Unit V

Approximation Algorithms

Approximation Approach

Apply a fast (i.e., a polynomial-time) approximation algorithm to get a solution that is not necessarily optimal but hopefully close to it

Accuracy measures:

accuracy ratio of an approximate solution s_a

$r(s_a) = f(s_a) / f(s^*)$ for minimization problems

$r(s_a) = f(s^*) / f(s_a)$ for maximization problems

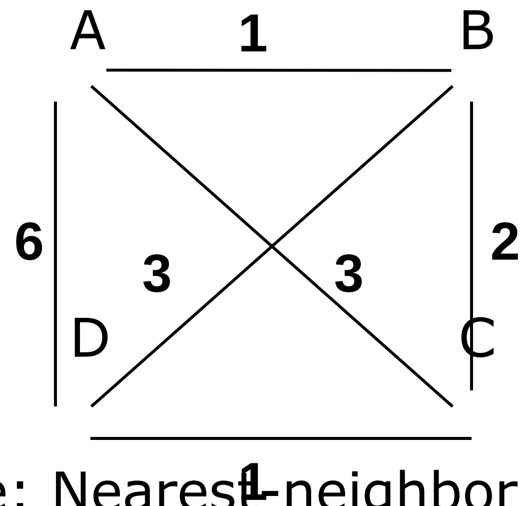
where $f(s_a)$ and $f(s^*)$ are values of the objective function f for the approximate solution s_a and actual optimal solution s^*

performance ratio of the algorithm A

the lowest upper bound of $r(s_a)$ on all instances

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

Multifragment-Heuristic Algorithm

Stage 1: Sort the edges in nondecreasing order of weights. Initialize the set of tour edges to be constructed to empty set

Stage 2: Add next edge on the sorted list to the tour, skipping those whose addition would've created a vertex of degree 3 or a cycle of length less than *n*. Repeat this step until a tour of length *n* is obtained

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

Twice-Around-the-Tree Algorithm

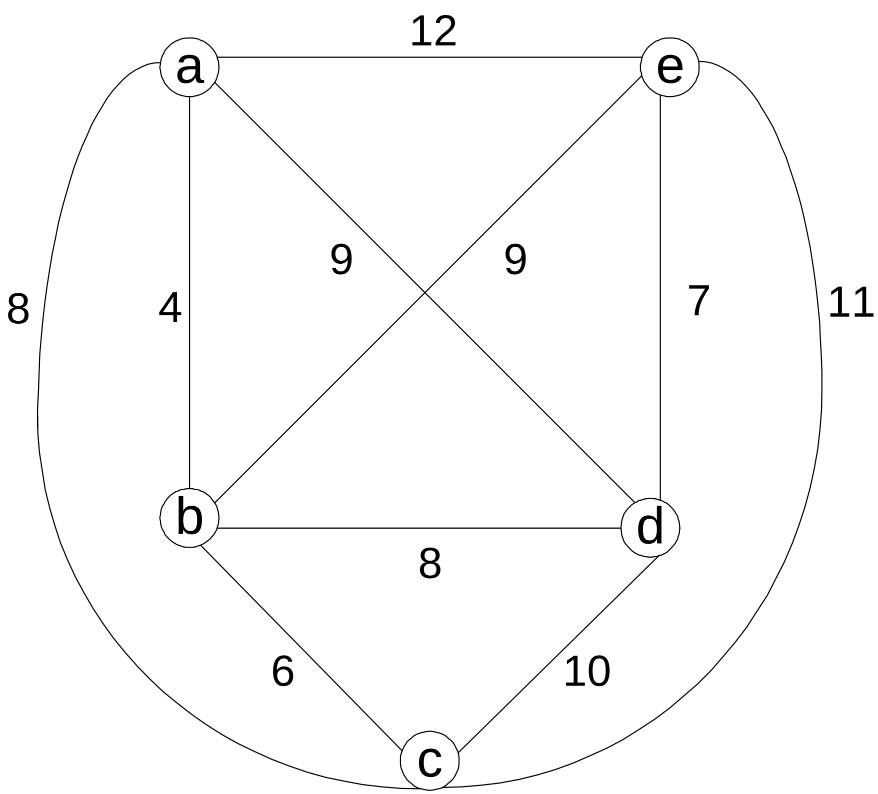
Stage 1: Construct a minimum spanning tree of the graph (e.g., by Prim's or Kruskal's algorithm)

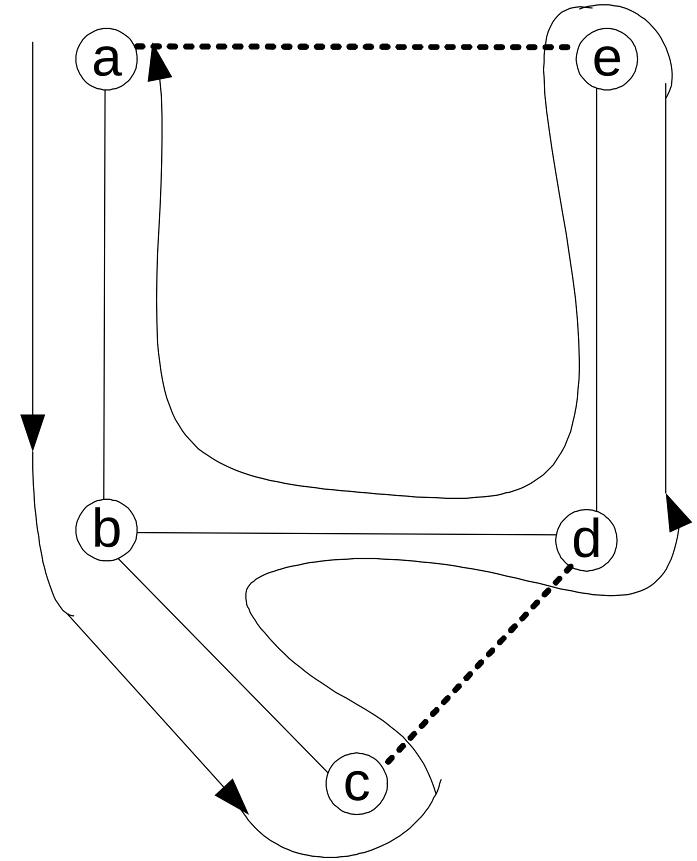
Stage 2: Starting at an arbitrary vertex, create a path that goes twice around the tree and returns to the same vertex

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

Example





Walk: a – b – c – b – d – e – d – b – a

Tour: a – b – c – d – e – a

Christofides Algorithm

Stage 1: Construct a minimum spanning tree of the graph

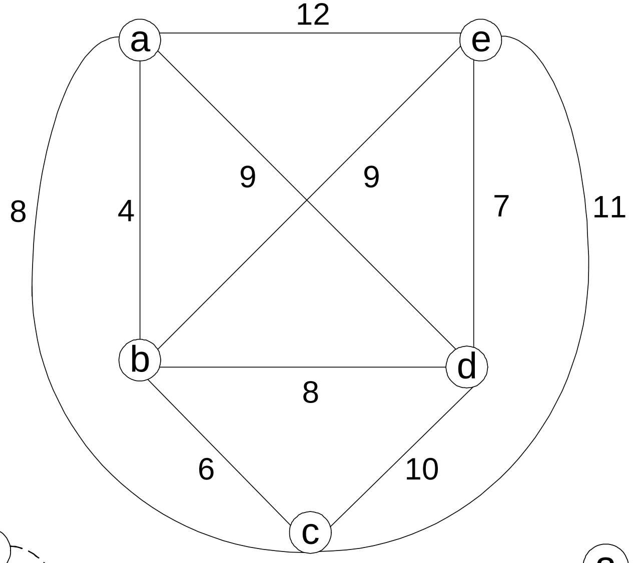
Stage 2: Add edges of a minimum-weight matching of all the odd vertices in the minimum spanning tree

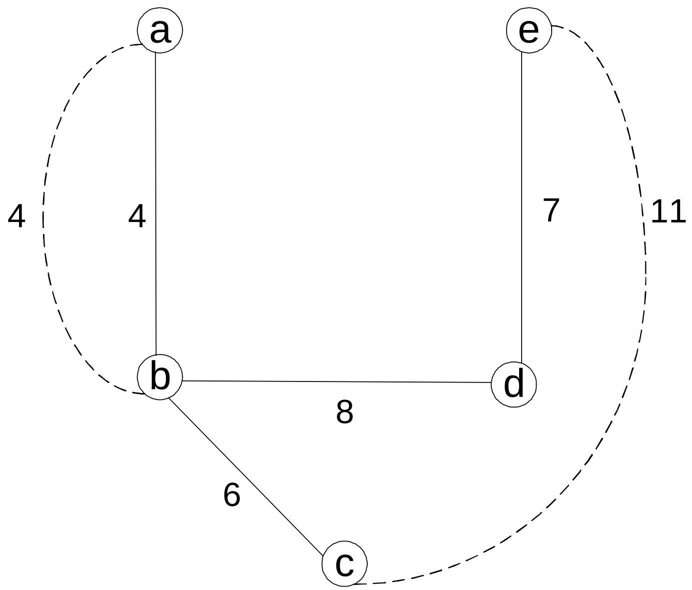
Stage 3: Find an Eulerian circuit of the multigraph obtained in Stage 2

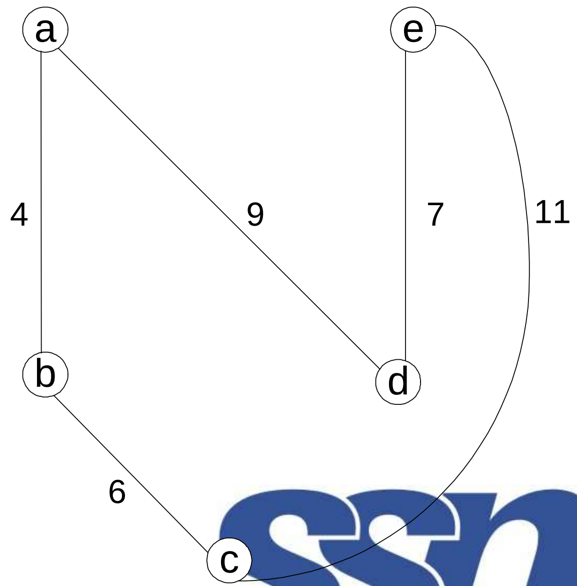
Stage 3: Create a tour from the path constructed in Stage 2 by making shortcuts to avoid visiting intermediate vertices more than once

$R_A = \infty$ for general instances, but it tends to produce better tours than the twice-around-the-minimum-tree alg.

Example: Christofides Algorithm







Euclidean Instances

Theorem   If $P \neq NP$, there exists no approximation algorithm for TSP with a finite performance ratio.

Definition An instance of TSP is called *Euclidean*, if its distances satisfy two conditions:

1. *symmetry* $d[i, j] = d[j, i]$ for any pair of cities i and j
2. *triangle inequality* $d[i, j] \leq d[i, k] + d[k, j]$ for any cities i, j, k

For Euclidean instances:

approx. tour length / optimal tour length $\leq 0.5(\lceil \log_2 n \rceil + 1)$

for nearest neighbor and multifragment heuristic;

approx. tour length / optimal tour length ≤ 2

for twice-around-the-tree;

approx. tour length / optimal tour length ≤ 1.5

for Christofides

Local Search Heuristics for TSP

Start with some initial tour (e.g., nearest neighbor). On each iteration, explore the current tour's neighborhood by exchanging a few edges in it. If the new tour is shorter, make it the current tour; otherwise consider another edge change. If no change yields a shorter tour, the current tour is returned as the output.

Example of a 2-change

Greedy Algorithm for Knapsack Problem

Step 1: Order the items in decreasing order of relative values: $v_1/w_1 \geq \ldots \geq v_n/w_n$

Step 2: Select the items in this order skipping those that don't fit into the knapsack

Example: The knapsack's capacity is 16

item	weight	value	v/w
1	2	\$40	20
2	5	\$30	6
3	10	\$50	5
4	5	\$10	2

Accuracy

- R_A is unbounded (e.g., $n = 2$, $C = m$, $w_1=1$, $v_1=2$, $w_2=m$, $v_2=m$)
- yields exact solutions for the continuous version

Approximation Scheme for Knapsack Problem

Step 1: Order the items in decreasing order of relative values:
$v_1/w_1 \geq \ldots \geq v_n/w_n$

Step 2: For a given integer parameter k, $0 \leq k \leq n$, generate all subsets of k items or less and for each of those that fit the knapsack, add the remaining items in decreasing order of their value to weight ratios

Step 3: Find the most valuable subset among the subsets generated in Step 2 and return it as the algorithm's output

Knapsack Problem

- Accuracy: $f(s^*) / f(s_a) \leq 1 + 1/k$ for any instance of size n
- Time efficiency: $O(kn^{k+1})$
- There are fully polynomial schemes: algorithms with polynomial running time as functions of both n and k

Summary

- Approximation algorithms are often used to find approximate solutions to difficult problems of combinatorial optimization.
- The performance ratio is the principal metric for measuring the accuracy of such approximation algorithms.
- Approximation Scheme for Knapsack Problem
- Approximation Scheme for traveling salesman problem

Test your understanding

a. Apply the nearest-neighbor algorithm to the instance defined by the intercity distance matrix below. Start the algorithm at the first city, assuming that the cities are numbered from 1 to 5.

$$\begin{bmatrix} 0 & 14 & 4 & 10 & \infty \\ 14 & 0 & 5 & 8 & 7 \\ 4 & 5 & 0 & 9 & 16 \\ 10 & 8 & 9 & 0 & 32 \\ \infty & 7 & 16 & 32 & 0 \end{bmatrix}$$

b. Compute the accuracy ratio of this approximate solution.