NP-Completeness

Chapter 6

Outline

- NP-completeness and the classes P and NP
- Polynomial time verification
- NP-completeness and reducibility
- NP-complete problems

Efficient algorithm

Classical definition:

An algorithm is **efficient** iff it runs in polynomial time: $O(n^c)$ for some constant c, where n is the size of the input to the problem

Runtimes of “efficient” algorithms: $O(n)$ $O(n \log n)$ $O(n^3 \log_2 n)$

Runtimes of “inefficient” algorithms: $O(2^n)$ $O(n!)$

Tractability and Intractability

A problem is called **tractable** iff there is an efficient algorithm that solves it.

A problem is called **intractable** iff there is no efficient algorithm that solves it.

Deterministic and non-deterministic algorithms

Given a particular input, a **deterministic algorithm** will always produce the same output.

Even for the same input, a **nondeterministic algorithm** can exhibit different behaviors on different runs.

Such algorithms have multiple choices in some points during its control flow. The choice is not determined by the input value and the state; instead an arbitrary choice among the possibilities can be made in a given run of the program, e.g. due to a *race condition,* or a random number generator etc.

The class P

- A **decision problem** is a problem with a yes/no answer.
- **P** is the class of decision problems that can be solved by a polynomial-time algorithm.
- In other words, given an instance of the problem of class P, the answer yes or no can be decided in polynomial time.
- **Example**: Is there a path from node x to node y in a graph G? This problem can be solved in linear time using depth-first search.
- *Note: To be in P, a problem only needs an algorithm that runs in time less than some polynomial. Hence, problems that can be solved in n log n time are in P.*

The class NP

The **class NP** (**N**on-deterministic **P**olynomial time) consists of those problems that are “verifiable” in polynomial time.

What do we mean by a **problem being verifiable**?

A. if someone gives us an instance of the problem and a certificate to the answer being yes, we can check that it is correct in polynomial time.
   In other words, if we were given the solution, we can verify a Yes answer quickly.

Example: Hamiltonian cycle problem

The class NP

Hamiltonian cycle problem:

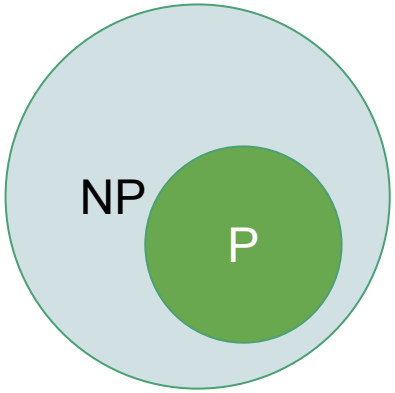
A **hamiltonian cycle** of a directed graph G = (V, E) is a simple cycle that contains each vertex in V.

There is no polynomial-time algorithm for determining whether a directed graph has a hamiltonian cycle.

However, if we are given a certificate, which would be a sequence $\langle v_1, v_2, \ldots, v_{|V|} \rangle$ of |V| vertices, we could easily check in polynomial time that $(v_i, v_{i+1}) \in E$ for i = 1, 2, 3, .., |V| - 1 and that $(v_{|V|}, v_1) \in E$ as well.

The classes P and NP

If a problem is in P, then it is also in NP.



However, **the open question is whether or not P is a proper subset of NP** (i.e. if P = NP, P would not be a proper subset of NP).

If the solution to a problem is easy to check for correctness, must the problem be easy to solve?

This is one of the seven Millenium Prize problems, selected by the Clay Mathematics Institute

P = NP ?

An answer to the P = NP question would determine whether problems that can be verified in polynomial time can also be solved in polynomial time.

If it turned out that P ≠ NP, it would mean that there are problems in NP that are harder to compute than to verify.

NP-completeness

A problem M is said to be **NP-complete** if it is in NP, and for every problem L in NP, there is a **polytime reduction** from L to M.

Reduction:

A problem Q can be **reduced to** another problem Q' if any instance of Q can be **"easily rephrased"** as an instance of Q', the solution to which provides a solution to the instance of Q.

If a problem Q reduces to another problem P in polynomial time (i.e. *polytime reduction*), we write $Q \leq_P P$

NP-completeness

Example: SAT: The satisfiability problem

Given a boolean formula, for example, $(x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee x_5) \wedge x_6$, is it possible to assign the inputs $x_1, x_2, \ldots, x_n$ such that the formula evaluates to true?

Cook's theorem states that this problem is NP-complete (proof not in the scope of this course)

NP-completeness

Example: 3-SAT. (aka 3-CNF-SAT), 3 CNF satisfiability problem

A boolean formula is in **conjunctive normal form (CNF)** if it is a conjunction (AND) of several clauses, each of which is the disjunction (OR) of several literals, each of which is either a variable or its negation. Example: $(A \vee B \vee C) \wedge (A \vee \neg C \vee D)$

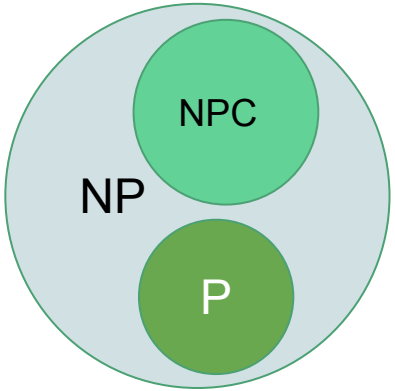
The problem: Given a 3CNF formula, is there an assignment to the variables that makes the formula evaluate to true?

Karp showed the problem 3SAT to be NP-complete by showing how to reduce (in polynomial time) any instance of SAT to an equivalent instance of 3SAT.

Example: Reduction

3SAT ≤ 0/1 Knapsack

NP-completeness



A problem M is said to be **NP-complete** if

1. $M \in NP$, and
2. $L \leq_P M$ for every $L \in NP$

Fundamental property:

If M has a polytime algorithm, then so does L. That means, if anyone finds a polynomial algorithm for even one NP-complete problem, then that would imply a polynomial algorithm for every NP-complete problem, i.e., if M is in P, then every L in NP is also in P, or “P = NP.”

As of now, there are no known polynomial-time algorithms for any NP-complete problem.

NP-hard

Recall: A problem M is said to be **NP-complete** if

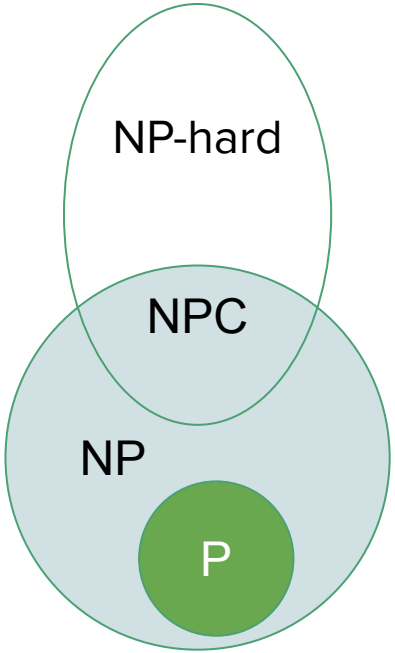
1. $M \in NP$, and
2. **$L \leq_P M$ for every $L \in NP$**

If a problem M' satisfies property 2, but not necessarily property 1, we say that M' is NP-hard.

In other words, NP-hard problems are those problems that are at least as hard as the NP-complete problems.

Note that NP-hard problems do not have to be in NP, and they do not have to be decision problems.

P, NP, NP-Complete, and NP-hard



Note that these classes characterize a problem, not an algorithm

NP-complete problems

The clique problem:

- A clique is a complete subgraph of G.
- The size of a clique is the number of vertices it contains.
- The clique problem is the optimization problem of finding a clique of maximum size in a graph
- As a decision problem, we ask simply whether a clique of a given size k exists in the graph.
  CLIQUE = {(G, k): G is a graph containing a clique of size k}
- To prove that the clique problem is NP-complete, we show that CLIQUE $\in$ NP, and a known NP-complete problem can be reduced to CLIQUE, e.g.
  3-CNF-SAT $\leq_P$ CLIQUE

NP-complete problems

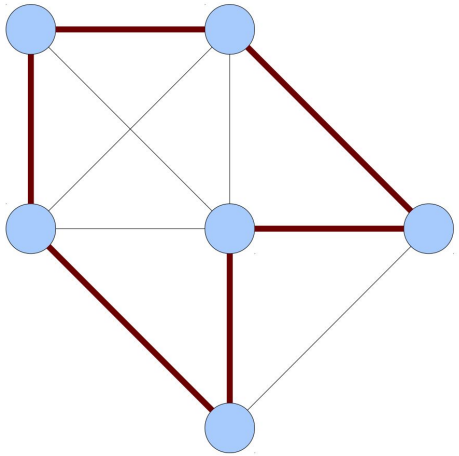
The vertex-cover problem:

- Given an undirected graph G(V, E), $S \subseteq E$ is a vertex cover if every vertex is incident on a vertex in S.
- The vertex-cover problem is to find a vertex cover of minimum size in a given graph.
  VERTEX-COVER = {(G, k): graph G has a vertex cover of size k}
- To prove that this problem is NP-complete, we show that VERTEX-COVER $\in$ NP and CLIQUE $\leq_P$ VERTEX-COVER

NP-complete problems

The hamiltonian-cycle problem:

- The problem is to determine whether a directed graph has a hamiltonian cycle
  HAM-CYCLE = {G: G is a hamiltonian graph}



NP-complete problems

The traveling-salesman problem:

- We are given a complete undirected graph G = (V, E) that has a nonnegative integer cost c(u, v) associated with each edge (u, v) ∈ E, and we must find a hamiltonian cycle (a tour) of G with minimum cost.