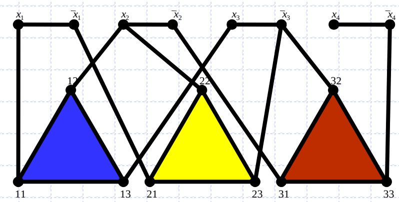
Presentation for use with the textbook, Algorithm Design and Applications, by M. T. Goodrich and R. Tamassia, Wiley, 2015

NP-Completeness

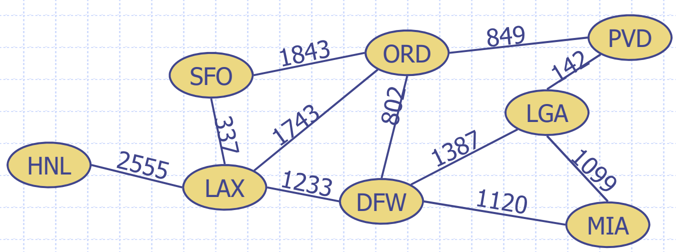


1

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

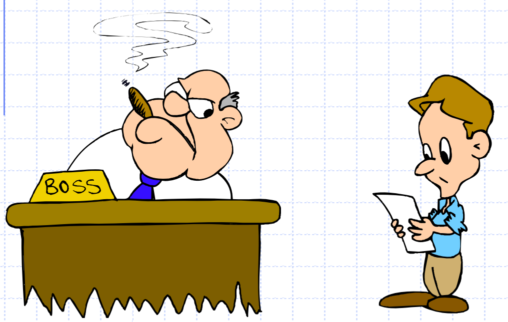


2

2

Dealing with Hard Problems

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



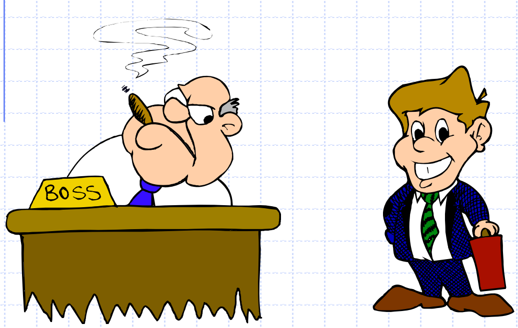
I couldn't find a polynomial-time algorithm;
I guess I'm too dumb.

(cartoon inspired by [Garey-Johnson, 79]) 3

3

Dealing with Hard Problems

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



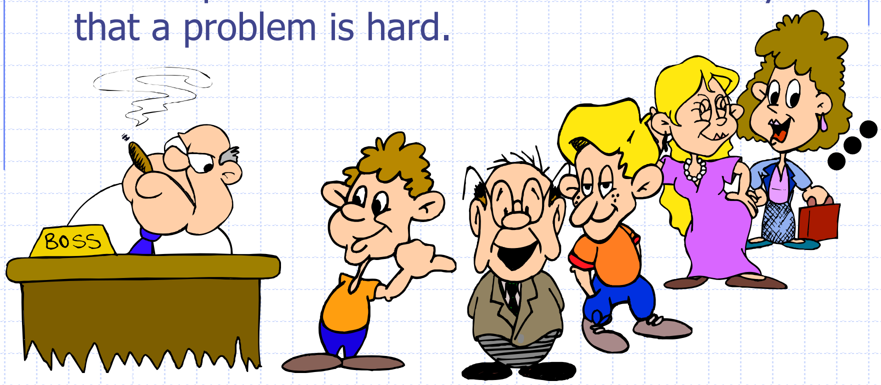
I couldn't find a polynomial-time algorithm,
because no such algorithm exists!

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

4

Dealing with Hard Problems

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



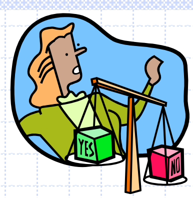
I couldn't find a polynomial-time algorithm,
but neither could all these other smart people.

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

5

5

P: Polynomial-Time Decision Problems



- To simplify the notion of "hardness," we will focus on the following:
  - Polynomial-time as the cut-off for efficiency: a problem is **hard** if it doesn't have a polynomial-time algorithm
  - Decision problems: output is 1 or 0 ("yes" or "no")
  - Examples:
    - Does a given graph G have a Hamiltonian 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?

6

6

The Complexity Class NP

- We say that an algorithm is non-deterministic if it uses the following operation:
  - **Choose(n)**: non-deterministically chooses a value k, 0 ≤k < n.

  Choose(n) looks like random(n), but it may make wise choices.
  Can be used to choose a list of numbers.
- We say that a non-deterministic algorithm A **accepts** an input 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 problems accepted by **polynomial-time non-deterministic** algorithms.

7

7

NP example

- Problem: Decide if TSP has a tour bounded by K
- Algorithm getTSP(V, E)

```
// Non-deterministically choose a set T of n edges:
T = { };
while (|T| < n) {
    k = choose(|E|);
    move e_k from E to T; }
Test that T forms a tour
Test that T has weight at most K
If both tests are okay, return "yes" else return "no"
```

- Analysis: the while loop and testing takes O(n+m) time, so this algorithm runs in polynomial time.

8

8

The Complexity Class NP Alternate Definition

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

9

9

NP example (2)

- Problem: Decide if TSP has a tour bounded by K
- Verification Algorithm:
  VerifyTSP(V, E, T)
  1. Use T as a certificate, where T is a set of n edges
  2. Test that T forms a tour
  3. Test that T has weight at most K
  4. If both tests are okay, return "yes", otherwise, "no"
- Analysis: Verification takes O(n+m) time, so this algorithm runs in polynomial time.

10

10

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 B 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 B.
- Suppose B is a verification algorithm
  - Non-deterministically choose a certificate y
  - Run B on x and y
  - If B(x, y) runs in polynomial-time, so does this non-deterministic algorithm A.

11

11

getTSP vs VerifyTSP

- Problem: Decide if TSP has a tour bounded by K
- Algorithm getTSP(V, E)

  // Non-deterministically choose a set T of n edges:
  T = { };
  while (|T| < n) { k = choose(|E|); move e_k from E to T; }
  Test that T forms a tour
  Test that T has weight at most K
  If both tests are okay, return "yes" else return "no"
- VerifyTSP(V, E, T)
  1. Use T as a certificate, where T is a set of n edges
  2. Test that T forms a tour
  3. Test that T has weight at most K
  4. If both tests are okay, return "yes", otherwise, "no"

12

12

The most famous open problem in Computer Science

- By (either) definition, P is a subset of NP.
- Major open question: P = NP?
- Most researchers believe that P and NP are different.

13

13

Possible Quiz Question

- The decision version of the Knapsack problem: Given a collection of items with weights w_i and benefits v_i, $1 \leq i \leq n$, is there a subset of items whose total weight is at most W and whose total benefit is at least K?
- Show this decision problem is in NP by providing a polynomial-time verification algorithm.

14

14

Polynomial-Time Reduction

- Suppose we could solve Y in polynomial-time by algorithm B. Can we solve problem X using B in polynomial time?
- **Reduction**: Problem X polynomially reduces to problem Y if arbitrary instances (i.e., inputs) of problem X can be solved using:
  - Polynomial number of standard computational steps, plus
  - Polynomial number of calls to the algorithm B that solves problem Y.
  - Notation: $X \le_P Y$.

15

15

Polynomial-Time Reduction

If $X \le_P Y$, and assume the algorithm to solve Y is B, we may obtain the algorithm A which uses B to solve X (the time spent by B is not cared).

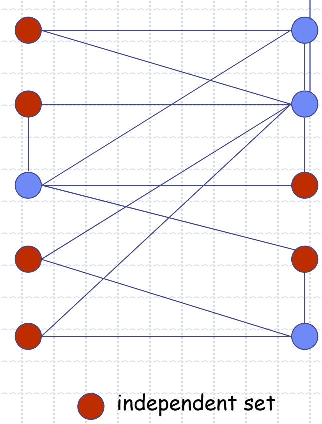
- **Design algorithms.** If $X \le_P Y$ and Y can be solved in polynomial-time, then X can also be solved in polynomial time. That is, if Y is easy, so is X.
- **Establish equivalence.** If $X \le_P Y$ and $Y \le_P X$, we use notation $X \equiv_P Y$.
- **Prove Hardness of Y**: If $X \le_P Y$ and X is known to be hard, then Y must be hard as well.

16

16

Independent Set

- INDEPENDENT SET: Given a graph G = (V, E) and an integer k, is there a subset of vertices S ⊆ V such that |S| ≥ k, and for each edge at most one of its endpoints is in S?
- Ex. Is there an independent set of size ≥ 6? Yes.
- Ex. Is there an independent set of size ≥ 7? No.



17

17

Clique

- Clique: Given a graph G = (V, E) and an integer k, is there a subset of vertices S ⊆ V such that |S| ≥ k, and for each pair (x, y) of points in S, (x, y) is an edge of E?

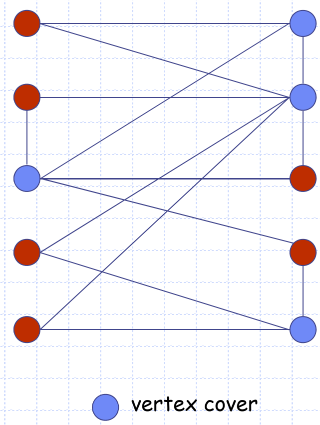
- Claim. CLIQUE $\equiv_P$ INDEPENDENT-SET.

Proof. We show S is an independent set of G iff S is a clique of G′, where G′ is the complement of G: G′ = (V, V^2 – E).

18

Vertex Cover

- VERTEX COVER: Given a graph G = (V, E) and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in S?



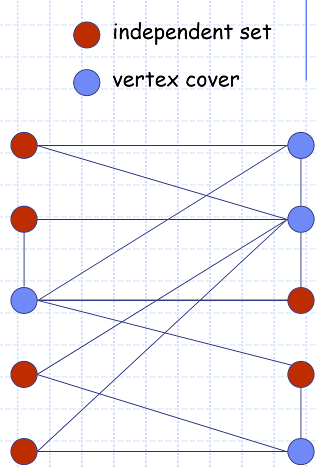
- Ex. Is there a vertex cover of size ≤ 4? Yes.
- Ex. Is there a vertex cover of size ≤ 3? No.

19

19

Vertex Cover and Independent Set

- Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET.



Proof. We show S is an independent set iff V – S is a vertex cover.

Consequently, S is a maximum independent set iff V – S is a minimum vertex cover. If we have an efficient algorithm to solve one, we will have efficient algorithm to solve the other.

20

Vertex Cover and Independent Set

- Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET.
- Proof. We show S is an independent set iff V – S is a vertex cover.
- $\Rightarrow$
  - Let S be any independent set.
  - Consider an arbitrary edge (u, v).
  - S independent $\Rightarrow$ u $\notin$ S or v $\notin$ S $\Rightarrow$ u $\in$ V – S or v $\in$ V – S.
  - Thus, V – S covers (u, v).
- $\Leftarrow$
  - Let V – S be any vertex cover.
  - Consider two nodes u $\in$ S and v $\in$ S.
  - Observe that (u, v) $\notin$ E since V – S is a vertex cover.
  - Thus, no two nodes in S are joined by an edge $\Rightarrow$ S independent set. ▪

21

21

Set Cover

- SET COVER: Given a set U of elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of U, and an integer k, does there exist a collection of $\leq$ k of these sets whose union is equal to U?

- Sample application.
  - m available pieces of software.
  - Set U of n capabilities that we would like our system to have.
  - The i^{th} piece of software provides the set $S_i \subseteq U$ of capabilities.
  - Goal: achieve all n capabilities using fewest pieces of software.

- Ex:

U = { 1, 2, 3, 4, 5, 6, 7 }
k = 2

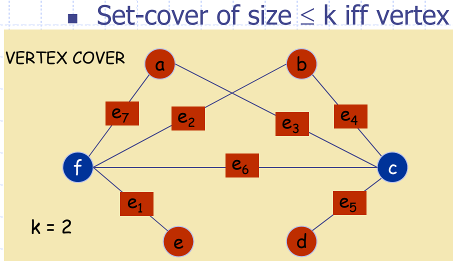
S_1 = {3, 7}	S_4 = {2, 4}
S_2 = {3, 4, 5, 6}	S_5 = {5}
S_3 = {1}	S_6 = {1, 2, 6, 7}

22

22

Vertex Cover Reduces to Set Cover

- Claim. VERTEX-COVER $\leq_P$ SET-COVER.
- Proof. Given a VERTEX-COVER instance G = (V, E), k, we construct a set cover instance whose size equals the size of the vertex cover instance.
- Construction.
  - Create SET-COVER instance:
    - k = k, U = E, S_v = {e ∈ E : e incident to v }
  - Set-cover of size ≤ k iff vertex cover of size ≤ k. ▪



SET COVER

U = { 1, 2, 3, 4, 5, 6, 7 }
k = 2
S_a = {3, 7} S_b = {2, 4}
S_c = {3, 4, 5, 6} S_d = {5}
S_e = {1} S_f = {1, 2, 6, 7}

23

23

Decision vs Optimization: Clique

For the clique problem, suppose we have an algorithm A to answer the decision problem: A(G, k) = yes iff G has a clique of size k. How can we find the maximum clique of G?

- Step 1: Decide k, the size of the maximum clique:
  for i from n downto 1, if A(G, i) = yes and A(G, i+1) = no, then return i;
- Step 2: Decide the actual max clique:
  for each vertex v in G
      if (A(G - v, k) = yes) G = G - v;
      // G – v means v is deleted from G.

- Suppose we have an algorithm B which returns the maximum clique of G. How can we solve the decision version of the clique problem?

24

Decision vs Optimization: Vertex Cover

- Decision Problem:
  - Input: a graph $G = (V, E)$, integer k.
  - Question: Does G have a vertex cover of size $\leq$ k?
- Optimization problem. Find vertex cover of minimum cardinality of G.

- Self-reducibility: Decision and Optimization Problems are all equivalent ($=_P$)
- Applies to all (NP-complete) problems.
  - Justifies our focus on decision problems.

25

25

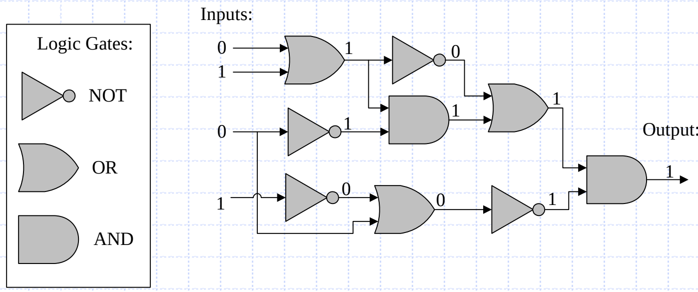
Vertex Cover Problem

- Ex: Reduce Optimization to Decision
- To find min cardinality vertex cover.
  - (Binary) search for cardinality k* of min vertex cover.
  - Find a vertex v such that G – v has a vertex cover of size $\leq$ k* - 1. (delete v and all incident edges)
    - any vertex in any min vertex cover will have this property
  - Include v in the vertex cover.
  - Recursively find a min vertex cover in G – v.

26

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.

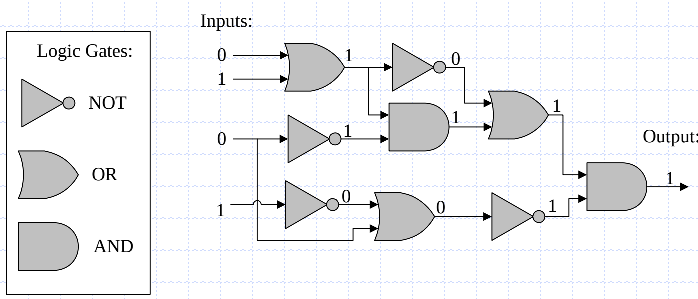


27

27

CIRCUIT-SAT is in NP

- Non-deterministically choose a set of Boolean values for all inputs of the circuit, then test each gate's I/O.

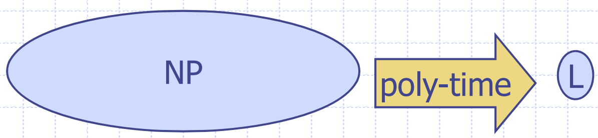


28

28

NP-Completeness

- A problem L is **NP-hard** if every problem X in NP can be reduced to L in polynomial time.
- That is, for each problem X in NP, we can take an input x for X, **transform** x 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.

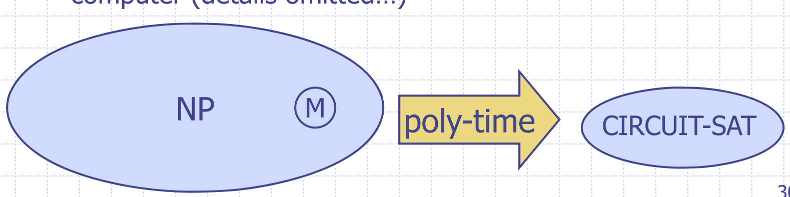


29

29

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 problem 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(x, y).
  - Let S be a circuit of size at most $O(n^{2c})$ that simulates a computer (details omitted…)

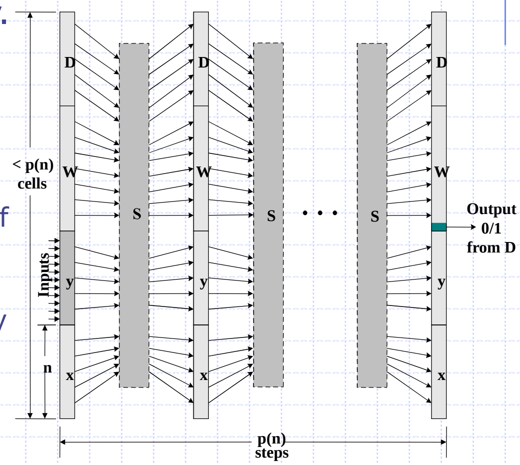


30

30

Cook-Levin Proof

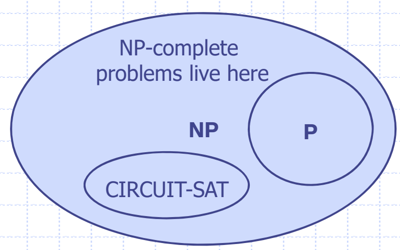
- We can build a circuit that simulates the verification of x's membership in M using y.
  - Let W be the working storage for D (including registers, such as program counter); let D be given in RAM "machine code."
  - Simulate $O(n^c) = p(n)$ steps of D by replicating circuit S for each step of D. Only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



31

31

Thoughts about P and NP



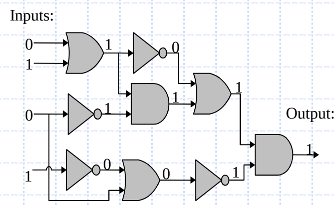
- Belief: P is a proper subset of NP.
- Implication: the NP-complete problems are the hardest in NP.
- Why: Because if we could solve an NP-complete problem in polynomial time, we could solve every problem in NP in polynomial time.
- That is, if an NP-complete problem is solvable in polynomial time, 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.

32

32

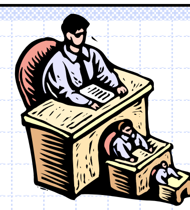
Recall of Definitions

- A decision problem M can be described by the set of true-instances of M. For example, let PRIMES be the problem of deciding a number is prime, then PRIMES = { 2, 3, 5, 7, … }.
- A problem M is polynomial-time **reducible** to a problem L if an instance (input) x for M can be transformed in polynomial time to an instance y for L such that x is in M iff y is in L. That is, x is a true-instance of M if and only if y is a true-instance of L.
  - Denote this by $M \le_P L$.
- A problem L is **NP-hard** if every problem in NP is polynomial-time reducible to L.
- A problem is **NP-complete** if it is in NP and it is NP-hard.
- CIRCUIT-SAT is NP-complete:
  - CIRCUIT-SAT is in NP
  - For every M in NP, $M \le_P$ CIRCUIT-SAT.



33

Transitivity of Reducibility

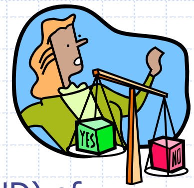


- If $A \le_P B$ and $B \le_P C$, then $A \le_P C$.
  - An input x for A can be converted to y for B, such that x is in A if and only if y is in B. Likewise, for B to C.
  - Convert y into z for C such that y is in B iff z is in C.
  - Hence, if x is in A, y is in B, then z is in C.
  - Likewise, if z is in C, y is in B, then x is in A.
  - Thus, $A \le_P C$, since polynomials are closed under composition.

34

34

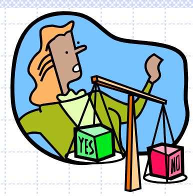
SAT



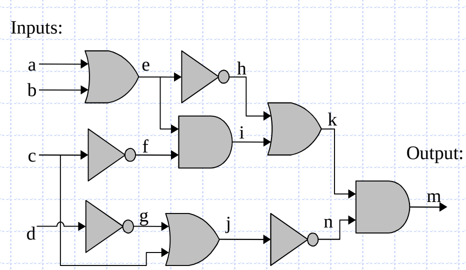
- A CNF Boolean formula is a conjunction (AND) of clauses; a clause is a disjunction (OR) of literals; a a literal is a variable or negation of a variable:
  - $(a \vee b \vee \neg d \vee e) \wedge (\neg a \vee \neg c) \wedge (\neg b \vee c \vee d \vee e) \wedge (a \vee \neg c \vee \neg e)$
  - OR: $\vee$, AND: $\wedge$, NOT: $\neg$
- SAT: Given a Boolean formula S, is S satisfiable, that is, can we assign 0's and 1's to the variables so that S is 1 ("true")?
  - Easy to see that SAT is in NP:
    - Non-deterministically choose an assignment of 0's and 1's to the variables and then evaluate each clause. If they are all 1 ("true"), then the formula is satisfiable. 35

35

SAT is NP-complete



- Reduce CIRCUIT-SAT to SAT.
  - Given a Boolean circuit, make a variable for every input and gate.
  - Create a sub-formula for each gate, characterizing its effect. Form the formula as the output variable AND-ed with all these sub-formulas:
    - Ex: $m \wedge ((a \vee b) \leftrightarrow e) \wedge (c \leftrightarrow \neg f) \wedge (d \leftrightarrow \neg g) \wedge (e \leftrightarrow \neg h) \wedge (e \wedge f \leftrightarrow i) \ldots$



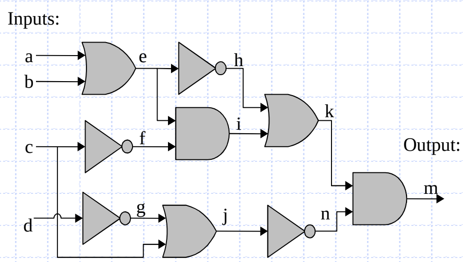
The formula is satisfiable if and only if the Boolean circuit is satisfiable.

36

36

3SAT

- The SAT problem is still NP-complete even if the formula is a conjunction of disjuncts, that is, it is in conjunctive normal form (CNF).
- The SAT problem is still NP-complete even if it is in CNF and every clause has just 3 literals (a variable or its negation):
  - $(a \lor b \lor \neg d) \land (\neg a \lor \neg c \lor e) \land (\neg b \lor d \lor e) \land (a \lor \neg c \lor \neg e)$
- SAT and CIRCUIT-SAT are equivalent. Reduction from SAT. E.g., $m \land ((a \lor b) \leftrightarrow e) \land (c \leftrightarrow \neg f) \land (d \leftrightarrow \neg g) \land (e \leftrightarrow \neg h) \land (e \land f \leftrightarrow i) \ldots$

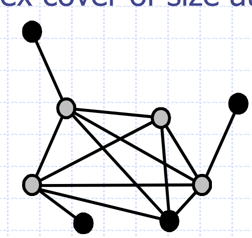


37

37

Vertex Cover

- A vertex cover of graph G=(V,E) is a subset W of V, such that, for every edge (a,b) in E, a is in W or b is in W.
- VERTEX-COVER: Given a graph G and an integer K, is does G have a vertex cover of size at most K?



- VERTEX-COVER is in NP: Non-deterministically choose a subset W of size K and check that every edge is covered by W.

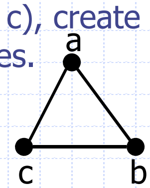
38

38

Vertex-Cover is NP-complete

- Reduce 3SAT to VERTEX-COVER.
  - A CNF is a conjunction of m clauses.
  - Let S be a Boolean formula in CNF with each clause having 3 literals (i.e., variables or negation of variables).
  - For each variable x, create a node for x and ¬x, and connect these two:

  - For each clause (a ∨ b ∨ c), create a triangle and connect these three nodes.

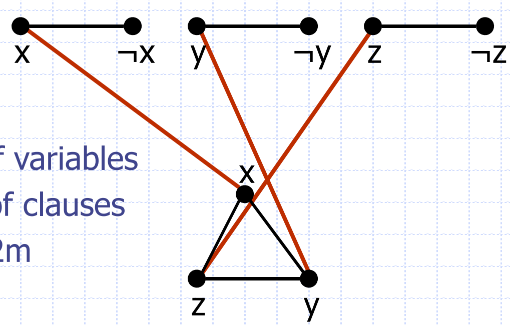


39

39

Vertex-Cover is NP-complete

- Completing the construction
  - Connect each literal in a clause triangle to its copy in a variable pair.
  - E.g., a clause (x ∨ y ∨ z)



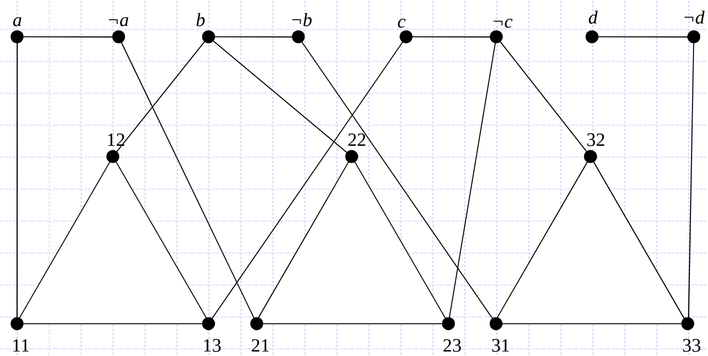
  - Let n = # of variables
  - Let m = # of clauses
  - Set K = n+2m

40

40

Vertex-Cover is NP-complete

- Example: $(a \vee b \vee c) \wedge (\neg a \vee b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg d)$
- Graph has vertex cover of size K=4+6=10 iff formula is satisfiable.



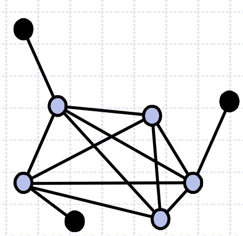
41

41

Clique

- A **clique** of a graph G=(V,E) is a subgraph C that is fully-connected (every pair in C has an edge).
- CLIQUE: Given a graph G and an integer K, is there a clique in G of size at least K?

This graph has a clique of size 5



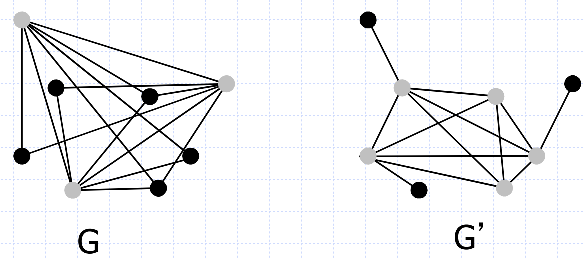
- CLIQUE is in NP: non-deterministically choose a subset C of size K and check that every pair in C has an edge in G.

42

42

CLIQUE is NP-Complete

- Reduction from VERTEX-COVER.
- A graph G has a vertex cover of size K if and only if it' s complement has a clique of size n-K.



43

43

Possible Quiz Question

- An **independent set** of a graph G=(V,E) is a subset S of V such that there are no edges in E connecting any two points of S.
- INDEPENDENT-SET: Given a graph G and an integer K, is there an independent set in G of size at least K?

Prove formally that INDEPENDENT-SET is NP-complete.

44

44

Some Other NP-Complete Problems

- SET-COVER: Given a collection of m sets, are there K of these sets whose union is the same as the whole collection of m sets?
  - NP-complete by reduction from VERTEX-COVER
- SUBSET-SUM: Given a set of integers and a distinguished integer K, is there a subset of the integers that sums to K?
  - NP-complete by reduction from VERTEX-COVER

45

45

Some Other NP-Complete Problems

- 0/1 Knapsack: Given a collection of items with weights and benefits, is there a subset of weight at most W and benefit at least K?
  - NP-complete by reduction from SUBSET-SUM
- Hamiltonian-Cycle: Given an graph G, is there a cycle in G that visits each vertex exactly once?
  - NP-complete by reduction from VERTEX-COVER
- Traveling Salesman Tour: Given a complete weighted graph G, is there a cycle that visits each vertex and has total cost at most K?
  - NP-complete by reduction from Hamiltonian-Cycle.

46

46

Integer Linear Programming

Types of Integer Linear Programming Models
Graphical Solution for an All-Integer LP
Spreadsheet Solution for an All-Integer LP
Application Involving 0-l Variables
Special 0-1 Constraints

47

Example: Integer Linear Programming

Consider the following all-integer linear program:

$$\text{Max} \quad 3x_1 + 2x_2$$

$$\begin{aligned} \text{s.t.} \quad 3x_1 + x_2 &\le 9 \\ x_1 + 3x_2 &\le 7 \\ -x_1 + x_2 &\le 1 \end{aligned}$$

$x_1, x_2 \ge 0$ and integer

48

Integer Linear Programming

A linear program in which all the variables are restricted to be integers is called an integer linear program (ILP).

If only a subset of the variables are restricted to be integers, the problem is called a mixed integer linear program (MILP).

Binary variables are variables whose values are restricted to be 0 or 1. If all variables are restricted to be 0 or 1, the problem is called a 0-1 or binary integer program.

49

Special 0-1 Constraints

When x_i and and x_j represent binary variables designating whether projects *i* and *j* have been completed, the following special constraints may be formulated:

- At most *k* out of *n* projects will be completed: $\Sigma x_j \leq k$
- Project *j* is conditional on project i: $x_j - x_i \leq 0$
- Project *i* is a co-requisite for project *j*: $x_j - x_i = 0$
- Projects *i* and *j* are mutually exclusive: $x_i + x_j \leq 1$

50

Decision Problem: 0-1 Programming

0-1 PROGRAMMING. Given a n by m matrix A, a vector B of m numbers, a vector X of n variables, is there a binary solution of X such that $AX \le B$?

Claim. 3-SAT $\le_P$ 0-1 PROGRAMMING.

Pf.

51

51

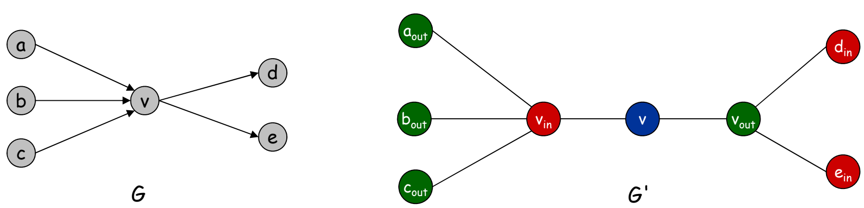
Directed Hamiltonian Cycle

DIR-HAM-CYCLE: given a digraph G = (V, E), does there exists a simple directed cycle Γ that contains every node in V?

Claim. HAM-CYCLE $\le_P$ DIR-HAM-CYCLE.

Claim. DIR-HAM-CYCLE $\le_P$ HAM-CYCLE.

Pf. Given a directed graph G = (V, E), construct an undirected graph G' with 3|V| nodes.



52

52

Directed Hamiltonian Cycle

Claim. G has a Hamiltonian cycle iff G' does.

Pf. ⇒

- Suppose G has a directed Hamiltonian cycle Γ.
- Then G' has an undirected Hamiltonian cycle (same order).

Pf. ⇐

- Suppose G' has an undirected Hamiltonian cycle Γ'.
- Γ' must visit nodes in G' using one of following two orders:
  …, B, G, R, B, G, R, B, G, R, B, …
  …, B, R, G, B, R, G, B, R, G, B, …
- Blue nodes in Γ' make up directed Hamiltonian cycle Γ in G, or reverse of one. ▪

53

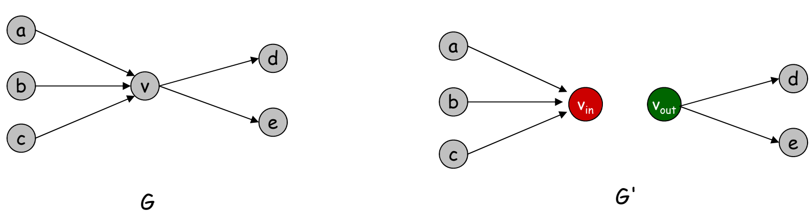
53

Directed Hamiltonian Path

DIR-HAM-PATH: given a digraph $G = (V, E)$, does there exists a simple directed path Γ that contains every node in V?

Claim. DIR-HAM-CYCLE $\leq_P$ DIR-HAM-PATH.

Pf. Given a directed graph $G = (V, E)$, construct a directed graph G' with $|V|+1$ nodes.



54

54

Directed Hamiltonian Path

DIR-HAM-PATH: given a digraph $G = (V, E)$, does there exists a simple directed path Γ that contains every node in V?

Claim. DIR-HAM-PATH $\leq_P$ DIR-HAM-CYCLE.

Pf. Given a directed graph $G = (V, E)$, construct a directed graph G' with $|V|+2$ nodes: $G' = (V \cup \{s, t\}, E \cup \{ (s, x), (x, t), (t, s) \mid x \in V \})$.

55

55

Traveling Salesman Problem

Traveling Salesman Problem (TSP): Given a complete graph with nonnegative edge costs, find a minimum cost cycle visiting every vertex exactly once.

Example: Given a number of cities and the costs of traveling from any city to any other city, what is the cheapest round-trip route that visits each city exactly once and then returns to the starting city

TSP: Given a complete weighted graph $G = (V, E, W)$ and an number d, does there exists a simple cycle Γ that contains every node in V and its total weight is bounded by d?

Claim. HAM-CYCLE $\leq_P$ TSP.

Pf. Given a graph $G = (V, E)$, construct a complete weighted graph $G' = (V, V\times V, W)$, such that $W(e) = 1$ for e in E and $W(e) = 2$ for e not in E, and $d = |V|$.

56

Subset Sum

SUBSET-SUM. Given a set of natural numbers $w_1, \ldots, w_n$ and an integer W, is there a subset that adds up to exactly W?

Ex: { 1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344 }, W = 3754.
Yes. 1 + 16 + 64 + 256 + 1040 + 1093 + 1284 = 3754.

Remark. With arithmetic problems, input integers are encoded in binary. Polynomial reduction must be polynomial in the size of binary encoding.

Claim. 3-SAT $\le_P$ SUBSET-SUM.
Pf. Given an instance Φ of 3-SAT, we construct an instance of SUBSET-SUM that has solution iff Φ is satisfiable.

57

57

Subset Sum

Construction. Given 3-SAT instance Φ with n variables and k clauses, form 2n + 2k decimal integers, each of n+k digits, as illustrated below.

Claim. Φ is satisfiable iff there exists a subset that sums to W.
Pf. No carries possible.

$C_1 = \bar{x} \vee y \vee z$
$C_2 = x \vee \bar{y} \vee z$
$C_3 = \bar{x} \vee \bar{y} \vee \bar{z}$

	x	y	z	C_1	C_2	C_3	
x	1	0	0	0	1	0	100,010
¬ x	1	0	0	1	0	1	100,101
y	0	1	0	1	0	0	10,100
¬ y	0	1	0	0	1	1	10,011
z	0	0	1	1	1	0	1,110
¬ z	0	0	1	0	0	1	1,001
dummies to get clause columns to sum to 4	0	0	0	1	0	0	100
	0	0	0	2	0	0	200
	0	0	0	0	1	0	10
	0	0	0	0	2	0	20
	0	0	0	0	0	1	1
	0	0	0	0	0	2	2
W	1	1	1	4	4	4	111,444

58

58

Set Partition

PARTITION. Given a set of natural numbers $w_1, \ldots, w_n$, is there a subset that adds up to exactly half sum of all w_i?

Claim. PARTITION $\leq_P$ SUBSET-SUM.

Pf. PARTITION is a special of SUBSET-SUM.

Claim. SUBSET-SUM $\leq_P$ PARTITION.

Pf.

59

59

Possible Quiz Question

- Given an instance (S, k) of SubsetSum problem, where S is a set of integers and k is another integer, we construct $S' = S \cup \{ x, y \}$, where $x = sum(S) + k$, $y = 2sum(S) - k$, and $sum(S) = \sum_{x \in S} x$. Prove that S' can be constructed from S in polynomial time and there exists a subset $X \subseteq S'$ such that $sum(X) = k$ iff S' can be partitioned into X and Y such that $sum(X) = sum(Y)$, where $S' = X \cup Y$ and $X \cap Y = \emptyset$.

60

60

Bin Packing

BIN-PACKING. Given a set S of real numbers $w_1, \ldots, w_n$, $0 < w_i \leq 1$, and integer K, is there a partition of S into K subsets such that each subset adds up no more than 1?

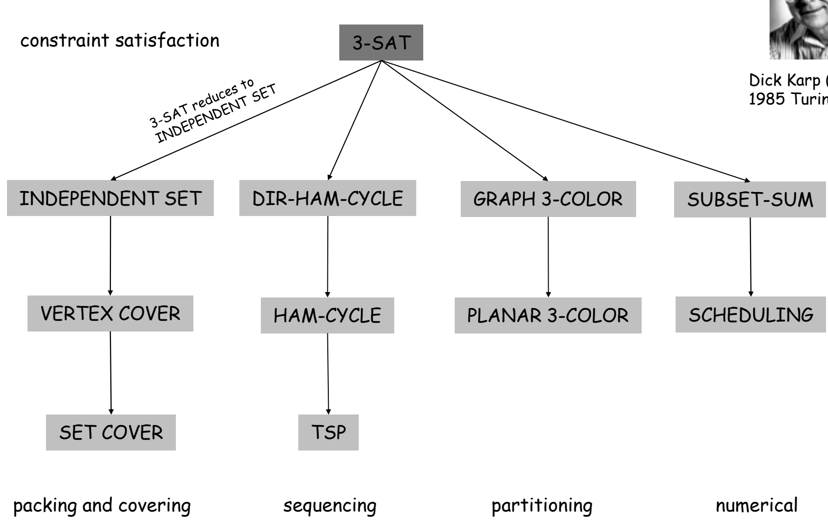
Claim. PARTITION $\leq_P$ BIN-PACKING.

Pf.

61

61

Polynomial-Time Reductions



Dick Karp (1972)
1985 Turing Award

62

62