Classifying Problem Complexity

Is the problem ***tractable***, i.e., is there a polynomial-time ($O(p(n))$) algorithm that solves it?

Possible answers:

- yes (give examples)
- no
  - because it's been proved that no algorithm exists at all (e.g., Turing's ***halting problem***)
  - because it's been be proved that any algorithm takes exponential time
- unknown

 1

Problem Types: Optimization and Decision

- ***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*: find Hamiltonian cycle of length $\leq m$

Decision problems are more convenient for formal investigation of their complexity.

 2

Class *P*

P: the class of decision problems that are solvable in $O(p(n))$ time, where $p(n)$ is a polynomial of problem's input size n

Examples:

- searching
- element uniqueness
- graph connectivity
- graph acyclicity
- primality testing (finally proved in 2002)

Class *NP*

NP (*nondeterministic polynomial*): class of decision problems whose proposed solutions can be verified in polynomial time = solvable by a *nondeterministic polynomial algorithm*

A *nondeterministic polynomial algorithm* is an abstract two-stage procedure that:

- generates a random string purported to solve the problem
- checks whether this solution is correct in polynomial time

By definition, it solves the problem if it's capable of generating and verifying a solution on one of its tries

Why this definition?

- led to development of the rich theory called "computational complexity"

Example: CNF satisfiability

Problem: Is a boolean expression in its conjunctive normal form (CNF) satisfiable, i.e., are there values of its variables that makes it true?

This problem is in *NP*. Nondeterministic algorithm:
- **Guess truth assignment**
- **Substitute the values into the CNF formula to see if it evaluates to true**

Example: (A | ¬B | ¬C) & (A | B) & (¬B | ¬D | E) & (¬D | ¬E)

Truth assignments:

A	B	C	D	E
0	0	0	0	0
.	.	.		
1	1	1	1	1

Checking phase: O(*n*)

What problems are in *NP*?

- **Hamiltonian circuit existence**
- **Partition problem: Is it possible to partition a set of *n* integers into two disjoint subsets with the same sum?**
- **Decision versions of TSP, knapsack problem, graph coloring, and many other combinatorial optimization problems. (Few exceptions include: MST, shortest paths)**
- **All the problems in *P* can also be solved in this manner (no guessing is necessary), so we have:**

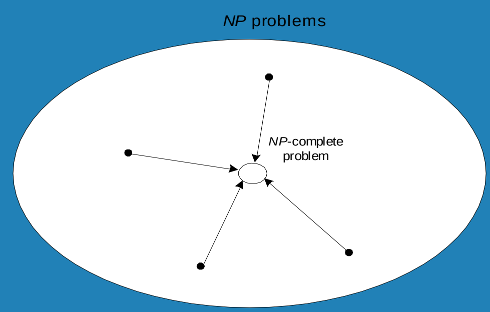
$$P \subseteq NP$$

- **Big question: $P = NP$?**

NP-Complete Problems

A decision problem *D* is NP-complete if it's as hard as any problem in *NP*, i.e.,

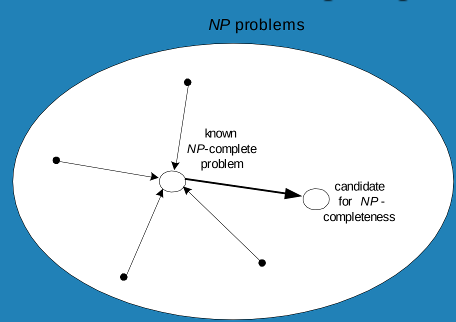
- *D* is in *NP*
- every problem in *NP* is polynomial-time reducible to *D*



Cook's theorem (1971): CNF-sat is *NP*-complete

NP-Complete Problems (cont.)

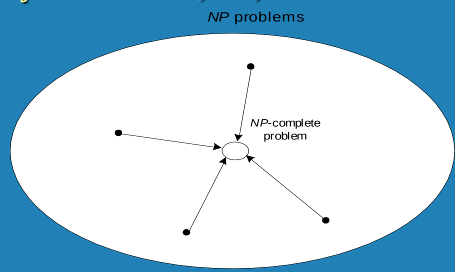
Other *NP*-complete problems obtained through polynomial-time reductions from a known *NP*-complete problem



Examples: TSP, knapsack, partition, graph-coloring and hundreds of other problems of combinatorial nature

$P = NP$? Dilemma Revisited

- $P = NP$ 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., $P = NP$



- Most but not all researchers believe that $P \neq NP$, i.e. P is a proper subset of NP

 9