CSE 589 Part V

One of the symptoms of an approaching nervous breakdown is the belief that one's work is terribly important.

Bertrand Russell

Reading

Skiena, chapter 6

CLR, chapter 36

Easy vs. Hard Problems

The standard definition of a tractable problem is one that can be solved in time that is polynomial in the size of the input. Why?

- Very few practical problems require time which is high-degree polynomial
- equivalent on different models of computation
- nice closure properties

This class of problems called P.

NP -- class of problems whose solution can be verified in polynomial time. Framed as "decision problems".

Many many many many many many many many many many important problems are in NP (in fact, NP-complete.)

Hardest problems in NP are NP-complete

Provably NP-complete problems --

If any one can be solved in polynomial time, then all of them can.

Right now, no known efficient algorithm known. Biggest open problem in CS:

P = NP?

Heuristics/approximation algorithms typically used. Sometimes solved exactly too -- depends on the application area.

Most notorious hard graph problem

Traveling Salesman Problem

- Input description: A weighted graph G, B >= 0
- Output description: Is there a cycle going through each of the vertices once of total cost at most B?

Example: optimization of tool path for manufacturing equipment. E.g., robot arm assigned to solder all connections on printed circuit board.

Essence of NP-completeness

We haven't a clue, so when we're given a problem we can't solve efficiently, we try to find whether it is equivalent to other problems we can't solve efficiently.

Hundreds (thousands?) of equivalently hard NP-complete problems of immense practical importance. (scheduling, resource allocation, hardware design and test,……..)

Polynomial Time Reductions

Let R and Q be two problems. We say that R is **polynomially reducible** to Q if there is a polynomial time algorithm that converts each input r to R to another input q to Q such that r is a yes-instance of R if and only if q is a yes-instance of Q.

Theorem: If R is polynomially reducible to Q and there is a polynomial time algorithm for Q, then there is a polynomial time algorithm for R.

Other Factoid: Polynomial reducibility is transitive.

Definitions, etc.

NP : class of problems whose solutions can be verified in polynomial time.

A problem Q is **NP-hard** if every problem in NP is polynomially reducible to Q.

A problem Q is **NP-complete** if

- Q belongs to NP
- Q is NP-hard

Proving a Problem is NP-complete

Cook proved that there exist NP-complete problems (satisfiability).

Once we have one, can start blazing.

To prove a problem Q is NP-complete

- show Q is in NP
- show R is polynomially reducible to Q, for some NP-complete problem R.

Remarkable Theorem of Steve Cook (1971)
Proved that there exist NP-complete problems

The Satisfiability problem is NP-complete.

Satisfiability: Given a boolean formula in conjunctive normal form (and of ors), is there an assignment of the variables to 0's and 1's so that the resulting formula evaluates to 1?

Example:

[redacted]

Idea of Proof

In NP: easy.

If a problem is in NP, there is a nondeterministic Turing machine that recognizes yes-instances.

A Turing machine and all its operations on a given input can be described by a Boolean expression such that:

Expression is satisfiable if and only if the Turing machine will terminate at an accepting state for given input.

=> any NP algorithm can be described by an instance of SAT

Some NP-complete Problems

Satisfiability

Input description: Given a boolean formula in conjunctive normal form

Problem description: Is there a truth assignment for the variables that causes the formula to evaluate to 1.

Special case where every clause is disjunction of exactly 3 literals also NP complete (called 3-SAT)

Example: digital design, hardware testing,….

Traveling Salesman Problem

Input description: A weighted graph G, L

Output description: Is there a tour of length at most L that visits each of the vertices exactly once.

Optimization version: minimize the length of the tour.

Clique

Input description: A graph G=(V,E), k

Problem description: Is there a subset S of V of size at least k such that for all x,y in S, (x,y) in E.

Optimization Version: Find maximum sized subset S.

Vertex Coloring

Input description: A graph G=(V,E), k

Problem description: Is it possible to color the vertices of the graph using at most k colors such that for each edge (i,j) in E, vertices i and j have different colors

Optimization version: minimize the number of colors used.

Example: Register allocation for compilers.

Independent Set

Input description: A graph G=(V,E), k

Problem description: Is there a subset S of V of size at least k such that no pair of vertices in S has an edge between them.

Example:

- Identifying location for a new franchise service such that no two locations are close enough to compete with each other.
- Highest capacity code for given communication channel.

Hamiltonian Cycle

Input description: A graph G=(V,E)

Problem description: Is there an ordering of the vertices such that adjacent vertices in the ordering are connected by an edge and each vertex is visited exactly once.

Example:

Triangle strip problem.

✓

Graph Partition

Input description: A weighted graph G=(V,E) and integers j,k

Problem description: Is there a partition of the vertices into two subsets such that each subset has size at most j, and the weight of edges connecting the two subsets is at most k.

Example:
VLSI layout

Steiner Tree

Input description: A graph G=(V,E), a subset T of the vertices V, and a bound B

Problem description: Is there a tree connecting all the vertices of T of total weight at most B?

Example:
Network design and wiring layout.

Integer Linear Programming

Input description: A linear functional **cx**, a set of linear constraints **Ax** >= **b**, a set of non-negative variables **x** that can take on only integer values, say 0 or 1, a value V.

Problem description: Are there 0/1 integer values for the variables **x** satisfying the linear constraints such that the linear functional **cx** <= V.

Optimization version: minimize V.

Example: absolutely everything

How you prove a problem Q is NP-complete.

1. Prove it's in NP
2. Select a known NP-complete problem R.
3. Describe a polynomial time computable algorithm that computes a function f mapping every instance of R to some instance of Q.
4. Prove that for every yes-instance of R maps to a yes-instance of Q, and every no-instance of R maps to a no-instance of Q.

Remember

showing a problem A is NP-complete is showing that you can **use** A to solve a known NP-complete problem

Let's do some NP-completeness proofs

Suppose somebody else has already shown that the following problems are NP-complete

- 3SAT
- Hamiltonian Cycle
- Vertex Cover

✔

Let's show the following problems are NP-complete

Clique

Independent Set

Travelling Salesman Problem

Proof that Clique is NP-complete

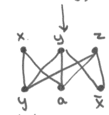
1. Clique $\in$ NP
2. Reduction from 3SAT

arbitrary instance F of 3SAT with m clauses → instance of clique (G, m)

$\forall$ clause $(x, y, z) \rightarrow$ 3 vertices

$\forall$ pair of clauses edges as follows

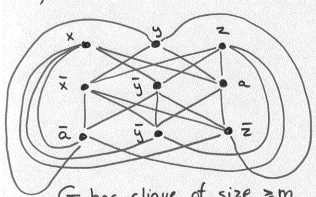
(x, y, z) $(y, a, \bar{x})$



edge between <u>non</u> complements

$F = (x, y, z)\ (\bar{x}, \bar{y}, a)\ (\bar{a}, \bar{y}, \bar{z})$

G, m = 3



G has clique of size $\geq m$ iff F is satisfiable

$\Rightarrow$ G has clique of size $\geq m$ $(= m)$
can't be var & its complement in set of vertices
$\Rightarrow$ those assignments satisfy F

$\Leftarrow$ $\exists$ satisfying truth assignment for F
pick one true literal from each clause

Comments on NP-completeness proofs

- hardest part -- choosing a good problem from which to do reduction
- must do reduction from arbitrary instance
- common error -- backwards reduction. Remember that you are using your problem as a black box for solving known NPC problem
- freedom in reduction: if problem includes parameter, can set it in a convenient way
- size of problem can change as long as it doesn't increase by more than polynomial

Comments cont.

- if problem is generalization of known NP-complete problem, reduction is usually easy.

Example: Set Cover

- given U, set of elements, and collection S1, S2,.., Sn of subsets of U, and an integer k
- determine if there is a subset W of U of size at most k that intersects ever set Si

Reduction from Vertex Cover

- U set of vertices
- Si is ith edge