中山大學
SUN YAT-SEN UNIVERSITY

Lecture 8
NP-completeness

Algorithm Design

zhangzizhen@gmail.com

QQ group: 117282780

Decision problems vs. optimization problems

- A decision problem is a question in some formal system that can be posed as a yes-no question.
- Optimization problems are concerned with finding the best answer to a particular input.
- There are standard techniques for transforming function and optimization problems into decision problems.
  - For example, in the traveling salesman problem, the optimization problem is to produce a tour with minimal weight. The associated decision problem is: for each *N*, to decide whether the graph has any tour with weight less than *N*. By repeatedly answering the decision problem, it is possible to find the minimal weight of a tour.

Undecidable problems

- **Halting problem** (Alan Turning 1936): given a computer program and an input to it, determine whether the program will halt on the input or continue working indefinitely on it.
- Assume that A is an algorithm that solves the halting problem, that is for any program P and input I,
  - A(P,I)=1, if program P halts on input I
  - A(P,I)=0, if program P does not halt on input I
- Construct a program Q for pair (P,P)
  - Q(P)=halt, if A(P,P)=0, i.e., if program P does not halt on input P
  - Q(P)=not halt, if A(P,P)=1, i.e., if program P halts on input P
- Substituting Q for P
  - Q(Q)=halt, if program Q does not halt on input Q
  - Q(Q)=not halt, if program Q halts on input Q

Polynomial Time

- We have mentioned efficient algorithms several times in this course. An efficient algorithm is generally taken to mean one whose running time depends polynomially on the size of the input.
- Problems that can be solved in polynomial time are called tractable, and problems that cannot be solved in polynomial time are called intractable.
- Examples:
  - Shortest Path Problem
  - Minimum Spanning Tree Problem
  - Fractional Knapsack Problem
  - Gaussian Elimination
  - …

Time-Bounded Turning Machine

- A Turing machine that, given an input of length n, always halts within T(n) moves is said to be *T(n)-time bounded*.
  - The TM can be multitape.
  - Sometimes, it can be nondeterministic.
- The deterministic, multitape case corresponds roughly to "an O(T(n)) running-time algorithm."

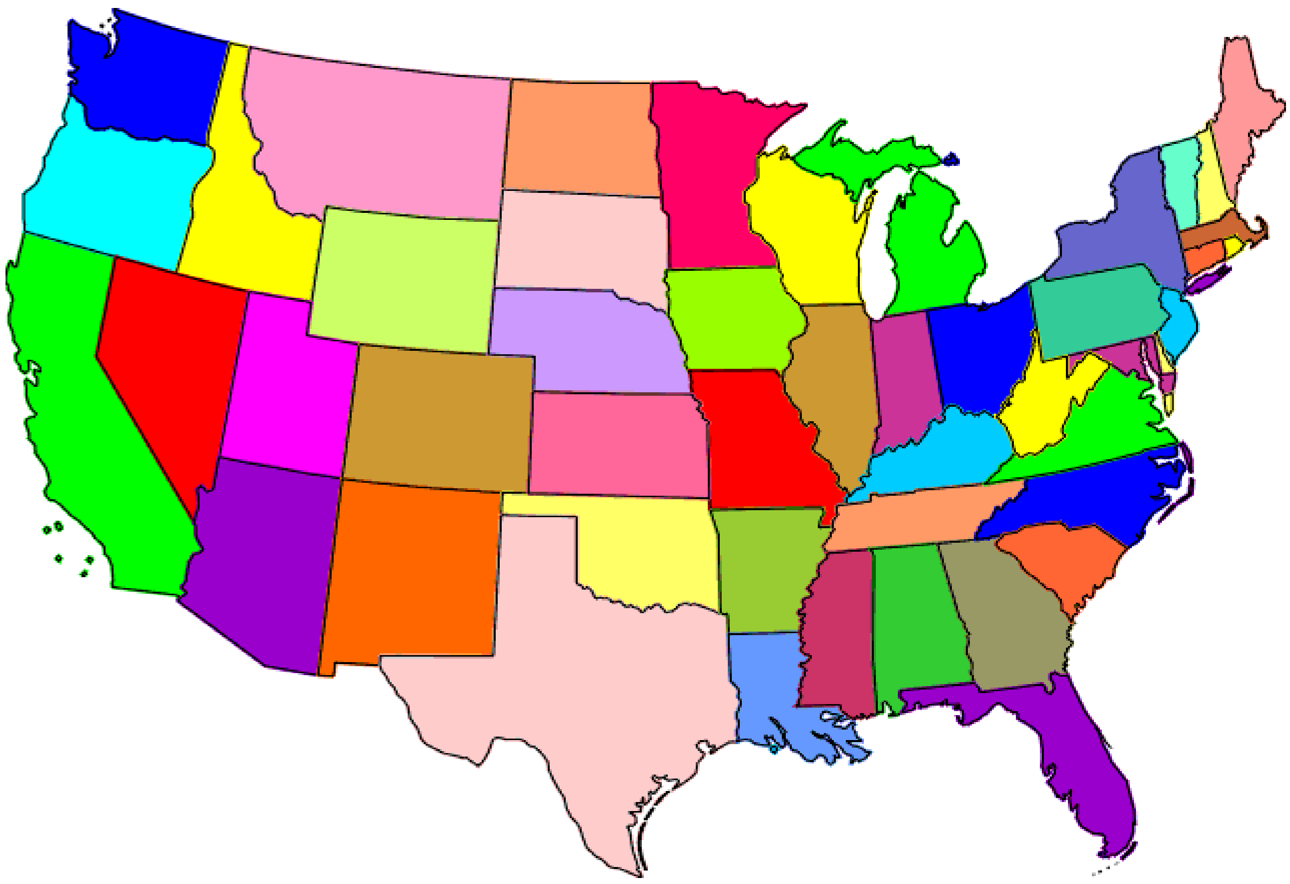
The Class P

- If a DTM M is T(n)-time bounded for some polynomial T(n), then we say M is *polynomial-time* (“*polytime*”) bounded.
- When we talk of **P**, it doesn’t matter whether we mean “by a computer” or “by a Turing machine”.

- You might worry that something like O(n log n) is not a polynomial.
- However, to be in **P**, a problem only needs an algorithm that runs in time less than some polynomial.
- Surely O(n log n) is less than the polynomial O(n^2).

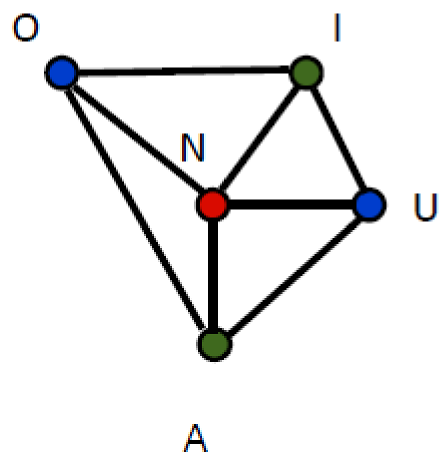
What is NP?

- Example: In a map, if we don't want neighboring states to be the same color. How many colors are needed?



Map Coloring

- Every map (planar graph) can be colored with four colors.
- Some can be colored with three.
- Some can be colored with two. Can you tell which?



- 3-coloring Problem:
  - Given a map, output “yes” if it can be colored with three colors, “no” otherwise.
  - It is an *NP-complete* problem. Currently we cannot find polynomial-time algorithms to solve it.

The Class NP

- The running time of a Nondeterministic TM is the maximum number of steps taken along any branch.
- If that time bound is polynomial, the NTM is said to be *polynomial-time bounded*.
- And its language/problem is said to be in the class **NP**. (Note: NP stands for nondeterministic polynomial time, not non-polynomial time)
- NP is the class of problems which have solutions that can be efficiently verified. (NP discuss with decision problems)
  - As usual, efficiently means polynomial in size of input.
- 3-coloring is in NP. Given a proposed coloring, we can quickly check if it works. Fractional Knapsack and 0-1 Knapsack are also in NP.

P vs. NP

- **P**: problems which we can efficiently solve.
- **NP**: problems which, given a proposed solution, we can efficiently check if it works.
- Every problem in P is also in NP.
- One of the most important open problems the question is **P = NP or P ≠ NP**?
- There are thousands of problems that are in **NP** but appear not to be in **P**.
- But no proof that they aren't really in **P**.

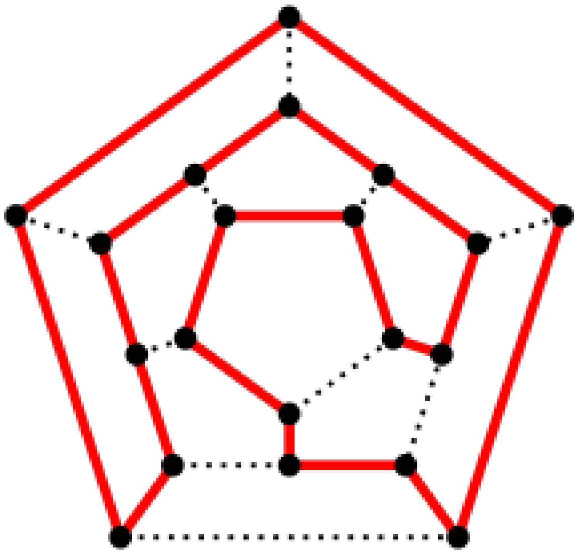
NP-complete

- One way to address the **P** = **NP** question is to identify *complete problems* for NP.
- An *NP-complete problem* has the property that if it is in **P**, then every problem in **NP** is also in **P**.
- Today over 3000 NP-complete problems known across all the sciences.
- Google Scholar search of NP-complete and biology returns over 10,000 articles.

- Defined formally via “polytime reductions.”

Hamiltonian Path and Cycle

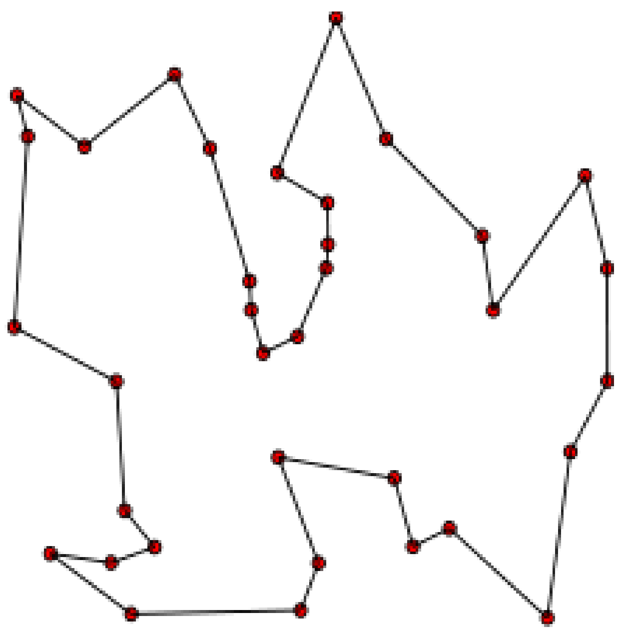
- a Hamiltonian path is a path in an undirected or directed graph that visits each vertex exactly once. A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian path that is a cycle. Determining whether such paths and cycles exist in graphs is the Hamiltonian path problem, which is NP-complete.



Travelling Salesman Problem

- The **Travelling Salesman Problem (TSP)** asks the following question:

  Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?



Reducibility

- 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.
  - Example: The problem “solving linear equations” can be reduced to the problem “solving quadratic equations”.
- 归约：一个问题A可以归约为问题B的含义是，可以用问题B的解法解决问题A，或者说，问题A可以“变成”问题B。
- “问题A可约化为问题B”有一个重要的直观意义：B的时间复杂度高于或者等于A的时间复杂度。也就是说，问题A不比问题B难。
- 归约具有传递性，如果问题A可归约为问题B，问题B可归约为问题C，则问题A可归约为问题C。

Polytime Reductions

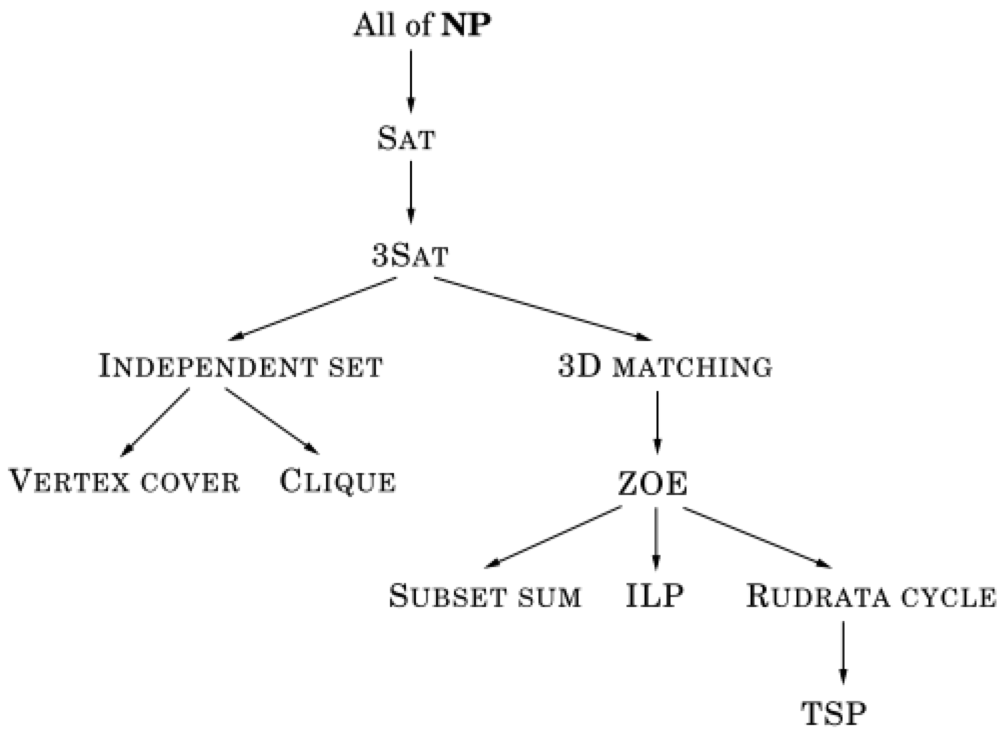
- A language L_1 is polynomial-time reducible to a language L_2, written $L_1 \leq_p L_2$.
- A problem/language M is said to be *NP-complete* if for every language L in **NP**, there is a polytime reduction from L to M.
- A language L is ***NP-complete***, if
  1. $L \in NP$, and
  2. $L' \leq_p L$ for every $L' \in NP$.
- Goal: find a way to show problem *L* to be NP-complete by reducing every language/problem in **NP** to *L* in such a way that if we had a deterministic polytime algorithm for *L*, then we could construct a deterministic polytime algorithm for any problem in **NP**.

Example: Hamiltonian Cycle to TSP

- Write a program the solves Hamiltonian Cycle
- You have a subroutine TSP(all pair distances D and an integer k) that reports the solution to TSP
  - Hamiltonian Cycle: given a graph, is there a cycle which visits all vertices **exactly once?**
  - TSP: Given a list of cities and pairwise distances between them, is there a tour which visits each city exactly once and has length at most k?
- Given G = (V, E), create a TSP instance where distance between u and v is
  - 1, if $(u,v) \in E$
  - 2, if $(u,v) \notin E$
- Report back TSP(D, |V|)
- Therefore, Hamiltonian Cycle $\leq_p$ TSP

NP-complete Problems

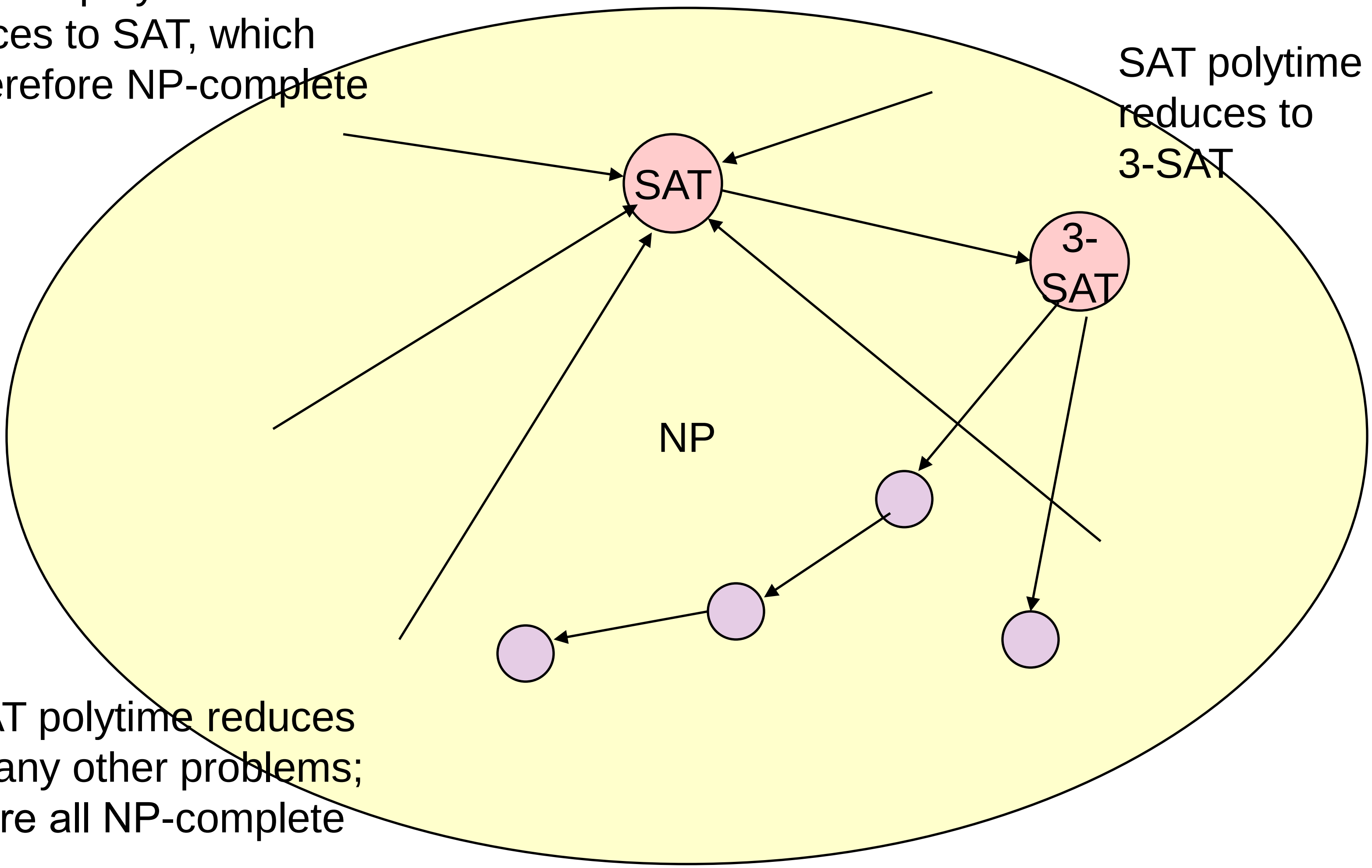
- Fundamental property: if M has a polytime algorithm, then L also has a polytime algorithm, i.e., if M is in **P**, then every L in **NP** is also in **P**, or “**P** = **NP**.”
- The first NP-complete problem: SAT.



NP-complete Problems

All of **NP** polytime reduces to SAT, which is therefore NP-complete

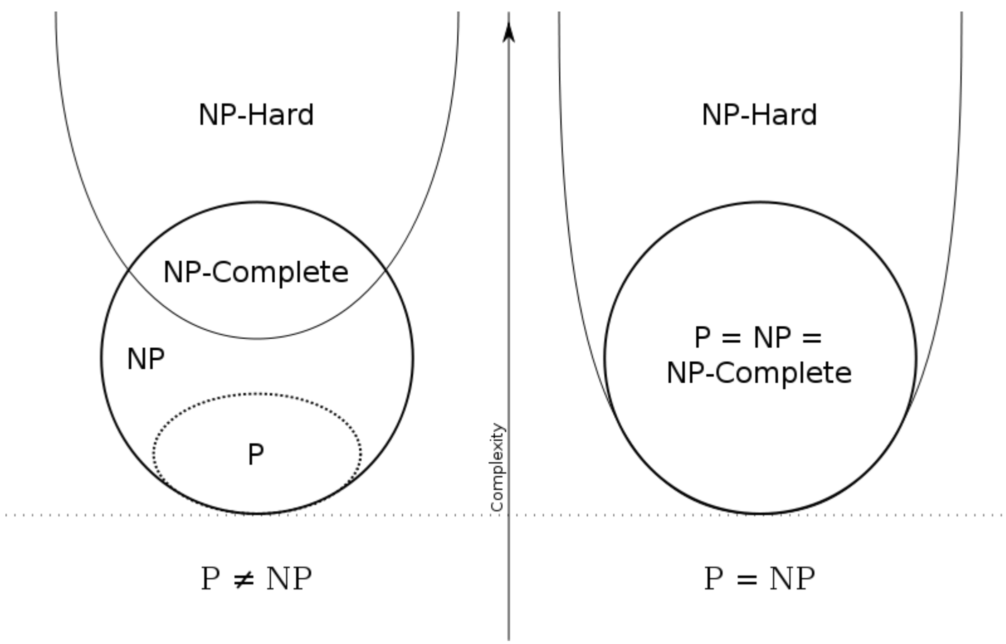
SAT polytime reduces to 3-SAT



3-SAT polytime reduces to many other problems; they're all NP-complete

P, NP, NP-complete and NP-hard

- NP-hard: Class of problems which are at least as hard as the hardest problems in NP. Problems in NP-hard do not have to be elements of NP, indeed, they may not even be decidable problems.
- Euler diagram for P, NP, NP-complete, and NP-hard set of problems:



Thank you!