Lecture 24: P and NP

COSC242: Algorithms and Data Structures

Brendan McCane

Department of Computer Science, University of Otago

Hard Problems

Pretty much all the problems and algorithms we've covered so far we have been able to solve in a polynomial number of steps (e.g. $O(n^2)$). Are there problems that are harder than polynomial ones?

There are problems that are definitely harder. For example:

- listing all possible subsets of a given set
- determining if there is a winning strategy in generalised games (chess, go etc) requires exponential time. Generalised games have arbitrary sized boards.

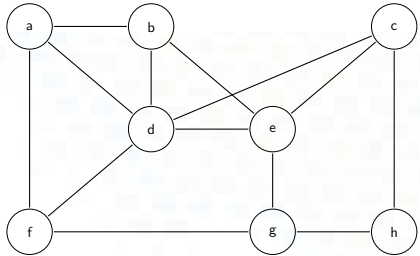
Easy or Hard?

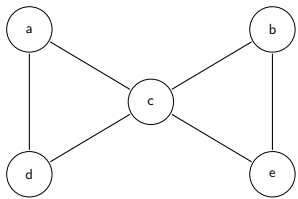
There are also a whole class of problems for which we don't know if any efficient algorithms exist. Here are some examples:

- finding a Hamilton cycle in a graph
- factoring an integer into a product of primes
- finding a tour of minimum distance that visits a set of cities once only. Called the travelling salesman problem
- 0-1 knapsack problem and bin-packing
- timetabling

Hamilton cycle problem

Let G be an undirected graph. A Hamilton cycle in G is a sequence of adjacent vertices and distinct edges in which every vertex of the graph appears exactly once (except that the first and last vertices are the same).





Solving the Hamilton Cycle Problem

The simplest algorithm to solve the HCP is a brute force algorithm:

1. list every permutation of the n vertices in G
2. for each permutation, check whether G has edges connecting the neighbouring vertices.

How many permutations of vertices are there?

What is the complexity of the check procedure in step 2?

What is the complexity of this algorithm?

The classes $\mathcal{P}$ and $\mathcal{NP}$

$\mathcal{P}$ is the class of all problems solvable by algorithms that are $O(n^k)$. That is, problems that are *polynomial*.

$\mathcal{NP}$ is the class of problems for which a proposed solution can be checked in polynomial time. You might think that $\mathcal{NP}$ stands for "not polynomial", but it doesn't. It actually stands for *non-deterministic polynomial*.

$\mathcal{P}$ an $\mathcal{NP}$

We know:

- $\mathcal{P} \subseteq \mathcal{NP}$
- $HCP \in \mathcal{NP}$

What about:

- is $HCP \in \mathcal{P}$?
- is $\mathcal{P} \subset \mathcal{NP}$?
- is $\mathcal{P} = \mathcal{NP}$?

HCP and $\mathcal{P}$

How could we show that $HCP \in \mathcal{P}$?

Just provide an algorithm that solves HCP in polynomial time.

No-one has been able to do that yet, so perhaps $HCP \notin \mathcal{P}$.

How could we show that $HCP \notin \mathcal{P}$?

We would have to show that no such algorithm exists that could run in polynomial time. This is much harder to prove, and no-one has been able to prove this either.

In fact, it is one of the Millenium Prize problems worth US$1,000,000 if you can solve it.

$\mathcal{NP}$-complete problems

HCP is one of several problems that have been shown to be $\mathcal{NP}$-complete.

A problem X is $\mathcal{NP}$-complete if $X \in \mathcal{NP}$ and every other problem in $\mathcal{NP}$ can be efficiently converted to an example of problem X, so that an algorithm solving X can easily be modified to solve every other problem in $\mathcal{NP}$.

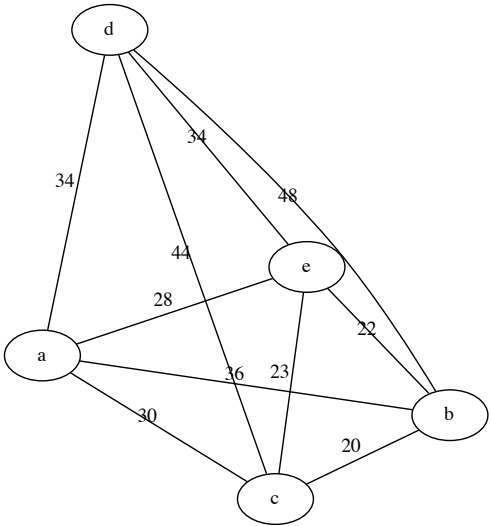
Now imagine if we could find an algorithm of polynomial complexity to solve HCP.

Then every problem in $\mathcal{NP}$ can be solved by this algorithm (after modifying it), and so every problem in $\mathcal{NP}$ would have a polynomial time solution. Thus it would follow that $\mathcal{P} = \mathcal{NP}$.

But all problems in $\mathcal{NP}$, like HCP, have stubbornly resisted all efforts to find polynomial time solutions. By now, everyone strongly suspects that HCP and the other $\mathcal{NP}$-complete problems are too hard to be solved by algorithms that are of less than exponential complexity.

Travelling Salesman Problem (TSP)

Given a set of cities, a salesman must travel to each city once, starting and ending at the same city. Which route is the shortest? E.g: starting at a, what is the shortest tour?



Brute Force

Brute force uses essentially the same algorithm as HCP:

1. Generate all possible tours (Hamilton cycles)
2. Return the solution with shortest distance.

Complexity?

Dynamic programming

We can do better by using a dynamic programming approach. Let $G = (V, E)$, and consider a start node s. Also, let's define the minimum path between two nodes starting at a and finishing at b and including all nodes in the set N as $m_{tour}(G, a, b, N)$.

The minimum tour of G, starting at a and finishing at b, consists of the following recurrence:

$$m_{tour}(G, a, b, N) = \min_{n \in N - a} [d(a, n) + m_{tour}(G, n, b, N - a)]$$

If we start and finish at s, we need to solve $m_{tour}(G, s, s, V - s)$. A naive recursive implementation will give us the $|V|!$ solution. If we memoise though, we can do a bit better.

Memoised complexity

How big does our memo array need to be? Here's the recurrence again:

$$m_{tour}(G, a, b, N) = \min_{n \in N-a} [d(a, n) + m_{tour}(G, n, b, N - a)]$$

The only things that change in the recurrence are the start node and the set of nodes to tour through.

At the first level of recursion, we look at all subsets of size $|V| - 1$, then at the second level all subsets of size $|V| - 2$ etc.

The memo array needs to be big enough to fit all possible subsets of the nodes in our graph, and that is $2^{|V|}$.

The complexity of the dynamic programming solution is actually $|V|^2 2^{|V|}$. Well, that's better than factorial, but it's still pretty bad. No algorithm faster than $O(2^{|V|})$ is known to exist.

Relevant parts of the texbook

Chapter 34 discusses the problem of $\mathcal{NP}$-completeness and talks about the classes $\mathcal{P}$ and $\mathcal{NP}$. The textbook delves into the topic in rather more detail than we do here, so if you're interested, then that's a good place to look.