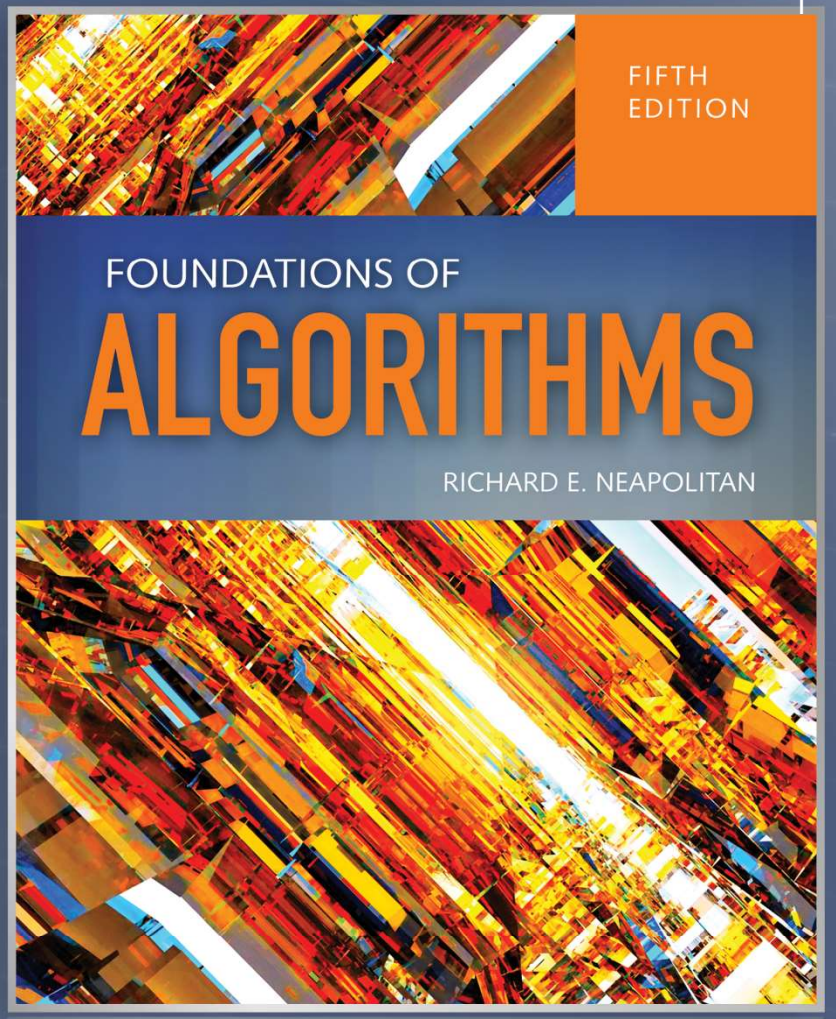
Computational Complexity and Intractability: An Introduction to the Theory of NP

Chapter 9



Objectives

- Classify problems as tractable or intractable
- Define decision problems
- Define the class P
- Define nondeterministic algorithms
- Define the class NP
- Define polynomial transformations
- Define the class of NP-Complete

Input Size and Time Complexity

- Time complexity of algorithms:
  - Polynomial time (efficient) vs. Exponential time (inefficient)

$f(n)$	$n = 10$	30	50
n	0.00001 sec	0.00003 sec	0.00005 sec
n^5	0.1 sec	24.3 sec	5.2 mins
2^n	0.001 sec	17.9 mins	35.7 yrs

“Hard” and “Easy” Problems

- “Easy” problems can be solved by polynomial time algorithms
  - Searching problem, sorting, Dijkstra’s algorithm, matrix multiplication, all pairs shortest path
- “Hard” problems cannot be solved by polynomial time algorithms
  - 0/1 knapsack, traveling salesman
- Sometimes the dividing line between “easy” and “hard” problems is a fine one. For example,
  - Find the shortest path in a graph from X to Y (easy)
  - Find the longest path (with no cycles) in a graph from X to Y (hard)

“Hard” and “Easy” Problems

- Motivation: is it possible to efficiently solve “hard” problems? Efficiently solve means polynomial time solutions.
- Some problems have been proved that no efficient algorithms for them. For example, print all permutation of a number n.
- However, many problems we cannot prove there exists no efficient algorithms, and at the same time, we cannot find one either.

Traveling Salesperson Problem

- No algorithm has ever been developed with a Worst-case time complexity better than exponential
- It has never been proven that such an algorithm is not possible
- NP-Complete

Intractability

- Dictionary Definition of intractable: “difficult to treat or work.”
- Computer Science: problem is intractable if a computer has difficulty solving it

Tractable

- A problem is tractable if there exists a polynomial-bound algorithm that solves it.
- Worst-case growth rate can be bounded by a polynomial
- Function of its input size
- $P(n) = a_n n^k + \ldots + a_1 n + a_0$ where k is a constant
- $P(n)$ is $\theta(n^k)$
- $n\lg n$ not a polynomial
  - $n\lg n < n^2$ bound by a polynomial

Intractable

- "Difficult to treat or work"
- A problem in CS is intractable if a computer has difficulty solving it
- A problem is intractable if it is not tractable
- Any algorithm with a growth rate not bounded by a polynomial
- c^n, $c^{.01n}$, $n^{\log n}$, $n!$, etc.
- Property of the problem not the algorithm

Three General Categories of Problems

1. Problems for which polynomial-time algorithms have been found
2. Problems that have been proven to be intractable
3. Problems that have not been proven to be intractable, but for which polynomial-time algorithms have never been found

Polynomial-time Algorithms

- $\Theta(n \lg n)$ for sorting
- $\Theta(\lg n)$ for searching
- $\Theta(n^3)$ for chained-matrix multiplication

Proven to be Intractable

- Unrealistic definition of the Problem (List all permutations of n numbers)
- Towers of Hanoi
- Un-Decidable problems: The Halting Problem (proven un-decidable by Alan Turing).
- Decidable intractable problems: researchers have shown some problems from automata and mathematical logic intractable

Not proven to be intractable no existing polynomial time algorithm

- Traveling salesperson
- 0-1 Knapsack
- Graph coloring
- Sum of subsets

Define

- Decision problems
- The class P
- Nondeterministic algorithms
- The class NP
- Polynomial transformations
- The class of NP-Complete

Decision problem

- Problem where the output is a simple “yes” or “no”
- Theory of NP-completeness is developed by restricting problems to decision problems
- Optimization problems can be transformed into decision problems
- Optimization problems are at least as hard as the associated decision problem
- If polynomial-time algorithm for the optimization problem is found, we would have a polynomial-time algorithm for the corresponding decision problem

Decision Problems

- Traveling Salesperson
  - For a given positive number d, is there a tour having length <= d?
- 0-1 Knapsack
  - For a given profit P, is it possible to load the knapsack such that total weight <= W?

More Examples

- Graph-Coloring optimization problem is to determine the minimum number of colors needed to color a graph so that no two adjacent vertices are colored the same color.
- Decision problem?

- A clique in an undirected graph G=(V, E) is a subset W of V such that each vertex in W is adjacent to all the other vertices in W. A maximal clique is a clique of maximal size.
- The optimization problem is to determine the size of a maximal clique for a given graph.
- Decision problem?

Class P

- The set of all decision problems that can be solved by polynomial-time algorithms
- Decision versions of searching, shortest path, spanning tree, etc. belong to P
- Do problems such as traveling salesperson and 0-1 Knapsack (no polynomial-time algorithm has been found), etc., belong to P?
  - No one knows
  - To know a decision problem is not in P, it must be proven it is not possible to develop a polynomial-time algorithm to solve it

Nondeterministic Algorithms – consist of 2 phases

1. Nondeterministic phase – Guessing Phase: given an instance of a problem, a solution is guessed (represented by some string s); We call it nondeterminisitic because unique step-by-step instructions are not specified
2. Deterministic phase – Verification Phase

Deterministic Phase – Verification Phase

- Input
  - Instance of the problem
  - String s: the guess
- Phase proceeds in an ordinary, deterministic manner:
  - Eventually halts with an answer yes – the guess, s, is a solution to the problem
  - Eventually halts with an answer no – the guess, s, is not a solution to the problem
  - Continues executing for ever

Example (Textbook P 409)

Polynomial-time Nondeterministic Algorithm (NDA)

- A nondeterministic algorithm whose verification stage is a polynomial-time algorithm

Class NP

- The set of all decision problems that can be solved by polynomial-time nondeterministic algorithms
- Nondeterministic polynomial
- For a problem to be in NP, there must be an algorithm that does the verification in polynomial time
- Traveling salesperson decision problem belongs to NP
  - Show a guess, s, length polynomial bounded
  - Yes answer verified in a polynomial number of steps

Suppose answer yes for a given instance of traveling sales person

- You may guess all (n-1)!tours before guessing the tour with the yes answer
- Polynomial-time verifiability, not solvability
- The guess for that tour is done in polynomial time
- The verification of the guess is done in polynomial time
- Branch and bound probably produces a better solution
- Purpose: problem classification

Example: CNF

- CNF – Conjunctive Normal Form
- Logical (Boolean) variable: can have one of two values: TRUE or FALSE
- Literal: logical variable or the negation of a logical variable
  - x is a logical variable
  - ¬x is the negation of a logical variable

CNF

- A clause is a disjunction of literals (e.g. $p \vee q \vee s$)
- A logical expression is in Conjunctive Normal Form if it is a conjunction of disjunctive clauses
  - $(p \vee q \vee s) \wedge (\neg q \vee r) \wedge (\neg p \vee r) \wedge (\neg r \vee s) \wedge (\neg p \vee \neg s \vee \neg q)$

CNF-Satisfiability Decision Problem

- Is there a truth assignment for the variables of a CNF expression which evaluates to true?
- For the answer to be yes, each clause must evaluate to TRUE
- Assume n variables, 2^n rows in the truth table
- Easy to write a polynomial-time algorithm takes as input a logical CNF expression and a set of truth assignments to the variables and verifies whether it evaluates to TRUE

Example

$$W=(x_1 \wedge x_2) \vee (x_3 \wedge \overline{x_4})$$

This is a positive instance with a truth assignment as follows:

$$x_1 \leftarrow T, \quad x_2 \leftarrow T, \quad x_3 \leftarrow T, \quad x_4 \leftarrow F$$

$$W=x_1 \wedge \overline{x_1}$$

This is a negative instance since W can never be satisfied.

$(x_1 \vee x_2) \wedge (x_1 \vee \overline{x}_2) \wedge (\overline{x}_1)$ Positive? Or negative?

Show that CNF Satisfiability is NP

- First, we can nondeterministicly guess a truth assignment for all variables. It takes O(n) for n variables.
- Second, we verify if every clause is evaluated to be true given the assignment. It takes O(m) for m clauses. Thus, polynomial time verification.

In Class Exercise

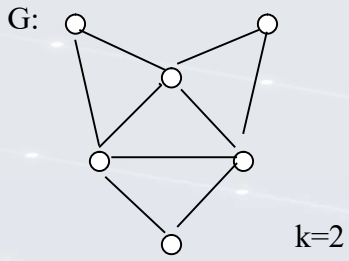
Dominating Set

Instance: A finite undirected graph G and a positive integer k.

Question: Are there at most k vertices such that all vertices in G are adjacent to one of them?

(a) Show that Dominating Set problem $\in$ NP.

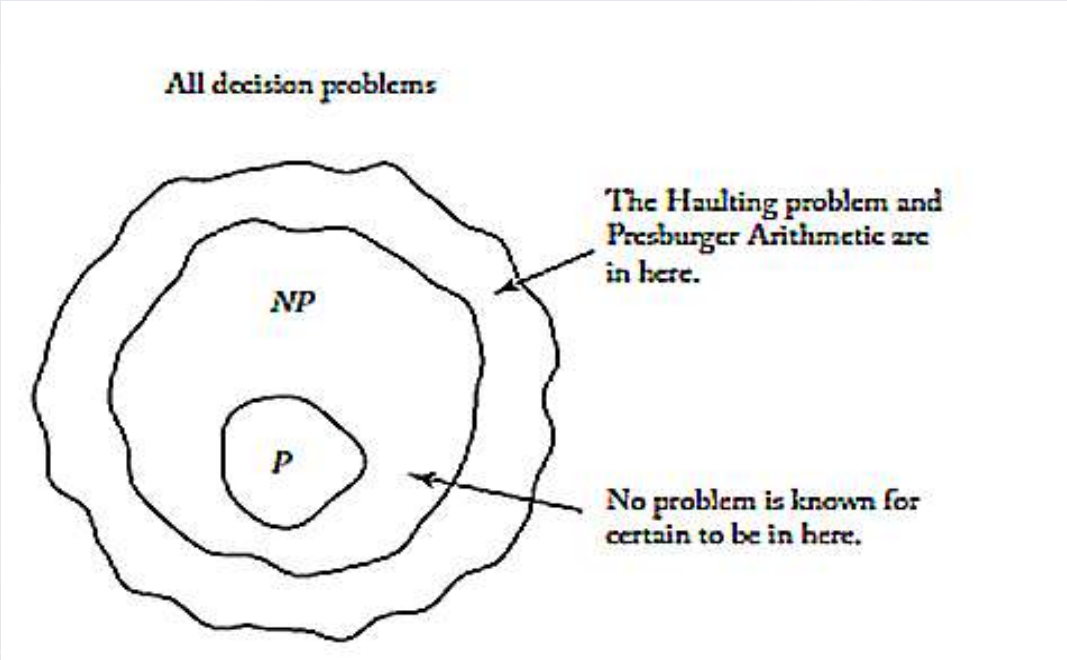
(b) Is the following instance positive or negative? Why?



Is P ⊆ NP?

- It has not been proven that there is a problem in NP that is not in P
- NP-P may be empty
- P=NP? One of the most important questions in CS
- To show P!=NP, find a problem in NP that is not in P
- To show P = NP, find polynomial-time algorithm for each problem in NP

Figure 9.3



CNF Satisfiability Decision Problem

- Problem is in NP
- No polynomial-time algorithm for it has been found
- Cook proved that if CNF-Satisfiability is in P, P = NP

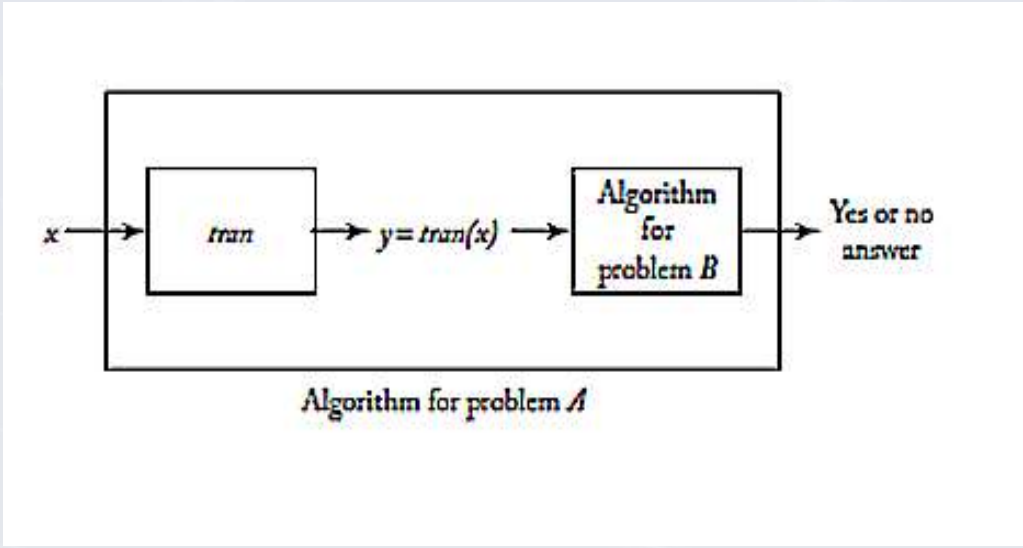
Polynomial-time Reducibility

- Want to solve decision problem A
- Have an algorithm to solve decision problem B
- Can write an algorithm that creates instance y of problem B from every instance x of problem A such that:
  - Algorithm for B answers yes for y if the answer to problem A is yes for x

Polynomial-time Reducibility

- Transformation algorithm
  - Function that maps every instance of problem A to an instance of problem B
  - y = trans(x)
- Transformation algorithm + algorithm for problem B yields an algorithm for problem A

Figure 9.4



Polynomial-time many-one reducible

- If there exists a polynomial-time transformation algorithm from decision problem A to decision problem B, problem A is polynomial-time many-one reducible to problem B
- $A \propto B$
- Many-one: transformation algorithm is a function that may map many instances of problem A to one instance of problem B
- If the transformation algorithm is polynomial-time and the algorithm for problem B is polynomial, The algorithm for A must be polynomial

Theorem 9.1

- If decision problem B is in P and A $\propto$ B, then decision problem A is in P

Proof

- Let p be the polynomial bound on the computation of the transformation algorithm T
- Let q be the polynomial bound on the algorithm M for B
- The size of T(x) is at most p(n) for input x of size n
- Algorithm M with input T(x) does at most q(p(n)) steps
- The total work to transform x to T(x) and then apply M to get the correct answer:
  - p(n) + q(p(n)), which is polynomial in n

NP-Complete

- A problem B is called NP-complete if both the following are true:

1. B is in NP
2. For every other problem A in NP, $A \propto B$

Theorem 9.2 – Cook's Theorem

- CNF-Satisfiability is NP-complete.
- Proof found in Cook (1971) and in Gary and Johnson (1979)

Theorem 9.3

- A problem C is NP-complete if both of the following are true:

1. C is in NP
2. For some other NP-complete problem B, $B \propto C$

Proof is based on the transitivity of reducibility

Figure 9.7