Approximation Algorithms

Introduction

- Many problems of practical significance are NP-complete and can be solved in following ways:

1. If the actual inputs are small, an algorithm with exponential running time may be perfectly satisfactory.
2. Identify important special cases that can be solved in polynomial time.
3. Approximation Algorithm
   - Algorithms that runs in polynomial time and always produce a solution close to the optimal.

Performance Ratios

- An algorithm for a problem has an approximation ratio of $\rho(n)$ if, for any input of size n, the cost C of the solution produced by the algorithm is within a factor of $\rho(n)$ of the cost C^* of an optimal solution:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$$

- If an algorithm achieves an approximation ratio of $\rho(n)$, it is called as a $\rho(n)$ -approximation algorithm.
- $\rho(n) \geq 1$
  - 1-approximation algorithm produces an optimal solution.

Contd…

- Definitions of approximation ratio and $\rho(n)$-approximation algorithm can be applied to both minimization and maximization problems.
- For a maximization problem,
  - $0 < C \le C^*$, and the ratio C^*/C gives the factor by which the cost of an optimal solution is larger than the cost of the approximate solution.
- For a minimization problem,
  - $0 < C^* \le C$, and the ratio C/C^* gives the factor by which the cost of the approximate solution is larger than the cost of an optimal solution.

Contd…

- Inputs to (1 + ε)-approximation algorithm
  - An instance of the problem.
  - A value $\varepsilon > 0$.
- Polynomial-time approximation scheme (PTAS)
  - Runs in time polynomial in size n of its input instance.
  - Running time increases very rapidly as ε decreases.
  - Example: $O(n^{2/\varepsilon})$
- Fully polynomial-time approximation scheme (FPTAS)
  - Runs in time polynomial in both 1/ ε and size n of its input instance.
  - Running time increases by a constant-factor with any constant-factor decrease in ε.
  - Example: $O((1/\varepsilon)^2 n^3)$

Traveling Salesman Problem

Traveling Salesman Problem

- Given a complete undirected graph $G = (V, E)$ with a nonnegative integer cost $c(u,v)$ associated with each edge $(u,v) \in E$, find a hamiltonian cycle (a tour) of G with minimum cost.
- Consider two cases:
  - with and without triangle inequality.
  - c satisfies triangle inequality, if for all vertices u, v, $w \in V$, $$c(u,w) \leq c(u,v) + c(v,w)$$
- Finding an optimal solution is NP-complete in both cases.

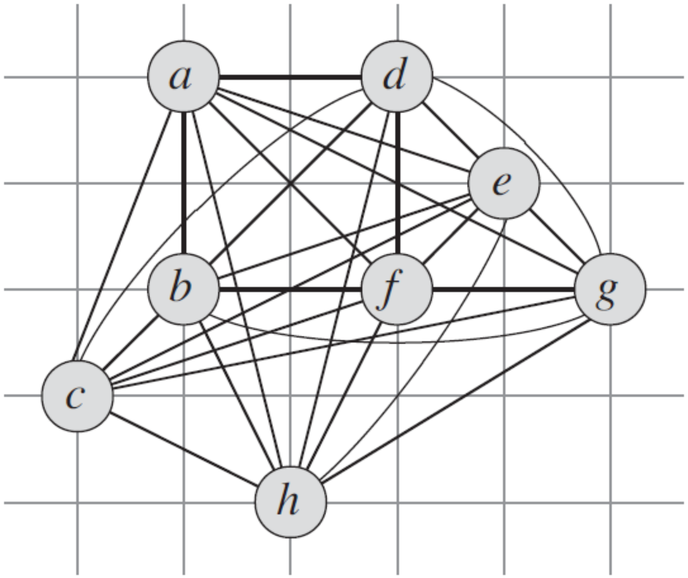
TSP with Triangle Inequality

- Compute a minimum spanning tree, whose weight gives a lower bound on the length of an optimal traveling-salesman tour.
- Use the minimum spanning tree to create a tour whose cost is no more than twice that of the minimum spanning tree's weight, as long as the cost function satisfies the triangle inequality.
- Assuming,
  - G – a complete undirected graph.
  - c – a cost function satisfying the triangle inequality.

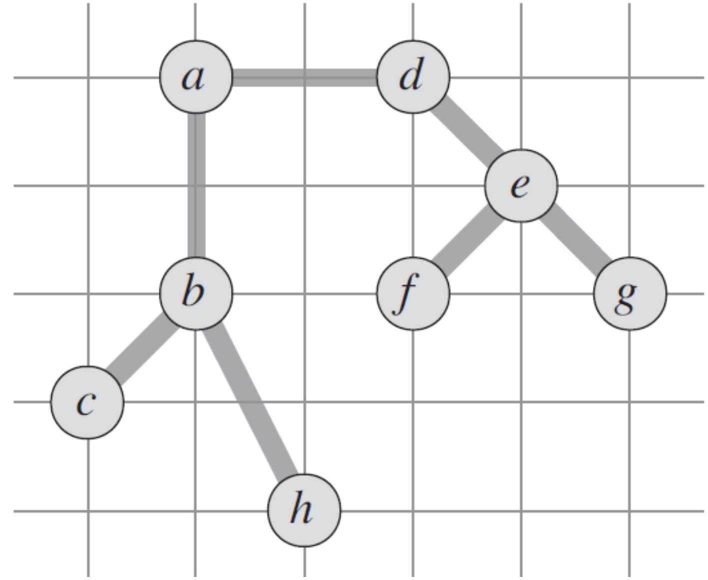
APPROX-TSP-TOUR(G,c)

1. Select a vertex $r \in G.V$ to be a "root" vertex
2. Compute a minimum spanning tree T for G from root r using MST-PRIM(G,c,r)
3. Let H be a list of vertices, ordered according to when they are first visited in a preorder tree walk of T
4. **return** the hamiltonian cycle H

Example

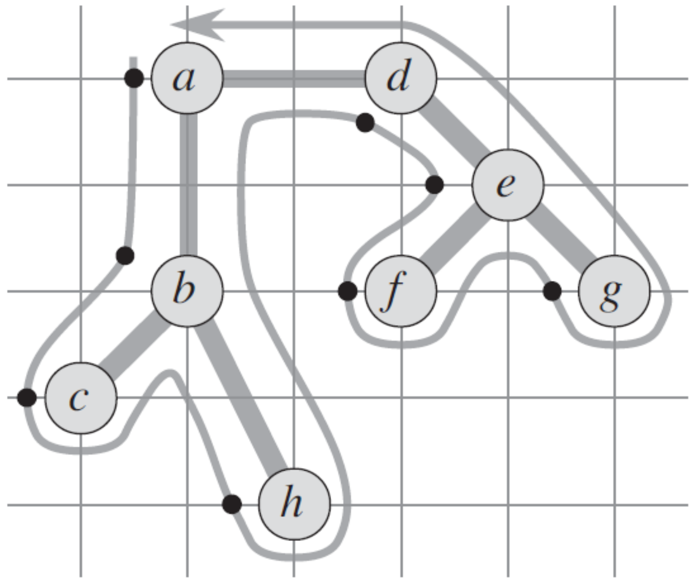


1. A complete undirected graph. Vertices lie on intersections of integer grid lines. For example, *f* is one unit to the right and two units up from *h*. The cost function between two points is the ordinary Euclidean distance.

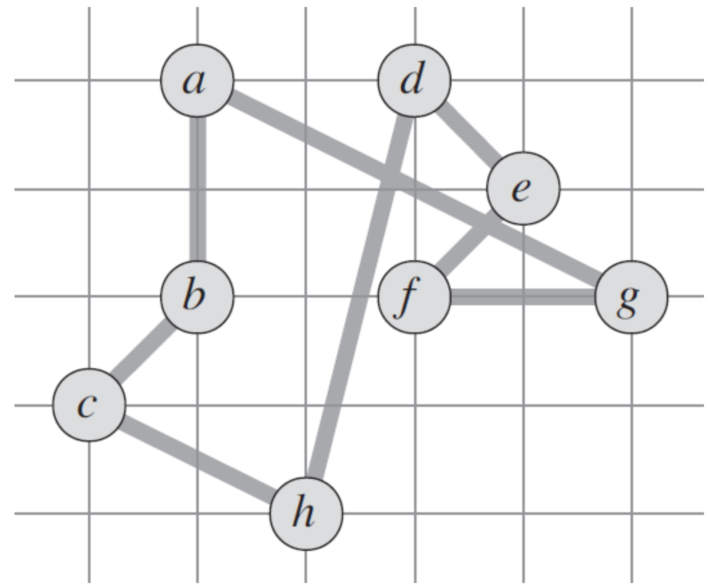


2. A minimum spanning tree *T* of the complete graph, as computed by MST-PRIM. Vertex *a* is the root vertex. Only edges in the minimum spanning tree are shown. The vertices happen to be labeled in such a way that they are added to the main tree by MST-PRIM in alphabetical order.

Contd…



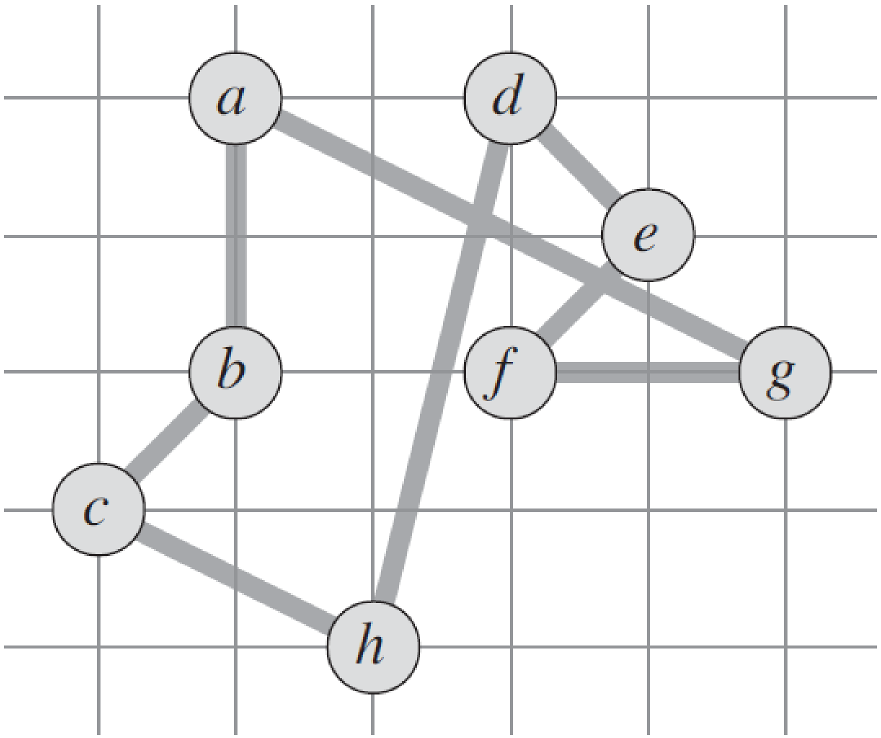
3. A walk of *T*, starting at *a*. A full walk of the tree visits the vertices in the order *a b c b h b a d e f e g e d a*. A preorder walk of *T* lists a vertex just when it is first encountered, as indicated by the dot next to each vertex, yielding the ordering *a b c h d e f g*.



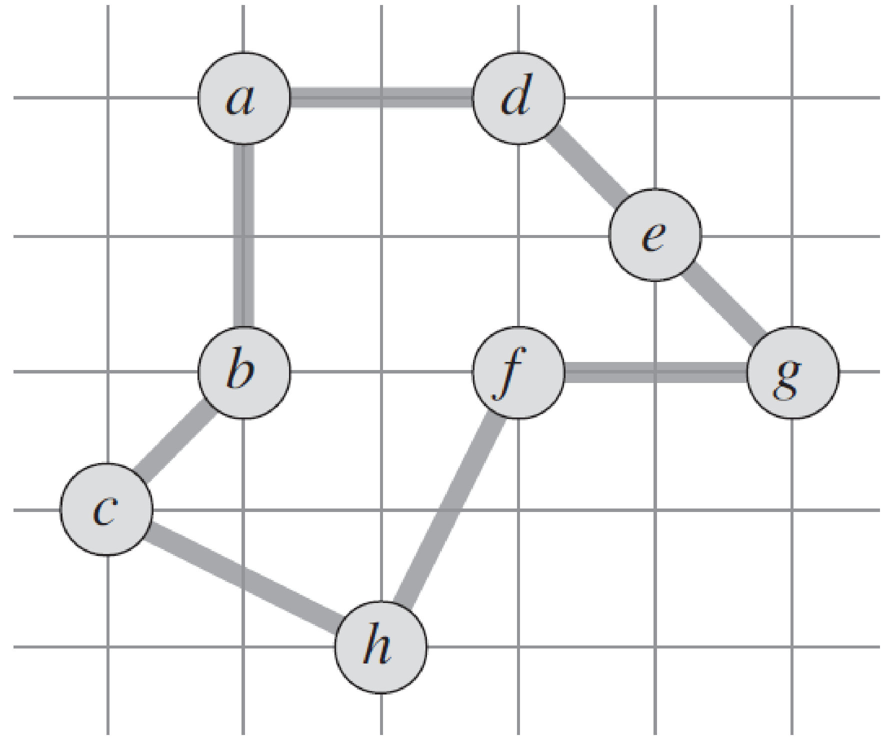
4. A tour obtained by visiting the vertices in the order given by the preorder walk, which is the tour *H* returned by APPROX-TSP-TOUR. Its total cost is approximately 19.074.

Contd…

It is known that APPROX-TSP-TOUR is a polynomial-time 2-approximation algorithm, i.e. 19.074 <= 2 * 14.715. The relation is clearly maintained for the considered example.



Tour H obtained using APPROX-TSP-TOUR. Cost = 19.074



An optimal tour H* for the original complete graph. Cost = 14.715

Theorem:

APPROX-TSP-TOUR is a polynomial-time 2-approximation algorithm for the traveling-salesman problem with the triangle inequality.

- Polynomial running time obvious, simple MSTPRIM takes $\theta(|V|^2)$, computing pre-order walk takes no longer.
- Correctness obvious, pre-order walk is always a tour.

Contd…

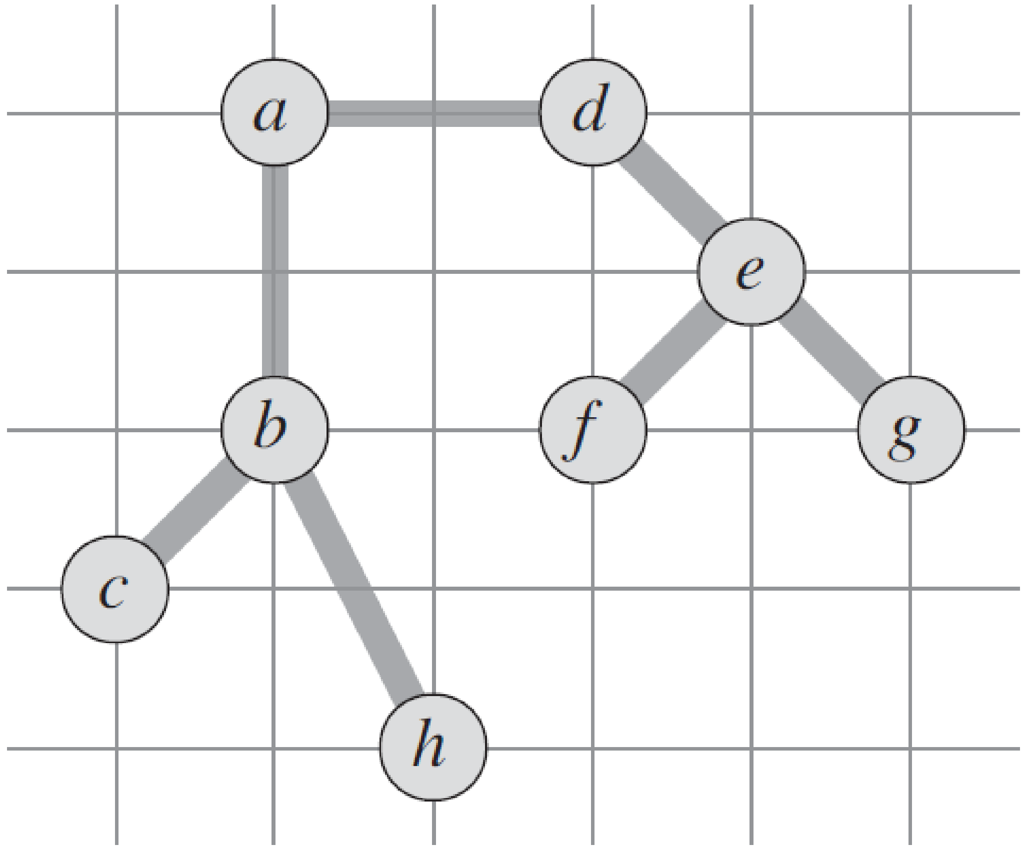
- Let H^* denotes an optimal tour for given set of vertices.
- Deleting any edge from H^* gives a spanning tree.
- Thus, weight of minimum spanning tree (T) is lower bound on cost of optimal tour:

$$c(T) \leq c(H^*)$$

- A full walk of T lists vertices when they are first visited, and also when they are returned to, after visiting a subtree.
- The full walk (W) of our example gives the order

$$a, b, c, b, h, b, a, d, e, f, e, g, e, d, a$$

Contd…



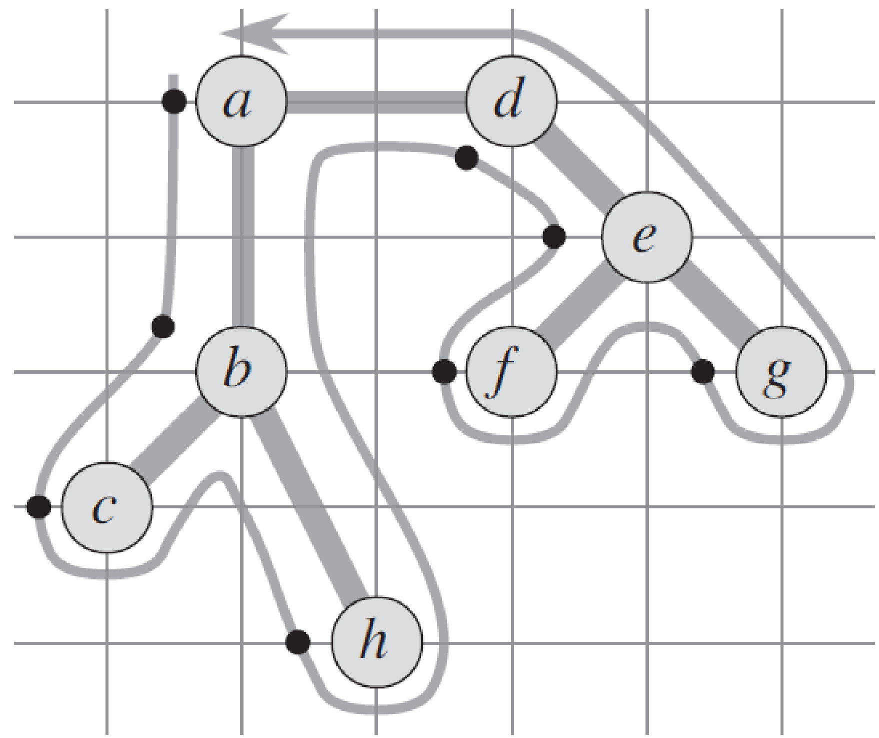
$a, b, c, b, h, b, a, d, e, f, e, g, e, d, a$

Contd…

- Full walk W traverses every edge exactly twice, thus
$$c(W) = 2c(T)$$
- Together with $c(T) \leq c(H^*)$, this gives
$$c(W) = 2c(T) \leq 2c(H^*)$$
- Find a connection between cost of W and cost of "our" tour.
  – Problem: W is in general not a proper tour, since vertices may be visited more than once.
  – But: using the triangle inequality, we can delete a visit to any vertex from W and cost does not increase.
  – Deleting a vertex v from walk W between visits to u and w means going from u directly to w, without visiting v.

Contd…

- We can consecutively remove all multiple visits to any vertex.



- Example:
  - full walk $a, b, c, b, h, b, a, d, e, f, e, g, e, d, a$
  - becomes a, b, c, h, d, e, f, g

Contd…

- This ordering (with multiple visits deleted) is identical to that obtained by pre-order walk of T (with each vertex visited only once).
- It certainly is a Hamiltonian cycle. Let's call it H.
- H is just what is computed by APPROX-TSP-TOUR.
- H is obtained by deleting vertices from W, thus

$$c(H) \le c(W)$$

- Conclusion:

$$c(H) \le c(W) \le 2c(H^*)$$

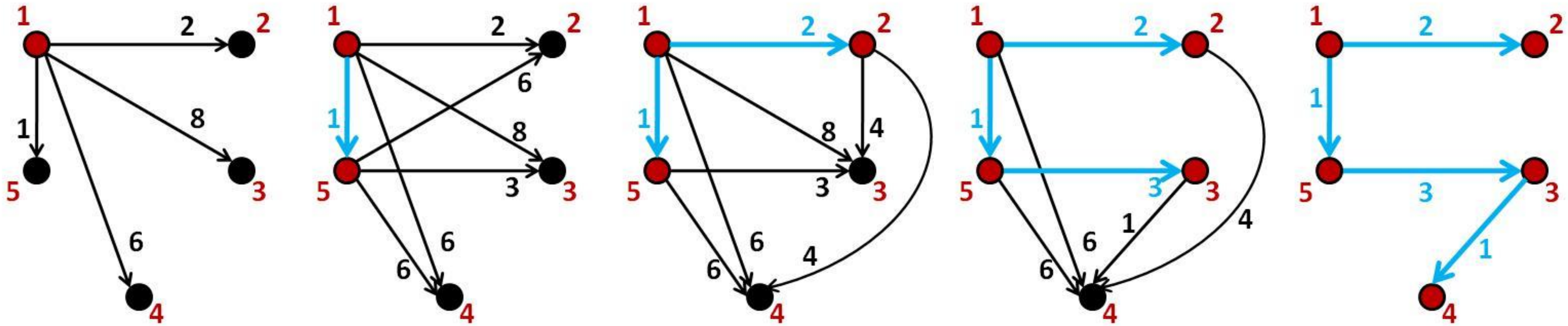
Contd…

- Although factor 2 looks nice, there are better algorithms.
- There's a 3/2 approximation algorithm by Christofides(with triangle inequality).
- In general, TSP cost function c does not satisfy triangle inequality.
- Theorem:
  – If P ≠ NP, then for any constant $\rho \geq 1$, there is no polynomial time approximation algorithm with approximation ratio ρ for the general traveling-salesman problem.

Example

	1	2	3	4	5
1	0	2	8	6	1
2	1	0	4	4	2
3	5	3	0	1	5
4	4	7	2	0	1
5	2	6	3	6	0

- Let the starting vertex be '1'.
- Computing MST using Prims.



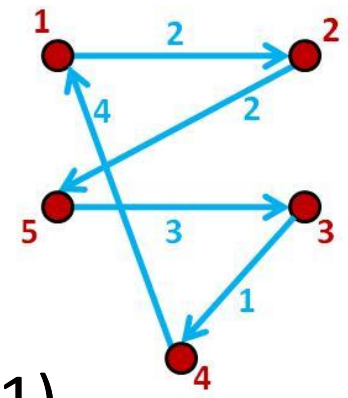
(i) (ii) (iii) (iv) (v)

- Preorder traversal of MST
  - 1 2 5 3 4, or
  - 1 5 3 4 2

Contd…

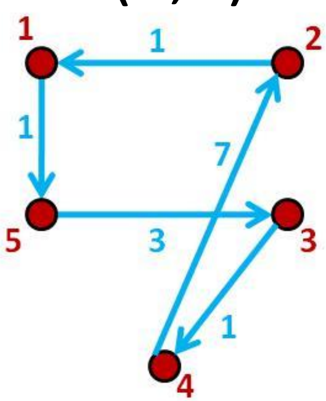
	1	2	3	4	5
1	0	2	8	6	1
2	1	0	4	4	2
3	5	3	0	1	5
4	4	7	2	0	1
5	2	6	3	6	0

- Computing tour cost.
  - 1 2 5 3 4
  - Cost = c(1,2) + c(2,5) + c(5,3) + c(3,4) + c(4,1)
    = 2 + 2 + 3 + 1 + 4 = 12



OR

  - 1 5 3 4 2
  - Cost = c(1,5) + c(5,3) + c(3,4) + c(4,2) + c(2,1)
    = 1 + 3 + 1 + 7 + 1 = 13



Proving 2-approximation algorithm

- To prove,
  - The cost of optimal tour is required.
- To compute optimal tour use Held-Karp algorithm, also called Bellman–Held–Karp algorithm.
- It is a dynamic programming algorithm to solve the Traveling Salesman Problem (TSP).
- In worst case,
  - Time complexity is $O(2^n n^2)$
  - Space complexity is $O(2^n n)$

Held-Karp Algorithm

- Let,
  - N be the number of cities i.e. $\{1, 2, \ldots, N\}$.
  - 1 is the 'start' city.
  - d_{ij} is the distance between city i and city j.
  - $S \subseteq \{2, \ldots, N\}$ of cities and,
  - For $c \in S$, let $D(S, c)$ be the minimum distance, starting at city 1, visiting all cities in S and finishing at city c.

$$D(S,c) = \min_{x \in S} (D(S-c, x) + d_{xc})$$

  - k be the size of S.

Example (Same as in slide 21)

- $k = 0, S = \emptyset$
  - $D(\emptyset,2) = 2, \quad p(\emptyset,2) = 1$
  - $D(\emptyset,3) = 8, \quad p(\emptyset,3) = 1$
  - $D(\emptyset,4) = 6, \quad p(\emptyset,4) = 1$
  - $D(\emptyset,5) = 1, \quad p(\emptyset,5) = 1$

	1	2	3	4	5
1	0	2	8	6	1
2	1	0	4	4	2
3	5	3	0	1	5
4	4	7	2	0	1
5	2	6	3	6	0

- $k = 1, S = \{2\}$
  - $D(\{2\},3) = D(\emptyset,2) + d_{23} = 2 + 4 = 6, \quad p(\{2\},3) = 2$
  - $D(\{2\},4) = D(\emptyset,2) + d_{24} = 2 + 4 = 6, \quad p(\{2\},4) = 2$
  - $D(\{2\},5) = D(\emptyset,2) + d_{25} = 2 + 2 = 4, \quad p(\{2\},5) = 2$

Contd…

	1	2	3	4	5
1	0	2	8	6	1
2	1	0	4	4	2
3	5	3	0	1	5
4	4	7	2	0	1
5	2	6	3	6	0

- $k = 1$, $S = \{3\}$
  - $D(\{3\},2) = D(\emptyset,3) + d_{32} = 8 + 3 = 11$, $p(\{3\},2) = 3$
  - $D(\{3\},4) = D(\emptyset,3) + d_{34} = 8 + 1 = 9$, $p(\{3\},4) = 3$
  - $D(\{3\},5) = D(\emptyset,3) + d_{35} = 8 + 5 = 13$, $p(\{3\},5) = 3$

- $k = 1$, $S = \{4\}$
  - $D(\{4\},2) = D(\emptyset,4) + d_{42} = 6 + 7 = 13$, $p(\{4\},2) = 4$
  - $D(\{4\},3) = D(\emptyset,4) + d_{43} = 6 + 2 = 8$, $p(\{4\},3) = 4$
  - $D(\{4\},5) = D(\emptyset,4) + d_{45} = 6 + 1 = 7$, $p(\{4\},5) = 4$

- $k = 1$, $S = \{5\}$
  - $D(\{5\},2) = D(\emptyset,5) + d_{52} = 1 + 6 = 7$, $p(\{5\},2) = 5$
  - $D(\{5\},3) = D(\emptyset,5) + d_{53} = 1 + 3 = 4$, $p(\{5\},3) = 5$
  - $D(\{5\},4) = D(\emptyset,5) + d_{54} = 1 + 6 = 7$, $p(\{5\},4) = 5$

Contd…

	1	2	3	4	5
1	0	2	8	6	1
2	1	0	4	4	2
3	5	3	0	1	5
4	4	7	2	0	1
5	2	6	3	6	0

- $k = 2, S = \{2,3\}$
  - $D(\{2,3\},4) = \min(\boldsymbol{D(\{2\},3) + d_{34}}, D(\{3\},2) + d_{24})$
    $= \min(\underline{\mathbf{6 + 1}}, 11 + 4) = 7, \quad p(\{2,3\},4) = 3$
  - $D(\{2,3\},5) = \min(\boldsymbol{D(\{2\},3) + d_{35}}, D(\{3\},2) + d_{25})$
    $= \min(\underline{\mathbf{6 + 5}}, 11 + 2) = 11, \quad p(\{2,3\},5) = 3$

- $k = 2, S = \{2,4\}$
  - $D(\{2,4\},3) = \min(\boldsymbol{D(\{2\},4) + d_{43}}, D(\{4\},2) + d_{23})$
    $= \min(\underline{\mathbf{6 + 2}}, 13 + 4) = 8, \quad p(\{2,4\},3) = 4$
  - $D(\{2,4\},5) = \min(\boldsymbol{D(\{2\},4) + d_{45}}, D(\{4\},2) + d_{25})$
    $= \min(\underline{\mathbf{6 + 1}}, 13 + 2) = 7, \quad p(\{2,3\},5) = 4$

- $k = 2, S = \{2,5\}$
  - $D(\{2,5\},3) = \min(\boldsymbol{D(\{2\},5) + d_{53}}, D(\{5\},2) + d_{23})$
    $= \min(\underline{\mathbf{4 + 3}}, 7 + 4) = 7, \quad p(\{2,5\},3) = 5$
  - $D(\{2,5\},4) = \min(\boldsymbol{D(\{2\},5) + d_{54}}, D(\{5\},2) + d_{24})$
    $= \min(\underline{\mathbf{4 + 6}}, 7 + 4) = 10, \quad p(\{2,5\},4) = 5$

Contd…

	1	2	3	4	5
1	0	2	8	6	1
2	1	0	4	4	2
3	5	3	0	1	5
4	4	7	2	0	1
5	2	6	3	6	0

- $k = 2, S = \{3,4\}$
  - $D(\{3,4\},2) = \min(D(\{3\},4) + d_{42}, \boldsymbol{D(\{4\},3) + d_{32}})$
    $= \min(9 + 7, \underline{\mathbf{8 + 3}}) = 11,$ $p(\{3,4\},2) = 3$
  - $D(\{3,4\},5) = \min(\boldsymbol{D(\{3\},4) + d_{45}}, D(\{4\},3) + d_{35})$
    $= \min(\underline{\mathbf{9 + 1}}, 8 + 5) = 10,$ $p(\{3,4\},5) = 4$

- $k = 2, S = \{3,5\}$
  - $D(\{3,5\},2) = \min(D(\{3\},5) + d_{52}, \boldsymbol{D(\{5\},3) + d_{32}})$
    $= \min(13 + 6, \underline{\mathbf{4 + 3}}) = 7,$ $p(\{3,5\},2) = 3$
  - $D(\{3,5\},4) = \min(D(\{3\},5) + d_{54}, \boldsymbol{D(\{5\},3) + d_{34}})$
    $= \min(13 + 6, \underline{\mathbf{4 + 1}}) = 5,$ $p(\{3,5\},4) = 3$

- $k = 2, S = \{4,5\}$
  - $D(\{4,5\},2) = \min(\boldsymbol{D(\{4\},5) + d_{52}}, D(\{5\},4) + d_{42})$
    $= \min(\underline{\mathbf{7 + 6}}, 7 + 7) = 13,$ $p(\{4,5\},2) = 5$
  - $D(\{4,5\},3) = \min(D(\{4\},5) + d_{53}, \boldsymbol{D(\{5\},4) + d_{43}})$
    $= \min(7 + 3, \underline{\mathbf{7 + 2}}) = 9,$ $p(\{4,5\},3) = 4$

Contd…

	1	2	3	4	5
1	0	2	8	6	1
2	1	0	4	4	2
3	5	3	0	1	5
4	4	7	2	0	1
5	2	6	3	6	0

- $k = 3$, $S = \{2,3,4\}$
  - $D(\{2,3,4\},5) = \min(\boldsymbol{D(\{2,3\},4) + d_{45}}, D(\{2,4\},3) + d_{35}, D(\{3,4\},2) + d_{25})$
    $= \min(\underline{\mathbf{7 + 1}}, 8 + 5, 11 + 2) = 8,$ $p(\{2,3,4\},5) = 4$

- $k = 3$, $S = \{2,3,5\}$
  - $D(\{2,3,5\},4) = \min(D(\{2,3\},5) + d_{54}, \boldsymbol{D(\{2,5\},3) + d_{34}}, D(\{3,5\},2) + d_{24})$
    $= \min(11 + 6, \underline{\mathbf{7 + 1}}, 7 + 4) = 8,$ $p(\{2,3,5\},4) = 3$

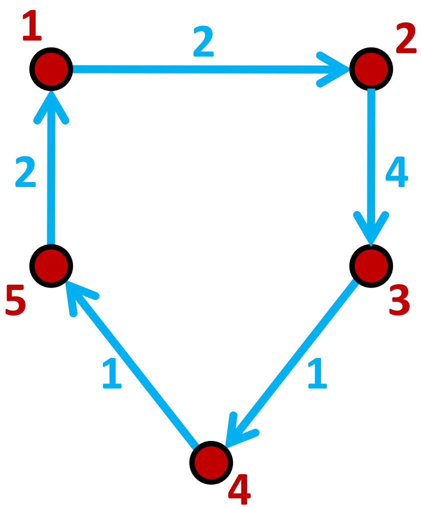
- $k = 3$, $S = \{2,4,5\}$
  - $D(\{2,4,5\},3) = \min(\boldsymbol{D(\{2,4\},5) + d_{53}}, D(\{2,5\},4) + d_{43}, D(\{4,5\},2) + d_{23})$
    $= \min(\underline{\mathbf{7 + 3}}, 10 + 2, 13 + 4) = 10,$ $p(\{2,4,5\},3) = 5$

- $k = 3$, $S = \{3,4,5\}$
  - $D(\{3,4,5\},2) = \min(D(\{3,4\},5) + d_{52}, \boldsymbol{D(\{3,5\},4) + d_{42}}, D(\{4,5\},3) + d_{32})$
    $= \min(10 + 6, \underline{\mathbf{5 + 7}}, 9 + 3) = 12,$ $p(\{3,4,5\},2) = 4$

Contd...

	1	2	3	4	5
1	0	2	8	6	1
2	1	0	4	4	2
3	5	3	0	1	5
4	4	7	2	0	1
5	2	6	3	6	0

- $k = 4$, $S = \{2,3,4,5\}$
  - $D(\{2,3,4,5\},1) = \min(\boldsymbol{D(\{2,3,4\},5) + d_{51}}, D(\{2,3,5\},4) + d_{41}, D(\{2,4,5\},3) + d_{31}, D(\{3,4,5\},2) + d_{21})$
    $= \min(\underline{\mathbf{8 + 2}}, 8 + 4, 10 + 5, 12 + 1) = 10$, $p(\{2,3,4,5\},1) = 5$



- $p(\{2,3,4,5\},1) = 5$
- $p(\{2,3,4\},5) = 4$
- $p(\{2,3\},4) = 3$
- $p(\{2\},3) = 2$
- $p(\{\varnothing\},2) = 1$
- Optimal tour is 1 2 3 4 5 1 with cost 10.
- Cost of tour obtained using 2-approximation algorithm is 12 or 13. Both these cost values are less than twice of optimal tour cost, i.e. $2 \times 10 = 20$.
- Hence proved.

Christofides’ Algorithm (3/2 approximation algorithm)

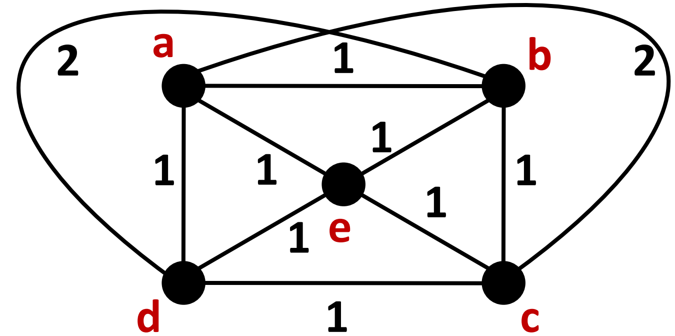
1. Find a minimum spanning tree of G.

2. Compute a minimum cost perfect matching M on the set of odd-degree vertices of MST. Add M to MST to obtain an Eulerian graph.

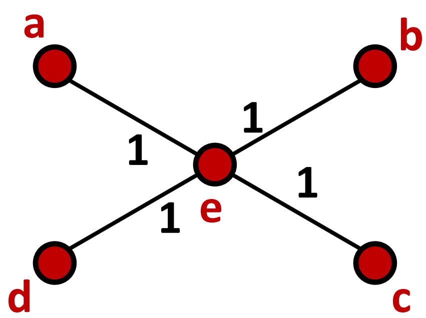
3. Find a Eulerian tour J of the Eulearian graph.

4. Convert J to a tour T by going through the vertices in the same order of T, skipping vertices that were already visited.

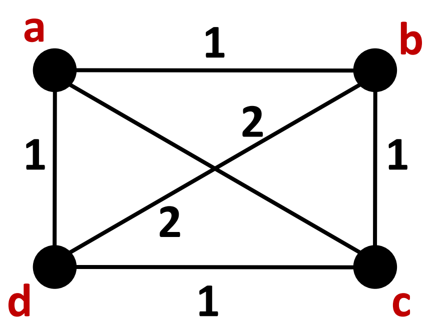
Example – 1



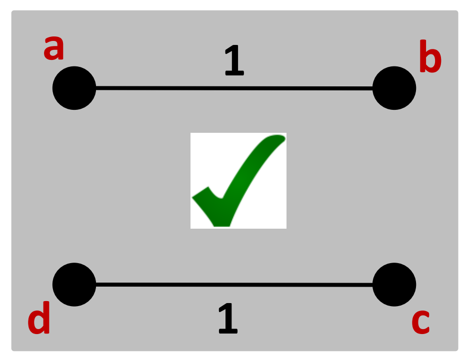
1. Compute MST.

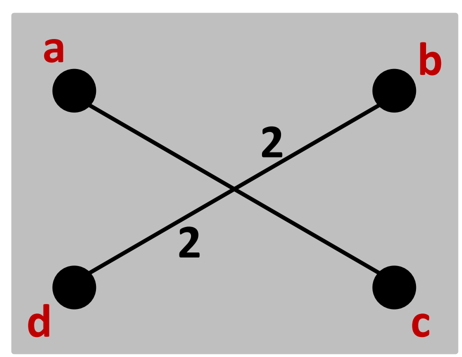


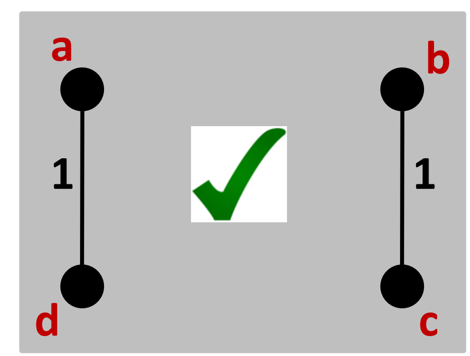
2. Compute sub-graph with set of odd-degree vertices of MST.



3. Compute minimum cost perfect matching

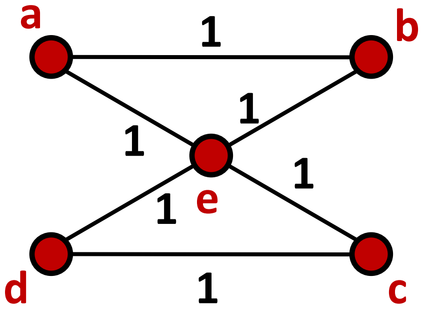




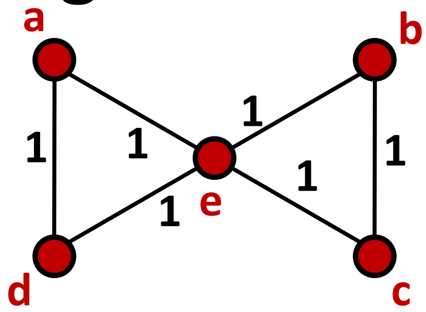


Contd…

- Add minimum cost perfect matching to MST.



OR

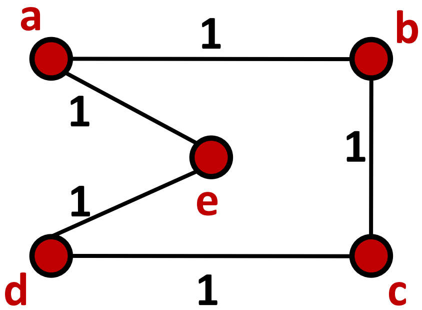


- Compute Eulerian tour (Pick any possible tour)

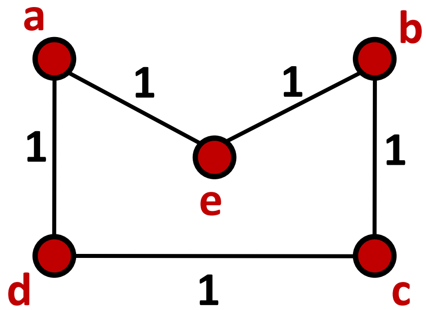
a e d c e b a OR a e b c e d a

- Compute required tour by removing repeated cities.

a e d c b a



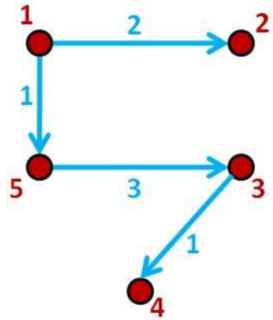
a e b c d a



Example – 2 (Same as in slide 21)

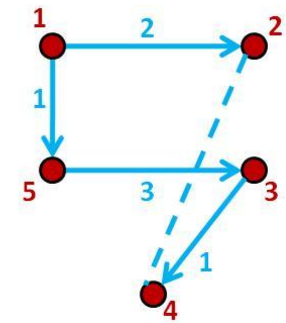
	1	2	3	4	5
1	0	2	8	6	1
2	1	0	4	4	2
3	5	3	0	1	5
4	4	7	2	0	1
5	2	6	3	6	0

- Computing MST.

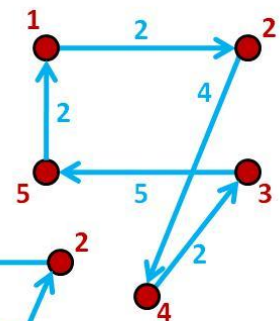


- Compute minimum cost perfect matching with set of odd-degree vertices of MST.

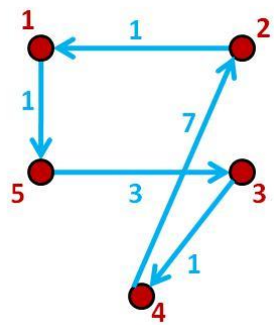
- Updated MST



- Eulerian Tour (has no repeated city)
  - 1 2 4 3 5 1. Cost = 15, or

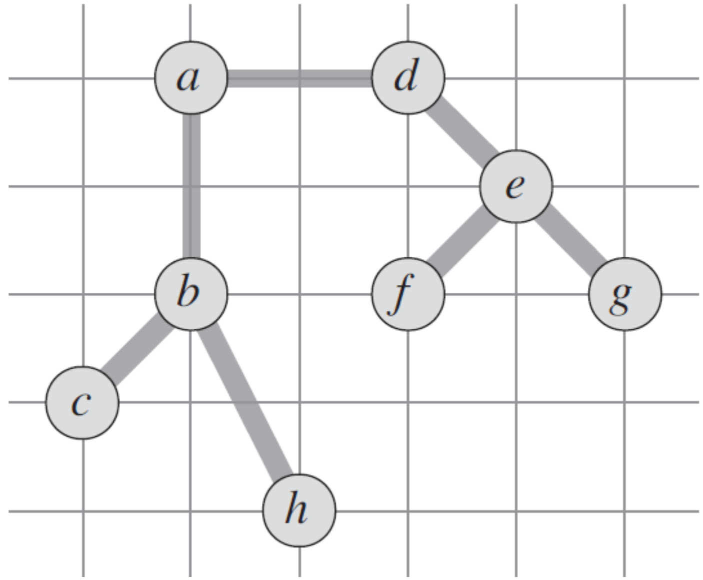


  - 1 5 3 4 2 1. Cost = 13.

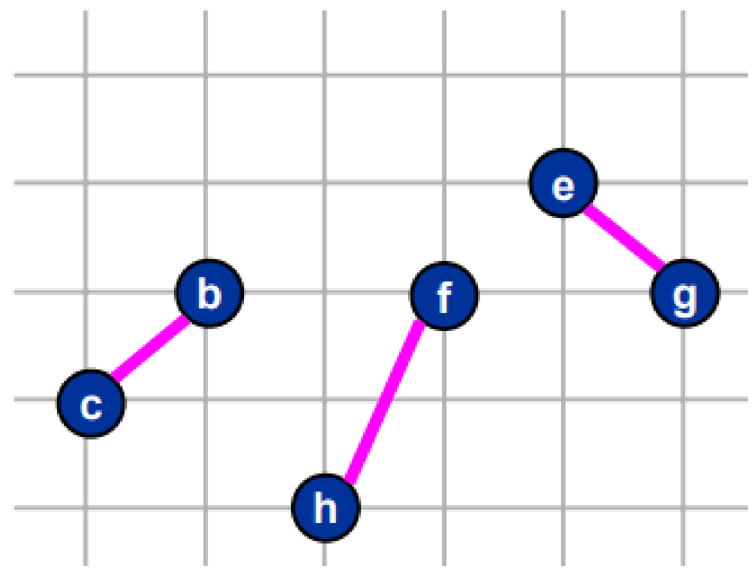


Example – 3 (Same as in slide 10)

- Computing MST.

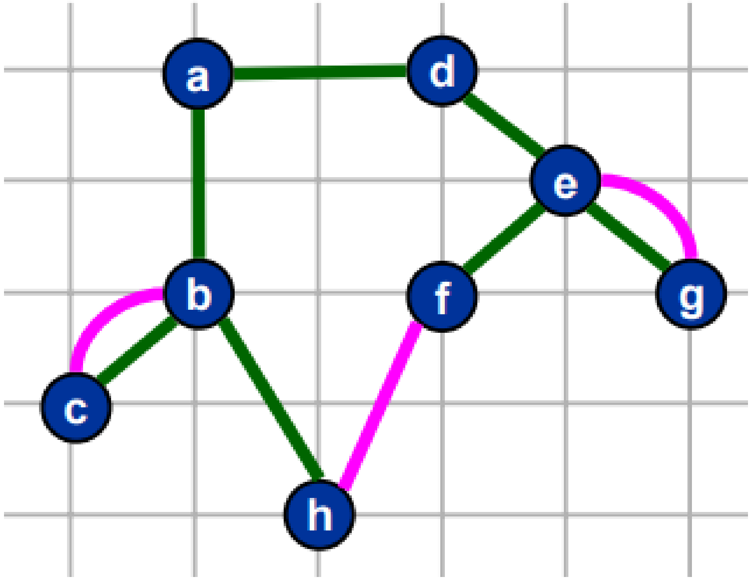


- Compute minimum cost perfect matching with set of odd-degree vertices of MST.



Contd…

- Updated MST.

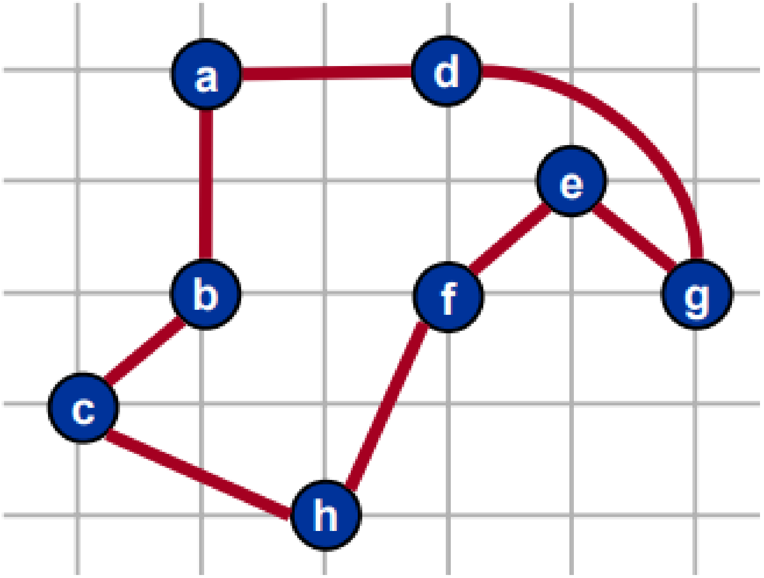


- Eulerian tour: a b c b h f e g e d a
- Required tour:

  a b c h f e g d a. Cost = 15.543
- Another possible Eulerian tour

  a d e g e f h b c b a
- Required tour:

  a d e g f h b c a. Cost = 15.877



0/1 Knapsack Problem

0/1 Knapsack Problem

- Given:
  - A knapsack of size C.
  - n distinct items.
  - Each item i has weight w_i and profit p_i.
- Goal:
  - Find subset of the items such that,

    Total profit $\sum x_i p_i$ is maximum, and

    $\sum x_i w_i \leq C$, where $x_i \in \{0,1\}$

Contd…

- Knapsack is NP-hard.
- No polynomial time algorithm exists to solve it.
- It does have a pseudo-polynomial time algorithm using dynamic programming.
- This algorithm finds the optimal solution.
- Let,
  - $K[i][j]$ be the optimal solution of this instance.
  - $K[i][j]$ represents the value of the most valuable subsets of the first i items that fit into the knapsack of capacity j.

Pseudocode – DP_Knapsack()

- for $i = 0$ to n
-         $K[i][0] = 0$
- for $j = 0$ to C
-         $K[0][j] = 0$
- for $i = 1$ to n
-         for $j = 1$ to C
-             if $j < w_i$
-                 $K[i][j] = K[i-1][j]$
-             else
-                 $K[i][j] = \max(K[i-1][j], p_i + K[i-1][j-w_i])$

Total running time = $O(n^2P_{max})$

P_{max} = profit of the most profitable object.

Pseudocode – DP_Knapsack_GetSelected(i,j)

- if $j > 0$
- if $K[i][j] == K[i-1][j]$
- DP_Knapsack_GetSelected($i - 1$, j)
- else
- print [Item i is selected]
- DP_Knapsack_GetSelected($i - 1$, $j - w_i$)

- Note:
  - First function call is DP_Knapsack_GetSelected(n, C)

Example

- Capacity of a Knapsack is 8.
- Size of K is 5 rows and

  9 columns

Item #	Profit	Weight
1	15	1
2	10	5
3	9	3
4	5	4

p_i	w_i		**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
		0	0	0	0	0	0	0	0	0	0
15	1	**1**	0								
10	5	**2**	0								
9	3	**3**	0								
5	4	**4**	0								

Contd…

p_i	w_i		0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
15	1	1	0	15	15	15	15	15	15	15	15
10	5	2	0								
9	3	3	0								
5	4	4	0								

$i = 1$, $w_1 = 1$, $p_1 = 15$.

- $j = 1$. $1 == 1$. $K[1][1] = \max(K[0][1], p_1 + K[0][0]) = \max(0, \underline{\mathbf{15 + 0}}) = 15$.
- $j = 2$. $2 > 1$. $K[1][2] = \max(K[0][2], p_1 + K[0][1]) = \max(0, \underline{\mathbf{15 + 0}}) = 15$.
- $j = 3$. $3 > 1$. $K[1][3] = \max(K[0][3], p_1 + K[0][2]) = \max(0, \underline{\mathbf{15 + 0}}) = 15$.
- $j = 4$. $4 > 1$. $K[1][4] = \max(K[0][4], p_1 + K[0][3]) = \max(0, \underline{\mathbf{15 + 0}}) = 15$.
- $j = 5$. $5 > 1$. $K[1][5] = \max(K[0][5], p_1 + K[0][4]) = \max(0, \underline{\mathbf{15 + 0}}) = 15$.
- $j = 6$. $6 > 1$. $K[1][6] = \max(K[0][6], p_1 + K[0][5]) = \max(0, \underline{\mathbf{15 + 0}}) = 15$.
- $j = 7$. $7 > 1$. $K[1][7] = \max(K[0][7], p_1 + K[0][6]) = \max(0, \underline{\mathbf{15 + 0}}) = 15$.
- $j = 8$. $8 > 1$. $K[1][8] = \max(K[0][8], p_1 + K[0][7]) = \max(0, \underline{\mathbf{15 + 0}}) = 15$.

Contd…

p_i	w_i		0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
15	1	1	0	15	15	15	15	15	15	15	15
10	5	2	0	15	15	15	15	15	25	25	25
9	3	3	0								
5	4	4	0								

$i = 2$, $w_2 = 5$, $p_2 = 10$.

- $j = 1$. $1 < 5$. K[2][1] = K[1][1] = 15.
- $j = 2$. $2 < 5$. K[2][2] = K[1][2] = 15.
- $j = 3$. $3 < 5$. K[2][3] = K[1][3] = 15.
- $j = 4$. $4 < 5$. K[2][4] = K[1][4] = 15.
- $j = 5$. $5 == 5$. K[2][5] = max(K[1][5], p_2 + K[1][0]) = max(**15**, 10 + 0) = 15.
- $j = 6$. $6 > 5$. K[2][6] = max(K[1][6], p_2 + K[1][1]) = max(15, **10 + 15**) = 25.
- $j = 7$. $7 > 5$. K[2][7] = max(K[1][7], p_2 + K[1][2]) = max(15, **10 + 15**) = 25.
- $j = 8$. $8 > 5$. K[2][8] = max(K[1][8], p_2 + K[1][3]) = max(15, **10 + 15**) = 25.

Contd…

p_i	w_i		0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
15	1	1	0	15	15	15	15	15	15	15	15
10	5	2	0	15	15	15	15	15	25	25	25
9	3	3	0	15	15	15	24	24	25	25	25
5	4	4	0								

i = 3, w_3 = 3, p_3 = 9.

- j = 1. 1 < 3. K[3][1] = K[2][1] = 15.
- j = 2. 2 < 3. K[3][2] = K[2][2] = 15.
- j = 3. 3 == 3. K[3][3] = max(K[2][3], p_3 + K[2][0]) = max(**<u>15</u>**, 9 + 0) = 15.
- j = 4. 4 > 3. K[3][4] = max(K[2][4], p_3 + K[2][1]) = max(15, **<u>9 + 15</u>**) = 24.
- j = 5. 5 > 3. K[3][5] = max(K[2][5], p_3 + K[2][2]) = max(15, **<u>9 + 15</u>**) = 24.
- j = 6. 6 > 3. K[3][6] = max(K[2][6], p_3 + K[2][3]) = max(**<u>25</u>**, 9 + 15) = 25.
- j = 7. 7 > 3. K[3][7] = max(K[2][7], p_3 + K[2][4]) = max(**<u>25</u>**, 9 + 15) = 25.
- j = 8. 8 > 3. K[3][8] = max(K[2][8], p_3 + K[2][5]) = max(**<u>25</u>**, 9 + 15) = 25.

Contd…

p_i	w_i		0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
15	1	1	0	15	15	15	15	15	15	15	15
10	5	2	0	15	15	15	15	15	25	25	25
9	3	3	0	15	15	15	24	24	25	25	25
5	4	4	0	15	15	15	24	24	25	25	29

$i = 4$, $w_4 = 4$, $p_4 = 5$.

- $j = 1$. $1 < 4$. $K[4][1] = K[3][1] = 15$.
- $j = 2$. $2 < 4$. $K[4][2] = K[3][2] = 15$.
- $j = 3$. $3 < 4$. $K[4][3] = K[3][3] = 15$.
- $j = 4$. $4 == 4$. $K[4][4] = \max(K[3][4], p_4 + K[3][0]) = \max(\underline{\mathbf{24}}, 5 + 0) = 24$.
- $j = 5$. $5 > 4$. $K[4][5] = \max(K[3][5], p_4 + K[3][1]) = \max(\underline{\mathbf{24}}, 5 + 15) = 24$.
- $j = 6$. $6 > 4$. $K[4][6] = \max(K[3][6], p_4 + K[3][2]) = \max(\underline{\mathbf{25}}, 5 + 15) = 25$.
- $j = 7$. $7 > 4$. $K[4][7] = \max(K[3][7], p_4 + K[3][3]) = \max(\underline{\mathbf{25}}, 5 + 15) = 25$.
- $j = 8$. $8 > 4$. $K[4][8] = \max(K[3][8], p_4 + K[3][4]) = \max(25, \underline{\mathbf{5 + 24}}) = 29$.

Contd…

p_i	w_i		0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
15	1	1	0	15	15	15	15	15	15	15	15
10	5	2	0	15	15	15	15	15	25	25	25
9	3	3	0	15	15	15	24	24	25	25	25
5	4	4	0	15	15	15	24	24	25	25	29

- DP_Knapsack_GetSelected(4,8)
  - 8 > 0. K[4][8] != K[3][8]. Item 4 is selected.
- DP_Knapsack_GetSelected(3,4)
  - 4 > 0. K[3][4] != K[2][4]. Item 3 is selected.
- DP_Knapsack_GetSelected(2,1)
  - 1 > 0. K[2][1] == K[1][1].
- DP_Knapsack_GetSelected(1,1)
  - 1 > 0. K[1][1] != K[0][1]. Item 1 is selected.
- DP_Knapsack_GetSelected(0,0)
  - 0 == 0. Stop.

Selected Item #	Profit	Weight
4	5	4
3	9	3
1	15	1
Total	**29**	**8**

FPTAS (Fully Polynomial Time Approximation Scheme) for 0/1 Knapsack

- Given ε > 0, let $K = \frac{\varepsilon \times P_{\max}}{n}$, where P_{max} is profit of the most profitable object among n objects.
- for j = 1 to n, $P_j^{'} = \left\lfloor \frac{P_j}{K} \right\rfloor$
- Solve the problem with updated profit values using dynamic programming.
- Return the solution.

$$\boxed{P_{approx} \geq (1-\varepsilon)P_{optimal}}$$

Example

- $\varepsilon = 1/50$, $P_{max} = 28343199$, $n = 5$.
- K = 113372.796
- Capacity of a Knapsack is 11.

Item #	**Profit**	**Weight**	**Floor(Profit/K)**
1	134221	1	1
2	656342	2	5
3	1810013	5	15
4	22217800	6	195
5	28343199	7	250

Contd…

P'_i	w_i		0	1	2	3	4	5	6	7	8	9	10	11
		0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
5	2	2	0	1	5	6	6	6	6	6	6	6	6	6
15	5	3	0	1	5	6	6	15	16	20	21	21	21	21
195	6	4	0	1	5	6	6	15	195	196	200	201	201	210
250	7	5	0	1	5	6	6	15	195	250	251	255	256	256

Selected Item #	Profit	Weight	Actual Profit
5	250	7	28343199
2	5	2	656342
1	1	1	134221
Total	**256**	**10**	**29133762**

For this example, $P_{approx} = P_{optimal} = 29133762$

Solve using FPTAS

- $\varepsilon = ½$.
- Capacity of a Knapsack is 9.

Item #	**Profit**	**Weight**
1	25	2
2	31	3
3	48	6
4	56	7
5	16	5
6	27	9
7	31	4