variable.
    - We can transform a satisfying assignment to a satisfying input by transferring values from the variables to the wires.
- For 3SAT to MaxIndSet:
    - We transformed an arbitrary 3CNF formula into a graph and a specific integer k in polynomial time.
    - We transformed a satisfying assignment into an independent set of size k.
    - We transformed independent sets of size k into a satisfying assignment.

Clique and Vertex Cover

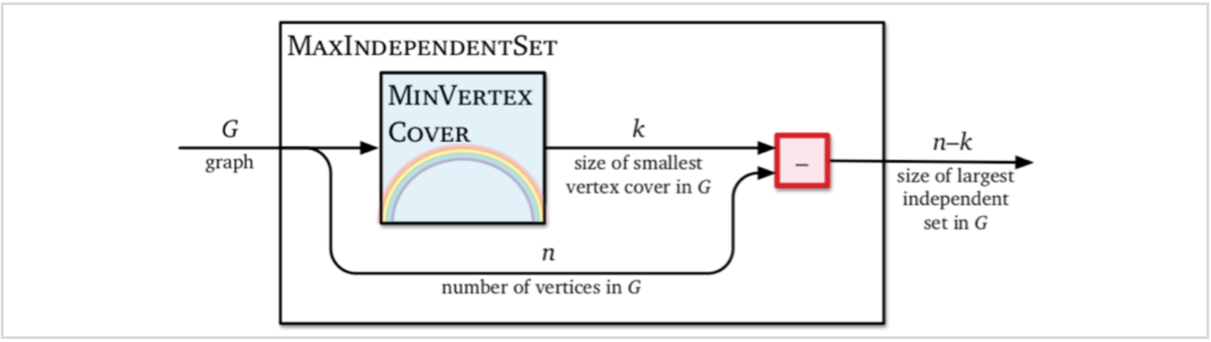
- Let's define two more problems. A *clique* is another name for a complete graph. The *MaxClique* problem asks for the number of vertices in the largest complete subgraph of G.
- A *vertex cover* is a set of vertices that touch every edge in the graph. *MinVertexCover* asks for the size of the smallest vertex cover in the graph.



- Claim: MaxClique and MinVertexCover are both NP-hard.
- For MaxClique, we define the edge-complement -G of G as the graph with the same vertices but the opposite set of edges so uv is an edge in -G if and only if it wasn't an edge in G.
- A set of vertices is independent in G if and only if it is a clique in -G, so we can solve MaxIndSet by solving MaxClique in the complement!


MaxIndependentSet
G
graph
complement in $O(V^2)$ time
$\bar{G}$
graph
MaxClique
k
size of largest clique in $\bar{G}$
k
size of largest independent set in $\bar{G}$

- For MinVertexCover, 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.


MaxIndependentSet
G
graph
MinVertex Cover
k
size of smallest vertex cover in G
$-$
n
number of vertices in G
$n-k$
size of largest independent set in G