# Analysis of Algorithms

## NP & NP-Complete

Prof. Muhammad Saeed

## Classes Of Problems

1. P
2. NP
3. NP-Complete (NPC, NP-C)

# …… Classes Of Problems

1. **P:**

   Consists of those problems that are solvable in polynomial time. More specifically, they are problems that can be solved in time $O(n^k)$ for some constant k, where n is the size of the input to the problem. They are called easy or tractable. Problems that require superpolynomial time as being intractable, or hard.

2. **NP:**

   Consists of those problems that are verifiable in polynomial time, it means if we were somehow given a 'certificate' of a solution, then we could verify that the certificate is correct in polynomial time in the size of the input to the problem.

3. **NPC (NP-Complete)**:
   Problems in NP and as hard as any problem in NP. No polynomial-time algorithm has yet been discovered for an NP-complete problem, nor has anyone yet been able to prove that no polynomial-time algorithm can exist for any one of them.

# Complexity Class P

- Deterministic in nature
- Solved by conventional computers in polynomial time
  - $O(1)$ Constant
  - $O(\log n)$ Sub-linear
  - $O(n)$ Linear
  - $O(n \log n)$ Nearly Linear
  - $O(n^2)$ Quadratic
  - $O(n^k)$ k-Polynomial
- Polynomial upper and lower bounds

**Shortest vs. longest simple paths:**
Even with negative edge weights, we can find shortest paths from a single source in a directed graph G = (V, E) in O(V E) time. Finding a longest simple path between two vertices is difficult, however. Merely determining whether a graph contains a simple path with at least a given number of edges is NP-complete.

**Euler tour vs. hamiltonian cycle: An Euler tour of a connected, directed graph**

G = (V, E) is a cycle that traverses each edge of G exactly once, although it may visit a vertex more than once. We can determine whether a graph has an Euler tour in only O(E) time and, in fact, we can find the edges of the Euler tour in O(E) time. A hamiltonian cycle of a directed graph G = (V, E) is a simple cycle that contains each vertex in V. Determining whether a directed graph has a hamiltonian cycle is NP-complete. determining whether an undirected graph has a hamiltonian cycle is NP-complete.

**2-CNF satisfiability vs. 3-CNF satisfiability: A boolean formula contains variables**

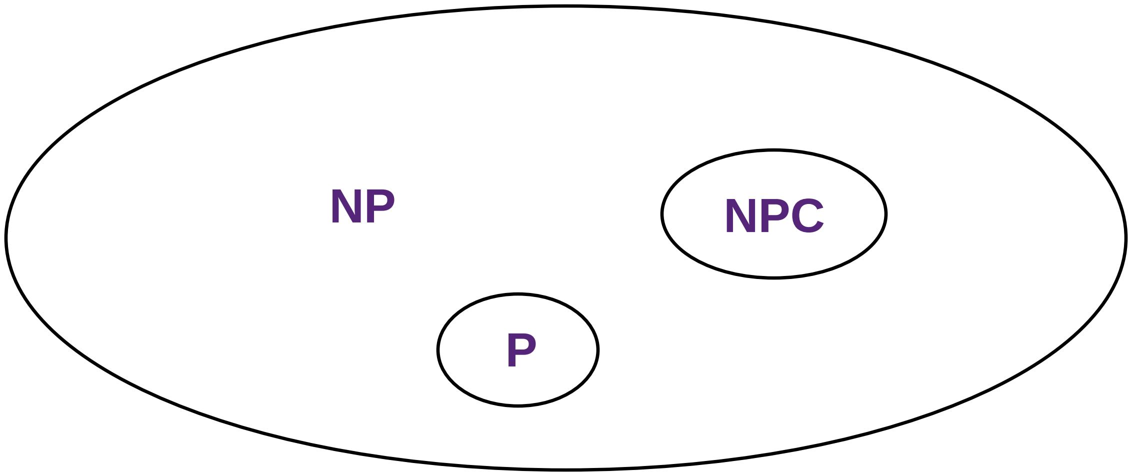
whose values are 0 or 1; boolean connectives such as ∧ (AND), ∨ (OR), and ¬ (NOT); and parentheses. A boolean formula is satisfiable if there is some assignment of the values 0 and 1 to its variables that causes it to evaluate to 1. A boolean formula is in k-conjunctive normal form, or k-CNF, if it is the AND of clauses of ORs of exactly k variables or their negations. For example, the boolean formula $(x1 \vee \neg x2) \wedge (\neg x1 \vee x3) \wedge (\neg x2 \vee \neg x3)$ is in 2-CNF. (It has the satisfying assignment x1 = 1, x2 = 0, x3 = 1.) There is a polynomial-time algorithm to determine whether a 2-CNF formula is satisfiable, a 3-CNF formula is satisfiable is NP-complete.

# Relation among P, NP, NPC

- $P \subseteq NP$ (Sure)
- $NPC \subseteq NP$ (sure)
- $P = NP$ (or $P \subset NP$, or $P \neq NP$) ???
- $NPC = NP$ (or $NPC \subset NP$, or $NPC \neq NP$) ???
- $P \neq NP$: one of the deepest, most perplexing open research problems in (theoretical) computer science since 1971.

> - *Any problem in P is also in NP, since if a problem is in P then we can solve it in polynomial time without even being given a certificate.*
>
> - *Most theoretical computer scientists believe that NPC is intractable (i.e., hard, and $P \neq NP$).*

# View of Theoretical Computer Scientists on P, NP, NPC




$$P \subset NP,\ NPC \subset NP,\ P \cap NPC = \emptyset$$

# Why discussion on NPC

- If a problem is proved to be NPC, a good evidence for its intractability (hardness).
- Not waste time on trying to find efficient algorithm for it
- Instead, focus on design approximate algorithm or a solution for a special case of the problem
- Some problems looks very easy on the surface, but in fact, is hard (NPC).

# Decision VS. Optimization Problems

- Decision problem: solving the problem by giving an answer “YES” or “NO”
- Optimization problem: solving the problem by finding the optimal solution.
- Examples:
  - **SHORTEST-PATH (optimization)**

    Given *G*, *u*,*v*, find a path from *u* to *v* with fewest edges.
  - **PATH (decision)**

    Given G, u,v, and k, whether exist a path from u to v consisting of at most k edges.

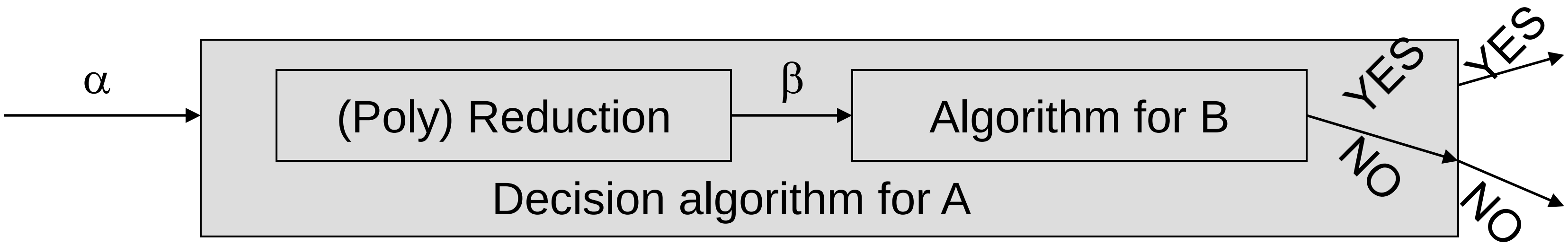
# Decision VS. Optimization Problems (Cont.)

- Decision is easier (i.e., no harder) than optimization
- If there is an algorithm for an optimization problem, the algorithm can be used to solve the corresponding decision problem.
- Example: SHORTEST-PATH for PATH
- If optimization is easy, its corresponding decision is also easy. Or in another way, if provide evidence that decision problem is hard, then the corresponding optimization problem is also hard.
- NPC is confined to decision problem. (also applicable to optimization problem.)
  - Another reason is that: easy to define reduction between decision problems.

# (Poly) reduction between decision problems

- Problem (class) and problem instance
- Instance $\alpha$ of decision problem A and instance $\beta$ of decision problem B
- A reduction from A to B is a transformation with the following properties:
  - The transformation takes poly time
  - The answer is the same (the answer for $\alpha$ is YES if and only if the answer for $\beta$ is YES).

# Implication of (poly) reduction




1. If decision algorithm for B is poly, so does A.
   A is no harder than B (or B is no easier than A)

2. If A is hard (e.g., NPC), so does B.

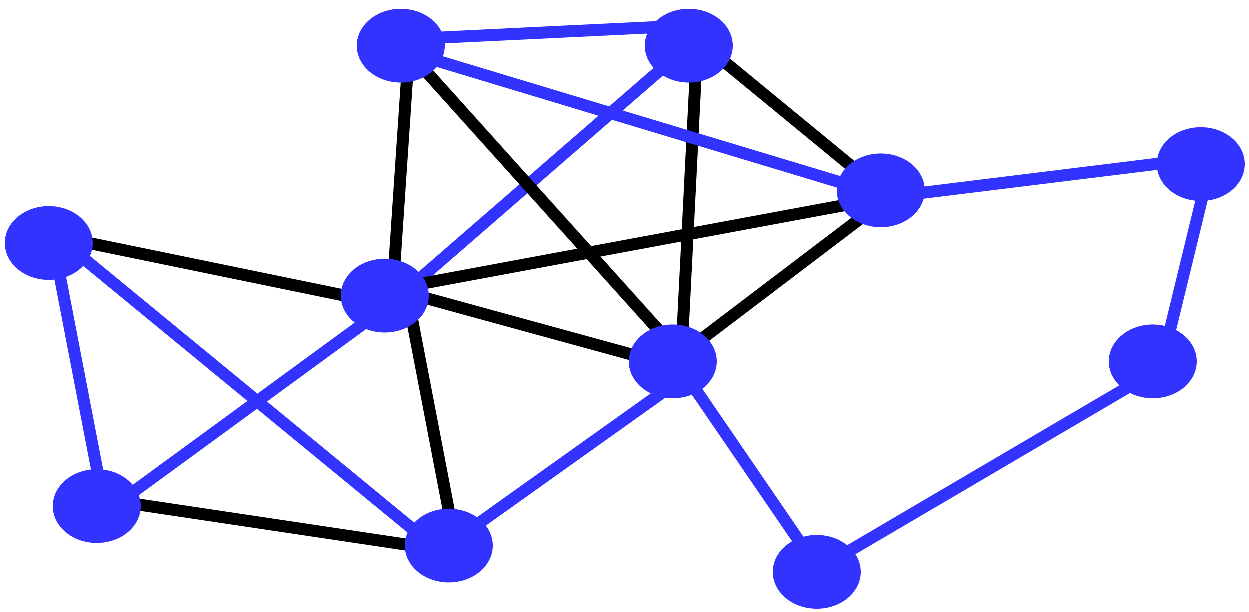
3. How to prove a problem B to be NPC ??

   (at first, prove B is in NP, which is generally easy.)
   3.1 find a already proved NPC problem A

Question: What is and how to prove the first NPC problem?

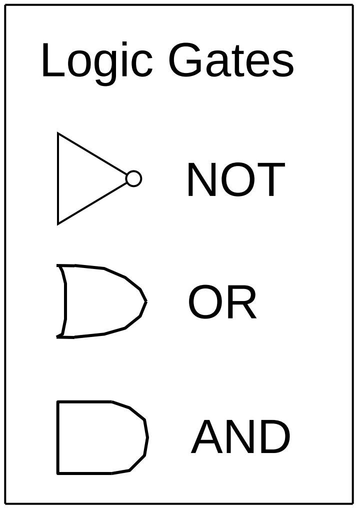
Circuit-satisfiability problem.

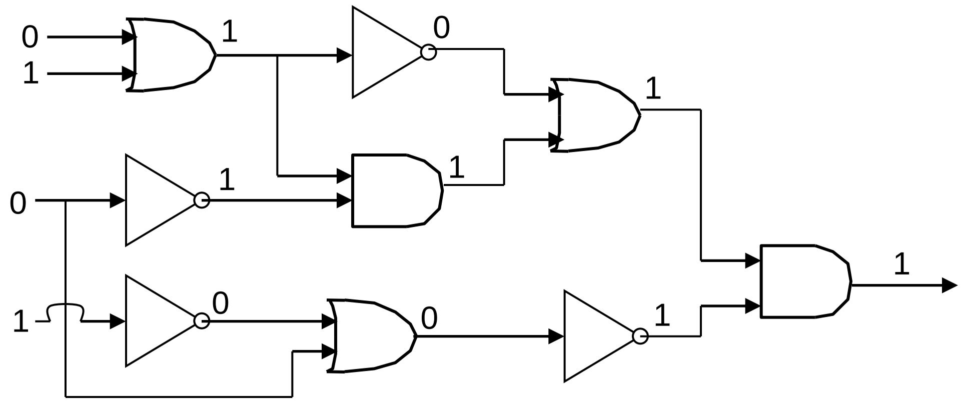
# Travelling Salesman Problem (TSP)



- For each two cities, an integer cost is given to travel from one of the two cities to the other. The salesperson wants to make a minimum cost circuit visiting each city exactly once.

# Circuit-SAT




- Take a Boolean circuit with a single output node and ask whether there is an assignment of values to the circuit's inputs so that the output is "1"

# Two instances of circuit satisfiability problems

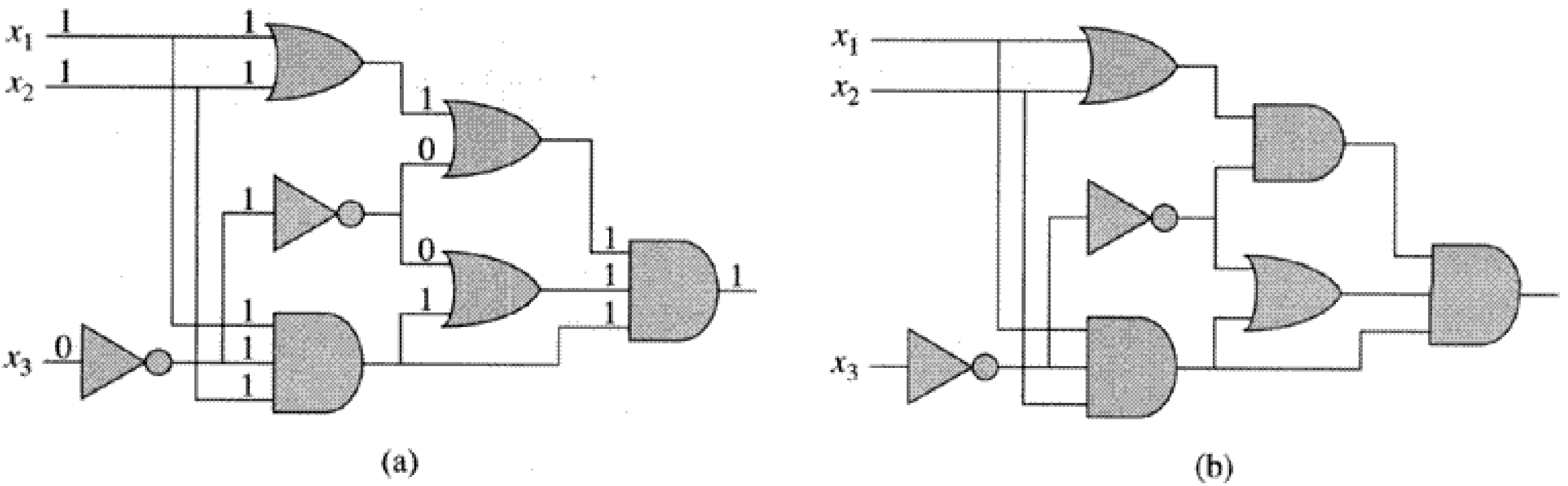
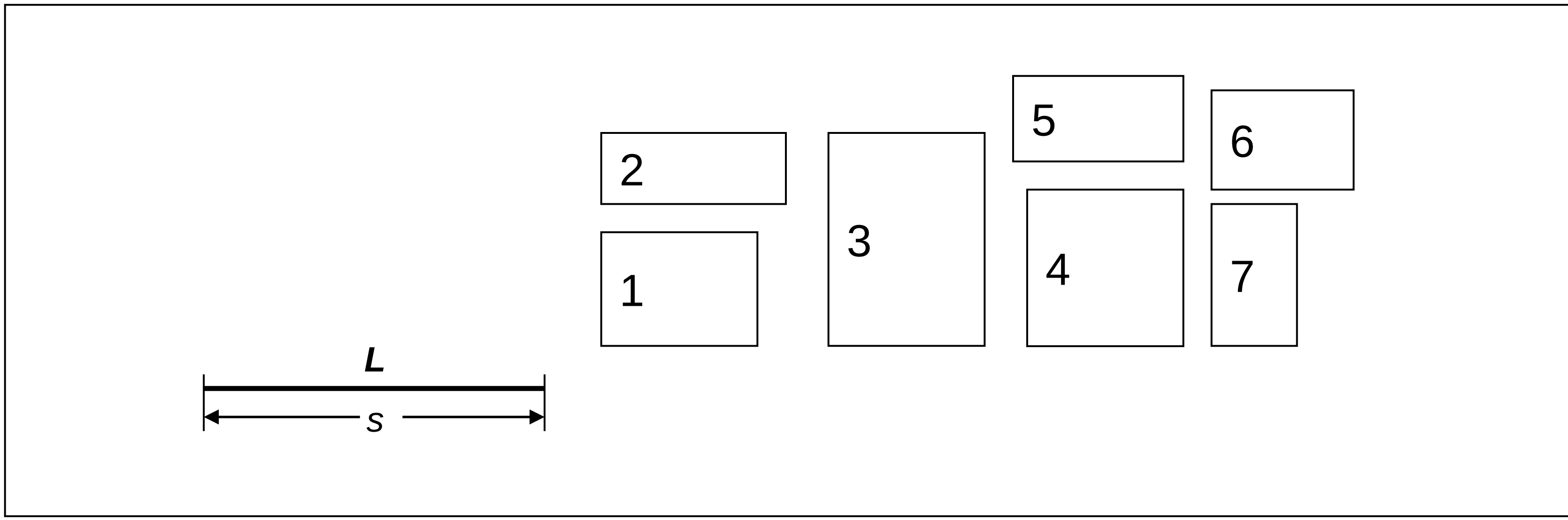


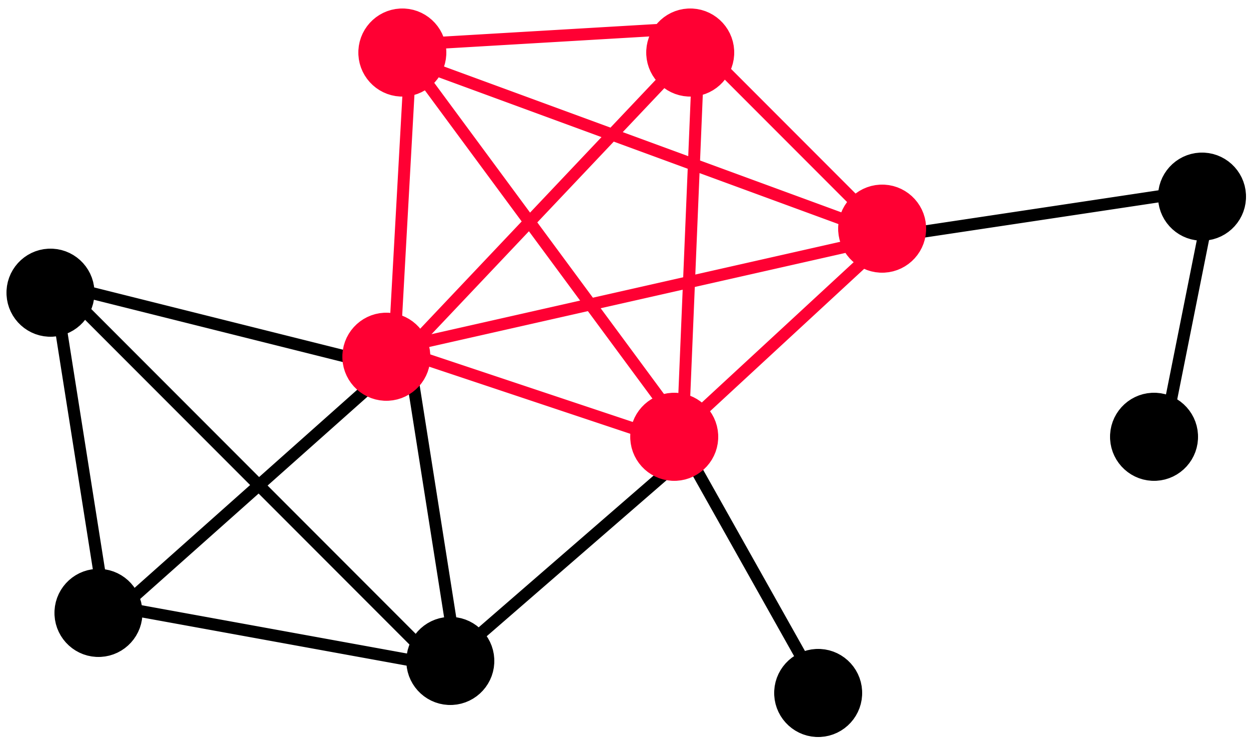

**Figure 34.8** Two instances of the circuit-satisfiability problem. **(a)** The assignment $\langle x_1 = 1, x_2 = 1, x_3 = 0\rangle$ to the inputs of this circuit causes the output of the circuit to be 1. The circuit is therefore satisfiable. **(b)** No assignment to the inputs of this circuit can cause the output of the circuit to be 1. The circuit is therefore unsatisfiable.

# Knapsack

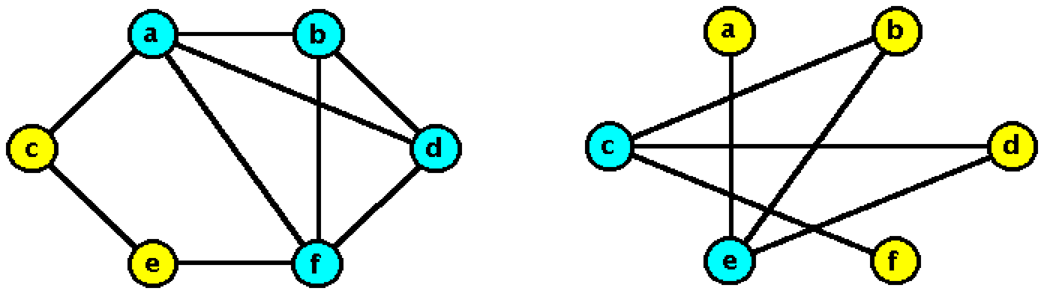



- Given s and w can we translate a subset of rectangles to have their bottom edges on L so that the total area of the rectangles touching L is at least w?

# 5-Clique



# Vertex Cover




**A graph and its complement.**

Note that there is a 4-clique (consisting of vertices a, b, d, and f) in the graph on the left. Note also that the vertices not in this clique (namely c and e) do form a cover for the complement of this graph (which appears on the right).

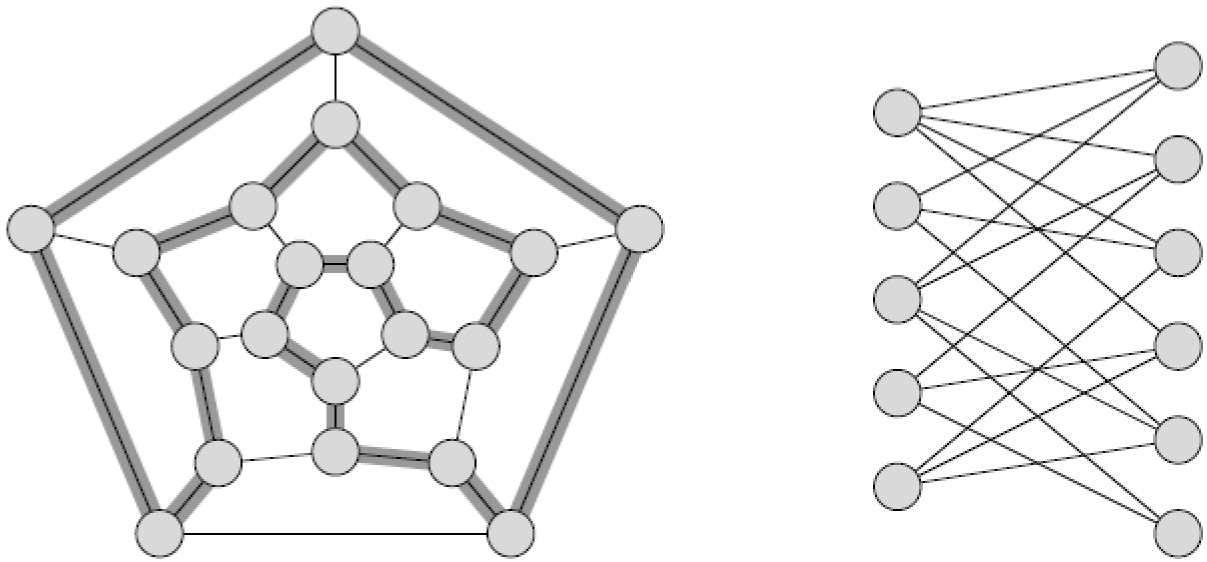
# The Halting Problem

- Given an algorithm A and an input I, will the algorithm reach a stopping place?

```
loop
  exit if (x = 1)
  if (even(x)) then
    x ← x div 2
  else
    x ← 3 * x + 1
endloop
```

- In general, we cannot solve this problem in finite time.

# Hamiltonian Path



(a) (b)

**Figure 34.2** (a) A graph representing the vertices, edges, and faces of a dodecahedron, with a hamiltonian cycle shown by shaded edges. (b) A bipartite graph with an odd number of vertices. Any such graph is nonhamiltonian.

**Euler tour vs. hamiltonian cycle: An *Euler tour of a connected, directed graph***
*G = (V, E) is a cycle that traverses each edge of G exactly once, although*
it may visit a vertex more than once. We can determine whether a graph has an Euler tour in only *O(E) time and, in fact, we can find*

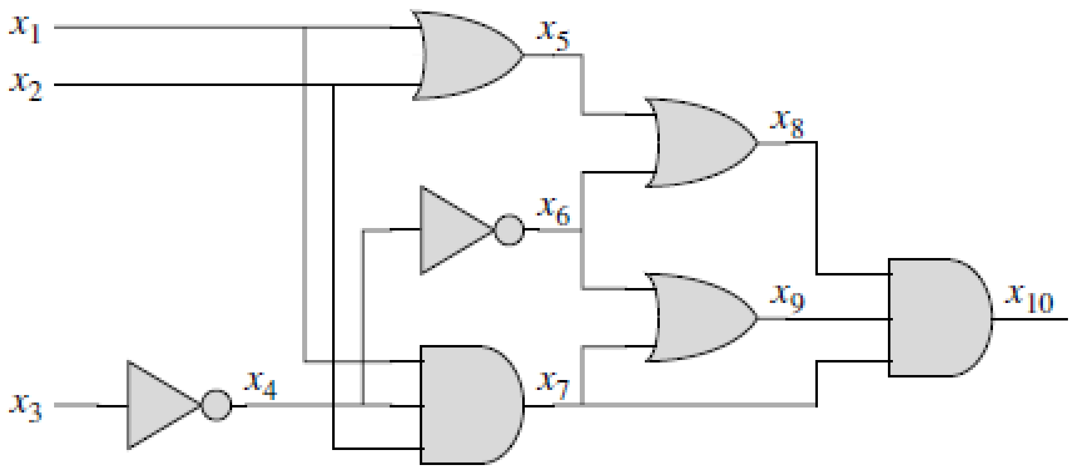



**Figure 34.10** Reducing circuit satisfiability to formula satisfiability. The formula produced by the reduction algorithm has a variable for each wire in the circuit.

# END