Lecture 22: May 8, 2001

Admin:

- PS 6 due today!

Today:

- NP, Reductions and NP-completeness
- 3SAT is NP-complete (stated w/o proof)
- Clique is NP-complete
- Vertex cover is NP-complete

Recall NP

- Stands for Nondeterministic Polynomial-time.
- I.e., some aspect about the problems can be "handled" in polynomial time. So membership in NP is not evidence of hardness. It is a proof of some form of easiness. In fact all of P is in NP.
- Formally, a Boolean computational problem A is in NP if there is a polynomial time verifier V that can verify "proofs" y that an input x is a TRUE input (i.e., $A(x)$ =TRUE.)
  - If $A(x)$ =TRUE, a proof y satisfying $V(x, y)$ =TRU exists.
  - If there exists a y such that $V(x, y)$ =TRUE, then $A(x)$ =TRUE.

Why NP?

- Not the hardest known problems! (E.g., Task of deciding if a computer program ever halts is much harder).
- We can definitely solve every NP problem in exponential time.
- We don't know if P = NP.
- Interest in NP comes from the fact that such problems occur commonly, in many different flavors, and we would love to solve them in polynomial time.
- Since we can't we try to find the hardest problems in NP.

Reductions (simplified)

We'll present a simplified notion of a reduction that works only for Boolean function computations.

- **Defn:** A reduces to B ($A \leq_p B$) if there exists a polynomial time computable function f such that for every x,

$$A(x) = \text{TRUE} \quad \Leftrightarrow \quad B(f(x)) = \text{TRUE}.$$

- $A \leq_p B$ asserts complexity of A is less than or equal to complexity of B (ignoring polynomial factors).
- If $A \leq_p B$ then a polynomial time algorithm to solve B gives a polynomial time algorithm to solve A.
- $A \leq_p B$ and $B \leq_p C$ implies $A \leq_p C$ (why?).

NP-Completeness

- Now that we have a way of comparing problems, it makes sense to ask: What is the hardest problem in NP (if there is one such)?
- Such a problem, say A, should satisfy $A' \leq_p A$ for every $A' \in NP$.
- Turns out many such problems exist - these problems are called NP-complete.
- **Defn:** A is NP-complete if:
  1. A is in NP.
  2. $A' \leq A$ for every A' in NP.

3SAT is NP-complete

- Recall 3SAT:
  - Input is m clauses of upto 3 terms on n Boolean variables.
  - Output is TRUE if there is a setting to the n variables that sets at least one term in every clause to TRUE.
- Legendary theorem (by CS scale of time):

  Thm: [Cook, Levin '72] 3SAT is NP-complete.

3SAT is NP-Complete (contd.)

- How to prove this?
- Need to consider every problem in NP, and reduce each one to 3SAT.
- Catch: There are infinitely many problems in NP. The above process will take infinite time!
- Need to do better: Come up with a uniform description of all problems, and show how the common description itself reduces to 3SAT. Not very hard once the right definitions are in place, but we won't do this in this course (try 6.045/6.840 if you want to learn more).
- We state this theorem without proof.

Clique is NP-Complete

- Recall Clique:
  - Input is undirected graph G and integer k.
  - Output is TRUE if G has a clique (complete subgraph) of size k.

  Thm: [Cook, Karp '72] Clique is NP-complete.

Clique is NP-complete (contd.)

- How to show this?
- Still face same problem as before?
- Not really: Only need to show
  - Clique is in NP (already know this).
  - 3SAT $\leq_p$ Clique.
- Why does latter suffice? Suppose A' is in NP.
  - $A' \leq_p$3SAT (by prev. thm.)
  - 3SAT $\leq_p$ Clique (as we'll show).
  - Hence $A' \leq_p$Clique (transitivity).

Reducing 3SAT to Clique

- What does this entail:
- Have to show a reduction, i.e., a function f such that:
  - f takes a 3SAT input, say ϕ.
  - f outputs a graph G and integer k.
  - ϕ is satisfiable iff G has a clique of size k.
- Reduction is rather clever but short. So we won't motivate it, but just give it away.

The reduction

- Given 3SAT input ϕ with m clauses, G will have $3m$ vertices, one vertex associated with every term of the input.
- (Have now told you what the vertices of G are. Now need to tell you what the edges are.)
- Edges in G are as follows: Two vertices are connected iff their associated terms satisfy the following
  - The terms are from different clauses
  - The terms are not complementary (i.e., x_1 and NOT x_1.)
- (What remains to do to complete the reduction? Need to specify k).
- $k = m$.
- Note all the above took only polynomial time.

Analysis

- Main idea: Picking vertices to belong to a clique, corresponds to picking terms that are TRUE from distinct clauses. Furthermore the set of terms being chosen to be TRUE is consistent, since no complementary pair is chosen.
- Proof 1: If ϕ is satisfiable, then G has a clique of size k: Take a satisfying assignment and pick one term from each clause that is set to TRUE. Take the set of corresponding vertices of G. This is a set of $k = m$ vertices which forms a clique.
- Proof 2: If G has a clique of size m, then ϕ is satisfiable.
  - The clique contains at most one vertex from each clause.
  - Since clique size is m, it must thus have exactly one vertex from each clause.
  - Assign variables of ϕ as follows: For every i, if there is a clique vertex associated with x_i, then

set x_i to TRUE, else set x_i to FALSE. (Notice that if the clique has a vertex setting x_i to TRUE, then it has no vertices corresponding to NOT x_i.)

– This assignment satisfies every clause.

– Moral of the story: Can cosntruct and analyze reductions.

– Thus can show a problem to be NP-complete even if we don't have a formal definition of the model.

Vertex Cover

We'll now try another NP-completeness reduction, as an exercise.

- A set $X \subset V$ is a vertex cover for a graph H if for every edge $u \leftrightarrow v$ in H, at least one of u or v is in X.
- Vertex Cover Problem:
  - Input is a graph H and integer l.
  - Output is TRUE if H has a cover of size at most l.
- **Thm:** Vertex Cover is NP-complete.
- Suffices to show Clique $\leq_p$ Vertex Cover.
- Reduction as follows:

$$(G, k) \mapsto (H = G^c, V - k)$$

  where G^c is the complement of G: u and v are adjacent in G^c iff u and v are not adjacent in G.
- Proof of correctness: Exercise.