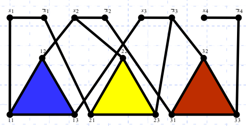
Presentation for use with the textbook, Algorithm Design and Applications, by M. T. Goodrich and R. Tamassia, Wiley, 2015

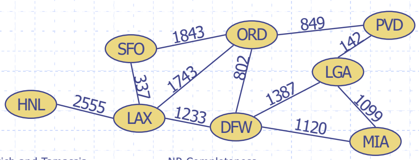
NP-Completeness



1

Running Time Revisited

- Input size, *n*
  - To be exact, let *n* denote the number of **bits** in a nonunary encoding of the input
- All the polynomial-time algorithms studied so far in this course run in polynomial time using this definition of input size.
  - Exception: any pseudo-polynomial time algorithm



2

Dealing with Hard Problems

- What to do when we find a problem that looks hard…

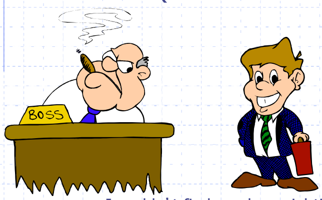
I couldn't find a polynomial-time algorithm;
I guess I'm too dumb.

(cartoon inspired by [Garey-Johnson, 79])

3

Dealing with Hard Problems

- Sometimes we can prove a strong lower bound… (but not usually)



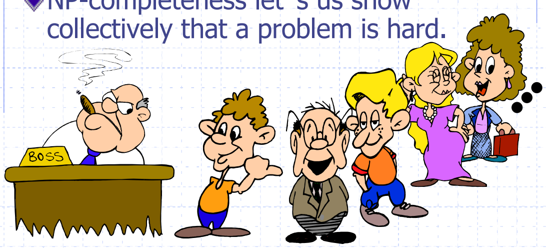
I couldn't find a polynomial-time algorithm, because no such algorithm exists!

 NP-Completeness (cartoon inspired by [Garey-Johnson, 79]) 4

4

Dealing with Hard Problems

- NP-completeness let's us show collectively that a problem is hard.

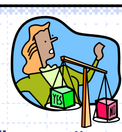


I couldn't find a polynomial-time algorithm, but neither could all these other smart people.

 NP-Completeness (cartoon inspired by [Garey-Johnson, 79]) 5

5

Polynomial-Time Decision Problems



- To simplify the notion of "hardness," we will focus on the following:
  - Polynomial-time as the cut-off for efficiency
  - Decision problems: output is 1 or 0 ("yes" or "no")
    - Examples:
    - Does a given graph G have an Euler tour?
    - Does a text T contain a pattern P?
    - Does an instance of 0/1 Knapsack have a solution with benefit at least K?
    - Does a graph G have an MST with weight at most K?

 NP-Completeness 6

6

P vs NP

- The **P versus NP problem** is a major unsolved problem in computer science. It asks whether every problem whose solution can be quickly verified can also be solved quickly.
-
- carries a US$1M for the first correct solution

Unsolved Sudoku Solved Sudoku

 NP-Completeness 7

7

NP example

- Problem: Decide if a graph has an MST of weight K
- Algorithm:
  1. Non-deterministically choose a set T of n-1 edges
  2. Test that T forms a spanning tree
  3. Test that T has weight at most K
- Analysis: Testing takes O(n+m) time, so this algorithm runs in polynomial time.

 NP-Completeness 8

8

Complexity Classes P, NP, NP-C

- class **P** = *decision problems* that can be solved on a deterministic sequential machine in an amount of time that is polynomial in the size of the input
- class **NP** = decision problems whose positive solutions can be verified in polynomial time given the right information, or equivalently, whose solution can be found in polynomial time on a non-deterministic machine.

 NP-Completeness 9

9

P versus NP complexity classes

- Clearly, **P** ⊆ **NP**.
- Arguably, the biggest open question in theoretical computer science concerns the relationship between those two classes:
- Is **P** equal to **NP**?

 NP-Completeness 10

10

NP-Complete complexity class

- **NP**-complete problems: set of problems
  - to each of which any other **NP**-problem can be reduced in polynomial time and
  - whose solution may still be verified in polynomial time.
- any **NP** problem can be transformed into any of the **NP**-complete problems.
- Informally, at least as "tough" as any other problem in **NP**.

 NP-Completeness 11

11

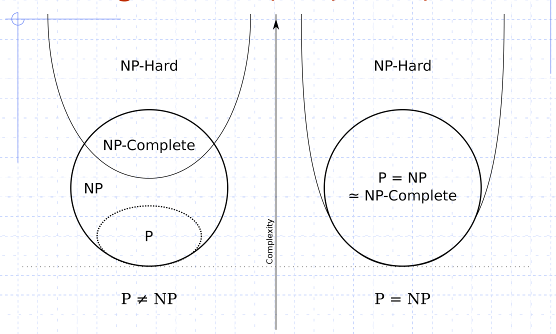
NP-Hard complexity class

- **NP**-hard problems are those at least as hard as **NP** problems
- all **NP** problems can be reduced (in polynomial time) to them.
- **NP**-hard problems need not be in **NP**, i.e., they need not have solutions verifiable in polynomial time.

 NP-Completeness 12

12

Euler Diagram for P, NP, NP-C, NP-Hard



 NP-Completeness 13

13

Major NPC problems

- SAT, 3-SAT
- Vertex Cover
- Clique
- Independent Set
- Hamiltonian Cycle
- Travelling Salesperson
- Partition
- Subset Sum
- Graph Coloring

 NP-Completeness 14

14

VERTEX-COVER problem

- given a graph G and an integer k, Is there is a vertex cover for G containing at most k vertices.
- *That is,* VERTEX-COVER asks whether there is a subset C of vertices of size at most k, such that for each edge (v,w), we have v ∈ C or w ∈ C.

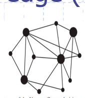


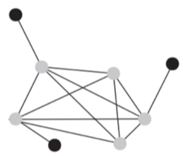
Figure 17.5: An instance of the VERTEX-COVER decision problem, where $k = 4$ and the answer is "yes." The vertices in the vertex cover are drawn as large disks.

 NP-Completeness 15

15

Clique

- A clique in a graph G is a subset C of vertices such that, for each v and w in C, with v <> w, (v, w) is an edge. That is, there is an edge between every pair of distinct vertices in C.
- Problem CLIQUE takes a graph G and an integer k as input and asks whether there is a clique in G of size at least k.



 NP-Completeness 17

17

Travelling Sales Person Problem

- given a set of N "cities" together with a distance function, d(v, w), which assigns an integer cost to each pair of cities, find a tour of all the cities that has minimum total cost.
- Viewing this as a decision problem, or language TSP, we assume we are also given an integer k, and we are asked whether there is a cycle that visits each city exactly once, returning to the starting city, such that the total cost of the tour is at most k. (See Figure 17.3.)

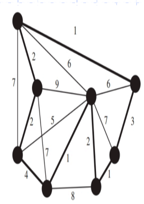


Figure 17.3: An example instance of the TSP decision problem, where $k = 18$ and the answer is "yes." City pairs with a finite cost between them are drawn as an edge along with its integer cost; missing edges are for city pairs with infinite cost between them. A tour with total cost at most k is drawn with bold edges.

 NP-Completeness 18

18

Is a problem Q in NP-C?

- Many ways
- Use reduction
  - Solve a problem using another problem
- A is easier than B
  - write A < B
  - if we can write down an algorithm for solving A that uses a small number of calls to a subroutine for B (with everything outside the subroutine calls being fast, polynomial time)

 NP-Completeness 19

19

Hamiltonian Cycle & Longest Paths

- Does a given graph G have a cycle visiting each vertex exactly once?
- solve using longest path as a subroutine:

```
for each edge (u,v) of G
    if there is a simple path of length n-1 from u to v
        return yes    // path + edge form a cycle
    return no
```

- This algorithm makes m calls to a longest path subroutine, and does $O(m)$ work outside those subroutine calls, so it shows that HamiltonianCycle < LongestPath.
- (It doesn't show that Hamiltonian cycle is in P, because we don't know how to solve the longest path problem quickly.)

20

RDS problem

- Given n jobs: 1, 2, …, i, …, n and a machine M
  - release times: r_i for job i
  - deadlines: d_i for job i
  - machine times: t_i for job i
- Is there a feasible schedule of the jobs on M?

21

RDS is NP-Complete

- RDS is in NP
  - positive solution verification
  - Given a schedule, it is easy to verify in $O(n^2)$ time whether job i is scheduled between r_i and d_i and that $(d_i - r_i) \geq t_i$. Also the schedules of jobs i and j don't overlap whenever $i \neq j$.
- Poly-time Reduction of the Partition problem PART < RDS
- PART: Given a set A = {a1, a2, …, an}, is there a partition $S \ni \Sigma\{ai \mid ai \in S\} = \Sigma\{ai \mid ai \in A - S\}$

NOTE: if we solve RDS using PART (i.e., RDS < PART), it does NOT show that RDS is NP-complete

22

PART < RDS : Poly-time Reduction

- Given an instance I of PART: $A=\{a_1, a_2, \dots, a_n\}$
- Generate the instance I′ of RDS with (n+1)-jobs as follows: Let $W = \Sigma\, a_i$

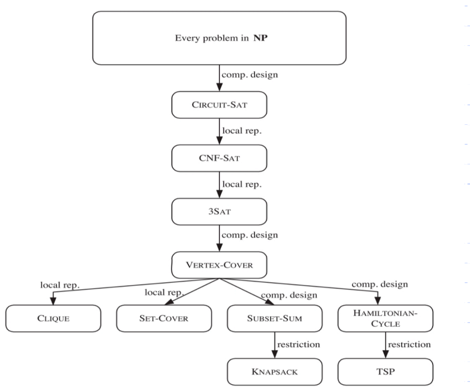
  $r_i = 0$; $d_i = W+1$; $t_i = a_i$; $1 \le i \le n$

  $r_{n+1} = \frac{W}{2}$; $d_{n+1} = \frac{W}{2}+1$; $t_{n+1} = 1$;
- Note if W is not even, partition of A does not exist (i.e., no solution to PART)
- Job (n+1) acts an an enforcer of a partition of A if it exists
- Now, Show that

  Solution to I′ of RDS iff Solution to I of PART

© 2015 Goodrich and Tamassia NP-Completeness 23

23



© 2015 Goodrich and Tamassia NP-Completeness 24

24

Problems and Languages

- A **language** L is a set of strings defined over some alphabet Σ
- Every decision algorithm A defines a language L
  - L is the set consisting of every string x such that A outputs "yes" on input x.
  - We say "A **accepts** x' ' in this case
    - Example:
    - If A determines whether or not a given graph G has an Euler tour, then the language L for A is all graphs with Euler tours.

© 2015 Goodrich and Tamassia NP-Completeness 25

25

The Complexity Class P

- A **complexity class** is a collection of languages
- P is the complexity class consisting of all languages that are accepted by **polynomial-time** algorithms
- For each language L in P there is a polynomial-time decision algorithm A for L.
  - If n=|x|, for x in L, then A runs in p(n) time on input x.
  - The function p(n) is some polynomial

 NP-Completeness 26

26

The Complexity Class NP

- We say that an algorithm is non-deterministic if it uses the following operation:
  - Choose(b): chooses a bit b
  - Can be used to choose an entire string y (with |y| choices)
- We say that a non-deterministic algorithm A **accepts** a string x if there exists some sequence of choose operations that causes A to output "yes" on input x.
- NP is the complexity class consisting of all languages accepted by **polynomial-time non-deterministic** algorithms.

 NP-Completeness 27

27

NP example

- Problem: Decide if a graph has an MST of weight K
- Algorithm:
  1. Non-deterministically choose a set T of n-1 edges
  2. Test that T forms a spanning tree
  3. Test that T has weight at most K
- Analysis: Testing takes O(n+m) time, so this algorithm runs in polynomial time.

 NP-Completeness 28

28

The Complexity Class NP Alternate Definition

- We say that an algorithm B **verifies** the acceptance of a language L if and only if, for any x in L, there exists a certificate y such that B outputs "yes" on input (x,y).
- NP is the complexity class consisting of all languages verified by **polynomial-time** algorithms.

- We know: P is a subset of NP.
- Major open question: P=NP?
- Most researchers believe that P and NP are different.

29

NP example (2)

- Problem: Decide if a graph has an MST of weight K

- Verification Algorithm:
  1. Use as a certificate, y, a set T of n-1 edges
  2. Test that T forms a spanning tree
  3. Test that T has weight at most K

- Analysis: Verification takes O(n+m) time, so this algorithm runs in polynomial time.

30

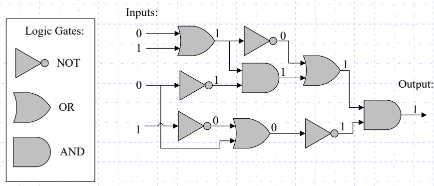
Equivalence of the Two Definitions

- Suppose A is a non-deterministic algorithm
  - Let y be a certificate consisting of all the outcomes of the choose steps that A uses
  - We can create a verification algorithm that uses y instead of A's choose steps
  - If A accepts on x, then there is a certificate y that allows us to verify this (namely, the choose steps A made)
  - If A runs in polynomial-time, so does this verification algorithm
- Suppose B is a verification algorithm
  - Non-deterministically choose a certificate y
  - Run B on y
  - If B runs in polynomial-time, so does this non-deterministic algorithm

31

An Interesting Problem

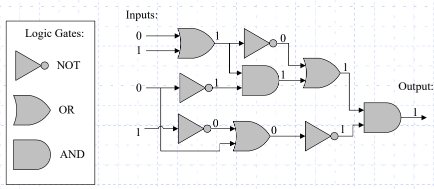
- A Boolean circuit is a circuit of AND, OR, and NOT gates; the CIRCUIT-SAT problem is to determine if there is an assignment of 0's and 1's to a circuit's inputs so that the circuit outputs 1.



32

CIRCUIT-SAT is in NP

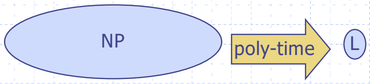
- Non-deterministically choose a set of inputs and the outcome of every gate, then test each gate's I/O.



33

NP-Completeness

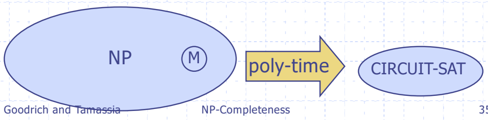
- A problem (language) L is **NP-hard** if every problem in NP can be reduced to L in polynomial time.
- That is, for each language M in NP, we can take an input x for M, **transform** it in polynomial time to an input x' for L such that x is in M if and only if x' is in L.
- L is **NP-complete** if it's in NP and is NP-hard.



34

Cook-Levin Theorem

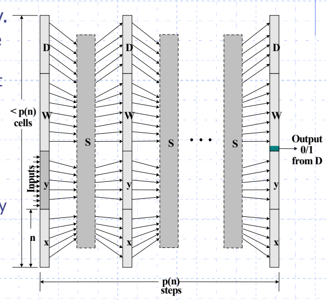
- CIRCUIT-SAT is NP-complete.
  - We already showed it is in NP.
- To prove it is NP-hard, we have to show that every language in NP can be reduced to it.
  - Let M be in NP, and let x be an input for M.
  - Let y be a certificate that allows us to verify membership in M in polynomial time, p(n), by some algorithm D.
  - Let S be a circuit of size at most $O(p(n)^2)$ that simulates a computer (details omitted…)



35

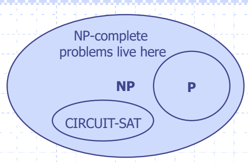
Cook-Levin Proof

- We can build a circuit that simulates the verification of x's membership in M using y.
  - Let W be the working storage for D (including registers, such as program counter); let D be given in RAM "machine code."
  - Simulate p(n) steps of D by replicating circuit S for each step of D. Only input: y.
  - Circuit is satisfiable if and only if x is accepted by D with some certificate y
  - Total size is still polynomial: $O(p(n)^3)$.



36

Some Thoughts about P and NP



- Belief: P is a proper subset of NP.
- Implication: the NP-complete problems are the hardest in NP.
- Why: Because if we could solve an NP-complete problem in polynomial time, we could solve every problem in NP in polynomial time.
- That is, if an NP-complete problem is solvable in polynomial time, then P=NP.
- Since so many people have attempted without success to find polynomial-time solutions to NP-complete problems, showing your problem is NP-complete is equivalent to showing that a lot of smart people have worked on your problem and found no polynomial-time algorithm.

37