Lecture 19:
Introduction to NP-Completeness

Steven Skiena

Department of Computer Science
State University of New York
Stony Brook, NY 11794–4400

http://www.cs.stonybrook.edu/~skiena

Reporting to the Boss

Suppose you fail to find a fast algorithm. What can you tell your boss?

- "I guess I'm too dumb..." (dangerous confession)
- "There is no fast algorithm!" (lower bound proof)
- "I can't solve it, but no one else in the world can, either..." (NP-completeness reduction)

The Theory of NP-Completeness

Several times this semester we have encountered problems for which we couldn't find efficient algorithms, such as the traveling salesman problem.

We also couldn't prove exponential-time lower bounds for these problems.

The theory of NP-completeness, developed by Stephen Cook and Richard Karp, provides the tools to show that all of these problems were really the same problem.

The Main Idea

Suppose I gave you the following algorithm to solve the *bandersnatch* problem:

Bandersnatch(G)

Convert G to an instance of the Bo-billy problem Y.
Call the subroutine Bo-billy on Y to solve this instance.
Return the answer of Bo-billy(Y) as the answer to G.

Such a translation from instances of one type of problem to instances of another type such that answers are preserved is called a *reduction*.

What Does this Imply?

Now suppose my reduction translates G to Y in $O(P(n))$:

1. If my Bo-billy subroutine ran in $O(P'(n))$ I can solve the Bandersnatch problem in $O(P(n) + P'(n'))$
2. If I know that $\Omega(P'(n))$ is a lower-bound to compute Bandersnatch, then $\Omega(P'(n) - P(n'))$ must be a lower-bound to compute Bo-billy.

The second argument is the idea we use to prove problems hard!

My Most Profound Tweet

An NP-completeness proof ensures that a dumb algorithm that is slow isn't a slow algorithm that is dumb.

What is a Problem?

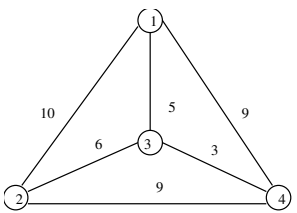
A *problem* is a general question, with parameters for the input and conditions on what is a satisfactory answer or solution.

Example: The Traveling Salesman

Problem: Given a weighted graph G, what tour $\{v_1, v_2, ..., v_n\}$ minimizes $\Sigma_{i=1}^{n-1} d[v_i, v_{i+1}] + d[v_n, v_1]$.

What is an Instance?

An instance is a problem with the input parameters specified.
TSP instance: $d[v_1, d_2] = 10$, $d[v_1, d_3] = 5$, $d[v_1, d_4] = 9$, $d[v_2, d_3] = 6$, $d[v_2, d_4] = 9$, $d[v_3, d_4] = 3$



Solution: $\{v_1, v_2, v_3, v_4\}$ cost= 27

Decision Problems

A problem with answers restricted to yes and no is called a *decision problem*.

Most interesting optimization problems can be phrased as decision problems which capture the essence of the computation.

For convenience, from now on we will talk *only* about decision problems.

The Traveling Salesman Decision Problem

Given a weighted graph G and integer k, does there exist a traveling salesman tour with cost $\leq k$?
Using binary search and the decision version of the problem we can find the optimal TSP solution.

Reductions

Reducing (tranforming) one algorithm problem A to another problem B is an argument that if you can figure out how to solve B then you can solve A.

We showed that many algorithm problems are reducible to sorting (e.g. element uniqueness, mode, etc.).

A computer scientist and an engineer wanted some tea...

Satisfiability

Consider the following logic problem:
Input: A set V of variables and a set of clauses C over V.
Problem: Does there exist a satisfying truth assignment for C?

Example 1: $V = v_1, v_2$ and $C = \{\{v_1, \overline{v}_2\}, \{\overline{v}_1, v_2\}\}$
A clause is satisfied when at least one literal in it is *true*. C is satisfied when $v_1 = v_2 =$true.

Not Satisfiable

Example 2: $V = v_1, v_2$,

$$C = \{\{v_1, v_2\}, \{v_1, \overline{v}_2\}, \{\overline{v}_1\}\}$$

Although you try, and you try, and you try and you try, you can get no satisfaction.

There is no satisfying assigment since v_1 must be false (third clause), so v_2 must be false (second clause), but then the first clause is unsatisfiable!

Satisfiability is Hard

Satisfiability is known/assumed to be a hard problem.
Every top-notch algorithm expert in the world has tried and failed to come up with a fast algorithm to test whether a given set of clauses is satisfiable.
Further, many strange and impossible-to-believe things have been shown to be true if someone in fact did find a fast satisfiability algorithm.

Input Encodings

Note that there are many possible ways to encode the input graph: adjacency matrices, edge lists, etc. All reasonable encodings will be within polynomial size of each other.

The fact that we can ignore minor differences in encoding is important. We are concerned with the difference between algorithms which are polynomial and exponential in the size of the input.

3-Satisfiability

Instance: A collection of clause C where each clause contains exactly 3 literals, boolean variable v.
Question: Is there a truth assignment to v so that each clause is satisfied?
Note that this is a more restricted problem than SAT. If 3-SAT is NP-complete, it implies SAT is NP-complete but not visa-versa, perhaps long clauses are what makes SAT difficult?! After all, 1-Sat is trivial!