Complexity Theory

Decision Problem: Problem with a YES/NO ~~an~~ output.

Example: ~~Given~~ Does a graph G, ~~and ver~~ contain a path between vertices s and t?

Most problems we've seen so far are **optimisation problems**. These can easily be changed into decision problems.

Example: "Find the shortest path between s and t in G."
$\Rightarrow$ "Does G contain an s-t-path of length at most k?"

If we have an algorithm that solves the optimisation problem, solving the decision problem is easy.

If we have an algorithm that solves the decision problem, we can use binary search to solve the optimisation problem.

Complexity Classes

- P = set of all decision problems that can be solved in polynomial time.

Formally, a problem is in P iff there exists an algorithm A that, gives the right answer for every input I and runs in $O(|I|^c)$ time, for some constant c.

To prove that a problem is in P, you give a polynomial-time algorithm that solves it.

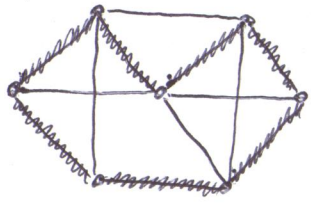
Example: Does G contain an s-t-path?

- NP = set of all decision problems whose YES-solutions can be verified in polynomial time.

Formally, a problem is in NP iff there exists an algorithm V such that, for every input I, there exists a certificate C of size $O(|I|^c)$ and V(I,C) returns YES in $O(|I|^c)$ time if and only if the answer is YES.

To prove that a problem is in NP:

- Describe the certificate for YES-answers.
- Give a polynomial-time algorithm that verifies the certificate.



Example: Hamiltonian Cycle

Input: Graph G

Output: YES if G contains a cycle that visits every vertex exactly once, NO o.w.

Claim: HamiltonianCycle is in NP.

Proof: Certificate = $[v_1, v_2, \ldots, v_n]$ such that $v_1, v_2, \ldots, v_n, v_1$ is a ~~cycle that visits~~ Hamiltonian cycle.

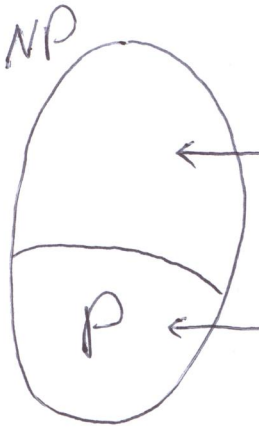
Verification algorithm:

- check that every vertex of G occurs ~~in~~ exactly once in the certificate
- check that it is a cycle in G:
  (v_i, v_{i+1}) is an edge for all $1 \leq i \leq n$.

This can be done in $O(n^2)$ time. □

Claim: $P \subseteq NP$

Proof: Let X be a problem in P. Then there is a polynomial-time algorithm A that solves X. We need to show that X is in NP. Certificate = $\emptyset$. Verification algorithm = A. The verification algorithm runs in polynomial time and returns YES if and only if the answer is YES. Thus, X is in NP. □



Big question:
Is $P = NP$?

Most believe that $P \neq NP$.

If we want to prove that $P \neq NP$, we have to show that there exists a problem X such that X is in NP, but X is not in P.

This X must be "very difficult".

⇒ Look at the "most difficult" problems in NP.

How can we compare the difficulty of problems?

⇒ Reductions!