Algebra and Discrete Mathematics (ADM)

Tutorial 11 Trees and networks

Lecturer: Bc. Xiaolu Hou, PhD.
xiaolu.hou@stuba.sk

Steiner Trees

Definition

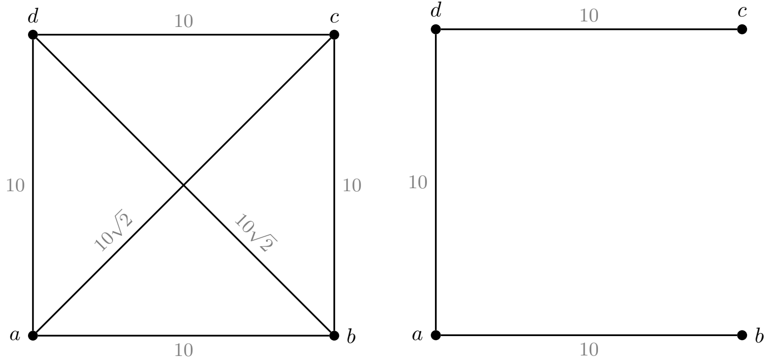
- For a graph G, a *Steiner point* is a new point p added to the graph that has vertex degree of 3 where the three edges incident to p form 120° angles.
- A *Steiner tree* is a tree that only consists of Steiner points and the original vertices of G.

- A Steiner point is similar to a Fermat point in that it is added to graph to find the shortest network connecting the original vertices
- It has been shown that finding a shortest network amounts to finding a minimum Steiner tree
- This problem is named for the 19th century Swiss mathematician Jakob Steiner

Finding Steiner Trees

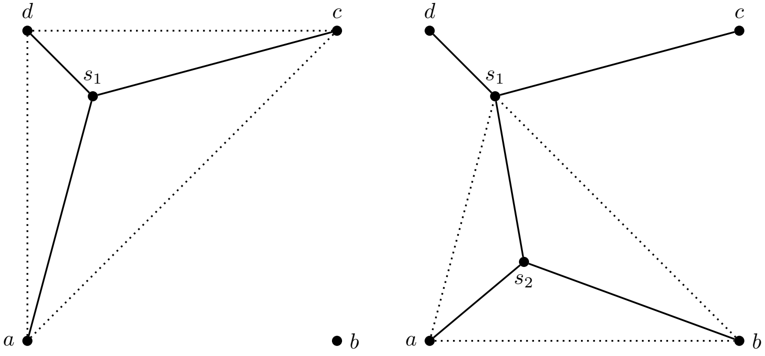
- Though the Steiner Tree Problem sounds fairly simple, it is in fact among a class of problems known as NP-Hard
  - solutions can be verified quickly
  - finding a solution can be quite hard

Four vertices – MST



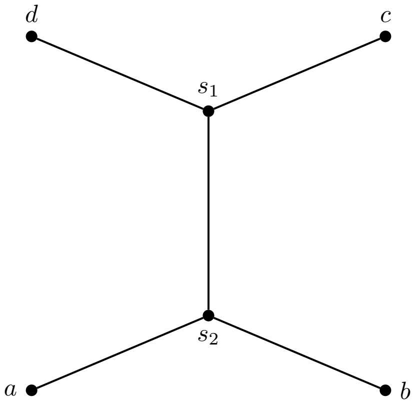
- A graph with four vertices that lie on a square
- A minimum spanning tree consists of three of the fours edges of length 10

Four vertices – Fermat points



- s_1: Fermat point for $\triangle adc$
- s_2: Fermat point for $\triangle abs_1$
- Length of network: ≈ 27.852
- Not Steiner points, since they do not have edges that form 120° angles

Four vertices – Steiner tree



- Total length 27.321
- Network with Fermat points: 27.852
- MST: 30

Steiner Network Method

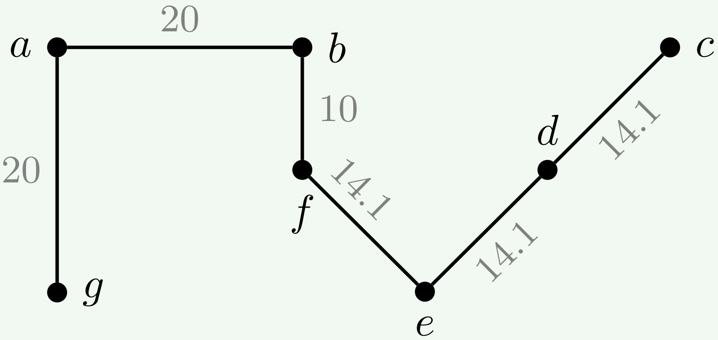
- We will not be finding the minimum Steiner tree
- Use the ideas behind a Steiner tree to find a shorter network than a MST, if possible

Steiner Network Method

- Step 1. Find the MST of the network
- Step 2. Form a triangle from two existing edges of the minimum spanning tree. If all angles of this triangle measure less than 120°, find the Fermat point.
- Step 3. Update the network by removing the two edges from the MST used in Step 2 and adding new edges to the Fermat point
- Step 4. Repeat steps 2 and 3 until all possible triangles have been considered

Steiner Network Method – example

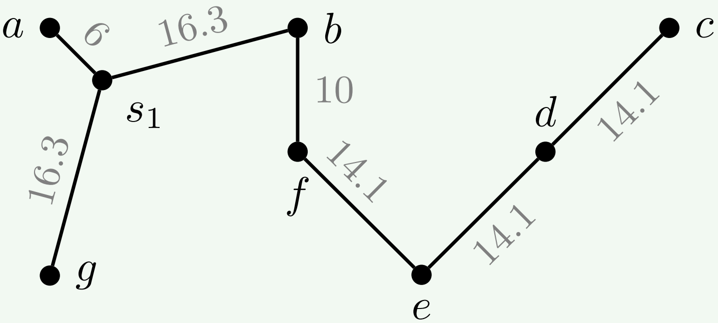
Example



- We begin with vertices a, b, g since the angle at a is $90°$

Steiner Network Method – example

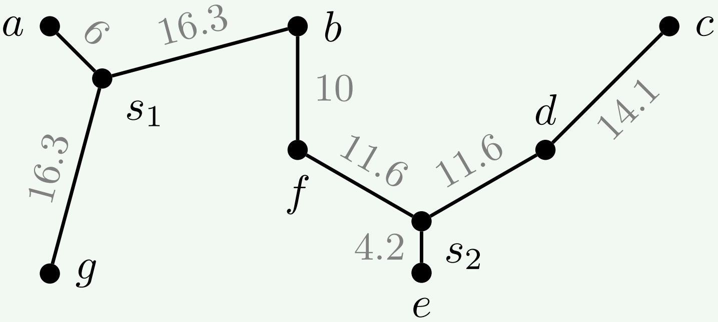
Example



- We begin with vertices a, b, g. Using Torricelli's Construction, we get point s_1

Steiner Network Method – example

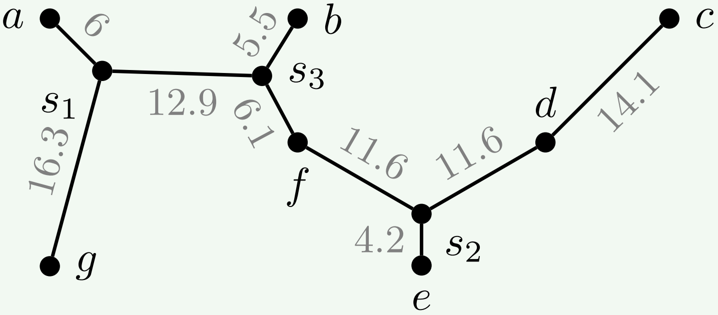
Example



- Take vertices f, e, d - angle at e is 90° Using Torricelli's Construction, we get point s_2
- At this point, due to angle measures, there is only one triangle to be considered: b, f, s_1

Steiner Network Method – example

Example



- Take vertices b, f, s_1 we get point s_3
- Total length: 88.3
- Original (MST) length: 92.3

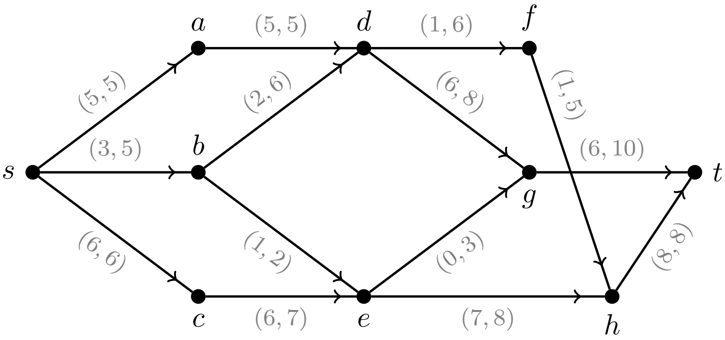
Remark

- Although we did not fully answer the optimization question for networks containing more than three points
- Using the Steiner Network Method provides a quick and simple procedure for finding locations for improvements to the minimum spanning tree

Augmenting Flow Algorithm – steps

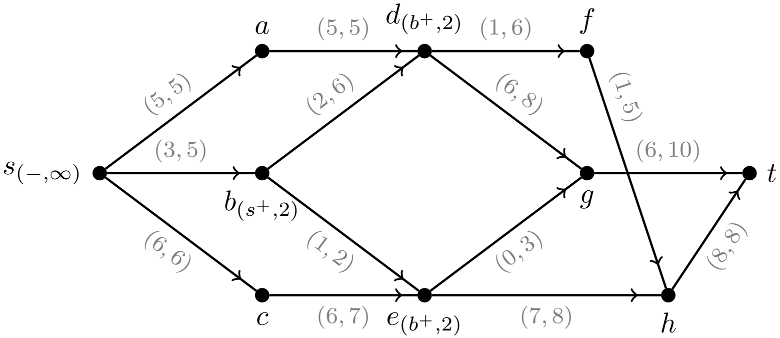
1. Label s with $(-, \infty)$, set $\sigma(v) = \infty$ for other vertices
2. Choose a labeled vertex x
   a. For any arc yx, if $f(yx) > 0$ and y is unlabeled, then label y with $(x^-, \sigma(y))$, where $\sigma(y) = \min\{\sigma(x), f(yx)\}$
   b. For any arc xy, if $k(xy) > 0$ and y is unlabeled, then label y with $(x^+, \sigma(y))$, where $\sigma(y) = \min\{\sigma(x), k(xy)\}$
3. If t has been labeled, go to Step 4. Otherwise, choose a different labeled vertex that has not been scanned and go to Step 2. If all labeled vertices haven been scanned, then f is a maximum flow.
4. Find an $s - t$ chain K of slack edges by backtracking from t to s. Along the edges of K, increase the flow by $\sigma(t)$ units if they are in the forward direction and decrease by $\sigma(t)$ units in they are in the backward direction. Remove all vertex labels except that of s and return to Step 2

Augmenting Flow Algorithm



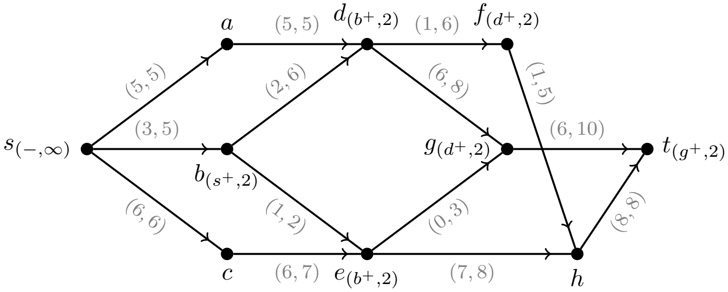
- Question 3 - 1

Augmenting Flow Algorithm



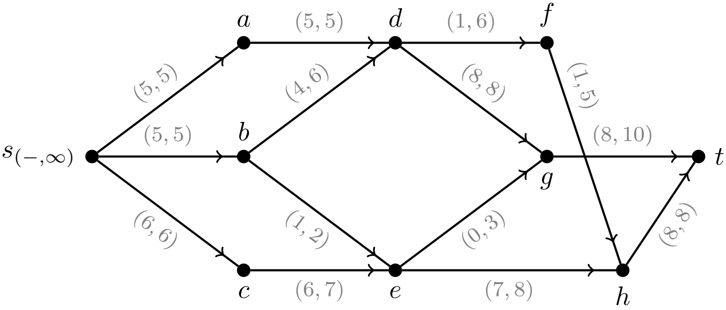
- Step 1. Label s
- Step 2. Label b
- Step 3. Choose b
- Step 2. Label d, e

Augmenting Flow Algorithm



- Step 3. Choose d
- Step 2. Label f, g
- Step 3. Choose g
- Step 2. Label t

Augmenting Flow Algorithm



- Step 3. go to step 4
- Step 4. find chain $sbdgt$
  - Increase the flow by $\sigma(t) = 2$ units along each of these edges since all are in the forward direction.
  - Update the network flow and remove all labels except for s
- No more vertex to label
- Value of flow $|f| = f^{+}(s) = 16$

Min-Cut Method

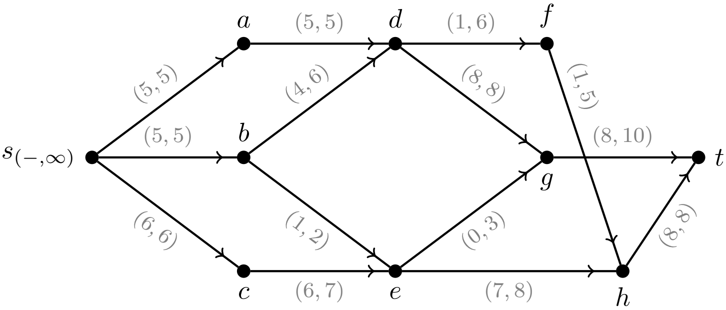
Steps

1. Let $G = (V, A, c)$ be a network with a designated source s and sink t and each arc is given a capacity c
2. Apply the Augmenting Flow Algorithm
3. Define an $s - t$ cut $(P, \overline{P})$ where P is the set of labeled vertices from the final implementation of the algorithm
4. $(P, \overline{P})$ is a minimum $s - t$ cut for G

Note

In practice, we can perform the Augmenting Flow Algorithm and the Min-Cut Method simultaneously, thus finding a maximum flow and providing a proof that it is maximum (through the use of a minimum cut) in one complete procedure.

Augmenting Flow Algorithm



- Value of flow $|f| = f^+(s) = 16$
- $P = \{s\}$
- $C(P, \overline{P}) = 5 + 5 + 6 = 16$

Breadth-First Search Tree

- Main objective is to add as many neighbors of the root as possible in the first step
- At each additional step, we are adding all available neighbors of the most recently added vertices.
- As with depth-first, we will use an alphabetical ordering neighbor lists
- Input: Simple connected graph $G = (V, E)$ and a designated root vertex r
- Output: Breadth-first tree T

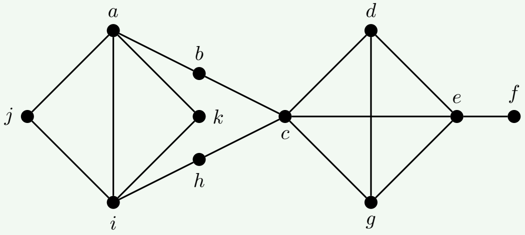
Breadth-First Search Tree – steps

1. Initialize the BFS tree $T = (V', E')$ with the root vertex r, i.e., $V' = \{r\}$, and mark r as visited.
2. Add all neighbors of r to V', and add the corresponding edges from r to each neighbor to E'. Mark all these neighbors as visited. Let this set of newly added vertices be the current level.
3. For each vertex v in the current level (in alphabetical order):
   - Add all unvisited neighbors x of v to V'.
   - Add the edge (v, x) to E'.
   - Mark each such neighbor x as visited.

   Let the collection of all such newly added vertices form the next level.
4. If T now includes all vertices of G, the process is complete. Otherwise, repeat step 3 using the next level as the current level.

Breadth-First Search Tree – example

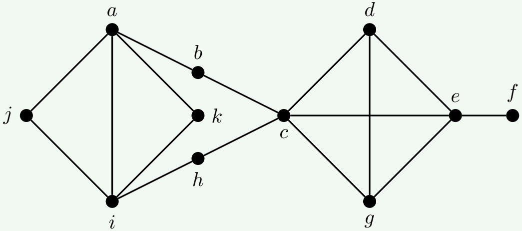
Example



- Let's consider the same example as from the lecture
- Take a as the root
- Step 1. add $a, b, i.j, k$
- Step 2. current level: b, i, j, k

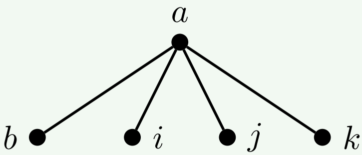
Breadth-First Search Tree – example

Example



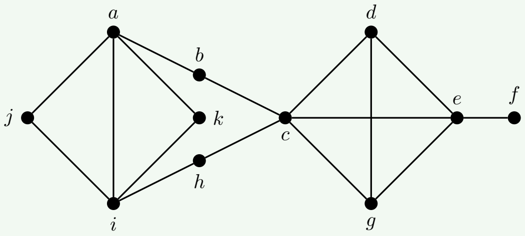
Step 3

- Vertex b: add neighbor c, and edge bc
- Vertex i: add neighbor h, and edge ih
- j and k do not have any unvisited neighbors

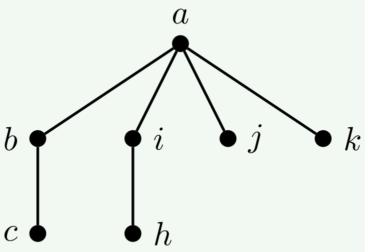


Breadth-First Search Tree – example

Example

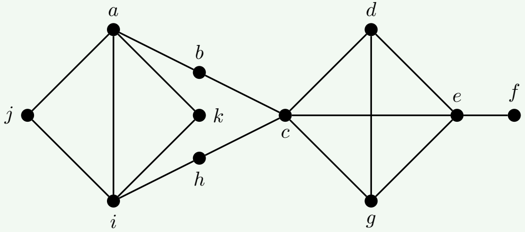


- Step 4. T does not contain all vertices, repeat step 3. Current level: c, h
- Step 3.
  - Vertex c: unvisited neighbors d, e, g
  - Vertex h does not have unvisited neighbors

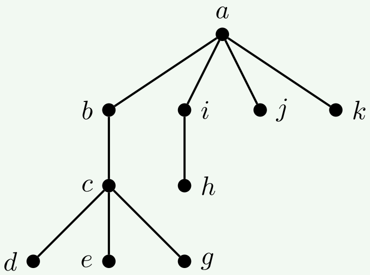


Breadth-First Search Tree – example

Example

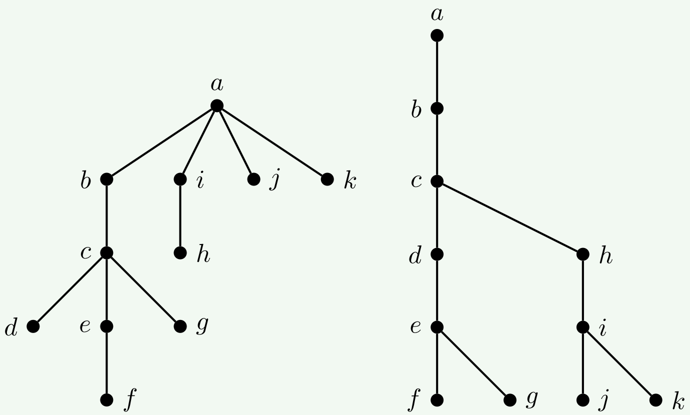


- Step 4. T does not contain all vertices, repeat step 3. Current level: d, e, g
- Step 3.
  - Vertex d has no unvisited neighbors
  - Vertex e: add neighbor f
  - Now we have all vertices



Breadth-First Search Tree – example

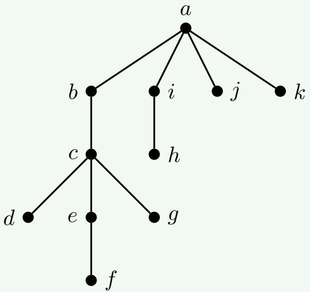
Example

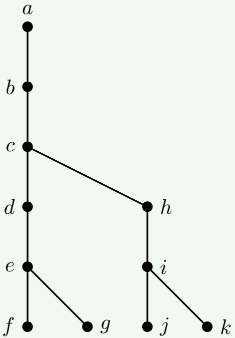


- Left: BFS tree; Right: DFS tree
- BFS trees are likely to be of shorter height than their DFS tree counterpart.

Breadth-First Search Tree – example

Example





- BFS tree: height 4 with four vertices on level 1, two vertices on level 2, three vertices on level 3, and one on level 4.
- DFS tree: height 5, one vertex each at level 1 and 2, two vertices each at levels 3 and 4, four vertices at level 5