Class 33

P, NP, AND NP-COMPLETE PROBLEMS

Review: Problem Reduction

Suppose problem X is polynomially reducable to problem Y.

X→Y

- What does this mean?
- What is the implication…
  - about the inability to efficiently solve one of the problems?
  - about efficiently solving one of the problems?

Problem Types

Optimization problem: find a solution that maximizes or minimizes some objective function

Decision problem: answer yes/no to a question

Many problems have decision and optimization versions.

- E.g., traveling salesman problem:
  - *optimization*: find Hamiltonian cycle of minimum length.
  - *decision*: given m, does there exist a Hamiltonian cycle of length $\leq m$?
- Decision version is usually easier, but only "polynomially" so.

Decision problems are more convenient for formal investigation of complexity.

Class *P*

The class of decision problems with complexity $O(p(n))$ for some polynomial function $p(n)$ of the input size n

A problem in P is called *tractable.* Problems that take exponential time are *intractable*.

Why the polynomial vs. exponential distinction?

- For exponential problems, even medium-sized inputs cannot be solved in practical time
- In practice, the degree of the polynomial is small, like ≤ 3, and the constants are reasonable
- P closed under composition, products; not true for exponential

Liars and such

Attempt to determine whether the following sentence is true:

- I am a liar.

Are All Decision Problems in *P*?

No, there are some problems that cannot be solved by any algorithm.

They are undecidable.

The *halting problem*: Given a computer program and an input to it, determine whether the program will halt on that input or not.

- Let *A* be a program that solves the halting problem.
- Let *P* be an arbitrary program
- Let *I* be the given input.

$$A(P, I) = \begin{cases} 1, & \text{if program } P \text{ halts on input } I; \\ 0, & \text{if program } P \text{ does not halt on input } I. \end{cases}$$

Are All Decision Problems in *P*?

Now let's do some magic.

$$A(P, I) = \begin{cases} 1, & \text{if program } P \text{ halts on input } I; \\ 0, & \text{if program } P \text{ does not halt on input } I. \end{cases}$$

Let's create a new program *Q* that takes a program *P* and tells whether it terminates on itself as input.

Just to ensure we properly twist your brains, suppose *Q*:

- enters an infinite loop after it determined that P halts on itself
- stops after it determined that P does not halt on itself.

$$Q(P) = \begin{cases} \text{halts}, & \text{if } A(P, P) = 0, \text{ i.e., if program } P \text{ does not halt on input } P; \\ \text{does not halt}, & \text{if } A(P, P) = 1, \text{ i.e., if program } P \text{ halts on input } P. \end{cases}$$

Undecidable Problems

Now let's replace *P* with *Q* in:

$$Q(P) = \begin{cases} \text{halts}, & \text{if } A(P, P) = 0, \text{ i.e., if program } P \text{ does not halt on input } P; \\ \text{does not halt}, & \text{if } A(P, P) = 1, \text{ i.e., if program } P \text{ halts on input } P. \end{cases}$$

To get:

$$Q(Q) = \begin{cases} \text{halts}, & \text{if } A(Q, Q) = 0, \text{ i.e., if program } Q \text{ does not halt on input } Q; \\ \text{does not halt}, & \text{if } A(Q, Q) = 1, \text{ i.e., if program } Q \text{ halts on input } Q. \end{cases}$$

Just like we were not able to determine whether I was lying or not, we cannot provide an answer to the last decision problem.

It is undecidable.

Class *NP*

NP stands for *nondeterministic polynomial*

NP is the class of decision problems that are solvable by a *nondeterministic polynomial algorithm*

Another way to look at this class is that it is those problems for which a proposed solutions can be **verified** in polynomial time.

Informally: "efficiently verifiable" problems

Example: CNF-SAT

The problem of determining whether a boolean expression in its conjunctive normal form (**CNF**) is **sat**isfiable.

In other words, are there truth assignments to its variables that make the entire expression true?

Conjunctive normal form consists of a conjunct of terms that are connected by disjunctions.

Example: $(A \lor \neg B \lor \neg C) \land (A \lor B) \land (\neg B \lor \neg D \lor E) \land (\neg D \lor \neg E)$

Example: CNF-SAT

Example: (A ∨ ¬B ∨ ¬C) ∧ (A ∨ B) ∧ (¬B ∨ ¬D ∨ E) ∧ (¬D ∨ ¬E)

This problem is in *NP*.

Nondeterministic algorithm:

1. Guess truth assignment
2. Substitute the values into the CNF formula to see if it evaluates to true

Checking phase: O(n), so it is nondeterministic polynomial solvable.

Problems that are in *NP*

Hamiltonian circuit problem. Determine whether a give graph has a Hamiltonian circuit – a path that starts and ends at the same vertex and passes through all the other vertices exactly once.

Travelling Salesperson problem. Find the shortest tour through n cities with known positive integer distances between them. In other words, find the shortest Hamiltonian circuit in a complete graph with positive integer weights.

Knapsack. Find the most valuable subset of n items of given positive integer weights and values that fin into a knapsack of a given positive integer capacity.

Graph coloring: can vertices of a graph be validly colored using ≤ k colors so that no two adjacent vertices are assigned the same number?

P vs. *NP*

All the problems in *P* can also be solved without guessing.

Hence, we have:

$$P \subseteq NP$$

Big question: $P = NP$ or $P \subset NP$?

NP-hard, *NP*-complete

Decision problem *D* is *NP*-hard if it's

Informally: "at least as hard as all *NP* problems"

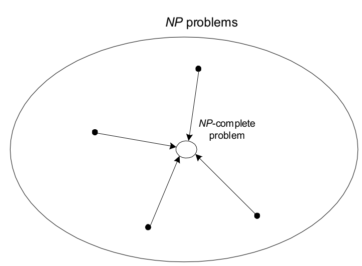
Formally: every problem in *NP* is polynomial-time reducible to *D.*

A problem is *NP*-complete if it's

in *NP* and is *NP*-hard.

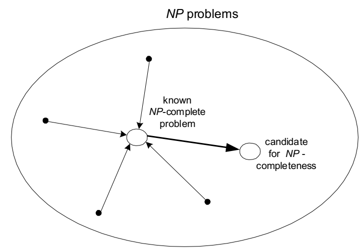
Informally:

"The hardest problems in *NP*"



Proving *NP*-completeness

Other *NP*-complete problems obtained through polynomial-time reductions from a known *NP*-complete problem.

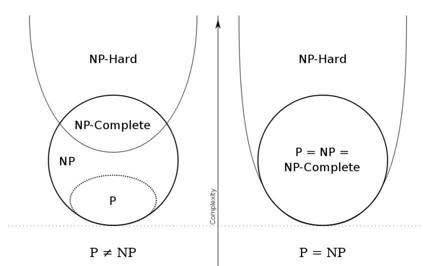


Does *P* = *NP*?

P = *NP* would imply that every problem in *NP,* including all *NP*-complete problems, could be solved in polynomial time

If a polynomial-time algorithm for just one *NP*-complete problem is discovered, then every problem in *NP* can be solved in polynomial time, i.e., *P* = *NP*

Almost everyone believes that $P \neq NP$, i.e. $P \subsetneq NP$



Interesting Problem Pairs

Eulerian circuit (existence of a walk visiting every edge of a graph exactly once) is in P.

Hamiltonian cycle (existence of a walk visiting every vertex of a graph exactly once) is NP-complete.

Decision version of shortest path in a graph is in P.

Decision version of longest path in a graph is NP-complete.

2-CNF-SAT (each clause has ≤ 2 literals) is in P.

3-CNF-SAT (each clause has ≤ 3 literals) is NP-complete.