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

Topic: Introduction to NP-Completeness

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.

Questions?

Topic: Problems and Reductions

What is a Problem?

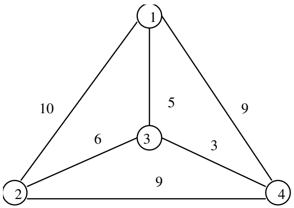
A *problem* is a general question, with parameters for the input and conditions on what is a satisfactory answer or solution.

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…

Questions?