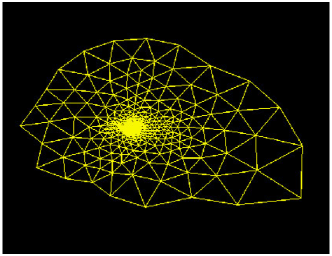
CI583: Data Structures and Operating Systems
What are NP-hard problems?



What is NP?

All of the algorithms we have looked at so far have had solutions in polynomial time or better.

The worst-case scenario is $O(n^m)$, where n is the size of the data.

So we can get a solution to a problem using one of these algorithms in a “reasonable” time.

This class of problems is called P (polynomial), and the problems in it are said to be tractable.

P and NP

This is in contrast to the class of problems for which there is no polynomial (or better) solution: NP (non-deterministic polynomial), the class of intractable problems.

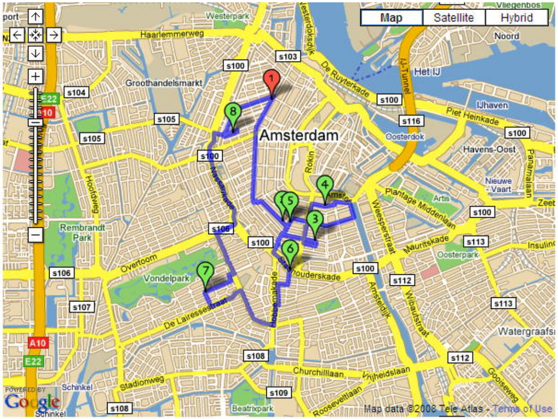
Algorithms known which can solve these problems tend to be of exponential ($O(2^n)$) or factorial ($O(n!)$) order.

If an algorithm is $O(2^n)$ then every time we add a single element to the input the time taken doubles.

Travelling salesman

As an example, consider the travelling salesman problem. We are given a set of cities and a cost of travelling between each of them. The goal is to visit each city once, ending up back at the city from which we began, and minimising the cost.

Google Maps Fastest Roundtrip Solver



Travelling salesman

This a problem that often needs to be solved in the real world – e.g. calculating the best route for a delivery van or how to efficiently send messages among nodes in a network.

If we have 8 cites, there are 40,320 orderings. A brute force solution is one that inspects every possibility.

For 10 cities there are 3,628,800 possibilities – to find the best route we have to examine them all.

For 15 cities, if we could do 100 route comparisons per second then it would take more than 400 years to find the answer by brute force.

P and NP

Thus, a deterministic algorithm (one that examines all possibilities) would take an extremely long time to complete.

To show that the problem is in NP we have to describe a two-step process for solving it – the first step generates a non-deterministic potential solution to the problem, the second step checks whether this is a true solution.

In our case, the first step would be to generate a list of the cities to visit in some random order. This step would run in $O(n)$. The second step would be to check the cost of the solution, also in $O(n)$. The point is that we don't know how many times we will need to repeat this.

NP-complete

We may be tempted to think that a problem is in NP just because we haven't come up with a polynomial algorithm yet.

The term NP-complete is used to describe the hardest problems in NP.

If we were ever to find a polynomial solution to an NP-complete problem then we could find a polynomial solution for all of them.

NP-complete

This is because a problem is NP-complete *if and only if every other problem in the class can be transformed into it*.

To show that problem A is NP-complete we need to show how to transform it into another NP-complete problem, B.

Because B is NP-complete, every problem in NP could be transformed into B.

P and NP

Typical NP problems:

1. Graph colouring: assign a “colour” to each node in a graph so that no adjacent nodes have the same colour. Applications include timetabling problems.
2. Bin packing: given a number of bins and a set of objects with varying sizes, what is the smallest number of bins we need to store all objects? This can be used to work out how to use materials efficiently, such as how to cut a piece of metal into smaller parts with minimal waste.
3. CNF-SAT problem: given a logical expression in *conjunctive normal form* (i.e. is a conjunction of disjunctive clauses), is there some assignment that makes the whole thing true?

P and NP

This sets the scene for one of the most famous question in CS: $P = NP$?

We know that P is a subset of NP – for every problem that has a polynomial solution (e.g. sorting) we could come up with a worse, non-deterministic solution. Given a problem in NP, X, for which there is no polynomial deterministic solution, we need to show that X is not in P.

All can say so far about whether X is also in P is that we *haven't found a deterministic solution yet*. From a common-sense point of view computer scientists are confident that $P \neq NP$, but the interesting thing is why this is so hard to prove. The search for a proof continues.