

CS 432: Algorithms

NP Completeness Continued: Reductions

Review: P And NP Summary

- **P** = set of problems that can be solved in polynomial time
- **NP** = set of problems for which a solution can be verified in polynomial time
- **P** \subseteq **NP**
- Open question: Does **P** = **NP**?

Review: Reduction

- A problem A can be *reduced* to another problem B if any instance of A can be rephrased to an instance of B, the solution to which provides a solution to the instance of A
 - This rephrasing is called a *transformation*
- Intuitively: If A reduces in polynomial time to B, A is "no harder to solve" than B
 - I.e. if B is polynomial, A is not exponential

Review: NP-Hard and NP-Complete

- If A is *polynomial-time reducible* to B, we denote this $A \leq_p B$
- Definition of NP-Hard and NP-Complete:
 - If all problems $R \in \mathbf{NP}$ are reducible to A, then A is *NP-Hard*
 - We say A is *NP-Complete* if A is NP-Hard and $A \in \mathbf{NP}$
- If $A \leq_p B$ and A is NP-Complete, B is also NP-Complete

Review: Proving NP-Completeness

- *What steps do we have to take to prove a problem Y is NP-Complete?*
 - Pick a known NP-Complete problem X
 - Assuming there is one! (More later.)
 - Reduce X to Y
 - Describe a transformation that maps instances of X to instances of Y, s.t. "yes" for Y = "yes" for X
 - Prove the transformation works
 - Prove it runs in polynomial time
 - Oh yeah, prove $Y \in \mathbf{NP}$

Order of the Reduction When Proving NP-Completeness

- To prove Y is **NP-c**, show $X \leq_p Y$ where $X \in \mathbf{NP-c}$
 - Why have the known NP-c problem "on the left"? Shouldn't it be the other way around? (No!)
- If $X \in \mathbf{NP-c}$, then: all NP problems $\leq_p X$
- If you show $X \leq_p Y$, then:
any-NP-problem $\leq_p X \leq_p Y$
- Thus any problem in NP can be reduced to Y if the two transformations are applied in sequence
 - And both are polynomial

Can a Problem be NP-Hard but not NP-C?

- So, find a reduction and then try to prove $Y \in \mathbf{NP}$
 - *What if you can't?*
- Are there any problems Y that are NP-hard but not NP-complete? This means:
 - All problems in NP reduce to Y. (A known NP-c problem can be reduced to Q.)
 - But, Y cannot be proved to be in NP
- Yes! Some examples:
 - Non-decision forms of known NP-Cs (e.g. TSP)
 - The halting problem. (Transform a SAT expression to a Turing machine.)
 - Others.

But You Need One NP-c First...

- If you have one NP-c problem, you can use the technique just described to prove other problems are NP-c
- The definition of NP-complete was created to prove a point
 - There *might be* problems that are at least as hard as "anything" (i.e. all NP problems)
- Are there really NP-complete problems?
 - Stephen Cook, 1971. **Cook's Theorem:**
The satisfiability problem is NP-Complete.
 - He proved this "directly", from first principles
 - Showed that any problem that meets the definition of NP can be transformed in polynomial time to a CNF formula.
 - Proof outside the scope of this course (lucky you)

More About 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: $((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$
- You might imagine that lots of decision problems could be expressed as a complex logical expression
 - And Cook proved you were right!
 - Proved the general result that any NP problem can be expressed

Conjunctive Normal Form

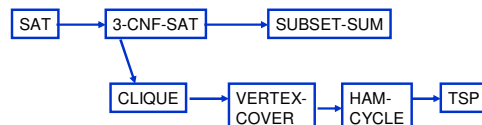
- Even if the form of the Boolean expression is simplified, the problem may be NP-Complete
 - **Literal:** an occurrence of a Boolean or its negation
 - A Boolean formula is in **conjunctive normal form**, or **CNF**, if it is an AND of clauses, each of which is an OR of literals
 - Ex: $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_5)$
 - **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
 - Note: Arbitrary expressions can be translated into CNF forms by introducing intermediate variables etc.

The 3-CNF Problem

- Satisfiability of Boolean formulas in 3-CNF form (the **3-CNF Problem**) is NP-Complete
 - Proof: not in this course
- 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

Joining the Club

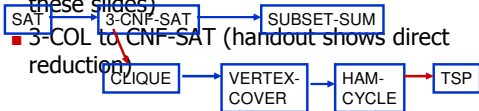
- Given one NP-c problem, others can join the club
 - Prove that SAT reduces to another problem, and so on...



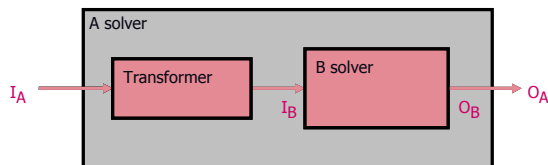
- Membership in NP-c grows...
- Classic textbook: Garey, M. and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1979.

Examples of Reductions

- Examples covered in class:
 - 3-CNF to k-Clique (in these slides)
 - Directed to Undirected Hamilton Cycle (slides & handout)
 - Hamilton Cycle to Traveling Salesperson (in these slides)
 - 3-COL to CNF-SAT (handout shows direct reduction)



Reminder: A reduces to B



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 → k-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
- And do this in polynomial time.

Reduction: 3-CNF → k-Clique

- Let $B = C_1 \wedge C_2 \wedge \dots \wedge C_k$ 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
 - Not consistent: x and $\neg x$, y and $\neg y$, etc.
 - Consistent: x and x , x and y , x and $\neg y$, etc.
- An example:
 - $B = (x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (x \vee y \vee z)$
 - (See graph for this next page: note minor notation changes)

Graph and Clique for Previous Example

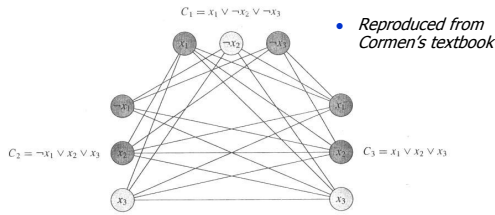


Figure 34.14 The graph G derived from the 3-CNF formula $\phi = C_1 \wedge C_2 \wedge C_3$, where $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$, $C_2 = (\neg x_1 \vee x_2 \vee x_3)$, and $C_3 = (x_1 \vee x_2 \vee x_3)$, in reducing 3-CNF-SAT to CLIQUE. A satisfying assignment of the formula has $x_2 = 0$, $x_3 = 1$, and x_1 may be either 0 or 1. This assignment satisfies C_1 with $\neg x_2$, and it satisfies C_2 and C_3 with x_3 , corresponding to the clique with lightly shaded vertices.

3-CNF \rightarrow k-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

Directed Hamiltonian Cycle \Rightarrow Undirected Hamiltonian Cycle

- *What was the hamiltonian cycle problem again?*
- For my next trick, I will reduce the *directed hamiltonian cycle* problem to the *undirected hamiltonian cycle* problem before your eyes
 - Why would I want to? To prove something in NP-C
 - *Question: Which variant am I proving NP-Complete?*
- Draw a directed example on the board
 - *Question: What transformation do I need to effect?*

Transformation: Directed \Rightarrow Undirected Ham. Cycle

- See handout (from page 563 in Baase textbook)
- Transform directed graph $G = (V, E)$ into undirected graph $G' = (V', E')$:
 - 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)
 - *Can this be implemented in polynomial time?*
 - *Argue that a directed hamiltonian cycle in G implies an undirected hamiltonian cycle in G'*
 - *Argue that an undirected hamiltonian cycle in G' implies a directed hamiltonian cycle in G*

Undirected Hamiltonian Cycle

- Thus we can reduce the directed problem to the undirected problem
- *What's left to prove the undirected hamiltonian cycle problem NP-Complete?*
- *Argue that the problem is in NP*

Hamiltonian Cycle \Rightarrow TSP

- The well-known *traveling salesman problem*:
 - Optimization variant: a salesman must travel to n cities, visiting each city exactly once and finishing where he begins. How to minimize travel time?
 - Model as complete graph with cost $c(i, j)$ to go from city i to city j
- *How would we turn this into a decision problem?*
 - A: ask if \exists a TSP with cost $< k$

Hamiltonian Cycle \Rightarrow TSP

- The steps to prove TSP is NP-Complete:
 - Prove that TSP \in NP (*Argue this*)
 - Reduce the undirected hamiltonian cycle problem to the TSP
 - So if we had a TSP-solver, we could use it to solve the hamiltonian cycle problem in polynomial time
 - *How can we transform an instance of the hamiltonian cycle problem to an instance of the TSP?*
 - *Can we do this in polynomial time?*

How to show HamCycle \leq_p TSP?

- Transform input for HamCycle into input for TSP
 - HamCycle: Given unweighted graph G1, does it have a ham. cycle?
 - TSP: Given weighted graph G2 and k, is there a ham. cycle with total cost less than k?
- Must convert unweighted graph to weighted
 - Add edges to G1 to make a complete graph G2
 - Add weights as follows:
 - wt(i,j) is 0 if edge i,j is in G1 (original graph for HamCycle)
 - wt(i,j) is 1 if edge i,j is not in G1
- G1 has ham. cycle iff G2 has TSP with k=0
 - Can you see why? Is this transformation polynomial?

The TSP

- Random asides:
 - TSPs (and variants) have enormous practical importance
 - E.g., for shipping and freighting companies
 - Lots of research into good approximation algorithms
 - Recently made famous as a DNA computing problem
 - "further reading" section of Baase textbook, Ch. 13 (I'll supply copy if you're interested)

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

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*
- *Graph coloring*: can a given graph be colored with k colors such that no adjacent vertices are the same color?
- Etc...

Reminders and Review!

Important: Input Size and P

- Sometimes a problem seems to be in P but really isn't
- Example: finding if value n is a prime
 - Just loop and do a mod: $\Theta(n)$
- Note that here " n " is not the count or number of data items.
 - There's just one input item.
 - But " n " is a value with a size that affects the execution time.
 - The size of is $\log(n)$
 - $T(\text{size}) = n$ but size is $\log(n)$.
 - $T(\log n) = n = 10^{\log n}$ This is really an exponential!
- Be careful if " n " is not a count of data items but a value
 - Dynamic programming problems, e.g. the 0/1 knapsack

Review (Again)

- A problem B is *NP-complete*
 - if it is in NP **and** it is NP-hard.
- A problem B is *NP-hard*
 - if *every* problem in NP is reducible to B.
- A problem A is *reducible* to a problem B if
 - there exists a polynomial reduction function T such that
 - For every string x ,
 - if x is a yes input for A, then $T(x)$ is a yes input for B
 - if x is a no input for A, then $T(x)$ is a no input for B.
 - T can be computed in polynomially bounded time.

NP-Complete Problems

- NP-Complete problems are the "hardest" problems in NP:
 - If any *one* NP-Complete problem can be solved in polynomial time...
 - ...then *every* NP-Complete problem can be solved in polynomial time...
 - ...and in fact *every* problem in **NP** can be solved in polynomial time (which would show **P = NP**)
 - Thus: solve any NP-Complete problem in $O(n^{100})$ time, you've proved that **P = NP**. Retire rich & famous.

What We Don't Know: Open Questions

- Is it **impossible** to solve an NP-c problem in polynomial time?
 - No one has proved an exponential lower bound for any problem in NP
 - But, computer scientists believe such a L.B. exists for NP-c problems.
- Are all problems in NP tractable or intractable? I.e., does $P=NP$ or not?
 - If someone found a polynomial solution to any NP-c problem, we'd know $P = NP$.
 - But, computer scientists believe $P \neq NP$.

Unused slides from Fall 2004

- Another reduction
 - k-CLIQUE to VertexCover

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 \rightarrow 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