

Introduction to Electronic Design Automation

Jie-Hong Roland Jiang
江介宏

Department of Electrical Engineering
National Taiwan University



Spring 2014

1

Computation & Optimization in a Nutshell

□ Course contents:

- Computational complexity
- NP-completeness; PSPACE-completeness
- Algorithmic paradigms
- Mathematical optimization

□ Readings

■ Chapter 4

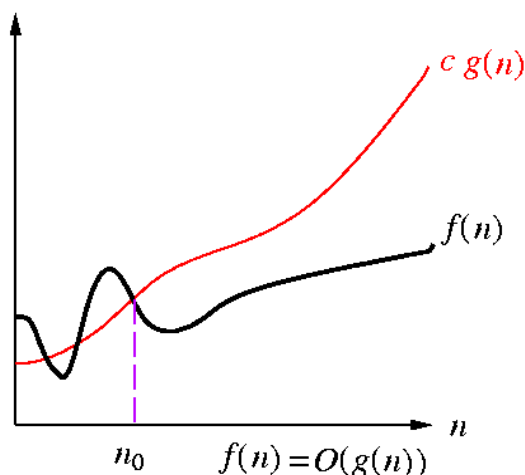
■ Reference:

- T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.
- M. Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 2nd edition, 2005.

2

O: Upper Bounding Function

- **Definition:** $f(n) = O(g(n))$ if $\exists c > 0$ and $n_0 > 0$ such that $0 \leq f(n) \leq c g(n)$ for all $n \geq n_0$
 - E.g., $2n^2 + 3n = O(n^2)$, $2n^2 = O(n^3)$, $3n \lg n = O(n^2)$
 - Intuition: $f(n) \leq g(n)$ when we ignore constant multiples and small values of n



3

Computation Complexity

- We would like to characterize the efficiency/hardness of problem solving
- By that, we can have a better idea on how to come up with good algorithms
 - Algorithm: a well-defined procedure transforming some *input* to a desired *output* in **finite** computational resources in *time* and *space* (c.f. semi-algorithm)
- Why does complexity matter?

Time	Big-Oh	$n = 10$	$n = 100$	$n = 10^3$	$n = 10^6$
500	$O(1)$	5×10^{-7} sec	5×10^{-7} sec	5×10^{-7} sec	5×10^{-7} sec
$3n$	$O(n)$	3×10^{-8} sec	3×10^{-7} sec	3×10^{-6} sec	0.003 sec
$n \log n$	$O(n \log n)$	3×10^{-8} sec	2×10^{-7} sec	3×10^{-6} sec	0.006 sec
n^2	$O(n^2)$	1×10^{-7} sec	1×10^{-5} sec	0.001 sec	16.7 min
n^3	$O(n^3)$	1×10^{-6} sec	0.001 sec	1 sec	31.7 yr
2^n	$O(2^n)$	1×10^{-6} sec	3×10^{14} cent.	∞	∞
$n!$	$O(n!)$	0.003 sec	∞	∞	∞

assuming 10^9 instructions per second

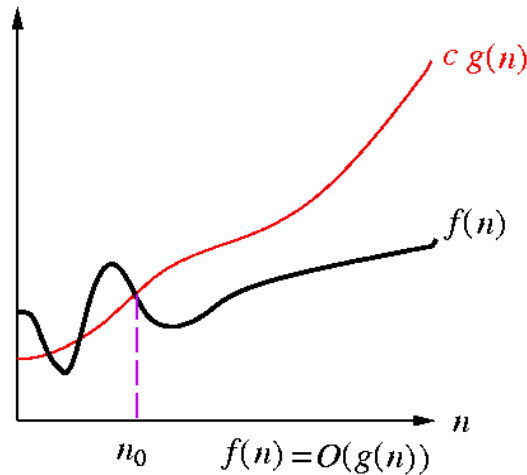
4

Big-O Notation

- How to show O (Big-Oh) relationships?

- $f(n) = O(g(n))$ iff $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c$ for some $c \geq 0$

- “An algorithm has worst-case running time $O(g(n))$ ”: there is a constant c s.t. for every n large enough, **every execution** on an **input of size n** takes **at most $c g(n)$** time



5

Big-O Notation (cont'd)

- Only the dominating term needs to be kept while constant coefficients are immaterial

- Example

$$0.3 n^2 = O(n^2)$$

$$3 n^2 + 152 n + 1777 = O(n^2)$$

$$n^2 \lg n + 3n^2 = O(n^2 \lg n)$$

The following are correct but not used

$$3n^2 = O(n^2 \lg n)$$

$$3n^2 = O(0.1 n^2)$$

$$3n^2 = O(n^2 + n)$$

6

Other Asymptotic Bounds

Other notations (though not important for now):

- **Definition:** $f(n) = \Omega(g(n))$ if $\exists c, n_0 > 0$ such that
 $0 \leq c g(n) \leq f(n)$ for all $n \geq n_0$.
 - Ω -notation provides an asymptotic *lower* bound on a function
- **Definition:** $f(n) = \Theta(g(n))$ if $\exists c_1, c_2, n_0 > 0$ such that
 $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$.
 - Θ -notation provides an asymptotic *tight* bound on a function
- Showing the complexity upper bound of solving a **problem (not an instance)** is often much easier than showing the complexity lower bound
 - Why?

7

Computational Complexity

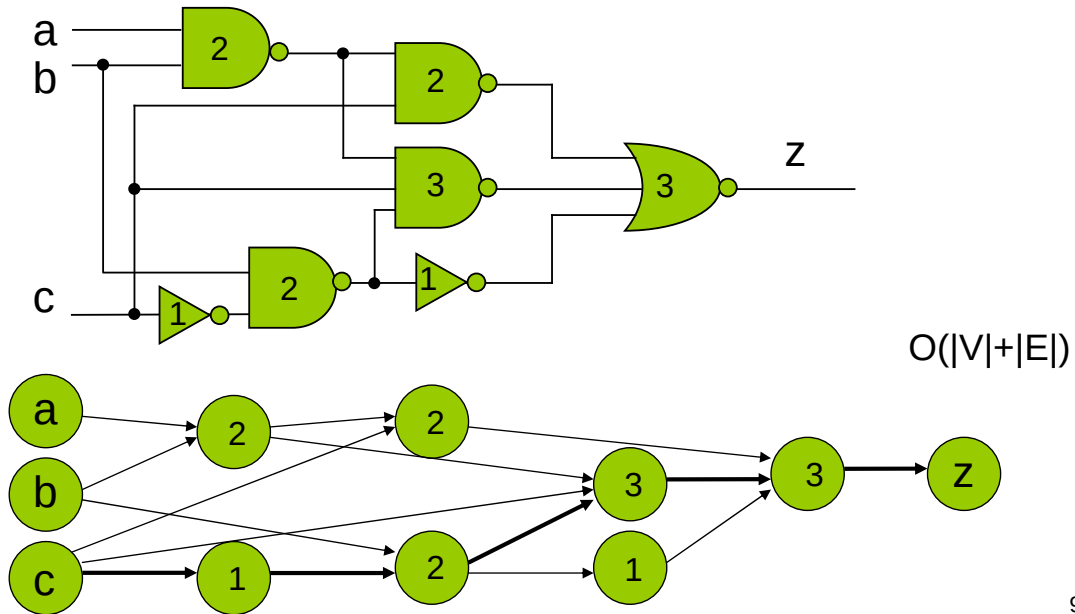
- **Computational complexity:** an abstract measure of the time and space necessary to execute an algorithm as function of its “**input size**”
- Input size examples:
 - sort n words of bounded length $\Rightarrow n$
 - the input is the integer $n \Rightarrow \lg n$
 - the input is the graph $G(V, E) \Rightarrow |V|$ and $|E|$
- **Time complexity** is expressed in *elementary computational steps* (e.g., an addition, multiplication, pointer indirection)
- **Space complexity** is expressed in *memory locations* (e.g. bits, bytes, words)

8

Computational Complexity

□ Example

- Computing **longest delay path** of a **directed acyclic graph**



9

Asymptotic Functions

- Polynomial-time complexity: $O(n^k)$, where n is the **input size** and k is a constant

- Example polynomial functions:

- 999: constant
- $\lg n$: logarithmic
- $\sqrt{n}$: sublinear
- n : linear
- $n \lg n$: loglinear
- n^2 : quadratic
- n^3 : cubic

- Example non-polynomial functions

- $2^n, 3^n$: exponential
- $n!$: factorial

Run-time Comparison

- Assume 1000 MIPS (Yr: 200x), 1 instruction /operation

Time	Big-Oh	$n = 10$	$n = 100$	$n = 10^3$	$n = 10^6$
500	$O(1)$	5×10^{-7} sec	5×10^{-7} sec	5×10^{-7} sec	5×10^{-7} sec
$3n$	$O(n)$	3×10^{-8} sec	3×10^{-7} sec	3×10^{-6} sec	0.003 sec
$n \log n$	$O(n \log n)$	3×10^{-8} sec	2×10^{-7} sec	3×10^{-6} sec	0.006 sec
n^2	$O(n^2)$	1×10^{-7} sec	1×10^{-5} sec	0.001 sec	16.7 min
n^3	$O(n^3)$	1×10^{-6} sec	0.001 sec	1 sec	31.7 yr
2^n	$O(2^n)$	1×10^{-6} sec	3×10^{14} cent.	∞	∞
$n!$	$O(n!)$	0.003 sec	∞	∞	∞

11

Computation Problems

- Two common types of problems in computer science:
 - Optimization problems
 - Often discrete/combinatorial rather than continuous
 - E.g., Minimum Spanning Tree (MST), Travelling Salesman Problem (TSP), etc.
 - Decision problems
 - E.g., Fixed-weight Spanning Tree, Satisfiability (SAT), etc.

12

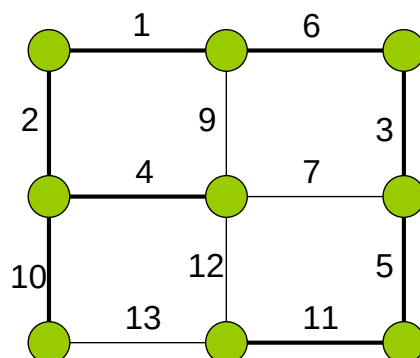
Terminology

- **Problem:** a general class, e.g., “the shortest-path problem for directed acyclic graphs”
- **Instance:** a specific case of a problem, e.g., “the shortest-path problem in a specific graph, between two given vertices”
- **Optimization problems:** those finding a legal configuration such that its cost is minimum (or maximum)
 - MST: Given a graph $G=(V, E)$, find the cost of a minimum spanning tree of G
- An instance $I = (F, c)$ where
 - F is the set of *feasible solutions*, and
 - c is a *cost function*, assigning a cost value to each feasible solution $c : F \rightarrow R$
 - The solution of the optimization problem is the feasible solution with optimal (minimal/maximal) cost
- c.f., **optimal** solutions/costs, optimal (**exact**) algorithms (Attn: optimal $\neq$ exact in the theoretic computer science community)

13

Optimization Problem: Minimum Spanning Tree (MST)

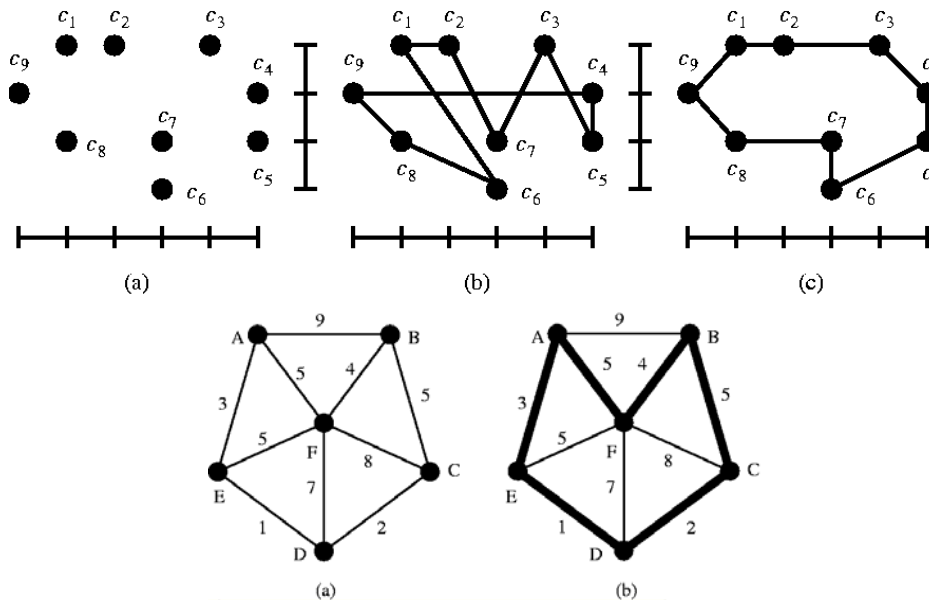
- MST: Given an undirected graph $G = (V, E)$ with weights on the edges, a **minimum spanning tree** of G is a subgraph $T \subseteq G$ such that
 - T has no cycles (i.e. a tree)
 - T contains all vertices in V
 - Sum of the weights of all edges in T is minimum



14

Optimization Problem: Traveling Salesman Problem (TSP)

- TSP: Given a set of cities and the distance between each pair of cities, find the distance of a **minimum tour** starts and ends at a given city and visits every city exactly once



15

Terminology

- **Decision problems:** problem that can only be answered with “yes” or “no”
 - MST: Given a graph $G=(V, E)$ and a bound K , is there a spanning tree with **a cost at most K ?**
 - TSP: Given a set of cities, distance between each pair of cities, and a bound B , is there a route that starts and ends at a given city, visits every city exactly once, and has **total distance at most B ?**
- A decision problem Π , has instances: $I = (F, c, k)$
 - The set of instances for which the answer is “yes” is given by Y_{Π}
 - A subtask of a decision problem is *solution checking*: given $f \in F$, checking whether the cost $c(f)$ is less than k
- Can apply binary search on decision problems to obtain solutions to optimization problems
- NP-completeness is associated with decision problems

16

Decision Problem: Fixed-weight Spanning Tree

- Given an undirected graph $G = (V, E)$, is there a spanning tree of G with weight c ?
- Can solve MST by posing it as a sequence of decision problems (with binary search)

17

Decision Problem: Satisfiability Problem (SAT)

- **Satisfiability Problem (SAT):**
 - **Instance:** A Boolean formula φ in conjunctive normal form (CNF), a.k.a. product-of-sums (POS)
 - **Question:** Is there an assignment of Boolean values to the variables that makes φ true ?
- A Boolean formula φ is *satisfiable* if there exists a set of Boolean input values that makes φ evaluate to true. Otherwise, φ is *unsatisfiable*.
 - $(a+b)(\neg a+c)(\neg b+\neg c)$ is satisfiable since $(a, b, c) = (0, 1, 0)$ makes the formula true.
 - $(a+b)(\neg a+c)(\neg b)(\neg c)$ is unsatisfiable

18

Decision Problem: Circuit Satisfiability Problem (CSAT)

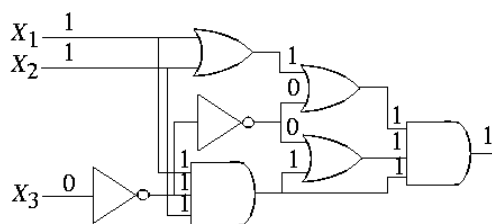
■ Circuit-Satisfiability Problem (CSAT):

■ **Instance:** A combinational circuit C composed of AND, OR, and NOT gates

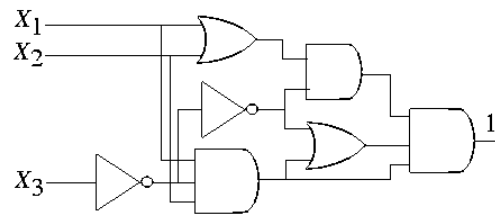
■ **Question:** Is there an assignment of Boolean values to the inputs that makes the output of C to be 1?

■ A circuit is satisfiable if there exists a set of Boolean input values that makes the output of the circuit to be 1

■ Circuit (a) is satisfiable since $(x_1, x_2, x_3) = (1, 1, 0)$ makes the output to be 1



(a) satisfiable circuit



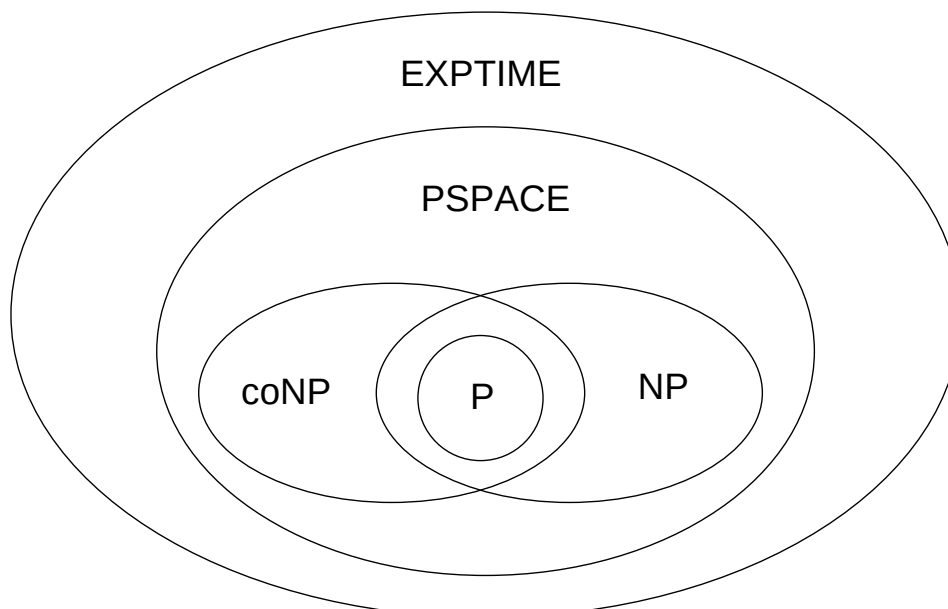
(b) unsatisfiable circuit

19

Complexity Hierarchy

■ Tractable: solvable in deterministic polynomial time (P)

■ Intractable: unsolvable in deterministic polynomial time (P)



20

Complexity Class P

- **Complexity class P** contains those problems that can be **solved** in polynomial time in the **size of input**
 - **Input size:** size of encoded “binary” strings
 - Edmonds: Problems in P are considered **tractable**
- The computer concerned is a **deterministic Turing machine**
 - **Deterministic** means that each step in computation is predictable
 - A **Turing machine** is a mathematical model of a universal computer (any computation that needs polynomial time on a Turing machine can also be performed in polynomial time on any other machine)
- MST and shortest path problems are in P

21

Complexity Class NP

- Suppose that **solution checking** for some problem can be done in polynomial time on a deterministic machine $\Rightarrow$ the problem can be solved in polynomial time on a **nondeterministic Turing machine**
 - **Nondeterministic:** the machine makes a guess, e.g., the right one (or the machine evaluates all possibilities in parallel)
- **The class NP (Nondeterministic Polynomial):** class of problems that can be **verified** in polynomial time in the size of input
 - NP: class of problems that can be solved in polynomial time on a nondeterministic machine
- Is TSP $\in$ NP?
 - Need to **check** a solution in polynomial time
 - Guess a tour
 - Check if the tour visits every city exactly once
 - Check if the tour returns to the start
 - Check if total distance $\leq B$
 - All can be done in $O(n)$ time, so TSP $\in$ NP

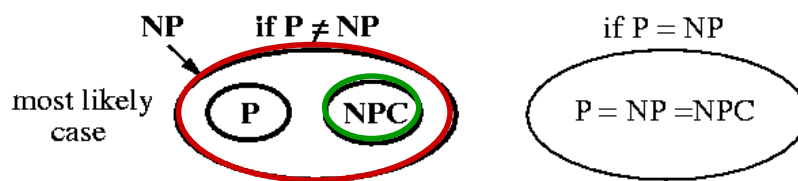
22

P vs. NP

□ An issue which is still unsettled:

$P \subset NP$ or $P = NP$?

- There is a strong belief that $P \neq NP$, due to the existence of NP-complete problems.
- One of the 7 *Clay Millennium Prize Problems*



23

NP-Completeness

□ The **NP-complete (NPC)** class:

- Developed by S. Cook and R. Karp in early 1970
 - Cook showed the first NP-complete problem (SAT) in 1971
 - Karp showed many other problems are NP-complete (by polynomial reduction) in 1972
- Thousands of combinatorial problems are known to be NP-complete
 - NP-complete problems: SAT, 3SAT, CSAT, TSP, Bin Packing, Hamiltonian Cycles, ...
- All problems in NPC have the same degree of difficulty:
 - Any** NPC problem can be solved in polynomial time $\Rightarrow$
 - All** problems in NP can be solved in polynomial time

24

Beyond NP

□ A quantified Boolean formula (QBF) is

$Q_1 x_1, Q_2 x_2, \dots, Q_n x_n. \varphi$

where Q_i is either an existential ($\exists$) or universal quantifier ($\forall$), x_i is a Boolean variable, and φ is a Boolean formula

■ $\Sigma_i: \exists X_1, \forall X_2, \exists X_3, \dots, Q_n X_i. \varphi$

■ $\Pi_i: \forall X_1, \exists X_2, \forall X_3, \dots, Q_n X_i. \varphi$

□ X_i is a variable set ($X_i \cap X_j = \emptyset$ for $i \neq j$)

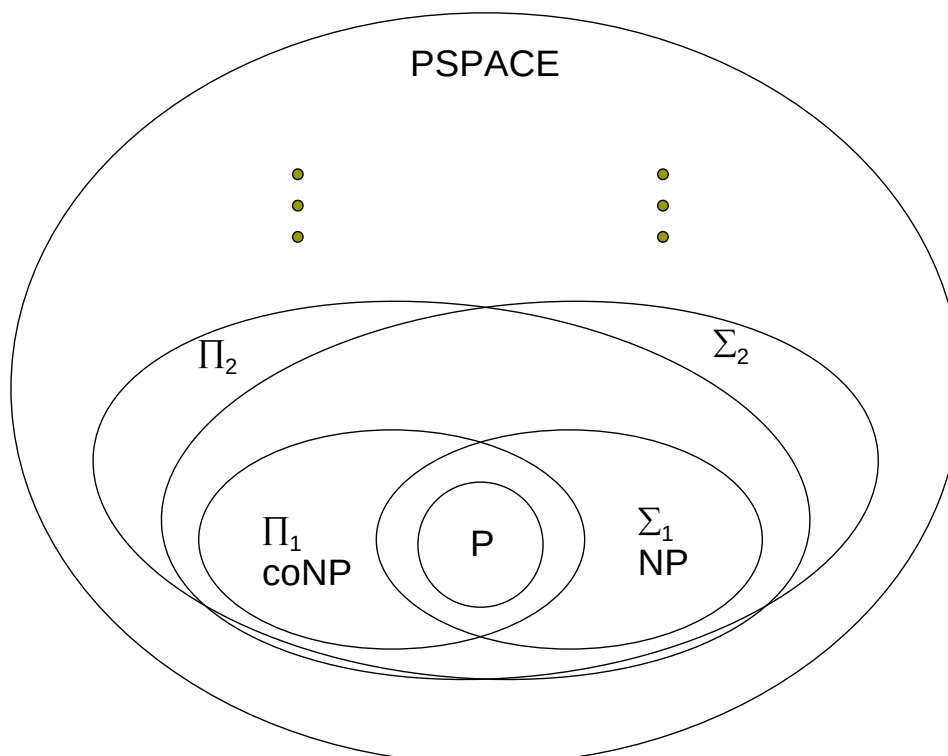
□ The polynomial-time hierarchy

■ $\Sigma_1 (= \text{NP}) \subseteq \Sigma_2 \subseteq \dots \subseteq \Sigma_i \subseteq \dots$

■ $\Pi_1 (= \text{coNP}) \subseteq \Pi_2 \subseteq \dots \subseteq \Pi_i \subseteq \dots$

25

Polynomial Hierarchy



26

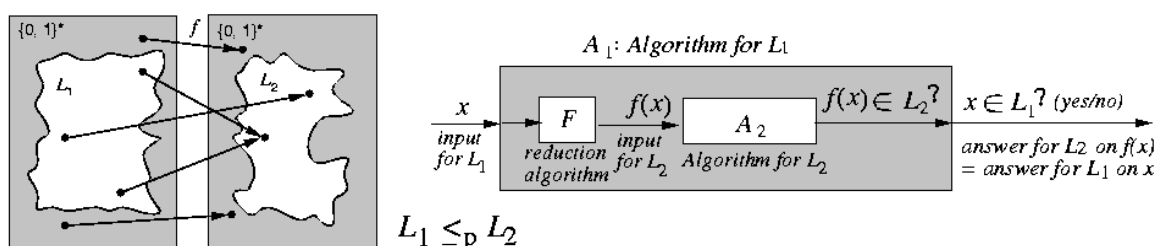
PSPACE-Completeness

- The satisfiability problem for quantified Boolean formulae (QSAT) is PSPACE-complete
 - GO is PSPACE-complete!
 - Many sequential verification problems are PSPACE-complete

27

Polynomial-time Reduction

- **Motivation:** Let L_1 and L_2 be two decision problems. Suppose algorithm A_2 can solve L_2 . Can we use A_2 to solve L_1 ?
- **Polynomial-time reduction f from L_1 to L_2 :** $L_1 \leq_p L_2$
 - f reduces input for L_1 into an input for L_2 s.t. the reduced input is a “yes” input for L_2 iff the original input is a “yes” input for L_1
 - $L_1 \leq_p L_2$ if $\exists$ polynomial-time computable function $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$ s.t. $x \in L_1$ iff $f(x) \in L_2, \forall x \in \{0, 1\}^*$
 - **L_2 is at least as hard as L_1**
- f is computable in polynomial time



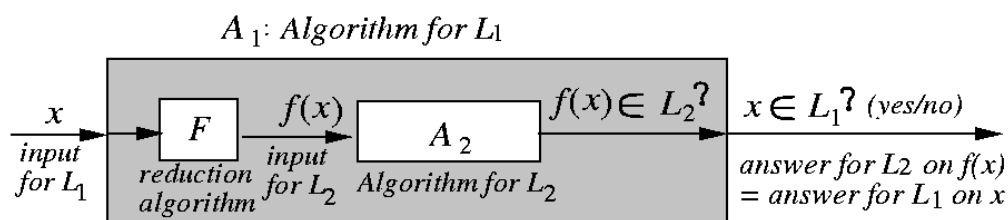
28

Significance of Reduction

□ Significance of $L_1 \leq_P L_2$:

- $\exists$ polynomial-time algorithm for $L_2 \Rightarrow \exists$ polynomial-time algorithm for L_1 ($L_2 \in P \Rightarrow L_1 \in P$)
- $\nexists$ polynomial-time algorithm for $L_1 \Rightarrow \nexists$ polynomial-time algorithm for L_2 ($L_1 \notin P \Rightarrow L_2 \notin P$)

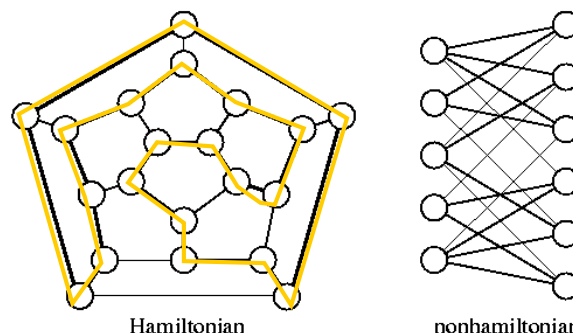
□ $\leq_P$ is transitive, i.e., $L_1 \leq_P L_2$ and $L_2 \leq_P L_3 \Rightarrow L_1 \leq_P L_3$



29

Polynomial-time Reduction

- The Hamiltonian Circuit, a.k.a. Hamiltonian Cycle, Problem (HC)
 - **Instance:** an undirected graph $G = (V, E)$
 - **Question:** is there a cycle in G that includes every vertex exactly once?
- TSP (The Traveling Salesman Problem)
- How to show $HC \leq_P TSP$?
 1. Define a function f mapping **any** HC instance into a TSP instance, and show that f can be computed in polynomial time
 2. Prove that G has an HC iff the reduced instance has a TSP tour **with distance $\leq B$** ($x \in HC \Leftrightarrow f(x) \in TSP$)



30

HC $\leq_p$ TSP: Step 1

□ Define a reduction function f for HC $\leq_p$ TSP

■ Given an arbitrary HC instance $G = (V, E)$ with n vertices

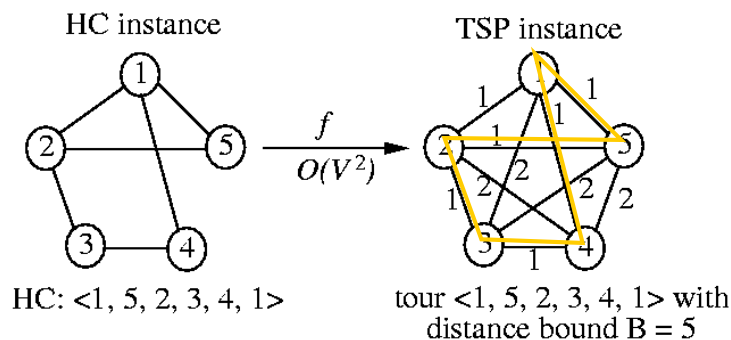
□ Create a set of n cities labeled with names in V

□ Assign distance between u and v

$$d(u, v) = \begin{cases} 1, & \text{if } (u, v) \in E, \\ 2, & \text{if } (u, v) \notin E. \end{cases}$$

□ Set bound $B = n$

■ f can be computed in $O(V^2)$ time



31

HC $\leq_p$ TSP: Step 2

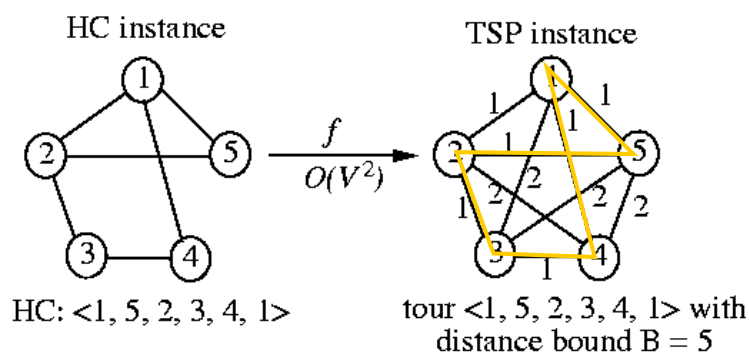
□ G has an HC iff the reduced instance has a TSP with distance $\leq B$

■ $x \in \text{HC} \Rightarrow f(x) \in \text{TSP}$

□ Suppose the HC is $h = \langle v_1, v_2, \dots, v_n, v_1 \rangle$. Then, h is also a tour in the transformed TSP instance

□ The distance of the tour h is $n = B$ since there are n consecutive edges in E , and so has distance 1 in $f(x)$

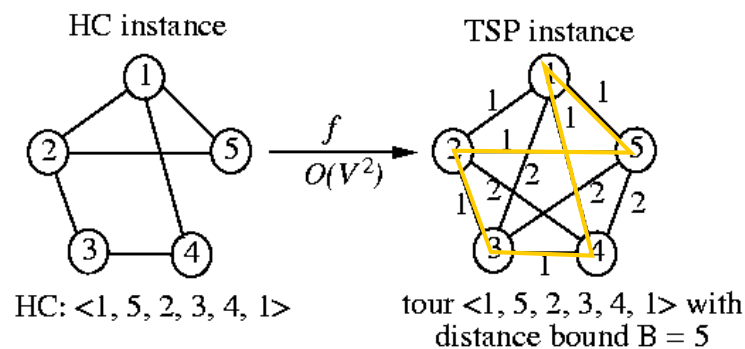
□ Thus, $f(x) \in \text{TSP}$ ($f(x)$ has a TSP tour with distance $\leq B$)



32

HC $\leq_p$ TSP: Step 2 (cont'd)

- G has an HC iff the reduced instance has a TSP with distance $\leq B$
 - $f(x) \in \text{TSP} \Rightarrow x \in \text{HC}$
 - Suppose there is a TSP tour with distance $\leq n = B$. Let it be $\langle v_1, v_2, \dots, v_n, v_1 \rangle$.
 - Since distance of the tour $\leq n$ and there are n edges in the TSP tour, the tour contains only edges in E
 - Thus, $\langle v_1, v_2, \dots, v_n, v_1 \rangle$ is a Hamiltonian cycle ($x \in \text{HC}$)



33

NP-Completeness and NP-Hardness

- **NP-completeness:** **worst-case** analyses for **decision** problems
- L is **NP-complete** if
 - **L** $\in$ **NP**
 - **NP-Hard:** $L' \leq_p L$ for every $L' \in \text{NP}$
- **NP-hard:** If L satisfies the 2nd property, but not necessarily the 1st property, we say that L is **NP-hard**
- Significance of NPC class:
 - Suppose $L \in \text{NPC}$
 - If $L \in P$, then there exists a polynomial-time algorithm for every $L' \in \text{NP}$ (i.e., $P = \text{NP}$)
 - If $L \notin P$, then there exists no polynomial-time algorithm for any $L' \in \text{NPC}$ (i.e., $P \neq \text{NP}$)

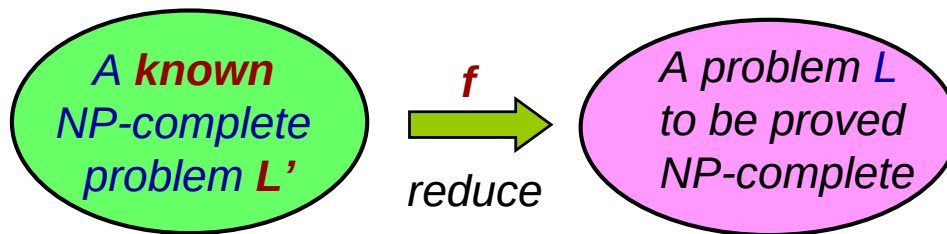
34

Proving NP-Completeness

□ Five steps for proving that L is NP-complete:

1. Prove $L \in \text{NP}$
2. Select a known NP-complete problem L'
3. Construct a reduction f transforming **every** instance of L' to an instance of L
4. Prove that $x \in L'$ iff $f(x) \in L$ for all $x \in \{0, 1\}^*$
5. Prove that f is a polynomial-time transformation

■ E.g., we showed that TSP is NP-complete



35

Easy vs. Hard Problems

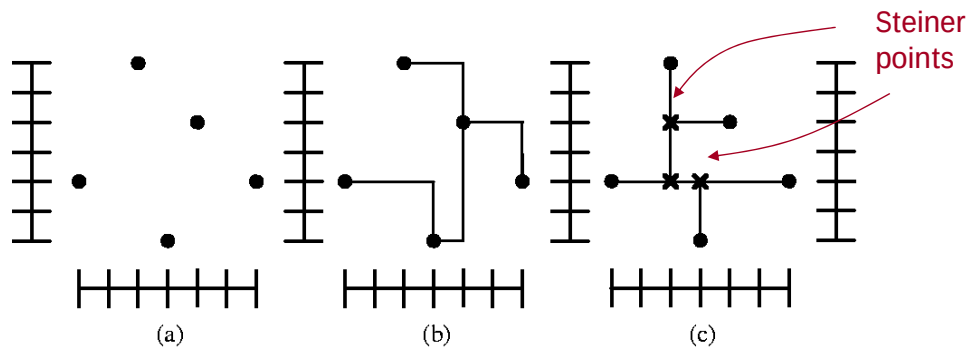
□ Many seemingly similar problems may have substantial difference in their inherent hardness

- Shortest path $\in \text{P}$; longest path $\in \text{NPC}$
- Spanning tree $\in \text{P}$; Steiner tree $\in \text{NPC}$
- Linear programming (LP) $\in \text{P}$; integer linear programming (ILP) $\in \text{NPC}$
- ...

36

Spanning Tree vs. Steiner Tree

- **Manhattan distance:** If two points (nodes) are located at coordinates (x_1, y_1) and (x_2, y_2) , the Manhattan distance between them is given by $d_{12} = |x_1 - x_2| + |y_1 - y_2|$
- **Rectilinear spanning tree:** a spanning tree that connects its nodes using Manhattan paths (Fig. (b) below)
- **Steiner tree:** a tree that connects its nodes, and additional points (**Steiner points**) are permitted to be used for the connections
- The minimum rectilinear spanning tree problem is in P, while the minimum rectilinear Steiner tree (Fig. (c)) problem is NP-complete
 - The spanning tree algorithm can be an *approximation* for the Steiner tree problem (at most 50% away from the optimum)



37

Hardness of Problem Solving

- Most optimization problems are **intractable**
 - Cannot afford to search the exact optimal solution
 - Global optimal (optimum) vs. local optimal (optimal)
- Search a reasonable solution within a reasonable bound on computational resources

38

Coping with NP-hard Problems

- **Approximation algorithms**
 - Guarantee to be a fixed percentage away from the optimum
 - E.g., MST for the minimum Steiner tree problem
- **Randomized algorithms**
 - Trade determinism for efficiency
- **Pseudo-polynomial time algorithms**
 - Has the form of a polynomial function for the complexity, but is not to the problem size
 - E.g., $O(nW)$ for the 0-1 knapsack problem
- **Restriction**
 - Work on some subset of the original problem
 - E.g., longest path problem restricted to directed acyclic graphs
- **Exhaustive search / Branch and bound**
 - Is feasible only when the problem size is small
- **Local search:**
 - Simulated annealing (hill climbing), genetic algorithms, etc.
- **Heuristics:** No guarantee of performance

39

Algorithmic Paradigms

- **Exhaustive search:** Search the entire solution space
- **Branch and bound:** A search technique with pruning
- **Greedy method:** Pick a locally optimal solution at each step
- **Dynamic programming:** Partition a problem into a collection of sub-problems, the sub-problems are solved, and then the original problem is solved by combining the solutions (applicable when the sub-problems are **NOT independent**)
- **Hierarchical approach:** Divide-and-conquer
- **Mathematical programming:** A system of solving an objective function under constraints
- **Simulated annealing:** An adaptive, iterative, non-deterministic algorithm that allows “uphill” moves to escape from local optima
- **Tabu search:** Similar to simulated annealing, but does not decrease the chance of “uphill” moves throughout the search
- **Genetic algorithm:** A population of solutions is stored and allowed to evolve through successive generations via mutation, crossover, etc.

40

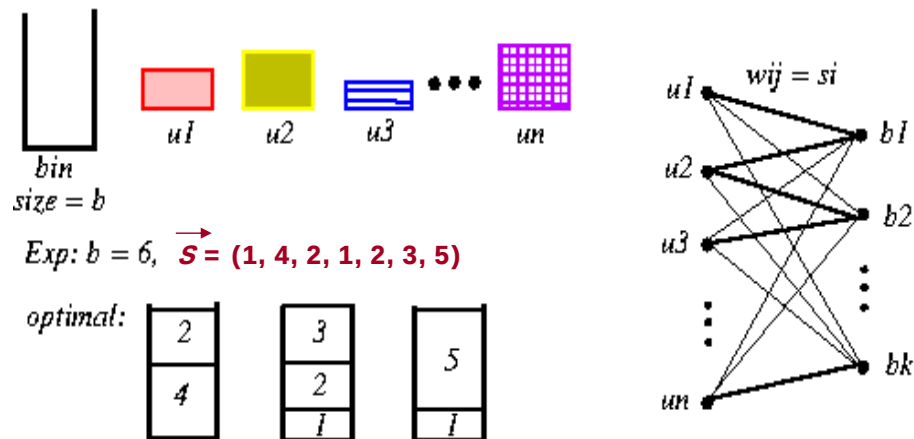
Example: Bin Packing

□ The Bin-Packing Problem Π :

Items $U = \{u_1, u_2, \dots, u_n\}$, where u_i is of an integer size s_i ;
set B of bins, each with capacity b

□ Goal:

Pack all items, minimizing # of bins used (**NP-hard!**)



43

Algorithms for Bin Packing

□ Greedy approximation algorithm:

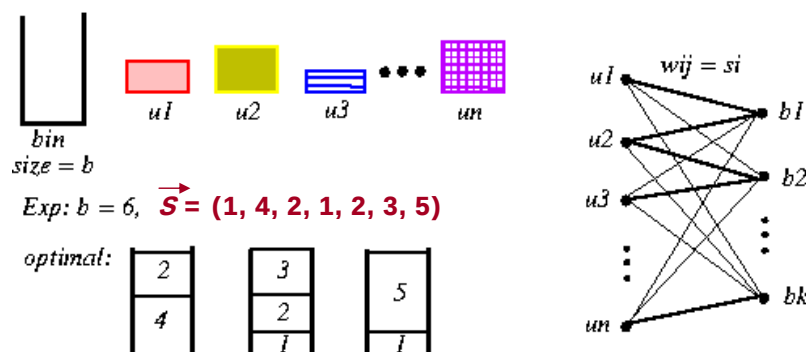
First-Fit Decreasing (FFD)

$$\text{FFD}(\Pi) \leq 11\text{OPT}(\Pi)/9 + 4$$

□ Dynamic Programming? Hierarchical Approach? Genetic Algorithm? ...

□ Mathematical Programming:

Use **integer linear programming (ILP)** to find a solution using $|B|$ bins, then search for the smallest feasible $|B|$



44

ILP Formulation for Bin Packing

- 0-1 variable: $x_{ij}=1$ if item u_i is placed in bin b_j , 0 otherwise

$$\begin{array}{ll}\max & \sum_{(i,j) \in E} w_{ij} x_{ij} \\ \text{subject to} & \sum_{i \in U} w_{ij} x_{ij} \leq b_j, \forall j \in B \quad /* \text{capacity constraint} */ \quad (1) \\ & \sum_{j \in B} x_{ij} = 1, \forall i \in U \quad /* \text{assignment constraint} */ \quad (2) \\ & \sum_{ij} x_{ij} = n \quad /* \text{completeness constraint} */ \quad (3) \\ & x_{ij} \in \{0, 1\} \quad /* 0, 1 \text{ constraint} */ \quad (4)\end{array}$$

- Step 1:** Set $|B|$ to the lower bound of the # of bins
- Step 2:** Use the ILP to find a **feasible solution**
- Step 3:** If the solution exists, the # of bins required is $|B|$. Then exit.
- Step 4:** Otherwise, set $|B| \leftarrow |B| + 1$. Goto Step 2.

45

Mathematical Programming

- Many optimization problems can be formulated as

$$\begin{array}{ll}\text{minimize (or maximize)} & f_0(\mathbf{x}) \quad \text{objective function} \\ \text{subject to} & f_i(\mathbf{x}) \leq c_i, i = 1, \dots, m. \quad \text{constraints}\end{array}$$

- Some special common mathematical programming
 - Linear programming (LP)
 - Integer linear programming (ILP)
 - Nonlinear programming
 - Convex optimization
 - Semi-definite programming, geometric programming, ...

46