CS 314 - Reductions

Announcements

- Next HW up today/tomorrow, written, due next Friday at beginning of class sometime

(probably Easter Wed.)
$P \subseteq \text{NP} \cap \text{co-NP}$

Consider decision problems - output is a single boolean (yes or no).

Define:
- \( P \): the set of problems that can be solved in polynomial time.
- \( \text{NP} \): the set of decision problems where if the answer is \text{Yes}, there is a proof of this that can be checked in polynomial time.
- \( \text{co-NP} \): if answer is \text{No}, that can be checked in polynomial time.
NP-Hard

**Def**: A problem \( P \) is **NP-Hard** if it can be solved in polynomial time, then \( P = NP \).

**Def**: A problem is **NP-Complete** if it is both **NP-Hard** and in **NP**.

These are the “hardest” problems in **NP**.
Circuit - SAT

Input: boolean circuit, with T/F inputs
Output: T or F

Q: Is there a set of assignments to the inputs so that output = T?

Ex: 5 inputs

A boolean circuit. Inputs enter from the left, and the output leaves to the right.

1 output
Cook–Levin Theorem:

Circuit-satisfiability is NP-Complete.

Comments:
The proof is amazing—takes any NP problem and changes it into a circuit with polynomial size, so that:

true answer for NP problem exists

implies

the resulting circuit is satisfiable
However:

This is pretty much the only direct proof to ever show a problem is NP-Complete!

So how do we say other problems are just as hard?
Reductions

Def.: \( Y \leq_p X \) (read \( Y \) is polynomial time reducible to \( X \))

if \( Y \) can be solved using a polynomial number of steps plus a polynomial number of calls to an algorithm (or "black box") that solves \( X \).

Ex: test question about near trees!

Found cycles

Ex: sorting
Thm: Suppose $Y \leq_p X$. If $X$ can be solved in polynomial time, then so can $Y$.

Pf: Replace "black box" to solve $X$ with code that runs in poly. time.

Solving $Y$ now takes poly time + (poly # of calls to $X$)*(running time for $X$) = poly. time
This is useful for algorithms!

But what if we don't know of a polynomial time algorithm?

NP-Complete?
Spps $Y \leq_p X$. If $X$ can be solved in polynomial time, then so can $Y$.

Take the contrapositive!

Spps $Y \leq_p X$. If $Y$ cannot be solved in polynomial time, then $X$ cannot be solved in polynomial time.
So if we take a "hard" problem $P$ and reduce it to another problem $X$, then $X$ must be at least as hard.

Useful!

If we want to show a problem is $NP$-Hard, reduce a known $NP$-Hard problem to it!

Important!!
Ex: SAT

Input: boolean formula

Q: Can we assign boolean values to the variables so that the formula evaluates to true?

Ex:

\[ (a \lor b \lor c \lor \bar{d}) \iff ((b \land \bar{c}) \lor (\bar{a} \Rightarrow d) \lor (c \neq a \land b)), \]

Show SAT is NP-Hard.
How to show NP-hard?

Reduce circuit SAT to SAT

\[ x_1 \rightarrow \square \quad \iff \quad x_1 \land x_2 \]

\[ x_1 \rightarrow \square \rightarrow \square \quad \iff \quad (x_1 \land x_2) \lor x_3 \]
Ex:

\[(y_1 = x_1 \land x_4) \land (y_2 = \overline{x_4}) \land (y_3 = x_3 \land y_2) \land (y_4 = y_1 \lor x_2) \land (y_5 = \overline{x_2}) \land (y_6 = \overline{x_5}) \land (y_7 = y_3 \lor y_5) \land (y_8 = y_4 \land y_7 \land y_6) \land y_8\]

Alg: BFS through circuit

So in O(n), I can transform circuit into a boolean formula.
But careful! Need conversion to be polynomial time, + size to stay polynomial.

Well, said takes $O(n)$ to transform.

Every gate gives us 1 clause in the output formula.
Reduction Picture

Circuit \( \text{SAT} \leq_p \text{SAT} \)

\[
\begin{align*}
\text{boolean circuit} & \xrightarrow{O(n)} \text{boolean formula} \\
\text{TRUE or FALSE} & \xrightarrow{\text{SAT}} \text{TRUE or FALSE} \\
T_{\text{CSAT}}(n) & \leq O(n) + T_{\text{SAT}}(O(n)) \implies T_{\text{SAT}}(n) \geq T_{\text{CSAT}}(\Omega(n)) - O(n)
\end{align*}
\]

Circuit is satisfiable \( \iff \) boolean formula is satisfiable
Another example: 3SAT

Def: A boolean formula is in conjunctive normal form (CNF) if it is a conjunction (or AND) of clauses, each of which is a disjunction (or OR) of variables or negations.

Ex: \((a \lor b \lor c \lor d) \land (b \lor \overline{c} \lor \overline{d}) \land (\overline{a} \lor b)\)

\[\uparrow\text{clause of variables or-ed together}\]
\[\uparrow\text{"and" the clauses together}\]
3SAT (cont)

\[(a \lor b \lor c) \land (\overline{d} \lor \overline{a} \lor b) \land \ldots\]

**Def:** A 3CNF formula has exactly 3 variables per clause.

**3SAT:** Given a 3CNF formula, is there an assignment of variables which makes the formula evaluate to true?

How do we show NP-Complete?

Reduce Circuit SAT or SAT to 3SAT.
Reduction from Circuit-SAT:

1. Make every and/or gate have only 2 inputs.

2. Write down clause for each gate (same as before)

\[ x_1 = x_1 \lor x_2 \] (not in CNF)

(replace with binary tree of \( k-1 \) 2-input gates)
3. Change each clause to CNF formula:

\[
\begin{align*}
\alpha &= \overline{b} \land \overline{c} \quad \rightarrow \quad (a \lor \overline{b} \lor \overline{c}) \land (\overline{a} \lor \overline{b} \lor \overline{c}) \land (\overline{a} \lor \overline{b} \lor \overline{c}) \\
\gamma &= \overline{a} \lor \overline{b} \lor \overline{c} \\
\beta &= b \lor \overline{c} \quad \rightarrow \quad (a \lor \overline{v} \lor \overline{c}) \land (a \lor \overline{v} \lor \overline{c}) \land (a \lor \overline{v} \lor \overline{c})
\end{align*}
\]

4. Make sure every clause has 3 literals:

\[
\begin{align*}
\alpha &\rightarrow (a \lor x \lor y) \land (a \lor \overline{x} \lor y) \land (a \lor x \lor \overline{y}) \land (a \lor \overline{x} \lor \overline{y}) \\
\alpha \lor b &\rightarrow (a \lor b \lor x) \land (a \lor b \lor \overline{x})
\end{align*}
\]
\[(y_1 \lor \overline{x_1} \lor \overline{x_4}) \land (\overline{y_1} \lor x_1 \lor z_1) \land (\overline{y_1} \lor x_4 \lor z_2) \land (\overline{y_1} \lor x_4 \lor \overline{z_2}) \land (y_2 \lor x_4 \lor z_3) \land (y_2 \lor x_4 \lor \overline{z_3}) \land (y_2 \lor x_4 \lor \overline{z_4}) \land (y_2 \lor \overline{x_4} \lor z_4) \land (y_3 \lor \overline{x_3} \lor y_2) \land (\overline{y_3} \lor x_3 \lor z_5) \land (\overline{y_3} \lor x_3 \lor \overline{z_5}) \land (y_3 \lor y_2 \lor z_6) \land (y_3 \lor y_2 \lor \overline{z_6}) \land (\overline{y_4} \lor y_1 \lor x_2) \land (y_4 \lor \overline{x_2} \lor z_7) \land (y_4 \lor \overline{x_2} \lor \overline{z_7}) \land (y_4 \lor y_1 \lor z_8) \land (y_4 \lor y_1 \lor \overline{z_8}) \land (y_5 \lor x_2 \lor z_9) \land (y_5 \lor x_2 \lor \overline{z_9}) \land (\overline{y_5} \lor x_2 \lor z_{10}) \land (\overline{y_5} \lor x_2 \lor \overline{z_{10}}) \land (y_6 \lor x_5 \lor z_{11}) \land (y_6 \lor x_5 \lor \overline{z_{11}}) \land (y_6 \lor \overline{x_5} \lor z_{12}) \land (y_6 \lor \overline{x_5} \lor \overline{z_{12}}) \land (y_7 \lor \overline{y_3} \lor y_5) \land (y_7 \lor \overline{y_3} \lor \overline{z_{13}}) \land (y_7 \lor y_5 \lor \overline{z_{14}}) \land (y_7 \lor y_5 \lor \overline{z_{14}}) \land (y_8 \lor y_4 \lor y_7) \land (y_8 \lor \overline{y_4} \lor \overline{z_{15}}) \land (y_8 \lor y_4 \lor \overline{z_{15}}) \land (y_8 \lor y_7 \lor \overline{z_{16}}) \land (y_8 \lor y_7 \lor \overline{z_{16}}) \land (y_9 \lor y_8 \lor y_6) \land (y_9 \lor y_8 \lor z_{17}) \land (y_9 \lor y_8 \lor \overline{z_{17}}) \land (y_9 \lor y_6 \lor \overline{z_{18}}) \land (y_9 \lor y_6 \lor \overline{z_{18}}) \land (y_9 \lor \overline{z_{19}} \lor z_{20}) \land (y_9 \lor \overline{z_{19}} \lor \overline{z_{20}}) \land (y_9 \lor \overline{z_{19}} \lor z_{20}) \land (y_9 \lor \overline{z_{19}} \lor \overline{z_{20}})\]
Looks huge!

But:
- every gate became at most \((k-1)\) gates
  where \(k\) = # of inputs
- Every gate then formed at most 5 clauses

So transformation + size are both polynomial.

\(O(n)\) algorithm +
\(O(n)\) size boolean in 3CNF form.
Recap:

boolean circuit \( O(n) \) boolean formula \( \text{SAT} \) 

\[ T_{\text{CSAT}}(n) \leq O(n) + T_{\text{SAT}}(O(n)) \implies T_{\text{SAT}}(n) \geq T_{\text{CSAT}}(\Omega(n)) - O(n) \]

Circuit is satisfiable \( \iff \) 3CNF formula is satisfiable

So 3SAT is NP-complete.
Next time: non-logic reductions
(I promise!)