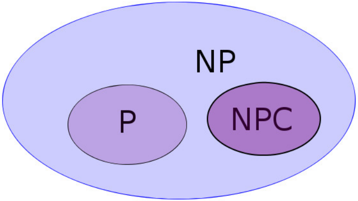
NP-Completeness

Dr Timothy Kimber

March 2015



What Problems Can We Solve?

We have seen a lot of fine algorithms, but this course is almost over. Looking ahead you might want to know:

- Is there a fast algorithm to solve every conceivable problem?
- Or are we sometimes faced with checking every possible solution?

Good question. What do you mean by fast?

- Fast is generally taken to mean polynomial time

There are various ways to justify this

- Exponential time is clearly too long
- If we have a $O(N^k)$ algorithm, k is usually small
- If we have a $O(N^k)$ algorithm, k usually improves with more research

Polynomial Time

We know that a polynomial time algorithm should:

- Run in $O(N^k)$ time for an input of size N

But what is an input of size N?

- So far I have assumed this is obvious
- Strictly, it is measured in terms of bits in the input data
- So, the input size is different depending on the way data is represented
- Assuming some sensible encoding in binary is OK

However, some algorithms that appear to be polynomial are not

Example (Prime Numbers)

To test whether some number N is prime, we can do:

- For $a = 2$ to $\sqrt{N}$
  - If N mod $a == 0$ then Return FALSE
- Return TRUE

Polynomial Time

Example (Prime Numbers)

- For $a = 2$ to $\sqrt{N}$
  - If N mod $a == 0$ then Return FALSE
- Return TRUE

The algorithm performs at most $\sqrt{N}$ divisions

- It appears to be sub-linear: $T(N) = O(\sqrt{N})$
- But the value of N is not the same as the size of the input
- The value of a B-bit input is $N \leq 2^B - 1$
- So, $T(B) = O(\sqrt{2^B})$, which is not polynomial

This is called a pseudo-polynomial algorithm

Polynomial Time Algorithms

Back to the question: what problems are solvable in polynomial time?

- We do not know
- There are polynomial algorithms for many problems
- There are also many with no known polynomial algorithm, but no proof that one does not exist

The question is usually studied for decision problems

- A decision problem has a yes/no answer
- Optimisation problems have a related decision form
- Shortest path becomes "is there a path of k or fewer edges"?
- The decision form is no harder than the optimisation form

Complexity Classes

Definition (P)

A decision problem is in the complexity class P if there is an algorithm that solves the problem in time $O(N^k)$, for some constant k, where N is the size of the input to the problem.

- If we have a p-time algorithm we know the problem is in P
- If we do not, the problem might be in P or might not

There is a related complexity class NP that is of interest

Definition (NP)

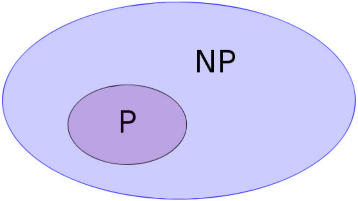
A decision problem is in the complexity class NP if there is an algorithm that, given a potential solution to the problem, can verify if the solution is correct in polynomial time.

NP Problems

Example

Given strings $T = \langle t_1, \dots t_N \rangle$ and $P = \langle p_1, \dots, p_M \rangle$, what are the shifts at which P occurs in T?

- Suppose we are told the shifts are $S = \langle 2, 5, 7 \rangle$
- The solution can be verified using the the algorithms we have seen
- Any problem with a p-time algorithm can be verified in p-time
- So $P \subseteq NP$

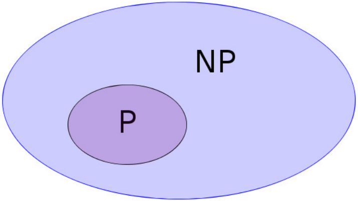


NP Problems

Example (Hamiltonian Path)

Given an undirected graph G, is there path in G that includes every vertex exactly once?

- There is no known polynomial algorithm to solve this problem
- Given a path it is simple to verify in polynomial time
- Follow the path and confirm if it includes all vertices once
- Just about all (all?) decision problems are NP



The Big Question

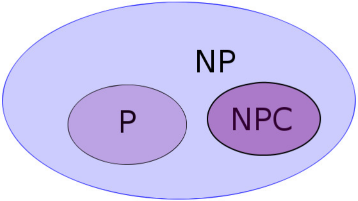
P = NP?

Does the fact that NP problems can be verified in p-time suggest that they can also be solved in p-time? Does P = NP?

- A huge open question in computing
- One of the Millenium Prize problems: www.claymath.org
- Most CS researchers would say no

The evidence comes from the NP-Complete (NPC) class of problems

- These problems are in NP: the consensus is that they are not in P

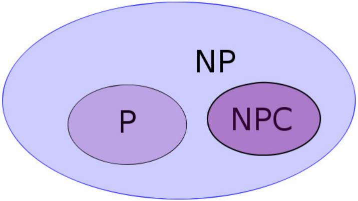


NP-Completeness

Definition (NPC)

A decision problem D is in the complexity class NPC if (i) it is in NP, and (ii) every other problem in NP can be reduced to D in polynomial time.

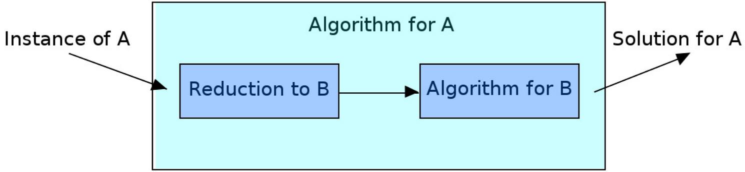
- The second condition alone means the problem is NP-Hard
- NP-Complete problems are at least as hard as any in NP
- The belief is that only the "easy" problems can be solved in p-time



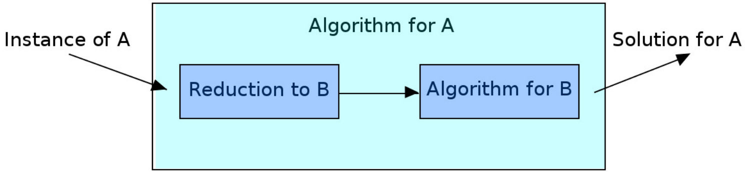
Reductions

A problem A is reduced to another problem B if every instance of A can be expressed as an instance of B

- If we have an algorithm for B this provides an algorithm for A
- Have just seen this with shortest paths and linear programming
- Another e.g. MST solved by sorting
- If both parts are polynomial, the algorithm for A is polynomial



Reductions



The reduction also provides a bound on the difficulty of B

- Using the reduction we can solve A just as fast as B
- B must be at least as hard as A
- If this is a poor choice of method for A, A could be easier than B

So, NP-Complete problems are at least as hard as every other problem in NP, because every problem in NP can be reduced to them.

NP-Complete Problems

It is not simple to prove that every problem reduces to a single one

- But after the first one it gets a lot simpler!
- Just have to prove that the first one reduces to another

The Cook–Levin theorem (1971) establishes that the boolean satisfiability problem (SAT) is NP-Complete.

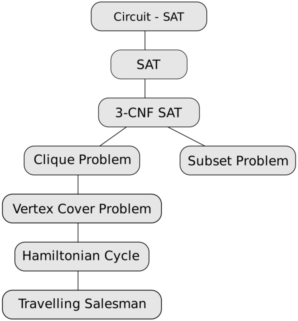
SAT

Given a set of propositional logic formulas, is there an assignment of the values *true* and *false* that makes all the formulas true?

- A solution can be verified using truth tables, so SAT is in NP
- See text books for the reduction!

NP-Complete Problems

Starting with SAT Robin Karp (1972) showed a further 21 problems to be NP-Complete, including



- Thousands more are found every year

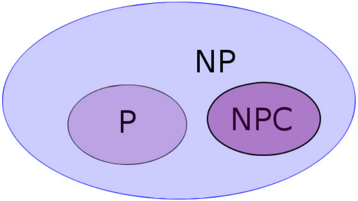
NPC and NP

Why are NP-Complete problems evidence that P $\neq$ NP?

NPC and NP

If a polynomial time algorithm exists for any NP-Complete problem then every problem in NP is solvable in polynomial time.

- None has ever been discovered
- This is taken as strong evidence that P and NPC are disjoint
- Which implies P $\neq$ NP



Why NP?

You might think NP stands for "non-polynomial". In fact it is Nondeterministic Polynomial. The original definition is

Definition (NP)

A decision problem is in the complexity class NP if there is a nondeterministic algorithm that solves the problem in polynomial time.

- A nondeterminstic algorithm starts by "guessing" the correct solution!
- It then proceeds to verify it in polynomial time

This captures the idea that some problems require a spark of inspiration to solve them

- They cannot be conquered by a plodding machine alone
- If P = NP then this is not true