Voronoi diagram and Delaunay triangulation

Yann GAVET, Johan DEBAYLE

Ecole Nationale Supérieure des Mines de Saint-Etienne
Laboratoire Georges Friedel, UMR CNRS 5307

Plan

Section 1

Introduction

Section 2

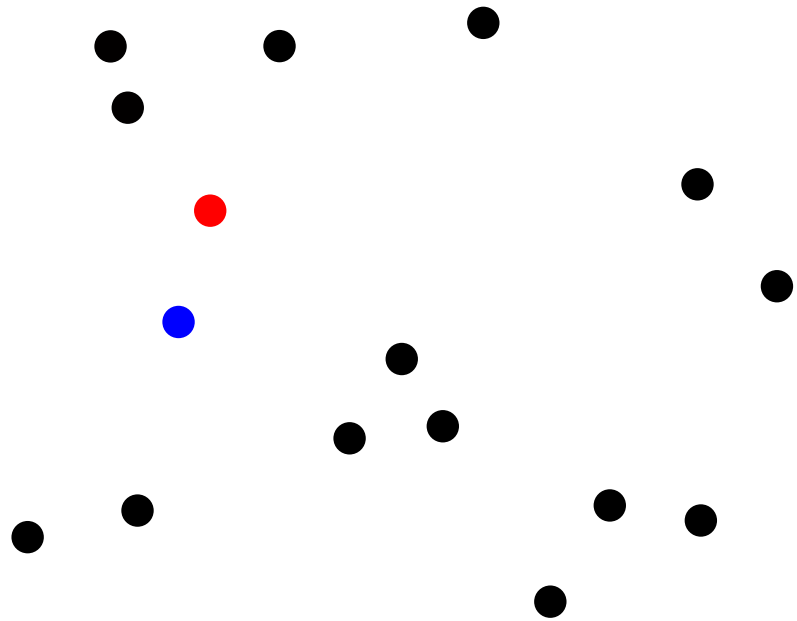
Nearest Neighbors

Nearest Neighbors search

Problem

Search for the (k) closest point(s) to a query point $q \in M$ in a set $S \subset M$.

- M is a not necessarily a metric space.
- Usual metrics are used: L_1, L_2, L_∞.
- $M = \mathbb{R}^d$ is common



Methods [Bhatia and Vandana, 2010]

Exact methods

- Linear search (naive algorithm)
- Space partitioning
  - KD tree
  - Ball tree
  - R tree

Approximate methods

return a points at most c times the distance from the query to its nearest points.

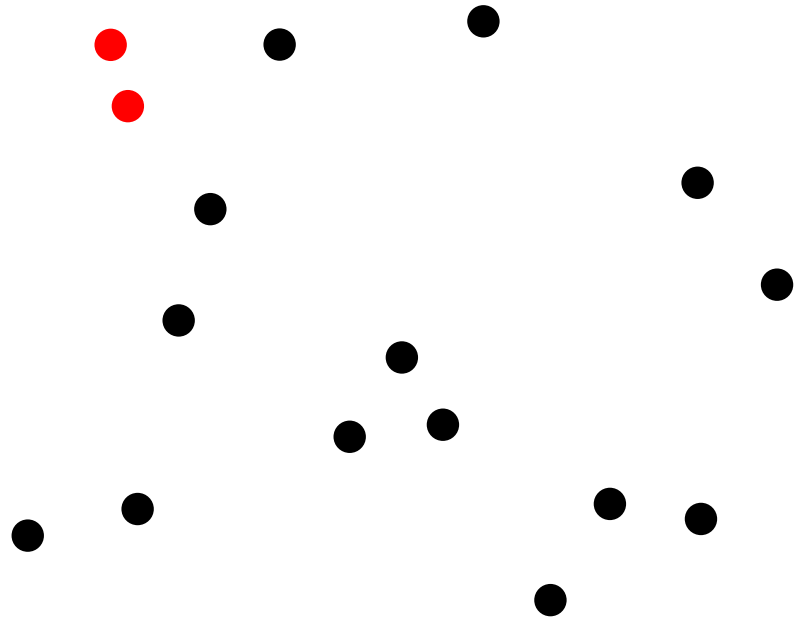
Particular case: closest pair of points

- $S \subset M$ set of points
- Minimum for all points $q \in S$
- search for closest point $p_q \in S$, $p_q \neq q$

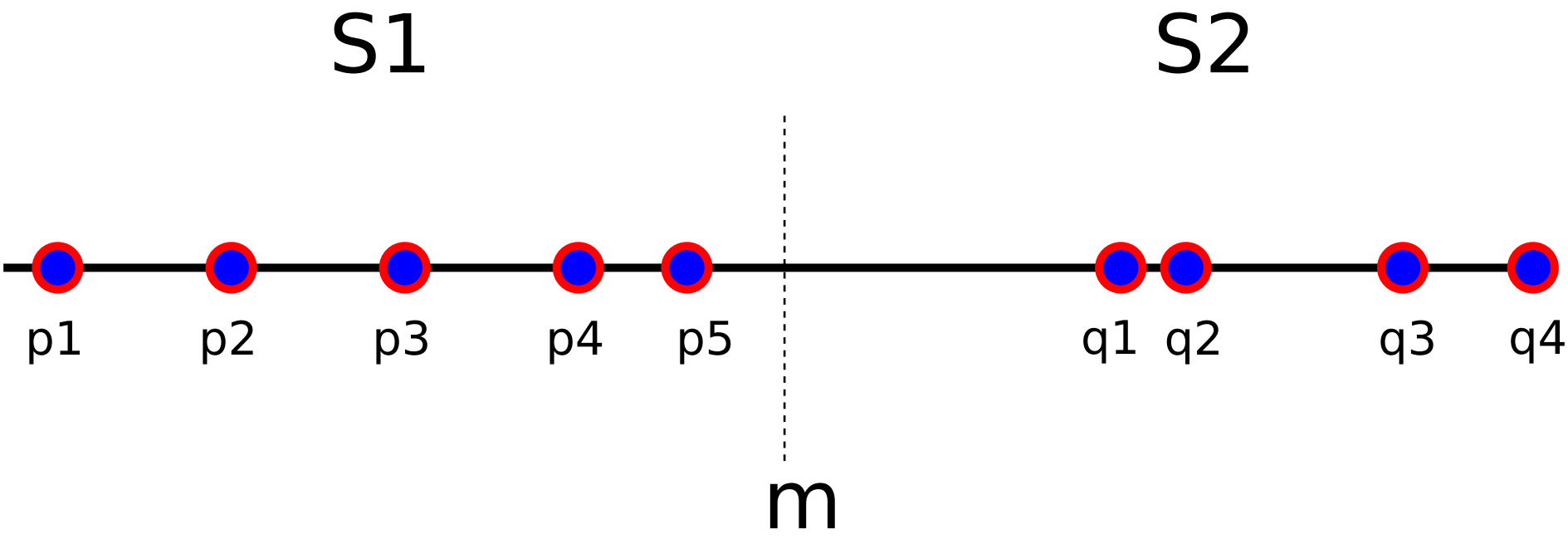
Closest pair of points

Exhaustive search

- n points in $\mathbb{R}^d$
- All pairs of points: $O(dn^2)$
- One query point: $O(nd)$
- NB: d is sometimes missed in complexity notations (constant)

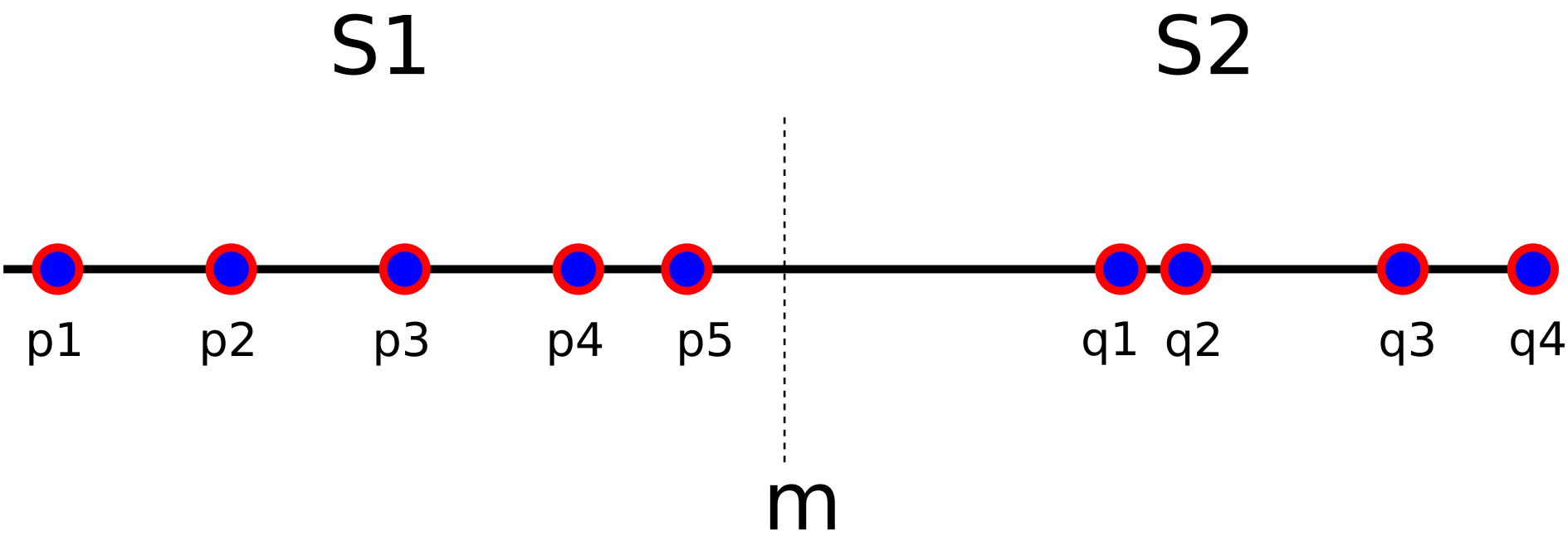


Closest pair of points: 1D problem



- Divide and conquer
  $O(n \log n)$
- Based on sorting

Closest pair of points: 1D problem



- Separate into 2 sets S_L and S_R by value m
- $p_i \in S_L$, $q_j \in S_R$
- $\forall i, j, p_i < q_j$

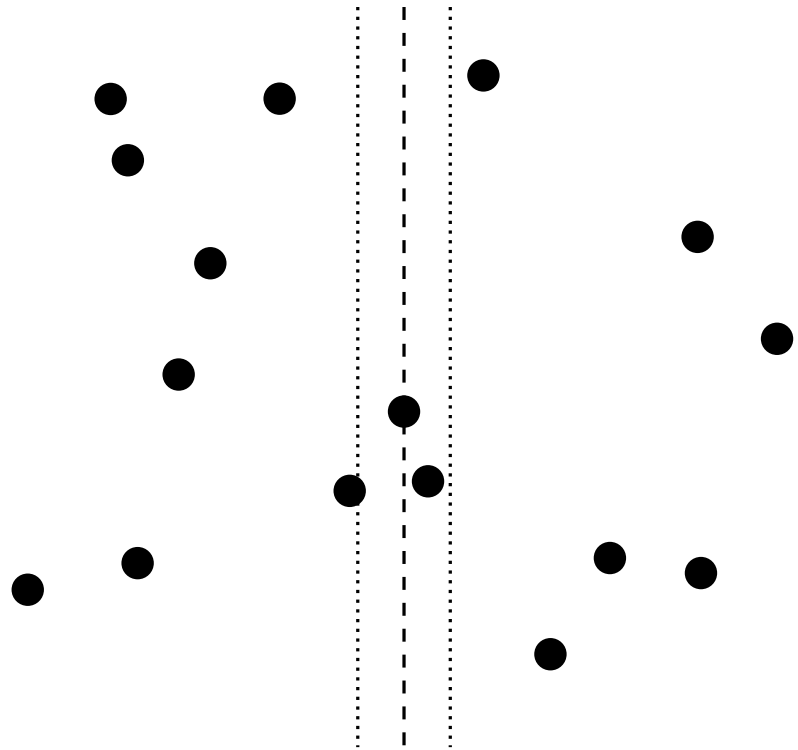
- Recursively compute closest pairs (p_k, p_{k+1}), (q_l, q_{l+1})
- $\delta = \min\{p_{k+1} - p_k, q_{l+1} - q_l\}$
- Merge: either (p_k, p_{k+1}), (q_l, q_{l+1}) or (p_m, q_n)

Closest pair of points: 2D problem [Cormen et al., 2009]

Algorithm

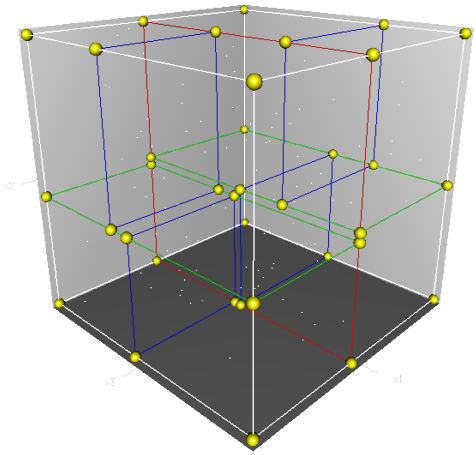
- 2 sets S_L and S_R
- $\delta = \min(\delta_L, \delta_R)$
- Combine: 8 points can lie in a $\delta \times 2\delta$ rectangle

- Complexity: $T(n) = 2T(n/2) + O(n)$
- $O(n \log n)$
- $O(n \log^{d-1} n)$ in $\mathbb{R}^d$



Space partitionning

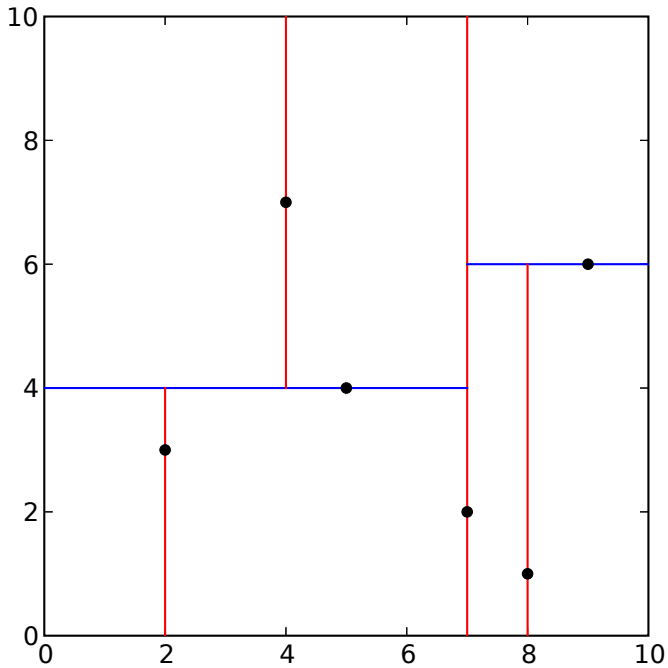
- Divides a space into non-overlapping regions
- Construction of a data structure
  - trees
  - graphs

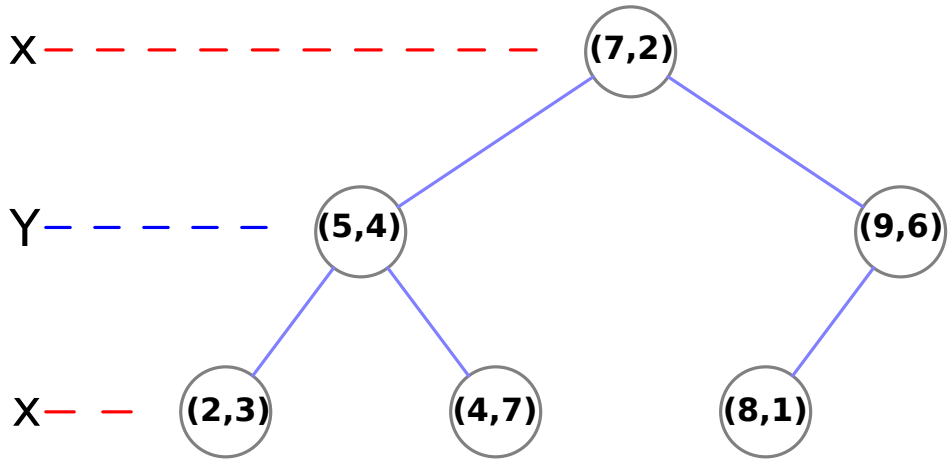


- BSP trees (Binary Space Partitioning)
- k-dtrees;
- Quadtrees;
- Octrees;
- Bins;
- R-trees;
- Bounding volume hierarchies.

k-d tree

- Partitions the space by hyperplanes
- in $\mathbb{R}^d$, performs partitionning axis by axis
- median value is used for balancing the tree





k-d tree of the points
$(2,3),(5,4),(9,6),(4,7),(8,1),(7,2)$

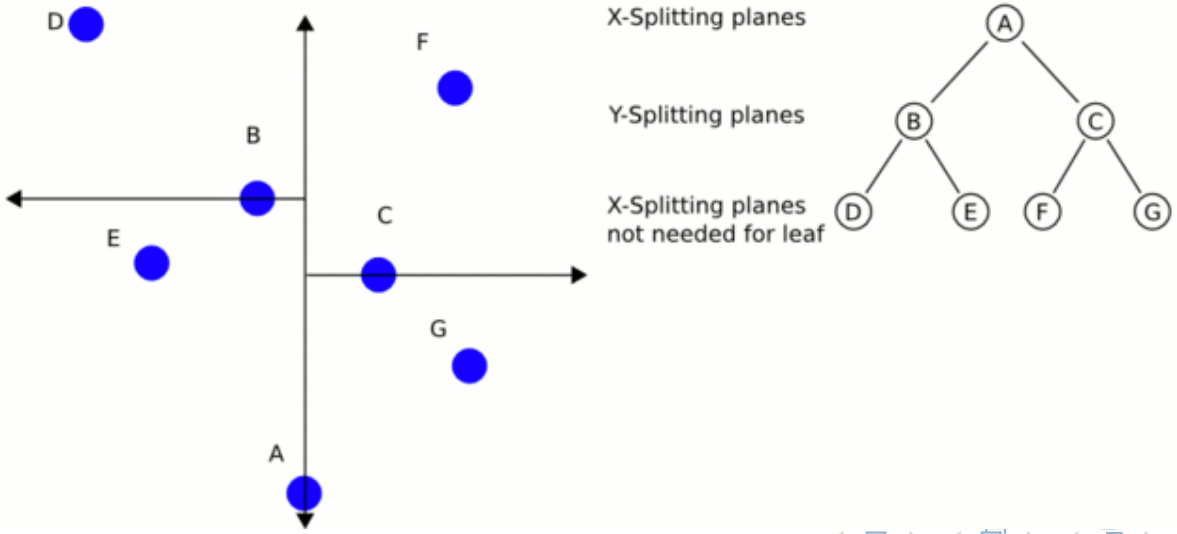
k-d tree algorithms

Median value

- Sorting in $O(n \log n)$
- ...would be inefficient
- Median approximation: sort a subsample

Complexity

	Average	**Worst**
Space	$O(n)$	$O(n)$
Search	$O(n \log n)$	$O(n)$
Insert	$O(n \log n)$	$O(n)$
Delete	$O(n \log n)$	$O(n)$



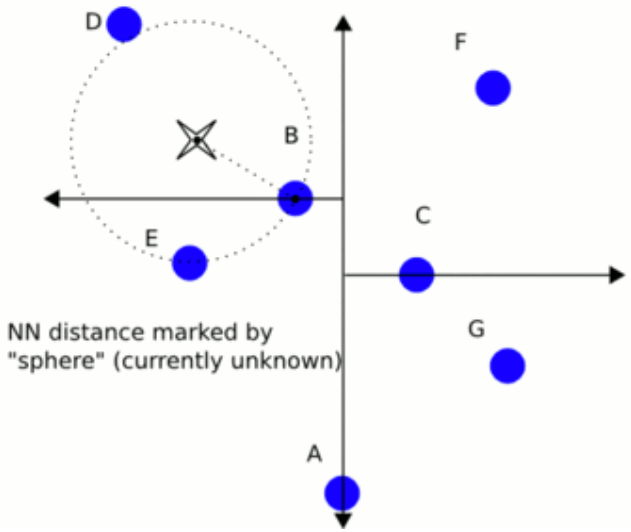
k-d tree algorithms

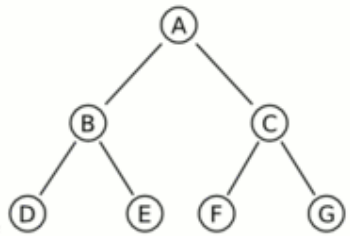
Median value

- Sorting in $O(n \log n)$
- ...would be inefficient
- Median approximation: sort a subsample

Complexity

	Average	**Worst**
Space	$O(n)$	$O(n)$
Search	$O(n \log n)$	$O(n)$
Insert	$O(n \log n)$	$O(n)$
Delete	$O(n \log n)$	$O(n)$





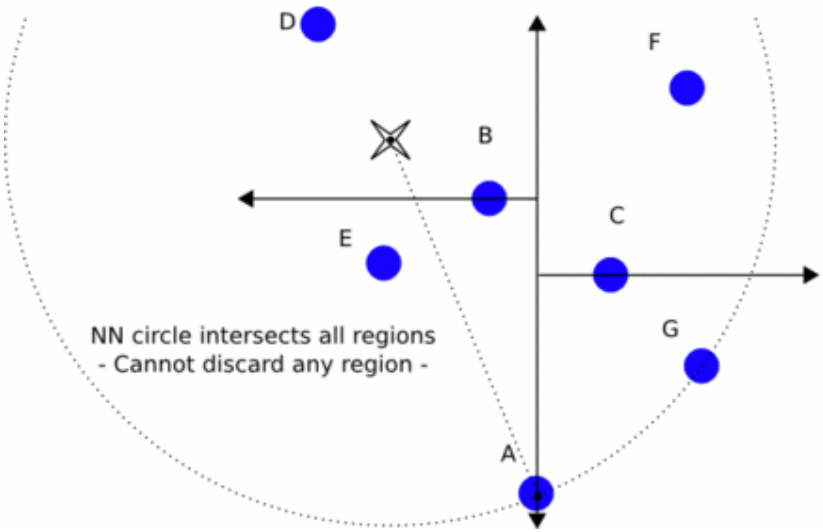
k-d tree algorithms

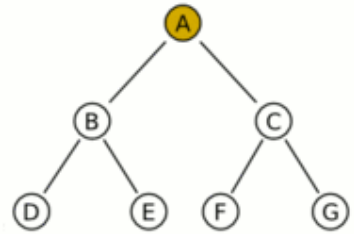
Median value

- Sorting in $O(n \log n)$
- ...would be inefficient
- Median approximation: sort a subsample

Complexity

	Average	**Worst**
Space	$O(n)$	$O(n)$
Search	$O(n \log n)$	$O(n)$
Insert	$O(n \log n)$	$O(n)$
Delete	$O(n \log n)$	$O(n)$





Start at A, then proceed in depth-first search (maintain a stack of parent-nodes if using a singly-linked tree). Set best estimate to A's distance Then examine left child node

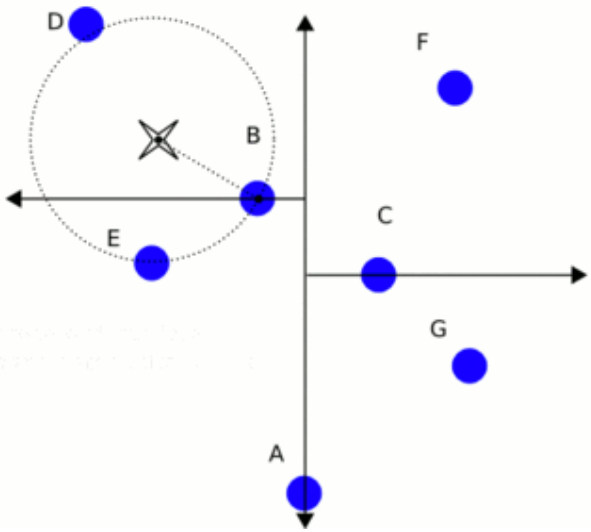
k-d tree algorithms

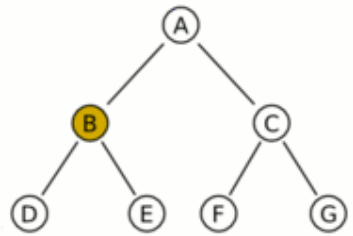
Median value

- Sorting in $O(n \log n)$
- ...would be inefficient
- Median approximation: sort a subsample

Complexity

	Average	Worst
Space	$O(n)$	$O(n)$
Search	$O(n \log n)$	$O(n)$
Insert	$O(n \log n)$	$O(n)$
Delete	$O(n \log n)$	$O(n)$





Calculate B's distance and
compare against best estimate
- It is smaller distance, so update
best estimate. Examine children (left then right)

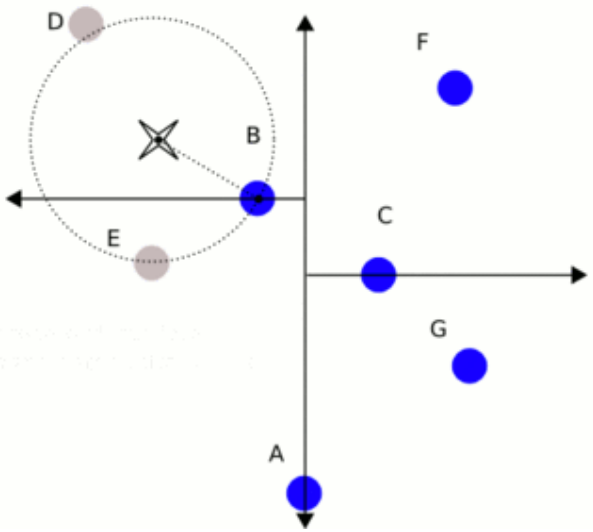
k-d tree algorithms

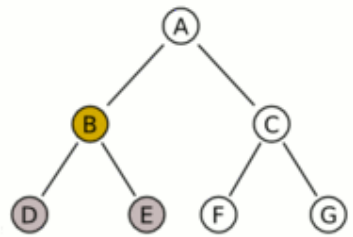
Median value

- Sorting in $O(n \log n)$
- ...would be inefficient
- Median approximation: sort a subsample

Complexity

	Average	**Worst**
Space	$O(n)$	$O(n)$
Search	$O(n \log n)$	$O(n)$
Insert	$O(n \log n)$	$O(n)$
Delete	$O(n \log n)$	$O(n)$





D & E Discarded as B
(already visited) is closer.
B is the best estimate for B's sub-branch
Proceed back to parent node

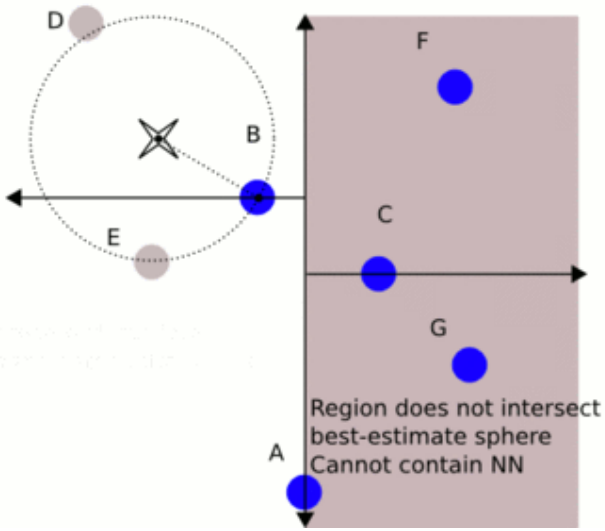
k-d tree algorithms

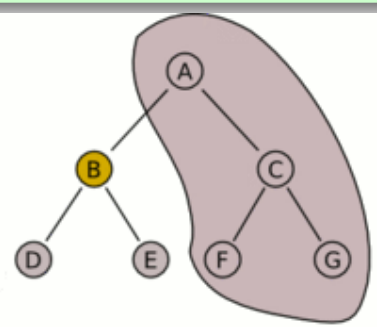
Median value

- Sorting in $O(n \log n)$
- ...would be inefficient
- Median approximation: sort a subsample

Complexity

	Average	**Worst**
Space	$O(n)$	$O(n)$
Search	$O(n \log n)$	$O(n)$
Insert	$O(n \log n)$	$O(n)$
Delete	$O(n \log n)$	$O(n)$





A's children have all been searched,
B is the best estimate for entire tree

k-d tree: improvements

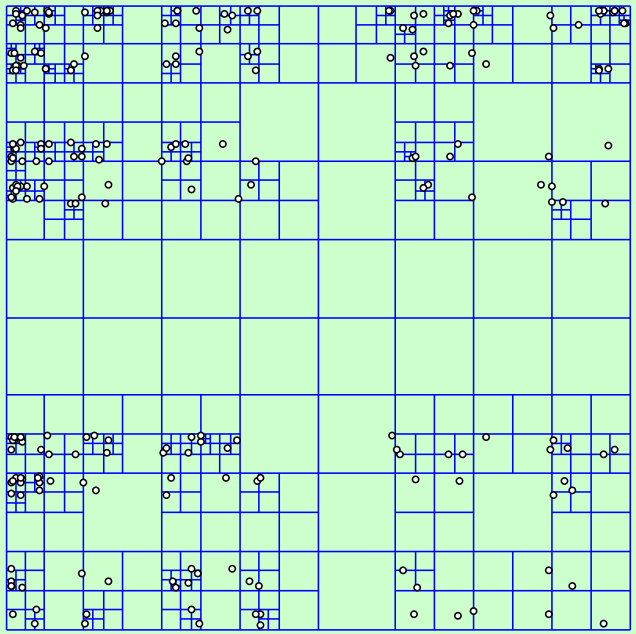
- Extension to k nearest neighbors
- Approximation: stop search after
  - some times
  - a number of branches
  - see best-bin-first search algorithm

Other trees

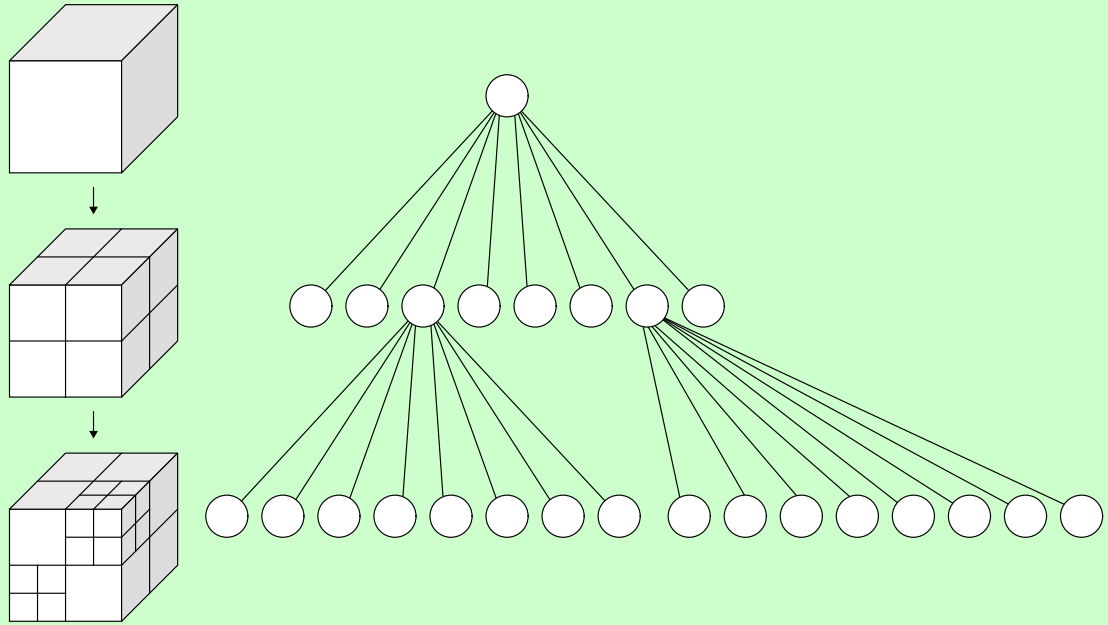
- quadtree: split into 4 children (2D)
- octree: split into 8 children (3D)
- Ball tree, a multi-dimensional space partitioning useful for nearest neighbor search
- R-tree and bounding interval hierarchy, for objects rather than points, with overlapping regions
- Vantage-point tree, a variant of a k-d tree that uses hyperspheres

Quadtree and Octree

Quadtree



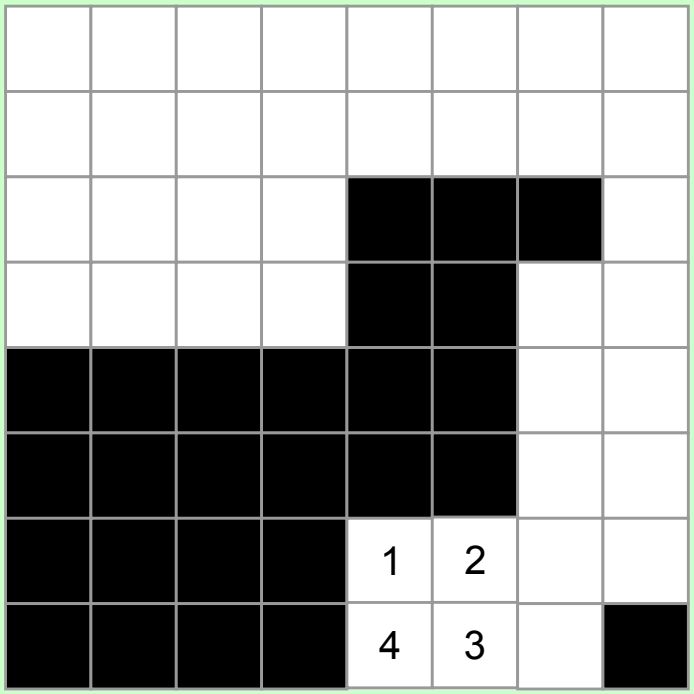
Octree

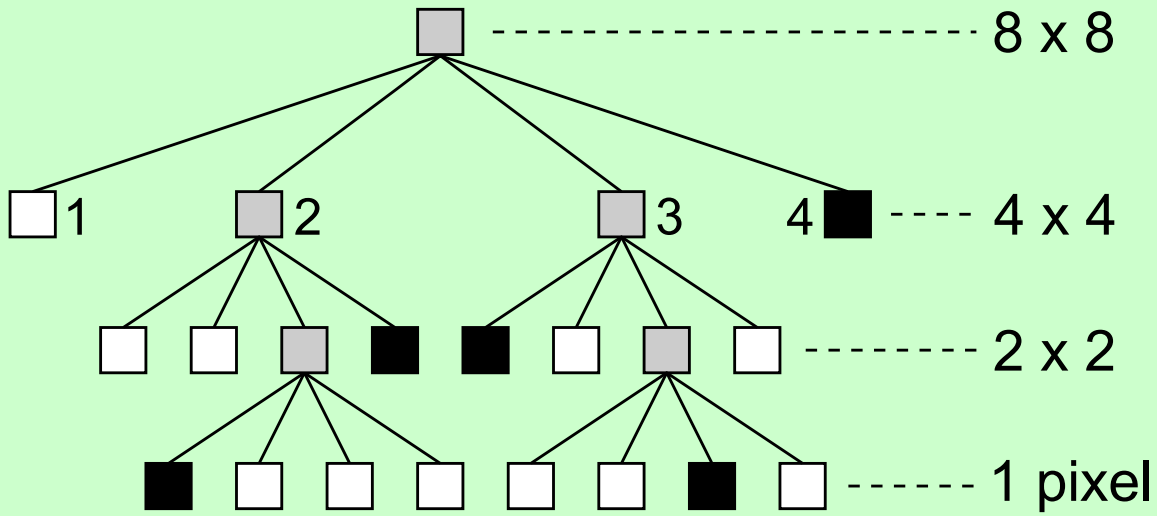


Quadtree usage

Bitmap

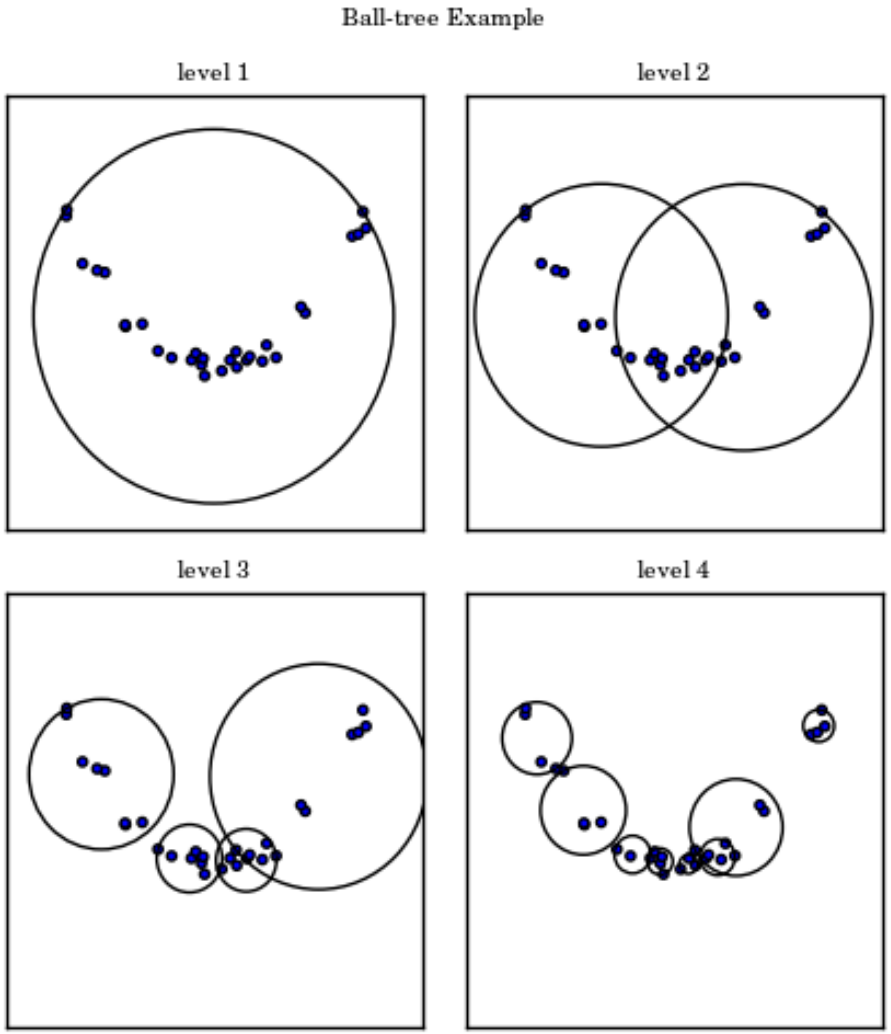
- compression
- segmentation: split/merge algorithm





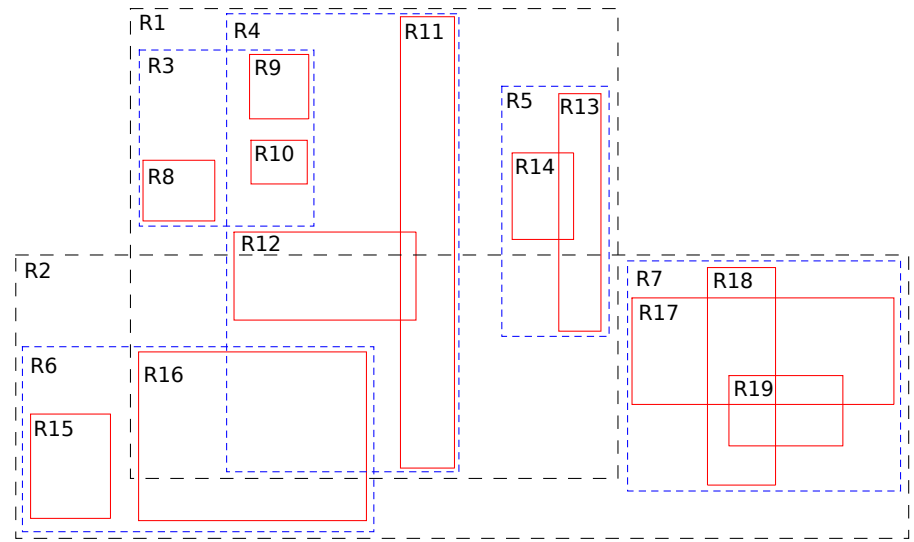
Ball tree (M-tree)

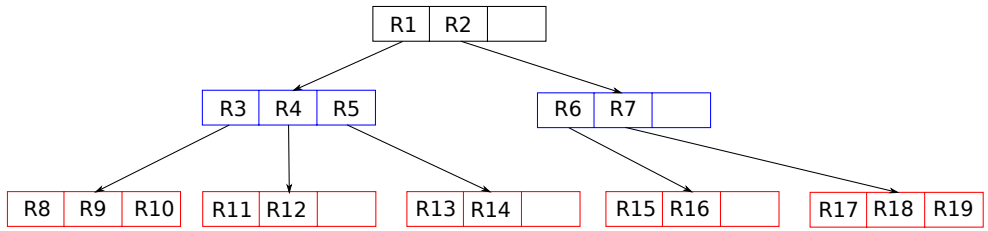
- M for Metric
- Use hyperspheres instead of hyperplanes



R tree

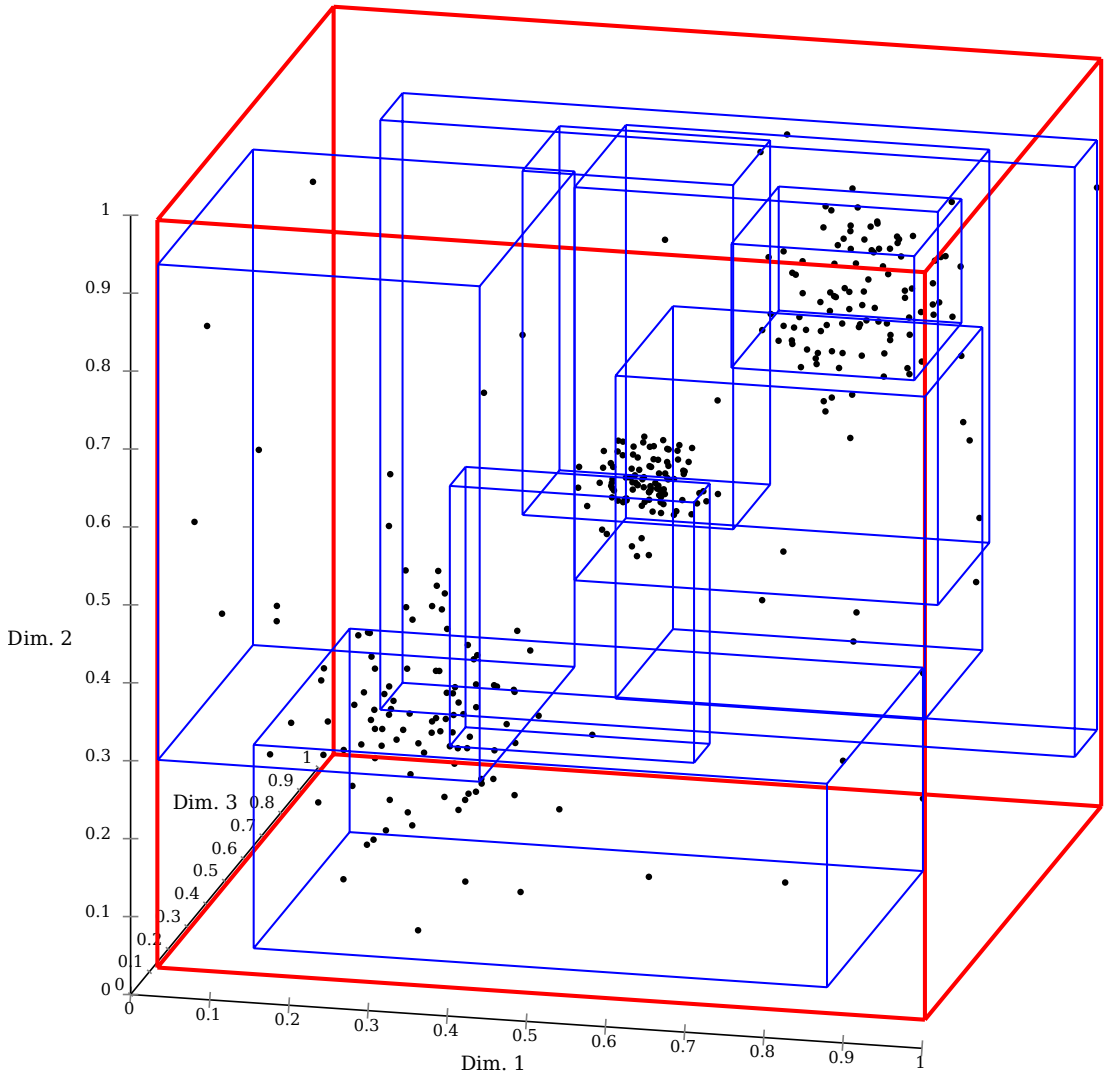
- groups objects
- rectangle bounding boxes





R tree

- groups objects
- rectangle bounding boxes



Vantage-Point tree (VP)

- partitions the space with spheres
- given a point (VP)
- and distances
- may be faster than k-d for high dimensions

Section 3

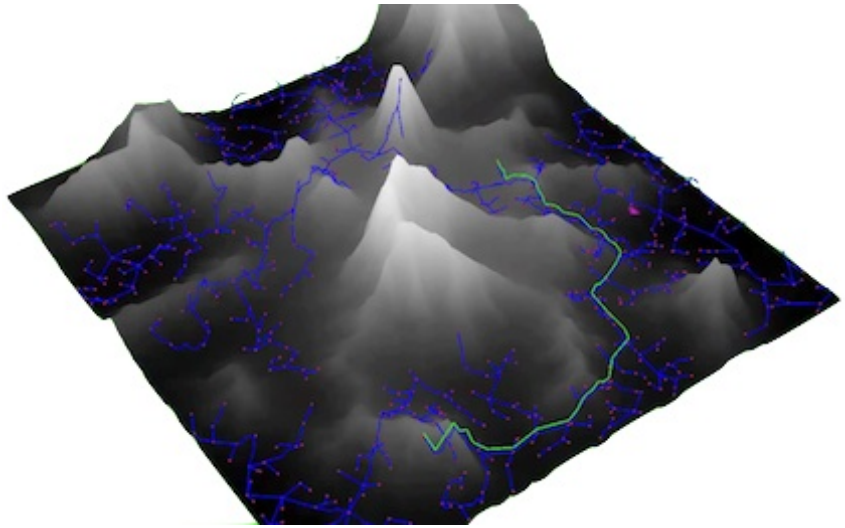
Voronoi Diagram

References

- [Devadoss and O'Rourke, 2011]
- [de Berg, 1997]

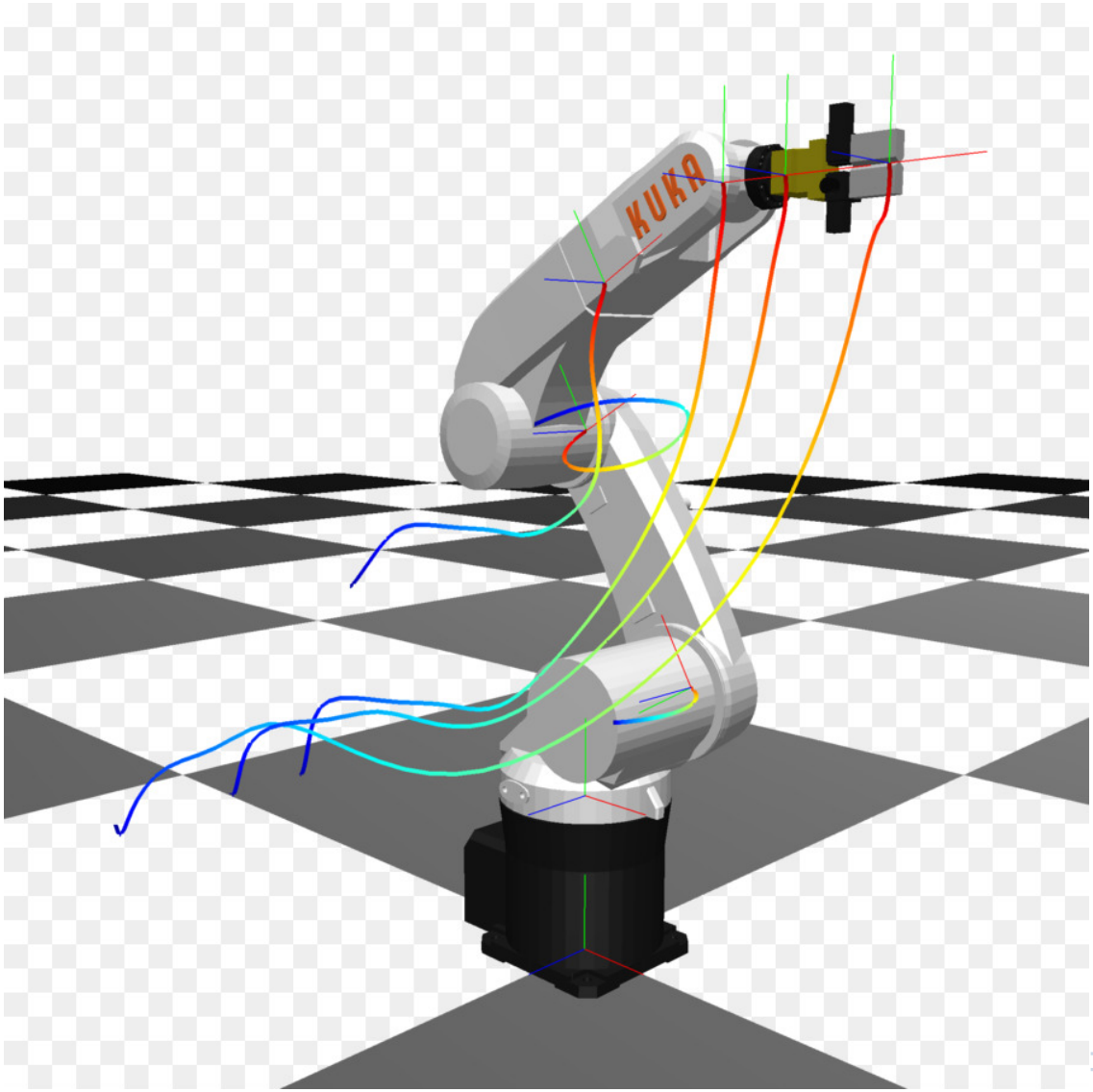
Illustrations and application domains

Motion planning



Illustrations and application domains

Motion planning: robotics

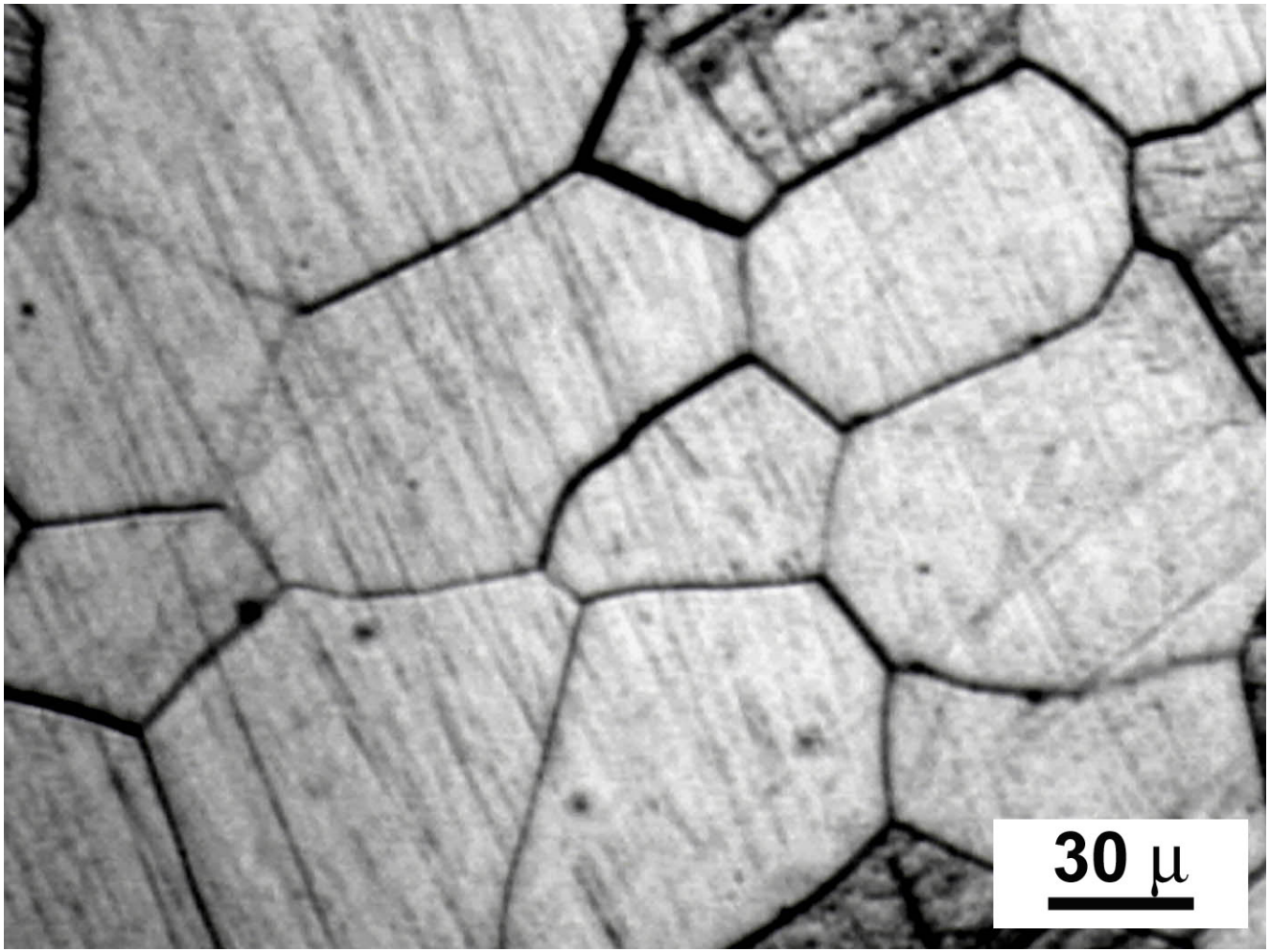


Illustrations and application domains

Post-office problem: delivery routes

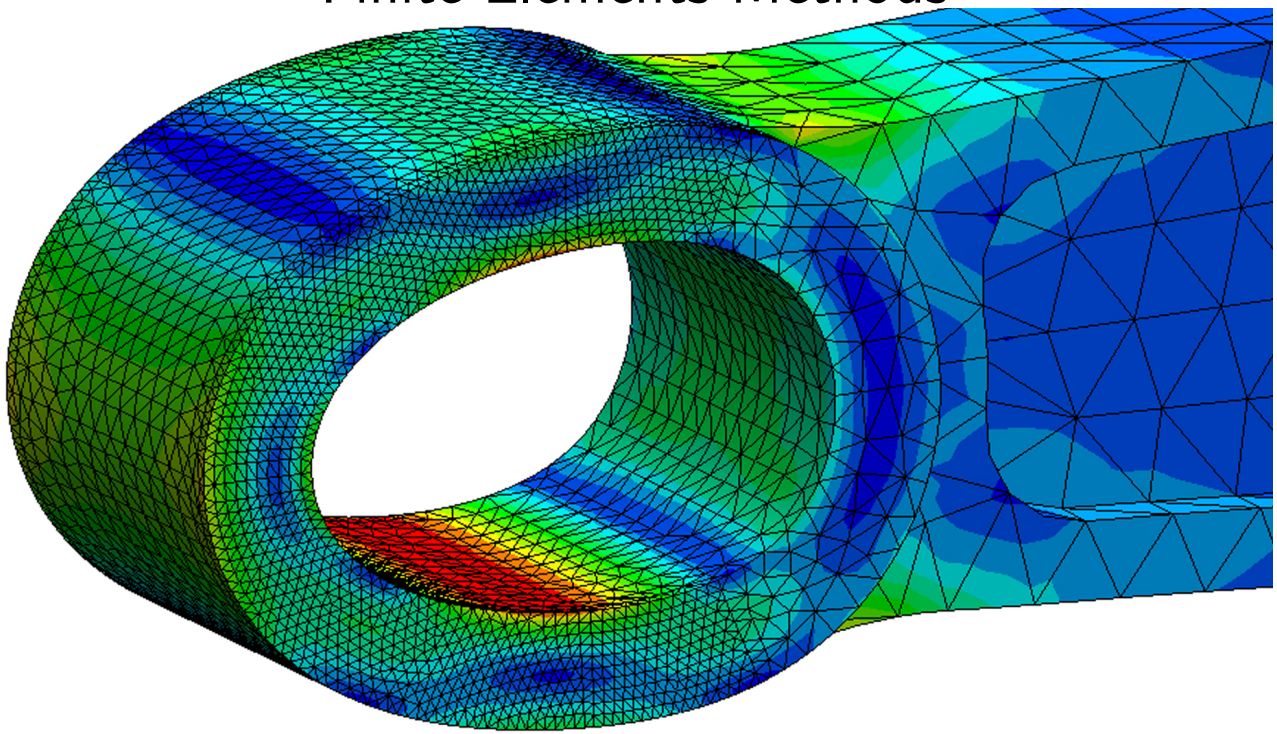
Illustrations and application domains

cristallography



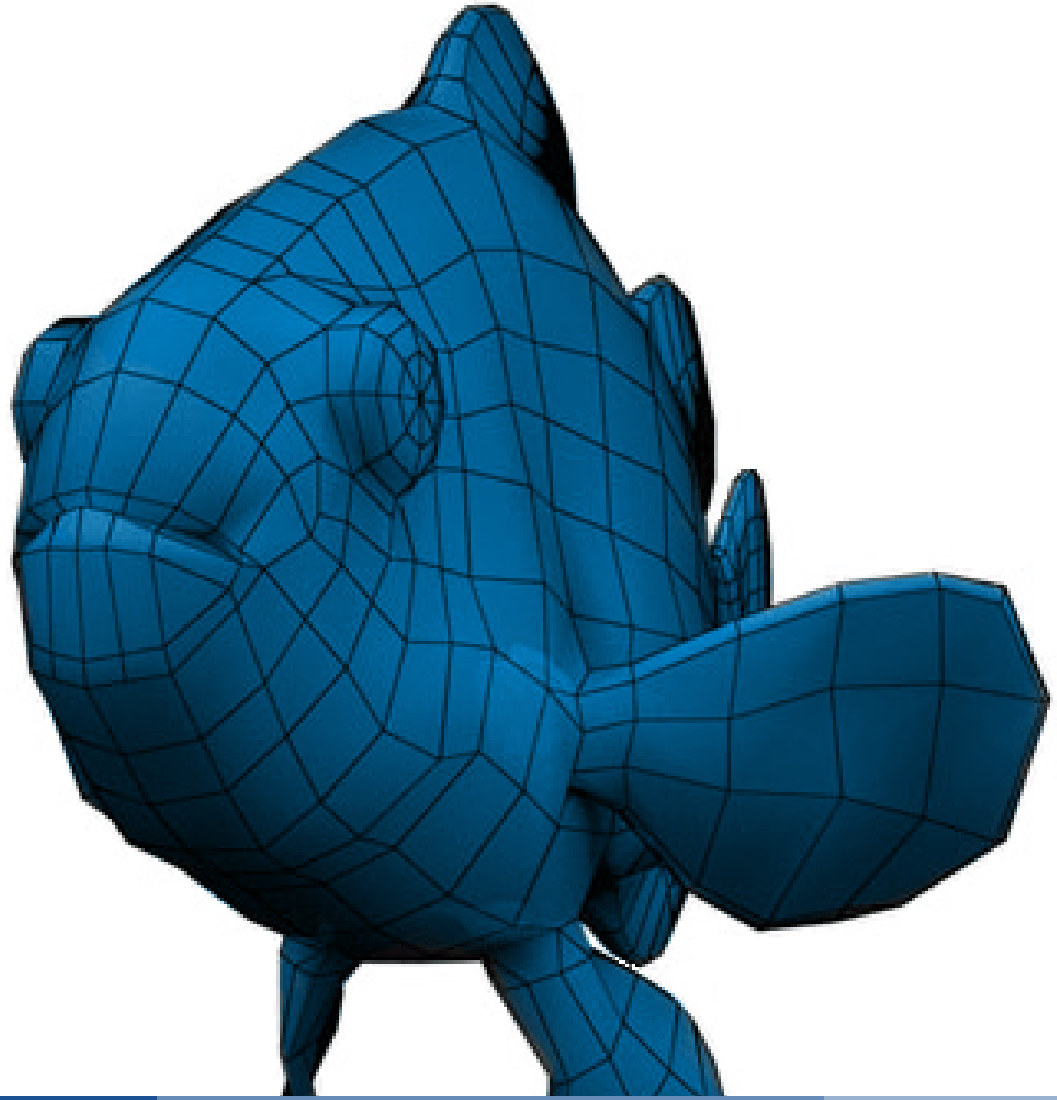
Illustrations and application domains

Finite Elements Methods



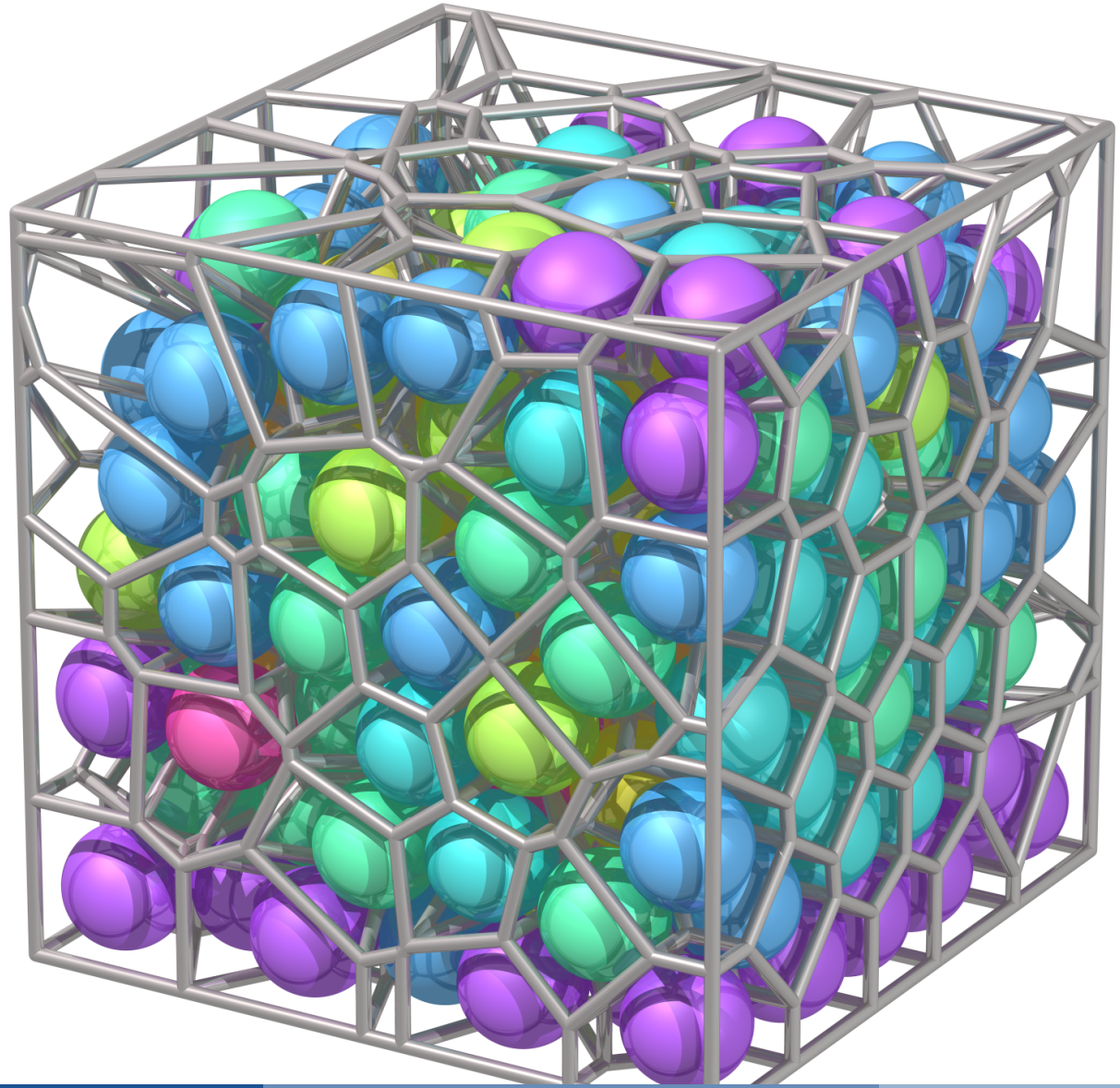
Illustrations and application domains

Meshing for video games and animations



Illustrations and application domains

Illustrations and application domains



Illustrations and application domains

Plan your trip to Venice

Voronoi diagram

- X is a metric space, with distance d.
- points (aka sites) $P_k \in X$, $k \in K$
- $S = \{P_k, k \in K\}$
- $d(x, A) = \inf\{d(x, a) \mid a \in A\}$ denotes the distance between the point x and the set A.

Definition (Voronoi region)

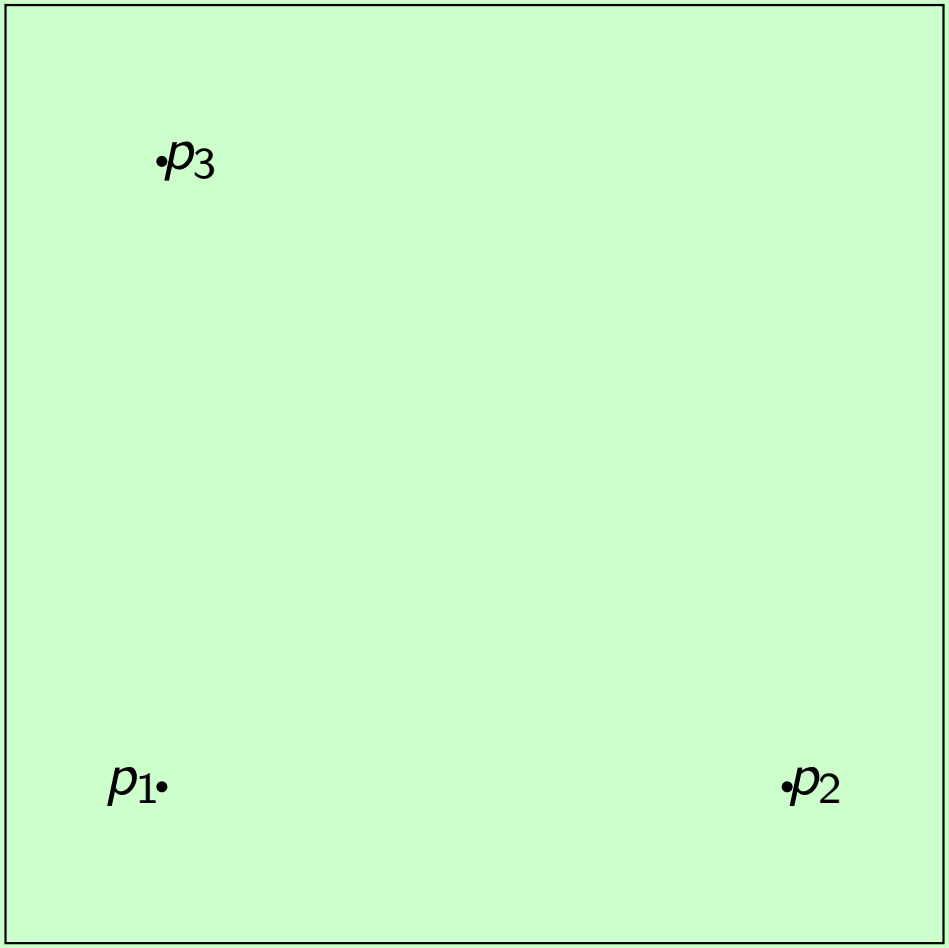
$$\mathrm{Vor}_S(P_k) = R_k = \{x \in X \mid d(x, P_k) \leq d(x, P_j)\ \forall j \neq k\}$$

Definition (Voronoi diagram)

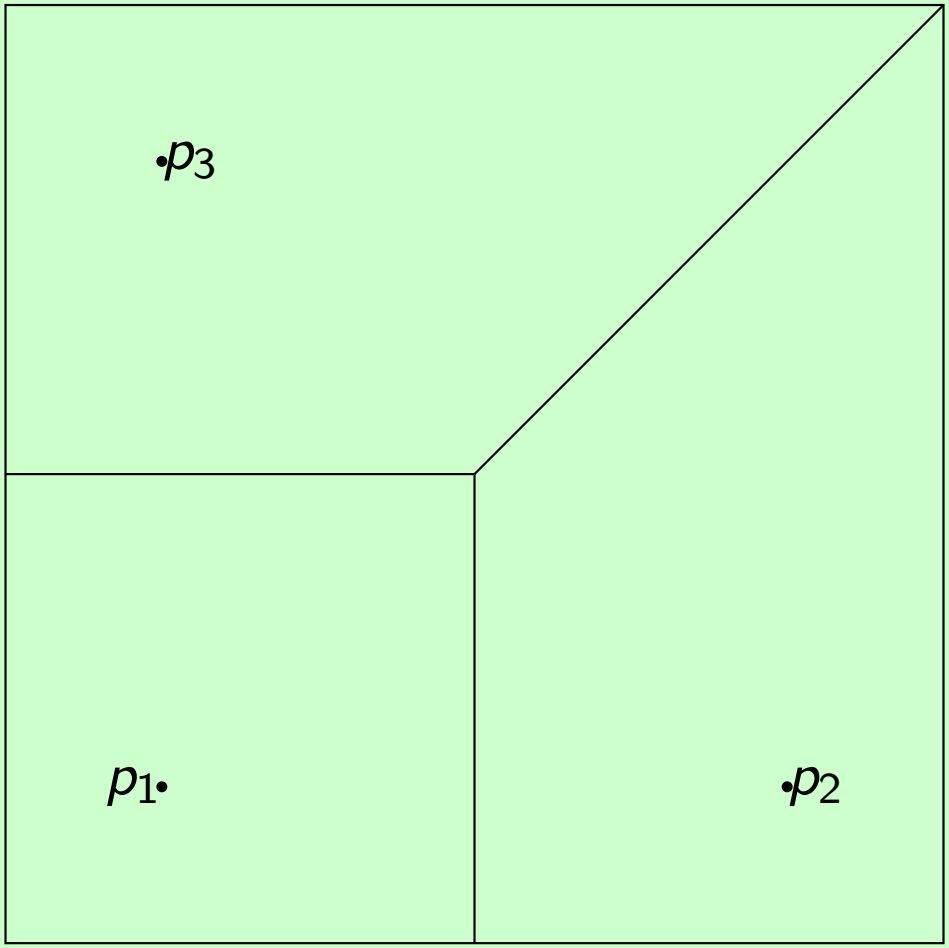
Set of all regions

$$\mathrm{Vor} = (R_k)_{k \in K}$$

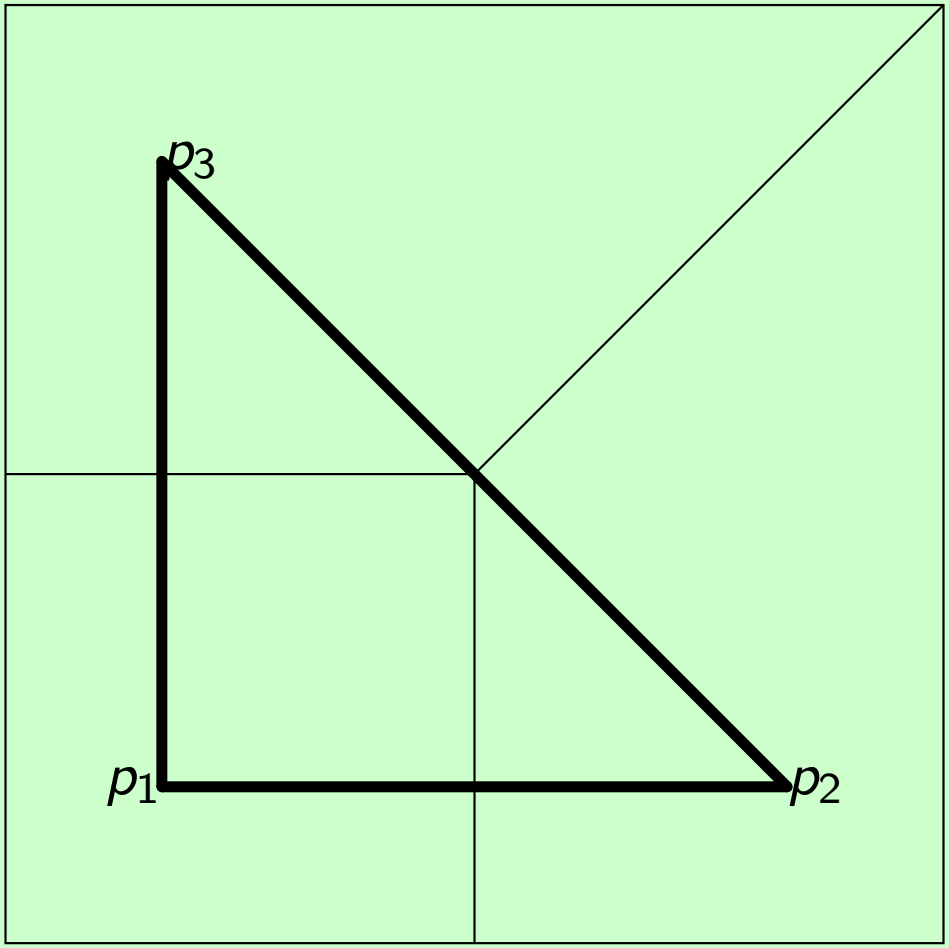
Illustration



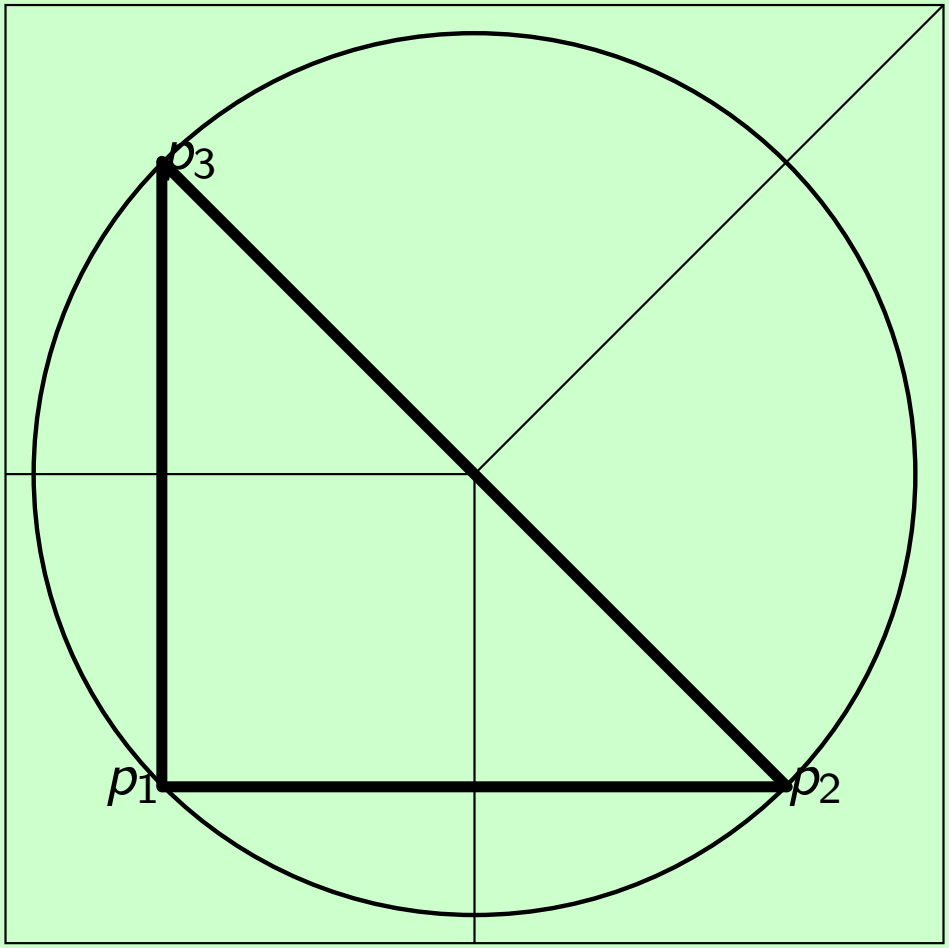
Illustration



Illustration

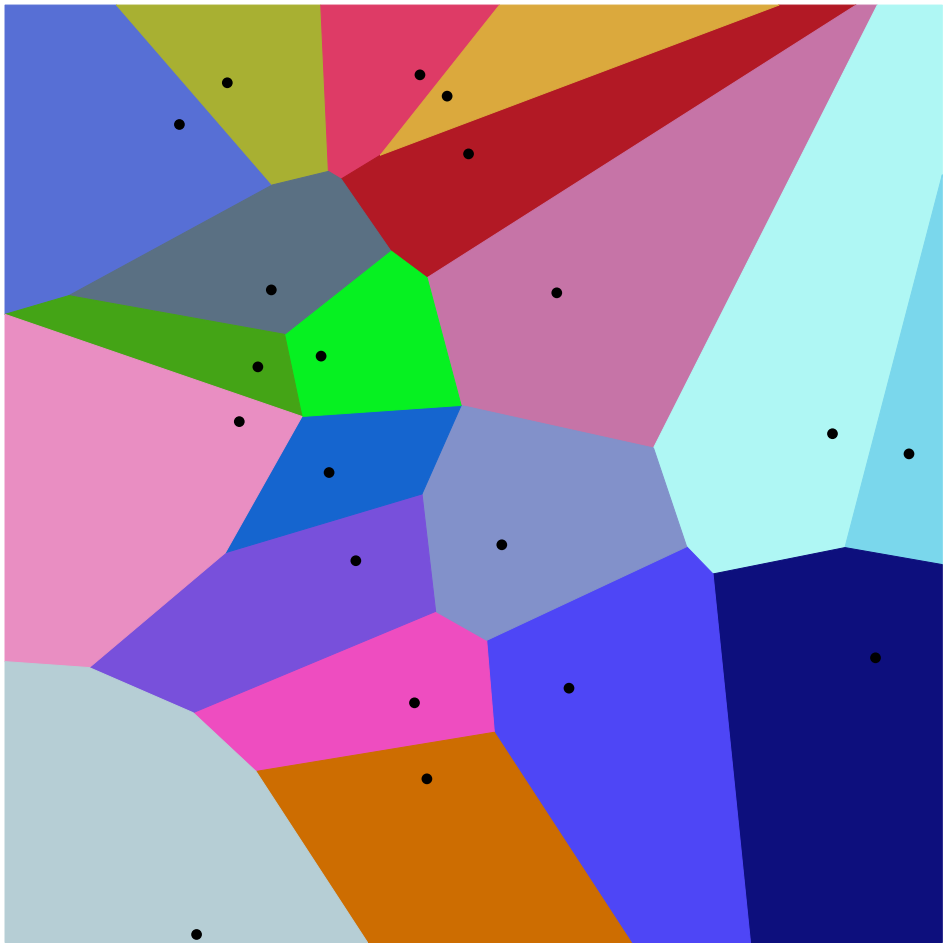


Illustration



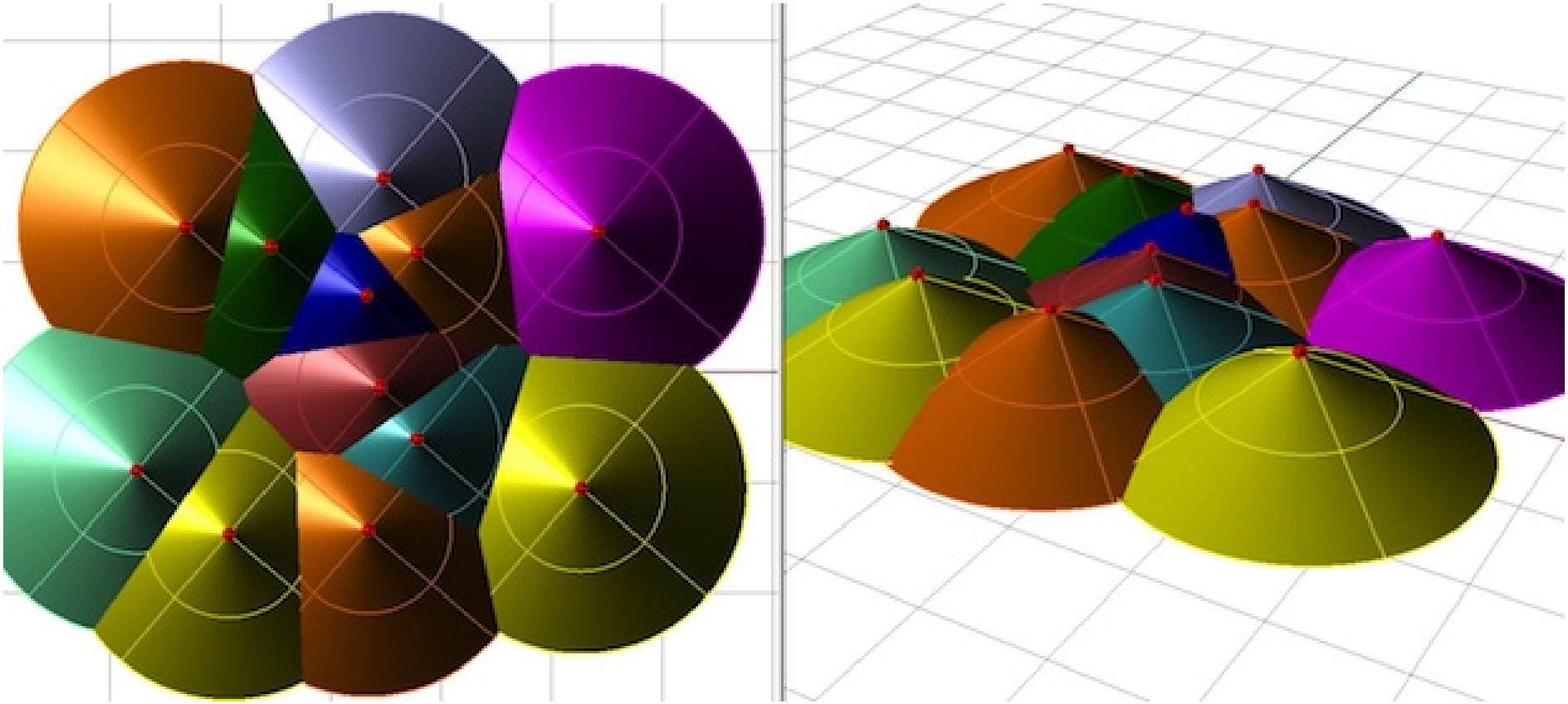
2D examples

Euclidean distance



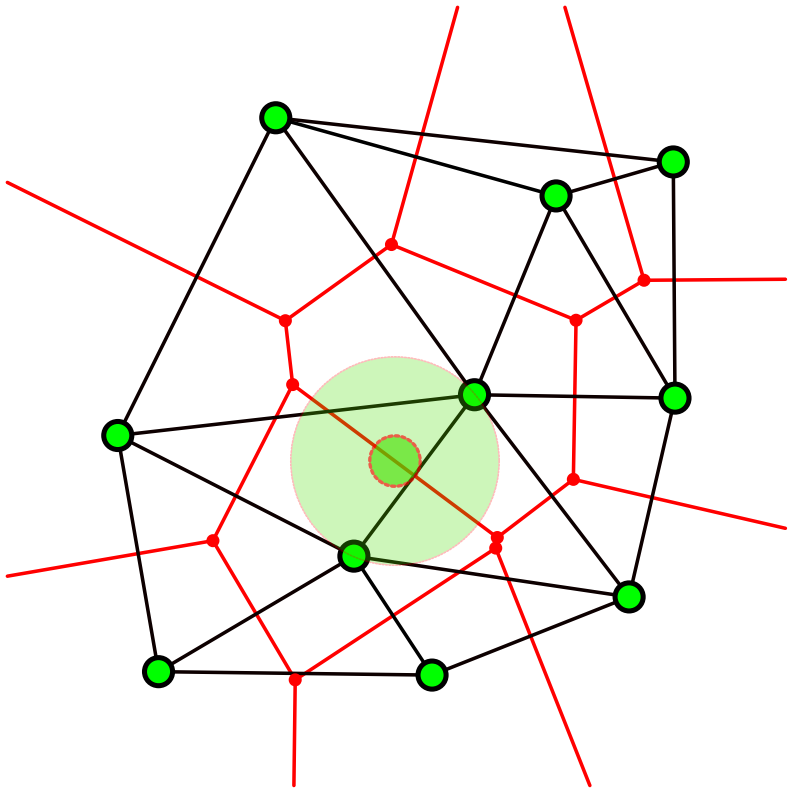
Manhattan distance

Intersection of cones



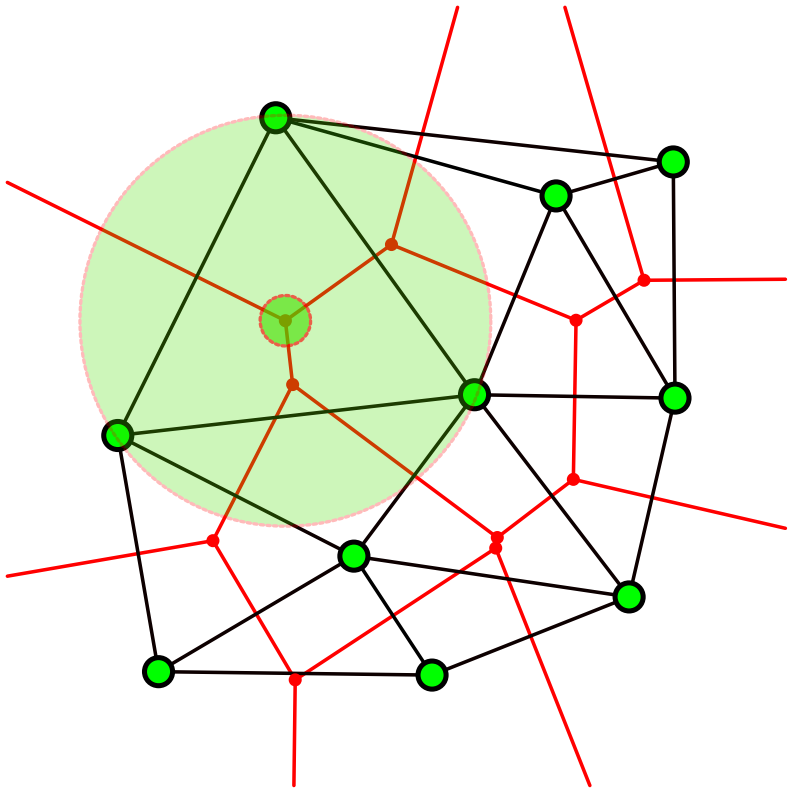
Some properties

- A point in an edge is at equal distance of 2 points in S



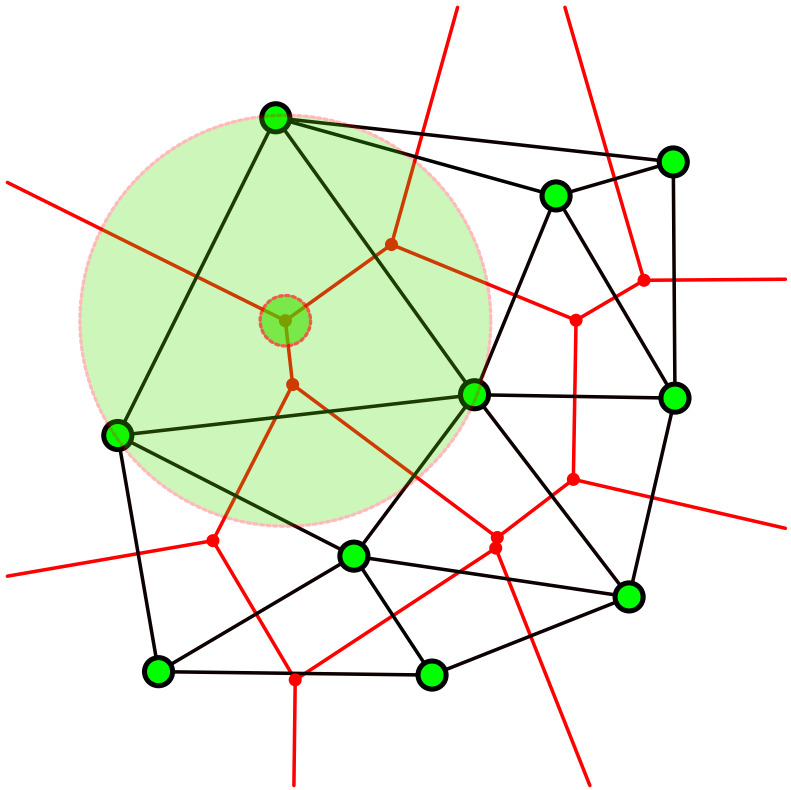
Some properties

- A point in an edge is at equal distance of 2 points in S
- A point in a vertex is at equal distance of 3 points in S



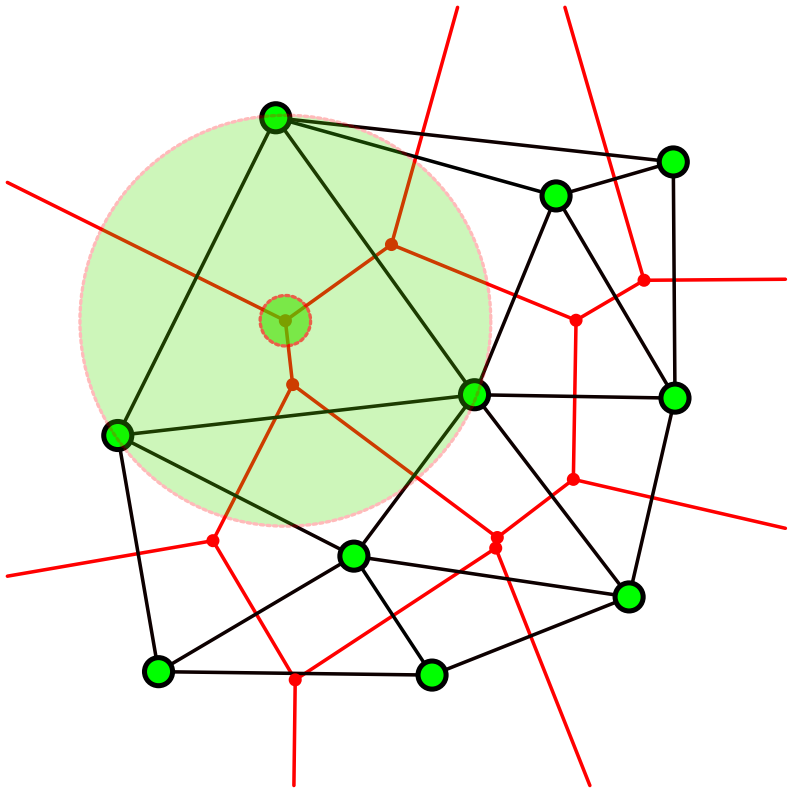
Some properties

- A point in an edge is at equal distance of 2 points in S
- A point in a vertex is at equal distance of 3 points in S
- The closest pair of points corresponds to two adjacent cells in the Voronoi diagram.



Some properties

- A point in an edge is at equal distance of 2 points in S
- A point in a vertex is at equal distance of 3 points in S
- The closest pair of points corresponds to two adjacent cells in the Voronoi diagram.
- Generally (Euclidean distance, unique points...) two points are adjacent on the convex hull if and only if their Voronoi cells share an infinitely long side.



Euler Number

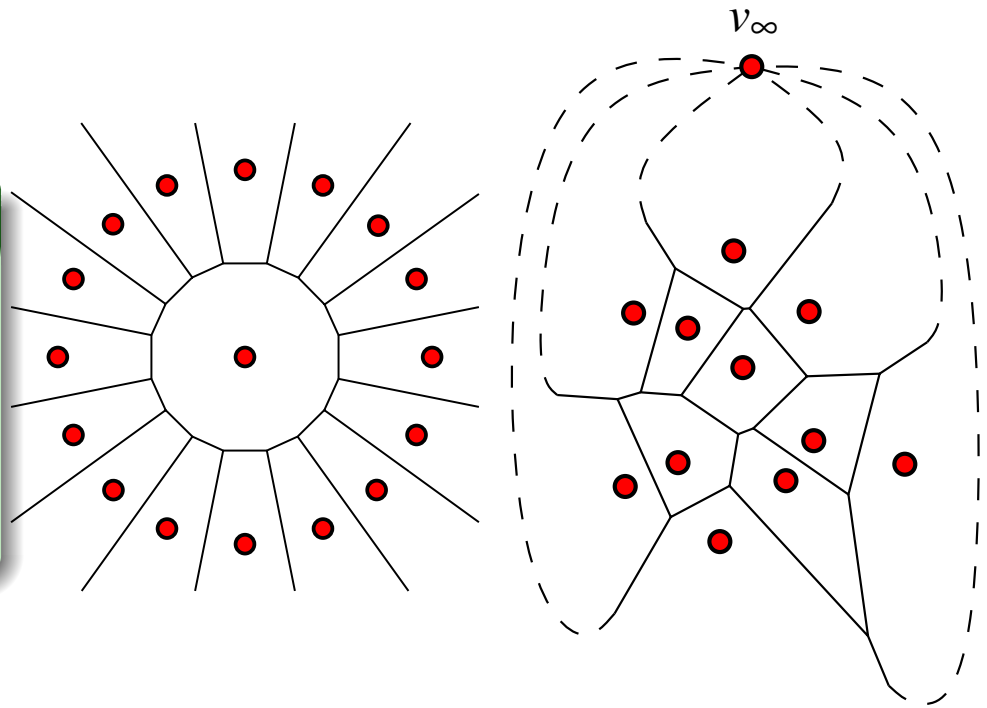
Theorem

For $n \geq 3$, the number of vertices in the Voronoi diagram of n points in the plane is at most $2n - 5$, and the number of edges is at most $3n - 6$.

Element of proof

consider a vertex at infinity so that Euler theorem can apply.

$$(n_v + 1) - n_e + n = 2$$



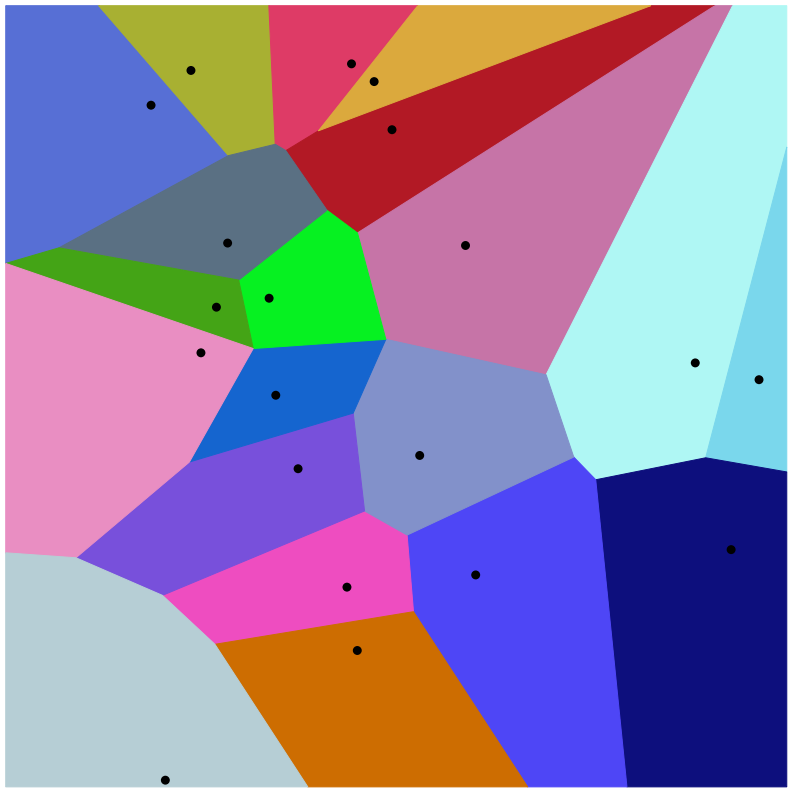
Algorithm from affine half-plane

Definition (Affine (bisector) hyperplane)

$$\mathrm{H}(p,q) = \{x \in \mathbb{R}^2 / d(x,p) \leq d(x,q)\}$$

$$\mathrm{Vor}_S(p) = \bigcap_{q \in \mathrm{S} \setminus \{p\}} \mathrm{H}(p,q)$$

- For a point, construct all half-planes
- Compute the intersection
- $O(n^2)$

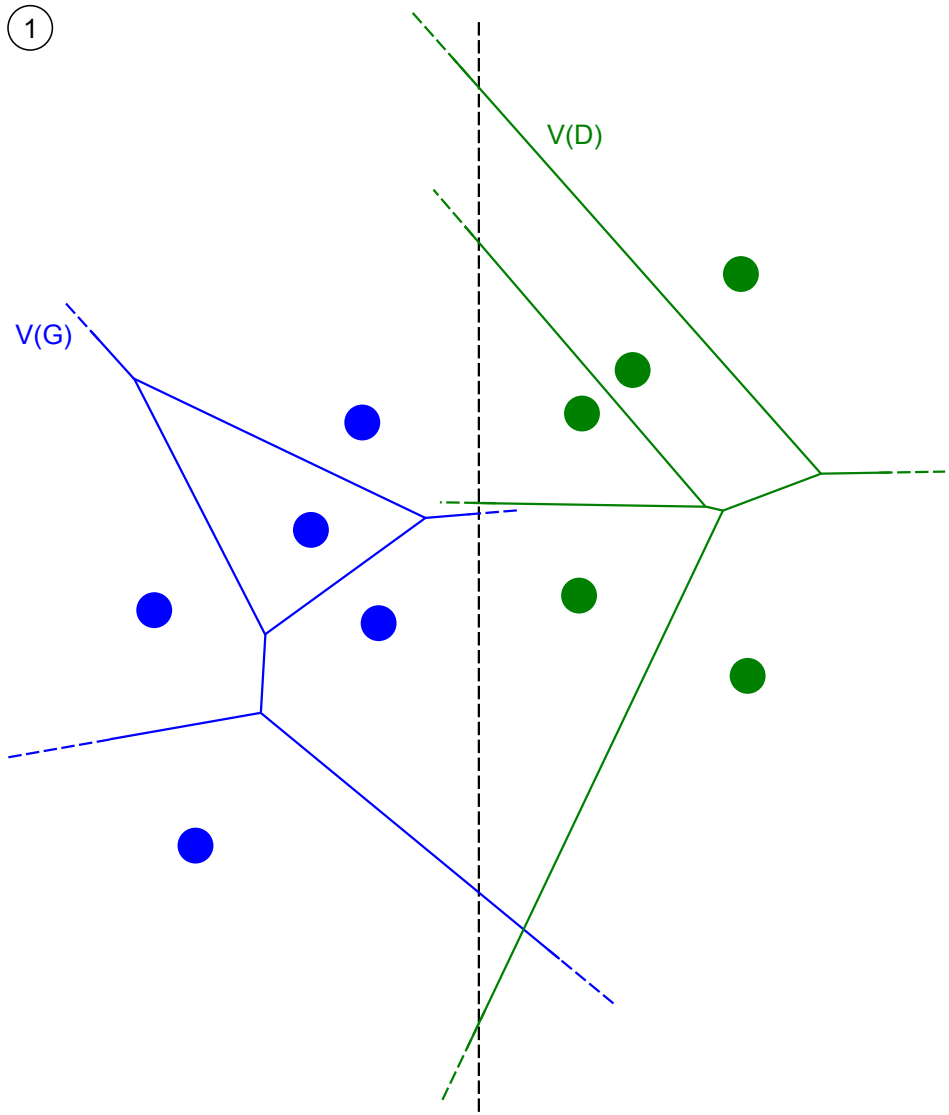


Shamos et Hoey, 1975

Divide and conquer algorithm

- Separate in 2 sets
- Compute Voronoi Diagram
- Compute convex hull
- Merge diagrams

$O(n \log n)$, but too hard to code!

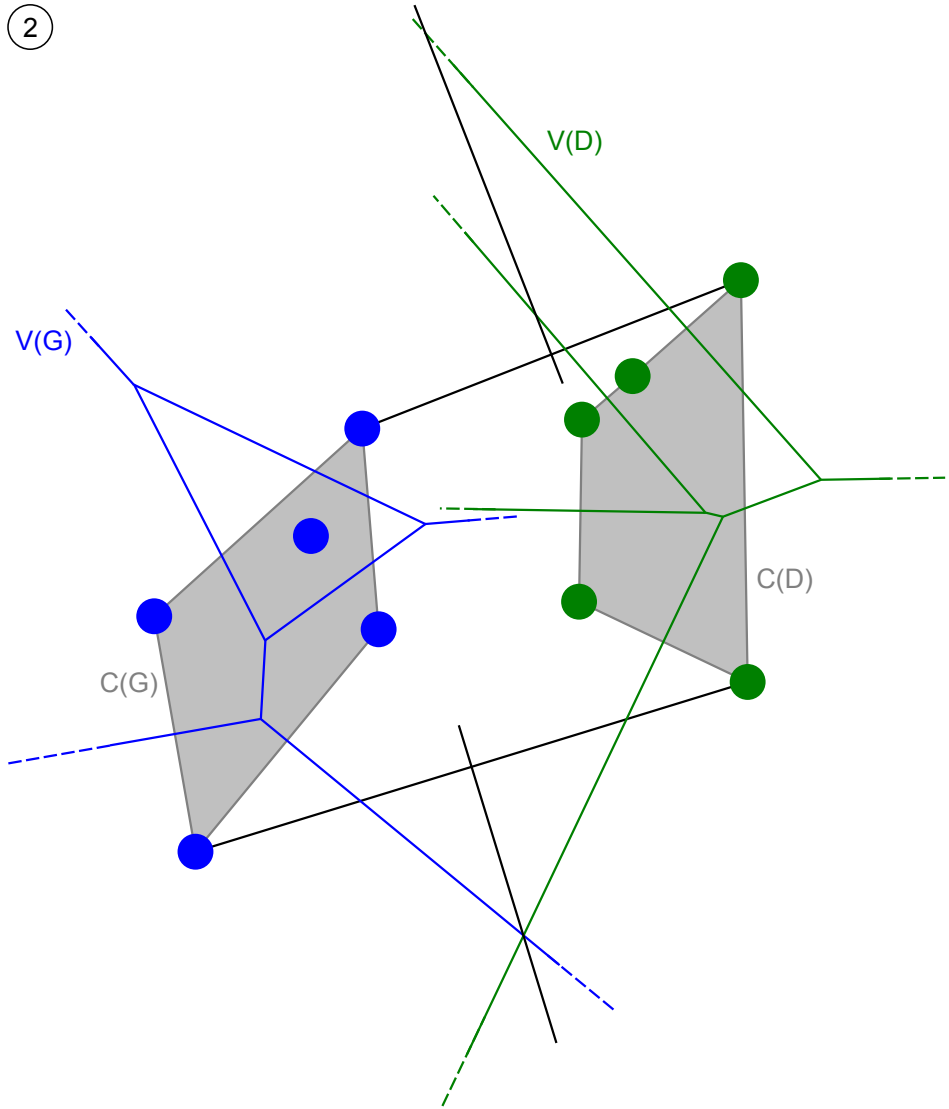


Shamos et Hoey, 1975

Divide and conquer algorithm

- Separate in 2 sets
- Compute Voronoi Diagram
- Compute convex hull
- Merge diagrams

$O(n \log n)$, but too hard to code!

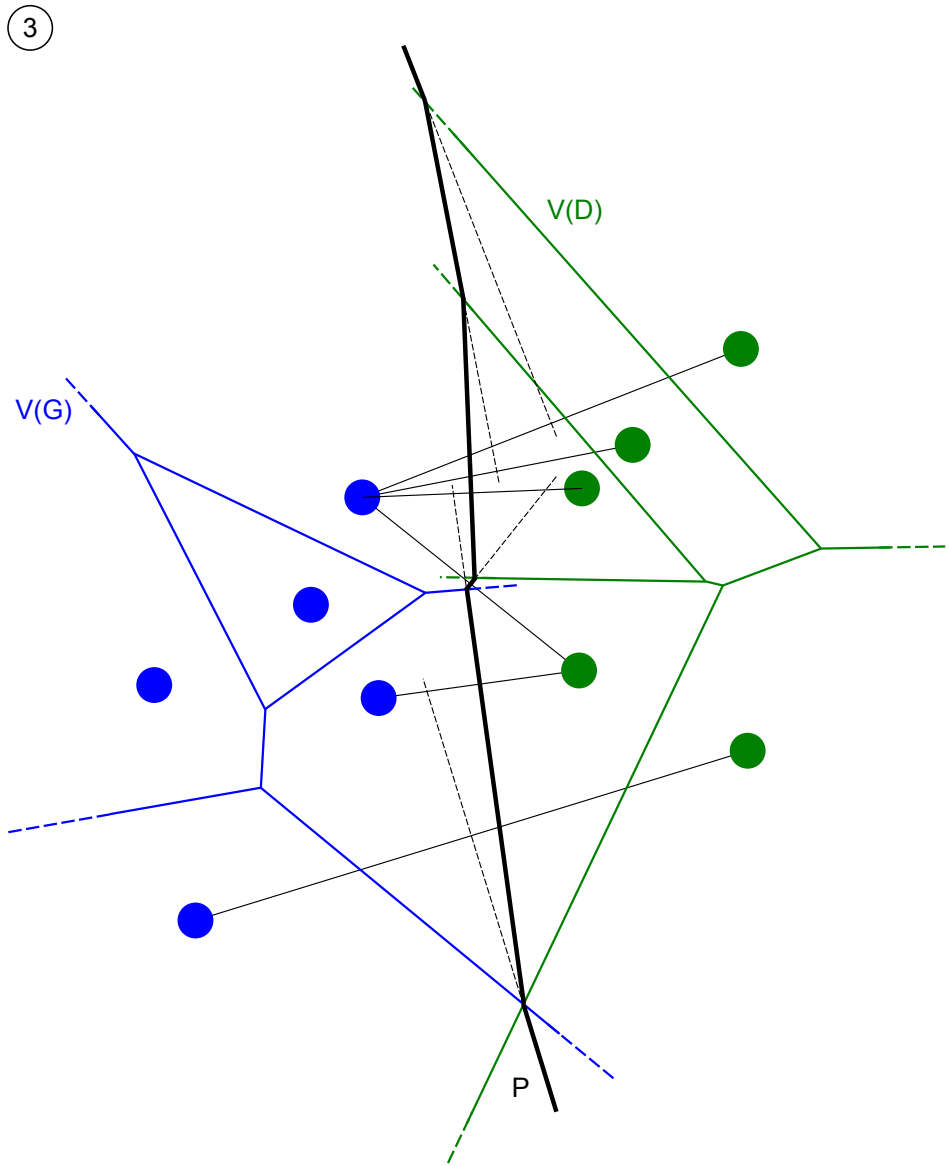


Shamos et Hoey, 1975

Divide and conquer algorithm

- Separate in 2 sets
- Compute Voronoi Diagram
- Compute convex hull
- Merge diagrams

$O(n \log n)$, but too hard to code!

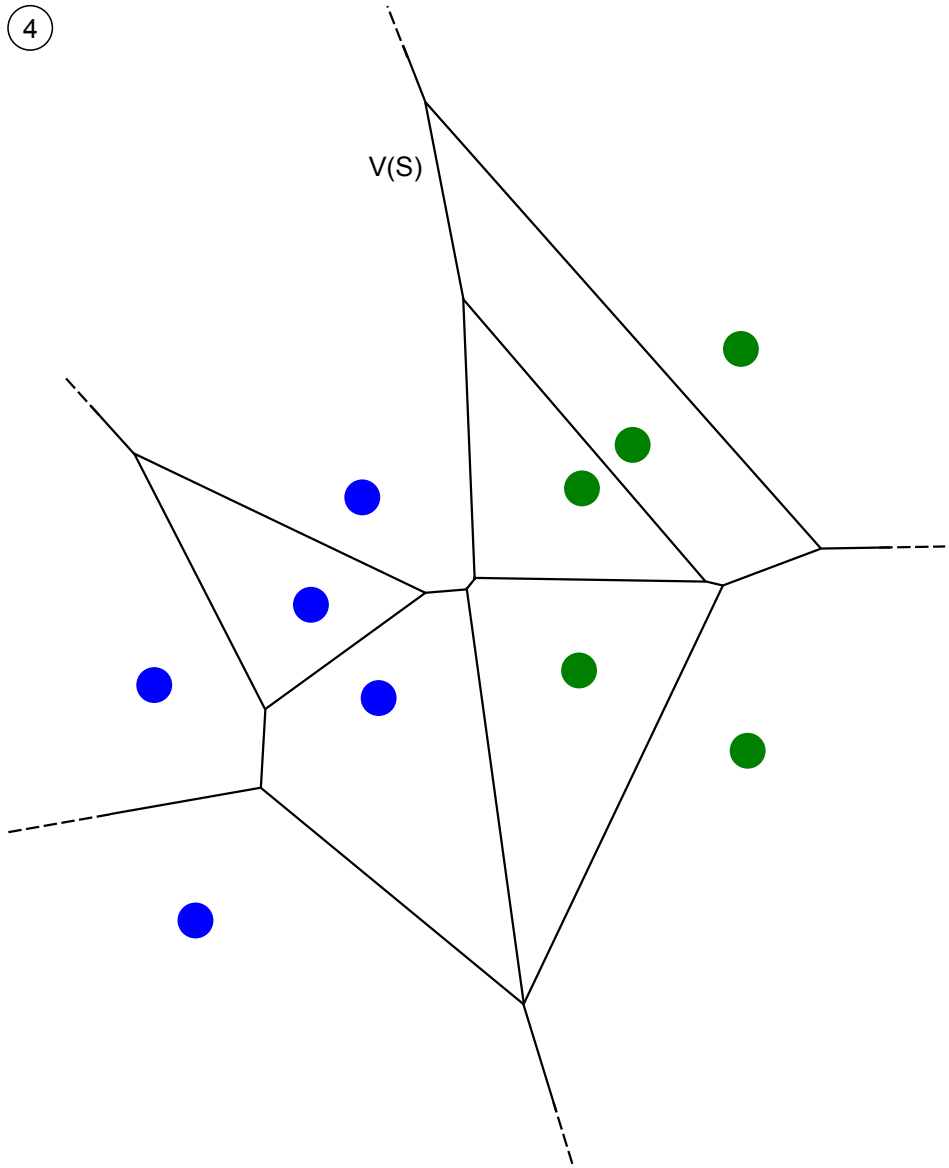


Shamos et Hoey, 1975

Divide and conquer algorithm

- Separate in 2 sets
- Compute Voronoi Diagram
- Compute convex hull
- Merge diagrams

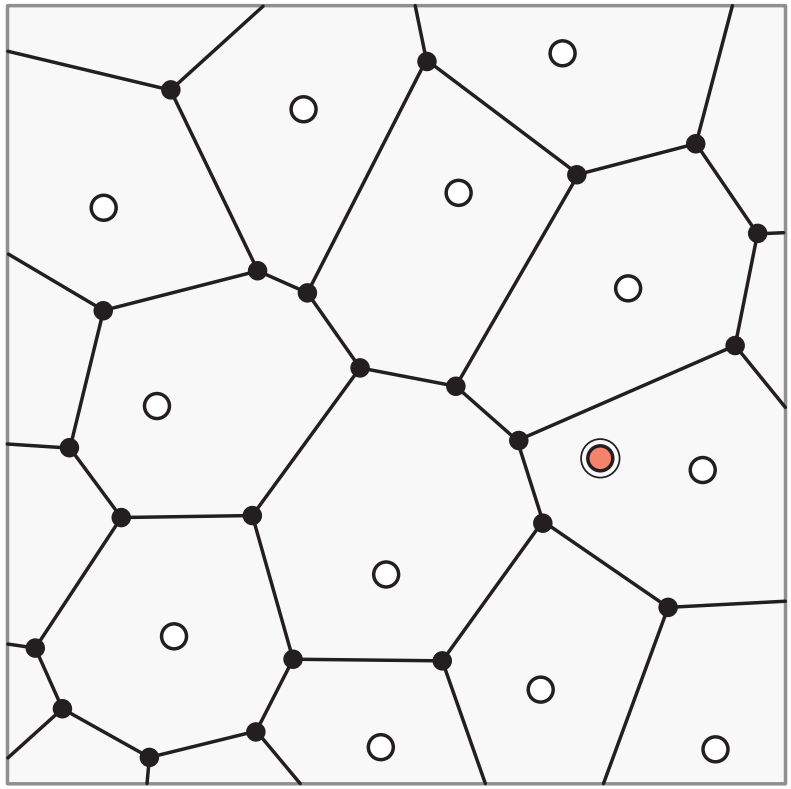
$O(n \log n)$, but too hard to code!



Green et Sibson, 1978

- Incremental algorithm
- Key idea: a new point will modify the diagram locally
- $O(n^2)$

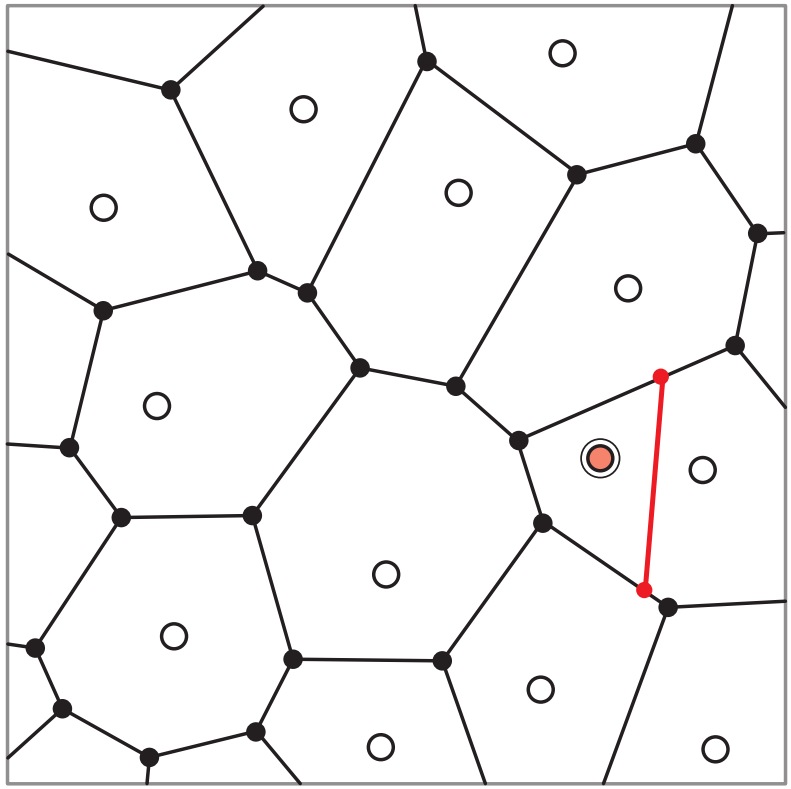
- Find the cell $\mathrm{Vor}_S(p_n)$ containing the new point q
- Compute bisector of $[q; p_n]$
- Find intersections with the cell
- Iterate from cells touching this intersection



Green et Sibson, 1978

- Incremental algorithm
- Key idea: a new point will modify the diagram locally
- $O(n^2)$

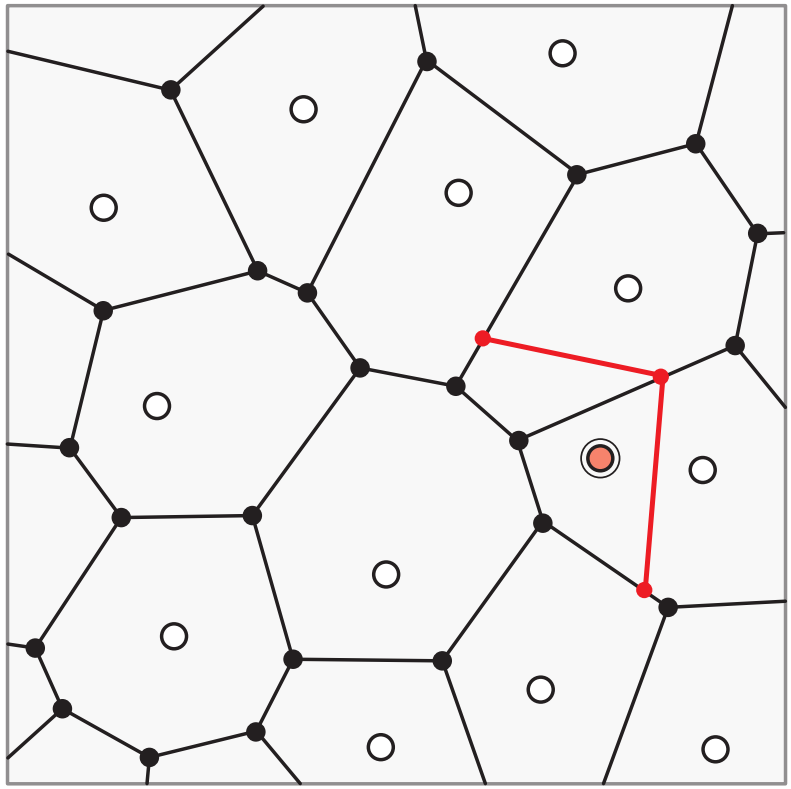
- Find the cell $\mathrm{Vor}_S(p_n)$ containing the new point q
- Compute bisector of $[q; p_n]$
- Find intersections with the cell
- Iterate from cells touching this intersection



Green et Sibson, 1978

- Incremental algorithm
- Key idea: a new point will modify the diagram locally
- $O(n^2)$

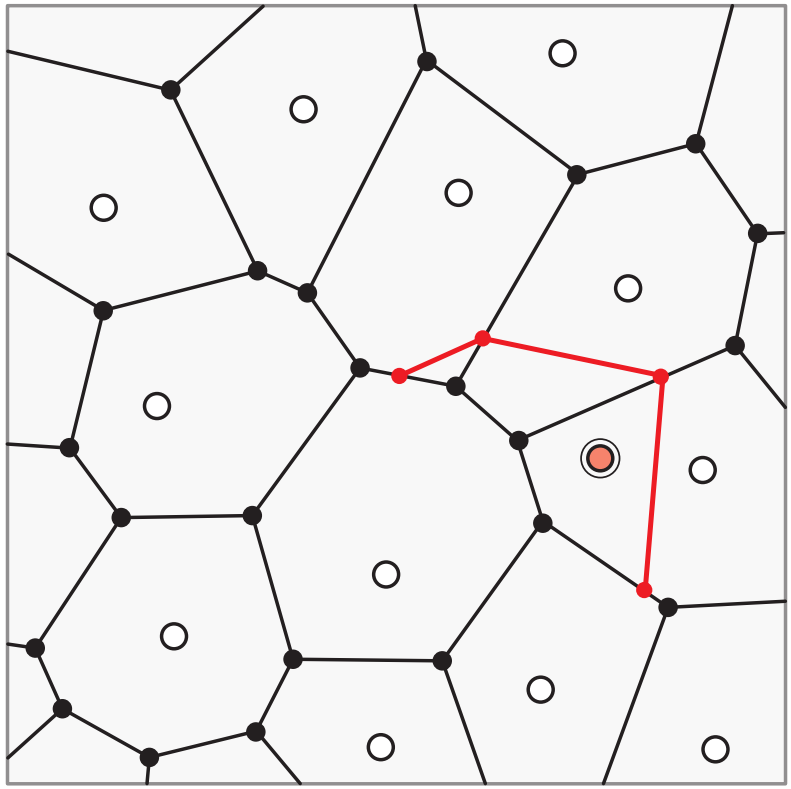
- Find the cell $\mathrm{Vor}_S(p_n)$ containing the new point q
- Compute bisector of $[q; p_n]$
- Find intersections with the cell
- Iterate from cells touching this intersection



Green et Sibson, 1978

- Incremental algorithm
- Key idea: a new point will modify the diagram locally
- $O(n^2)$

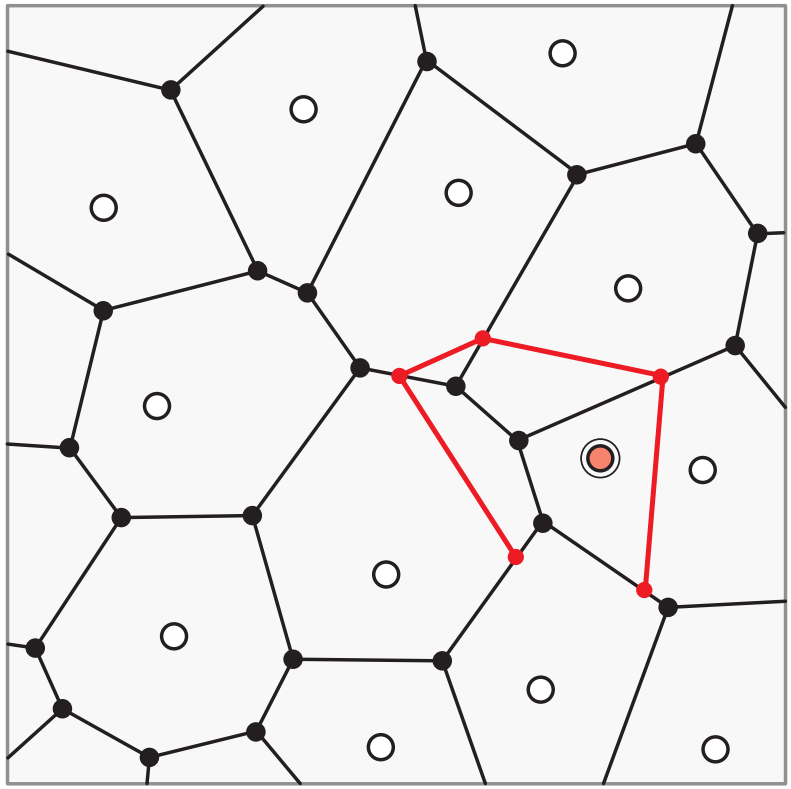
- Find the cell $\mathrm{Vor}_S(p_n)$ containing the new point q
- Compute bisector of $[q; p_n]$
- Find intersections with the cell
- Iterate from cells touching this intersection



Green et Sibson, 1978

- Incremental algorithm
- Key idea: a new point will modify the diagram locally
- $O(n^2)$

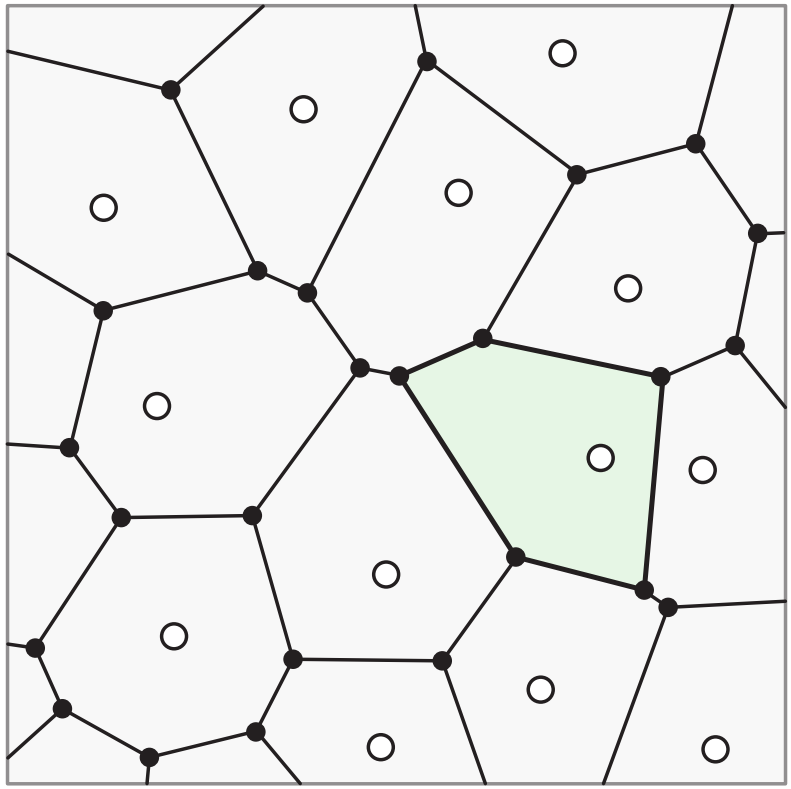
- Find the cell $\mathrm{Vor}_S(p_n)$ containing the new point q
- Compute bisector of $[q; p_n]$
- Find intersections with the cell
- Iterate from cells touching this intersection



Green et Sibson, 1978

- Incremental algorithm
- Key idea: a new point will modify the diagram locally
- $O(n^2)$

- Find the cell $\mathrm{Vor}_S(p_n)$ containing the new point q
- Compute bisector of $[q; p_n]$
- Find intersections with the cell
- Iterate from cells touching this intersection

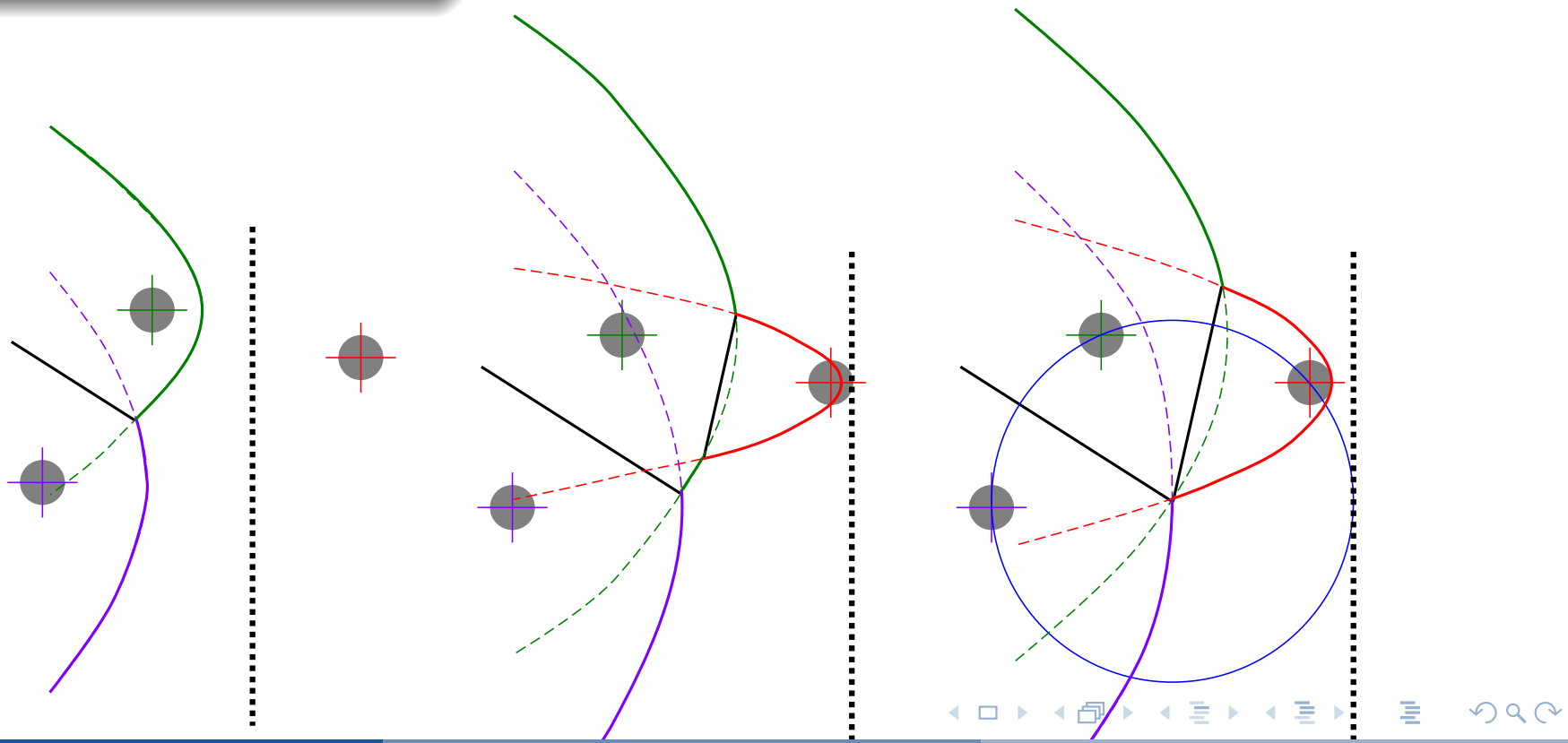


Fortune's algorithm, 1987

- Sweep line $\mathcal{L}$
- Beach curve $\mathcal{B}$
- Compute intersections

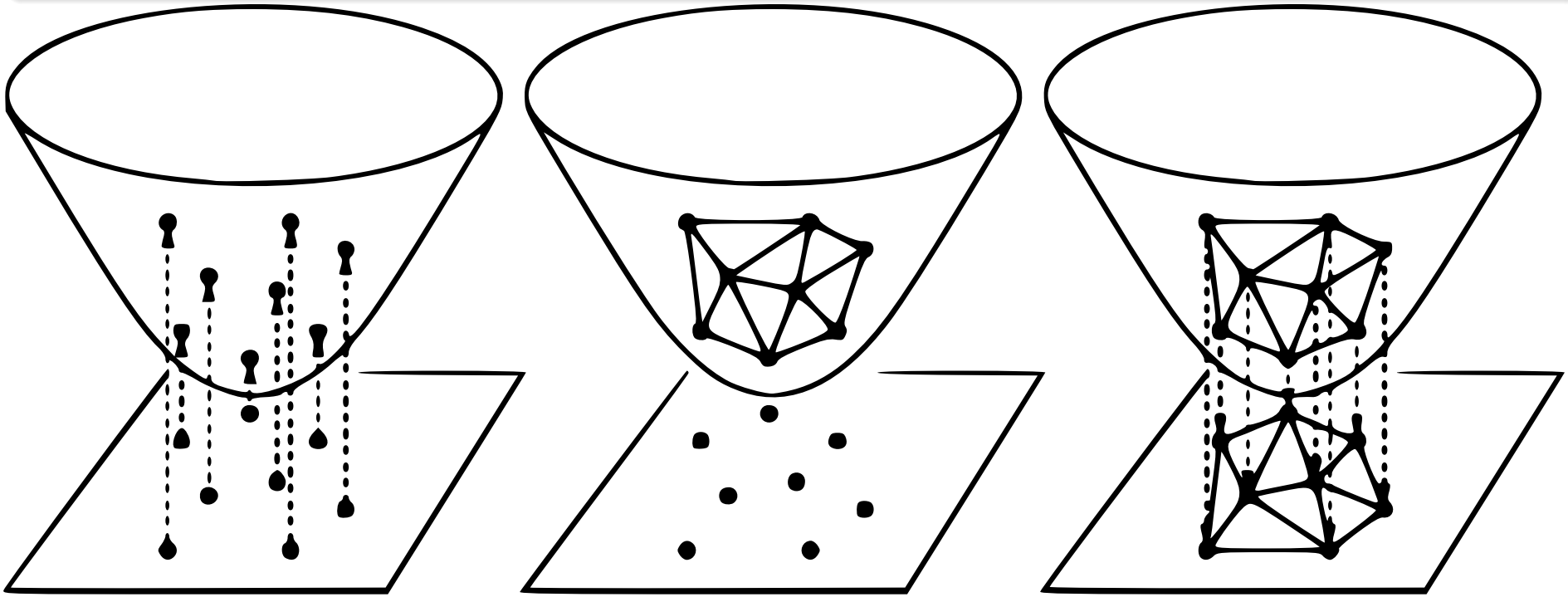
Optimal algorithm $O(n \log n)$

- 1D: equivalent to sorting points (x_i)
- Consider Vor_S, with $S = (x_i, x_i^2)$

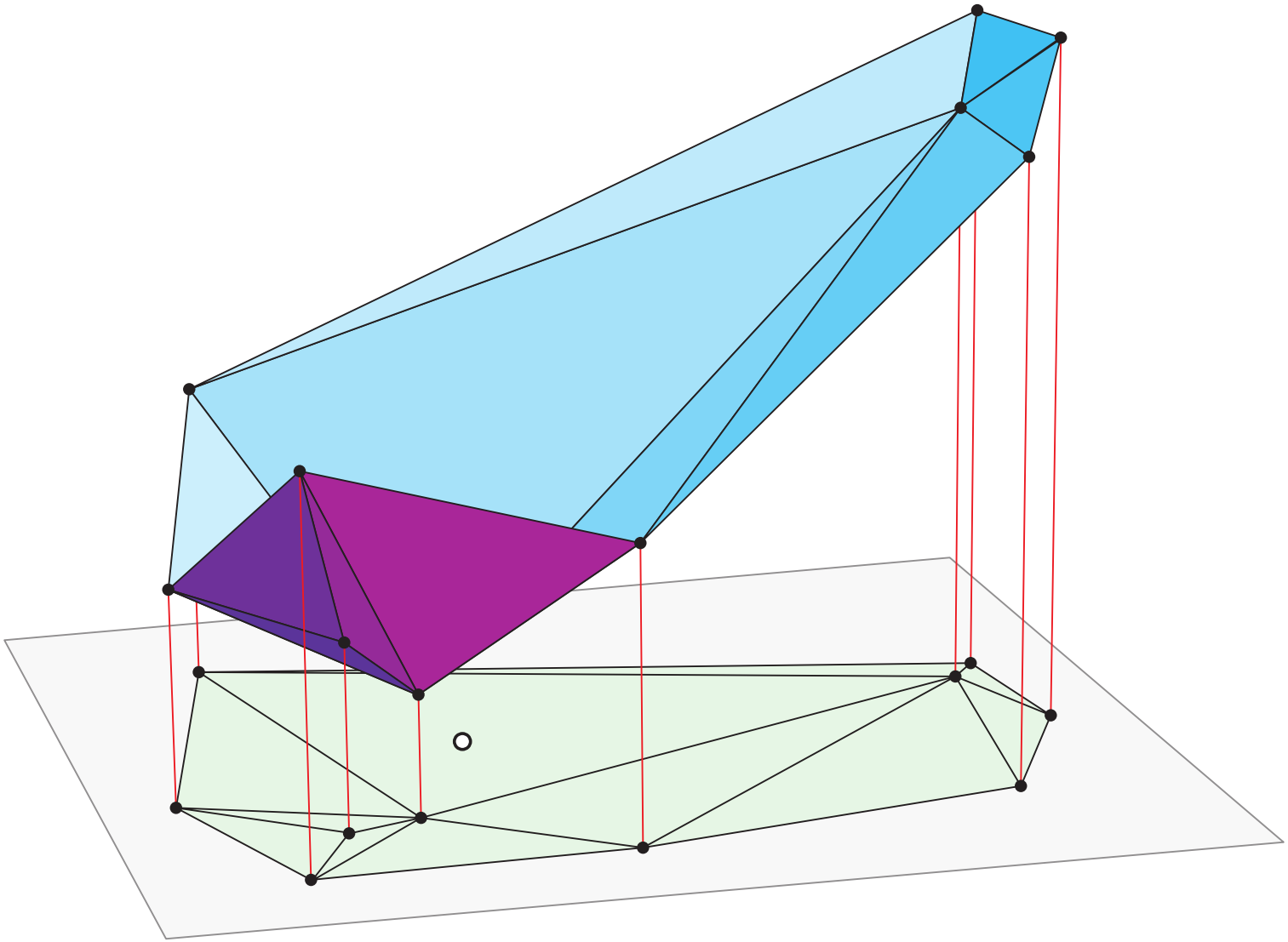


For 2D points: Convex Hull in 3D

- $S = \{(x_i, y_i)\}$
- Projection on a paraboloid: $S_{3D} = \{(x_i, y_i, x_i^2 + y_i^2)\}$
- Compute convex hull of S_{3D}
- Projection on the plane



For 2D points: Convex Hull in 3D



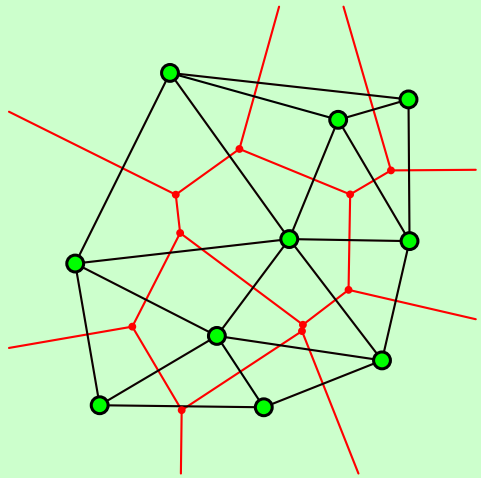
Delaunay triangulation

$$\mathrm{Vor}_S(p) = \bigcap_{q \in \mathrm{S} \setminus \{p\}} \mathrm{H}(p, q)$$

Definition (Delaunay triangulation)

$$\mathrm{DEL}(\mathrm{S}) = \{(p, q) \in \mathrm{S}^2 \ / \ \mathrm{Vor}_\mathrm{S}(p) \cap \mathrm{Vor}_\mathrm{S}(q) \neq \varnothing\}$$

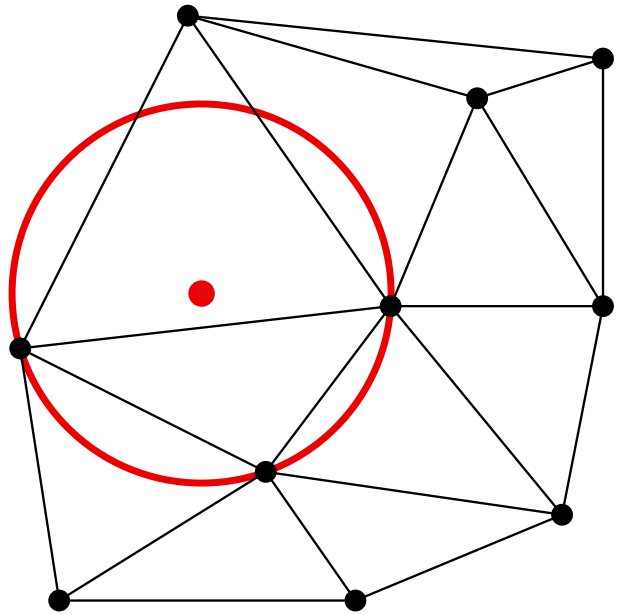
- Red: Voronoi graph
- Black: Delaunay triangulation
- Green: sites



Property: circumscribed circle

Property

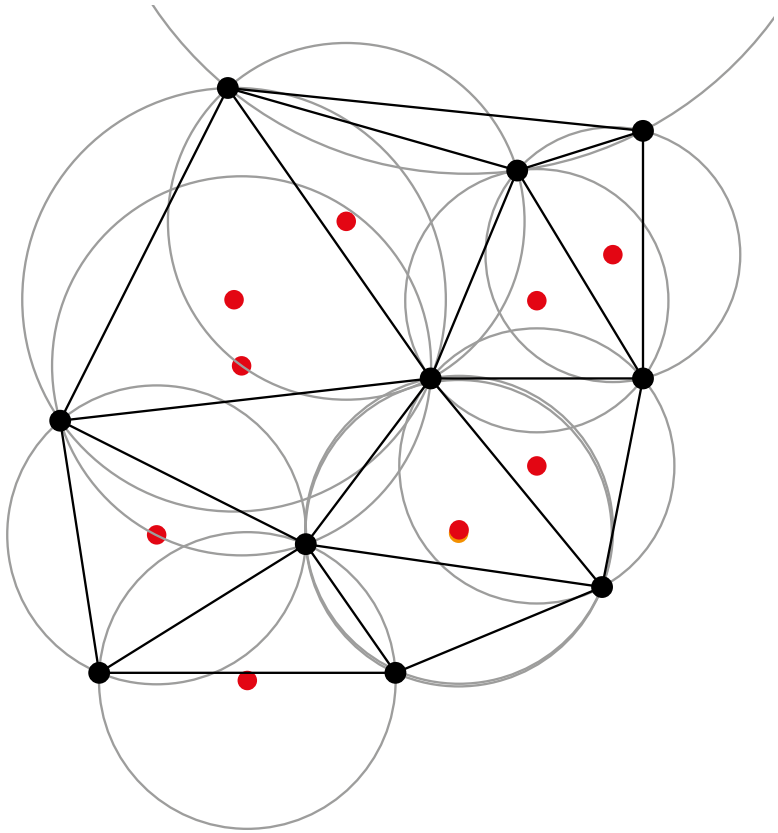
A circle circumscribing any Delaunay triangle does not contain any other input points in its interior



Property: circumscribed circle

Property

A circle circumscribing any Delaunay triangle does not contain any other input points in its interior



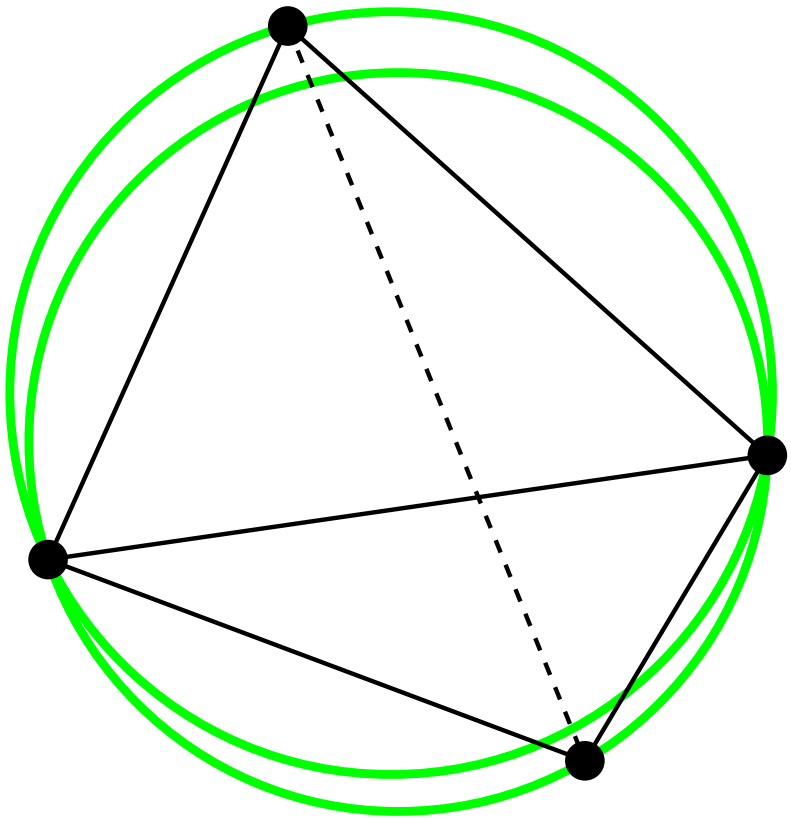
Flip algorithm

Construction algorithm

- Start by an arbitrary triangulation
- For all edges, test the 2 circles
- Flip edges if the property is not respected

Robust and fast method for in-circle testing

- use determinant of matrix
- $\Omega(n^2)$ operations



Power/Laguerre diagram

Definition (Power function)

- $S = \{(P_k, w_k)\}$
- Power of a point: $\mathrm{pow}(y, (P_k, w_i)) = d(y, P_k)^2 - w_k$
- Sphere of radius $r_k = \sqrt{w_k}$

Definition (Laguerre diagram)

$$\mathrm{Lags}(P_k) = \{x \in X \mid \mathrm{pow}(x, (P_k, w_k) \leq \mathrm{pow}(x, (P_j, w_k)) \; \forall j \neq k\}$$

- Laguerre cell can be empty
- It may not contain its seed
- It may contain other seeds

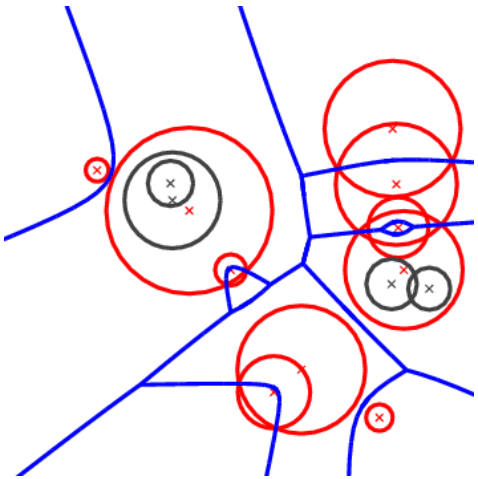
Apollonius diagram

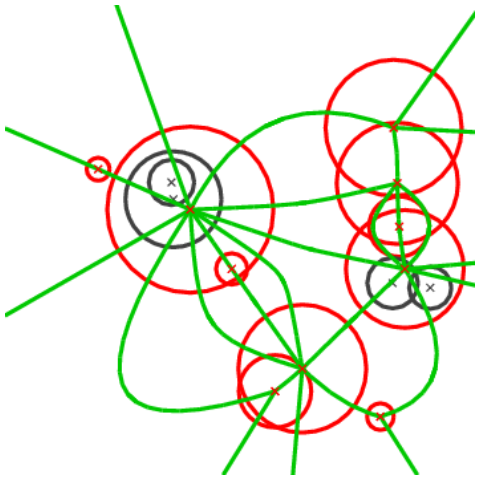
Definition (Apollonius diagram)

$$\mathrm{Apos}(P_k) = \{x \in X \mid \delta(x,(P_k,w_k) \leq \delta(x,(P_j,w_k))\ \forall j \neq k\}$$

Definition (Apollonius distance)

$$\delta(x,(P_k,w_k)) = d(x,P_k) - w_k$$



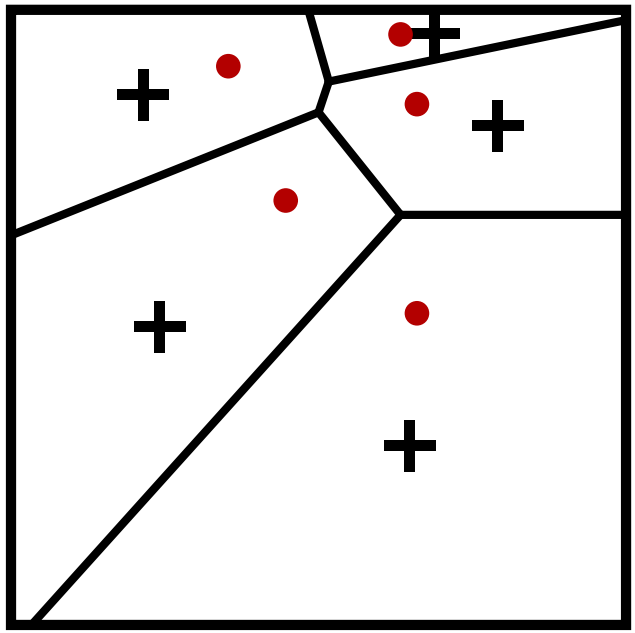


Centroidal Voronoi Tessellation

Lloyd's algorithm

Make several iterations of these steps:

- Compute the Voronoi diagram
- Compute the center of each cell
- Move each seed to its center



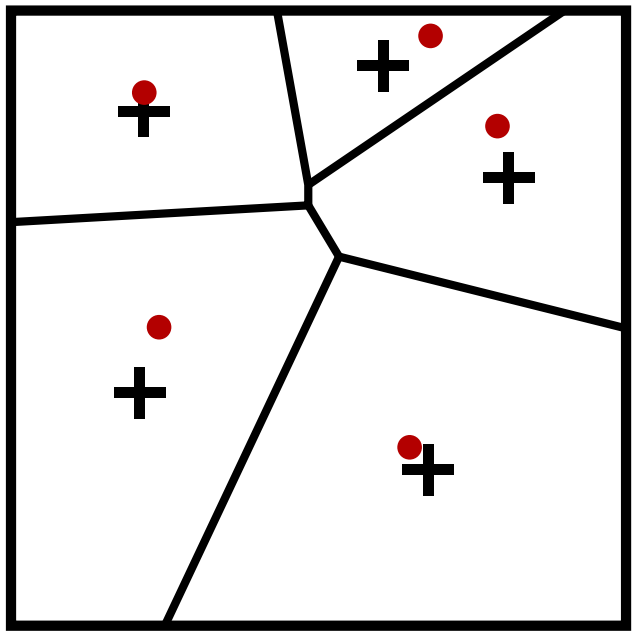
- It may take a long time to convergence
- How to handle infinite cells?
- Converges to hexagonal cells

Centroidal Voronoi Tessellation

Lloyd's algorithm

Make several iterations of these steps:

- Compute the Voronoi diagram
- Compute the center of each cell
- Move each seed to its center



- It may take a long time to convergence
- How to handle infinite cells?
- Converges to hexagonal cells

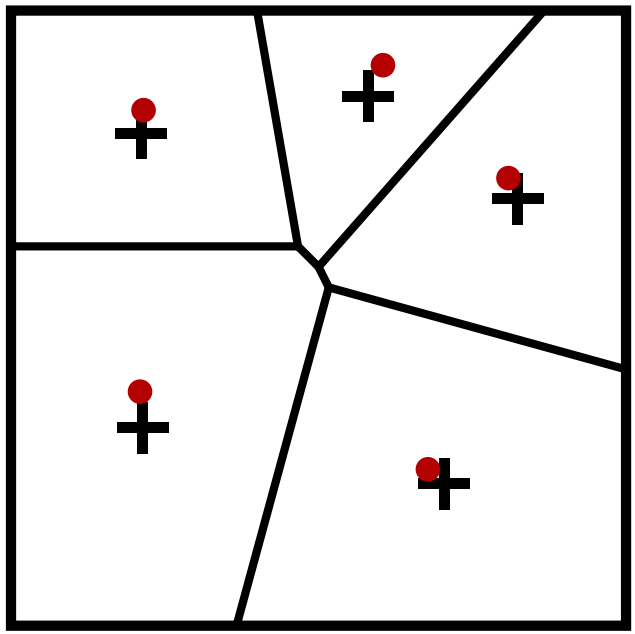
Centroidal Voronoi Tessellation

Lloyd's algorithm

Make several iterations of these steps:

- Compute the Voronoi diagram
- Compute the center of each cell
- Move each seed to its center

- It may take a long time to convergence
- How to handle infinite cells?
- Converges to hexagonal cells



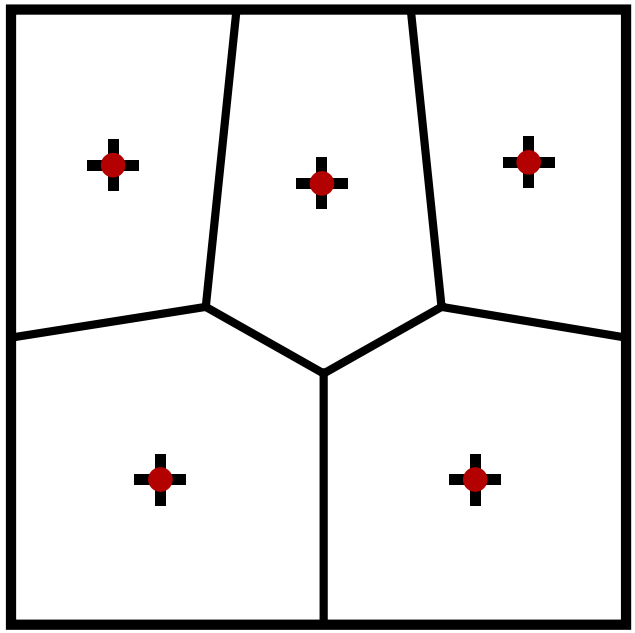
Centroidal Voronoi Tessellation

Lloyd's algorithm

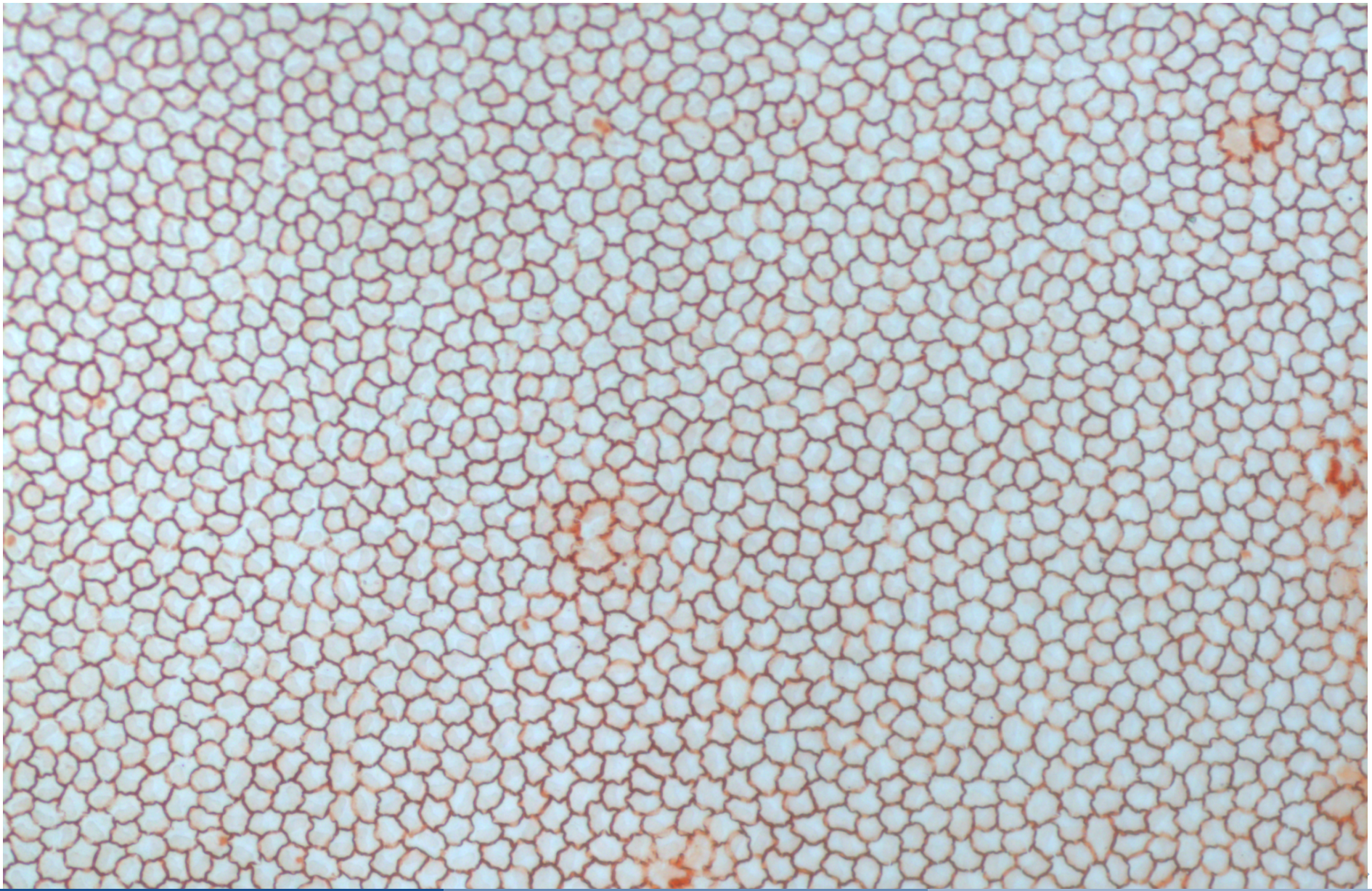
Make several iterations of these steps:

- Compute the Voronoi diagram
- Compute the center of each cell
- Move each seed to its center

- It may take a long time to convergence
- How to handle infinite cells?
- Converges to hexagonal cells



Application: simulation of corneal endothelium cells

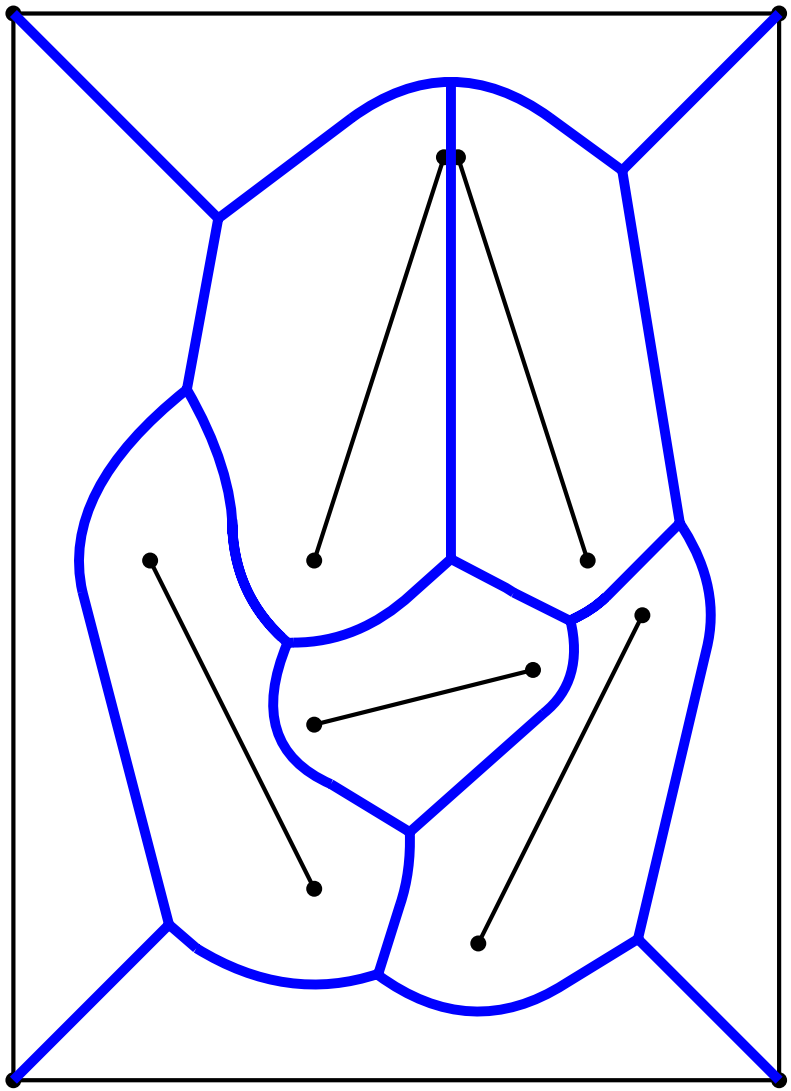


Voronoi diagram of objects

- Voronoi diagram of segments
- Voronoi diagram of objects
- Medial axis

Path planning

Medial axis is the location with the largest area around it.

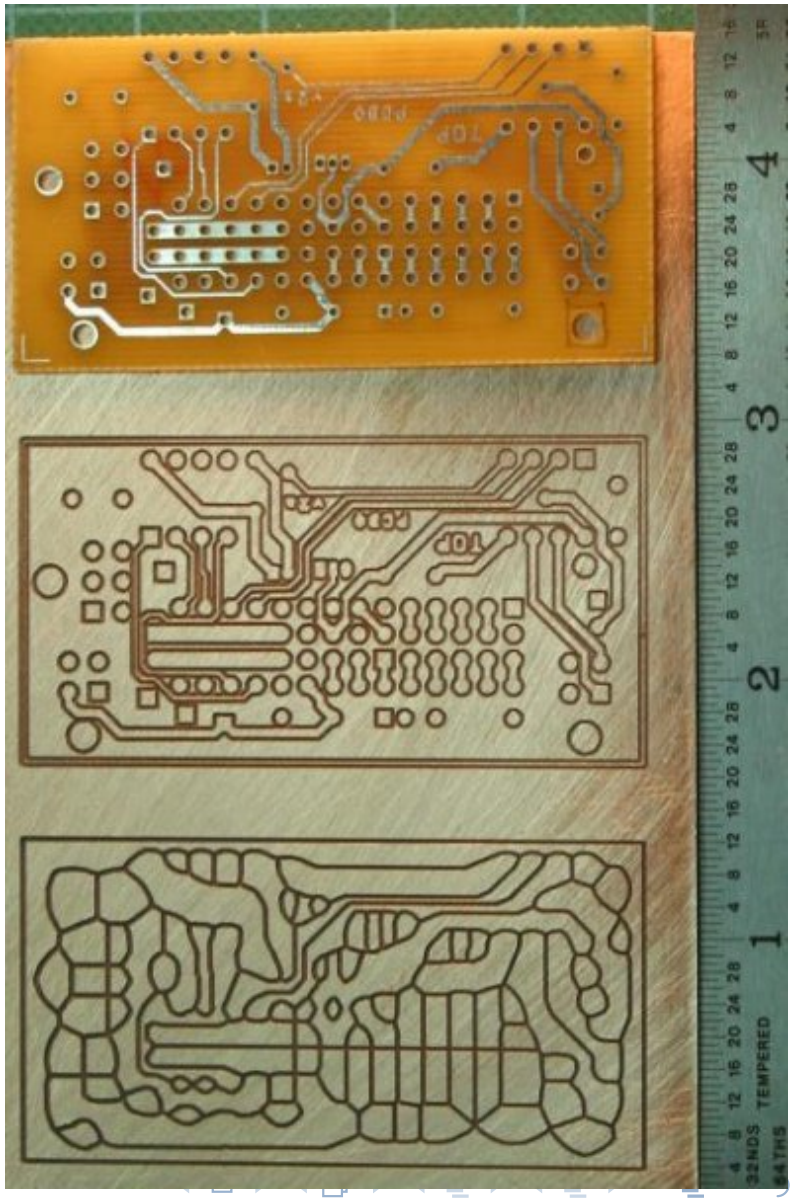


Example: PCB Printed Circuit Board conception

The same printed circuit board manufactured three different ways:

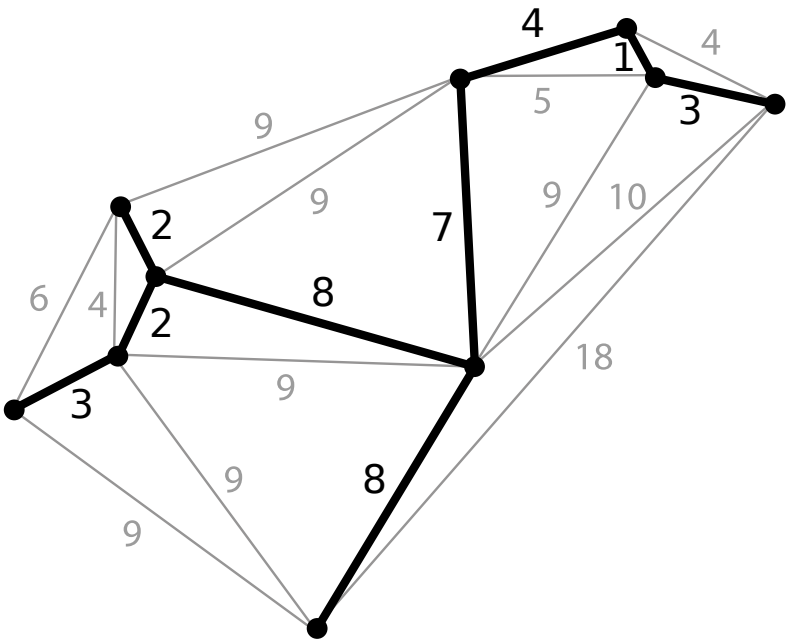
- by traditional photochemical process (top);
- by mechanical etch with standard outline toolpaths (middle);
- by mechanical etch with Voronoi toolpaths (bottom).

See [Vona and Rus, 2005].



Minimum spanning tree

- Planar graph
- No cycle
- Each vertex is connected
- Minimum weight
- Not necessarily unique



Greedy algorithms [Cormen et al., 2009]

n_v vertices, n_e edges.

- Borůvka, 1926: $O(n_e \log n_v)$
- Prim $O(n_e \log n_v)$ or $O(n_e + n_v \log n_v)$ (depends of data structure)
- Kruskal $O(n_e \log n_e)$
- Chazelle $O(n_e \alpha(n_e, n_v))$, α is the inverse of Ackerman function

$O(n_e \log n_v)$ is equivalent to $O(n_e \log n_e)$

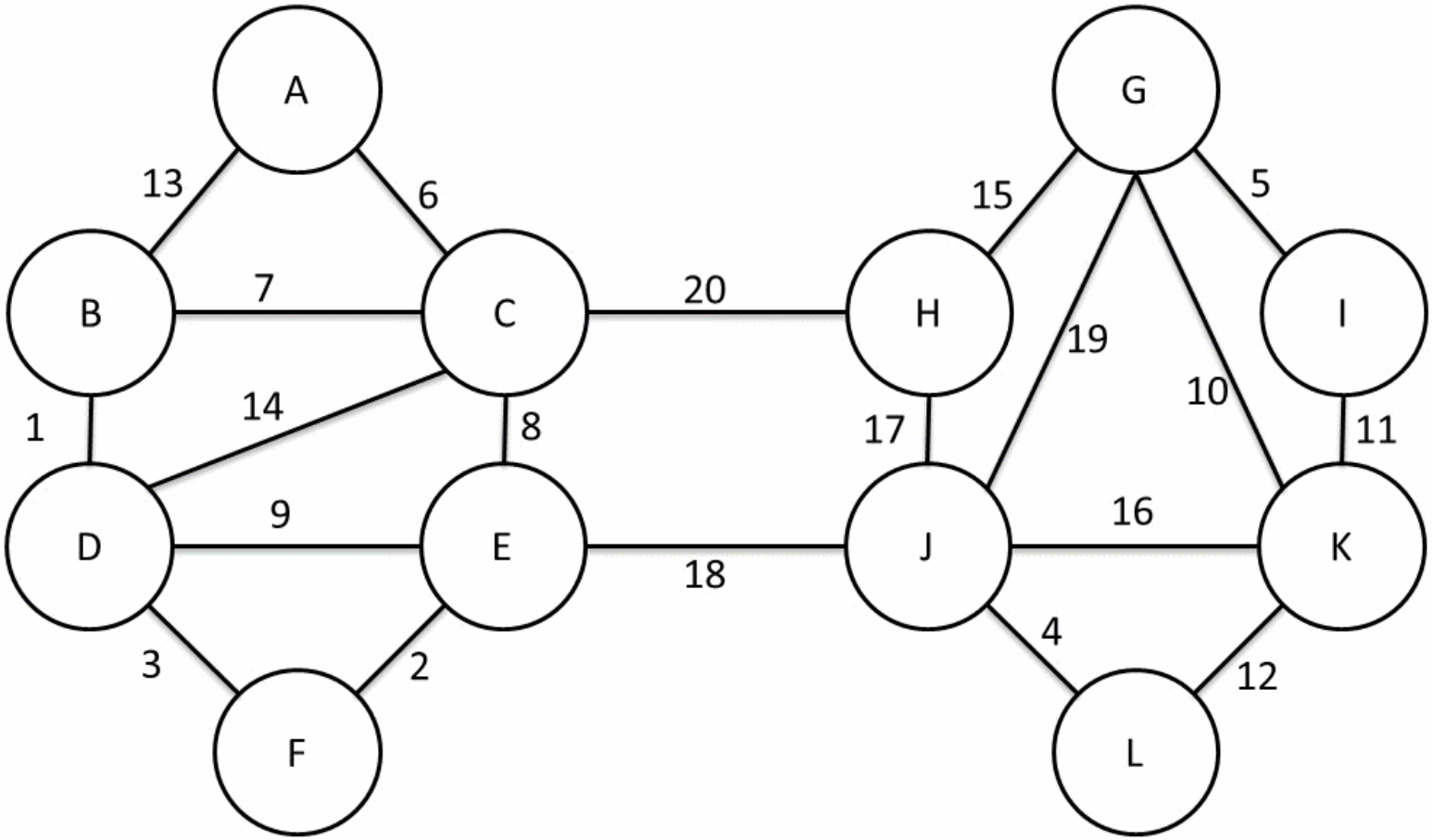
Boruvka algorithm, 1926

Warning

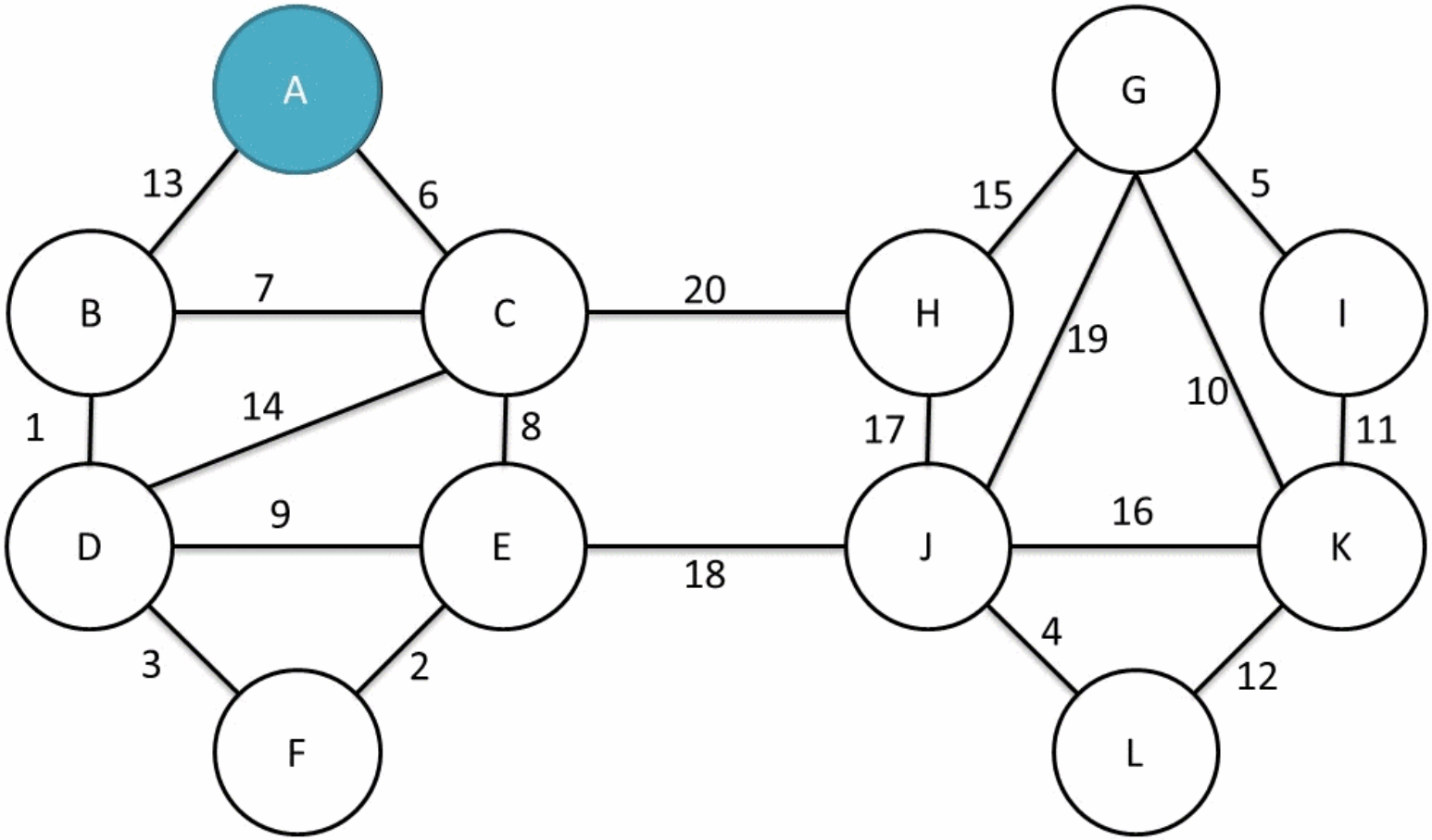
all edge weights are distinct,

Otakar Boruvka: an efficient electricity network for Moravia.

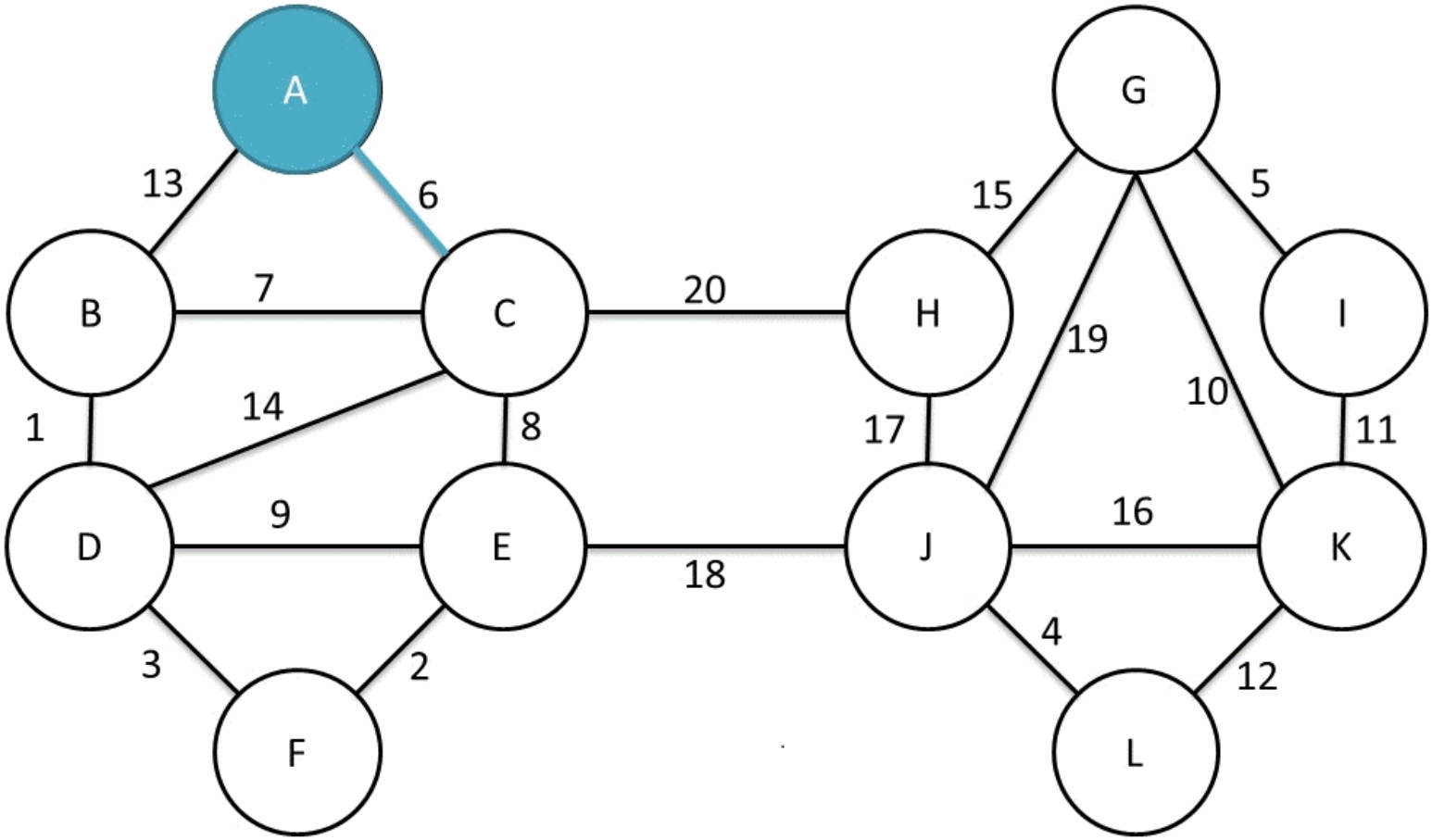
Boruvka algorithm, 1926



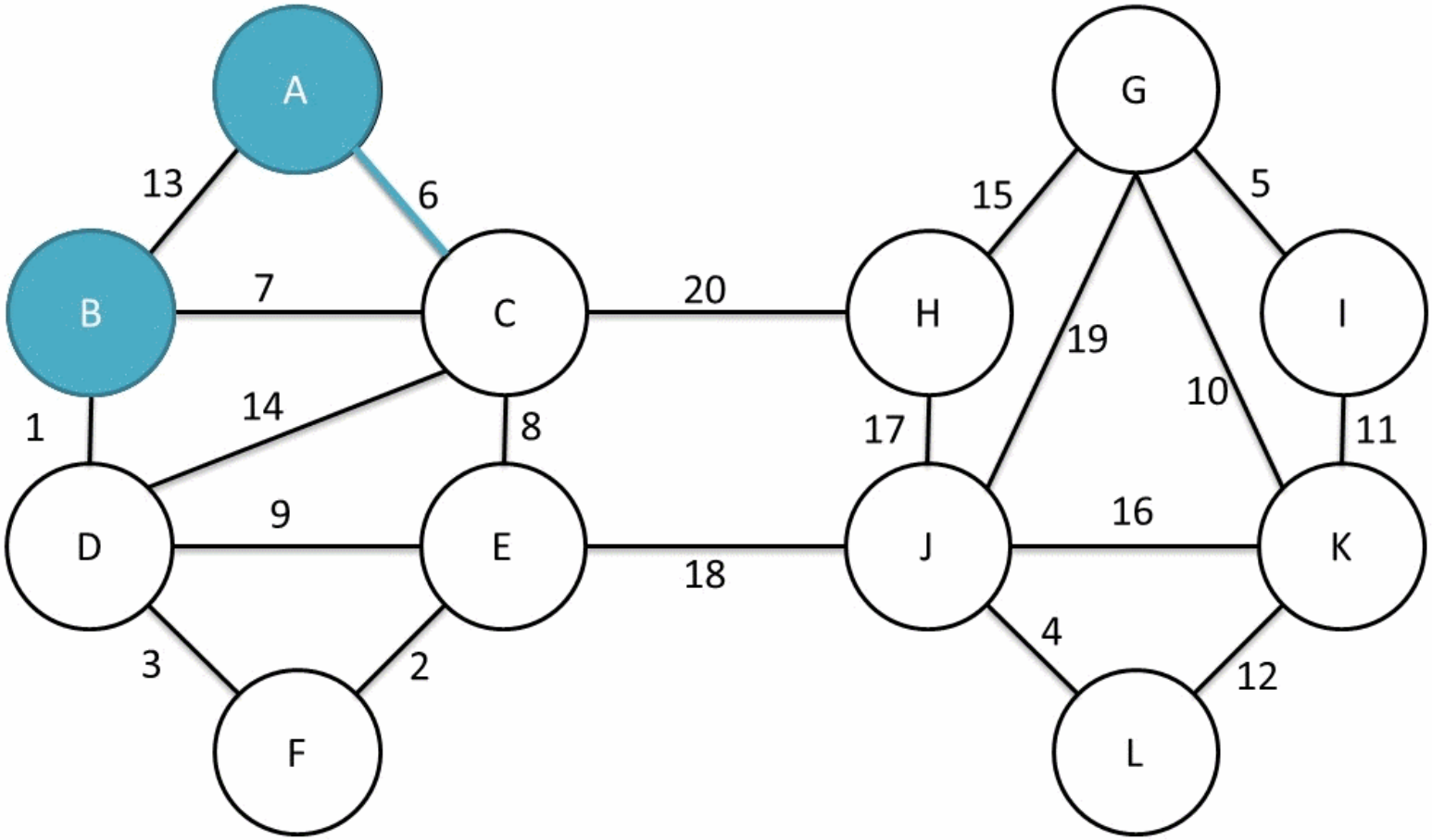
Boruvka algorithm, 1926



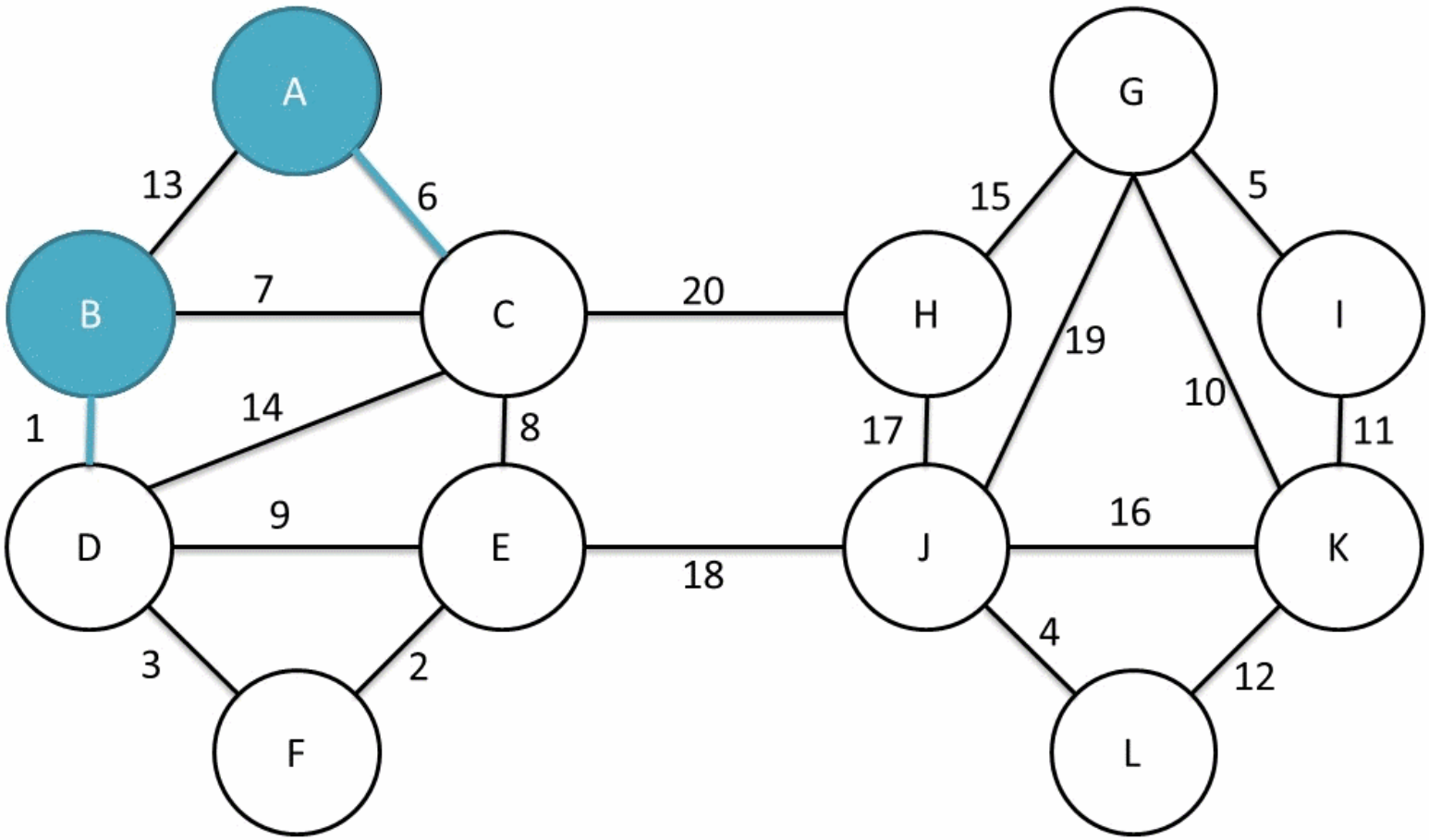
Boruvka algorithm, 1926



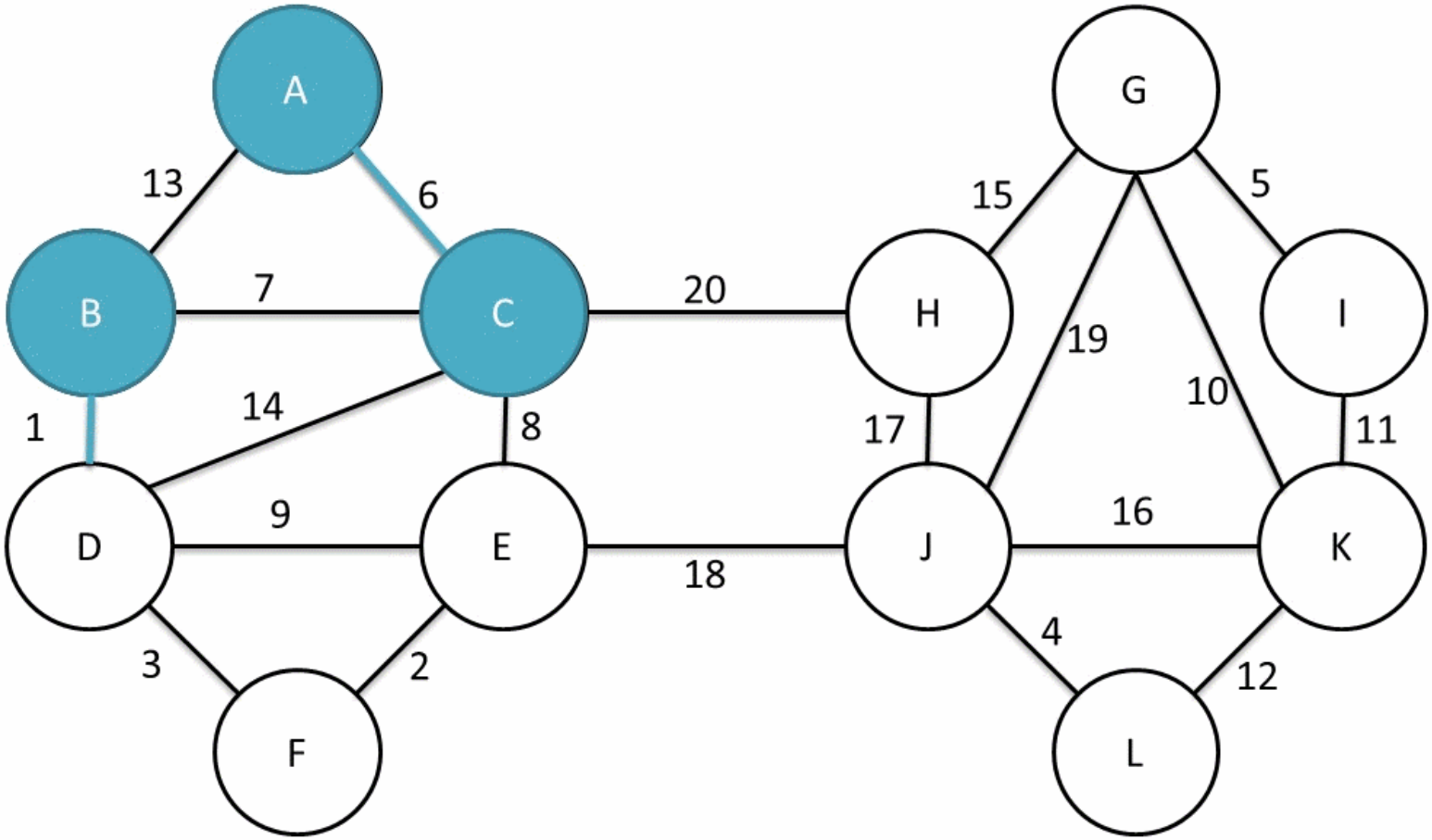
Boruvka algorithm, 1926



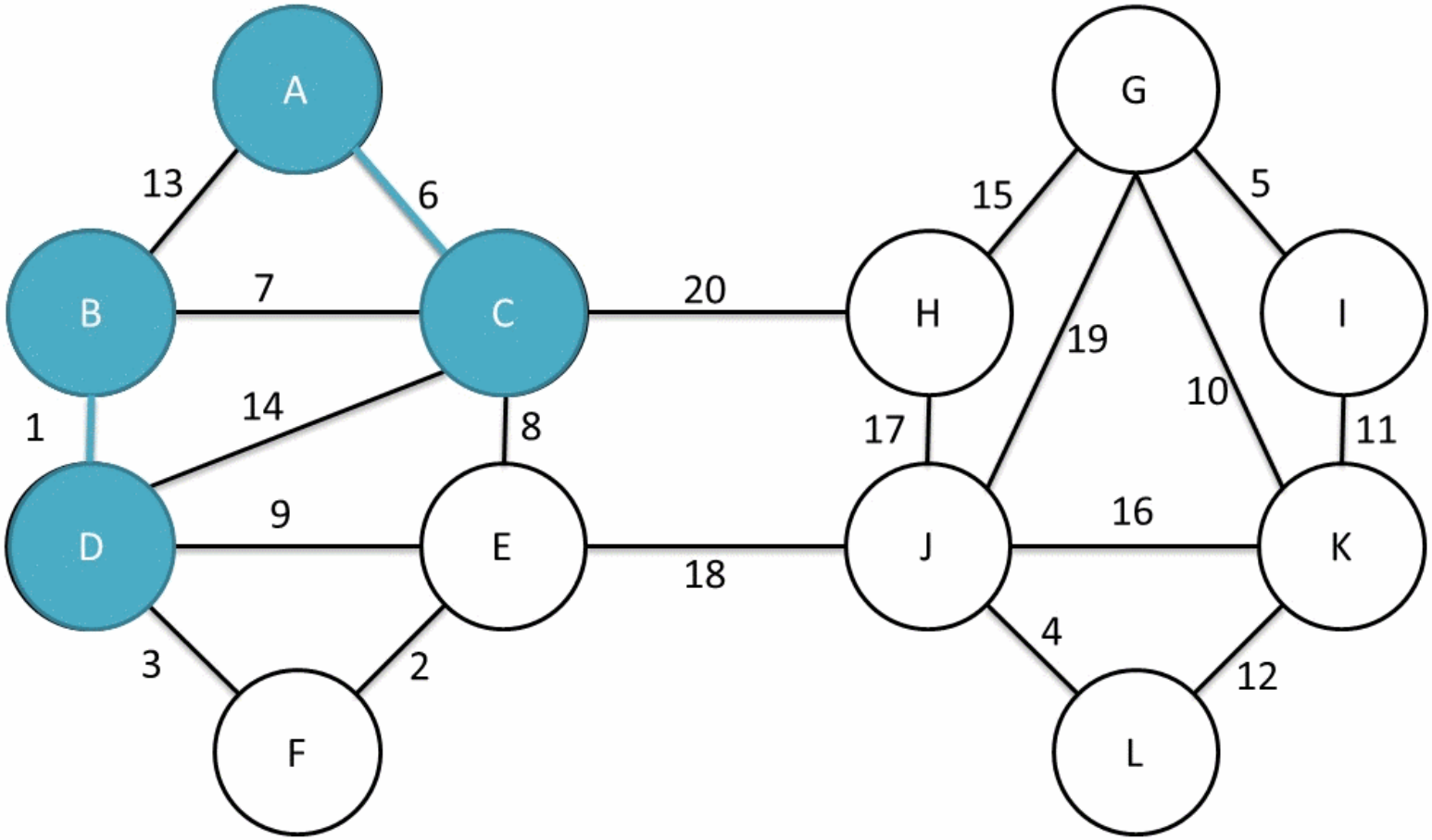
Boruvka algorithm, 1926



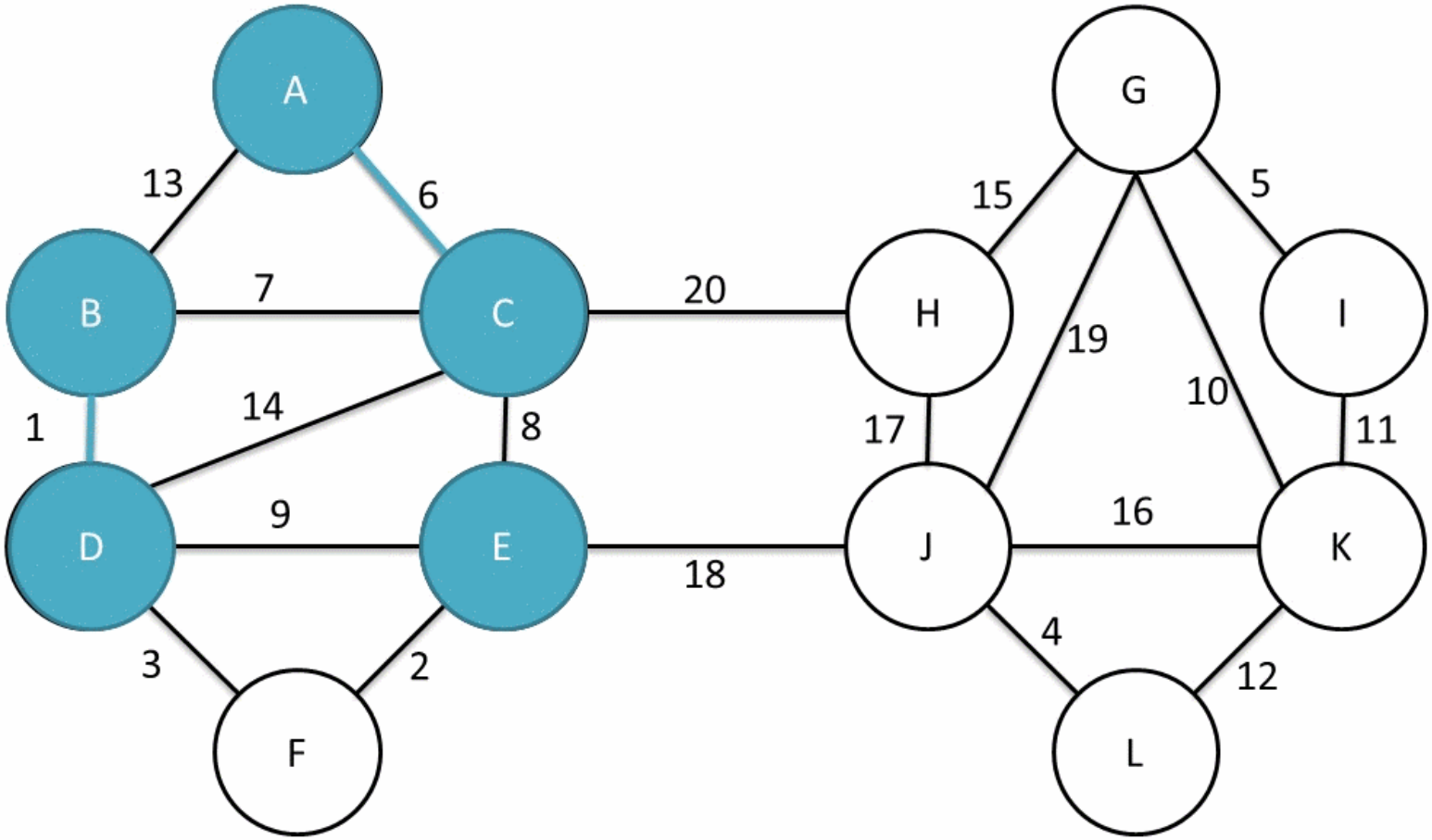
Boruvka algorithm, 1926



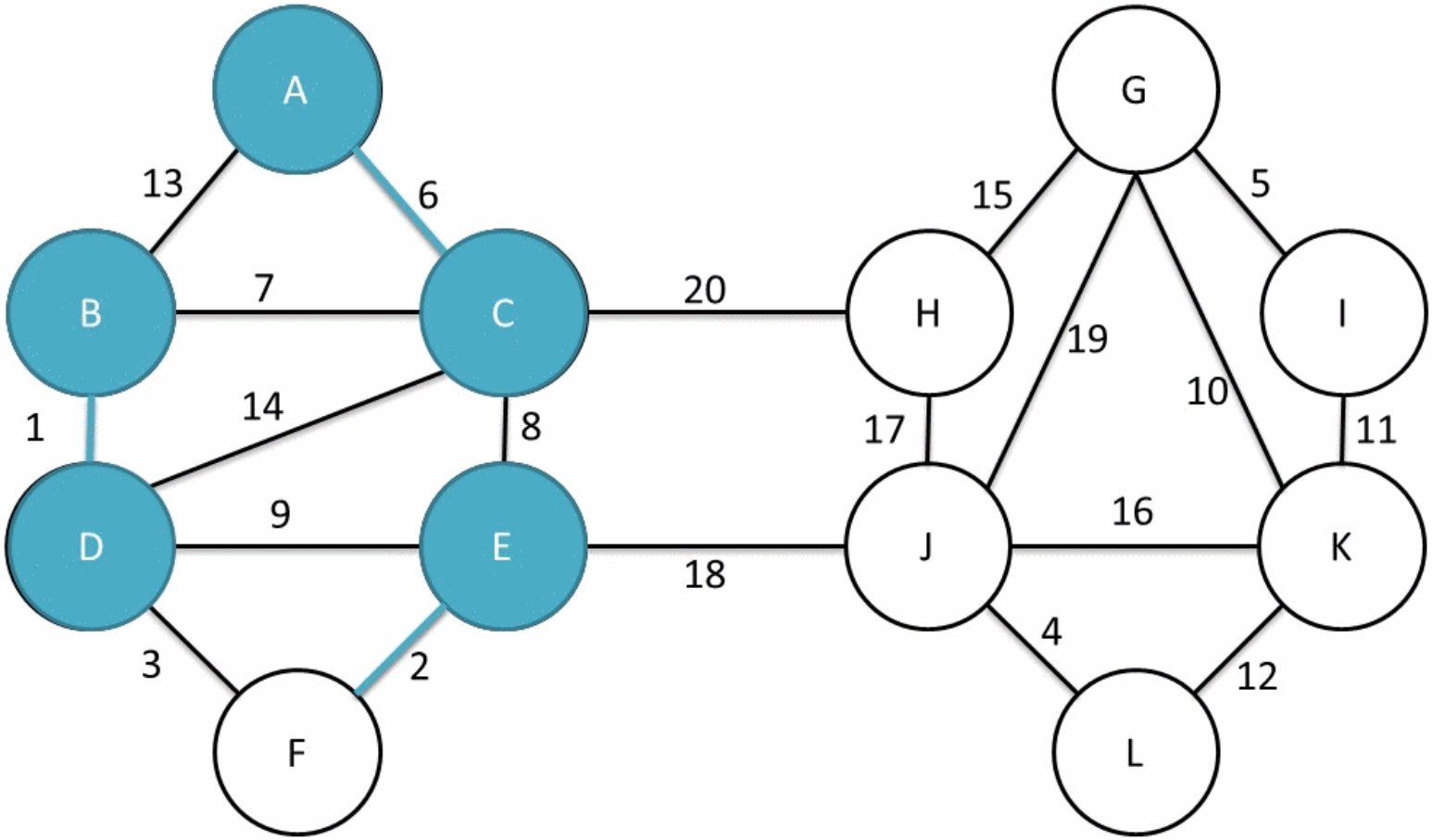
Boruvka algorithm, 1926



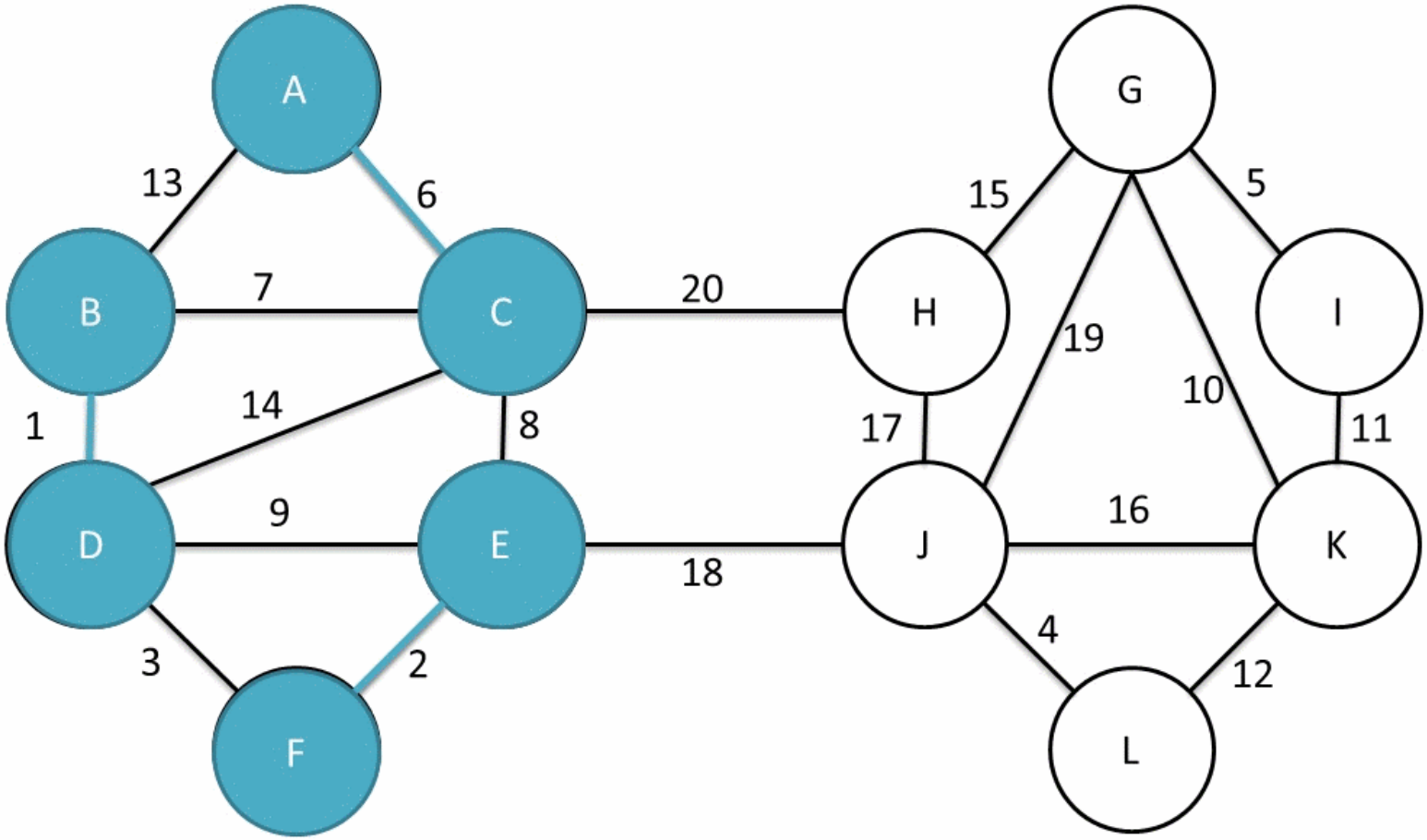
Boruvka algorithm, 1926



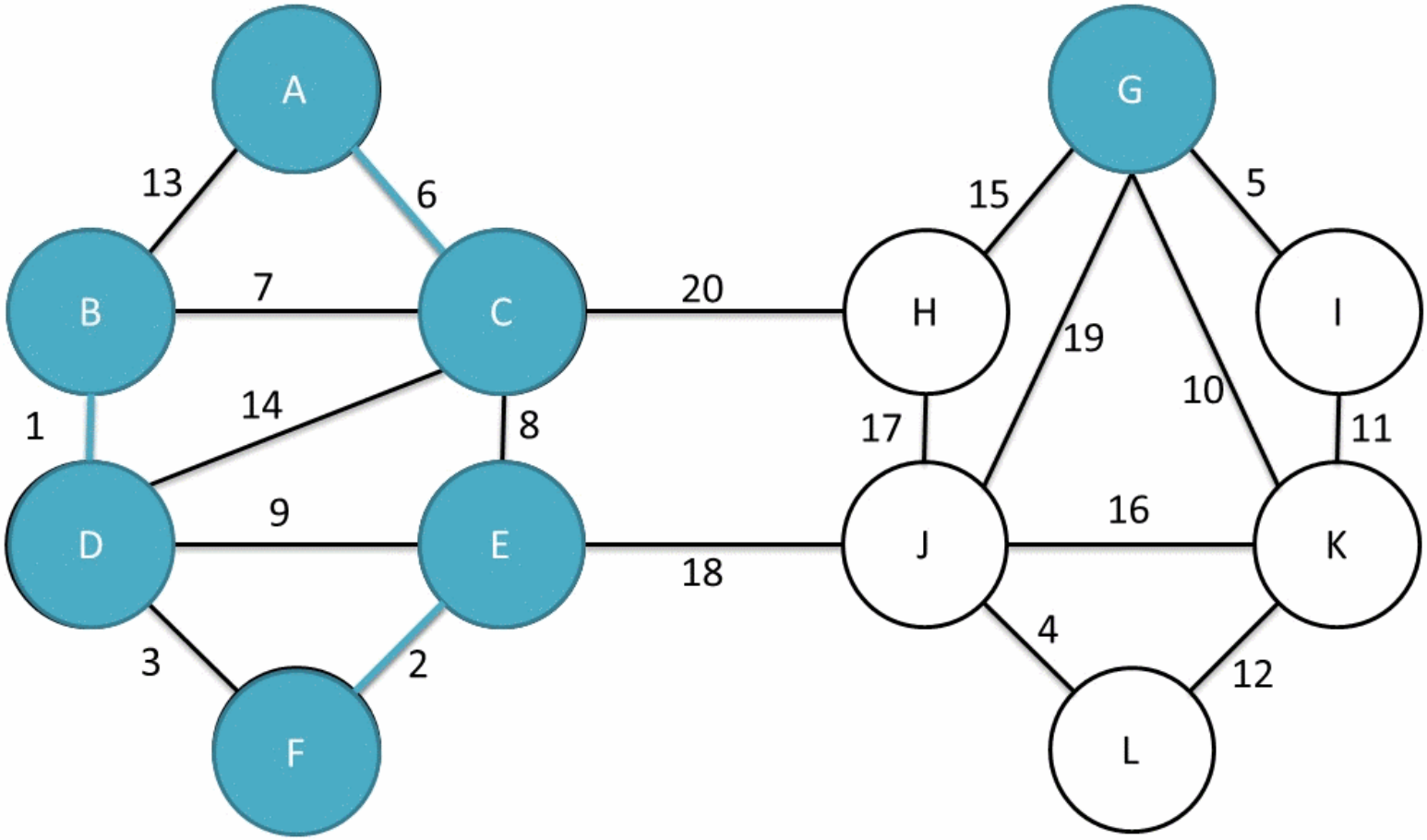
Boruvka algorithm, 1926



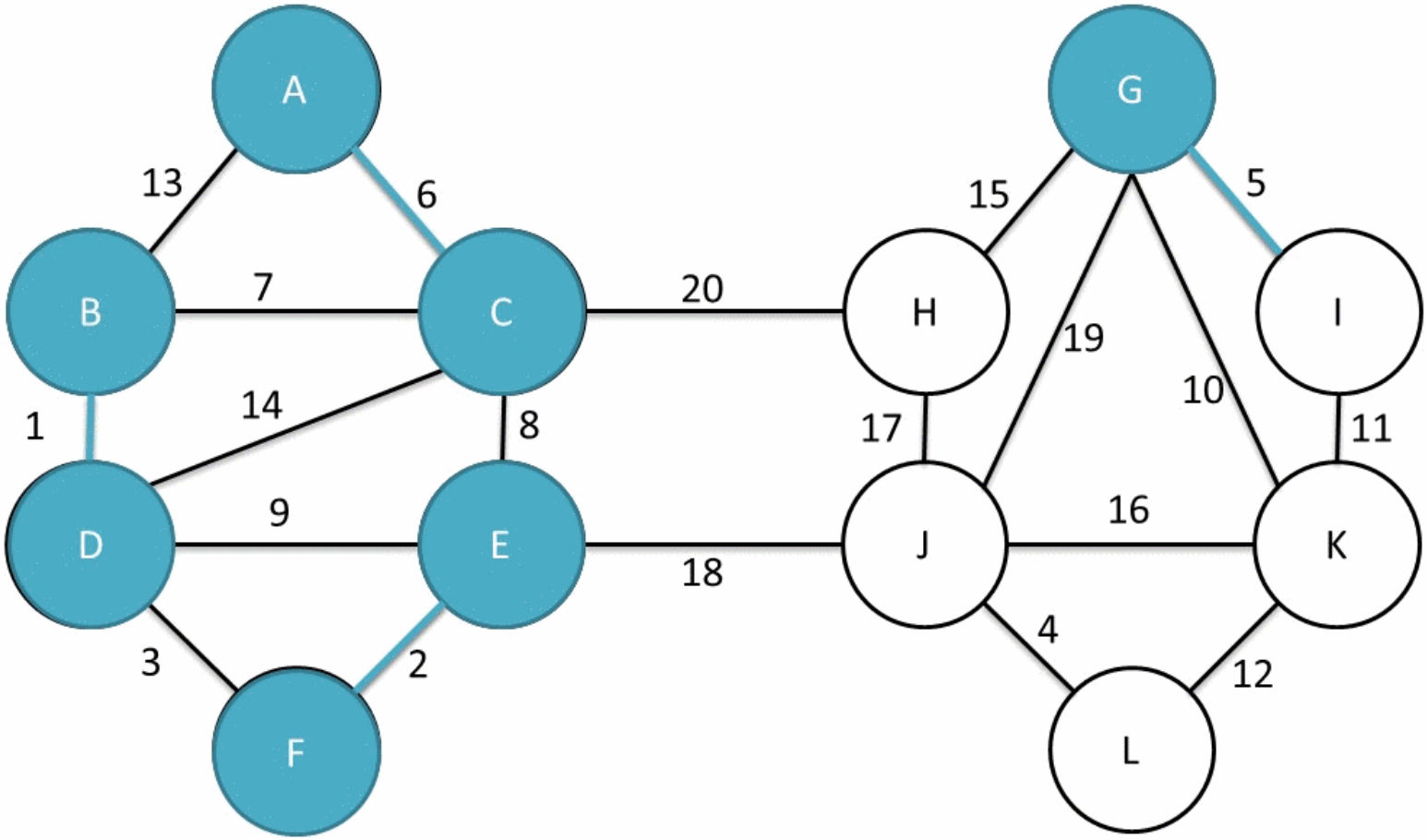
Boruvka algorithm, 1926



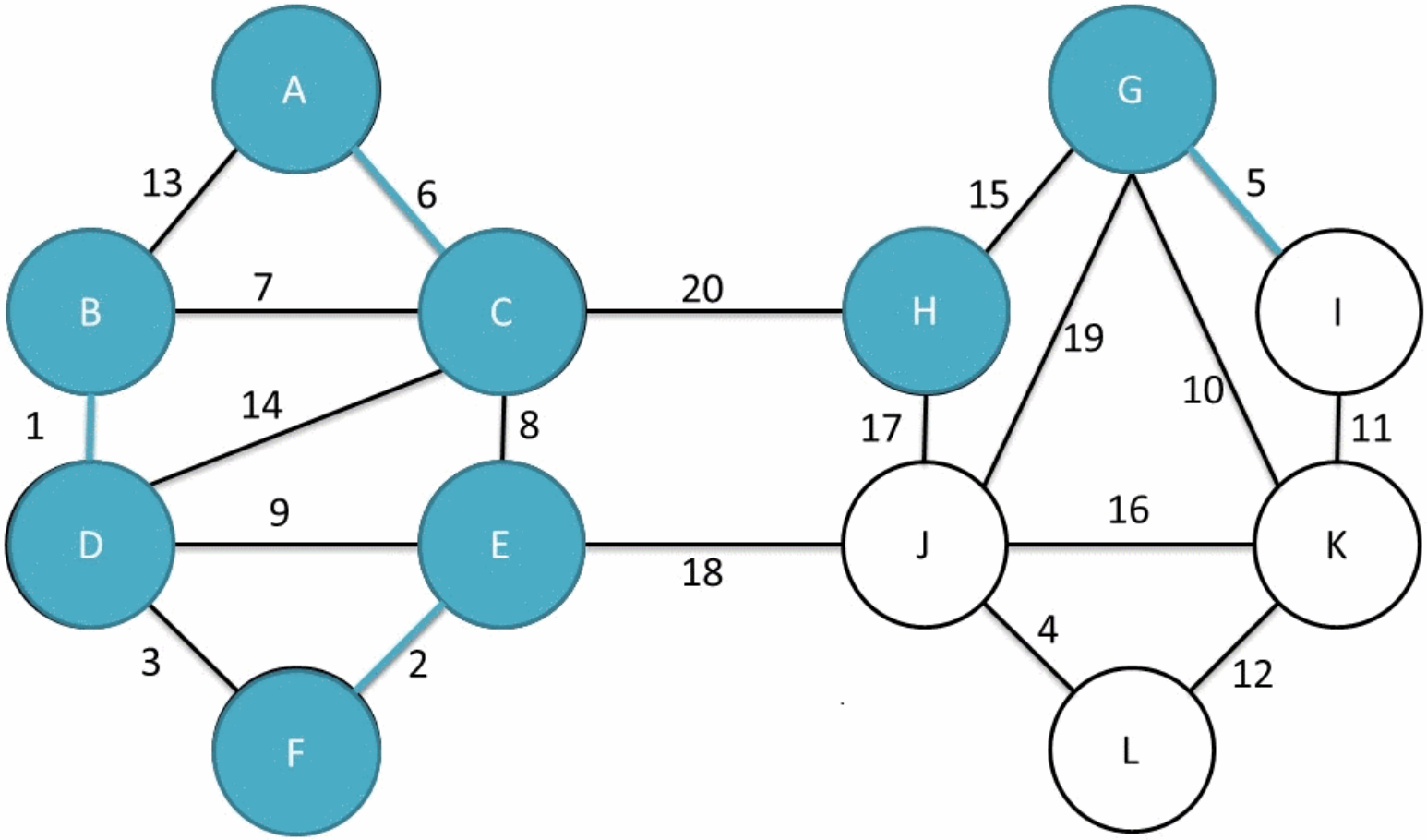
Boruvka algorithm, 1926



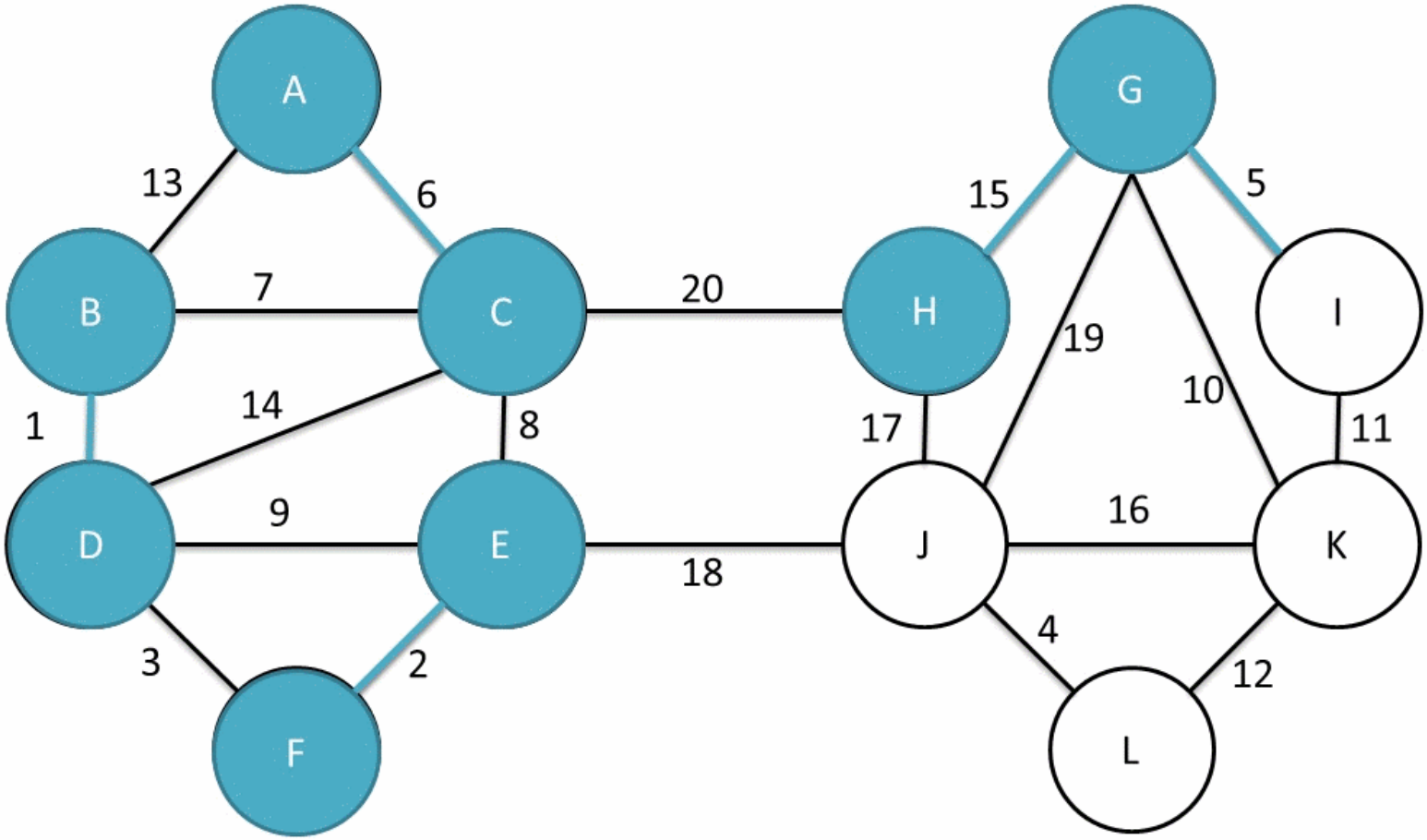
Boruvka algorithm, 1926



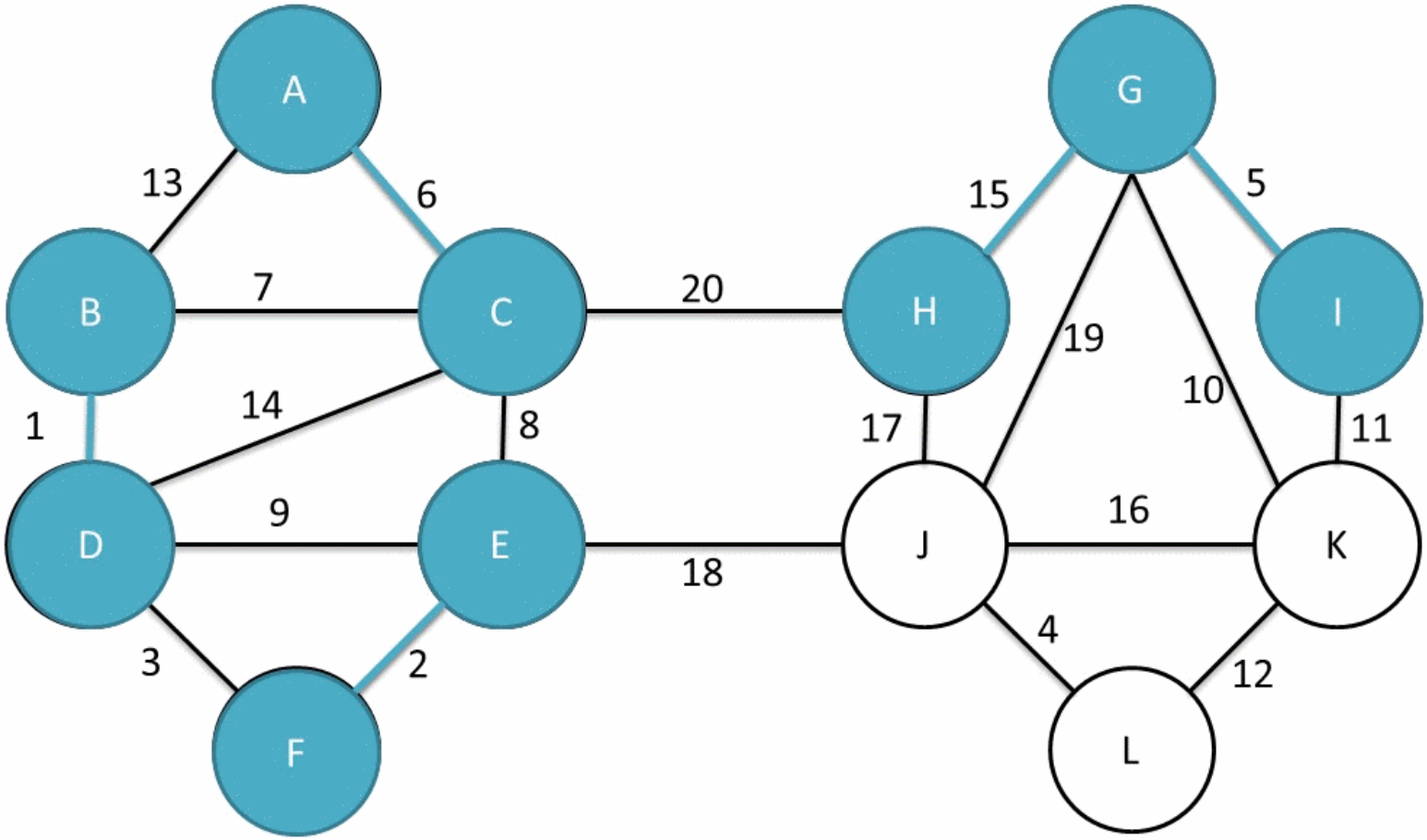
Boruvka algorithm, 1926



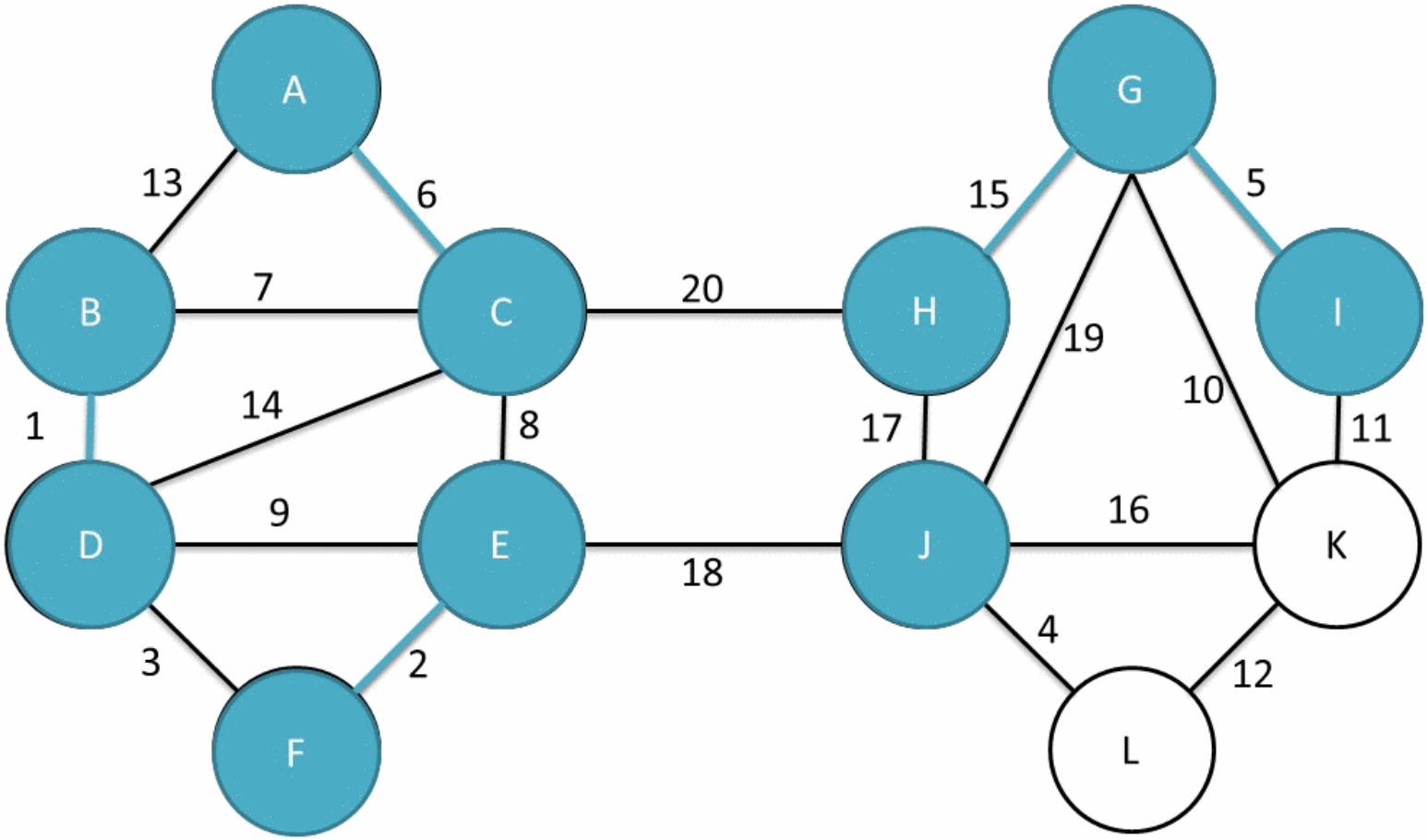
Boruvka algorithm, 1926



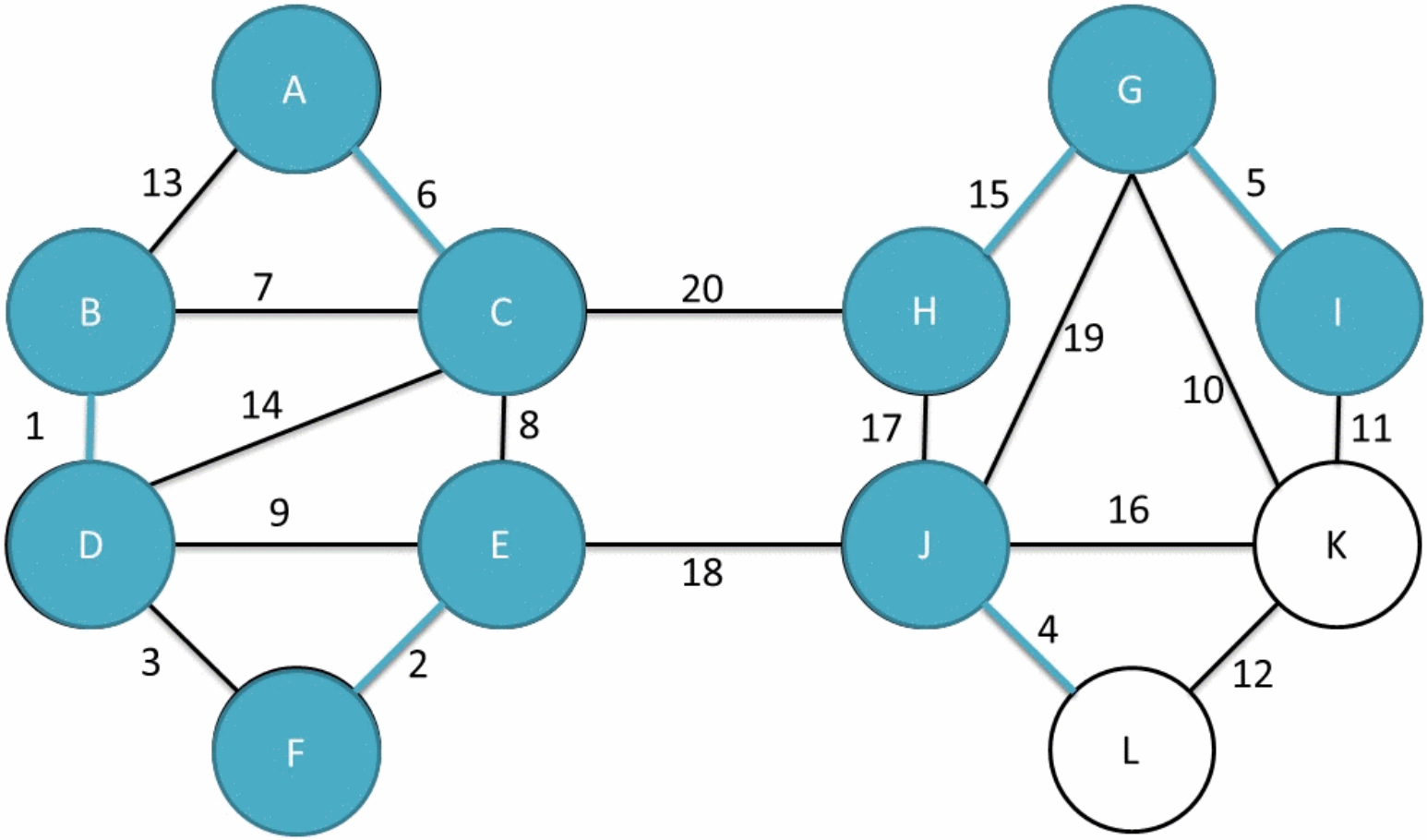
Boruvka algorithm, 1926



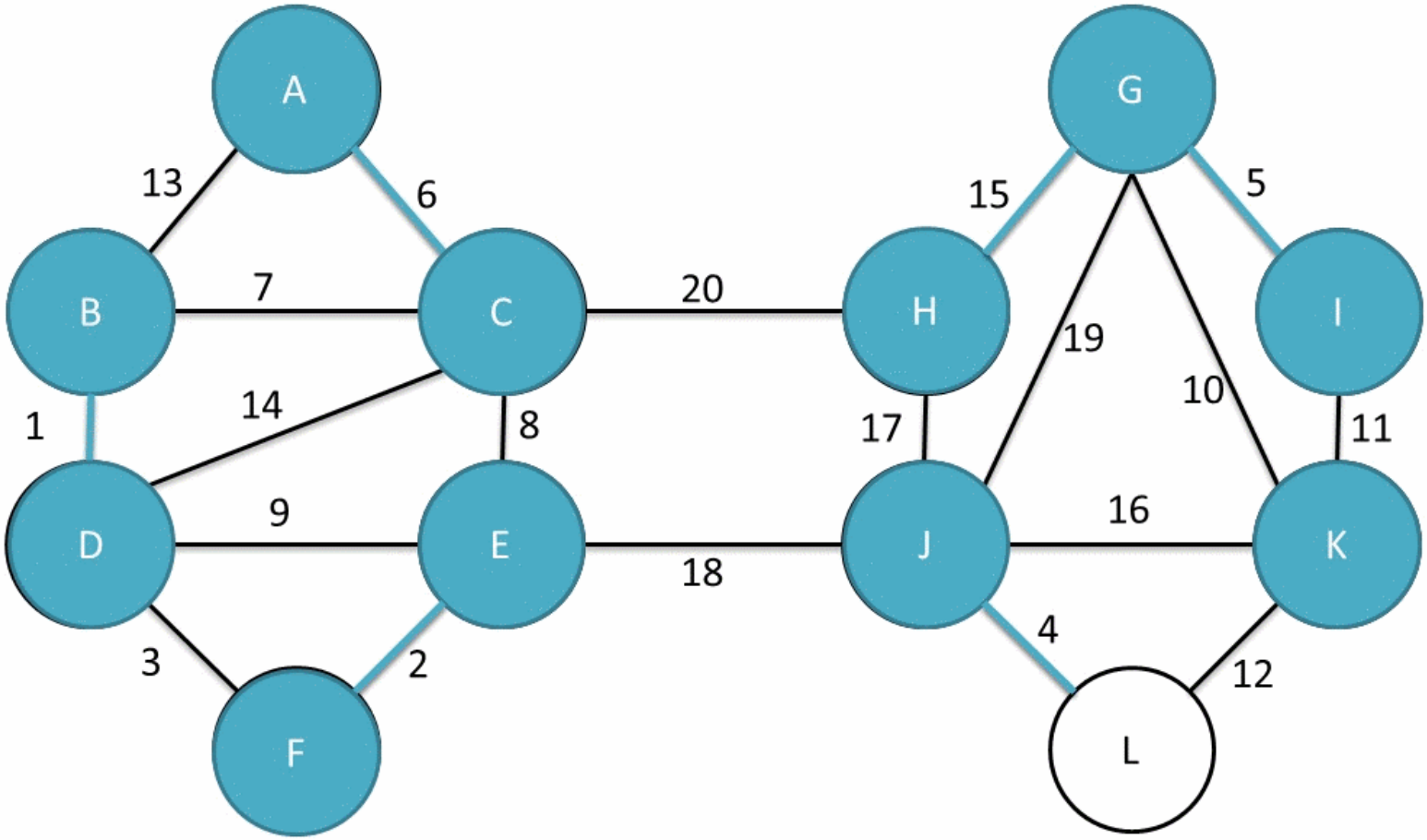
Boruvka algorithm, 1926



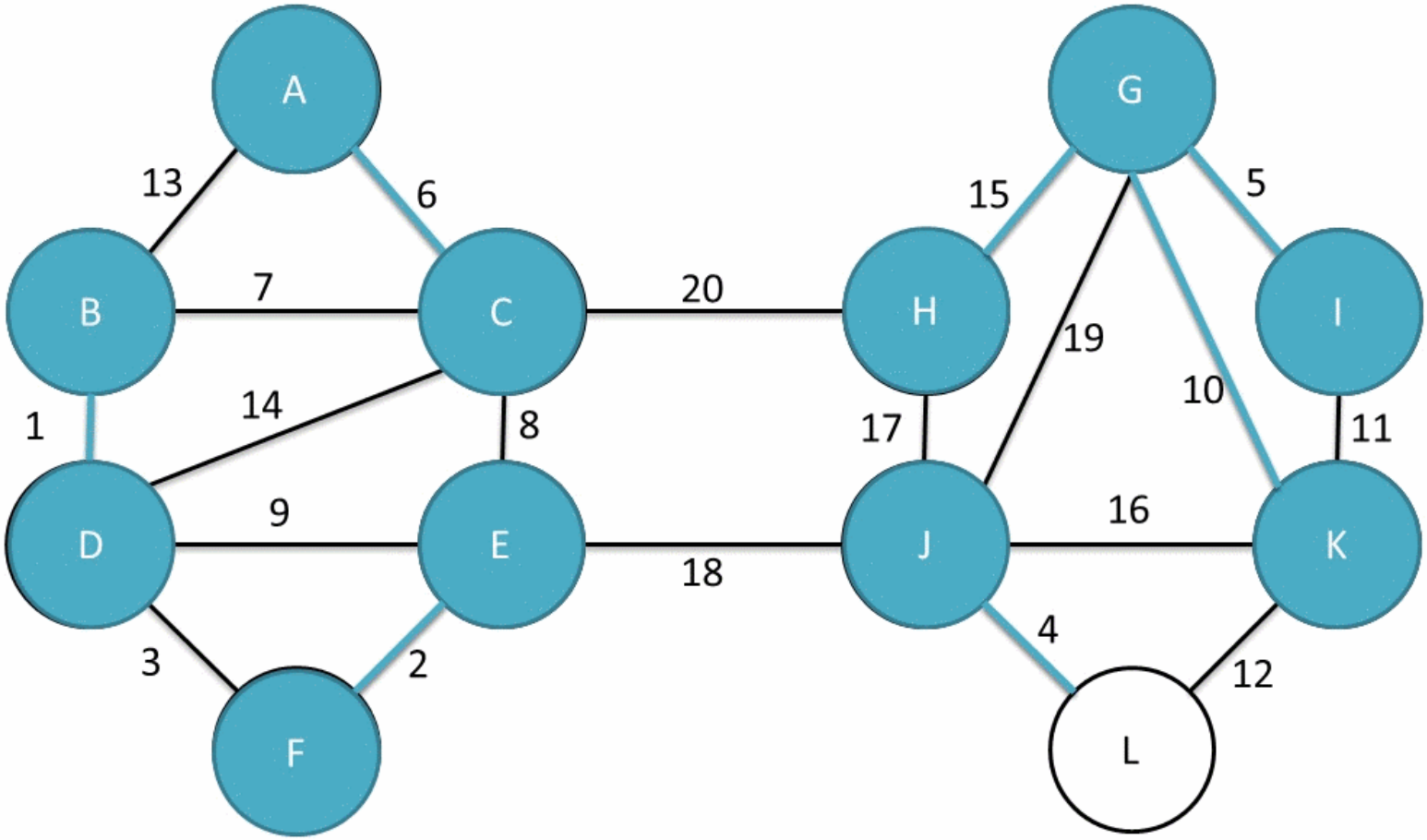
Boruvka algorithm, 1926



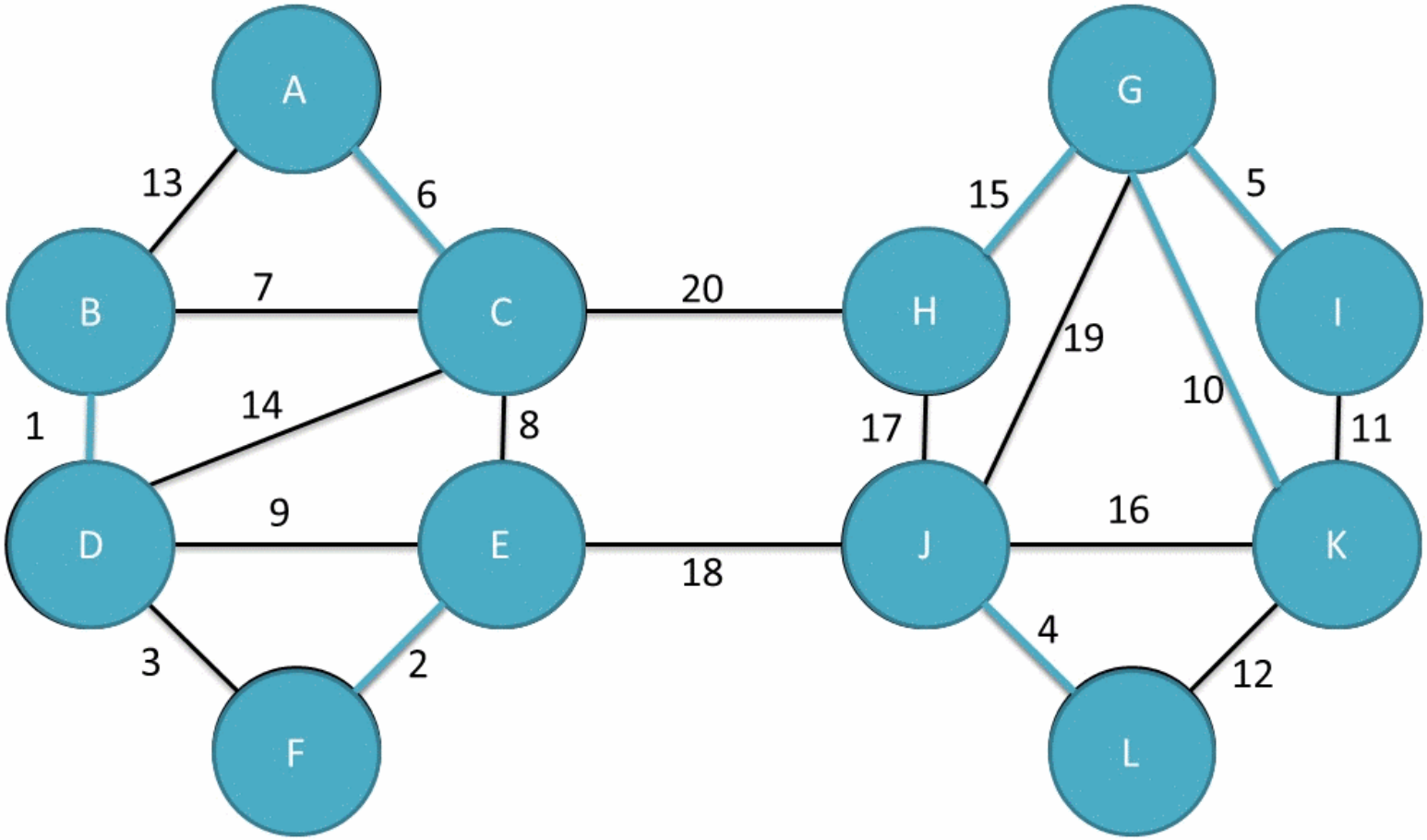
Boruvka algorithm, 1926



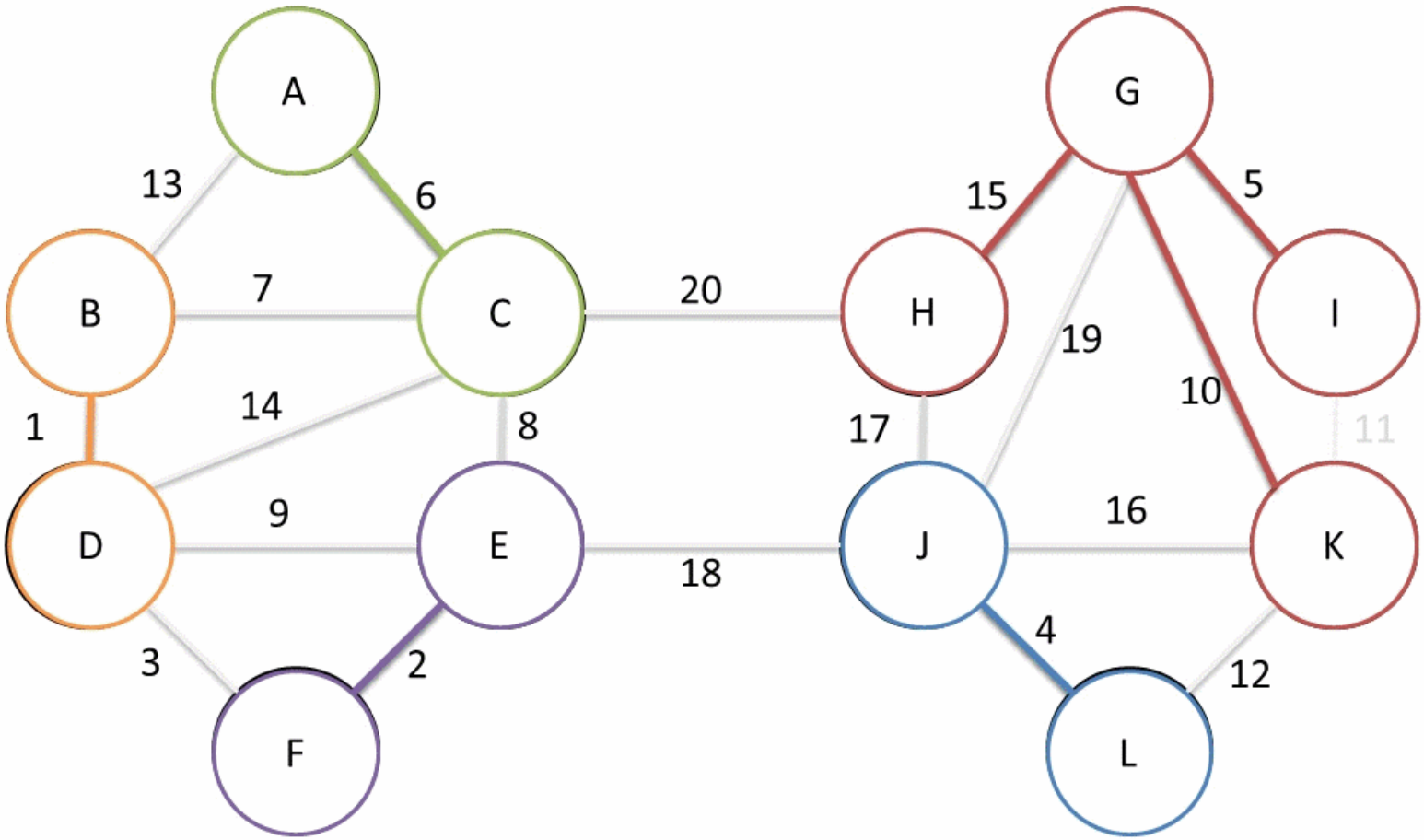
Boruvka algorithm, 1926



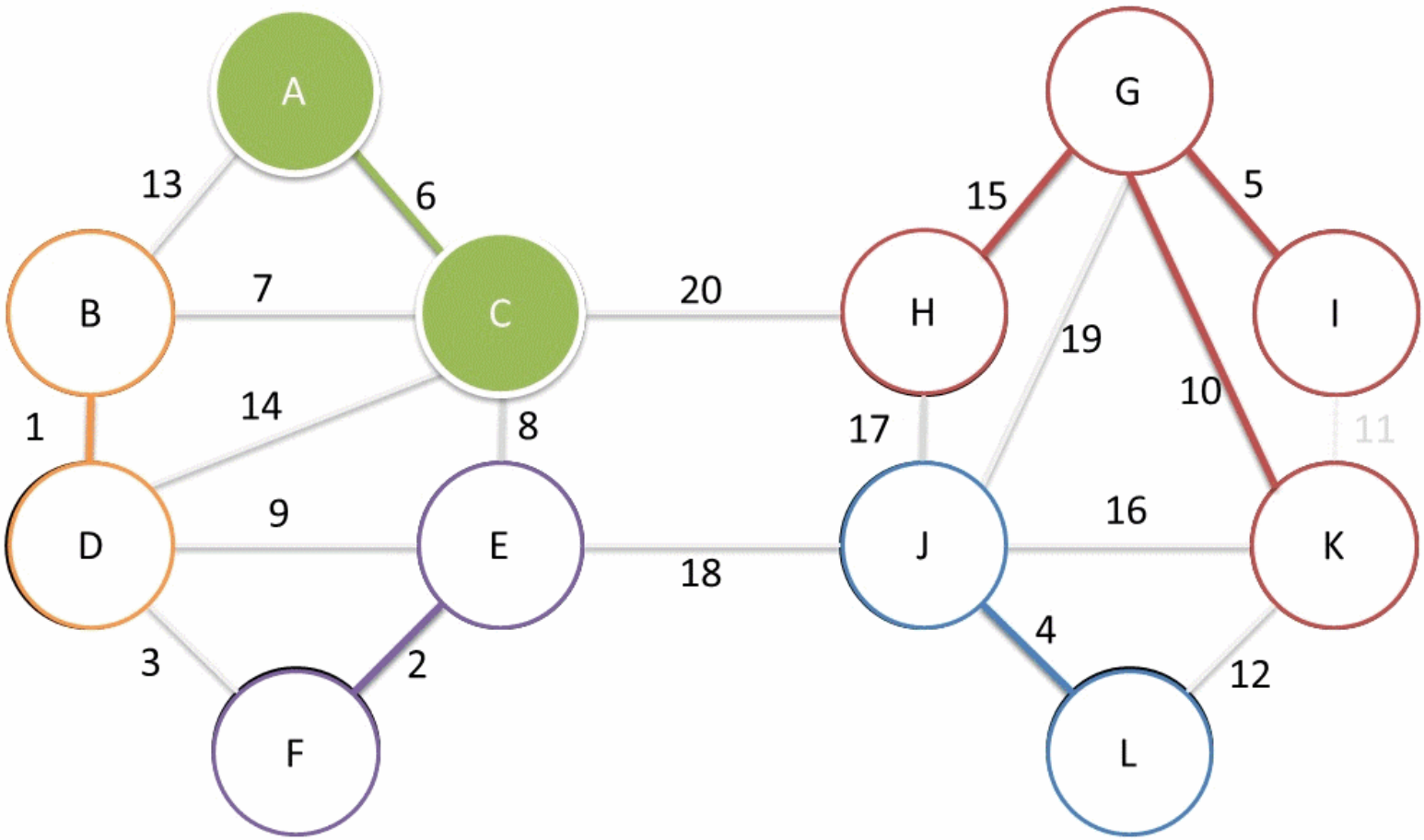
Boruvka algorithm, 1926



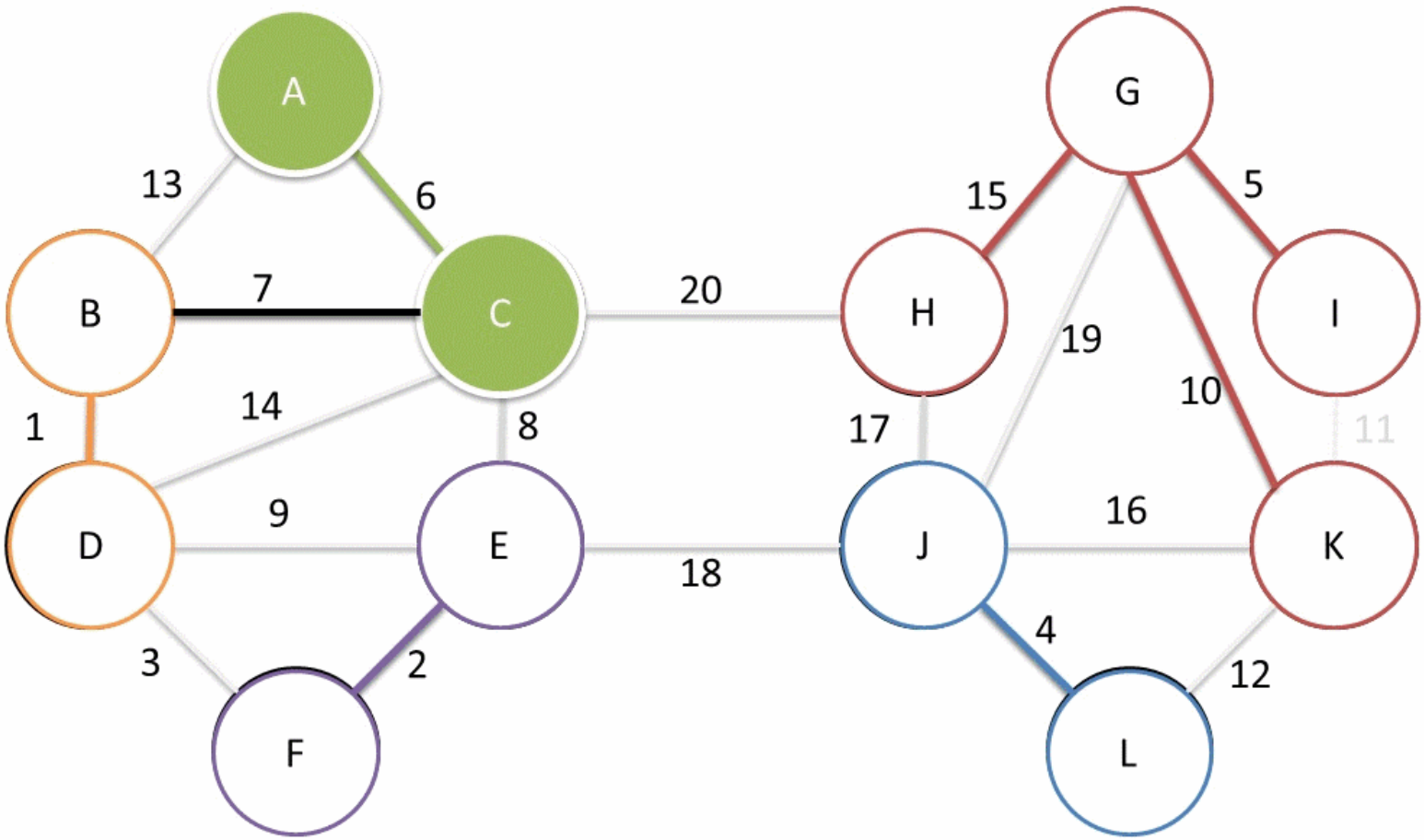
Boruvka algorithm, 1926



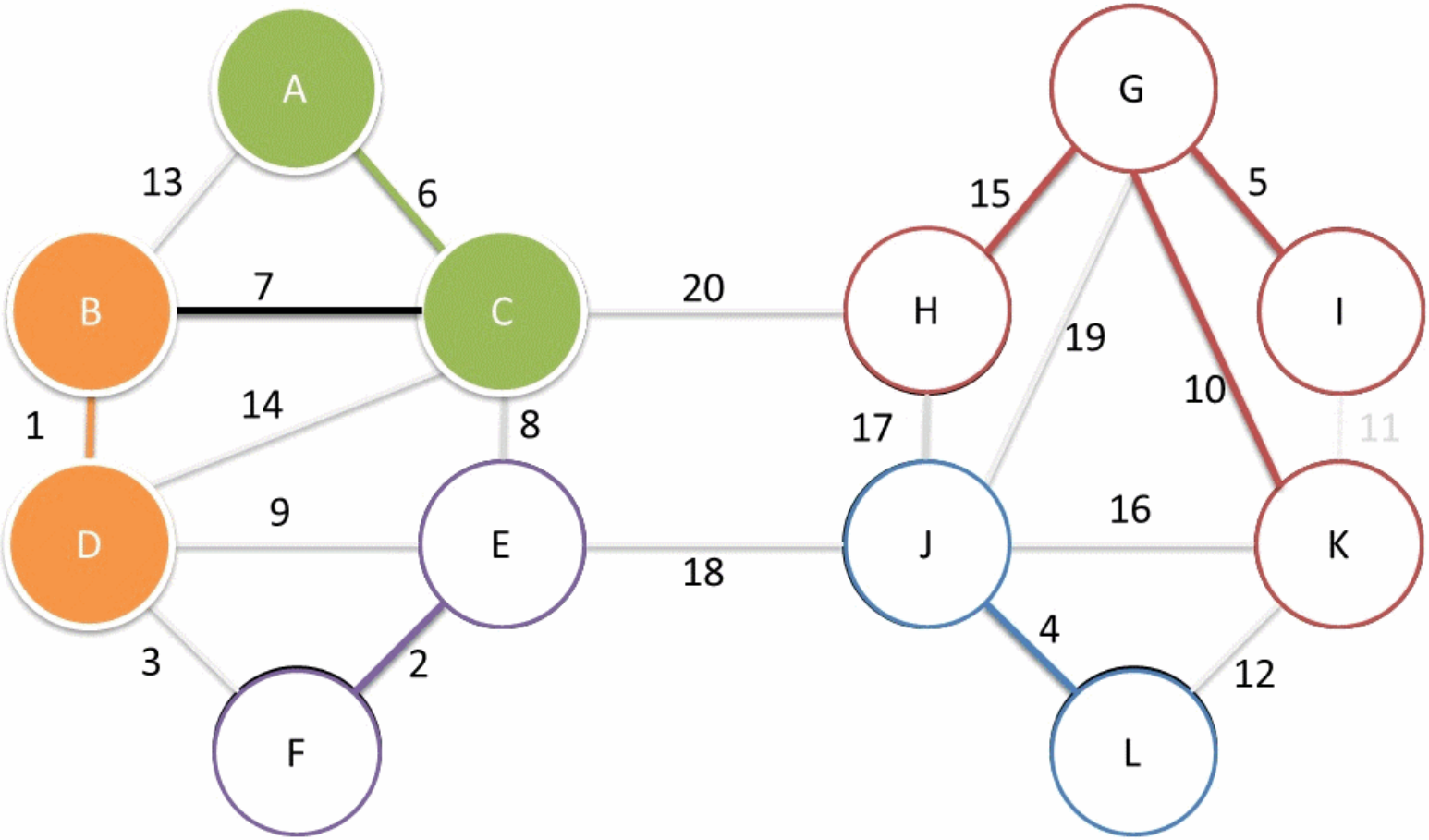
Boruvka algorithm, 1926



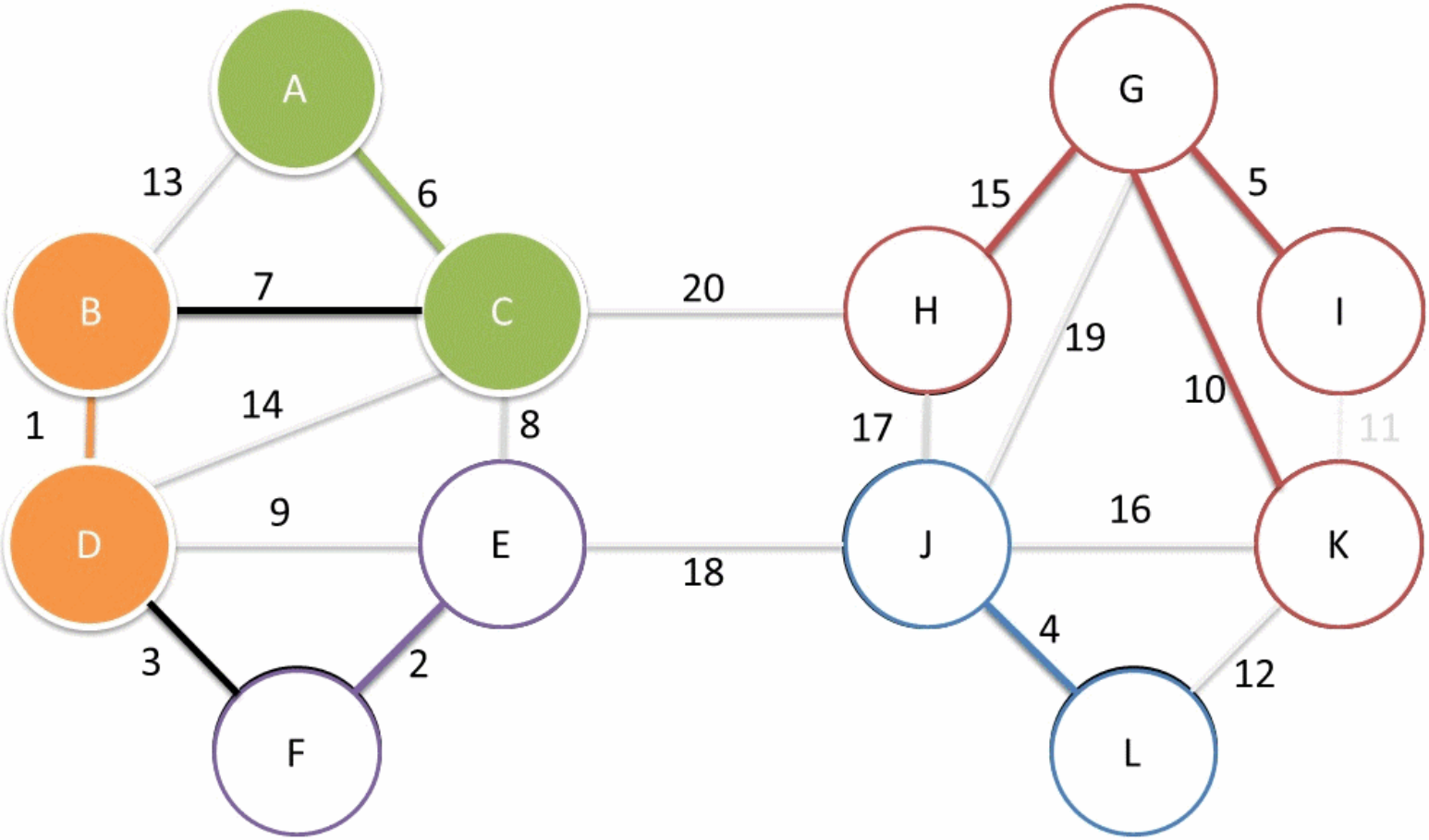
Boruvka algorithm, 1926



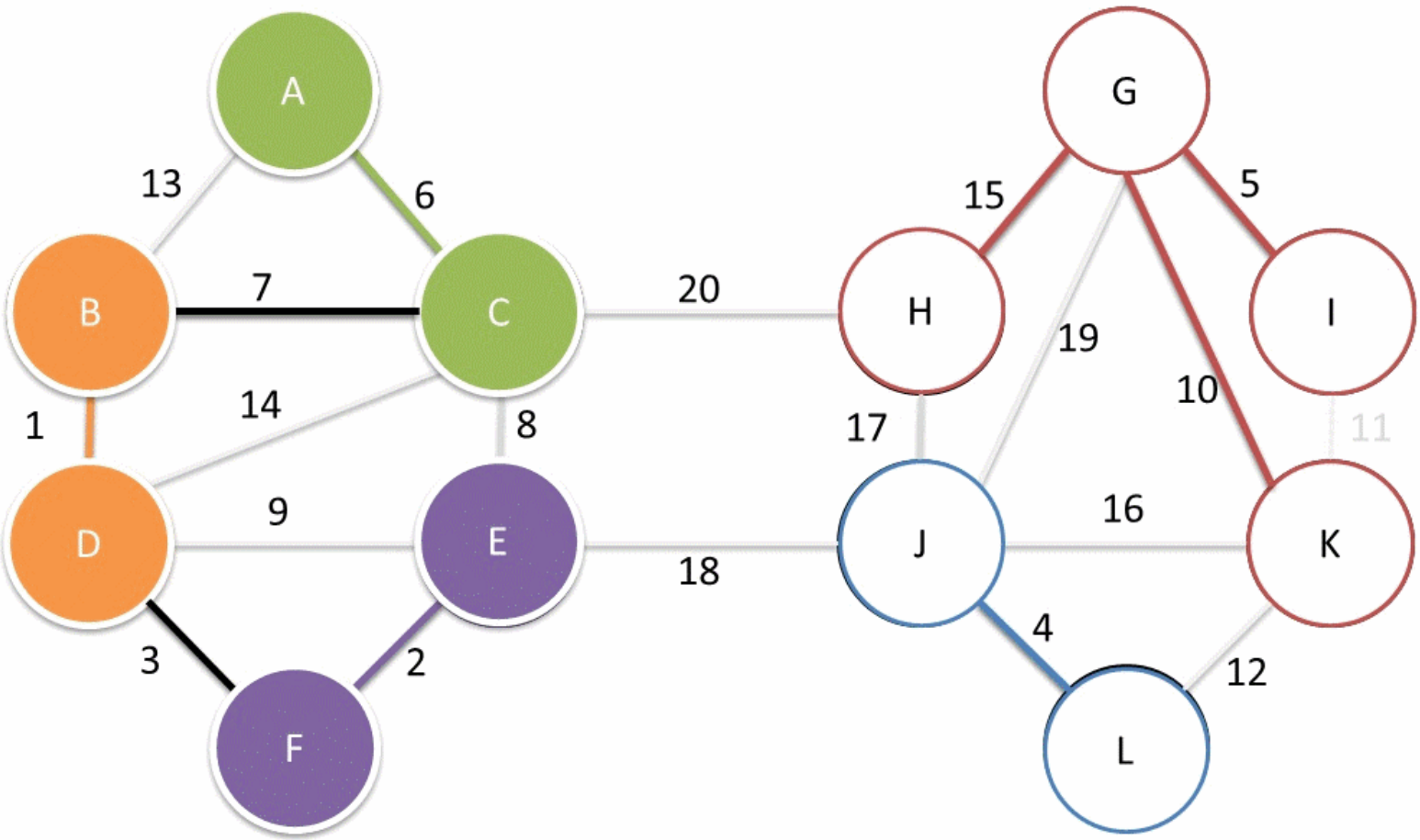
Boruvka algorithm, 1926



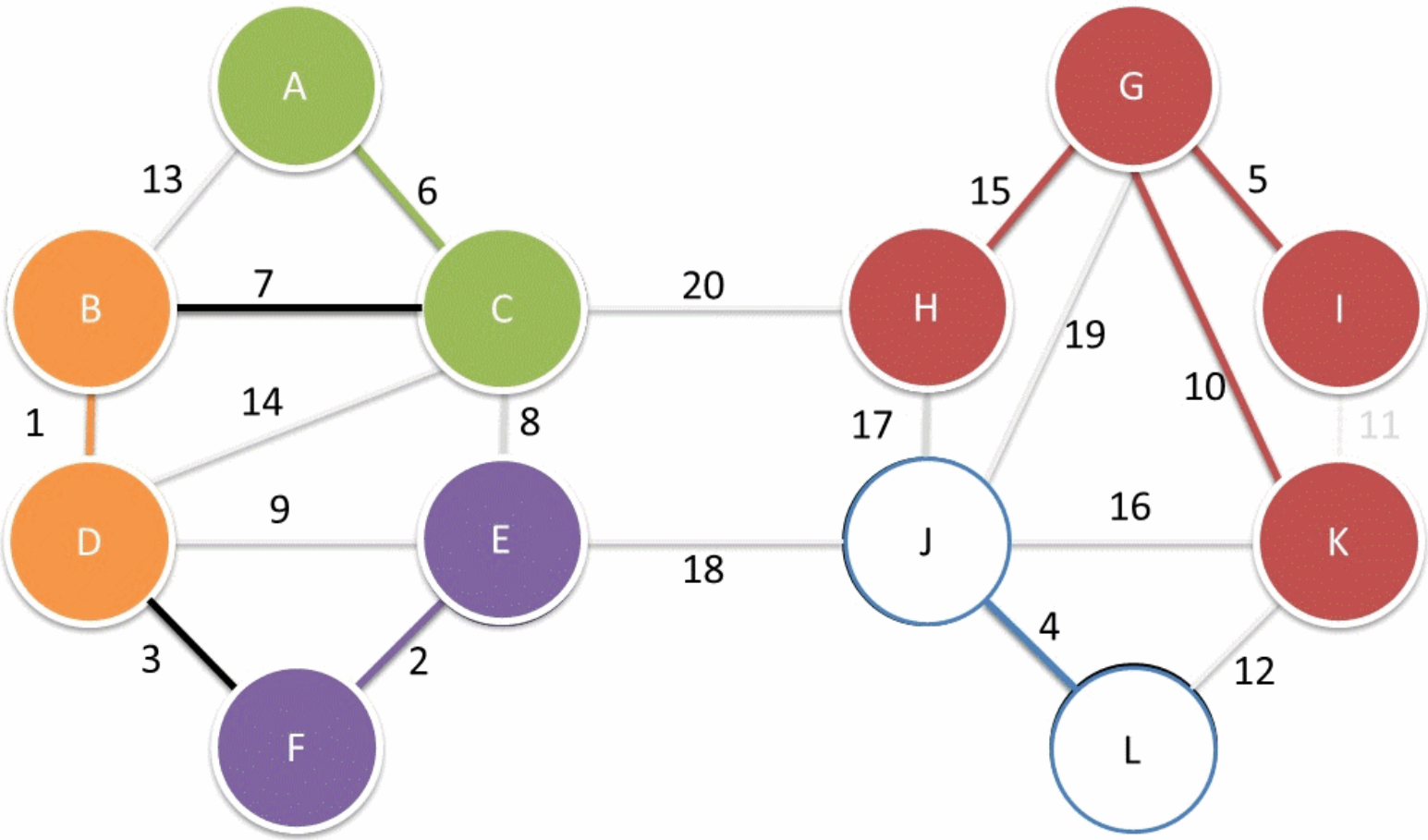
Boruvka algorithm, 1926



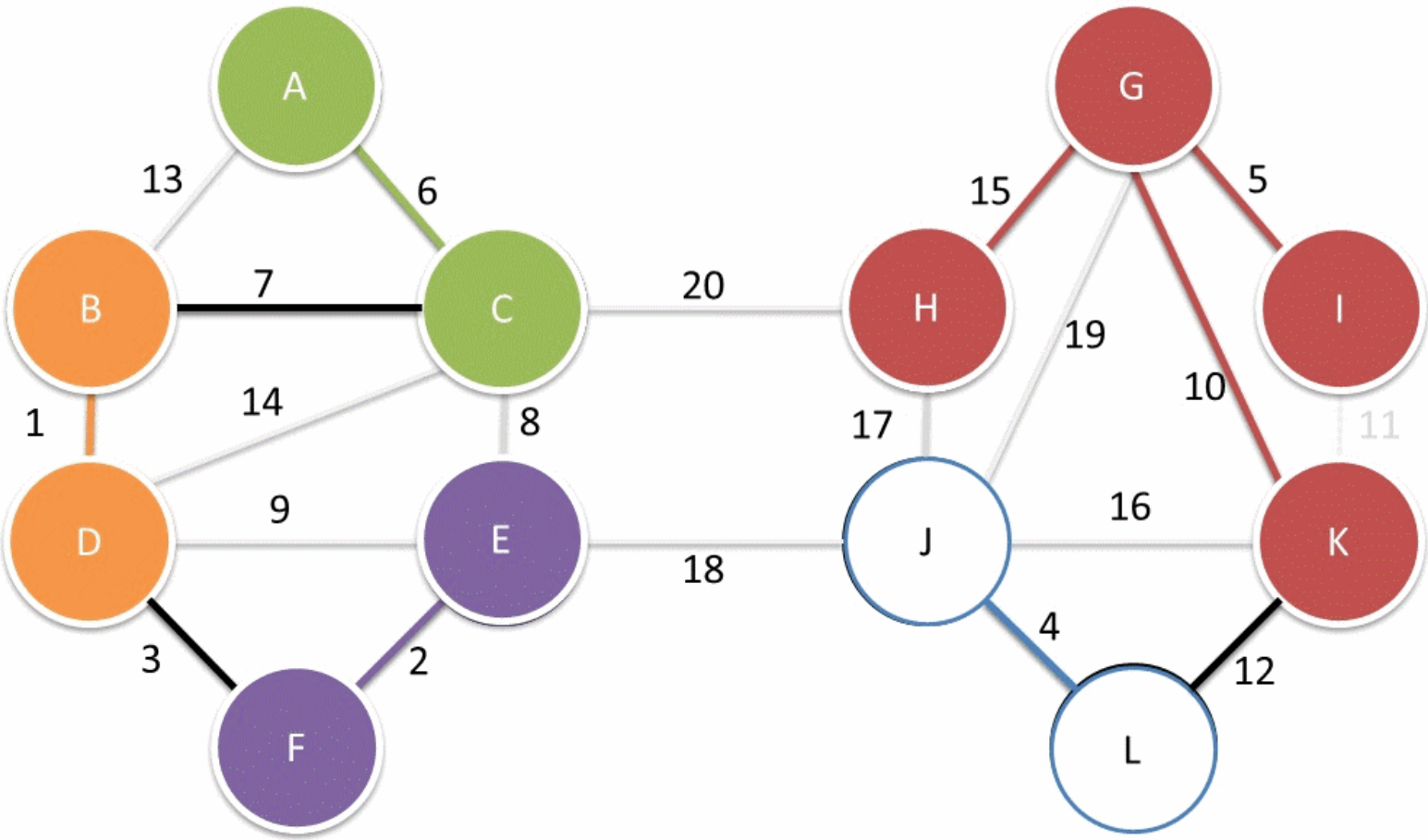
Boruvka algorithm, 1926



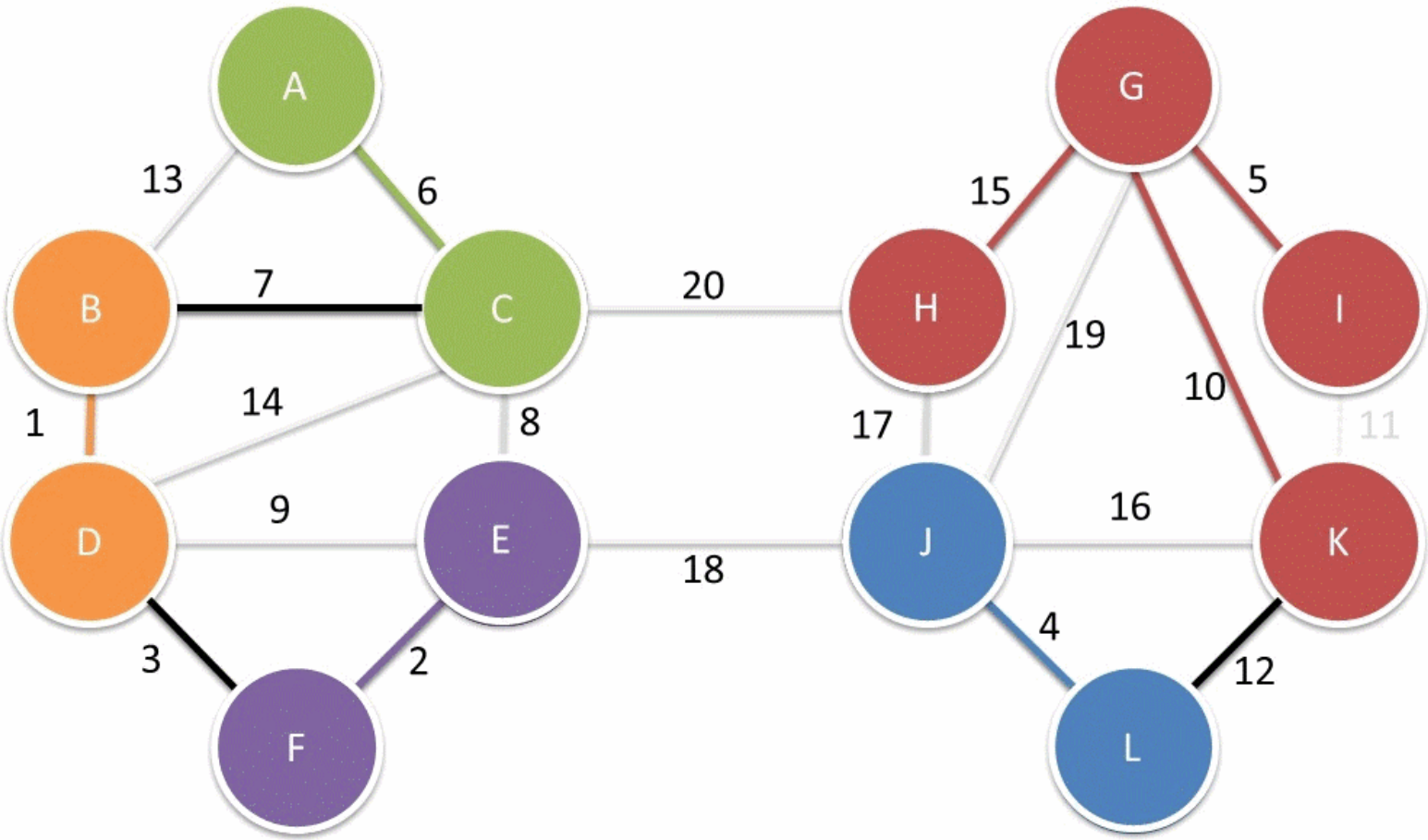
Boruvka algorithm, 1926



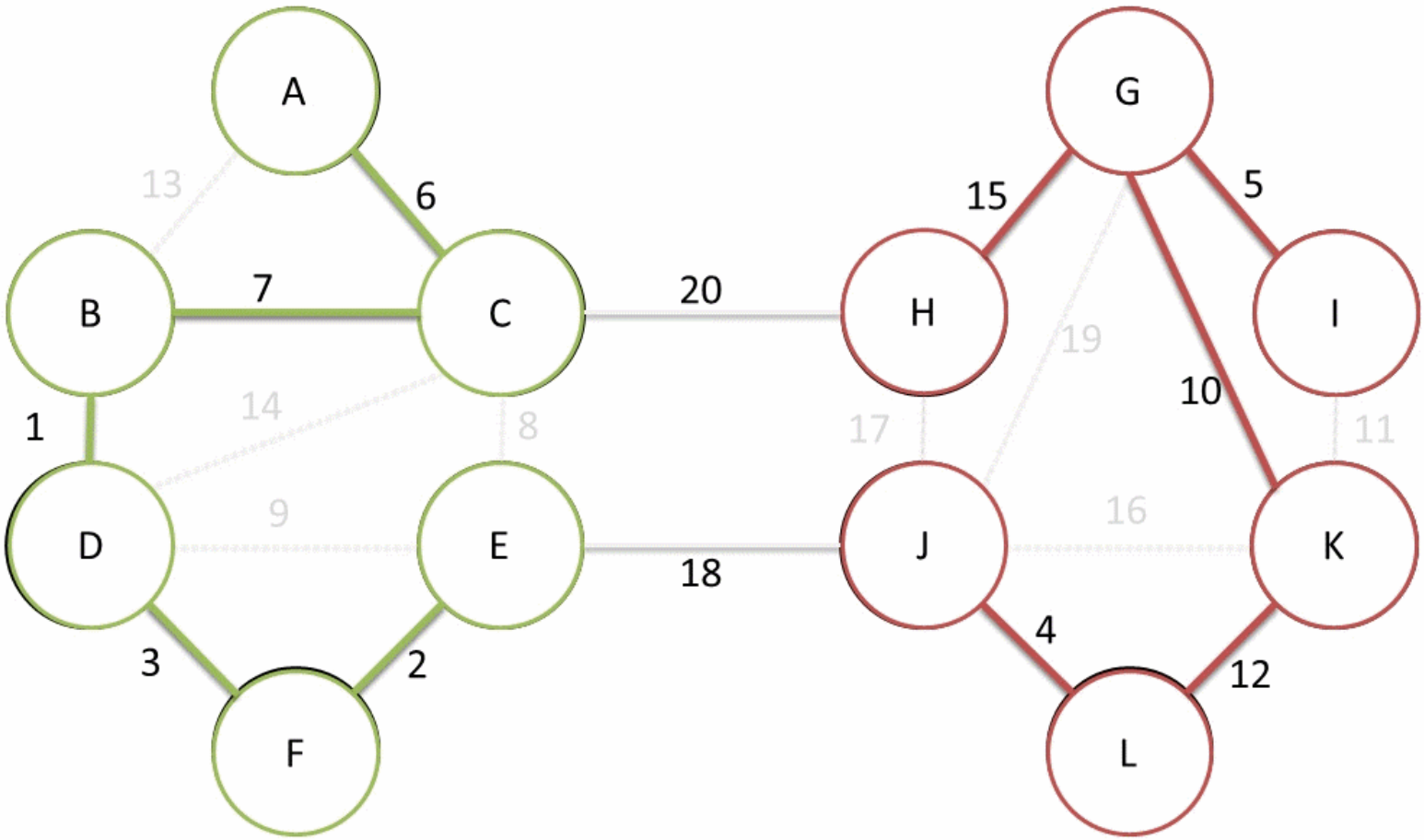
Boruvka algorithm, 1926



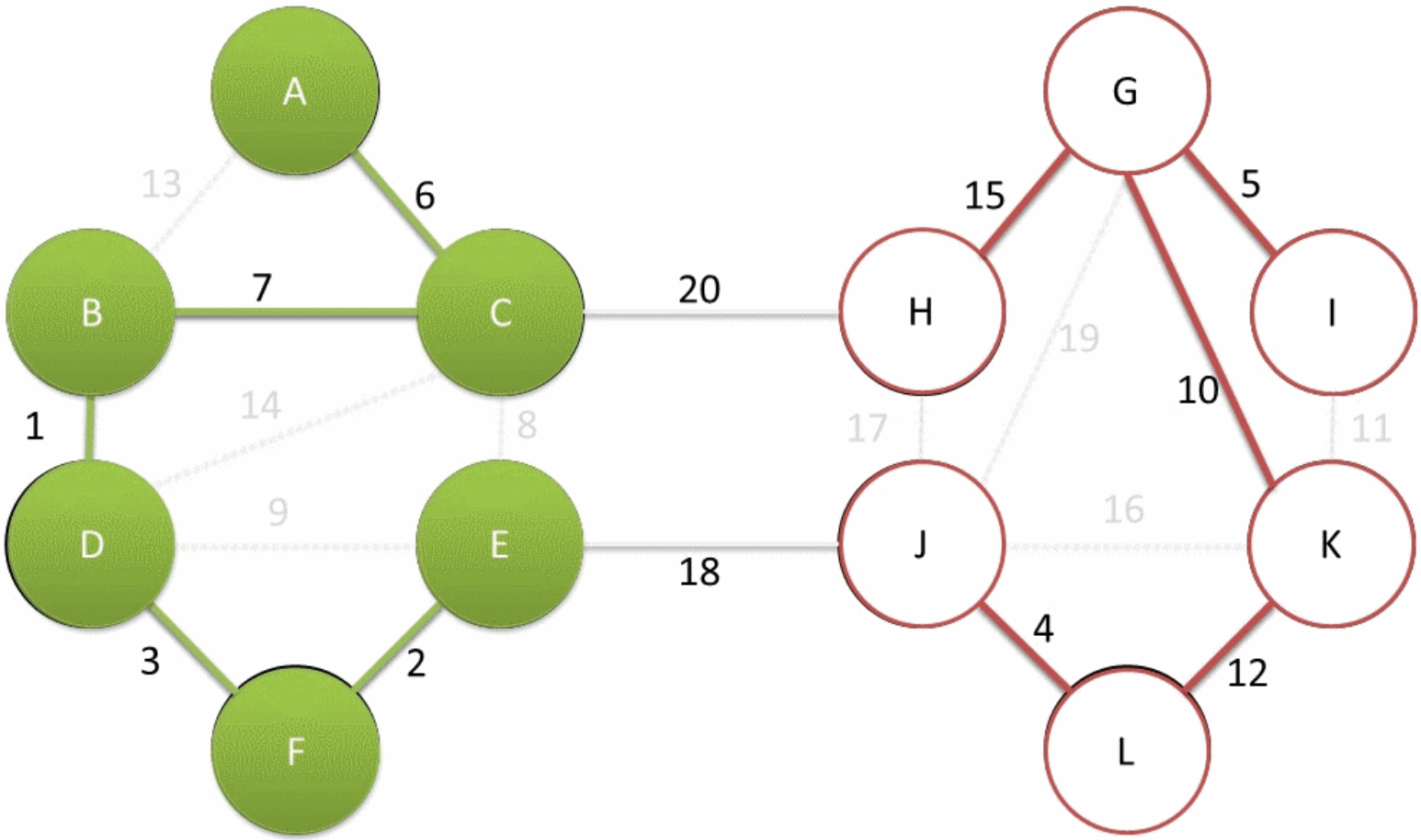
Boruvka algorithm, 1926



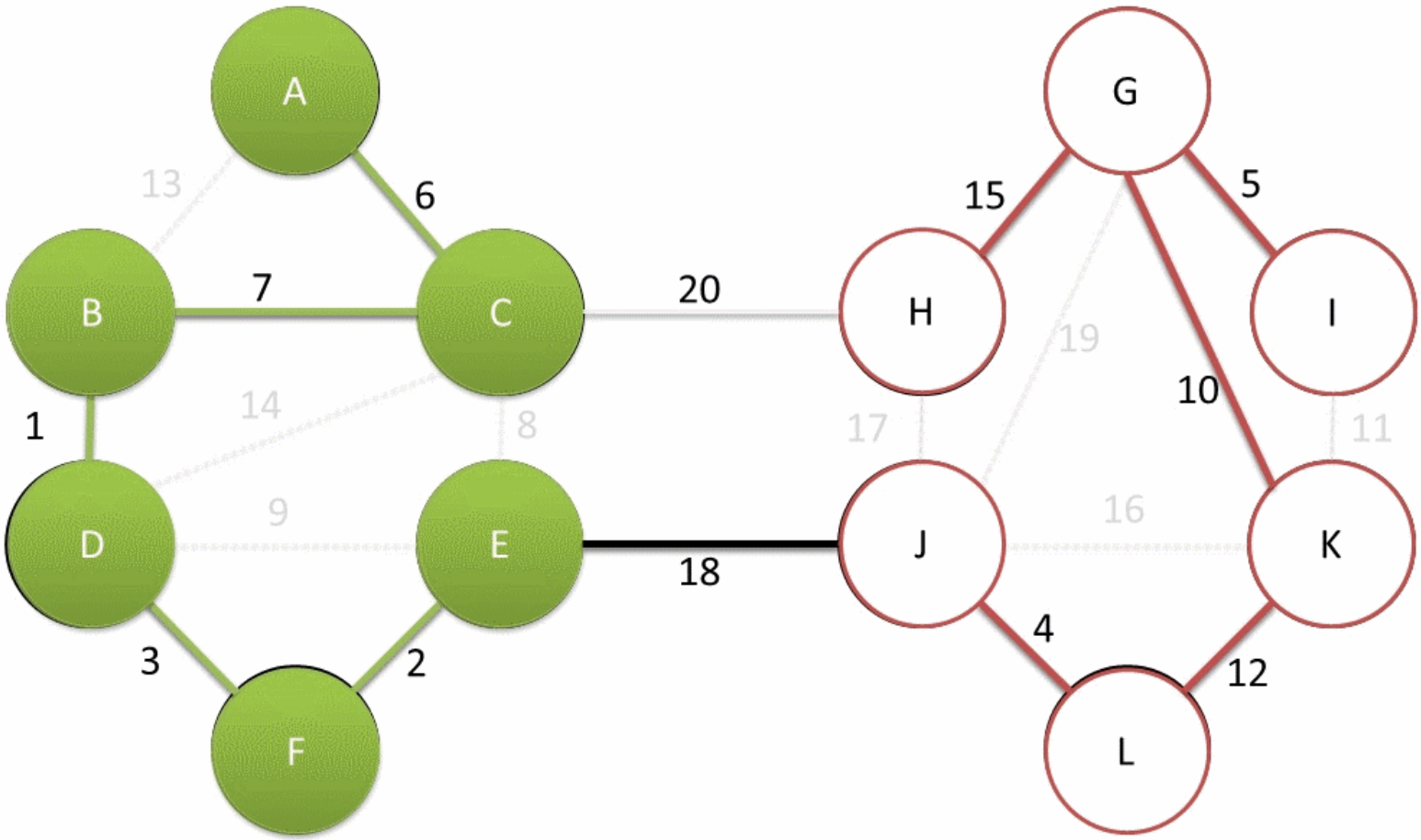
Boruvka algorithm, 1926



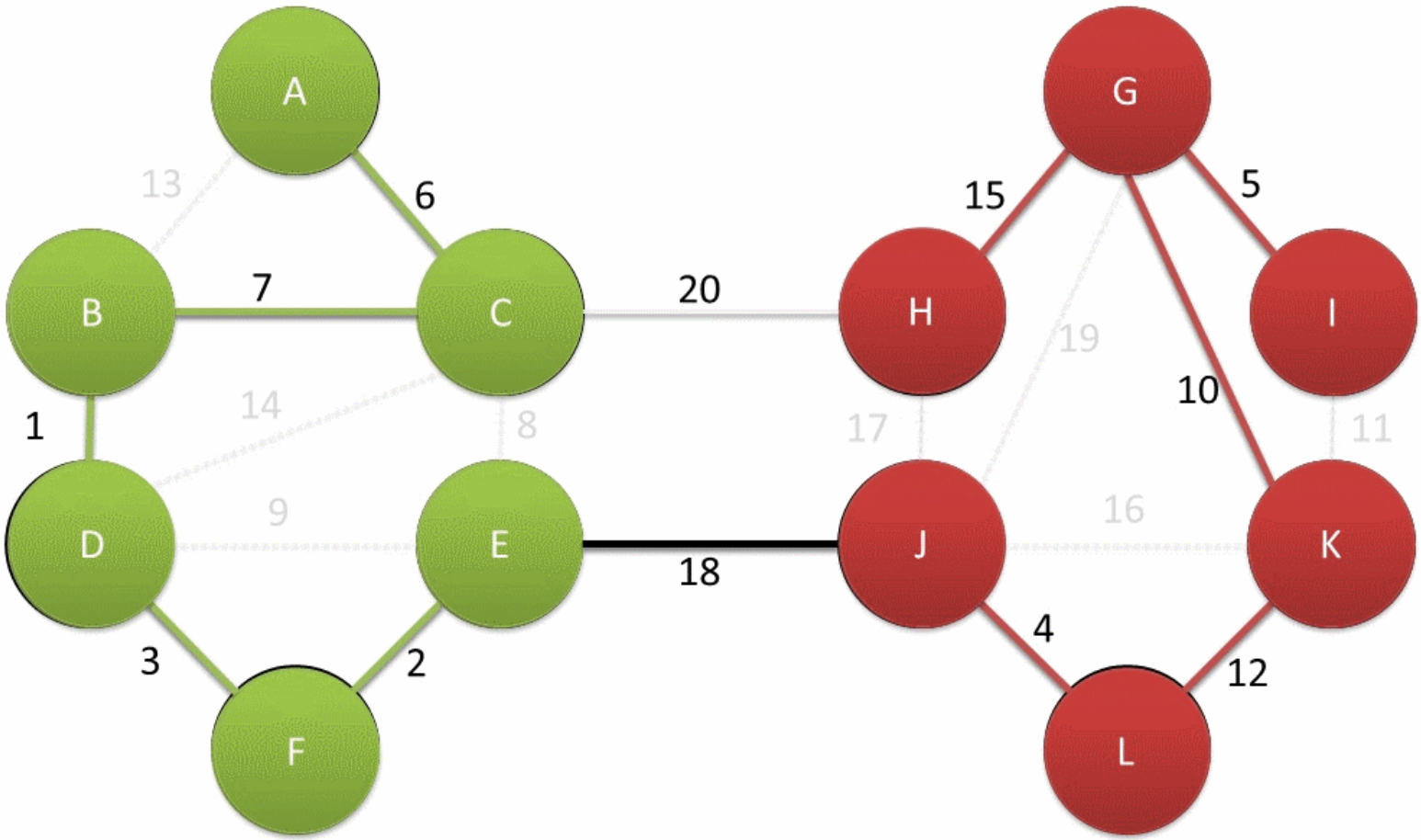
Boruvka algorithm, 1926



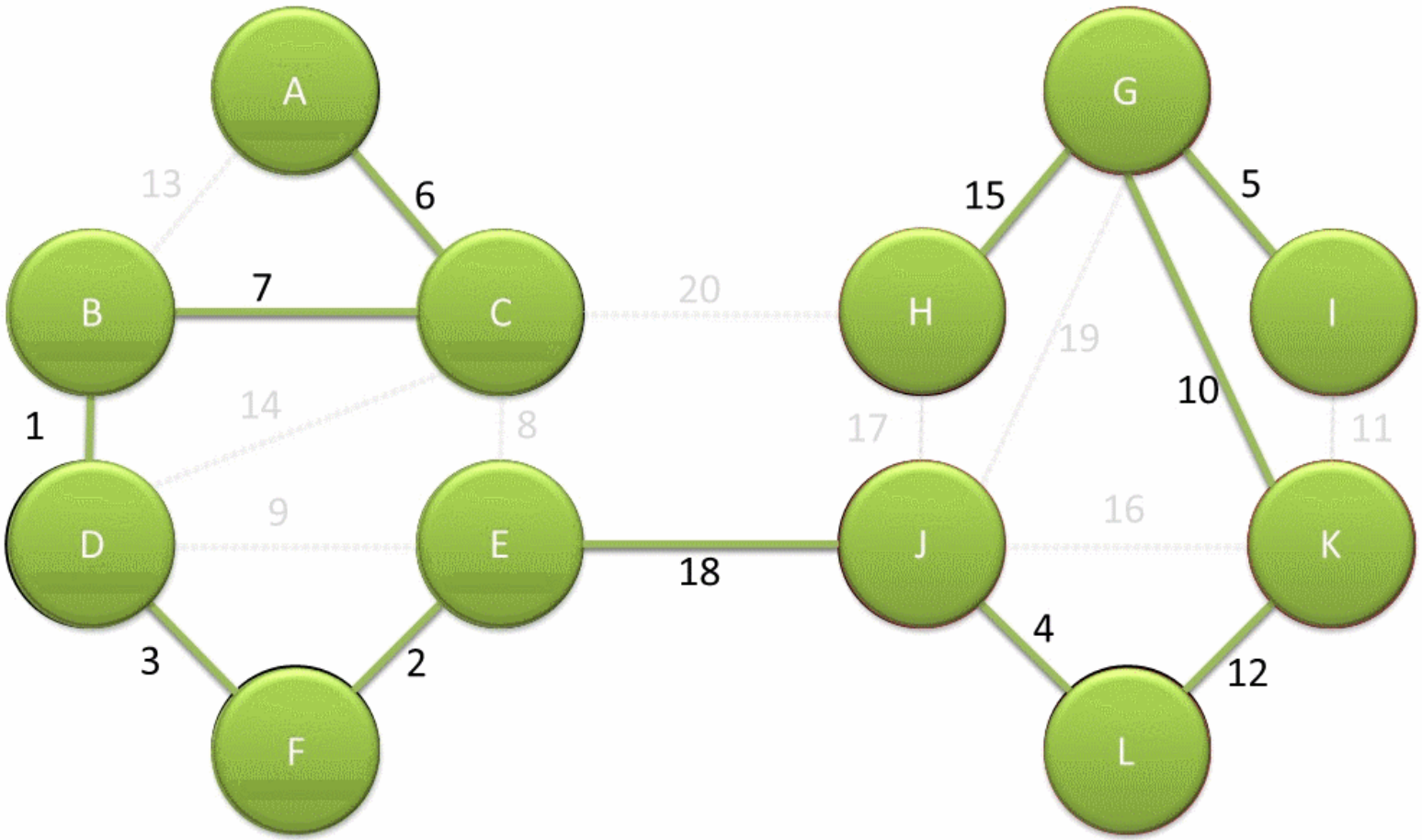
Boruvka algorithm, 1926



Boruvka algorithm, 1926



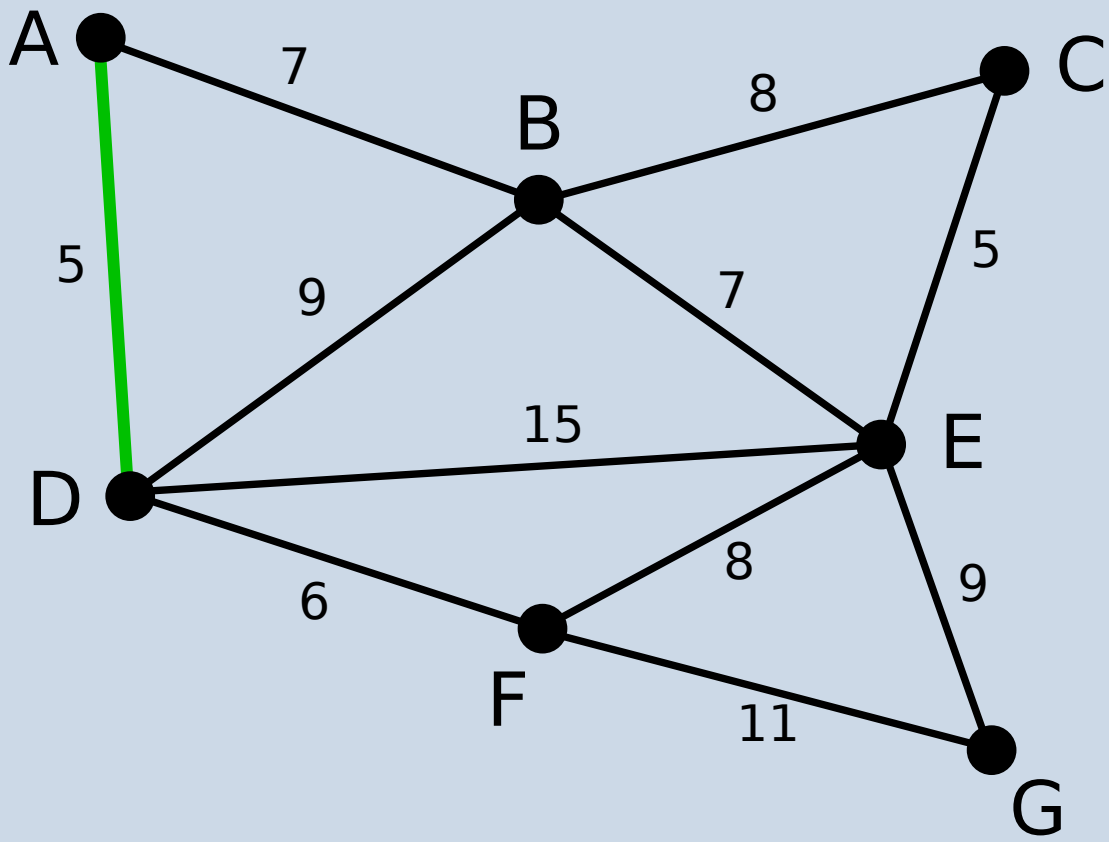
Boruvka algorithm, 1926



Kruskal's algorithm

- Sort the edges by weight
- Add all edges in this order so that they do not form a cycle

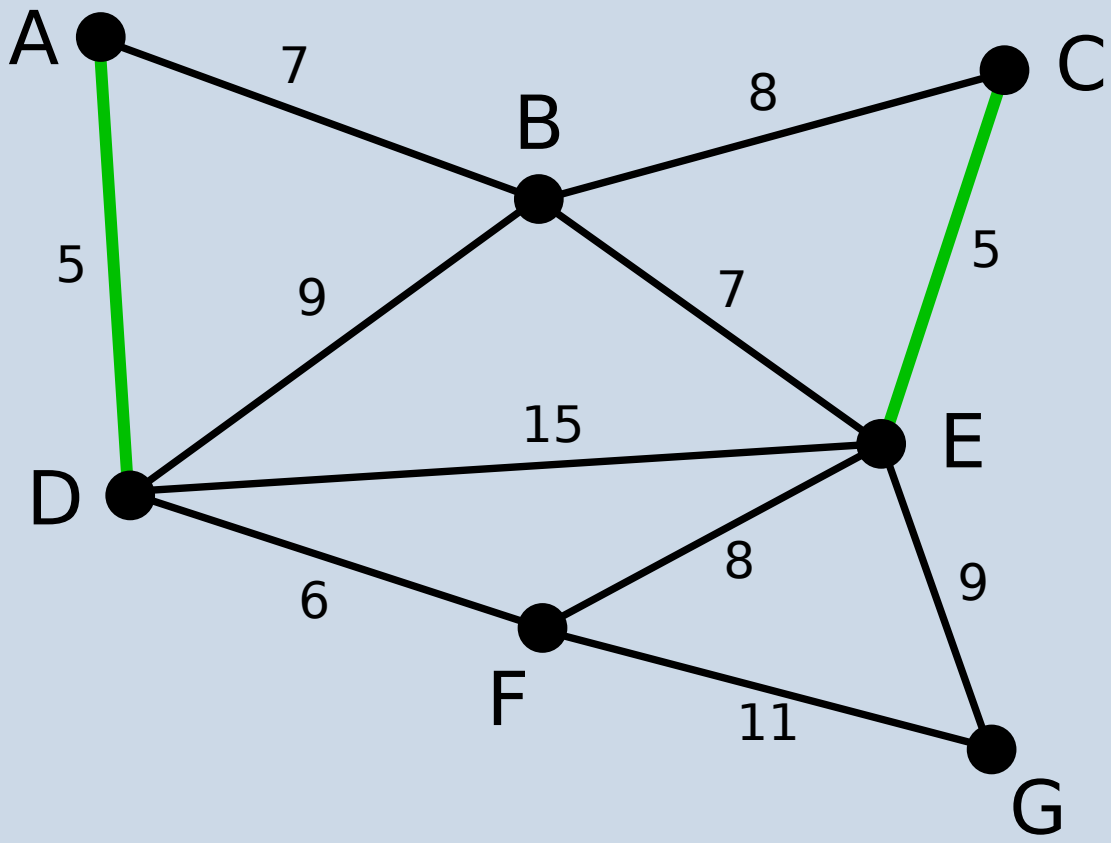
5	5
6	
7	7
8	8
9	9
11	
15	



Kruskal's algorithm

- Sort the edges by weight
- Add all edges in this order so that they do not form a cycle

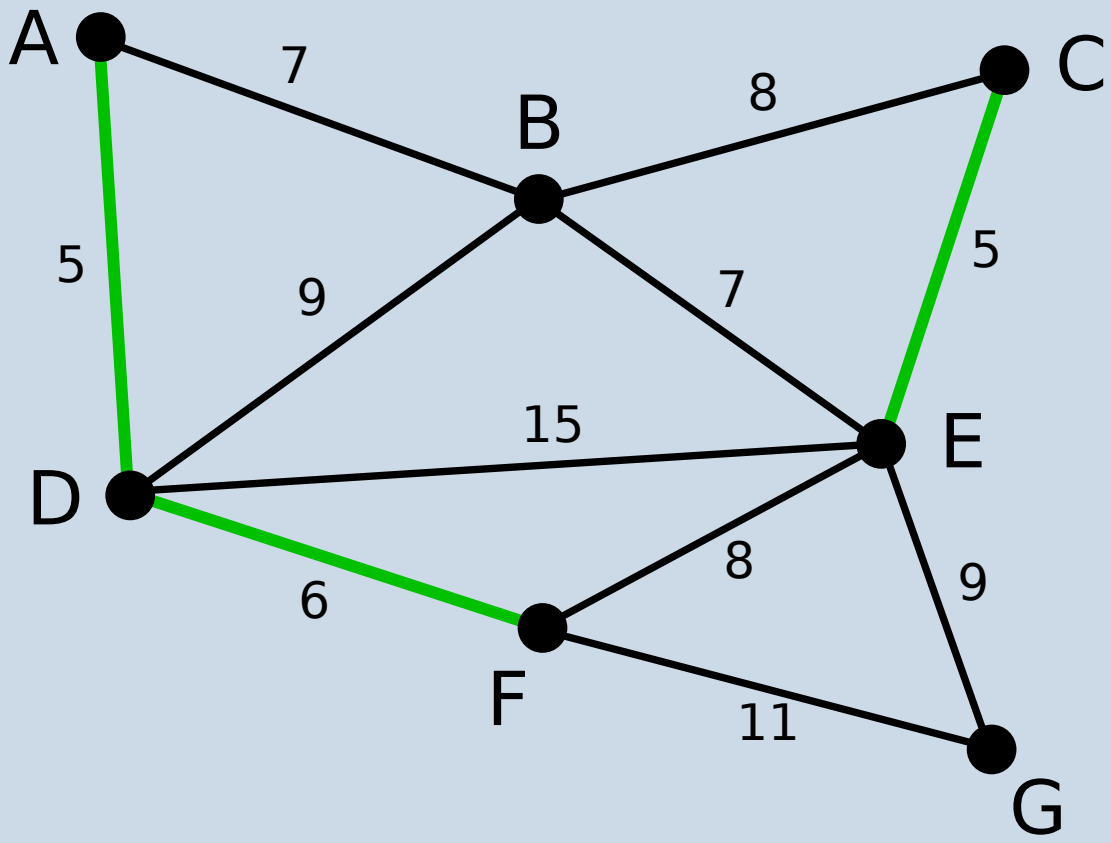
5	5
6	
7	7
8	8
9	9
11	
15	



Kruskal's algorithm

- Sort the edges by weight
- Add all edges in this order so that they do not form a cycle

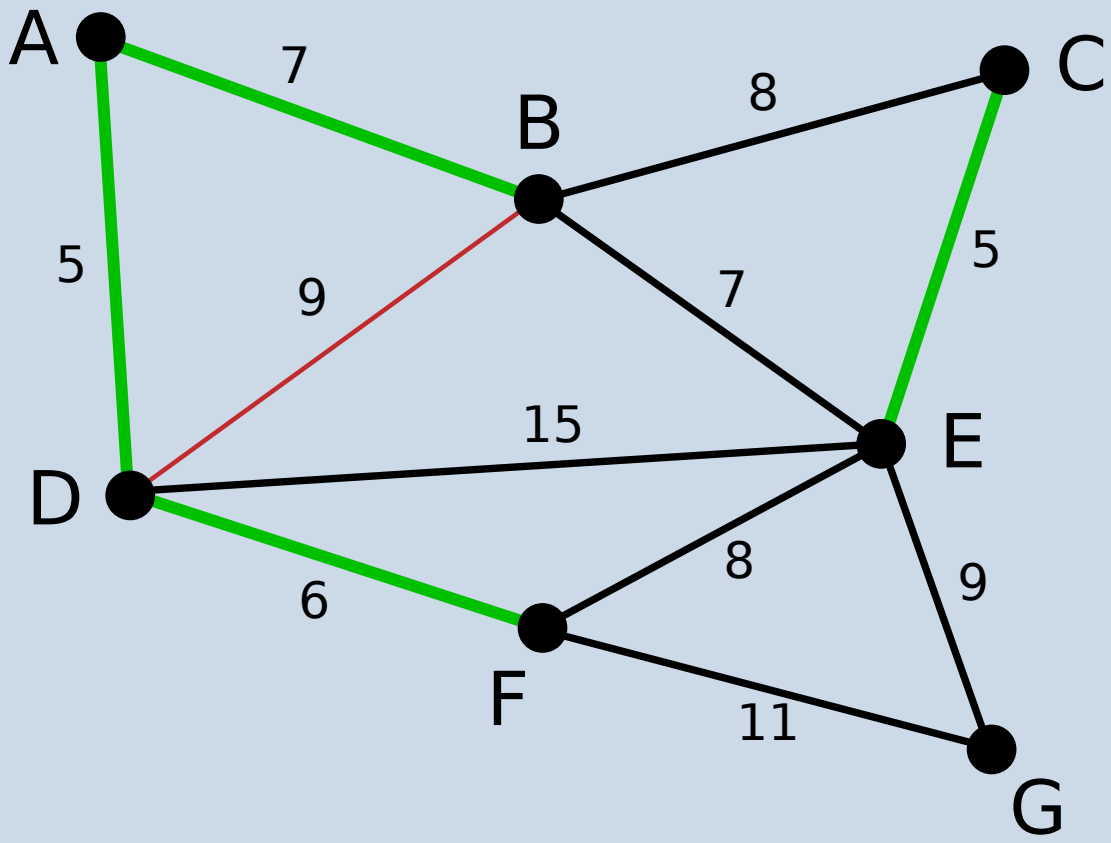
5	5
6	
7	7
8	8
9	9
11	
15	



Kruskal's algorithm

- Sort the edges by weight
- Add all edges in this order so that they do not form a cycle

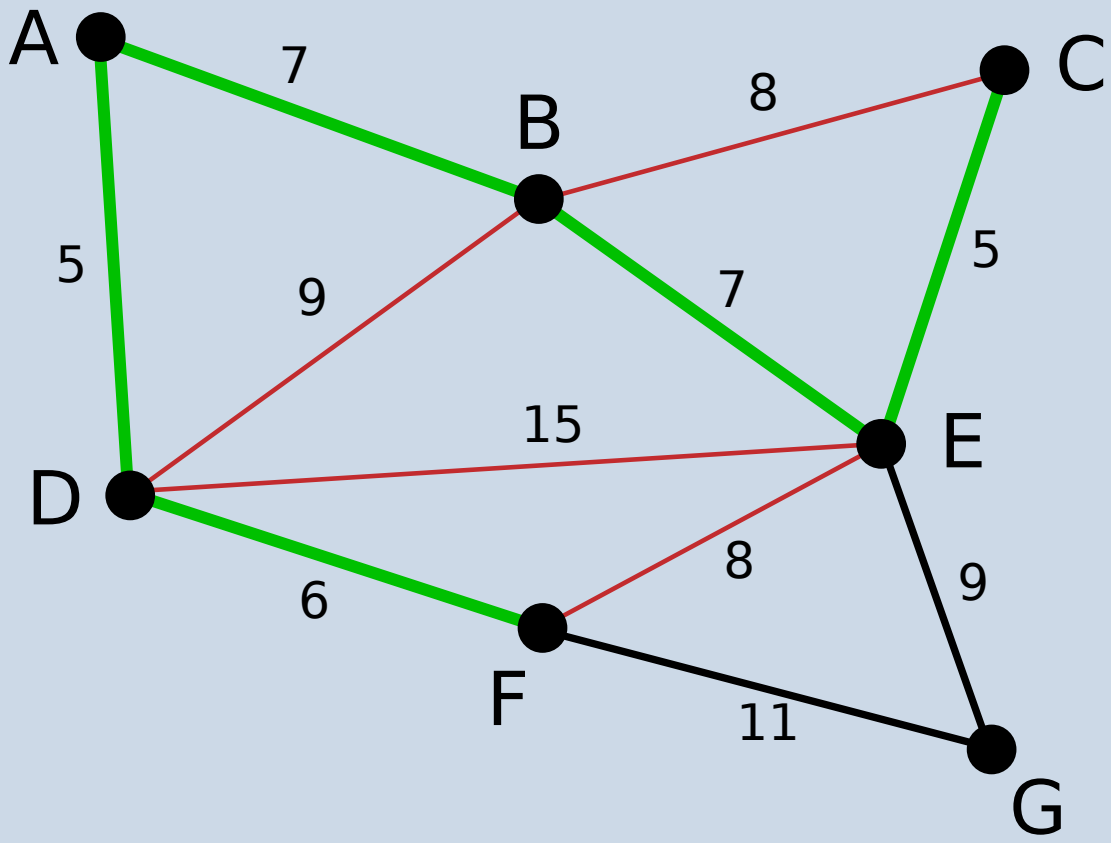
5	5
6	
7	7
8	8
9	9
11	
15	



Kruskal's algorithm

- Sort the edges by weight
- Add all edges in this order so that they do not form a cycle

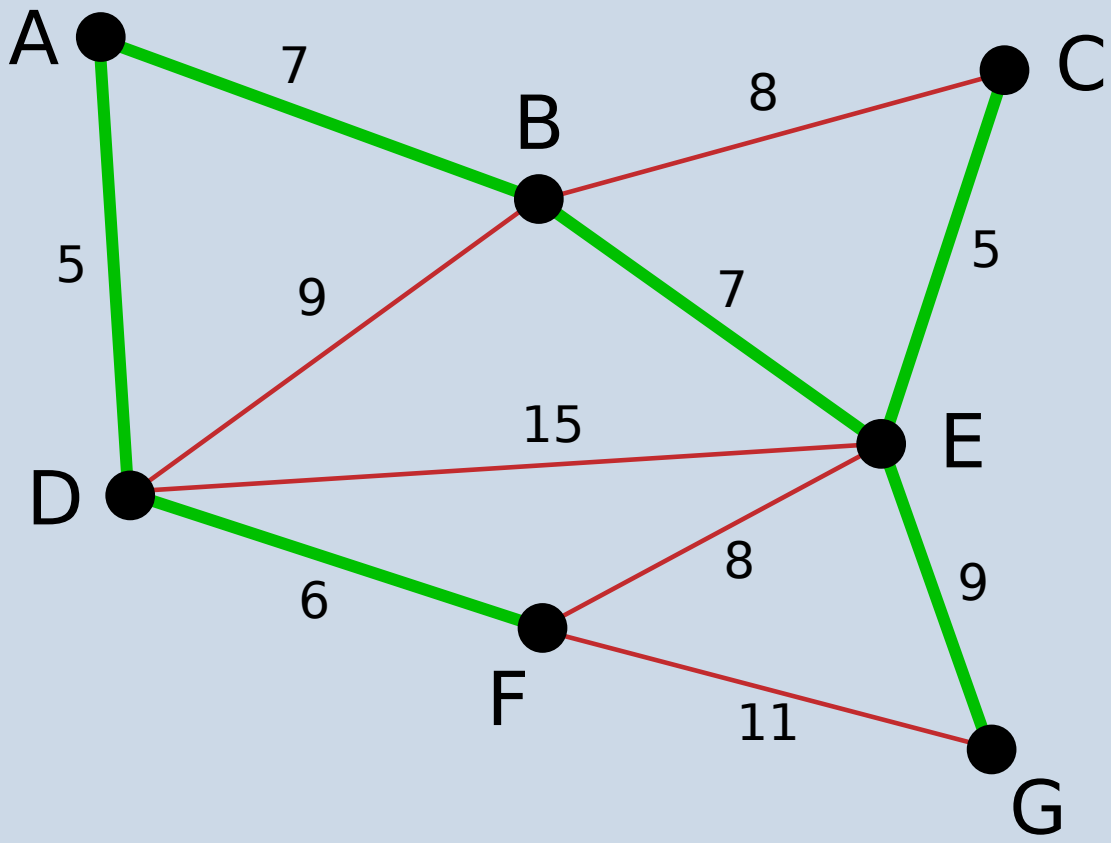
5	5
6	
7	7
8	8
9	9
11	
15	



Kruskal's algorithm

- Sort the edges by weight
- Add all edges in this order so that they do not form a cycle

5	5
6	
7	7
8	8
9	9
11	
15	

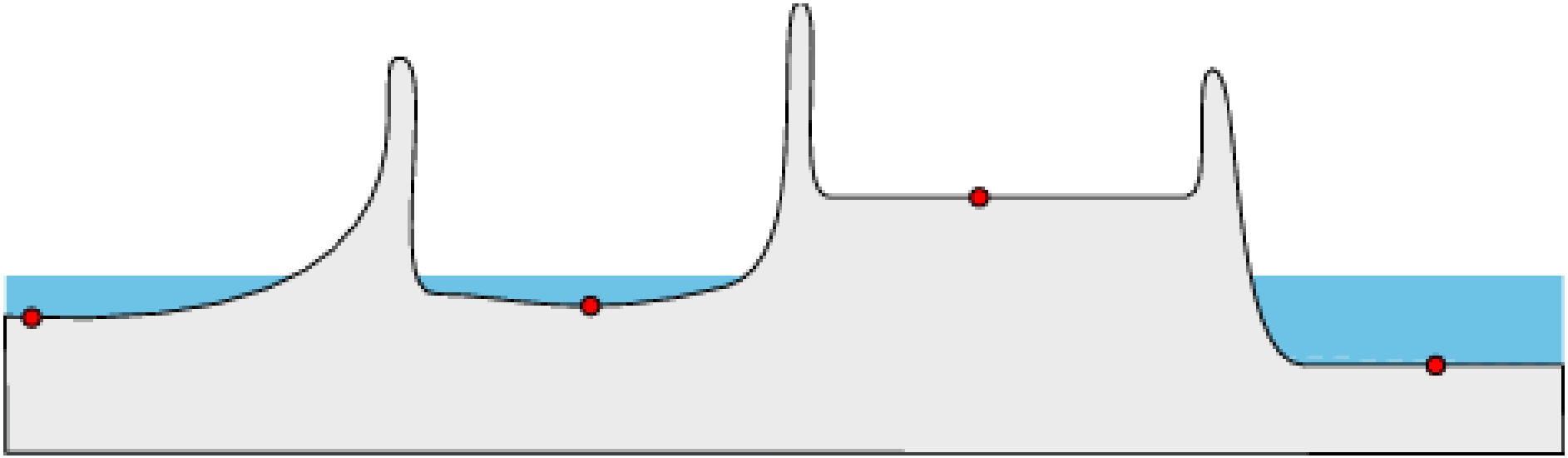


Section 4

Watershed

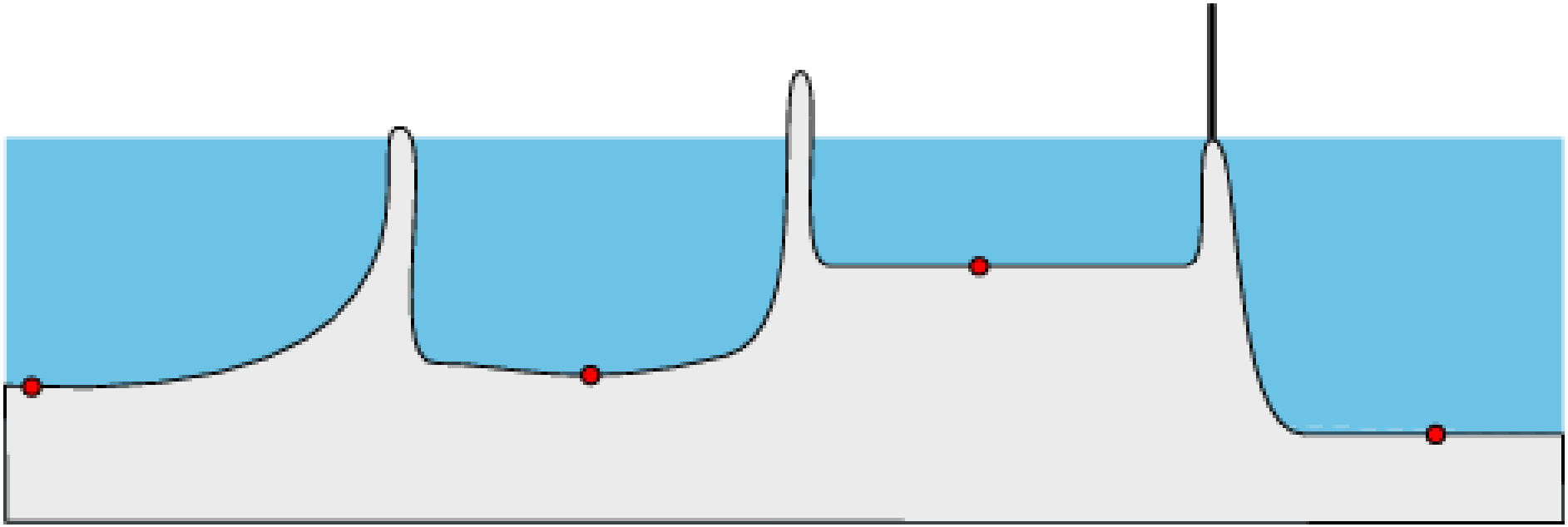
Watershed, catchment basin

Image processing tool: mathematical morphology



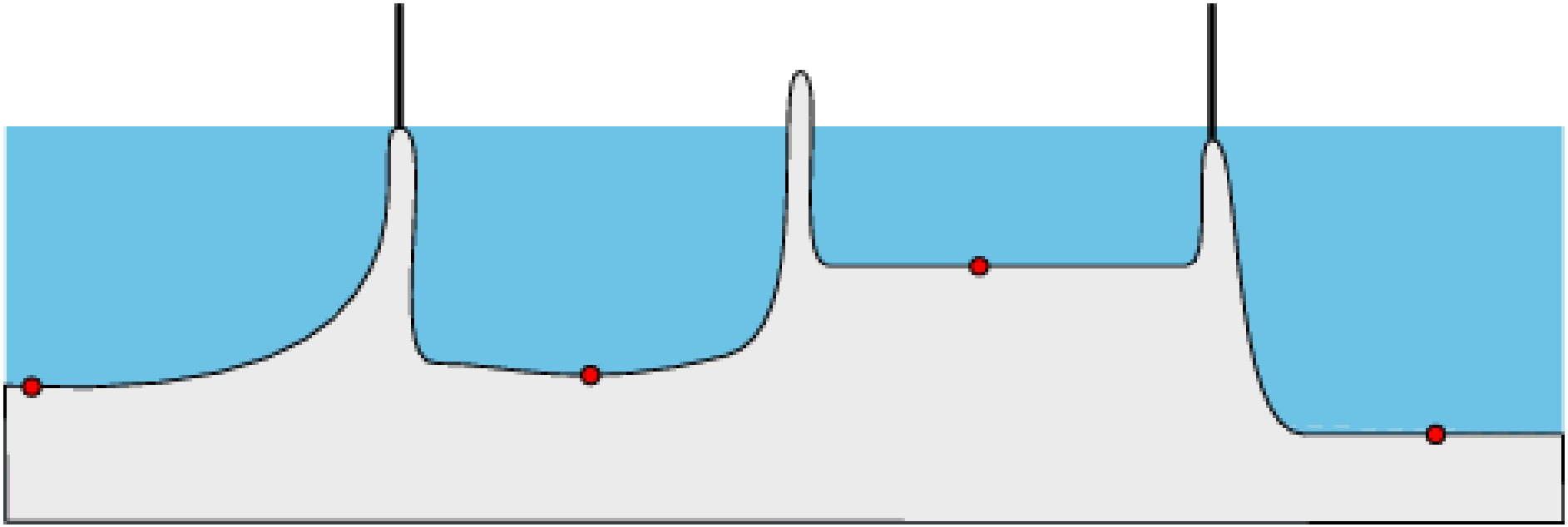
Watershed, catchment basin

Image processing tool: mathematical morphology



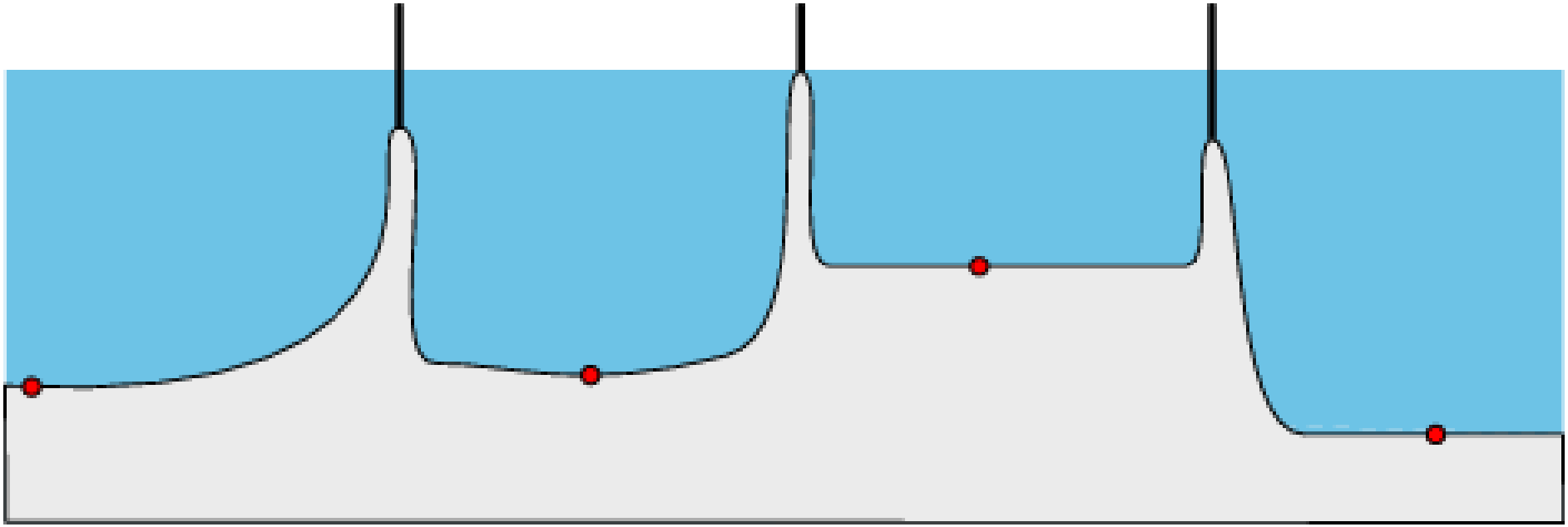
Watershed, catchment basin

Image processing tool: mathematical morphology



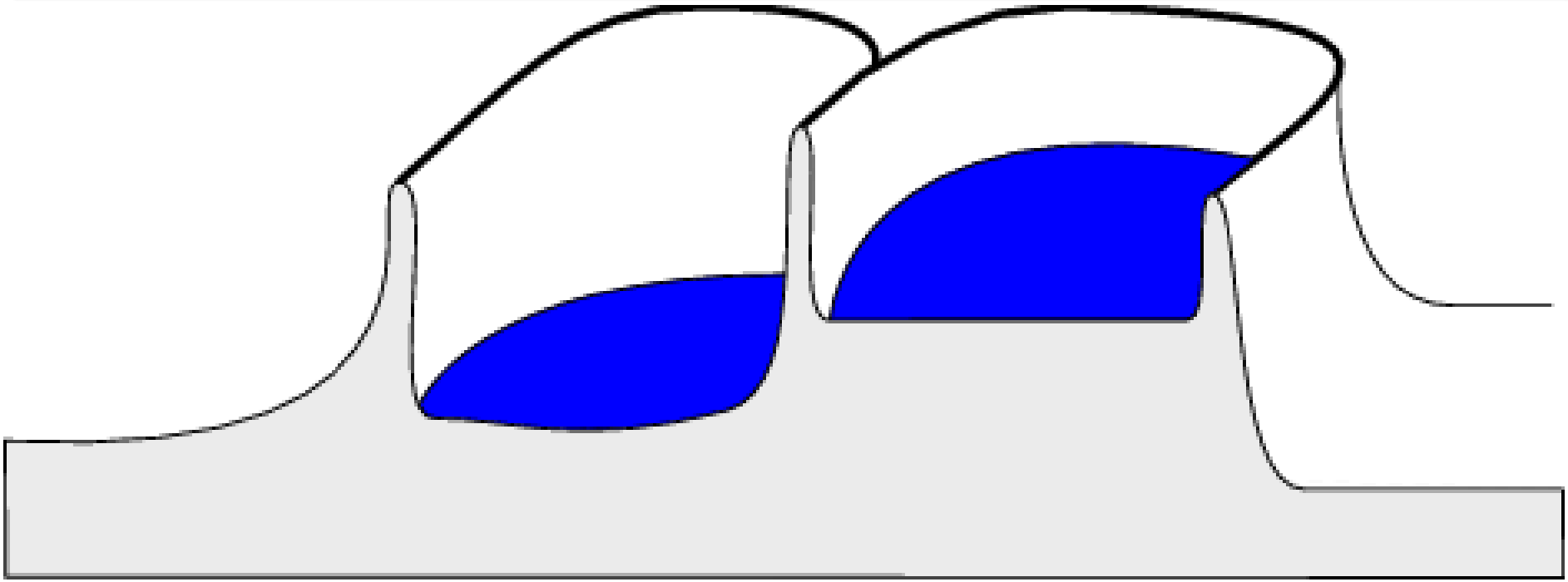
Watershed, catchment basin

Image processing tool: mathematical morphology



Watershed, catchment basin

Image processing tool: mathematical morphology



Geodesic distance [Roerdink and Meijster, 2001]

Definition (Geodesic distance between 2 points)

- $A \subseteq \mathbb{R}^n$ or $\mathbb{Z}^n$.
- $a, b \in A$.

Geodesic distance $d_A(a, b)$ between a and b on A is the path of minimal length on A that links a tp b (continuous case, **min** is **inf**).

Definition (Point to set geodesic distance)

$B \subset A$,

$$d_A(a, B) = \min_{b \in B}(d_A(a, b))$$

Geodesic influence zone

Definition (Influence zone)

- $B \subset A$, partitionned into k connected components, B_i, $i \in [1:k]$

The influence zone iz_A of $B_i \in A$ is defined by:

$$iz_A(B_i) = \{p \in A | \forall j \in [1:k] \setminus \{i\} : d_A(p, B_i) < d_A(p, B_j)\}$$

This is the definition of a **Voronoi** diagram.

SKIZ

Definition

L'union des zones d'influence est définie par: The union of influence zones is defined by:

$$IZ_A(B) = \bigcup_i iz_A(B_i)$$

Complementary set is called the SKIZ (skeleton by influence zone):

$$SKIZ_A(B) = A \setminus IZ_A(B)$$

Practically, the SKIZ connects all the points that are equidistant from at least 2 connected compoments B_i

Definition in continuous case

Definition (Topographic distance)

Not a geodesic distance. See it as a positive elevation change.

- $D \subset \mathbb{R}^n$ connected (spatial support).
- f is a function $\mathcal{C}^2(D)$ (Morse function, twice differentiable almost everywhere).
- 2 point $p, q \in D$
- γ is a **path** (smooth curve) included in D, $\gamma(0) = p$, $\gamma(1) = q$.

$$T_f(p, q) = \inf_{\gamma} \int_{\gamma} \|\nabla f_{|\gamma}(s))\| \mathrm{d}s \tag{1}$$

Watershed transform

Definition (Catchment basin CB)

Let $f \in \mathcal{C}(D)$, and $\{m_k\}_{k \in I}$ its minima.

$$CB(m_i) = \{ \quad x \in D | \forall j \in I \setminus \{i\} : \\ f(m_i) + T_f(x, m_i) < f(m_j) + T_f(x, m_j)\}$$

The watershed is the set of dams:

Definition (Watershed)

$$Wshed(f) = D \cap \left(\bigcup_i CB(m_i) \right)^c$$

Comments

- If f is a metric, the topographic distance becomes a **geodesic distance**.
  The watershed is then the **SKIZ** (by construction, slope is constant).

Comments

- If f is a metric, the topographic distance becomes a **geodesic distance**.
  The watershed is then the **SKIZ** (by construction, slope is constant).
- Differentiability constraints are too high on f.

Comments

- If f is a metric, the topographic distance becomes a **geodesic distance**.
  The watershed is then the **SKIZ** (by construction, slope is constant).
- Differentiability constraints are too high on f.
- Minima of a general function are numerous, thus leading to **oversegmentation**.
  Watershed constrained by markers.


Bhatia, N. and Vandana (2010).
Survey of nearest neighbor techniques.
CoRR, abs/1007.0085.


Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009).
Introduction to algorithms.
MIT press.


de Berg, M. (1997).
Computational geometry: algorithms and applications.


Devadoss, S. L. and O'Rourke, J. (2011).
Discrete and Computational Geometry.


Roerdink, J. and Meijster, A. (2001).
The watershed transform: definitions, algorithms and parallelization strategies.
Fundamenta Informaticae, 41:187–228.


Vona, M. and Rus, D. (2005).

Voronoi toolpaths for pcb mechanical etch: Simple and intuitive algorithms with the 3d gpu.
Proceedings of the 2005 IEEE International Conference on Robotics and Automation, pages 2759–2766.