CS472 Module 10 Part C - Intractability and the Theory of NP

Athens State University

April 4, 2016

Outline

Contents

1 Computational complexity: Introduction

Computational Complexity: Computational resources

- We have the need to understand the amount of resources needed to solve computational problems that we find interesting
  - Resources: time, memory, communication, randomness,...
- Analysis of Algorithms requires that we seek upper bounds on such amounts
- Suppose we look at the opposite question: what about the lower bounds on resource use
  - Looking for *negative results* showing problems require lots of resources
  - In particular, looking for *intractable problems*: computational problems that require impossibly large resources to be solved
- This leads us into the theory of *computational complexity*

Computational Complexity: What is a "Computational Problem"?

- *Computational problem*: Given some input, which we assumed encoded over the alphabet $\{0,1\}$, we want to return in output a solution satisfying some property inherent to the problem
- *Decision problem*: Given some input $x \in \{0,1\}^*$, we are asked to verify whether that input satisfies a certain property by outputting a "YES/NO" answer.
  - We can, without any loss of generality, rephrase "YES/NO" to 0/1.

- One is said to *specify* a decision problem by defining the set $L \subseteq \{0,1\}^*$ for which the answer to the problem is "YES".
- A subset of $\{0,1\}^*$ is also called a *language*, and thus we can specify every decision problem using a language.

Computational Complexity: What is a "Computational Problem"?

- *Decision Problems*
  - Example: The 3-coloring problem:
    - Given a graph $G =< V, E >$, is there a way to assign a "color" chosen from $\{00, 01, 10\}$ to each vertex s.t. no 2 adjacent vertices have the same color?
  - If we let $3COL \subseteq \{0,1\}^*$ be the (binary) descriptions of 3-colorable graphs, then $3COL$ is the language that specifies the 3-coloring problem.

Computational Complexity: What is a "Computational Problem"?

- Search problems: Given some input $x \in \{0,1\}^*$, find some $y \in \{0,1\}^*$ that is in some relation to x, if such a y exists.
  - So, we have a relation $R \subseteq \{0,1\}^* X \{0,1\}^*$ where $(x, y) \in R$ i.f.f. y is an admissible answer given x
- Consider the 3-coloring problem as a search problem: Given $G =< V, E >$, we want to find, if it exists, a coloring $c : V \to \{00, 01, 10\}$ of the vertices s.t. for every $(u, v) \in V$ we have $c(u) \neq c(v)$
- Note the extension to decision problems, as we now not only asking does a solution exist but, if it does, explain how to construct it.
- Formally, we say that the 3-coloring problem is specified by the relation R_{3COL} that contains all pairs (G, c) where G is a 3-colorable graph and c is a valid 3-coloring of G

2 P and NP

Complexity Class

- **Time Complexity Class**: Let $t : N \to R^+$ be a function. The complexity class $TIME(t(n))$ is collection of all languages that are decidable by a Turing machine in $O(t(n))$ time.
- Note that we have two problems to consider:
  - *Solution* Does the Turing machine halt and produce a YES/NO answer to problem? In other words, does a program generate a potential solution to the problem.
  - `Verification` Can we verify that the the Turing machine generated the correct answer?

Complexity Class

- Consider the language $3COL$. If we say that $3COL \in TIME(t(n))$, this implies that $3COL$ can be decided in time $O(t(n))$ and 3-coloring problem can be solved in $O(t(n))$.
- Note that we can, without loss of information, consider all running times in a class of functions to be equivalent.
  - For example, we can reason about algorithm behavior in terms of polynomial vs. exponential
  - In other words, we see little difference when thinking asymptotically about n^2 vs. n^3

P and NP: Asymptotic Complexity of Problems

- So, we use "Big-Oh" as way to specify the *asymptotic* complexity of problems
- Example: we seek to find the best asymptotic bounds (upper and lower) on the running time of an algorithm that solves 3-coloring on all instances.
- In truth, we are not so ambitious, we just ask is there an algorithm to solve a problem that is "tractable" in its asymptotic running time

P and NP: What do we mean by "tractable"?

- By convention, we will classify an algorithm as being *tractable* if it runs in polynomial time
- Formally, we ask if the run time of the algorithm is asymptotic to some polynomial p
  - In other words, the algorithm run in at most $p(n)$ time on inputs of length n
- We define P to be class of decision problems solvable in polynomial time
  - *Solvable*: Construct a solution and verify its correctness

P and NP: So what do we mean by "intractable"?

- A search problem defined by a relation R is an NP search problem if the relation is efficiently computable and such that solutions, if they exist, are short
- Formally, R is an NP search problem if there is a polynomial time algorithm that, given x and y, decides whether $(x, y) \in R$, and if there is a polynomial p such that if $(x, y) \in R$ then $|y| \leq p(|x|)$.

P and NP: So what do we mean by "intractable"?

- A decision problem L is an NP decision problem if there is some NP relation s.t. $x \in L$ iff there is y s.t. $(x, y) \in R$
- An equivalent formulation:
  - A decision problem L is an NP decision problem if there is a polynomial time algorithm V and a polynomial p such that $x \in L$ iff there is a y, $|y| \leq p(|x|)$, such that $V(x, y)$ accepts.
- We note by *NP* the class of NP decision problems.

3 NP-completeness

NP-completeness: Reductions

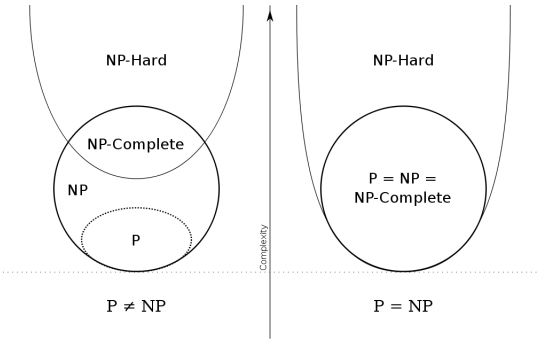
- Let A and B be two decision problems
- The problem A *reduces* to problem B, denoted as $A \leq B$ if there is a polynomial computable function f s.t. $x \in A$ iff $f(x) \in B$.
- Observations:
  - If $A \leq B$ and $B \in P$, then $A \in P$.
  - If $A \leq B$ and $B \leq C$, then $A \leq B$.

NP-completeness: Definition

- A decision problem A is *NP*-hard if $\forall L \in P$, $L \leq A$
- A decision problem A is *NP*-complete if A is *NP*-hard and A is in *NP*

4 The P versus NP condrmdum

P vs. NP: Context



The question "Is $P = NP$" is one of the most fundamental unanswered questions in computer science.

P vs. NP: Implications

- Consider the *subset sum problem*: given a set of integers, does some nonempty subset of those integers sum to 0?
  - Example: Does a subset of $\{-2, -3, 15, 7, -10\}$ sum to 0?
  - One can quickly respond "YES" because $\{-2, -3, -10, 15\}$ is such a subset
    - And we can quickly verify this fact
  - But no known algorithm exists to find such a subset in polynomial time
    - There is an algorithm that is $O(2^n - 1)$
- If $P = NP$, then we would have a clear indication that a problem that can be verified in polynomial time can be solved in polynomial time
- If $P \neq NP$, then there are problems in *NP* that are harder to compute than to verify

P vs. NP: Why do we believe that $P \neq NP$?

- Over 3000 important algorithms have been identified as being *NP*-complete problems
  - No one has been able to find a polynomial-time algorithm for any of these problems in over 50 years worth of research
- One can intuitively argue that the existence of problems that are hard to solve but for which it is easy to verify the solutions matches real-world experience

P vs. NP: What if we're wrong and $P = NP$?

- Due to the nature of the definition of the P and NP classes, finding a polynomial time algorithm for one NP-complete problem means that ALL NP-complete problems have such an algorithm
- For example, much of modern cryptography is designed around the fact that NP-complete problems are difficult to solve in reasonable time
- A lot of the logic underlying modern mathematics has connections to this part of complexity theory.

5 Key Points

Key Points

- Difference between algorithm analysis and complexity theory
- Definition of decision problem and search problem
- Concept of complexity class
  - Definition of P, NP, NP-complete, and NP-hard
  - Importance of the $P \neq NP$ conjecture