CSE 431: Introduction to the Theory of Computation
Lecture 13: NP-Completeness Reductions I
5/13/14

Review:

A language L is <u>NP-complete</u> if:

- $L \in NP$
- $If\ B \in NP, then\ B \leq_p L$

Cook-Levin Theorem: SAT is NP-complete.

Suppose *L* is NP-complete. If we know that $A \in NP$ and $L \leq_p A$, then *A* is NP-complete.

We already know that $3SAT \leq_p CLIQUE$. As a reminder:

SAT formula: $\phi = \left(x_1 \wedge (x_2 \vee \overline{x_3})\right) \vee \ (x_5 \vee \overline{x_2})$
3SAT formula: $\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_1} \vee x_4) \dots$

Also remember:

$$\phi = \phi_{start} \wedge \phi_{accept} \wedge \phi_{cell} \wedge \phi_{move}$$

$$\phi_{move} = \bigwedge_{1 \leq i,j \leq n^k} \bigvee_{legal\ windows} (x_{i,j,a} = 1 \wedge x_{i+1,j+1,a} = 1 \dots)$$

CNF form: $(x_1 \vee \overline{x_2} \vee x_7 \vee x_{13} \vee x_3) \wedge (\dots \vee \dots \vee \dots) \wedge \dots$
Clauses of ORs joined by ANDs.

Recall the Distributive Rule for logic: $(a \wedge b) \vee c = (a \vee b) \wedge (b \vee c)$

Ex:
$(a \wedge b) \vee (c \wedge d)$
$\left(a \vee (c \wedge d)\right) \wedge \left(b \vee (c \wedge d)\right)$
$\left((a \vee c) \wedge (a \vee d)\right) \wedge \left((b \vee c) \wedge (b \vee d)\right)$
$(a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$

<u>Claim:</u> ϕ_{move} can be written as a CNF $\Rightarrow \phi \rightarrow \phi_{CNF}$

We don't worry about the size of the expansion since the number of legal tables is a constant number.

Now define a new language *CNFSAT* as follows:

$$CNFSAT = \{<\phi>: CNF\ formulas\ that\ are\ satisfiable\}$$
$$\Downarrow$$
CNFSAT is NP-complete

Using this knowledge, we can then show that *3SAT* is NP-complete.

<u>Claim</u>: *3SAT is NP-complete*

<u>Proof:</u>

$3SAT \in NP$:
This is trivially true; if we are given a set of assignments as the certificate, we can easily verify in polynomial time that they satisfy the formula.

$CNFSAT \leq_p 3SAT$:
Recall that a CNF formula ϕ_{CNF} can have many clauses joined by ANDs, but in each clause there can be any amount of literals joined by the ORs. We need to somehow convert ϕ_{CNF} to an equivalent *3SAT* formula ϕ_{3SAT} in order to continue the proof. We can do this in the following way (using the first clause as an example):
$\phi_{CNF} = (x_1 \lor \overline{x_2} \lor x_3 \lor \overline{x_7} \lor x_9) \land (\ldots) \ldots$
$\rightarrow \phi_{3SAT} = (x_1 \lor \overline{x_2} \lor \boldsymbol{z_1}) \land (\overline{\boldsymbol{z_1}} \lor x_3 \lor \boldsymbol{z_2}) \land (\overline{\boldsymbol{z_2}} \lor \overline{x_7} \lor x_9)$

Notice that we have introduced the new variable *z* in our formula, using both the variable and its negative in neighboring clauses. This converts the clauses of ϕ_{CNF} to a logically equivalent ϕ_{3SAT} form with clauses of 3 literals.

Aside: For CNF formulas that have clauses shorter than 3 literals, we can still convert them in the following way:
$\phi_{CNF} = (x_7) \rightarrow \phi_{3SAT} = (x_7 \lor x_7 \lor x_7)$
$\phi_{CNF} = (x_6 \lor x_9) \rightarrow \phi_{3SAT} = (x_6 \lor x_6 \lor x_9)$

Now we need to show that ϕ_{3SAT} is satisfiable iff ϕ_{CNF} is satisfiable.

In one direction, we can easily prove that if ϕ_{CNF} is satisfied, then so is ϕ_{3SAT}. Each clause is 3 literals and we only need one of them to be true for ϕ_{CNF} to be satisfied. We can simply look at each clause and check if one of the original variables is true; if it is, great, if not, then we can simply set our new variable *z* to be true. In this way we can make each clause evaluate to true no matter what.

For the other direction, consider the case that ϕ_{CNF} was not satisfied by a given assignment. We can then try to create a ϕ_{3SAT} that is satisfied with *only* our new variable *z*. This is impossible though! In order to have true clauses, we would have to set every one of our *z* variables to 1; this means that in the last clause $(\overline{z_n} \vee x_i \vee x_j)$ *z* would evaluate to false and so the entire formula would be false. This implies that at least *one* of the original *x* variables must be true in order for ϕ_{3SAT} to be satisfiable. Since we can then change our *z* assignment in the clause with the true variable, the rest of the clauses will be true and hence ϕ_{3SAT} is satisfied.

$$\therefore \phi_{CNF}\ is\ satisfiable \iff \phi_{3SAT}\ is\ satisfiable\ and\ CNFSAT \leq_P 3SAT$$

Since we have shown both properties of NP-completeness, we can now say that *3SAT* is NP-complete.

■

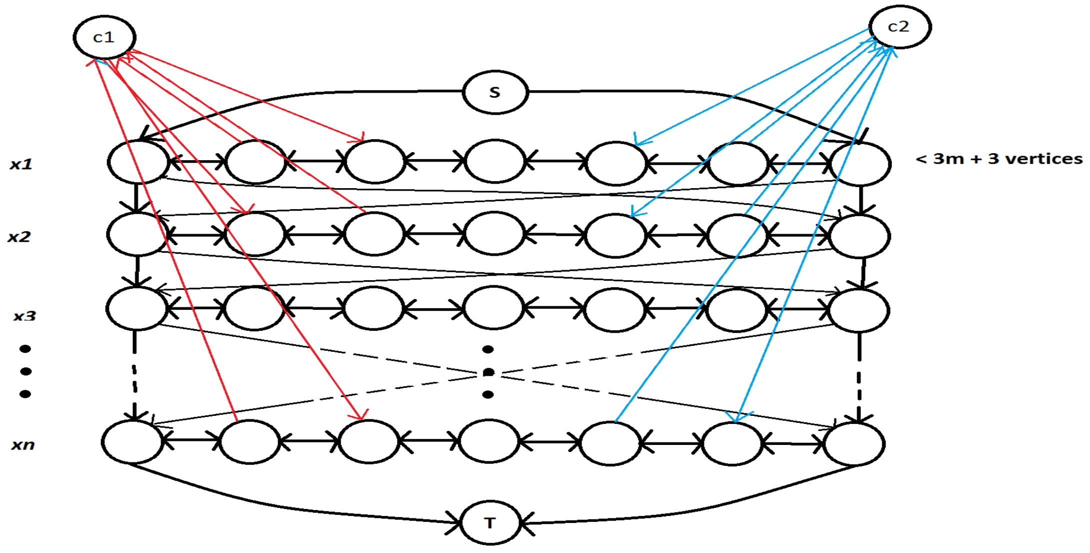
Claim: *HAMPATH* is NP-complete

Proof:

We have previously shown that $HAMPATH \in NP$, so that leaves us with proving that $3SAT \leq_p HAMPATH.$

We make the following claim to show this is true:
$\phi = C_1 \wedge C_2 \wedge C_3 \dots \wedge C_m \xrightarrow{O(n)time} Directed\ graph\ G$ where each clause C_m is in the form $(x_1 \vee x_2 \vee x_3)$ and the variables go from x_1 to x_n.

S is the starting node and *T* is the terminal node.
$C_1 = (x_1 \vee \overline{x_2} \vee x_n)$
$C_2 = (\overline{x_1} \vee x_2 \vee x_n)$



For this directed graph, each row represents a literal, and the way you traverse the graph represents a satisfying formula. In example, for the clause C_1, we can choose x_1 to be true. We traverse the graph from left to right when the variable is true, and when we see reach the node in the path that is connected to the clause node, we visit that clause node and then jump back to the next node in the path we were just on.

Similarly, if we have a false variable, we traverse the path from right to left, visiting the clause node in the same way. Notice that if we tried to traverse from right to left on a true instance, we would see the clause node, visit it, but then jump back to a node we've already seen; this is not allowed!

After we have fully traversed the graph, we will have a valid *HAMPATH* having visited all nodes (including the clause nodes).

For the other side of the proof, we can easily read a satisfying assignment from the way we traversed the graph as described above: if we are moving from left to right in a path for x_1 then it must be the case that x_1 is assigned to true, and vice versa. The graph is built in such a way that you must return to the path you were on and not a random path somewhere later in the graph; if we were able to do that we would more than likely get stuck!

$\therefore \phi_{3SAT}$ $has\ a\ satisfying\ assignment \iff G\ has\ a\ valid\ HAMPATH$

This shows that $3SAT \leq_p HAMPATH$ and so *HAMPATH* is NP-complete.

■