NP-completeness (cont.)

Polynomial Reduction

Recall: A language L is in P if there exists an algo A that decides, given any input string x, whether $x \in L$, and $A(x)$ runs in time $Poly(|x|)$.

$L \in P$ means L is "easy".

L is in NP iff there exists an algo $A(x,y)$ such that A runs in time $Poly(|x|)$ and for all strings x,

- if $x \in L$ then $\exists y$, $A(x,y)$ answers "yes" and $|y| = Poly(|x|)$
- if $x \notin L$ then there is no such y

y a proof of membership of x in L.

$A(x,y)$ says "yes" iff y is a valid proof.

Prop: $P \subseteq NP$. Why?

Pf: $L \in P \Rightarrow L \in NP$: verifier on input (x,y) ignores the proof y and runs the decision procedure for L on input x

Q: $NP \subseteq P$? (If so, $P = NP$).

Open, but widely believed to be false. (I.e., $P \neq NP$)

Polynomial reductions

A polynomial reduction (p-reduction) is a way of relating two languages L_1 and L_2 by comparing their difficulty.

Def: Let L_1 & L_2 be languages. We say that L_1 p-reduces (or is p-reducible) to L_2 ($L_1 \leq_p L_2$) if there exists a poly-time computable function f such that $\forall$ string x,

$x \in L_1$ iff $f(x) \in L_2$

Prop: Suppose $L_1 \leq_p L_2$. Then if $L_2 \in P$ then $L_1 \in P$.

Proof: P-time decision procedure A for L_1, given p-reduction f and a p-time decision procedure B for L_2:

A: on input x:

1. Let $y := f(x)$ (note: $|y| \in Poly(|x|)$)
2. Run $B(y)$ and output B's answer.

Correctness:

$x \in L_1 \Rightarrow y \in L_2$ (f is a reduction)
$\Rightarrow B(y)$ answers "yes"
$\Rightarrow A(x)$ answers "yes"

$x \notin L_1 \Rightarrow y \notin L_2$ (f is a p-reduction)
$\Rightarrow B(y)$ answers "no"
$\Rightarrow A(x)$ answers "no"

$\therefore A$ decides if x is in L_1

Efficiency:

Step 1: $Poly(|x|)$
Step 2: $Poly(|y|)$
$= Poly(Poly(|x|))$
$= Poly(|x|)$

$\therefore A$ runs in time $Poly(|x|)$
$\therefore L_1 \in P$. //

$L_1 \leq_p L_2$ means "L_1 is no harder than L_2"

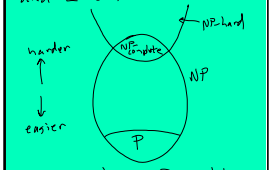
Think of $\leq_p$ as a quasiorder on Language difficulty.

Def: A language L is **NP-hard** if $L' \leq_p L$ for all $L' \in NP$.
("L is at least as hard as any problem in NP")

Obs: If L is NP-hard and $L \in P$, then $P = NP$.

Proof: For any $L' \in NP$,
$L' \leq_p L$ (by assumption)
$\therefore L' \in P$ (by the proposition assuming $L \in P$)

Def: A language L is **NP-complete** if $L \in NP$ and L is NP-hard.



Want to show NP-completeness (NP-hardness).
Technique: pReducibility.

Prop: $\leq_p$ is transitive;
if $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$, then $L_1 \leq_p L_3$

Pf: Compose the preductions. //

Cor: If L_1 is NP-hard and $L_1 \leq_p L_2$, then L_2 is NP-hard.

Theorem (Cook, Levin):
SAT is NP-hard.
(SAT $\in$ NP, so SAT is NP-complete).
Take this on faith.
In fact,
Thm: CNF-SAT is NP-complete.

CNF-SAT:

Instance: A Boolean formula φ in conjunctive normal form (cnf).

Question: Is φ satisfiable?

φ is in cnf means

$\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_k$

Each C_j is a **clause**, which is a disjunction of **literals**

$C_j = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_{n_j}$

where a literal is either a boolean variable (e.g., x) or the negation of a bool var (e.g., $\bar{x}$, means NOT x).

Proof: See CLRS or Garey & Johnson or take CSCE 551 //

Proofs of NP-hardness via preductions from CNF-SAT.

INDEPENDENT SET (IS):

Instance: a graph G and a number k.

Question: Does G have an independent set (of vertices) of size $\geq k$?

Def:
$S \subseteq G.V$ is **independent** if no two vertices in S are adjacent.

$IS \in NP$

Prop: IS is NP-hard, in fact CNF-SAT $\leq_p$ IS.
For this, we need a p-time f such that, for every cnf formula φ, $f(\varphi) = \langle G, k \rangle$ so that φ is satisfiable iff G has an indep. set of size $\geq k$.

f: Given a cnf formula φ,

1. Let $\varphi = C_1 \wedge \cdots \wedge C_k$ where each clause has some literals connected with $\vee$ (OR).
2. If any clause contains both x and $\bar{x}$, then just remove that clause from φ.
3. Construct a graph G as follows:
   a. One vertex for each literal occurrence in φ
   b. If two vertices rep literals in the same clause, make them adjacent.
   c. For any two literals of the form x and $\bar{x}$ (some var x) make them adjacent.

   This completes G's description.
4. Let k be the number of clauses.
5. Output $\langle G, k \rangle$.

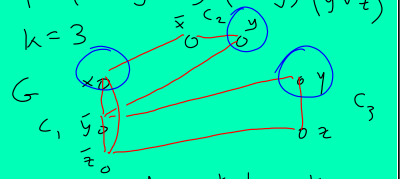
f is obviously p-time computable.

Correctness: want to show that φ is satisfiable iff G has an indep set of size $\geq k$.

$\varphi = (x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y) \wedge (y \vee z)$ (C_1, C_2, C_3)

$k = 3$

G



Note: any indep set has at most one vertex in each clause. So an indep set of size k must have exactly one vertex in each clause (there is no i.s. of size $>k$).

WTS: φ satisfiable $\Leftrightarrow$ k-size i.s. exists in G.

($\Rightarrow$) Suppose φ is satisfiable. Fix some satisfying assignment $\vec{a}$ for φ.

Let $S \subseteq G.V$ be such that a vertex with literal label ℓ is in S iff $\vec{a}$ makes ℓ true. ($\ell = 1$)

[If ℓ occurs twice in same clause, just put one occurrence in S.]

S is an independent set:
- it only ≤ 1 literal in each clause
- can't contain contradictory literals.

[Finish next time.]