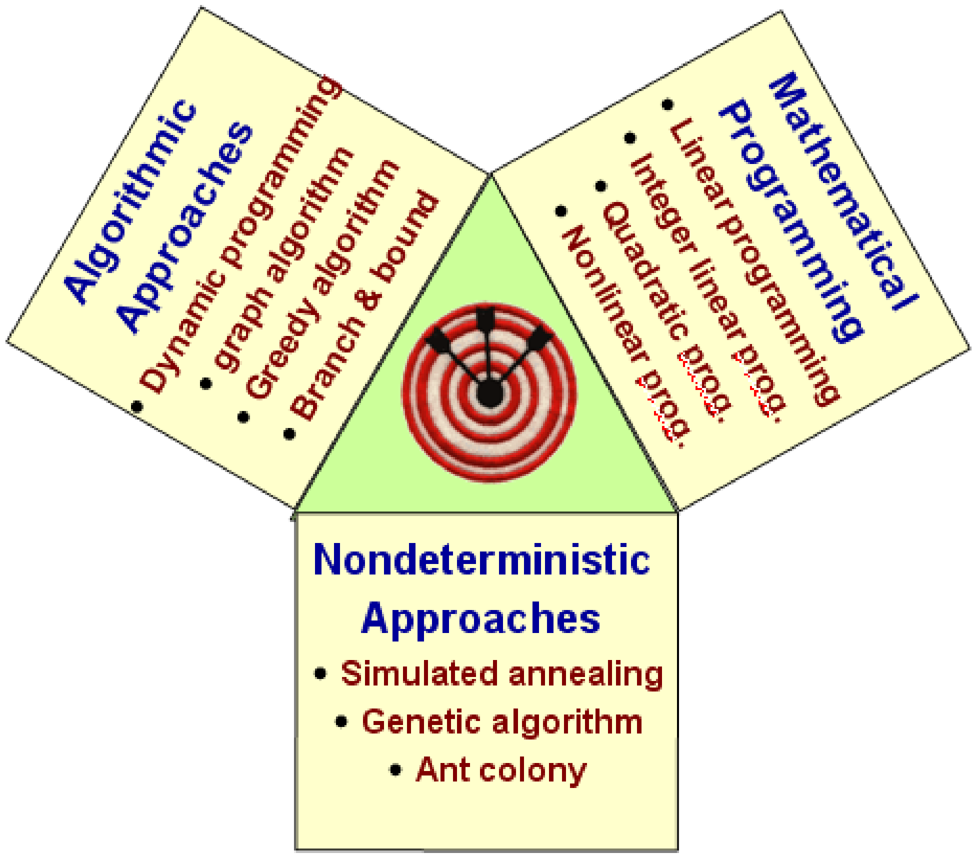
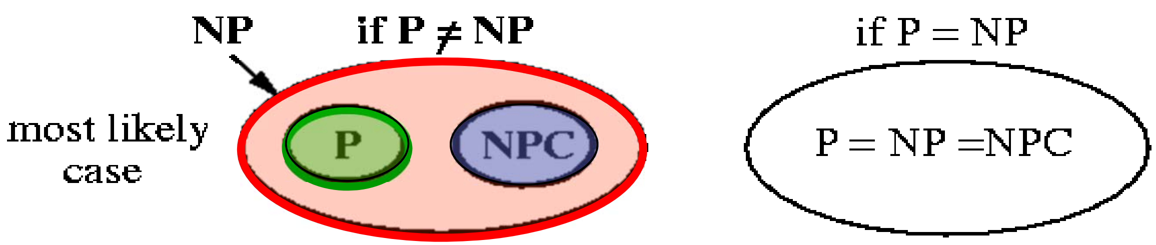
Unit 2: EDA Paradigms & Complexity

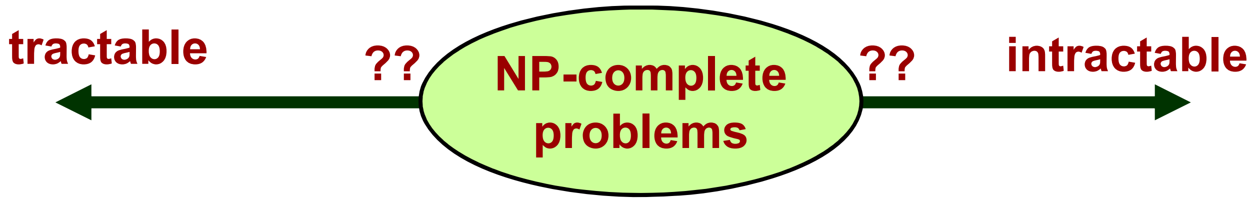
- Course contents:
  - EDA paradigms: Algorithms, Frameworks, Methodology
- Appendix
  - Computational Complexity & NP-completeness (self reading)
- Readings
  - W&C&C: Chapter 4
  - S&Y: Appendix A



Complexity Classes

- **Class P:** class of problems that can be **solved** in polynomial time in the **size of input**.
  - **Size of input:** size of encoded "binary" strings.
  - Edmonds: Problems in P are considered **tractable**.
- **Class NP** (**N**ondeterministic **P**olynomial): class of problems that can be **verified** in polynomial time in the size of input.
  - P = NP?
- **Class NP-complete (NPC): Any** NPC problem can be solved in polynomial time **=> All** problems in NP can be solved in polynomial time (i.e., P = NP).



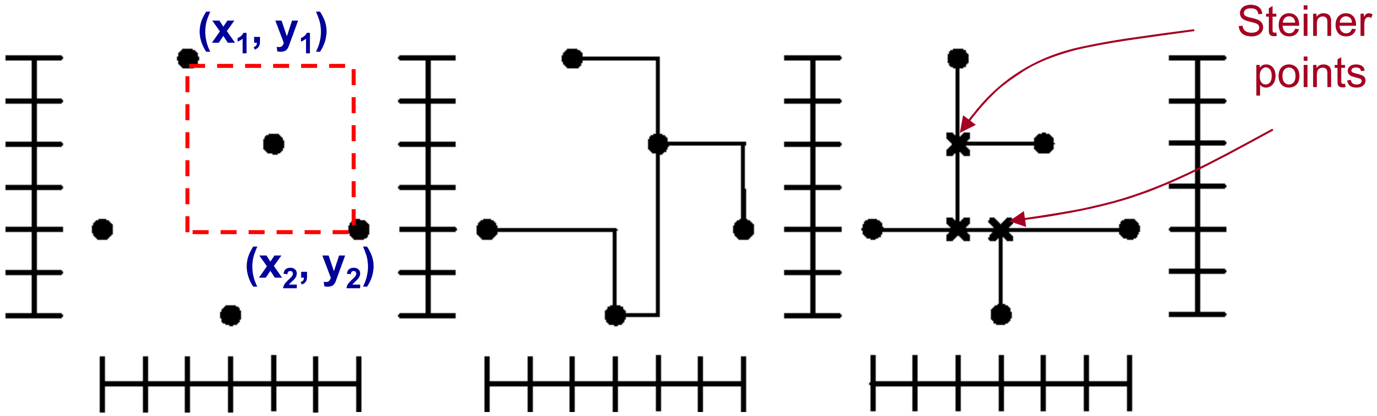


Coping with NP-Complete Problems

- **Approximation algorithms**
  - Guarantee to be a fixed percentage away from the optimum.
  - E.g., Minimum spanning trees (MST's) for the minimum Steiner tree problem
- **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 (n: # items, W: weight)
- **Restriction**
  - Work on some subset of the original (general) problem.
  - E.g., the maximum independent set on a tree.
- **Exhaustive search/Branch and bound**
  - Feasible when the problem size is small (e.g., Integer Linear Programming, ILP).
- **Nondeterministic approaches:**
  - Simulated annealing (hill climbing), genetic algorithms, machine learning, ant colony optimization, etc.
- **Heuristics:** No guarantee of performance.

Spanning Tree vs. Steiner Tree

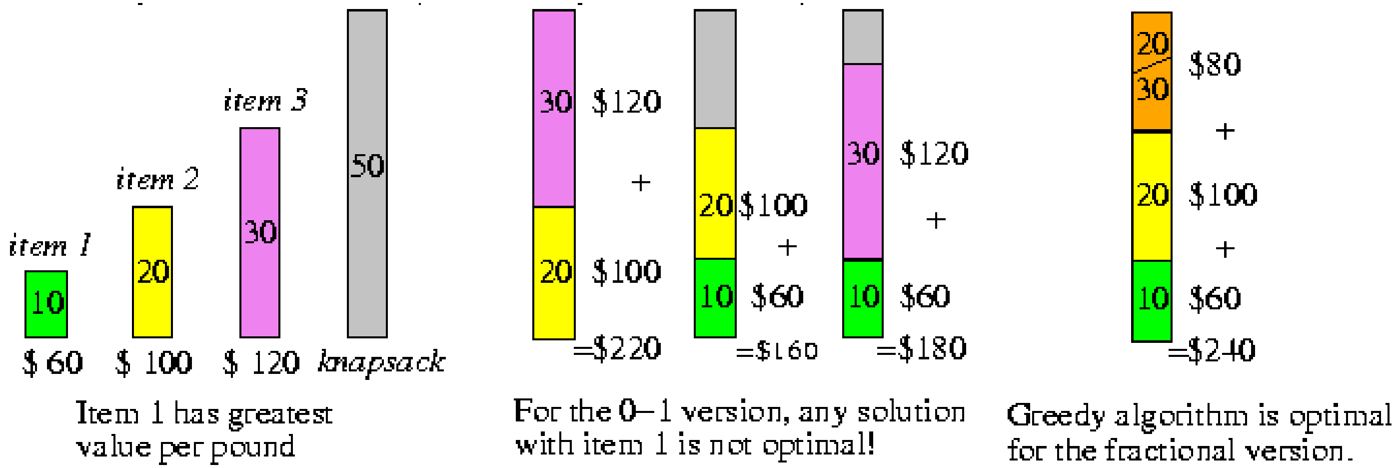
- **Manhattan distance:** If two nodes are located at coordinates (x_1, y_1) and (x_2, y_2), their Manhattan distance is $d_{12} = |x_1 - x_2| + |y_1 - y_2|$ (a.k.a. λ_1 metric)
- **Rectilinear spanning tree:** a spanning tree that connects its nodes using Manhattan paths.
- **Steiner tree:** a tree that connects its nodes, permitting additional points (**Steiner points**) to be used for the connections.
- The minimum rectilinear spanning tree problem is in **P**, while the minimum rectilinear **Steiner** tree problem is **NP-complete**.
  - The spanning tree algorithm can be a 1.5-*approximation* to the Steiner tree one (at most 50% away from the optimum).



(a) rect. spanning tree rect. Steiner tree

Knapsack Problem

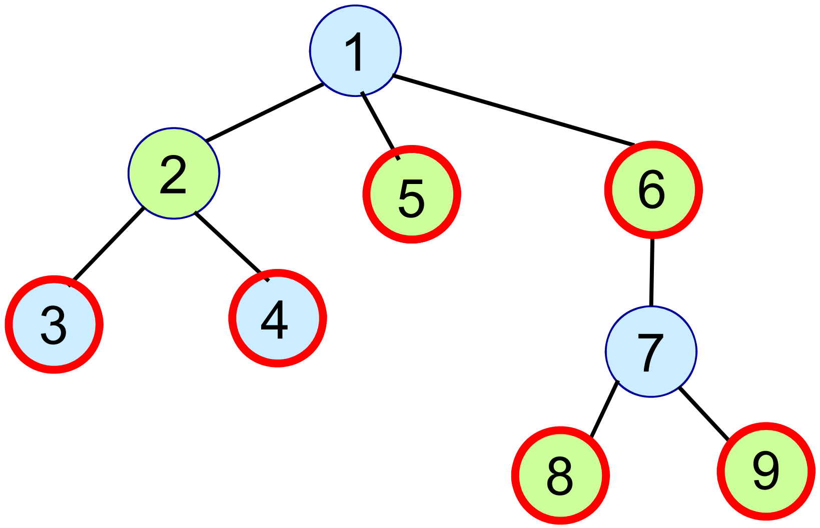
- **Knapsack Problem:** Given *n* items, with *i*th item worth v_i dollars and weighing w_i pounds, a thief wants to take as valuable a load as possible, but can carry at most *W* pounds in his knapsack.
- **The 0-1 knapsack problem:** Each item is either taken or not taken (0-1 decision).
- **The fractional knapsack problem:** Allow to take fraction of items.
- **Exp:** $\vec{v}$ = (60, 100, 120), $\vec{w}$ = (10, 20, 30), *W* = 50



- The 0-1 knapsack problem is NP-complete, but can be solved in $O(nW)$ **pseudo polynomial time** by dynamic programming (DP), while the fractional one can be solved by a greedy algorithm in $O(n \lg n)$ time.

Maximum Independent Set (MIS)

- An **independent set** of $G = (V, E)$ is a subset $V' \subseteq V$ such that G has no edge between any pair of vertices in V'
- The Maximum Independent Set Problem (MIS) is to find an independent set with the **maximum** cardinality
- MIS in general is NP-complete, but is efficiently solvable for many cases such as trees, bipartite graphs, etc.



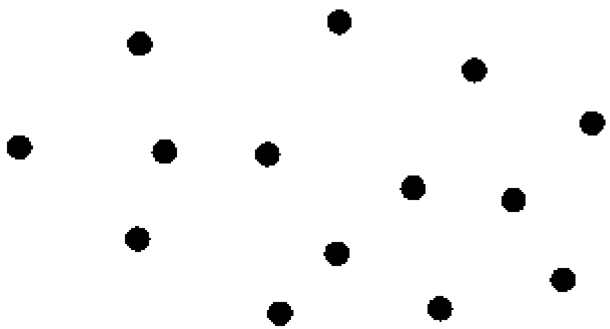
A = {1, 3}
A is an independent set (IS)

B = {1, 3, 4, 7}
C = {2, 5, 6, 8, 9}
B, C are **maximal IS's**

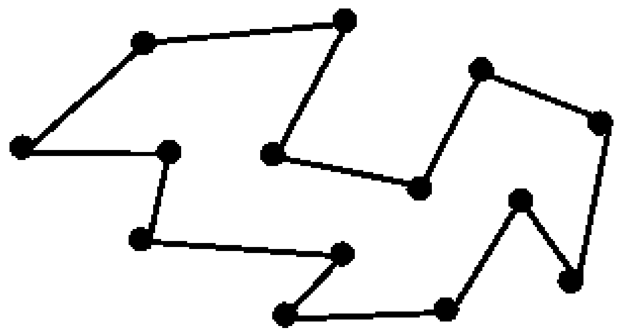
D = {3, 4, 5, 6, 8, 9}
|D| = 6
D is **the Maximum IS (MIS)**

The Traveling Salesman Problem (TSP)

- **Instance:** a set of *n* cities, a distance between each pair of cities, and a bound *B*.
- **Question (Decision Problem):** Is there a route that starts and ends at a given city, visits every city exactly once, and has total distance ≤ *B*?
- TSP is NP-complete.



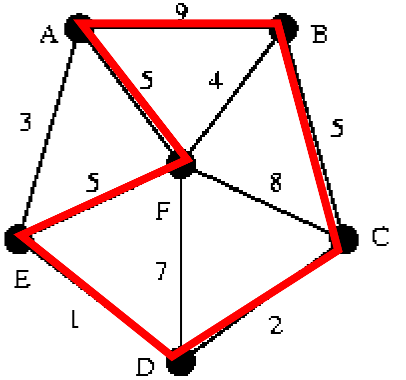
A TSP instance

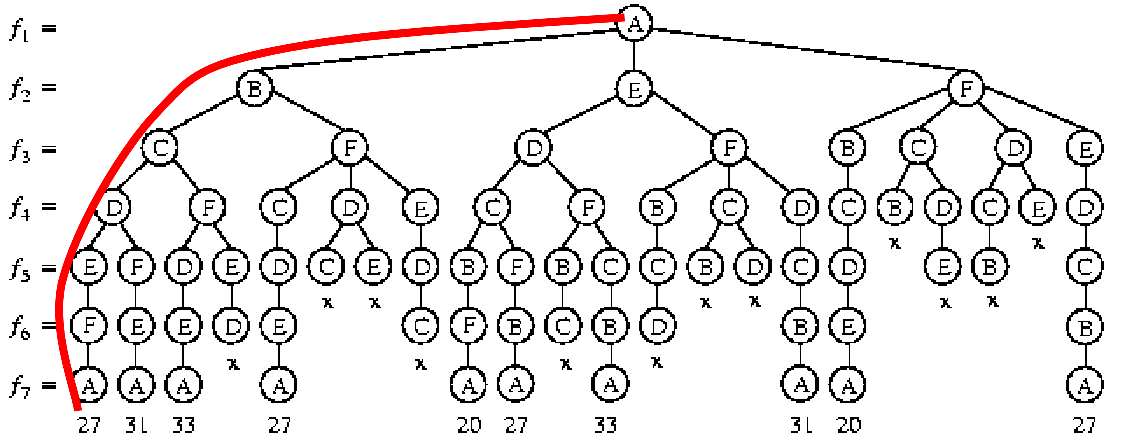


A TSP solution

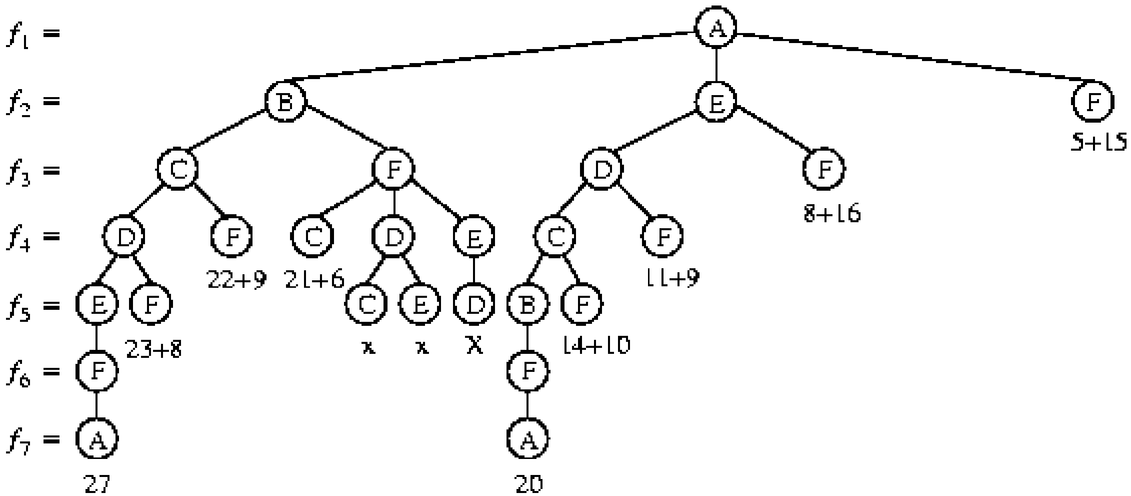
Exhaustive Search vs. Branch and Bound for TSP

- TSP example





Backtracking/exhaustive search



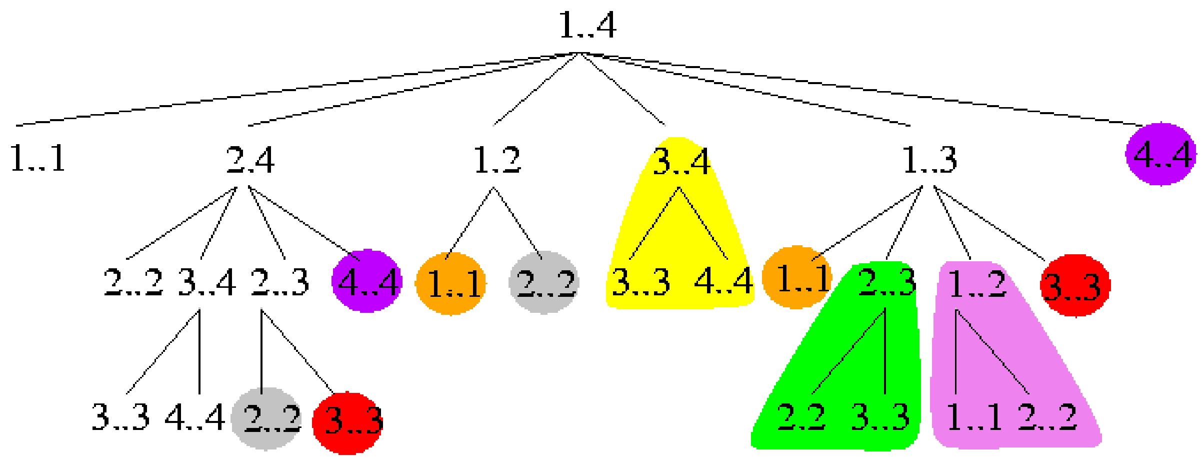
Branch and bound

Popular Algorithmic Paradigms

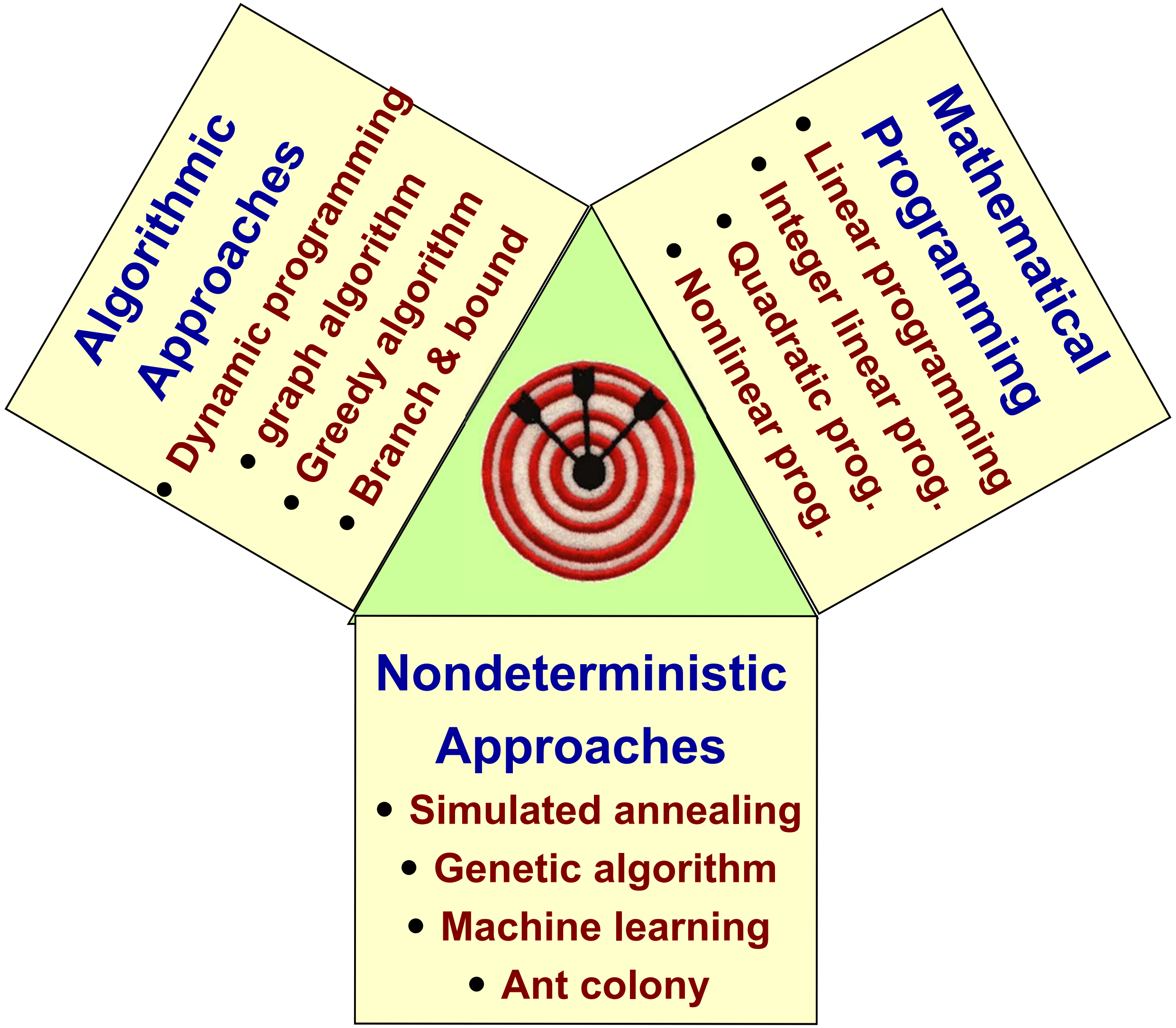
- **Exhaustive search:** Search the entire solution space.
- **Branch and bound:** A search technique with pruning.
- **Greedy method:** Pick a locally optimal solution at each step.
- **Dynamic programming:** Partition a problem into a collection of sub-problems, the sub-problems are solved, and then the original problem is solved by combining the solutions.
  - Work best when the sub-problems are **NOT independent & the # of sub-problems is polynomial, and their objects are linearly ordered & cannot be rearranged**.
- **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.

Dynamic Programming (DP) vs. Divide-and-Conquer

- Both solve problems by combining the solutions to subproblems.
- Divide-and-conquer algorithms
  - Partition a problem into **independent** subproblems, solve the subproblems recursively, and then combine their solutions to solve the original problem.
- Dynamic programming (DP)
  - Applicable when the subproblems are **dependent & the # of subproblems is "small."**
  - DP solves each subproblem just once.
  - **DP works best on objects that are linearly ordered and cannot be rearranged.**
    - Matrices in a chain, characters in a string, points around the boundary of a polygon, points on a circle, left-to-right leaves in a search tree, etc.

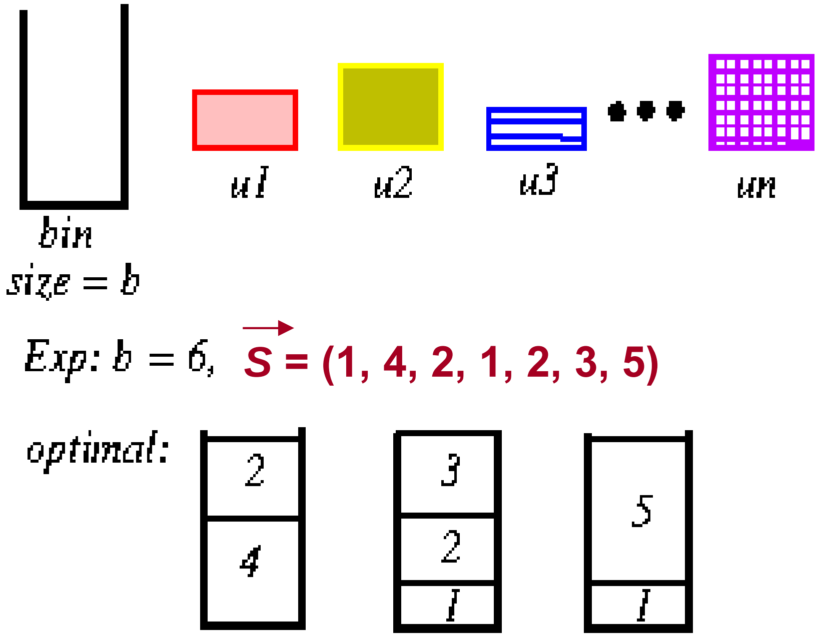


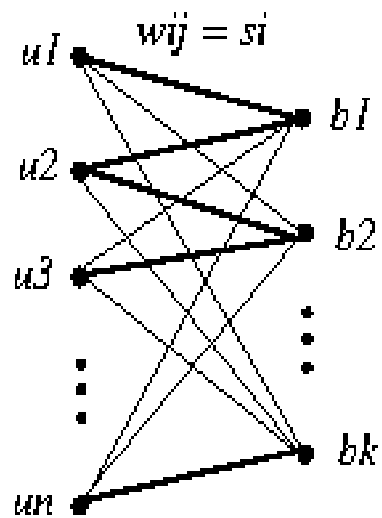
Classifications of Popular EDA Algorithms



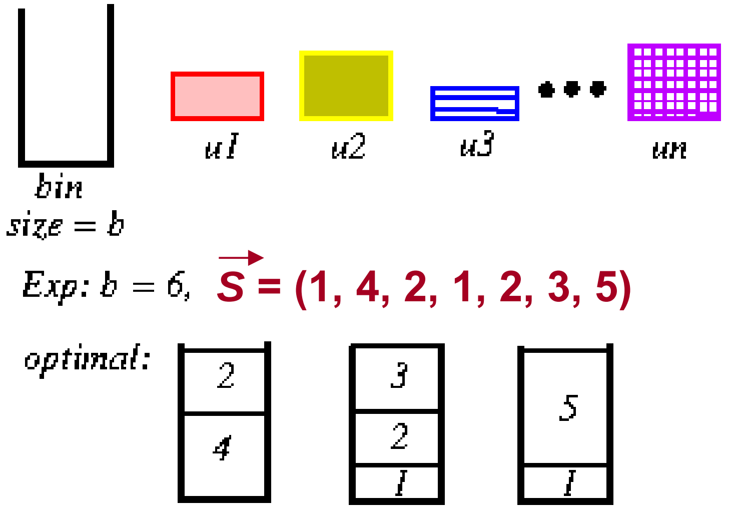
Example: Bin Packing

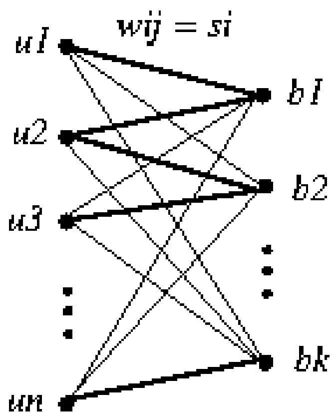
- **The Bin-Packing Problem** Π **:** Items $U = \{u_1, u_2, \ldots, u_n\}$, where u_i is of an integral size s_i; set B of bins, each with capacity b.
- **Goal:** Pack all items, minimizing # of bins used. (**NP-hard!**)





Algorithms for Bin Packing





- Greedy approximation alg.: First-Fit Decreasing (FFD)
  - $FFD(\Pi) \leq 11OPT(\Pi)/9 + 4$
- Dynamic Programming? Hierarchical Approach? Simulated annealing? …
- Mathematical Programming: Use **integer linear programming (ILP)** to find a solution using |*B*| bins, then search for the smallest feasible |*B*|.

ILP Formulation for Bin Packing

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

objective function

$$\max \sum_{(i,j)\in E} w_{ij} x_{ij}$$

subject to

constraints

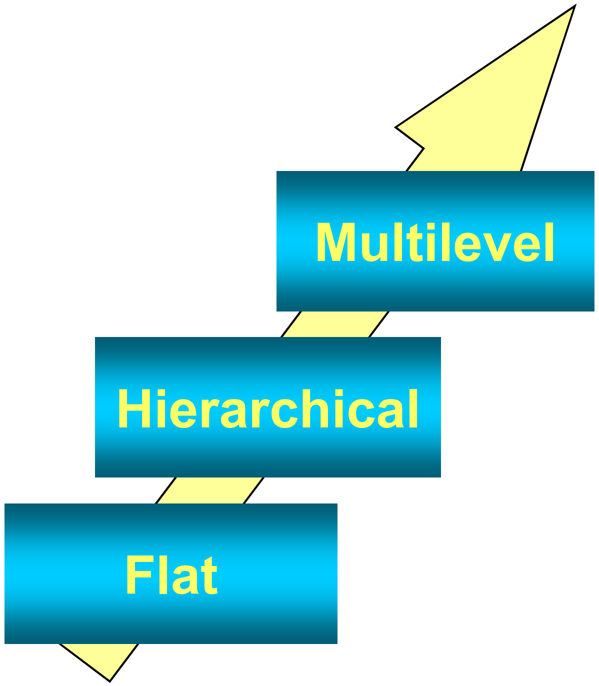
$$
\begin{aligned}
\sum_{\forall i \in U} w_{ij} x_{ij} &\leq b_j, \forall j \in B \quad /* \textit{ capacity constraint } */ && (1)\\
\sum_{\forall j \in B} x_{ij} &= 1, \forall i \in U \quad /* \textit{ assignment constraint } */ && (2)\\
\sum_{ij} x_{ij} &= n \quad /* \textit{ completeness constraint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

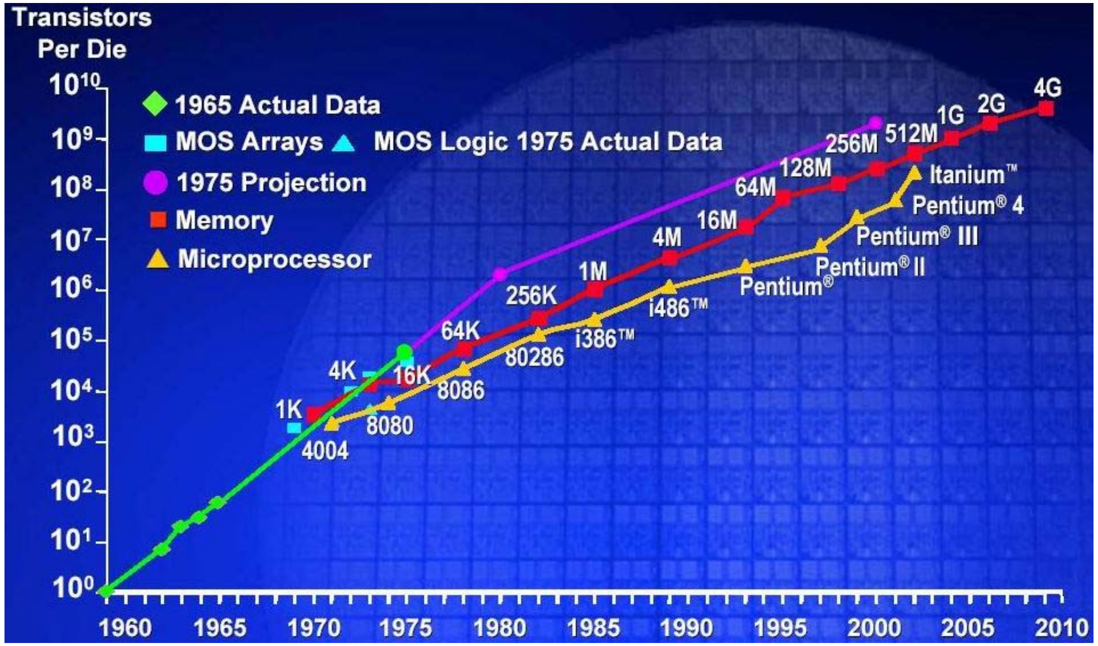
- **Step 1:** Set |*B*| to the lower bound of the # of bins.
- **Step 2:** Use the ILP to find a feasible solution.
- **Step 3:** If the solution exists, the # of bins required is |*B*|. Then exit.
- **Step 4:** Otherwise, set $|B| \leftarrow |B| + 1$. Goto Step 2.

Framework Evolution

- Billions of transistors can be fabricated in a single chip for nanometer technology.
- Need frameworks for very large-scale designs.
- Framework evolution for EDA tools:

  Flat ➔ Hierarchical ➔ Multilevel

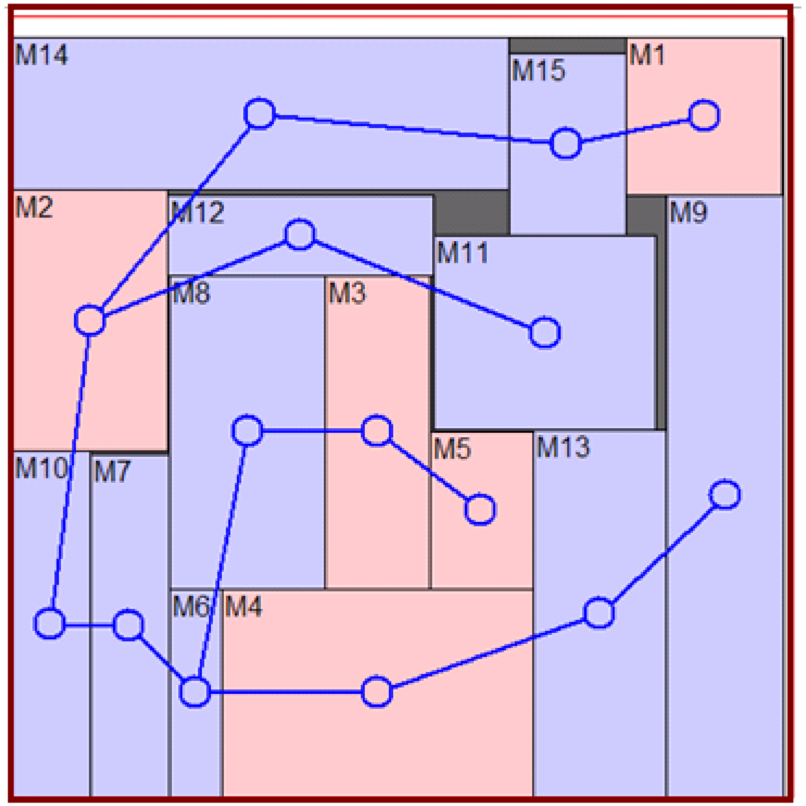




Source: Intel (ISSCC-03)

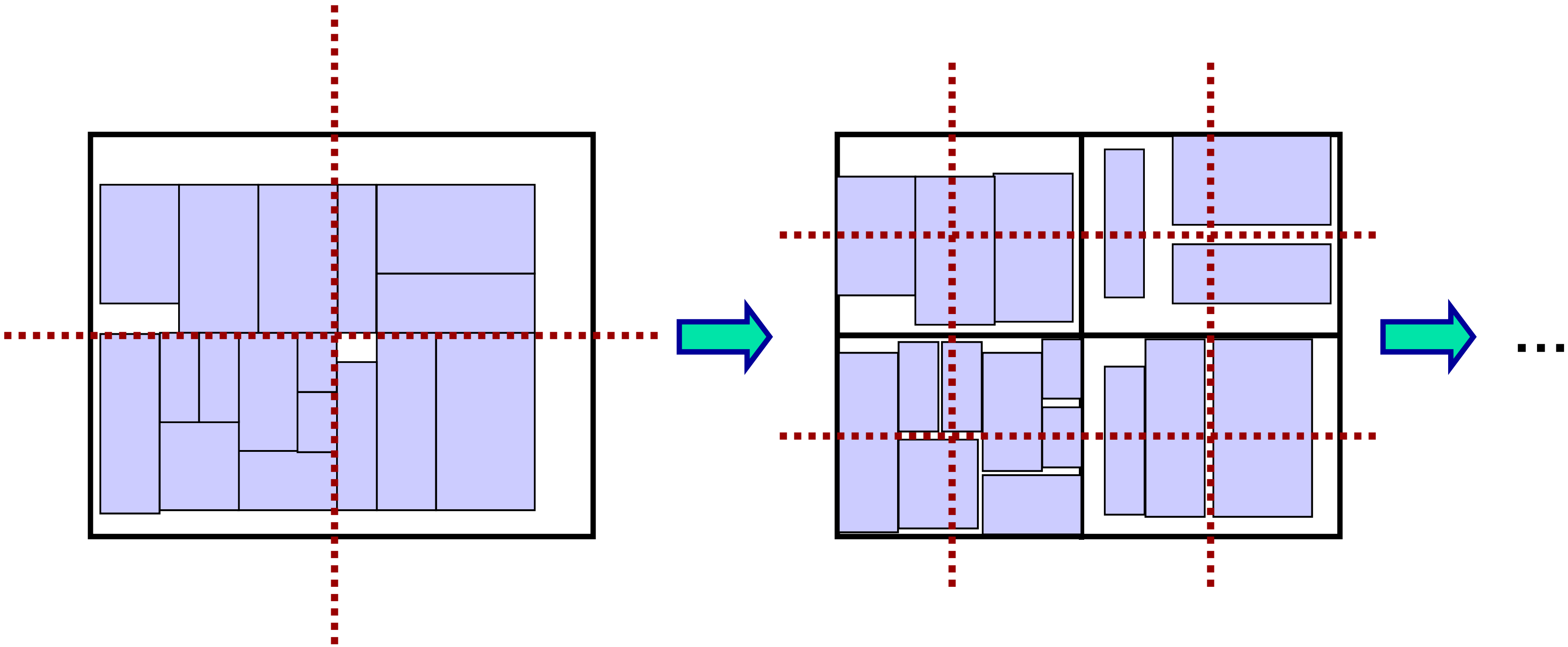
Flat Framework for 2D Bin Packing (Floorplanning)

- Process the circuit components in the whole chip
- Limitation: Good for small-scale designs, but hard to handle larger problems directly



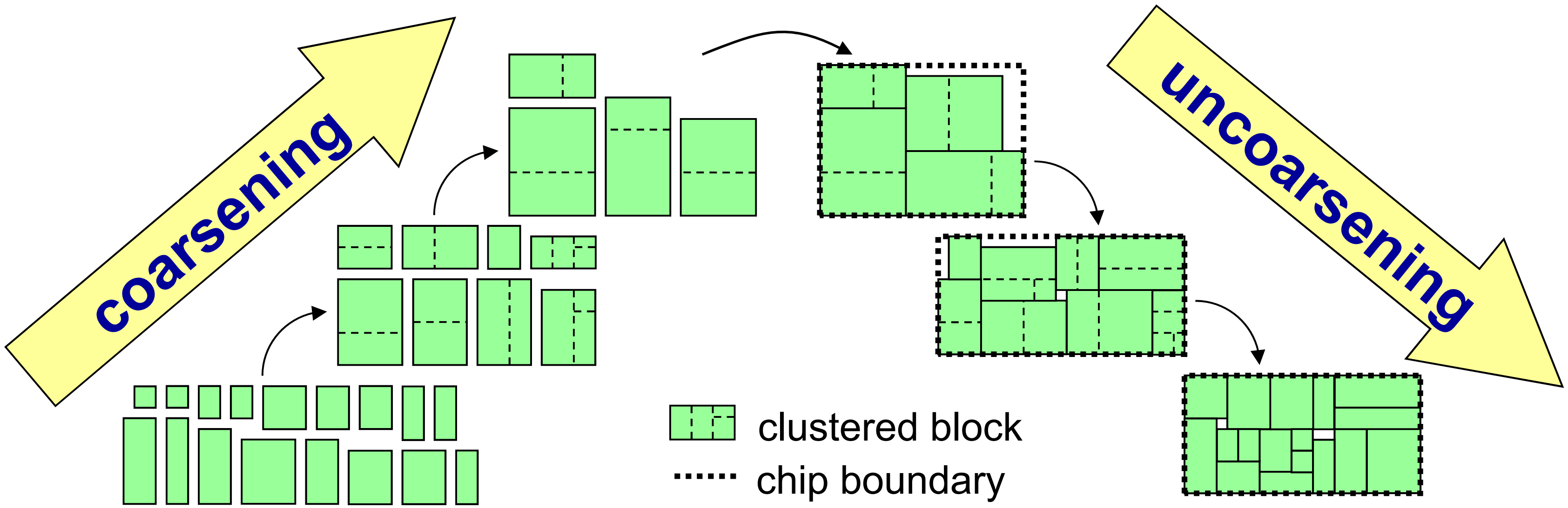
Hierarchical Framework

- The hierarchical approach recursively divides a circuit region into a set of subregions and solve those subproblems ***independently***.
- Good for scalability, but lack the global information for the interaction among subregions.

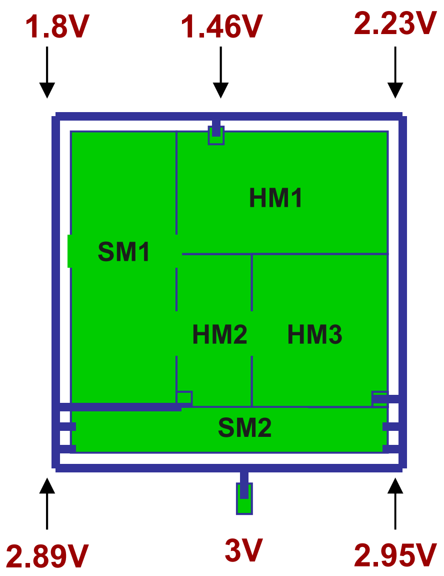


Multilevel Floorplanning

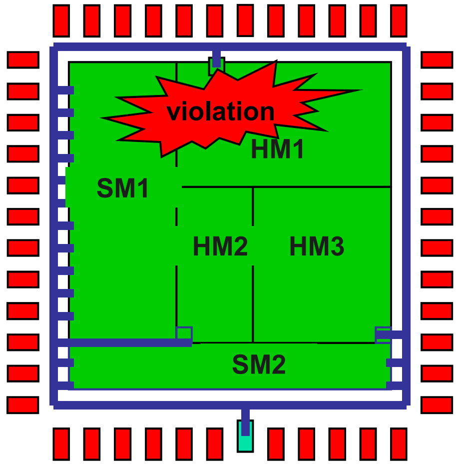
- Lee, Hsu, Chang, Yang, "Multilevel floorplanning/placement for large-scale modules using B*-trees," DAC-03 (TCAD-07)
- Bottom-up Coarsening (clustering): Iteratively groups a set of circuit components and collects global information.
- Top-down Uncoarsening (declustering): Iteratively ungroups clustered components and refines the solution.
- Good for scalability and quality trade-off

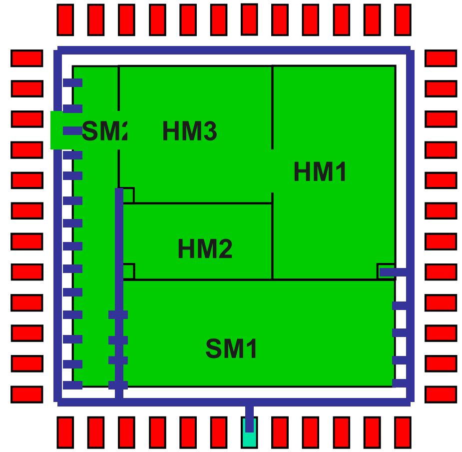


Power-Aware Floorplanning



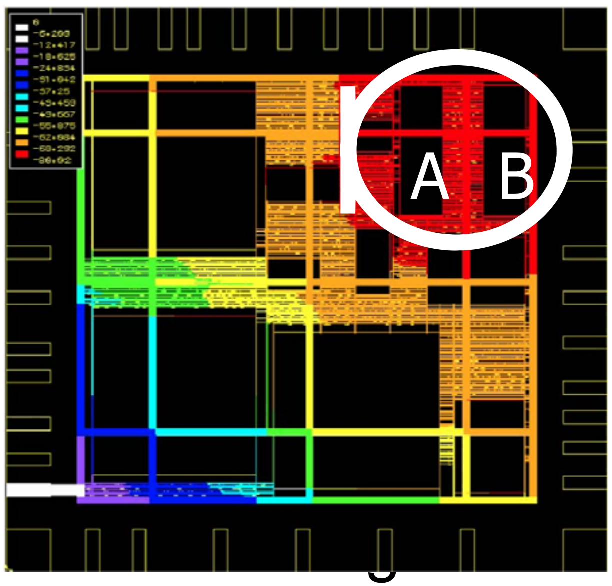
IR-drop violation

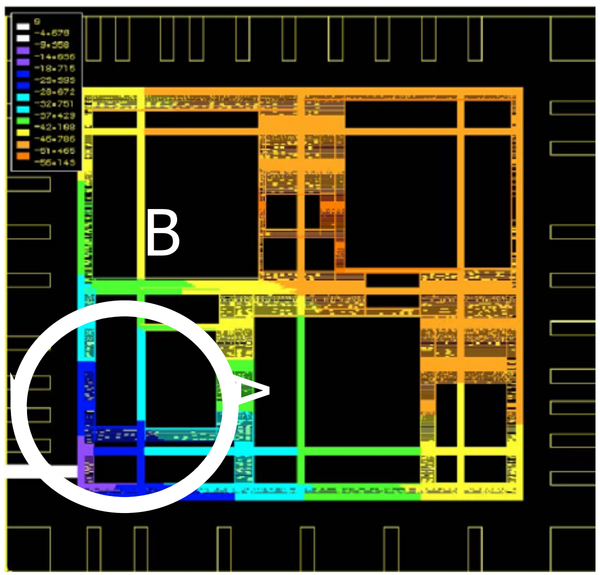




OpenRISC

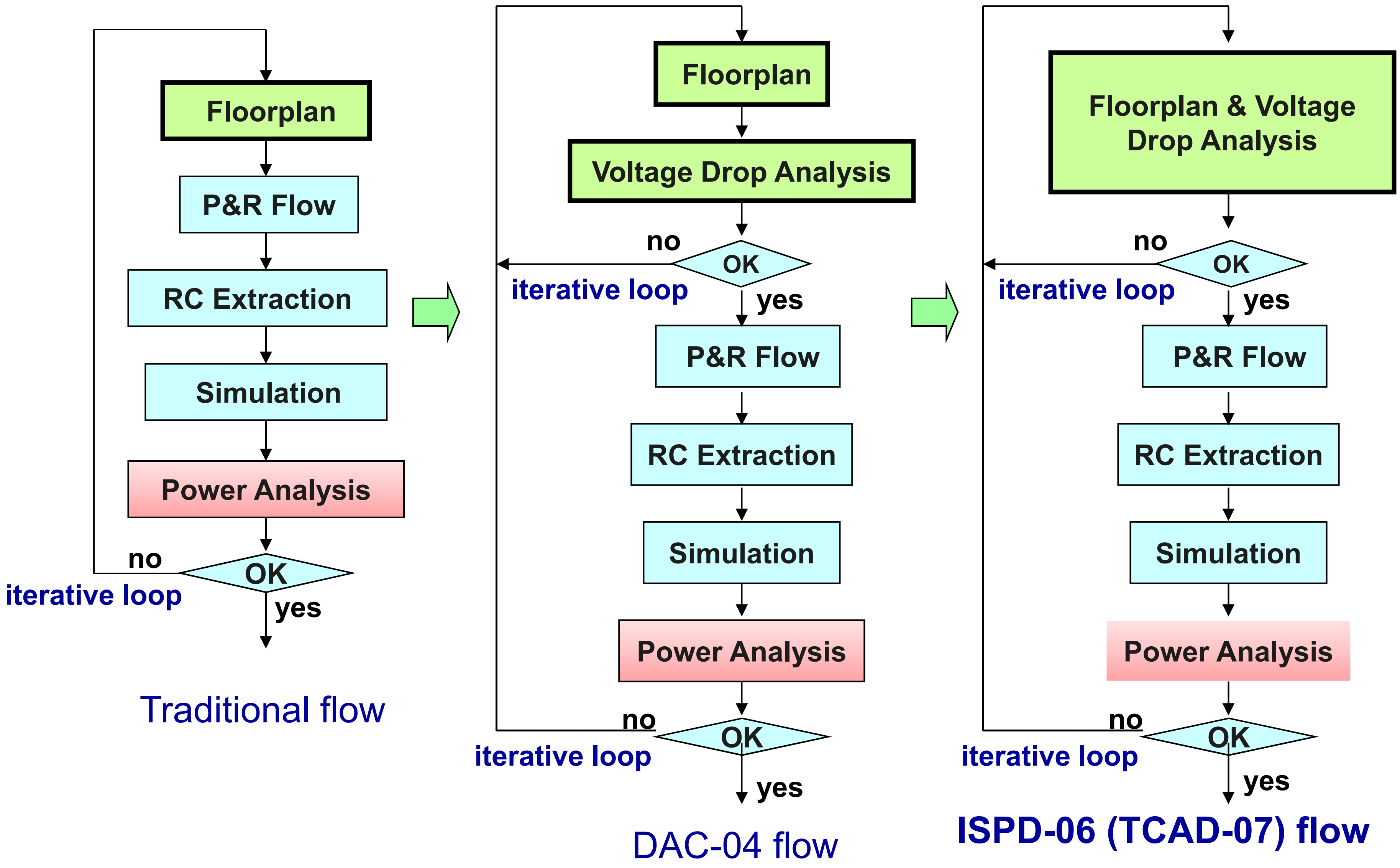
Design flow A



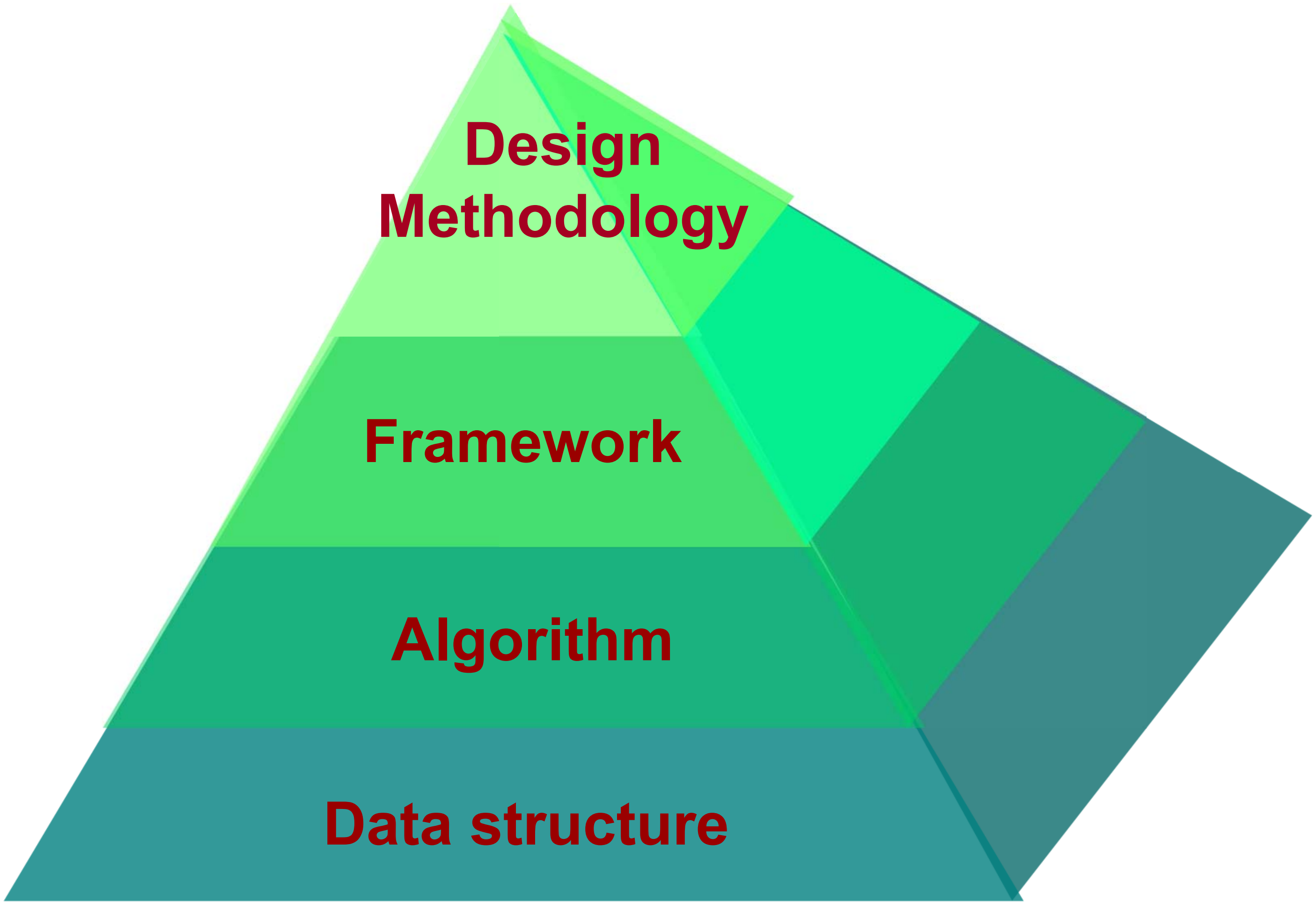


Design flow B

Design Methodology Evolution



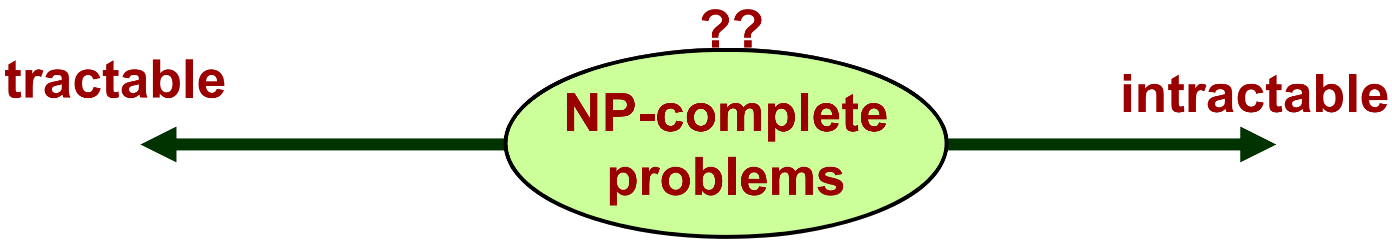
Pyramid for Solving an EDA Problem



Physical Design Related Conferences/Journals

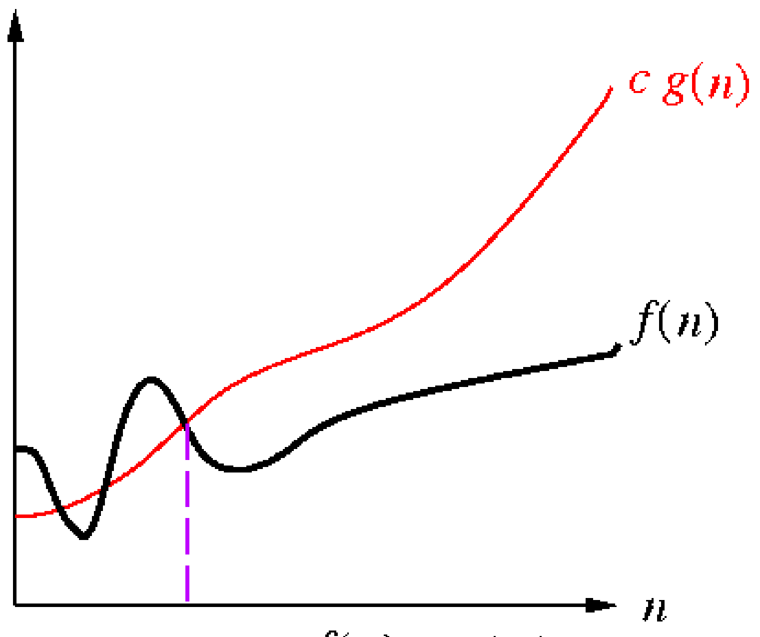
- PD Conferences:
  - **ACM/IEEE Design Automation Conference (DAC)**
  - **IEEE/ACM Int'l Conference on Computer-Aided Design (ICCAD)**
  - ACM Int'l Symposium on Physical Design (ISPD)
  - ACM/IEEE Asia and South Pacific Design Automation Conf. (ASP-DAC)
  - ACM/IEEE Design, Automation, and Test in Europe (DATE)
  - IEEE Int'l Conference on Computer Design (ICCD)
  - IEEE Int'l Symposium on Circuits and Systems (ISCAS)
  - Others: GLSVLSI, ISLPED, ISQED, VLSI-DAT, VLSI Design/CAD Symp.
- PD Journals/magazine:
  - **IEEE Transactions on Computer-Aided Design (TCAD)**
  - **ACM Transactions on Design Automation of Electronic Systems (TODAES)**
  - **IEEE Transactions on VLSI Systems (TVLSI)**
  - **IEEE Design and Test of Computers (magazine)**
  - IEEE Transactions on Computers (TC)
  - Others: INTEGRATION, IET journals/proceedings, IEICE transactions
- EDA Societies
  - **IEEE Council on Electronic Design Automation (CEDA)**
  - **ACM Special Interest Group on Design Automation (SIGDA)**
  - Others: IEEE CAS, CS, SSCS, etc.

Appendix:
Computational Complexity & NP-Completeness



O: Upper Bounding Function

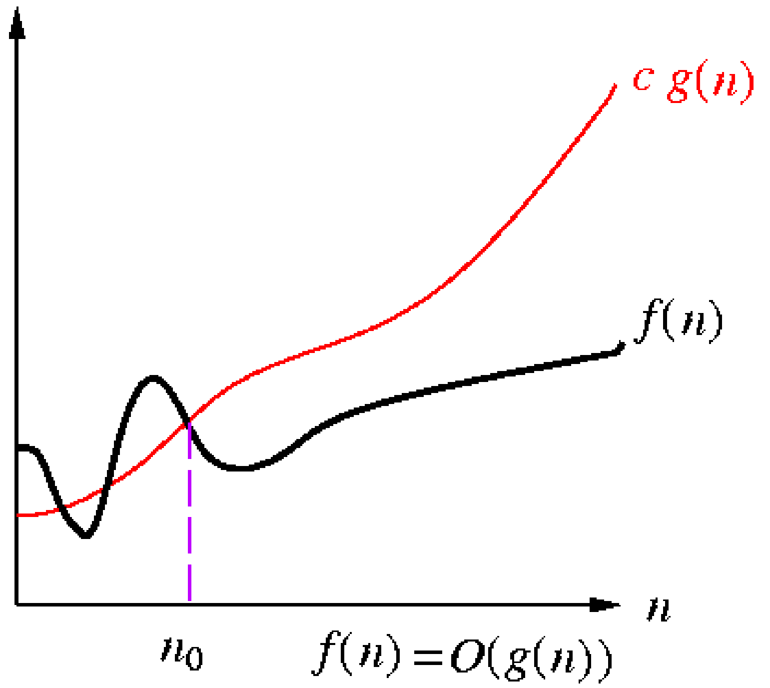
- **Def:** $f(n) = O(g(n))$ if $\exists\ c > 0$ and $n_0 > 0$ such that $0 \leq \boldsymbol{f(n)} \leq \boldsymbol{cg(n)}$ for all $n \geq n_0$.
  - Examples: $2n^2 + 3n = O(n^2)$, $2n^2 = O(n^3)$, $3n \lg n = O(n^2)$
- Intuition: $f(n)$ "$\leq$" $g(n)$ when we ignore constant multiples and small values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



$f(n) = O(g(n))$

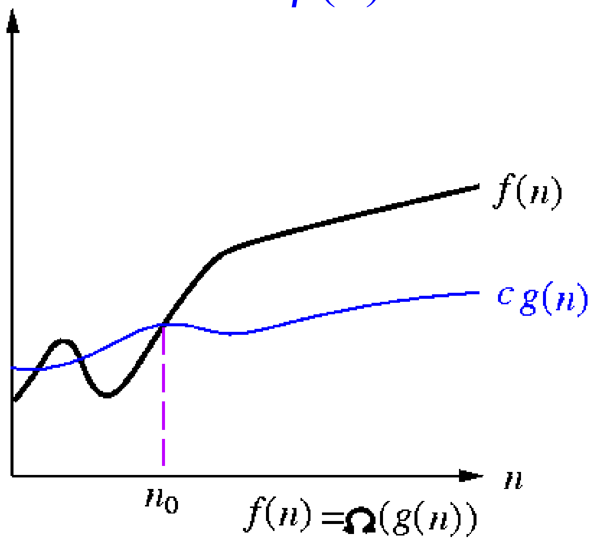
Big-*O* Notation

- How to show *O* (Big-Oh) relationships?
  - $f(n) = O(g(n))$ iff $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$ for some $c \geq 0$.
- "An algorithm has worst-case running time $O(f(n))$": there is a constant c s.t. for every n big enough, **every execution** on an input of size n takes **at most** $cf(n)$ time.



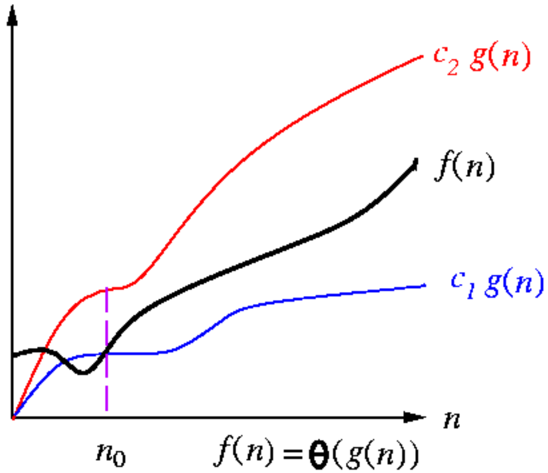
Ω : Lower Bounding Function

- **Def:** $f(n) = \Omega(g(n))$ if $\exists\ c > 0$ and $n_0 > 0$ such that $0 \le$ $\boldsymbol{cg(n) \le f(n)}$ for all $n \ge n_0$.
  - Examples: $2n^2 + 3n = \Omega(n^2)$, $2n^3 = \Omega(n^2)$, $3n \lg n \ne \Omega(n^2)$
- Intuition: $f(n)$ " $\ge$ " $g(n)$ when we ignore constant multiples and small values of n.
- How to show Ω (Big-Omega) relationships?
  - $f(n) = \Omega(g(n))$ if $\lim_{n \to \infty} \frac{g(n)}{f(n)} = c$ for some $c \ge 0$.



θ: Tightly Bounding Function

- **Def:** $f(n) = \theta(g(n))$ if $\exists\ c_1, c_2 > 0$ and $n_0 > 0$ such that $0 \leq \boldsymbol{c_1 g(n) \leq f(n) \leq c_2\, g(n)}$ for all $n \geq n_0$.
  - Examples: $2n^2 + 3n = \theta(n^2)$, $2n^3 \neq \theta(n^2)$, $3n \lg n \neq \theta(n)$
- Intuition: $f(n)$ " = " $g(n)$ when we ignore constant multiples and small values of n.
- How to show θ relationships?
  - Show both "big Oh" (O) and "Big Omega" (Ω) relationships.
  - $f(n) = \theta(g(n))$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$ for some $\boldsymbol{c > 0}$.



Computational Complexity

- Computational complexity: an abstract measure of the **time** and **space** necessary to execute an algorithm as functions of its "input size".
- Input size: size of encoded "binary" strings.
  - sort n words of bounded length   input size: n
  - **the input is the integer n input size: lg n**
  - the input is the graph $G(V, E)$ input size: $|V|$ and $|E|$
- Runtime comparison: assume 1 BIPS,1 instruction/op.

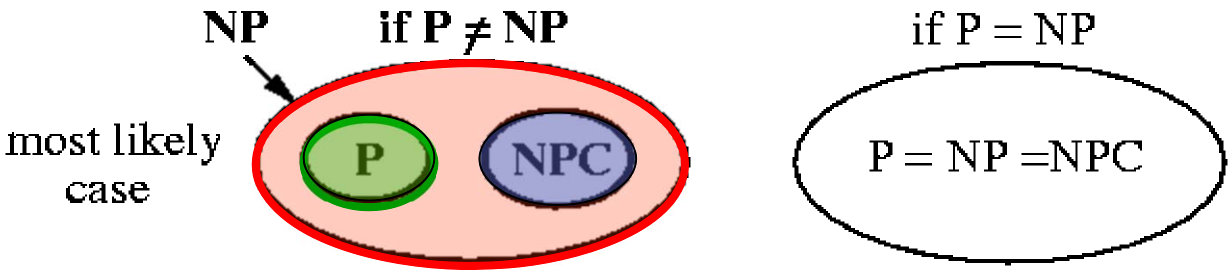
Time	Big-Oh	n = 10	n = 100	n = 10^4	n = 10^6	n = 10^8
500	O(1)	$5*10^{-7}$ sec	$5*10^{-7}$ sec	$5*10^{-7}$ sec	$5*10^{-7}$ sec	$5*10^{-7}$ sec
$3n$	O(n)	$3*10^{-8}$ sec	$3*10^{-7}$ sec	$3*10^{-5}$ sec	0.003 sec	0.3 sec
n lg n	O(n lg n)	$3*10^{-8}$ sec	$6*10^{-7}$ sec	$1*10^{-4}$ sec	0.018 sec	2.5 sec
n^2	O(n^2)	$1*10^{-7}$ sec	$1*10^{-5}$ sec	0. 1 sec	16.7 min	116 days
n^3	O(n^3)	$1*10^{-6}$ sec	0.001 sec	16.7 min	31.7 yr	∞
2^n	O(2^n)	$1*10^{-6}$ sec	$4*10^{11}$ cent.	∞	∞	∞
$n!$	O($n!$)	0.003 sec	∞	∞	∞	∞

Asymptotic Functions

- **Polynomial-time complexity:** $O(p(n))$, where n is the **input size** and $p(n)$ is a polynomial function of n **($p(n) = n^{O(1)}$).**
- 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

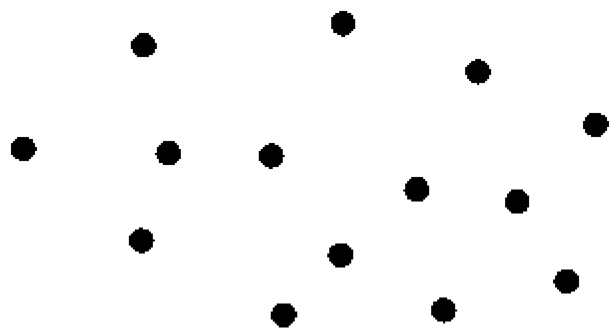
Complexity Classes

- Developed by S. Cook and R. Karp in early 1970.
- **Class P:** class of problems that can be **solved** in polynomial time in the **size of input**.
  - **Size of input:** size of encoded “binary” strings.
  - Edmonds: Problems in P are considered **tractable**.
- **Class NP** (**N**ondeterministic **P**olynomial): class of problems that can be **verified** in polynomial time in the size of input.
  - P = NP?
- **Class NP-complete (NPC): Any** NPC problem can be solved in polynomial time   **All** problems in NP can be solved in polynomial time (i.e., P = NP).

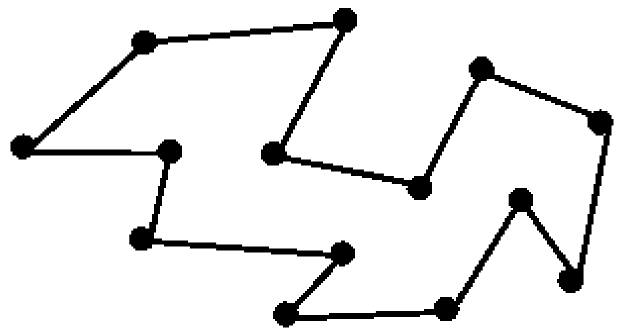


The Traveling Salesman Problem (TSP)

- **Instance:** a set of *n* cities, a distance between each pair of cities, and a bound *B*.
- **Question:** is there a route that starts and ends at a given city, visits every city exactly once, and has total distance $\leq B$?



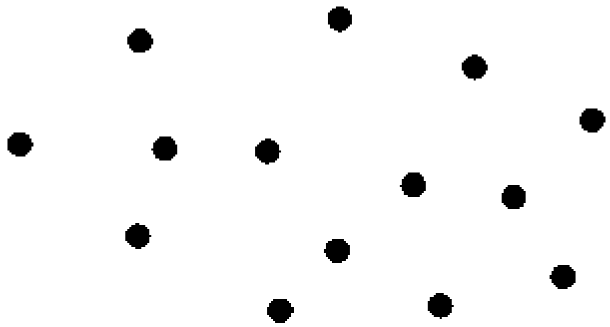
A TSP instance



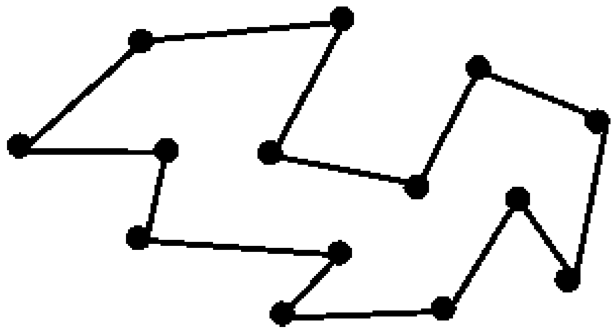
A TSP solution

NP vs. P

- TSP $\in$ NP.
  - Need to **check** a solution (tour) in polynomial time.
    - Guess a tour.
    - Check if the tour visits every city exactly once, returns to the start, and total distance $\leq B$.
- TSP $\in$ P?
  - Need to solve (find a tour) in polynomial time.
  - Still unknown if TSP $\in$ P.



A TSP instance



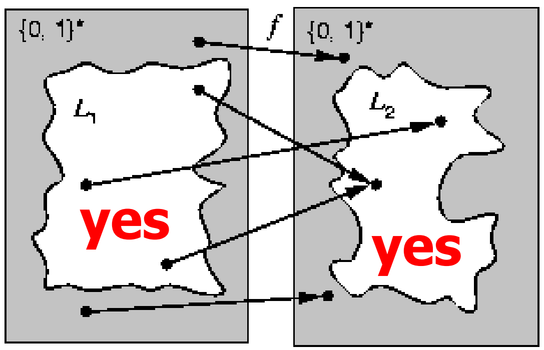
A TSP solution

Decision Problems and NP-Completeness

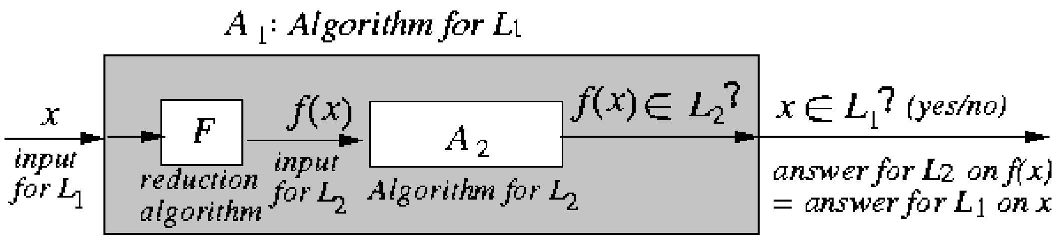
- **Decision problems:** those having yes/no answers.
  - TSP: Given a set of cities, a distance between each pair of cities, and a bound *B*, **is there a route** that starts and ends at a given city, visits every city exactly once, and has total distance at most *B*?
- **Optimization problems:** those finding a legal configuration such that its cost is minimum (or maximum).
  - TSP: Given a set of cities and a distance between each pair of cities, **find the distance of a "minimum route"** that starts and ends at a given city and visits every city exactly once.
- Could apply binary search on decision problems to obtain solutions to optimization problems.
- NP-completeness is associated with decision problems.
- cf., **Optimal** solutions/costs, optimal (**exact**) algorithms (Attn: optimal $\neq$ exact in the theoretic computer science community).

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?
  - E.g., System of difference constraints (L_1) vs. SSSP (L_2)
- **Polynomial-time reduction** f **from** L_1 **to** L_2: $\boldsymbol{L_1 \leq_P L_2}$
  - f reduces any 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: $\{0, 1\}^* \to \{0, 1\}^*$ s.t. $\boldsymbol{x \in L_1}$ **iff** $\boldsymbol{f(x) \in L_2}$, $\forall x \in \{0, 1\}^*$.
    - **L_2 is at least as hard as L_1.**
- f is computable in polynomial time.

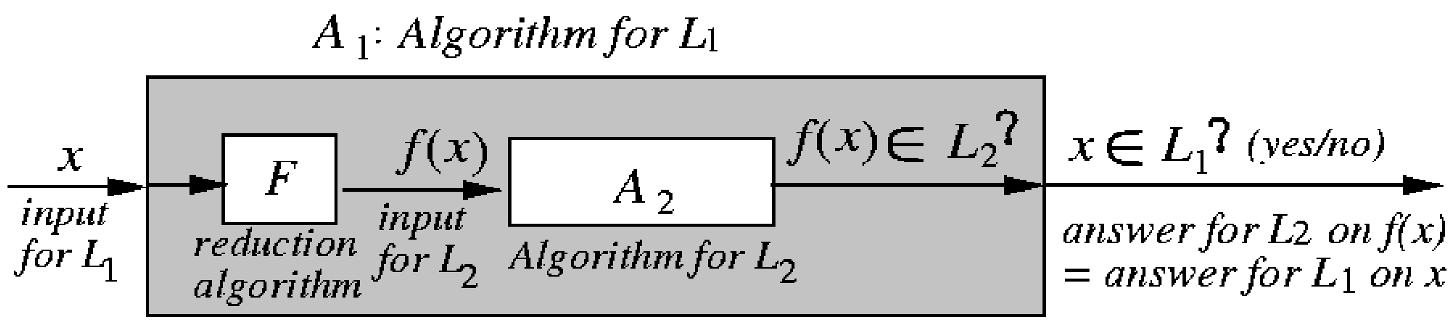


$L_1 \leq_p L_2$



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 $L_2 \leq_P L_3 \Rightarrow L_1 \leq_P L_3$.



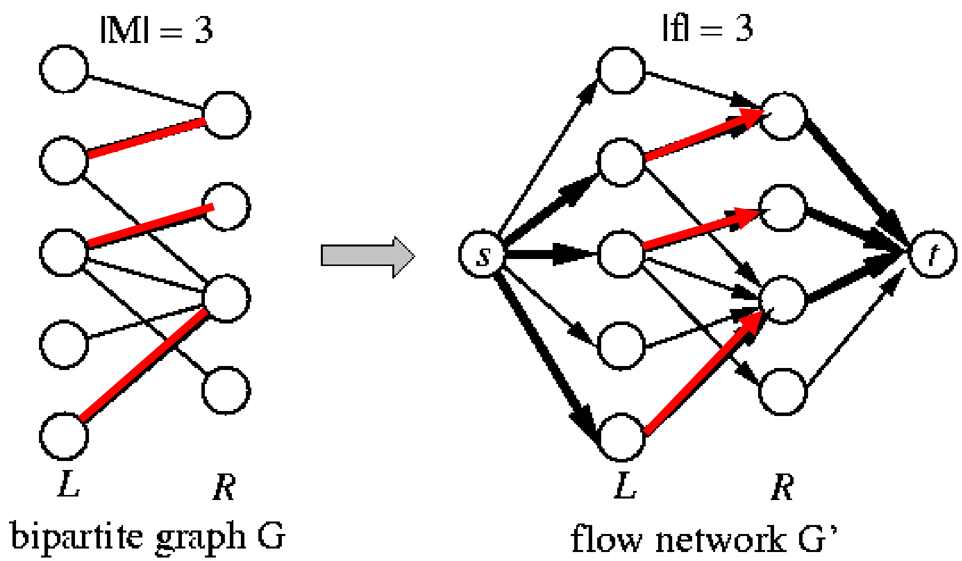
L_1: Bipartite cardinality matching vs. L_2: maximum flow

Example Reduction

- Example reduction from the matching problem to the max-flow one.
- Given a bipartite graph $G = (V, E)$, $V = L \cup R$, construct a unit-capacity flow network $G' = (V', E')$:

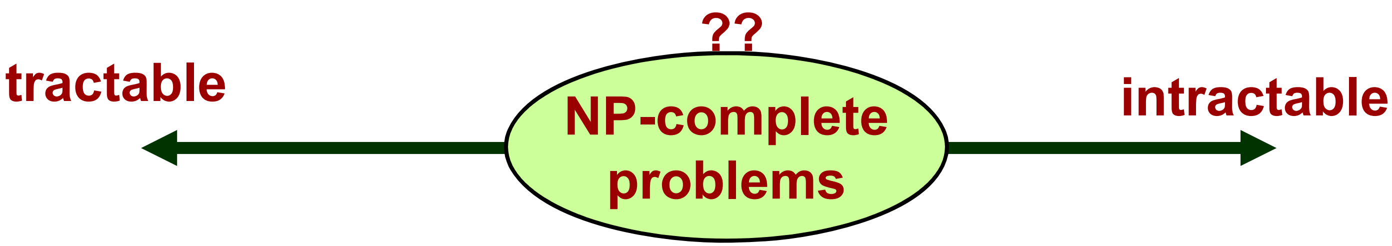
  $V' = V \cup \{s, t\}$

  $E' = \{(s, u): u \in L\} \cup \{(u, v): u \in L, v \in R, (u, v) \in E\} \cup \{(v, t): v \in R\}$.
- The cardinality of a maximum matching in G = the value of a maximum flow in G' (i.e., $|M| = |f|$).



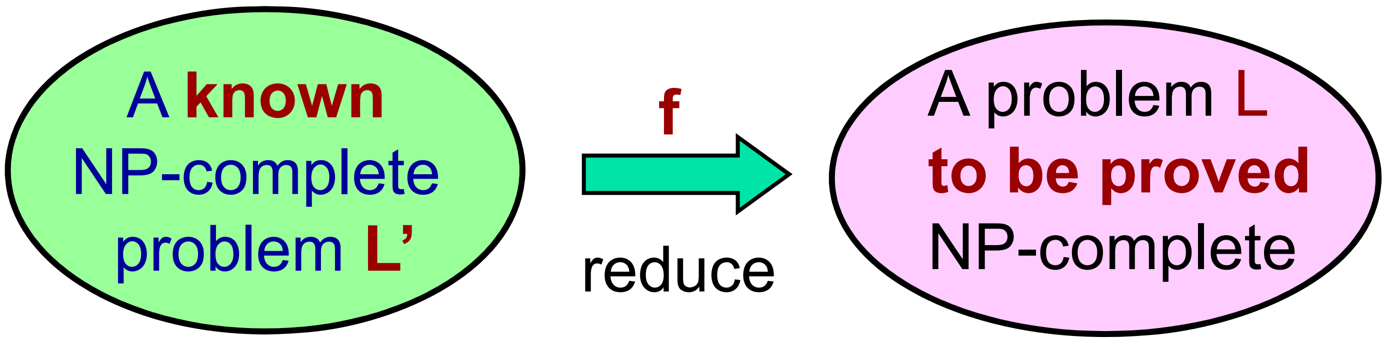
NP-Completeness

- A **decision** problem L is **NP-complete (NPC)** if
  1. $L \in$ NP, and
  2. $L' \leq_P L$ for every $L' \in$ NP.
- **NP-hard:** If L satisfies property 2, but not necessarily property 1, we say that L is **NP-hard**.
- Suppose $L \in$ NPC.
  - If $L \in P$, then there exists a polynomial-time algorithm for every $L' \in$ NP (i.e., P = NP).
  - If $L \notin P$, then there exists no polynomial-time algorithm for any $L' \in$ NPC (i.e., P $\neq$ NP).
- **NP-completeness: worst-case** analyse for a **decision** problem.



Proving NP-Completeness

- **Five steps for proving that *L* is NP-complete:**
  1. Prove $L \in$ 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.

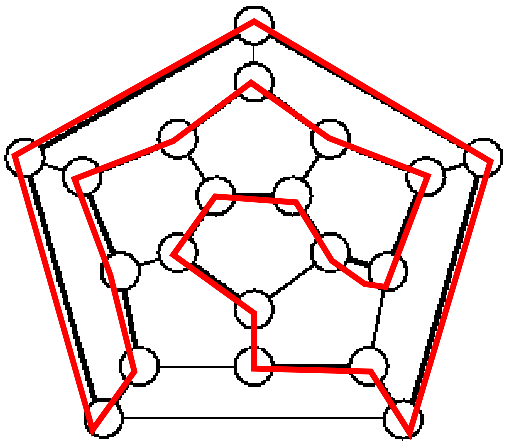


Here we intend to show how difficult L is!!

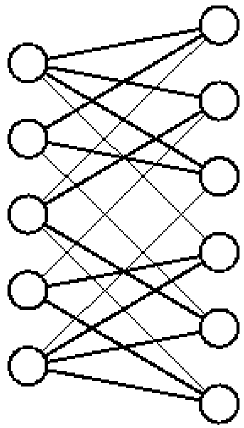
Cf. matching $\leq_P$ maximum flow to show how easy matching is!!

TSP Is NP-Complete

- **TSP (The Traveling Salesman Problem)** $\in$ NP
- **TSP is NP-hard:** 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).
- The Hamiltonian Circuit Problem (HC): known to be NP-complete
  - **Instance:** an undirected graph $G = (V, E)$.
  - **Question:** is there a cycle in G that includes every vertex exactly once?



Hamiltonian



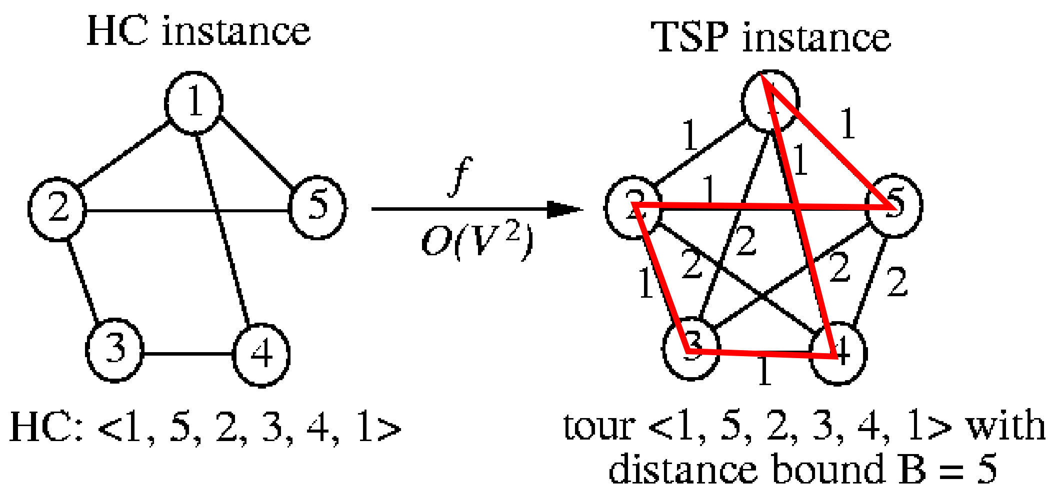
nonhamiltonian

HC $\leq_P$ TSP: Step 1

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 two cities u and v

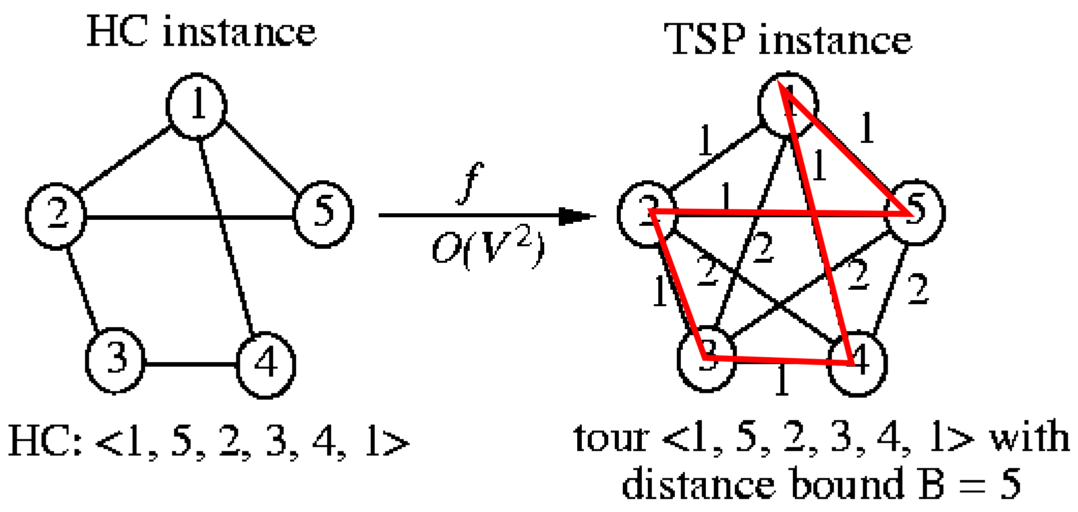
$$d(u,v) = \begin{cases} 1, & \text{if } (u,v) \in E, \\ 2, & \text{if } (u,v) \notin E. \end{cases}$$

     - Set bound $B = n$.
   - f can be computed in $O(V^2)$ time.



HC $\leq_P$ TSP: Step 2

2. G has an HC iff the reduced instance has a TSP with distance $\leq B$.
   - $x \in$ HC $\Rightarrow f(x) \in$ TSP.
     - Suppose the HC is $h = <v_1, v_2, \ldots, v_n, v_1>$. 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, $f(x) \in$ TSP ($f(x)$ has a TSP tour with distance $\leq B$).



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

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