Algorithms

NP Completeness Continued:

Reductions

Review: **P** and **NP**

- *What do we mean when we say a problem is in **P**?*
- *What do we mean when we say a problem is in **NP**?*
- *What is the relation between **P** and **NP**?*

Review: **P** and **NP**

- *What do we mean when we say a problem is in **P**?*
  - A: A solution can be found in polynomial time
- *What do we mean when we say a problem is in **NP**?*
  - A: A solution can be verified in polynomial time
- *What is the relation between **P** and **NP**?*
  - A: **P** $\subseteq$ **NP**, but no one knows whether **P = NP**

Review: NP-Complete

- *What, intuitively, does it mean if we can reduce problem P to problem Q?*
- *How do we reduce P to Q?*
- *What does it mean if Q is NP-Hard?*
- *What does it mean if Q is NP-Complete?*

Review: NP-Complete

- *What, intuitively, does it mean if we can reduce problem P to problem Q?*
  - P is "no harder than" Q
- *How do we reduce P to Q?*
  - Transform instances of P to instances of Q in polynomial time s.t. Q: "yes" iff P: "yes"
- *What does it mean if Q is NP-Hard?*
  - Every problem $P \in \mathbf{NP} \leq_p Q$
- *What does it mean if Q is NP-Complete?*
  - Q is NP-Hard and $Q \in \mathbf{NP}$

Review: Proving Problems NP-Complete

- *How do we usually prove that a problem R is NP-Complete?*
- A: Show R $\in$**NP**, and reduce a known NP-Complete problem Q to R

Review: Directed $\Rightarrow$ Undirected Ham. Cycle

- Given: directed hamiltonian cycle is NP-Complete (draw the example)
- Transform graph G = (V, E) into G' = (V', E'):
  - Every vertex v in V transforms into 3 vertices v^1, v^2, v^3 in V' with edges (v^1,v^2) and (v^2,v^3) in E'
  - Every directed edge (v, w) in E transforms into the undirected edge (v^3, w^1) in E' (draw it)

Review: Directed $\Rightarrow$ Undirected Ham. Cycle

- Prove the transformation correct:
  - If G has directed hamiltonian cycle, G' will have undirected cycle (straightforward)
  - If G' has an undirected hamiltonian cycle, G will have a directed hamiltonian cycle
    - The three vertices that correspond to a vertex v in G must be traversed in order v^1, v^2, v^3 or v^3, v^2, v^1, since v^2 cannot be reached from any other vertex in G'
    - Since 1's are connected to 3's, the order is the same for all triples. Assume w.l.o.g. order is v^1, v^2, v^3.
    - Then G has a corresponding directed hamiltonian cycle

Review: Hamiltonian Cycle $\Rightarrow$ TSP

- The well-known *traveling salesman problem*:
  - Complete graph with cost c(i,j) from city i to city j
  - $\exists$ a simple cycle over cities with cost $< k$?
- *How can we prove the TSP is NP-Complete?*
- A: Prove TSP $\in$ **NP**; reduce the undirected hamiltonian cycle problem to TSP
  - TSP $\in$ **NP**: straightforward
  - Reduction: need to show that if we can solve TSP we can solve ham. cycle problem

Review: Hamiltonian Cycle $\Rightarrow$ TSP

- To transform ham. cycle problem on graph G = (V,E) to TSP, create graph G' = (V,E'):
  - G' is a complete graph
  - Edges in E' also in E have weight 0
  - All other edges in E' have weight 1
  - TSP: is there a TSP on G' with weight 0?
    - If G has a hamiltonian cycle, G' has a cycle w/ weight 0
    - If G' has cycle w/ weight 0, every edge of that cycle has weight 0 and is thus in G. Thus G has a ham. cycle

The SAT Problem

- One of the first problems to be proved NP-Complete was *satisfiability* (SAT):
  - Given a Boolean expression on n variables, can we assign values such that the expression is TRUE?
  - Ex: $((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$
  - **Cook's Theorem:** The satisfiability problem is NP-Complete
    - **Note: Argue from first principles, not reduction**
    - **Proof: not here**

Conjunctive Normal Form

- Even if the form of the Boolean expression is simplified, the problem may be NP-Complete
  - *Literal*: an occurrence of a Boolean or its negation
  - A Boolean formula is in *conjunctive normal form*, or *CNF*, if it is an AND of clauses, each of which is an OR of literals
    - Ex: $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_5)$
  - *3-CNF*: each clause has exactly 3 distinct literals
    - Ex: $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_5 \vee x_3 \vee x_4)$
    - Notice: true if at least one literal in each clause is true

The 3-CNF Problem

- Thm 36.10: Satisfiability of Boolean formulas in 3-CNF form (the *3-CNF Problem*) is NP-Complete
  - Proof: Nope
- The reason we care about the 3-CNF problem is that it is relatively easy to reduce to others
  - Thus by proving 3-CNF NP-Complete we can prove many seemingly unrelated problems NP-Complete

3-CNF → Clique

- *What is a clique of a graph G?*
- A: a subset of vertices fully connected to each other, i.e. a complete subgraph of G
- The *clique problem*: how large is the maximum-size clique in a graph?
- *Can we turn this into a decision problem?*
- A: Yes, we call this the *k-clique problem*
- *Is the k-clique problem within **NP**?*

3-CNF $\rightarrow$ Clique

- *What should the reduction do?*
- A: Transform a 3-CNF formula to a graph, for which a k-clique will exist (for some k) iff the 3-CNF formula is satisfiable

3-CNF $\rightarrow$ Clique

- The reduction:
  - Let $B = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ be a 3-CNF formula with k clauses, each of which has 3 distinct literals
  - For each clause put a triple of vertices in the graph, one for each literal
  - Put an edge between two vertices if they are in different triples and their literals are *consistent*, meaning not each other's negation
  - Run an example:
    $B = (x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (x \vee y \vee z)$

3-CNF → Clique

- Prove the reduction works:
  - If B has a satisfying assignment, then each clause has at least one literal (vertex) that evaluates to 1
  - Picking one such "true" literal from each clause gives a set V' of *k* vertices. V' is a clique (*Why?*)
  - If G has a clique V' of size k, it must contain one vertex in each triple (clause) (*Why?*)
  - We can assign 1 to each literal corresponding with a vertex in V', without fear of contradiction

Clique → Vertex Cover

- A *vertex cover* for a graph G is a set of vertices incident to every edge in G
- The *vertex cover problem*: what is the minimum size vertex cover in G?
- Restated as a decision problem: does a vertex cover of size *k* exist in G?
- Thm 36.12: vertex cover is NP-Complete

Clique $\rightarrow$ Vertex Cover

- First, show vertex cover in **NP** (*How?*)
- Next, reduce *k*-clique to vertex cover
  - The *complement* G_C of a graph G contains exactly those edges not in G
  - Compute G_C in polynomial time
  - G has a clique of size *k* iff G_C has a vertex cover of size $|V| - k$

Clique → Vertex Cover

- Claim: If G has a clique of size k, G_C has a vertex cover of size |V| - k
  - Let V’ be the k-clique
  - Then V - V’ is a vertex cover in G_C
    - Let (u,v) be any edge in G_C
    - Then u and v cannot both be in V’ (*Why?*)
    - Thus at least one of u or v is in V-V’ (*why?*), so edge (u, v) is covered by V-V’
    - Since true for *any* edge in G_C, V-V’ is a vertex cover

Clique $\rightarrow$ Vertex Cover

- Claim: If G_C has a vertex cover $V' \subseteq V$, with $|V'| = |V| - k$, then G has a clique of size k
  - For all $u,v \in V$, if $(u,v) \in G_C$ then $u \in V'$ or $v \in V'$ or both (*Why?*)
  - Contrapositive: if $u \notin V'$ and $v \notin V'$, then $(u,v) \in E$
  - In other words, all vertices in V-V' are connected by an edge, thus V-V' is a clique
  - Since $|V| - |V'| = k$, the size of the clique is k

General Comments

- Literally hundreds of problems have been shown to be NP-Complete
- Some reductions are profound, some are comparatively easy, many are easy once the key insight is given
- You can expect a simple NP-Completeness proof on the final

Other NP-Complete Problems

- *Subset-sum*: Given a set of integers, does there exist a subset that adds up to some target T?
- *0-1 knapsack*: when weights not just integers
- *Hamiltonian path*: Obvious
- *Graph coloring*: can a given graph be colored with k colors such that no adjacent vertices are the same color?
- Etc…