CSCE 310J
Data Structures & Algorithms

P, NP, and NP-Complete

Dr. Steve Goddard
goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/CSCE310J

1

CSCE 310J
Data Structures & Algorithms

- Giving credit where credit is due:
  - » Most of the lecture notes are based on slides created by Dr. Cusack and Dr. Leubke.
  - » I have modified them and added new slides

2

Tractability

- Some problems are *intractable*: as they grow large, we are unable to solve them in reasonable time
- What constitutes reasonable time? Standard working definition: *polynomial time*
  - » On an input of size n the worst-case running time is $O(n^k)$ for some constant k
  - » Polynomial time: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$
  - » Not in polynomial time: $O(2^n)$, $O(n^n)$, $O(n!)$

3

Polynomial-Time Algorithms

- Are some problems solvable in polynomial time?
  - » Of course: every algorithm we've studied provides polynomial-time solution to some problem
- Are all problems solvable in polynomial time?
  - » No: Turing's "Halting Problem" is not solvable by any computer, no matter how much time is given
- Most problems that do not yield polynomial-time algorithms are either optimization or decision problems.

4

Optimization/Decision Problems

- Optimization Problems:
  - » An optimization problem is one which asks, "What is the optimal solution to problem X?"
  - » Examples:
    - 0-1 Knapsack
    - Fractional Knapsack
    - Minimum Spanning Tree
- Decision Problems
  - » An decision problem is one which asks, "Is there a solution to problem X with property Y?"
  - » Examples:
    - Does a graph G have a MST of weight ≤ W?

5

Optimization/Decision Problems

- An optimization problem tries to find an optimal solution
- A decision problem tries to answer a yes/no question
- Many problems will have decision and optimization versions.
  - » Eg: Traveling salesman problem
    - optimization: find hamiltonian cycle of minimum weight
    - decision: find hamiltonian cycle of weight < k
- Some problems are decidable, but *intractable*: as they grow large, we are unable to solve them in reasonable time
  - » *What constitutes "reasonable time"?*
  - » *Is there a polynomial-time algorithm that solves the problem?*

6

The Class *P*

P: the class of decision problems that have polynomial-time deterministic algorithms.

- » That is, they are solvable in $O(p(n))$, where $p(n)$ is a polynomial on n
- » A deterministic algorithm is (essentially) one that always computes the correct answer

Why polynomial?

- » if not, very inefficient
- » nice closure properties
- » machine independent in a strong sense

7

Sample Problems in P

- ◆ Fractional Knapsack
- ◆ MST
- ◆ Single-source shortest path
- ◆ Sorting
- ◆ Others?

8

The class *NP*

NP: the class of decision problems that are solvable in polynomial time on a *nondeterministic* machine (or with a non-deterministic algorithm)

- ◆ (A *determinstic* computer is what we know)
- ◆ A *nondeterministic* computer is one that can "guess" the right answer or solution
  - » Think of a nondeterministic computer as a parallel machine that can freely spawn an infinite number of processes
- ◆ Thus *NP* can also be thought of as the class of problems
  - » whose solutions can be verified in polynomial time; or
  - » that can be solved in polynomial time on a machine that can pursue infinitely many paths of the computation in parallel
- ◆ Note that *NP* stands for "Nondeterministic Polynomial-time"

9

Nondeterminism

- ◆ Think of a non-deterministic computer as a computer that magically "guesses" a solution, then has to verify that it is correct
  - » If a solution exists, computer always guesses it
  - » One way to imagine it: a parallel computer that can freely spawn an infinite number of processes
    - ❖ Have one processor work on each possible solution
    - ❖ All processors attempt to verify that their solution works
    - ❖ If a processor finds it has a working solution
  - » So: **NP** = problems *verifiable* in polynomial time

10

Sample Problems in NP

- ◆ Fractional Knapsack
- ◆ MST
- ◆ Single-source shortest path
- ◆ Sorting
- ◆ Others?
  - » Hamiltonian Cycle (Traveling Sales Person)
  - » Satisfiability (SAT)
  - » Conjunctive Normal Form (CNF) SAT
  - » 3-CNF SAT

11

Hamiltonian Cycle

- ◆ A *hamiltonian cycle* of an undirected graph is a simple cycle that contains every vertex
- ◆ The hamiltonian-cycle problem: given a graph G, does it have a hamiltonian cycle?
- ◆ *Describe a naïve algorithm for solving the hamiltonian-cycle problem. Running time?*
- ◆ The hamiltonian-cycle problem is in **NP**:
  - » No known deterministic polynomial time algorithm
  - » Easy to verify solution in polynomial time (*How?*)

12

The *Satisfiability* (SAT) Problem

- *Satisfiability* (SAT):
  - Given a Boolean expression on *n* variables, can we assign values such that the expression is TRUE?
  - Ex: $((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$
  - Seems simple enough, but no known deterministic polynomial time algorithm exists
  - Easy to verify in polynomial time!

13

Conjunctive Normal Form (CNF) and 3-CNF

- Even if the form of the Boolean expression is simplified, no known polynomial time algorithm exists
  - *Literal*: an occurrence of a Boolean or its negation
  - A Boolean formula is in *conjunctive normal form*, or *CNF*, if it is an AND of clauses, each of which is an OR of literals
    - Ex: $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_5)$
  - *3-CNF*: each clause has exactly 3 distinct literals
    - Ex: $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_5 \vee x_3 \vee x_4)$
    - Notice: true if at least one literal in each clause is true

14

Example: CNF satisfiability

- This problem is in *NP*. Nondeterministic algorithm:
  - Guess truth assignment
  - Check assignment to see if it satisfies CNF formula
- Example:
  $(A \vee \neg B \vee \neg C) \wedge (\neg A \vee B) \wedge (\neg B \vee D \vee F) \wedge (F \vee \neg D)$
- Truth assignments:

	A	B	C	D	E	F
1.	0	1	1	0	1	0
2.	1	0	0	0	0	1
3.	1	1	0	0	0	1
4.	... (how many more?)					

Checking phase: $\Theta(n)$

15

Review: P And NP Summary

- **P** = set of problems that can be solved in polynomial time
  - Examples: Fractional Knapsack, …
- **NP** = set of problems for which a solution can be verified in polynomial time
  - Examples: Fractional Knapsack,…, Hamiltonian Cycle, CNF SAT, 3-CNF SAT
- Clearly $\mathbf{P} \subseteq \mathbf{NP}$
- Open question: Does **P = NP**?
  - Most suspect not

16

NP-complete problems

- A decision problem *D* is <u>*NP*-complete</u> iff
  1. $D \in NP$
  2. every problem in *NP* is polynomial-time reducible to *D*
- Cook's theorem (1971): CNF-sat is *NP*-complete
- Other *NP*-complete problems obtained through polynomial-time reductions of known *NP*-complete problems

17

Review: Reduction

- A problem P can be *reduced* to another problem Q if any instance of P can be rephrased to an instance of Q, the solution to which provides a solution to the instance of P
  - This rephrasing is called a *transformation*
- Intuitively: If P reduces in polynomial time to Q, P is "no harder to solve" than Q

18

NP-Hard and NP-Complete

- If P is *polynomial-time reducible* to Q, we denote this $P \leq_p Q$
- Definition of NP-Hard and NP-Complete:
  - » If all problems $R \in$ **NP** are reducible to P, then P is *NP-Hard*
  - » We say P is *NP-Complete* if P is NP-Hard and $P \in$ **NP**
- If $P \leq_p Q$ and P is NP-Complete, Q is also NP-Complete

19

Proving NP-Completeness

- *What steps do we have to take to prove a problem Q is NP-Complete?*
  - » Pick a known NP-Complete problem P
  - » Reduce P to Q
    - Describe a transformation that maps instances of P to instances of Q, s.t. "yes" for Q = "yes" for P
    - Prove the transformation works
    - Prove it runs in polynomial time
  - » Oh yeah, prove $Q \in$ **NP** (*What if you can't?*)

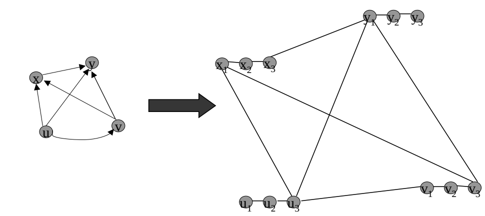
20

Directed Hamiltonian Cycle ⇒ Undirected Hamiltonian Cycle

- *What was the hamiltonian cycle problem again?*
- For my next trick, I will reduce the *directed hamiltonian cycle* problem to the *undirected hamiltonian cycle* problem before your eyes
  - » *Which variant am I proving NP-Complete?*
- Given a directed graph G,
  - » *What transformation do I need to effect?*

21

Directed Hamiltonian Cycle ⇒ Undirected Hamiltonian Cycle



22

Transformation: Directed ⇒ Undirected Ham. Cycle

- Transform graph G = (V, E) into G' = (V', E'):
  - » Every vertex *v* in V transforms into 3 vertices v^1, v^2, v^3 in V' with edges (v^1,v^2) and (v^2,v^3) in E'
  - » Every directed edge (*v*, *w*) in E transforms into the undirected edge (v^3, w^1) in E' (draw it)
  - » *Can this be implemented in polynomial time?*
  - » *Argue that a directed hamiltonian cycle in G implies an undirected hamiltonian cycle in G'*
  - » *Argue that an undirected hamiltonian cycle in G' implies a directed hamiltonian cycle in G*

23

Undirected Hamiltonian Cycle

- Thus we can reduce the directed problem to the undirected problem
- *What's left to prove the undirected hamiltonian cycle problem NP-Complete?*
- *Argue that the problem is in* **NP**

24

Hamiltonian Cycle ⇒ TSP

- The well-known *traveling salesman problem*:
  - » Optimization variant: a salesman must travel to n cities, visiting each city exactly once and finishing where he begins. How to minimize travel time?
  - » Model as complete graph with cost $c(i,j)$ to go from city i to city j
- *How would we turn this into a decision problem?*
  - » A: ask if $\exists$ a TSP with cost $< k$

25

Hamiltonian Cycle ⇒ TSP

- The steps to prove TSP is NP-Complete:
  - » Prove that TSP ∈ **NP** (*Argue this*)
  - » Reduce the undirected hamiltonian cycle problem to the TSP
    - So if we had a TSP-solver, we could use it to solve the hamiltonian cycle problem in polynomial time
    - *How can we transform an instance of the hamiltonian cycle problem to an instance of the TSP?*
    - *Can we do this in polynomial time?*

26

The TSP

- Random asides:
  - » TSPs (and variants) have enormous practical importance
    - E.g., for shipping and freighting companies
    - Lots of research into good approximation algorithms
  - » Recently made famous as a DNA computing problem

27

Review: P and NP

- *What do we mean when we say a problem is in **P**?*
  - » A: A solution can be found in polynomial time
- *What do we mean when we say a problem is in **NP**?*
  - » A: A solution can be verified in polynomial time
- *What is the relation between **P** and **NP**?*
  - » A: **P** ⊆ **NP**, but no one knows whether **P = NP**

28

Review: NP-Complete

- *What, intuitively, does it mean if we can reduce problem P to problem Q?*
  - » P is "no harder than" Q
- *How do we reduce P to Q?*
  - » Transform instances of P to instances of Q in polynomial time s.t. Q: "yes" iff P: "yes"
- *What does it mean if Q is NP-Hard?*
  - » Every problem P∈ **NP** $\leq_p$ Q
- *What does it mean if Q is NP-Complete?*
  - » Q is NP-Hard and Q ∈ **NP**

29

Review: Proving Problems NP-Complete

- *How do we usually prove that a problem R is NP-Complete?*
  - » A: Show R ∈ **NP**, and reduce a known NP-Complete problem Q to R

30

Other NP-Complete Problems

- *K-clique*
  - A clique is a subset of vertices fully connected to each other, i.e. a complete subgraph of G
  - The *clique problem*: how large is the maximum-size clique in a graph?
  - *No turn this into a decision problem?*
  - Is there a clique of size k?
- *Subset-sum*: Given a set of integers, does there exist a subset that adds up to some target *T*?
- *0-1 knapsack*: when weights not just integers
- *Hamiltonian path*: Obvious
- *Graph coloring*: can a given graph be colored with *k* colors such that no adjacent vertices are the same color?
- Etc…

31

General Comments

- Literally hundreds of problems have been shown to be NP-Complete
- Some reductions are profound, some are comparatively easy, many are easy once the key insight is given

32