INT102
Algorithmic Foundations And Problem Solving

Coping with the Limitations of Algorithm Power

(Approximation Algorithms)

Dr Jia WANG

Department of Computer Science

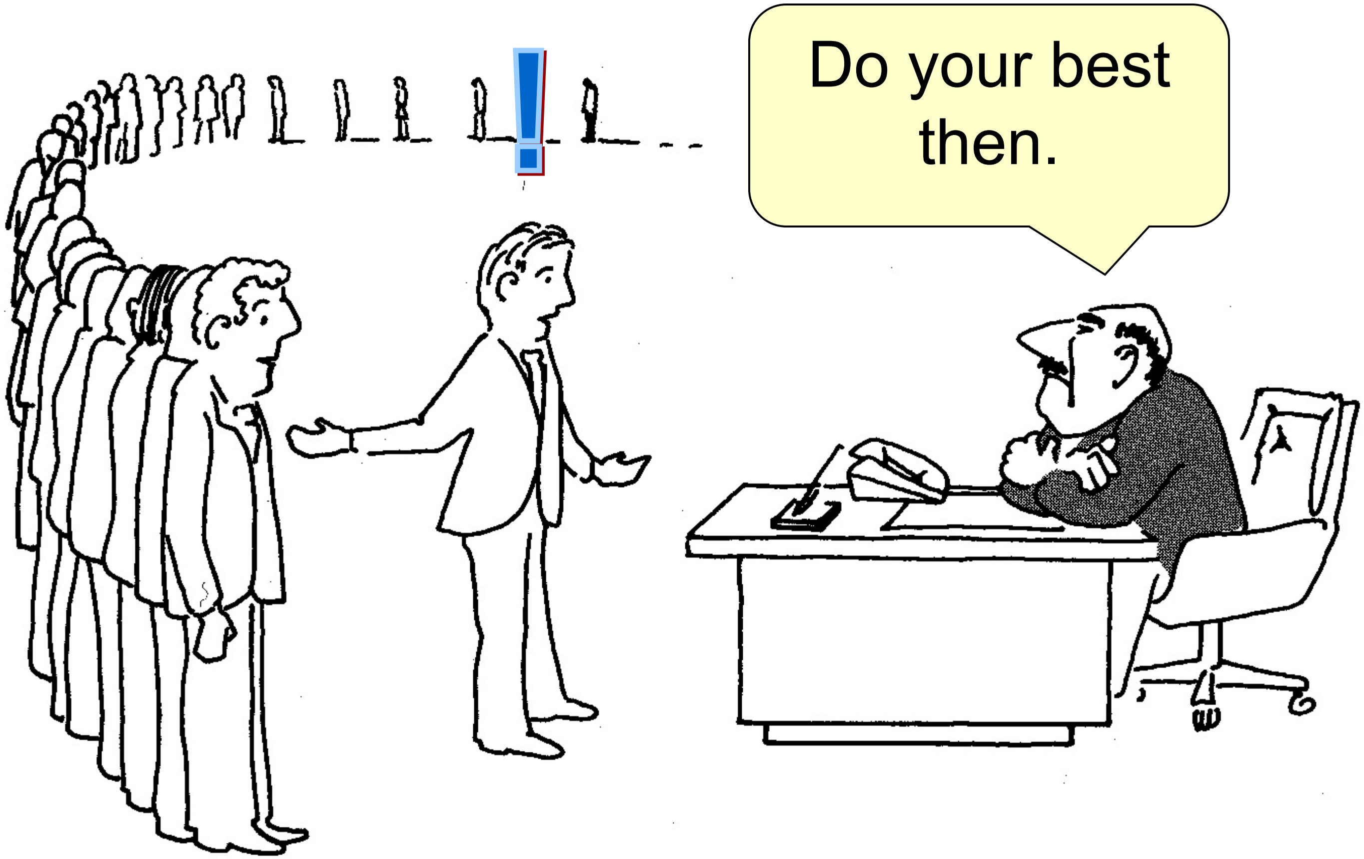
西交利物浦大学
Xi'an Jiaotong-Liverpool University

Acknowledgment: The slides are adapted from ones by Dr. Prudence Wong

Introduction

- Objectives:
  - To formalize the notion of approximation.
  - To demonstrate several such algorithms.
- Overview:
  - Optimization and Approximation
  - VERTEX-COVER, TSP

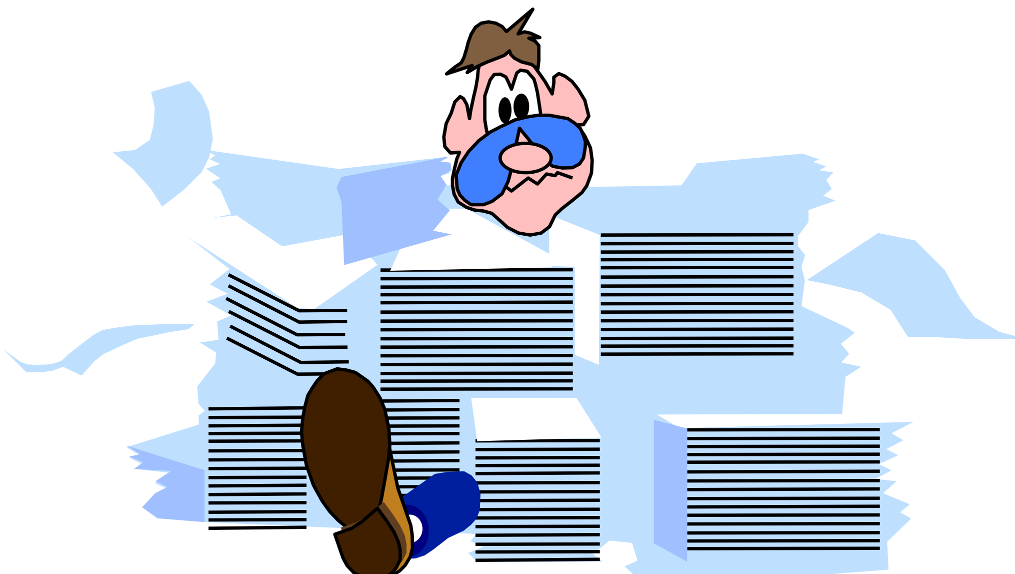
NP-completeness



"I can't find an efficient algorithm, but neither can all these famous people."

Motivation

- By now we've seen many NP-Complete problems.
- We conjecture none of them has polynomial time algorithm.

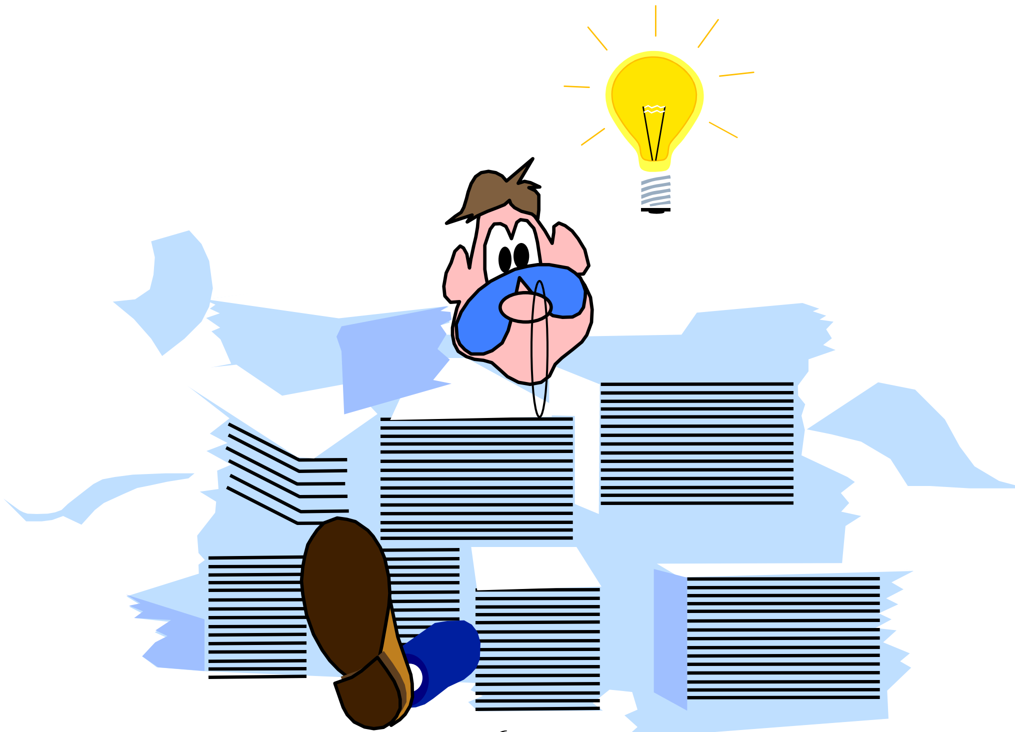


Motivation

- Is this a dead-end? Should we give up altogether?

Motivation

- Or maybe we can settle for good approximation algorithms?



Coping With NP-Hardness

Brute-force algorithms.

- Develop clever enumeration strategies.
- Guaranteed to find optimal solution.
- No guarantees on running time.

Heuristics.

- Develop intuitive algorithms.
- Guaranteed to run in polynomial time.
- No guarantees on quality of solution.

Approximation algorithms.

- Guaranteed to run in polynomial time.
- Guaranteed to find "high quality" solution, say within 1% of optimum.

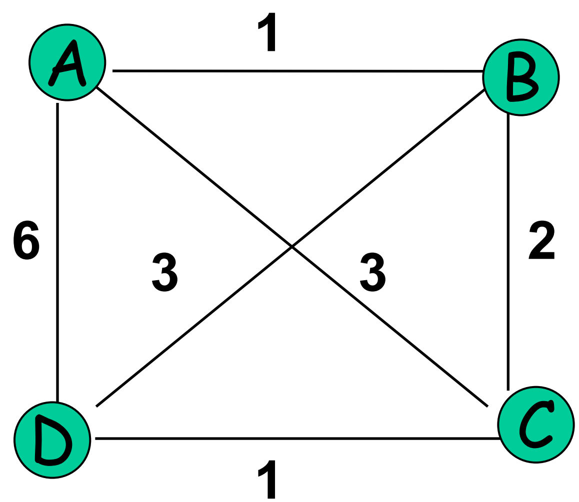
Obstacle: need to prove a solution's value is close to optimum, without even knowing what optimum value is!

Heuristics

- A heuristic is a common-sense *rule* drawn from experience
  - not a mathematically proven assertion
  - a “rule-of-thumb”
- Examples:
  - TSP: go to next nearest city
  - Knapsack: start with highest value/weight ratio

Nearest-Neighbor Algorithm for TSP

Starting at some city, always go to the nearest unvisited city, and, after visiting all the cities, return to the starting one



s_a : A – B – C – D – A of length 10

s^* : A – B – D – C – A of length 8

Note: Nearest-neighbor tour may depend on the starting city

Accuracy: $R_A = \infty$ (unbounded above) – make the length of AD arbitrarily large in the above example

Approximation Algorithms

- Find a "good" solution fast
  - sufficient for many applications
  - we often have inaccurate data to start with, so approximation may be as good as optimal solution

Accuracy Ratio

- minimization problems: $r(s_a) = f(s_a)/f(s^*)$
  - $f(s_a)$ = value of objective function for solution given by approximation algorithm
  - $f(s^*)$ = value of objective function for optimal solution
- maximization problems: $r(s_a) = f(s^*)/f(s_a)$
- in either case $r(s_a) >= 1$

Performance Ratio

- If there exists $c \geq 1$, such that $r(s_a) \leq c$ for all instances of a problem, the given algorithm is called a ***c-approximation algorithm***
- The smallest value of c that holds for all instances is called the ***performance ratio***, R_A, of the algorithm,

Unfortunately …

- c-approximation algorithms are good if you can find one
  - if c=1.1, your approximation is never more than 10% worse than optimal
- but … in some cases, no bound for c can be found
  - approx. alg. may be great for 99% of instances, but there are a few really terrible cases

for instance …

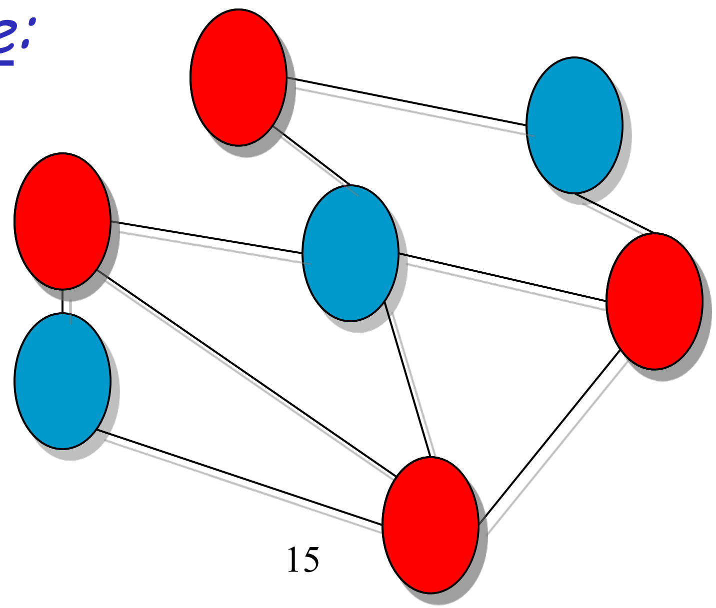
- THEOREM: if $P \neq NP$,
  no c-approximation algorithm for TSP exists

  - it's unlikely that we can find a poly-time approx. algorithm for TSP such that $f(s_a) \leq cf(s^*)$ for all instances

VERTEX-COVER

- Instance: an undirected graph $G=(V,E)$.
- Problem: find a set $C \subseteq V$ of minimal size s.t. for any $(u,v) \in E$, either $u \in C$ or $v \in C$.

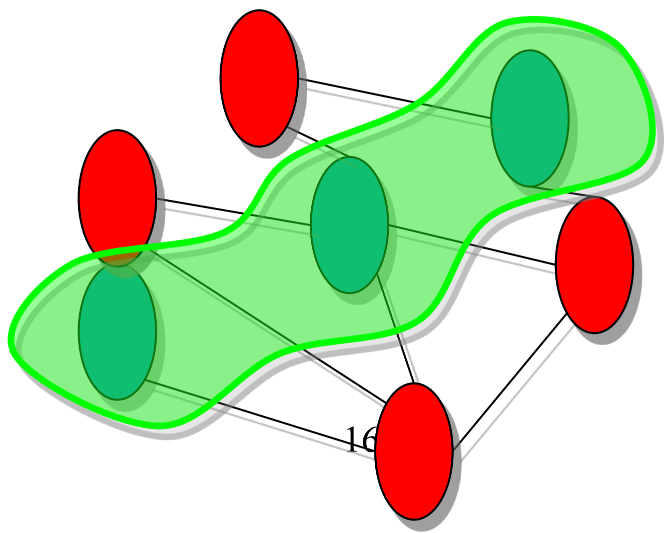
Example:



VERTEX-COVER

Observation: Let $G=(V,E)$ be an undirected graph. The complement $V \setminus C$ of a vertex-cover C is an independent-set of G.

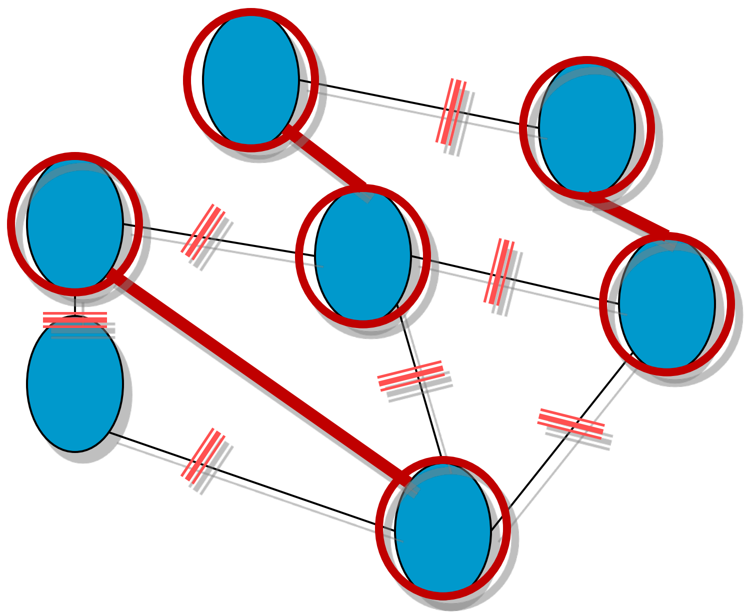
Proof: Two vertices outside a vertex-cover cannot be connected by an edge. ■



VC - Approximation Algorithm

- $C \leftarrow \phi$
- $E' \leftarrow E$
- while $E' \neq \phi$
  - do let (u,v) be an arbitrary edge of E'
  - $C \leftarrow C \cup \{u,v\}$
  - remove from E' every edge incident to either u or v.
- return C.

Demo



Polynomial Time

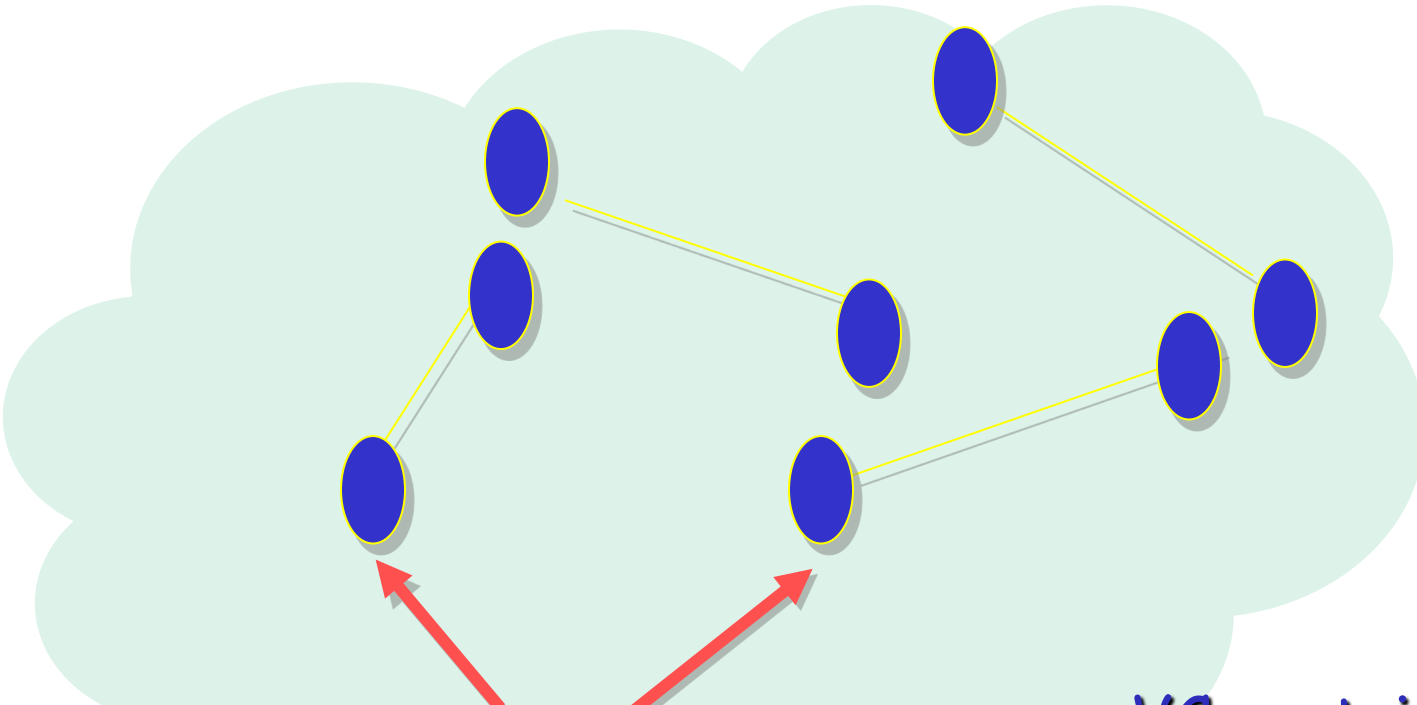
- $C \leftarrow \phi$
- $E' \leftarrow E$
- **while** $E' \neq \phi$ **do**
  - let (u,v) be an arbitrary edge of E'
  - $C \leftarrow C \cup \{u,v\}$
  - remove from E' every edge incident to either u or v — $O(n^2)$
- return C

Correctness

The set of vertices our algorithm returns is clearly a vertex-cover, since we iterate until every edge is covered.

How Good an Approximation is it?

Observe the set of edges our algorithm chooses



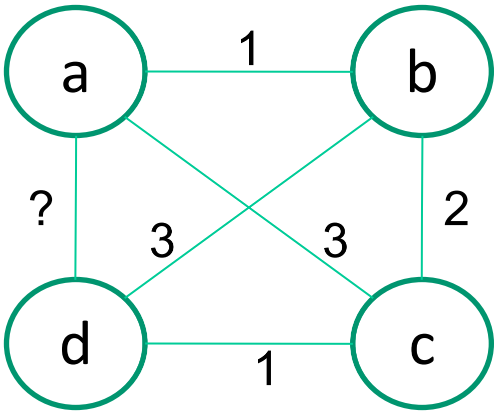
no common vertices! ⇒ any VC contains 1 in each

our VC contains both, hence at most twice as large

OPT$\leq$ $Our\ solution\ \leq 2 * CV$

Worst-case is 2*OPT, so it called 2-Approximation algorithm

TSP: Limited Cases



The problem with this instance is that there may be a very large distance associated with last edge a→d.

Is this a "real-world" instance?

If this graph represents straight-line distances between cities on a map, the length of a→d must be bounded, relative to the other distances (geometry)

If this graph represents costs of airline flights, the cost of a→d could be out of proportion to the other edges

TSP: Euclidean Instances

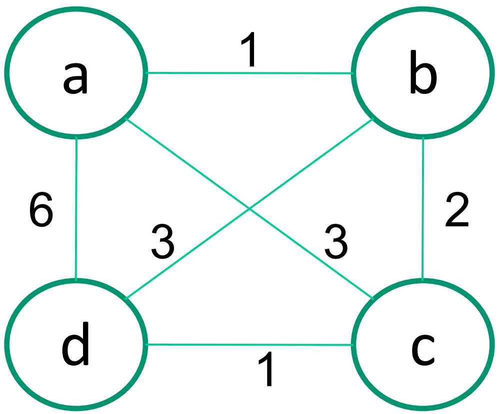
- *Euclidean instances* of the TSP problem obey the natural geometry of a 2D map.
  - triangle inequality: $d[i,j] \le d[i,k] + d[k,j]$
  - symmetry: $d[i,j] = d[j]$
- For Euclidean instances, the nearest neighbor algorithm satisfies:
  $r(s_a) \le \frac{1}{2}(\log_2 n + 1)$, where n = # cities
  - (still not a c-approximation algorithm)

TSP: Multifragment-heuristic

1) sort edges in increasing order
2) Repeat until tour of length n:
   Add next smallest edge, if it doesn't create a vertex of degree 3 and doesn't create a cycle of length < n

More expensive than nearest-neighbor, same accuracy ratio

TSP: Multifragment-heuristic



edge list: ***(a,b),*** (c,d), (b,c), (a,c), (b,d), (a,d)
tour:

edge list: ***(c,d)***, (b,c), (a,c), (b,d), (a,d)
tour: (a,b)

edge list: ***(b,c)***, (a,c), (b,d), (a,d)
tour: (a,b), (c,d)

edge list: ~~(a,c), (b,d),~~ ***(a,d)***
tour: (a,b), (c,d), (b,c)

tour: (a,b), (c,d), (b,c), (a,d) length 10

Twice-Around-the-Tree Algorithm

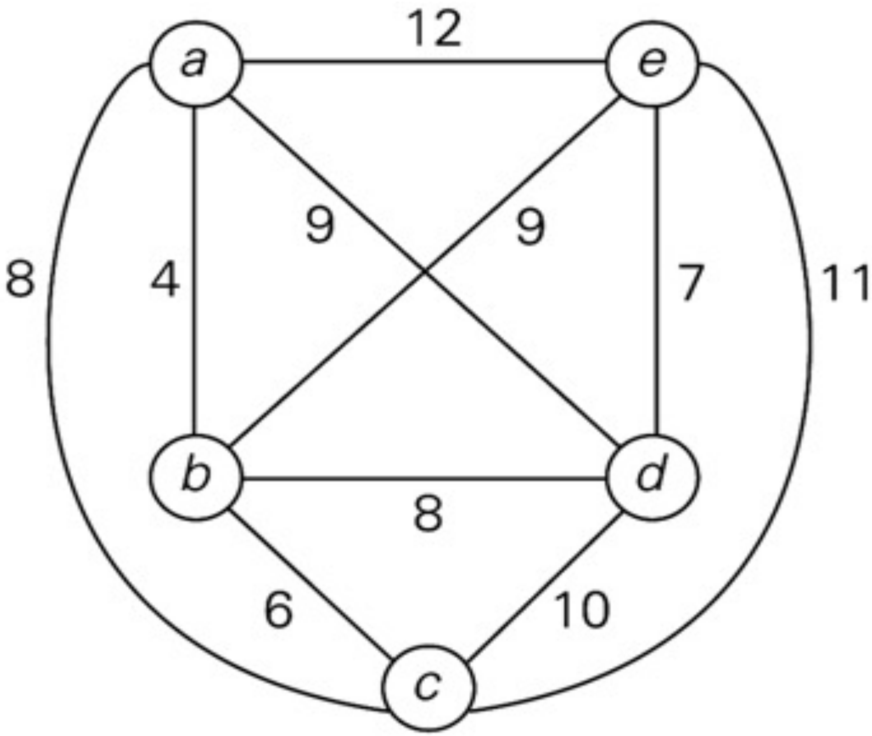
Stage 1: Construct a minimum spanning tree of the graph (e.g., by Prim's or Kruskal's algorithm)

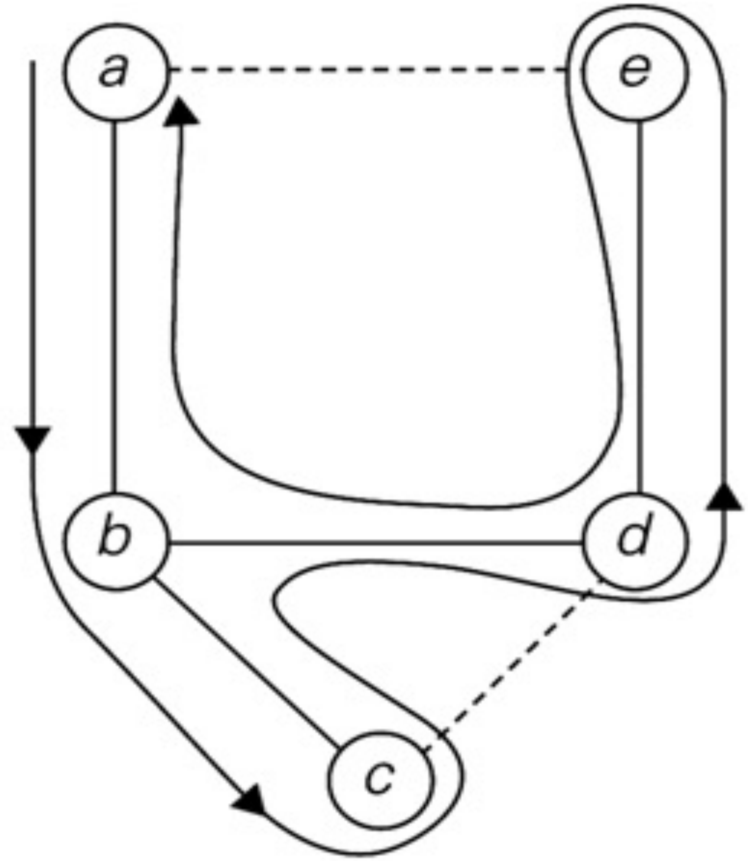
Stage 2: Starting at an arbitrary vertex, create a path that goes twice around the tree and returns to the same vertex

Stage 3: Create a tour from the circuit constructed in Stage 2 by making shortcuts to avoid visiting intermediate vertices more than once

Note: $R_A = \infty$ for general instances, but this algorithm tends to produce better tours than the nearest-neighbor algorithm

TSP: Twice Around the Tree





DFS walk: abcbdedba

eliminate repeated nodes: abcdea

TSP: Twice Around the Tree

$$\text{OPT} \leq MST\ Tour \leq 2 * MST$$

Worst-case is 2*OPT

CSE102
END OF TEACHING

What we have learnt

methodology / problems	Asymptotic idea	Brute force	Divide & Conquer	Dynamic Programming	Greedy	Space/Time	Branch & Bound	Backtracking	Complexity Theory
Efficiency	Big-O								
Sorting		Selection/ Bubble/insertion	Merge-sort			Count sorting			
Searching			Binary-searching						
String		searching		Alignment/LCS		Horspool algorithm			
Graph/Combinatory		DFS/BFS		Floyd's Algorithm/ Assembly-line Knapsack	MST(Prim's/ Kruskal's) Dikstra's For Shortest path		Traveling salesman, Job assignment	n-Queens Sum of subset Hamiltonian Problem	Approximation: TSP problem: Nearst-Neighbor/twice round/fragement algorithm
Complexity									P/NP Circuit-SAT/3-SAT

How will it be assessed

- **Two assignments (20% of the final mark)**

    1. Assignment 1 (week 6 - week 7)   (10%)
    2. Assignment 2 (week 11 – week 12) (10%)

- **Final Examination (80% of the final mark)**

  written examination: 80% MCQ's + 20% Problem Solving