Greedy algorithms

Yulia Burkatovskaya

Outline

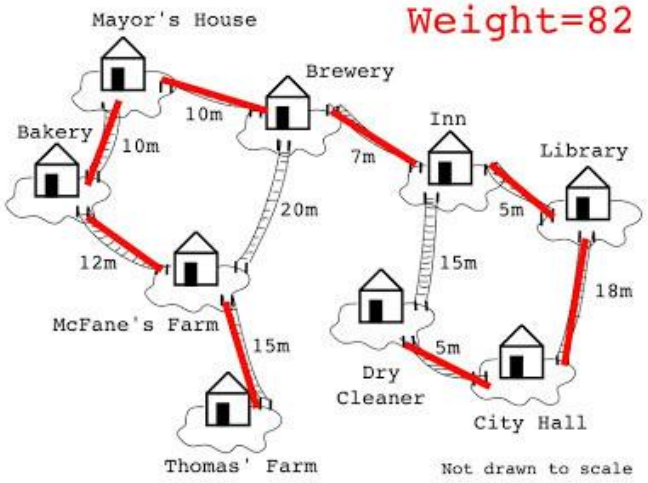
- Greedy algorithm
- Shortest spanning tree
- Travelling salesman problem

Greedy algorithm

- A greedy algorithm is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum.
- In many problems, a greedy strategy does not in general produce an optimal solution.

Shortest spanning tree

- How to join all houses and to minimize the length of the communications?
- In a weighted graph, the **shortest spanning tree** is the set of edges with the minimum total weight such that they connect all of the nodes.



- http://computationaltales.blogspot.ru/2011/08/minimum-spanning-trees-prims-algorithm.html

Shortest spanning tree

Application

- **Network design.**
  – *telephone, electrical, hydraulic, TV cable, computer, road*
  Approximation algorithms for NP-hard problems.
  – *traveling salesperson problem, Steiner tree*
- **Indirect applications.**
  – max bottleneck paths
  – LDPC codes for error correction
  – image registration with Renyi entropy
  – learning salient features for real-time face verification
  – reducing data storage in sequencing amino acids in a protein
  – model locality of particle interactions in turbulent fluid flows
  – autoconfig protocol for Ethernet bridging to avoid cycles in a network
- **Cluster analysis**
  k clustering problem can be viewed as finding an MST and deleting the k-1 most expensive edges.
- https://www.geeksforgeeks.org/?p=11110

Shortest spanning tree

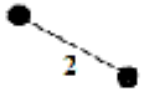
- **Kruskal's algorithm**
- Sort all the edges from low weight to high
- Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
- Keep adding edges until we have p−1 edges.

Shortest spanning tree

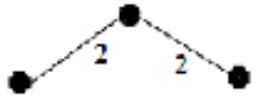
Kruskal's Algorithm

1 *Given a network……*

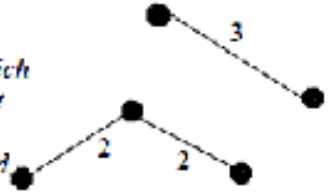
2 *Choose the shortest edge (if there is more than one, choose any of the shortest)……..*



3 *Choose the next shortest edge and add it……..*

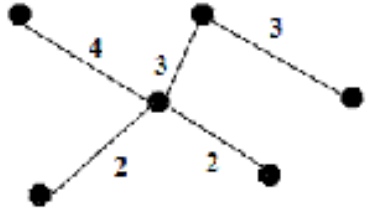


4 *Choose the next shortest edge which wouldn't create a cycle and add it.*



5 *Choose the next shortest edge which wouldn't create a cycle and add it.*

6 *Repeat until you have a minimal spanning tree.*



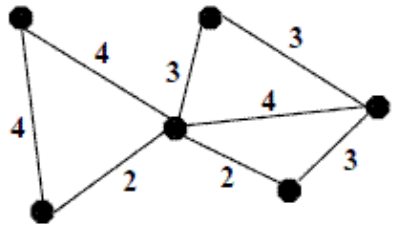
Shortest spanning tree

- **Prim's algorithm**
- Initialize the minimum spanning tree with a vertex chosen at random.
- Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree
- Keep adding edges until we have p−1 edges.

Shortest spanning tree

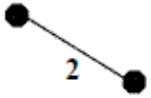
Prim's Algorithm

1 *Given a network……*

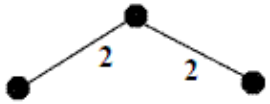


2 *Choose a vertex*

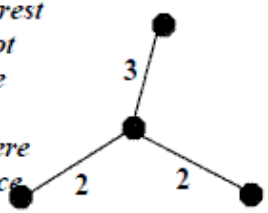
3 *Choose the shortest edge from this vertex.*



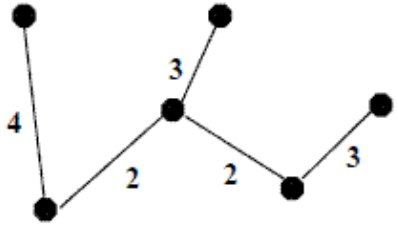
4 *Choose the nearest vertex not yet in the solution.*



5 *Choose the next nearest vertex not yet in the solution, when there is a choice choose either.*



6 *Repeat until you have a minimal spanning tree.*



Shortest spanning tree

- **Greedy algorithms** construct **the shortest tree**!

Traveling Salesman Problem

Consider graph G(V,E) in which each vertex represents a city and each edge represents a road connecting two cities, and $d(x,y)$ is the weight of the edge (x,y) standing by the distance between cities x and y.

Problem: finding a cycle in G which visits each vertex once in minimum total distance.

This problem is **NP-hard**.

Greedy algorithms don't result in the optimal solution

Some heuristic methods

- Cycle construction heuristics
- Cycle improvement heuristics

Nearest neighbor method

This is essentially a **greedy algorithm**.

Begin with any vertex *x* and find the vertex *y* so that *d*(*x*,*y*) is the smallest among all *y*.

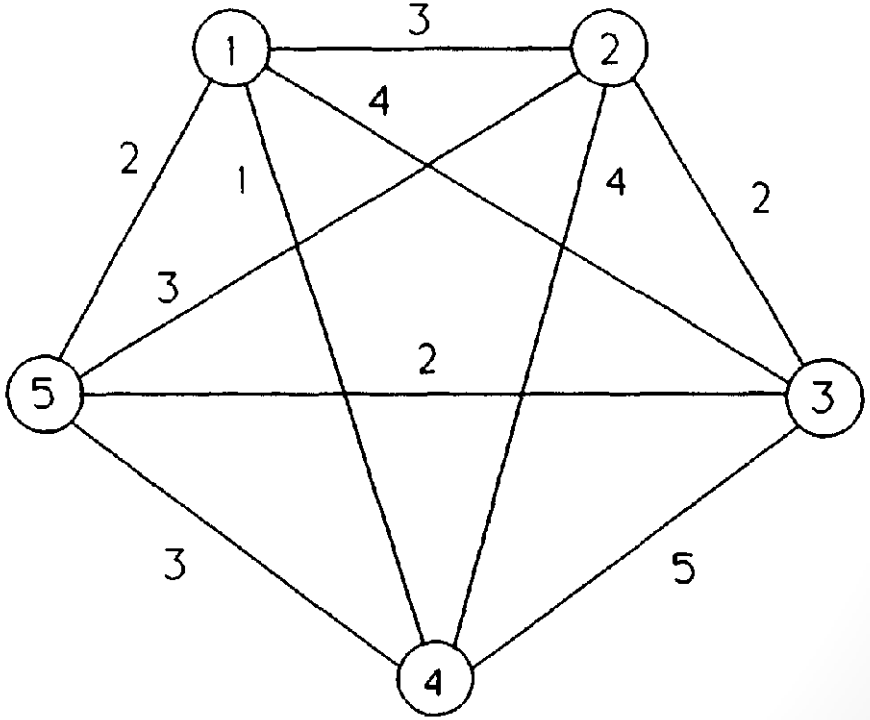
Next, find the closest vertex to y that is not already in the tour, say vertex *z*, and add edge (*y*, *z*) to the tour. Repeat this process until the last vertex is added and then join the first and last vertices by the unique edge between them.

Algorithm

Example.

The final cycle is

1–4–5–3–2–1.



Nearest insertion method

Select one vertex to start, say vertex *i*. Choose the nearest vertex, say j, and form the subtour $i - j - i$.

At each iteration, find the vertex κ not in the subtour that is closest to any vertex in the subtour. Find the edge (i,j) in the subtour which minimizes $d(i,k) + d(k,j) - d(i,j)$. Insert vertex κ between i and j.

Repeat this process until a tour is constructed. Note that in the iterative step, we try to add the least amount of distance to the current subtour by removing edge (i,j) and adding edges (i,k) and (k, j).

Algorithm

Example.

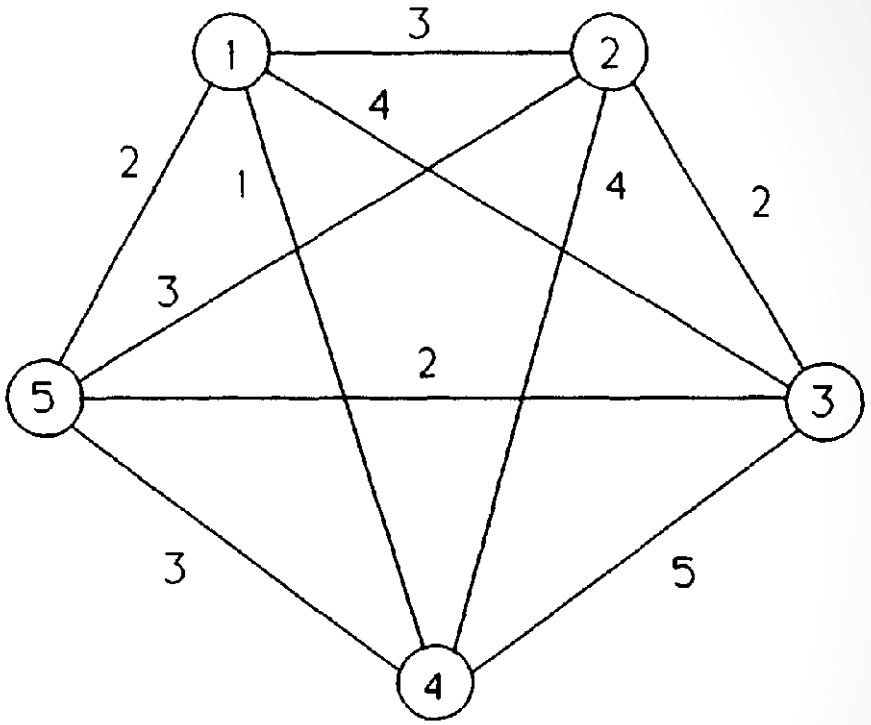
1–4–1

1–4–5–1

1–4–3–5–1

For the vertex 2

Arc	Incremental Cost
(1, 4)	3 + 4 − 1 = 6
(4, 3)	4 + 2 − 5 = 1
(3, 5)	3 + 2 − 2 = 3
(5, 1)	3 + 3 − 2 = 4



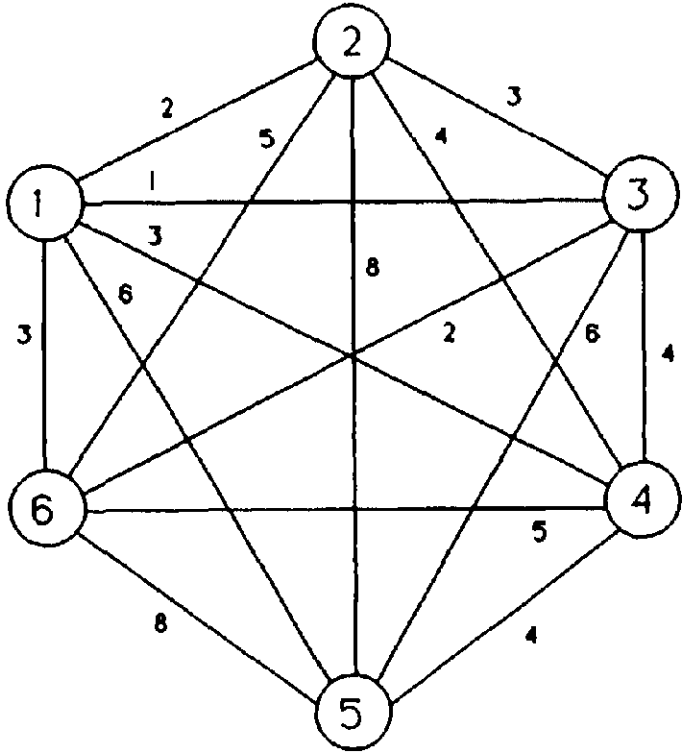
1–4–2–3–5–1

Christofides' Heuristic

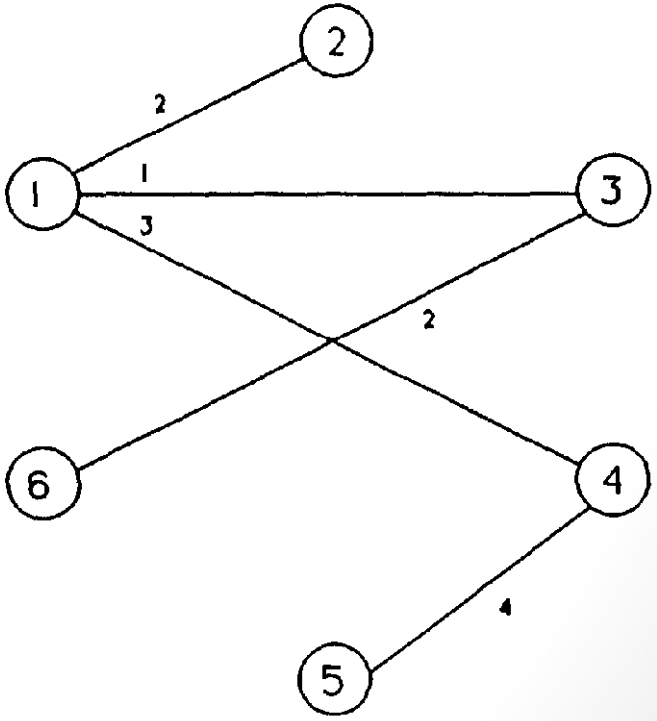
1. Construct the minimum spanning tree of the graph.
2. Find the minimum-cost matching of the odd-degree vertices in the spanning tree. Add the edges from the optimal matching to the tree to create an Euler graph.
3. Find an Euler circuits in this graph.
4. Transform the Euler circuits into a Hamiltonian cycle deleting repeated vertices.

Algorithm

Consider the TSP shown below.

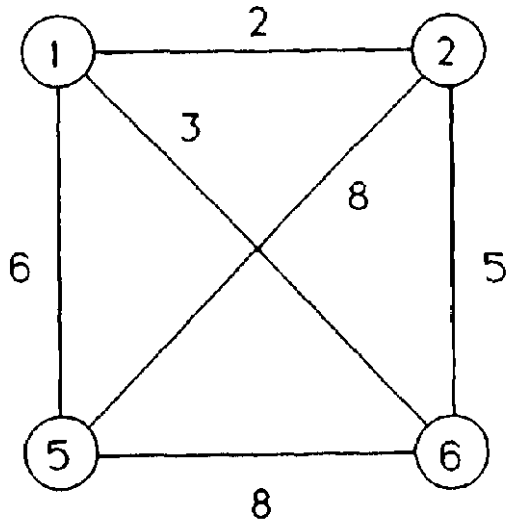


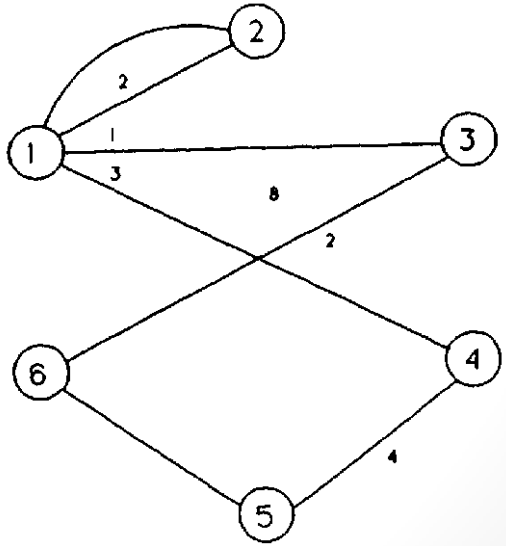
The minimal spanning tree is



Algorithm

The odd-degree vertices are 1, 2, 5, and 6. The matching problem on these vertices follows. The optimal matching is (1, 2) and (5, 6). Adding these edges to the minimal spanning tree creates the Euler graph





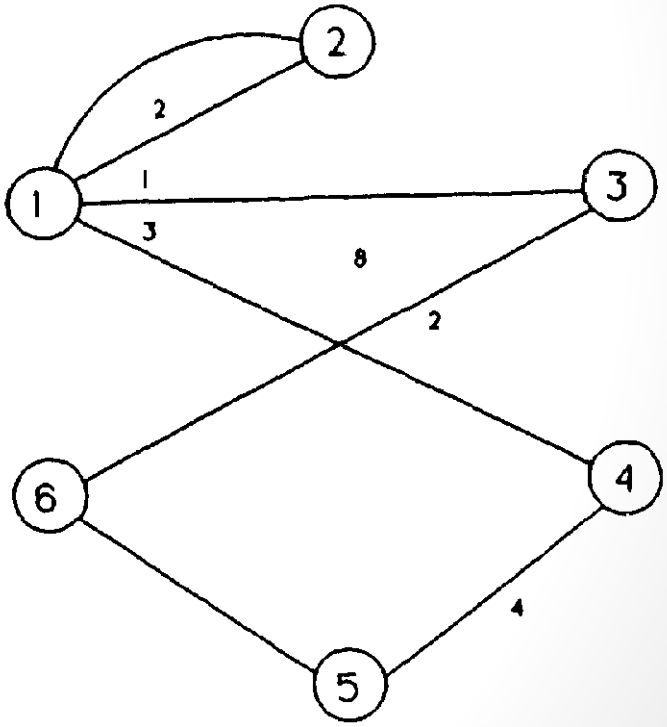
Algorithm

An Euler tour in this graph is

1-2-1-4-5-6-3-1.

When vertex 1 is repeated, we replace the path 2-1-4 by the edge (2, 4) to create the Hamiltonian cycle

1-2-4-5-6-3-1.



k-opt Heuristics

A *k*-change of a tour consists of deleting *k* edges and replacing them by *k* other edges to form a new tour.

The heuristic procedure begins with any feasible tour. From this tour, all possible *k*-changes are examined, if a tour is found that has a lower cost than the current solution, it becomes the new solution. The process is repeated until no further *k*-change results in a better solution.

Algorithm

All 2-changes

(1-2-3-5-4-1 minimal)

Tour	Cost
1-3-2-4-5-1	15
1-4-3-2-5-1	13
1-2-4-3-5-1	16
1-2-5-4-3-1	18
1-2-3-5-4-1	11