

MA/CSSE 473

Day 38

Problems

Decision Problems

P and NP



Polynomial-time algorithms

**INTRO TO COMPUTATIONAL
COMPLEXITY**



The Law of the Algorithm Jungle

- Polynomial good, exponential bad!
- The latter is obvious, the former may need some explanation
- We say that polynomial-time problems are **tractable**, exponential problems are **intractable**

tractable

1. (*obsolete*) Capable of being handled or touched; palpable; practicable; feasible; as, tractable measures.

*"I have always found horses, an animal I am attached to, very **tractable** when treated with humanity and steadiness." - Mary Wollstonecraft, "A Vindication of the Rights of Woman"*

1. Capable of being easily led, taught, or managed; docile; manageable; governable; as, tractable children; a tractable learner.

Polynomial time vs exponential time

- What's so good about polynomial time?
 - It's not exponential!
 - We can't say that every polynomial time algorithm has an acceptable running time,
 - but it is certain that if it **doesn't** run in polynomial time, it only works for small inputs.
 - Polynomial time is closed under standard operations.
 - If $f(t)$ and $g(t)$ are polynomials, so is $f(g(t))$.
 - also closed under sum, difference, product
- Almost all of the algorithms we have studied run in polynomial time.
 - Except those (like permutation and subset generation) whose output is exponential.



Decision problems

- When we define the class P, of “polynomial-time problems”, we will restrict ourselves to *decision problems*.
- *Almost any problem can be rephrased as a decision problem.*
- Basically, a decision problem is a question that has two possible answers, yes and no.
- The question is about some input.
- A *problem instance* is a combination of the problem and a specific input.



Decision problem definition

- The statement of a decision problem has two parts:
 - The ***instance description*** part defines the information expected in the input
 - The ***question part*** states the specific yes-or-no question; the question refers to variables that are defined in the instance description



Decision problem examples

- **Definition:** In a graph $G=(V,E)$, a **clique** E is a subset of V such that for all u and v in E , the edge (u,v) is in E .
- **Clique Decision problem**
 - Instance: an undirected graph $G=(V,E)$ and an integer k .
 - Question: Does G contain a clique of k vertices?
- **k-Clique Decision problem**
 - Instance: an undirected graph $G=(V,E)$. **Note that k is some constant, independent of the problem.**
 - Question: Does G contain a clique of k vertices?



Decision problem example

- **Definition:** The **chromatic number** of a graph $G=(V,E)$ is the smallest number of colors needed to color G , so that no two adjacent vertices have the same color
- **Graph Coloring Optimization Problem**
 - Instance: an undirected graph $G=(V,E)$.
 - Problem: Find G 's chromatic number and a coloring that realizes it
- **Graph Coloring Decision Problem**
 - Instance: an undirected graph $G=(V,E)$ and an integer $k>0$.
 - Question: Is there a coloring of G that uses no more than k colors?
- Almost every optimization problem can be expressed in decision problem form



Decision problem example

- **Definition:** Suppose we have an unlimited number of bins, each with capacity 1.0, and n objects with sizes s_1, \dots, s_n , where $0 < s_i \leq 1$ (all s_i rational)
- **Bin Packing Optimization Problem**
 - Instance: s_1, \dots, s_n as described above.
 - Problem: Find the smallest number of bins into which the n objects can be packed
- **Bin Packing Decision Problem**
 - Instance: s_1, \dots, s_n as described above, and an integer k .
 - Question: Can the n objects be packed into k bins?



Reduction

- Suppose we want to solve problem \mathbf{p} , and there is another problem \mathbf{q} .
- Suppose that we also have a function T that
 - takes an input x for \mathbf{p} , and
 - produces $T(x)$, an input for \mathbf{q} such that the correct answer for \mathbf{p} with input x is *yes* if and only if the correct answer for \mathbf{q} with input $T(x)$ is *yes*.
- We then say that \mathbf{p} is *reducible* to \mathbf{q} and we write $\mathbf{p} \leq \mathbf{q}$.
- If there is an algorithm for \mathbf{q} , then we can compose T with that algorithm to get an algorithm for \mathbf{p} .
- If T is a function with polynomially bounded running time, we say that \mathbf{p} is *polynomially reducible* to \mathbf{q} and we write $\mathbf{p} \leq_p \mathbf{q}$.
- From now on, *reducible* means *polynomially reducible*.



Classic 473 reduction

- Moldy Chocolate is reducible to 4-pile Nim
- $T(\text{rows_above}, \text{rows_below}, \text{cols_left}, \text{cols_right})$
→
 $\text{is_Nim_loss}(\text{rows_above}, \text{rows_below}, \text{cols_left}, \text{cols_right})$



Definition of the class P

- **Definition:** An algorithm is *polynomially bounded* if its worst-case complexity is big-O of a polynomial function of the input size n .
 - i.e. if there is a single polynomial p such that for each input of size n , the algorithm terminates after at most $p(n)$ steps.
 - The input size is the number of bits on the representation of the problem instance's input.
- **Definition:** A problem is polynomially bounded if there is a polynomially bounded algorithm that solves it
- **The class P**
 - P is the class of decision problems that are polynomially bounded
 - Informally (with slight abuse of notation), we also say that polynomially bounded optimization problems are in P



Example of a problem in P

- MST
 - Input: A weighted graph $G=(V,E)$ with n vertices [each edge e is labeled with a non-negative weight $w(e)$], and a number k .
 - Question: Is the total weight of a minimal spanning tree for G less than k ?
- How do we know it's in P ?



Example: Clique problems

- It is known that we can determine whether a graph with n vertices has a k -clique in time $O(k^2n^k)$.
- **Clique Decision problem 1**
 - Instance: an undirected graph $G=(V,E)$ and an integer k .
 - Question: Does G contain a clique of k vertices?
- **Clique Decision problem 2**
 - Instance: an undirected graph $G=(V,E)$. **Note that k is some constant, independent of the problem.**
 - Question: Does G contain a clique of k vertices?
- Are either of these decision problems in P ?



The problem class NP

- NP stands for Nondeterministic Polynomial time.
- The first stage assumes a “guess” of a possible solution.
- Can we **verify** whether the proposed solution really is a solution in polynomial time?



More details

- Example: Graph coloring. Given a graph G with N vertices, can it be colored with k colors?
- A solution is an actual k -coloring.
- A “proposed solution” is simply something that is in the right form for a solution.
 - For example, a coloring that may or may not have only k colors, and may or may not have distinct colors for adjacent nodes.
- The problem is in NP iff there is a polynomial-time (in N) algorithm that can check a proposed solution to see if it really is a solution.



Still more details

- A nondeterministic algorithm has two phases and an output step.
- The nondeterministic “guessing” phase, in which the proposed solution is produced. This proposed solution will be a solution if there is one.
- The deterministic verifying phase, in which the proposed solution is checked to see if it is indeed a solution.
- Output “yes” or “no”.



pseudocode

```
void checker(String input)  
// input is an encoding of the problem instance.  
  String s = guess(); // s is some “proposed solution”  
  boolean checkOK = verify(input, s);  
  if (checkOK)  
    print “yes”
```

- If the *checker* function would print “yes” for any string *s*, then the non-deterministic algorithm answers “yes”. Otherwise, the non-deterministic algorithm answers “no”.



The problem class NP

- NP is the class of decision problems for which there is a polynomially bounded nondeterministic algorithm.



Some NP problems

- Graph coloring
- Bin packing
- Clique



Problem Class Containment

- Define Exp to be the set of all decision problems that can be solved by a deterministic exponential-time algorithm.
- Then $P \subseteq NP \subseteq Exp$.
 - $P \subseteq NP$. A deterministic polynomial-time algorithm is (with a slight modification to fit the form) a polynomial-time nondeterministic algorithm (skip the guessing part).
 - $NP \subseteq Exp$. It's more complicated, but we basically turn a non-deterministic polynomial-time algorithm into a deterministic exponential-time algorithm, replacing the *guess* step by a systematic trial of all possibilities.



The \$10⁶ Question

- The big question is , does $P=NP$?
- *The $P=NP$? question is one of the most famous unsolved math/CS problems!*
- *In fact, there is a million dollar prize for the person who solves it. <http://www.claymath.org/millennium/>*
- *What do computer scientists THINK the answer is?*



August 6, 2010

- My 33rd wedding anniversary
- 65th anniversary of the atomic bombing of Hiroshima
- The day Vinay Dolaliker announced a proof that $P \neq NP$
- By the next day, the web was a'twitter!
- Gaps in the proof were found.
- If it had been proven, Dolaliker would have been \$1,000,000 richer!
 - <http://www.claymath.org/millennium/>
 - http://www.claymath.org/millennium/P_vs_NP/
- Other Millennium Prize problems:
 - Poincare Conjecture (solved)
 - Birch and Swinnerton-Dyer Conjecture
 - Navier-Stokes Equations
 - Hodge Conjecture
 - Riemann Hypothesis
 - Yang-Mills Theory



More P vs NP links

- The Minesweeper connection:
 - http://www.claymath.org/Popular_Lectures/Minesweeper/
- November 2010 CACM editor's article:
 - <http://cacm.acm.org/magazines/2010/11/100641-on-p-np-and-computational-complexity/fulltext>
 - <http://www.rose-hulman.edu/class/csse/csse473/201110/Resources/CACM-PvsNP.pdf>
- From the same magazine: Using Complexity to Protect Elections:
 - <http://www.rose-hulman.edu/class/csse/csse473/201110/Resources/ProtectingElections.pdf>



Other NP problems

- **Job scheduling with penalties**
- Suppose n jobs J_1, \dots, J_n are to be executed one at a time.
 - Job J_i has execution time t_i , completion deadline d_i , and penalty p_i if it does not complete on time.
 - A *schedule* for the jobs is a permutation π of $\{1, \dots, n\}$, where $J_{\pi(i)}$ is the i^{th} job to be run.
 - The total penalty for this schedule is P_π , the sum of the p_i based on this schedule.
- Scheduling decision problem:
 - Instance: the above parameters, and a non-negative integer k .
 - Question: Is there a schedule π with $P_\pi \leq k$?



Other NP problems

- **Knapsack**
- Suppose we have a knapsack with capacity C , and n objects with sizes s_1, \dots, s_n and profits p_1, \dots, p_n .
- Knapsack decision problem:
 - Instance: the above parameters, and a non-negative integer k .
 - Question: Is there a subset of the set of objects that fits in the knapsack and has a total profit that is at least k ?



Other NP problems

- **Subset Sum Problem**

- Instance: A positive integer C and n positive integers s_1, \dots, s_n .
- Question: Is there a subset of these integers whose sum is exactly C ?



Other NP problems

- CNF Satisfiability problem (introduction)
- A *propositional formula* consists of boolean-valued variables and operators such as \wedge (and), \vee (or), negation (I represent a negated variable by showing it in boldface), and \rightarrow (implication).
- It can be shown that every propositional formula is equivalent to one that is in *conjunctive normal form*.
 - A *literal* is either a variable or its negation.
 - A *clause* is a sequence of one or more literals, separated by \vee .
 - A CNF formula is a sequence of one or more clauses, separated by \wedge .
 - Example $(p \vee q \vee r) \wedge (p \vee \mathbf{s} \vee q \vee t) \wedge (s \vee \mathbf{w})$
- For any finite set of propositional variables, a **truth assignment** is a function that maps each variable to $\{true, false\}$.
- A truth assignment **satisfies a formula** if it makes the value of the entire formula true.
 - Note that a truth assignment satisfies a CNF formula if and only if it makes each clause true.



Other NP problems

- Satisfiability problem:
- Instance: A CNF propositional formula f (containing n different variables).
- Question: Is there a truth assignment that satisfies f ?



A special case

- 3-Satisfiability problem:
- A CNF formula is in 3-CNF if every clause has exactly three literals.
- Instance: A 3CNF propositional formula f (containing n different variables).
- Question: Is there a truth assignment that satisfies f ?



NP-hard and NP-complete problems

- A problem is NP-hard if every problem in *NP* is reducible to it.
- A problem is *NP*-complete if it is in *NP* and is NP-hard.
- Showing that a problem is NP complete is difficult.
 - Has only been done directly for a few problems.
 - Example: 3-satisfiability
- If **p** is NP-hard, and $p \leq_p q$, then **q** is NP-hard.
- So most *NP*-complete problems are shown to be so by showing that 3-satisfiability (or some other known NP-complete problem) reduces to them.



Examples of NP-complete problems

- satisfiability (3-satisfiability)
- clique (and its dual, independent set).
- graph 3-colorability
- Minesweeper: is a certain square safe on an $n \times n$ board?
 - <http://for.mat.bham.ac.uk/R.W.Kaye/minesw/ordmsw.htm>
- hamiltonian cycle
- travelling salesman
- register allocation
- scheduling
- bin packing
- knapsack

