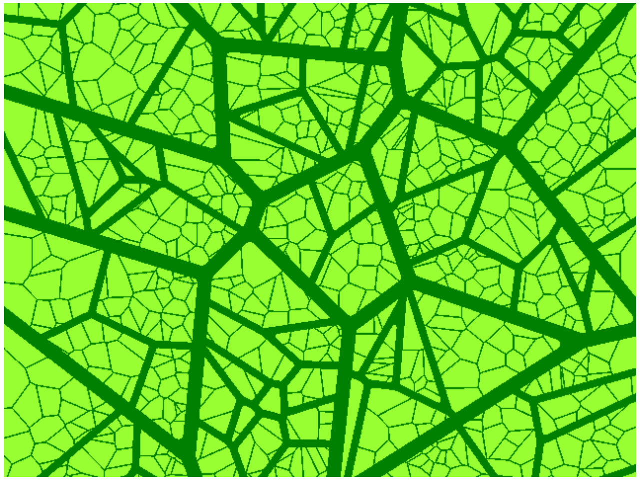
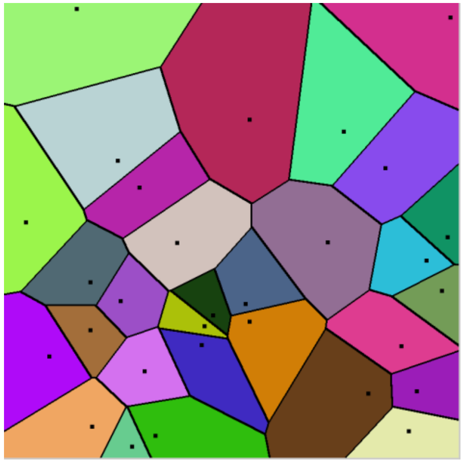
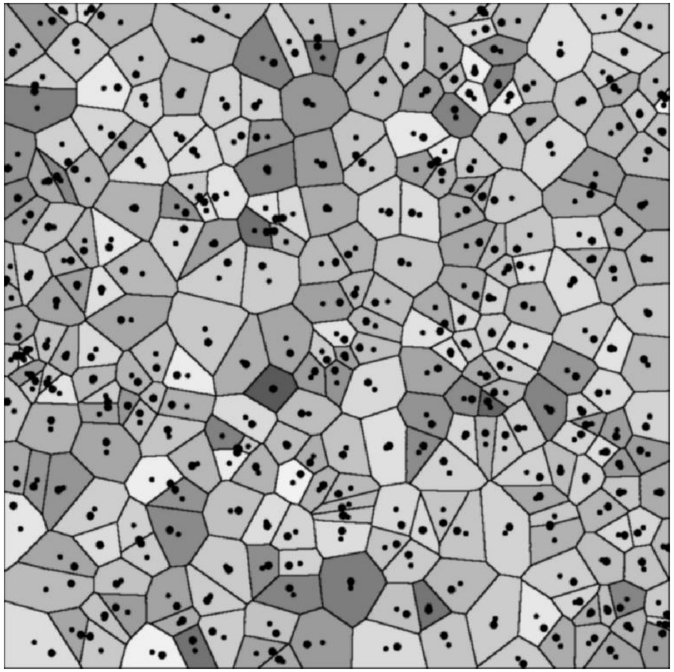


www.grasshopper3d.com

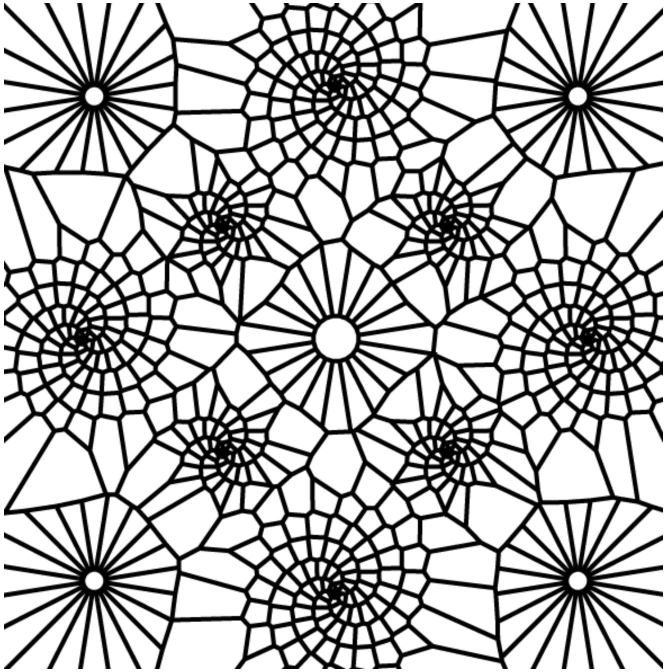


www.sonycsl.co.jp

Voronoi diagrams



cs.nyu.edu



http://newtextiles.media.mit.edu/?p=1906

Motivation

- Solves so-called **post office problem**
  - The goal is to plan a placement of new post office/supermarket/…
  - How many people will find the new supermarket attractive?
  - Let's consider the following simplified requirements:
    - The price of all goods is the same in all supermarkets
    - Total cost = cost for the goods + travelling cost to the supermarket
    - Travelling cost to the supermarket = Euclidean distance to the supermarket x fixed cost per distance unit
    - The goal of the customer is to minimize the costs
  - Consequence: the customers are using the service of the nearest supermarket

Motivation

- This model induces the division of the space to subregions according to the location of the supermarkets – each subregion contains **all points** being **closer** to the given supermarket than to any other supermarket
- Such a space division is called **Voronoi diagram**

Euclidean distance

- **Euclidean distance** between two points $P = [p_x, p_y]$ and $Q = [q_x, q_y]$ is defined as

$$|PQ| = \text{dist}(P,Q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

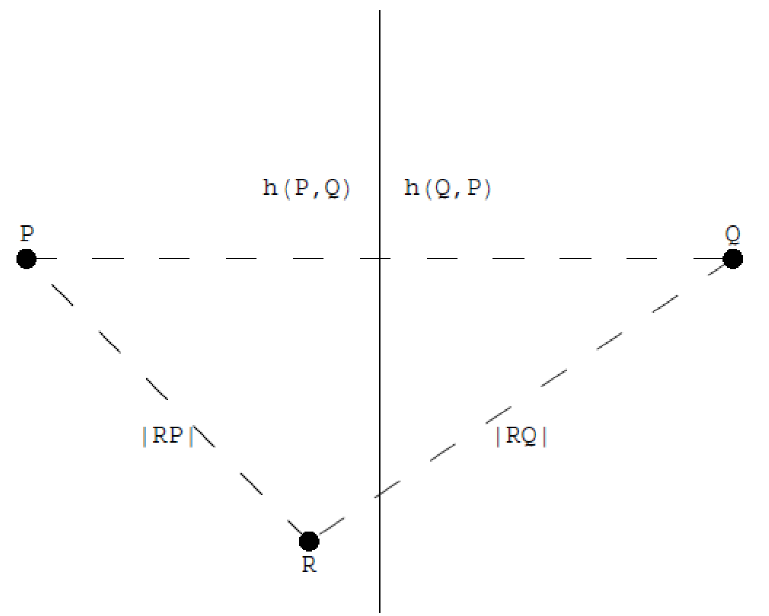
VD definition

- Let $P = \{P_1, \ldots, P_n\}$ be a set of n different points in space, called **generating points**.
- Voronoi diagram of P is the division to n cells connected with points P_i in that way that an arbitrary point Q lies in the cell of P_i only when

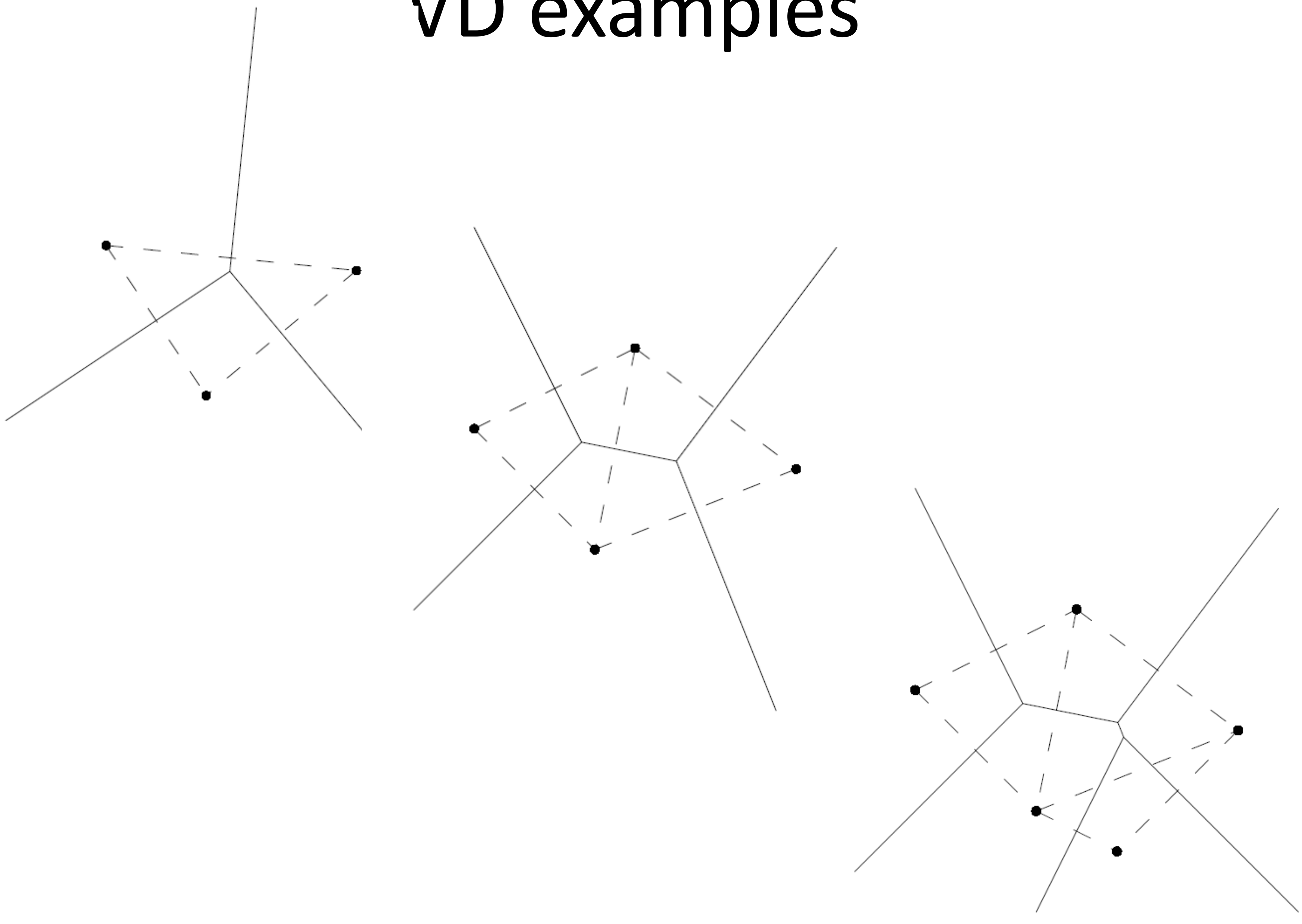
$$|QP_i| < |QP_j| \quad \text{for all } P_j \in P,\ j \neq i$$

VD definition

- Lets denote the Voronoi diagram of *P* as **Vor(*P*)**
- A cell of Vor(*P*), belonging to point *Pi*, is denoted as y(P_i) and we call it a **Voronoi cell of point P_i**

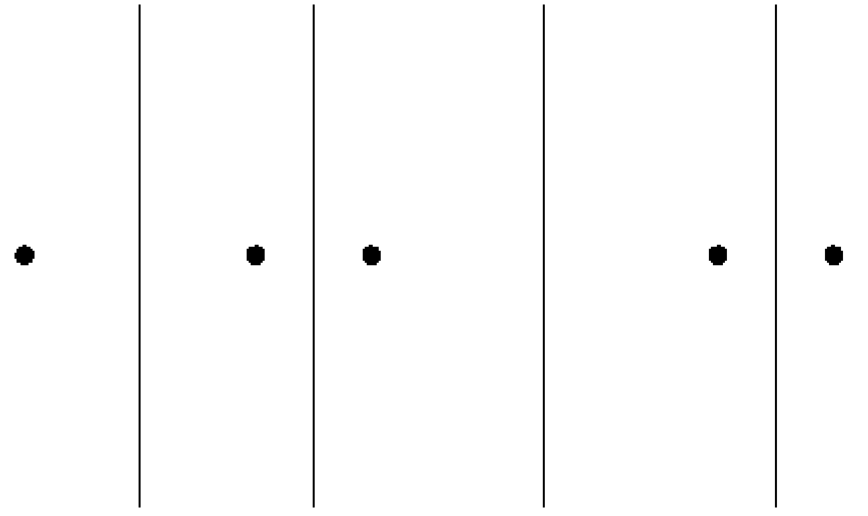


VD examples



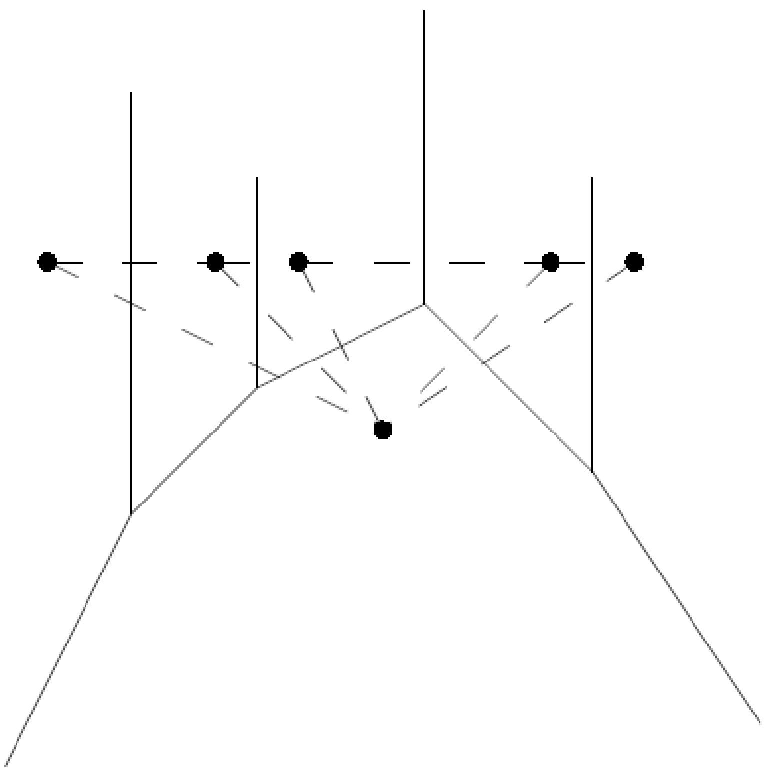
VD properties

- If all points in *P* are colinear, Vor(*P*) consists of *n – 1* parallel lines



VD properties

- If the points are not colinear, Vor(*P*) is continuous and its edges are line segments or half-segments

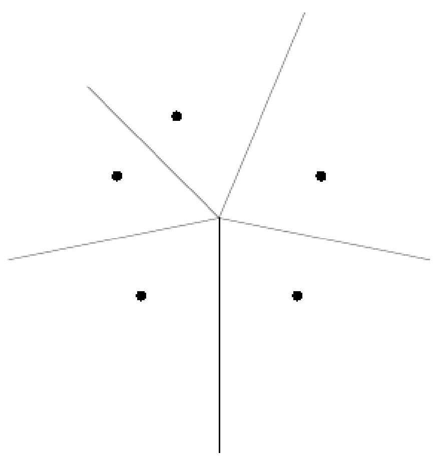


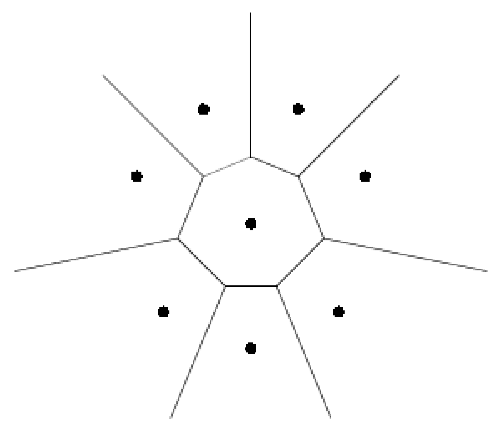
VD properties

- Voronoi cell y(P_i) is **unlimited** only when the point P_i belongs to an edge of the convex hull of P

VD properties

- If *P* contains 4 or more vertices lying on one circle, there is a Voronoi vertex formed by the intersection of Voronoi edges whose number corresponds to the number of points on that circle – we call it a **degenerated Voronoi diagram**





Algorithms for VD construction

- Generally, creating VD for *n* points lies in *O(n log n)*
- Algorithms:
  – Naïve approach
  – Incremental algorithm
  – Divide and conquer
  – Sweep line (Fortune's algorithm)
  …

Naïve approach

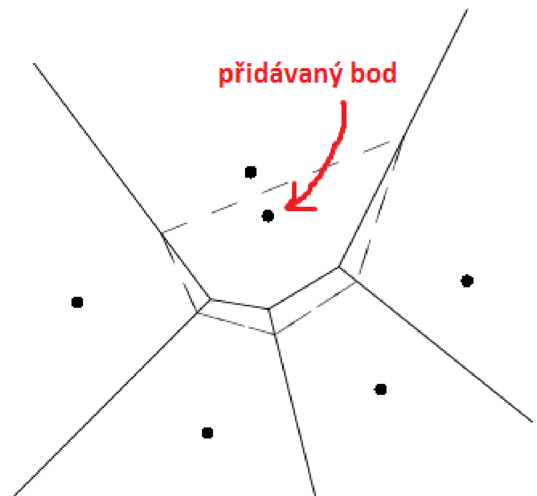
- Each region $y(P_i)$ of Voronoi diagram is generated as an intersection between halfplanes $h(P_i, P_j)$, for all $j \neq i$.
- The complexity of finding one region = $O(n \log n)$
- Total complexity = $\boldsymbol{O(n^2 \log n)}$

Incremental algorithm

1. For all points *P*:
   1. In the current VD, we localize the corresponding Voronoi cell containing P_{i+1} → y(P_{i1})
   2. We create the axis of line segment $P_{i+1}P_{i1}$
   3. We determine the intersections of this axis of line segment $P_{i+1}P_{i1}$ with the boundary of y(P_{i1})
   4. We select one of the intersections which determines the Voronoi cell with which our algorithm will continue in the next step → y(P_{i2})

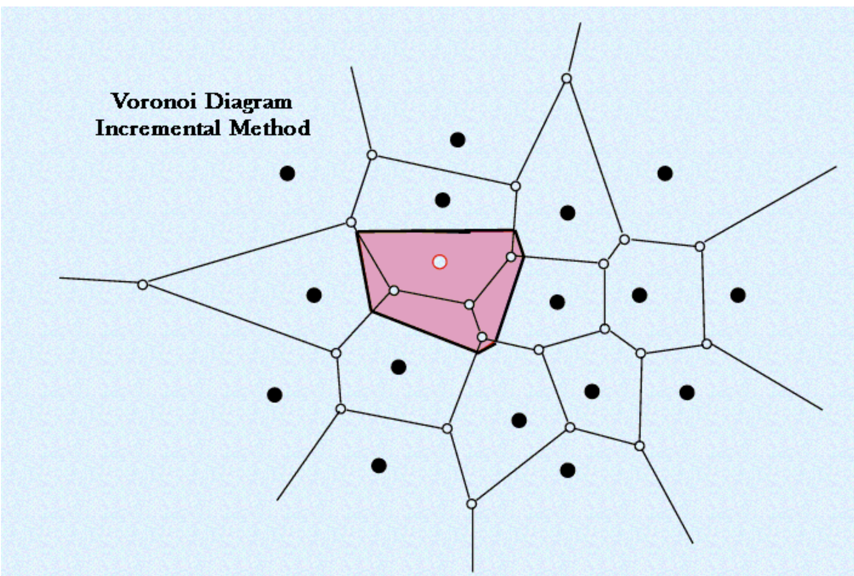
Incremental algorithm

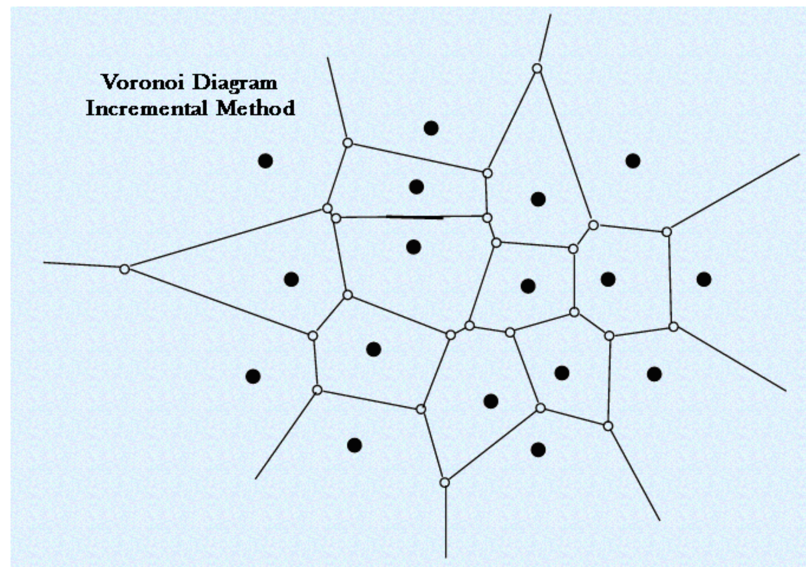
5. We create the axis of the line segment $P_{i+1}P_{i2}$ and its intersections with the boundary of y(P_{i2}). We select an intersection not lying on the common edge of y(P_{i1}) and y(P_{i2}) and we continue
6. We repeat step 5, until we reach the second intersection of the axis of line intersection $P_{i+1}P_{i1}$ with the boundary of y(P_{i1})
7. We remove the edges inside the newly created Voronoi cell



Incremental algorithm

- Complexity *O(n²)*, in special cases even *O(n)*





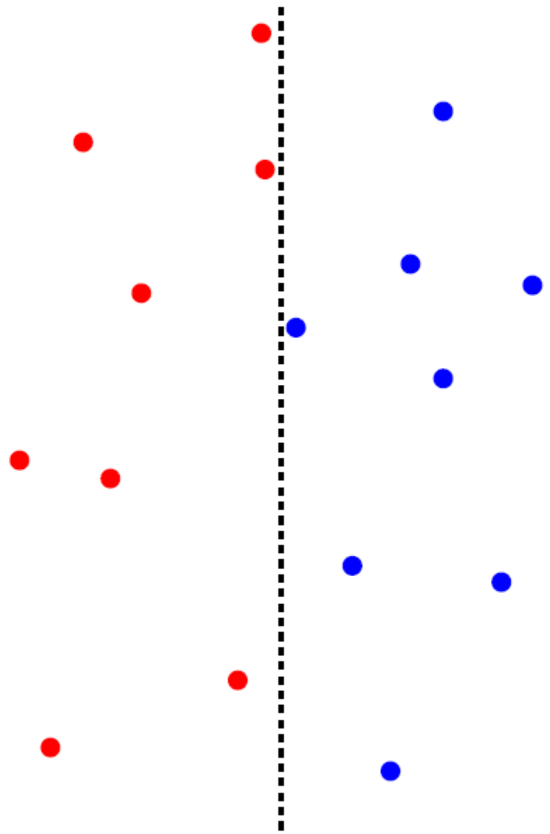
http://www.personal.kent.edu/~rmuhamma/Compgeometry/MyCG/Voronoi/Incremental2/incremental2.htm

Divide and conquer

- The input set is recursively divided to two subsets until we reach the set of three points for which we construct the VD easily
- The crucial part is the „backtracking step“, where the individual solutions have to be merged to one VD
- Complexity $O(n \log n)$

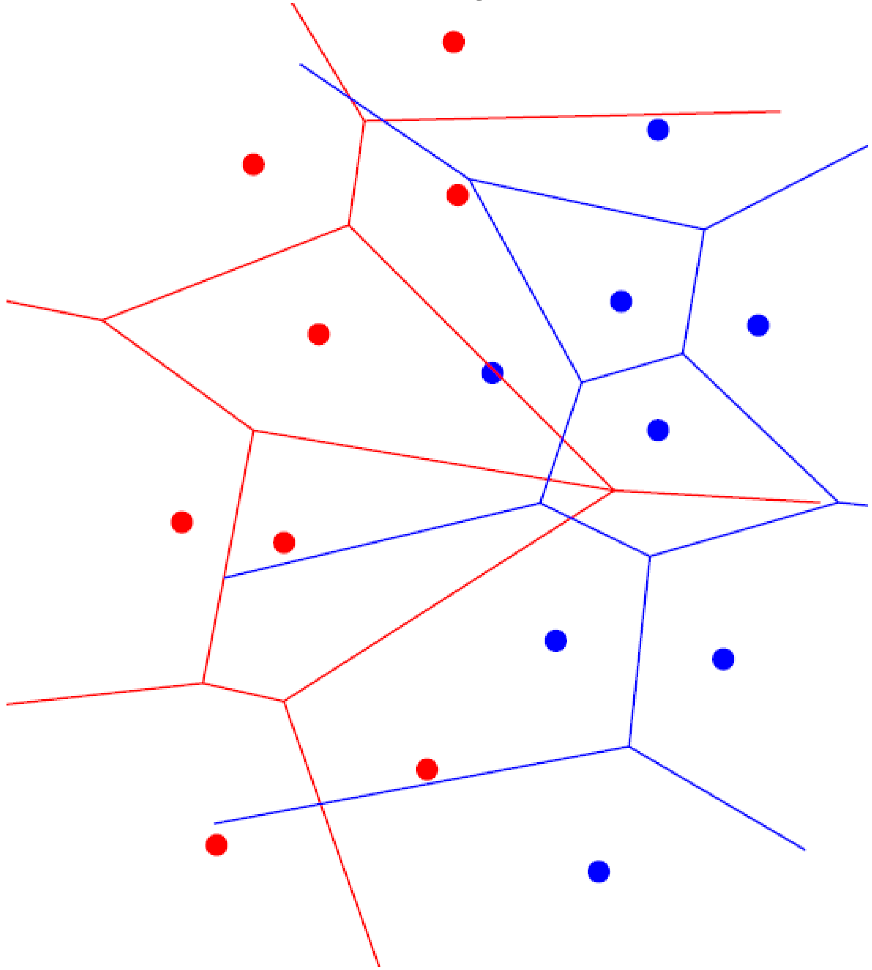
Divide and conquer

- We sort the input points and divide them vertically to two subsets *R* and *B* of approximately the same size



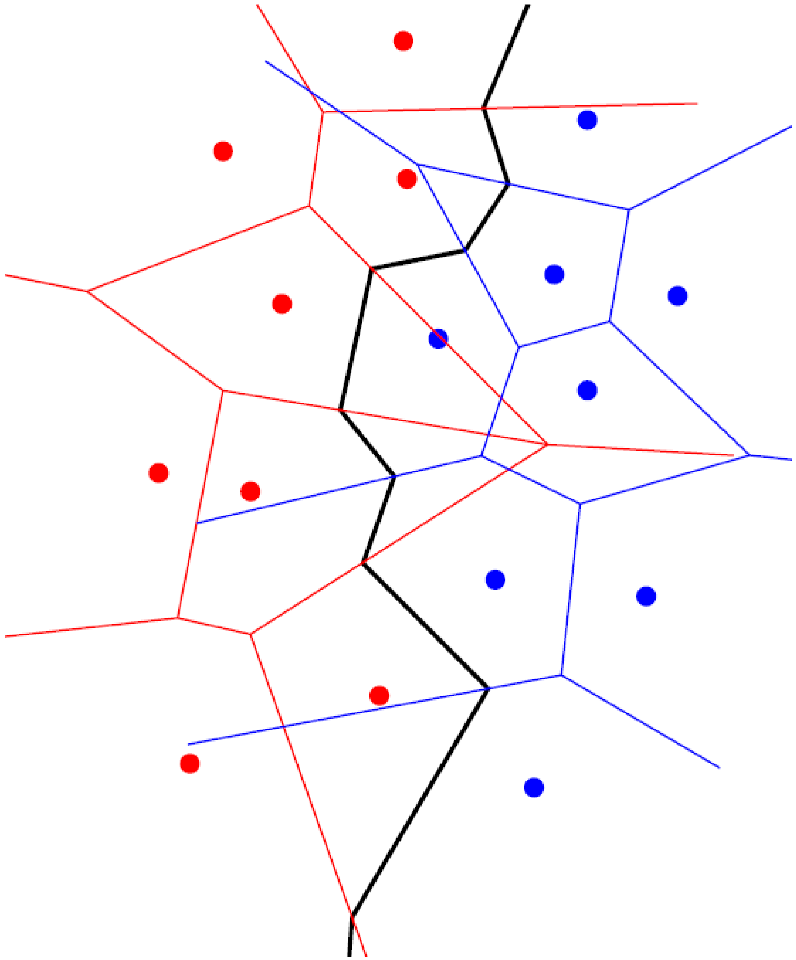
Divide and conquer

- We calculate recursively Vor(R) and Vor(B)



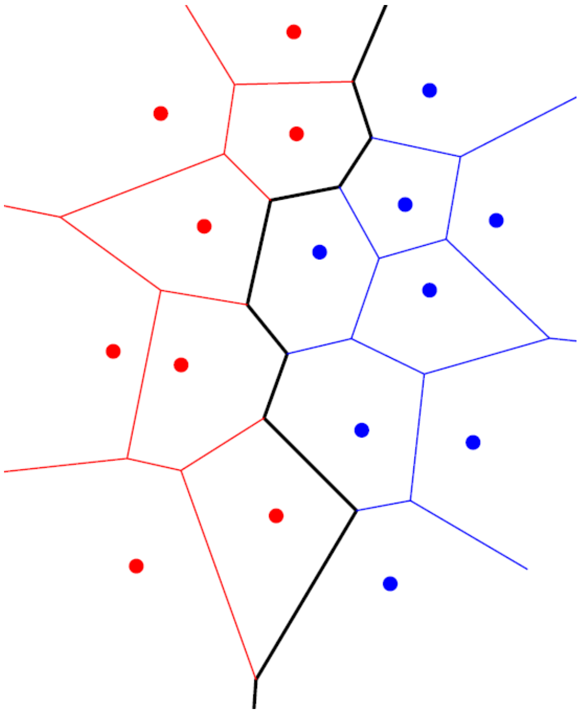
Divide and conquer

- We determine so called **separating chain**



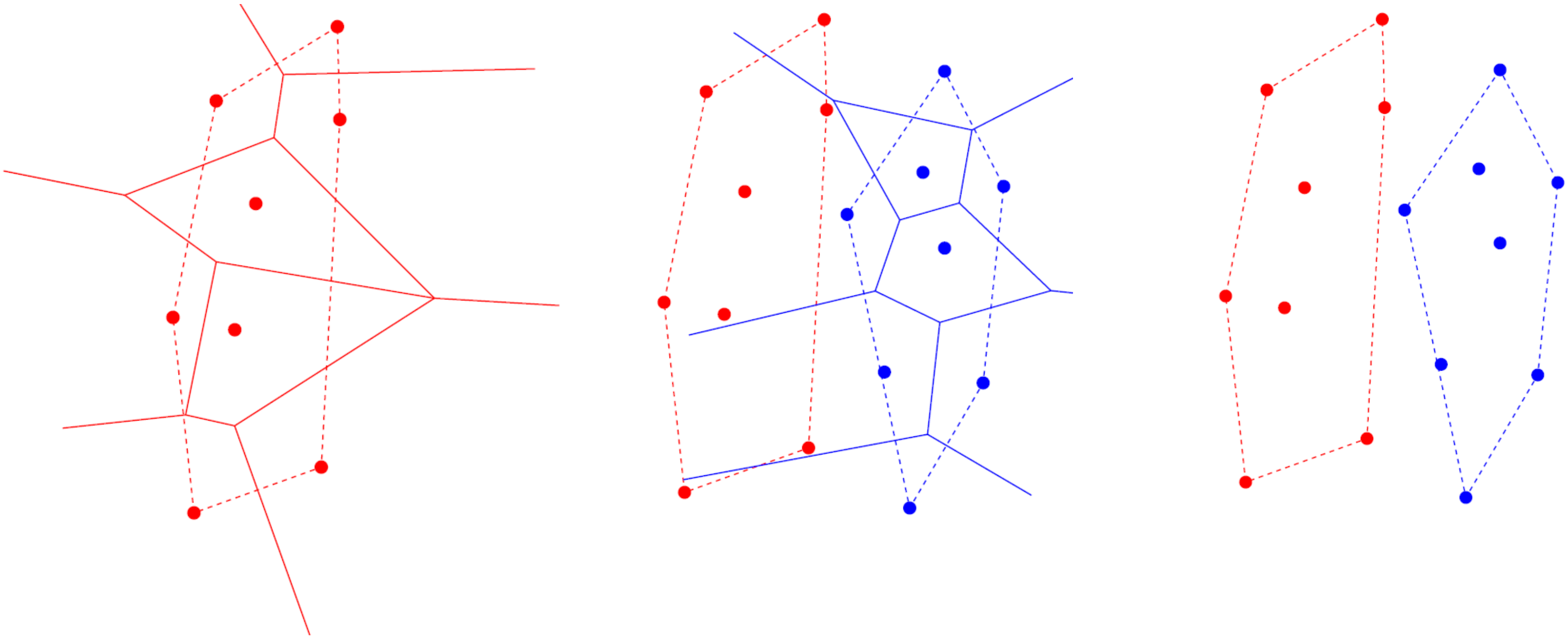
Divide and conquer

- We remove the part of Vor(*R*) lying on the right side from the separating chain and the par of Vor(*B*) lying on the left side from the separating chain



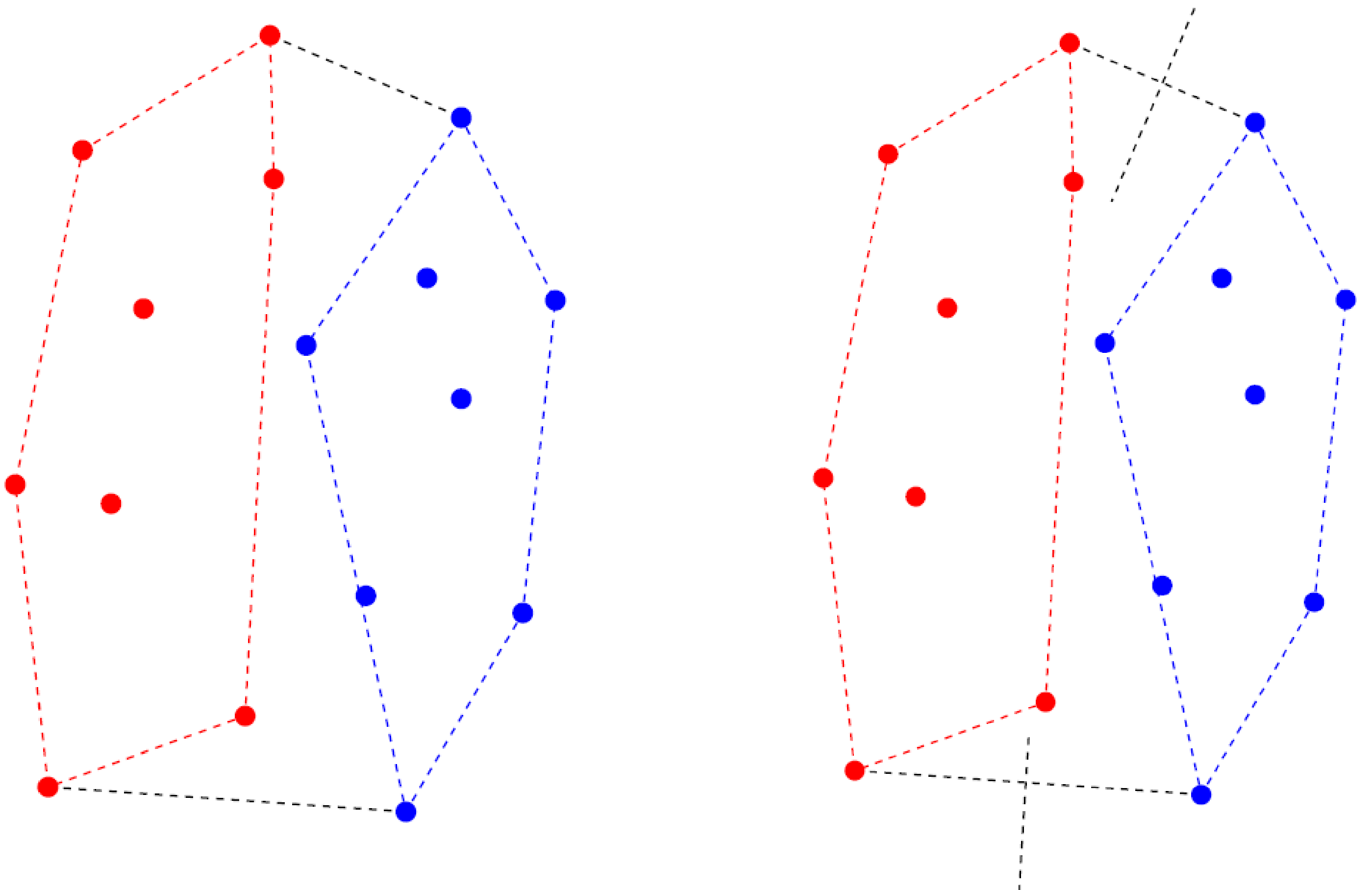
Divide and conquer

- Defining the separating chain:
  - First, we find two convex hulls …



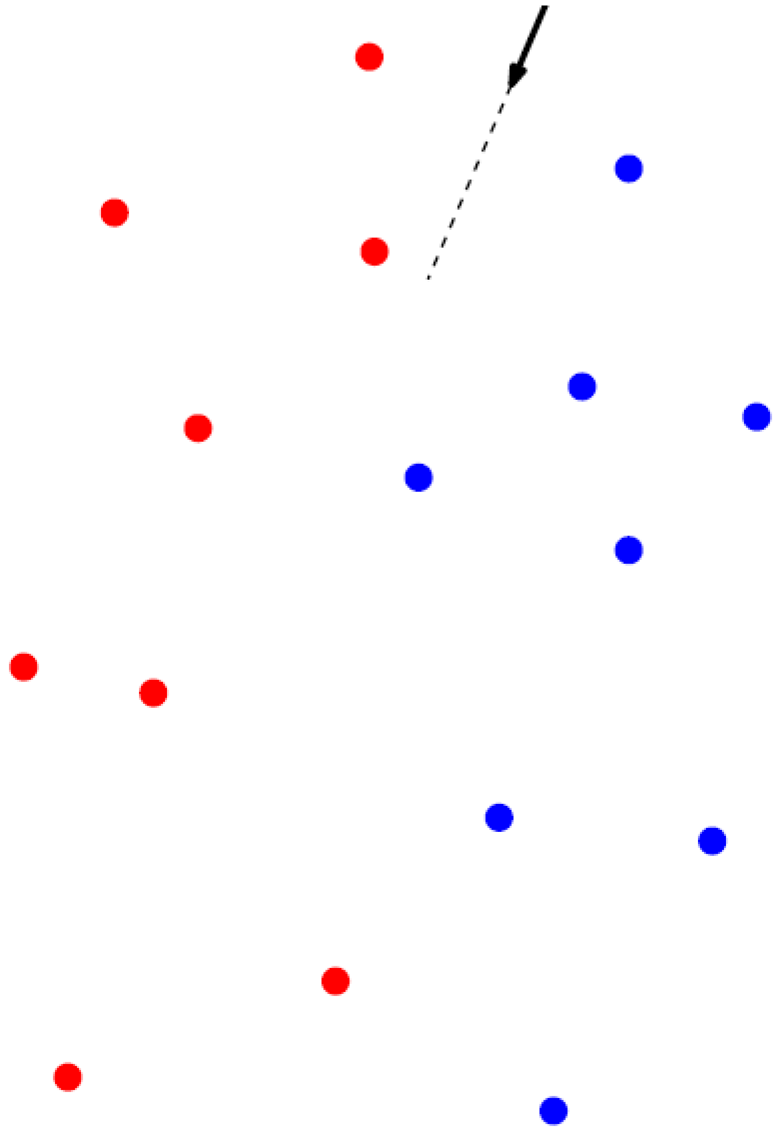
Divide and conquer

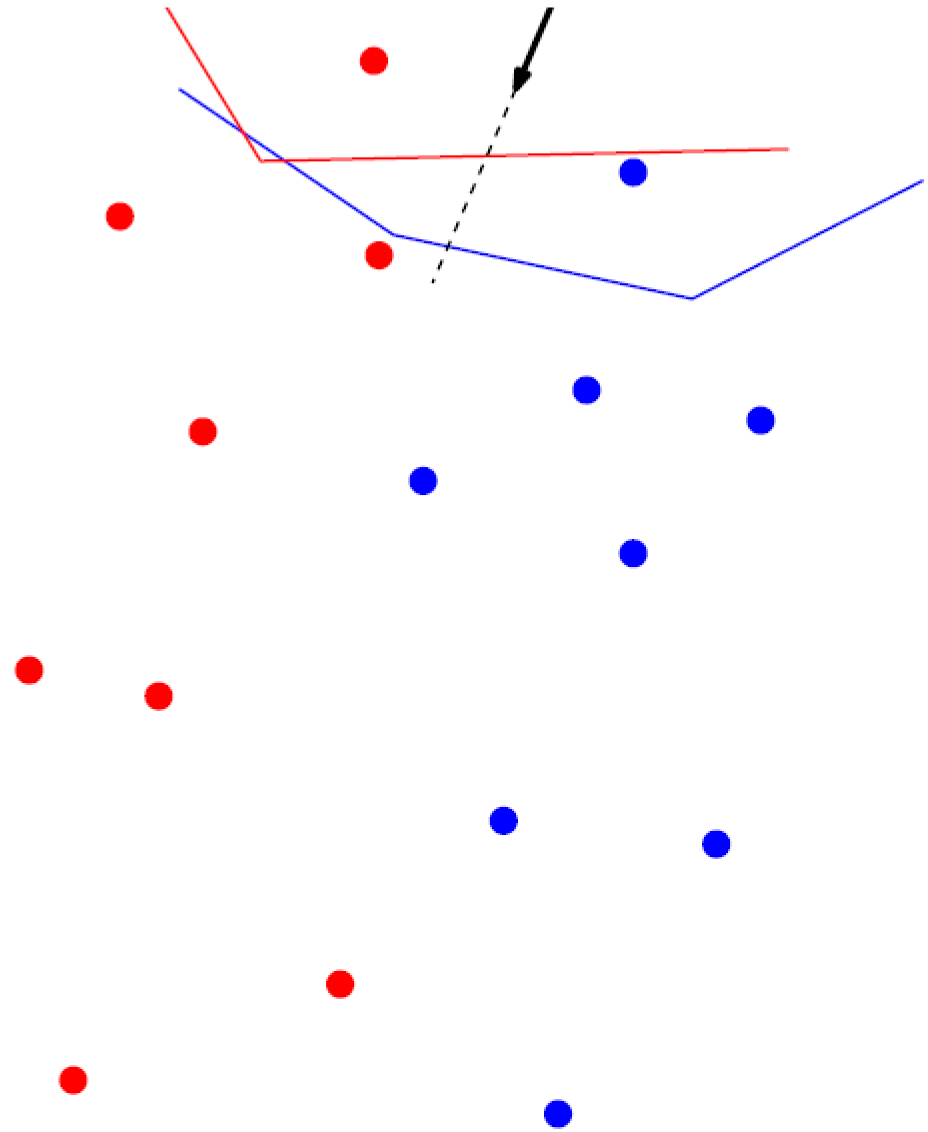
… and the upper and lower tangent and their perpendicular half-lines

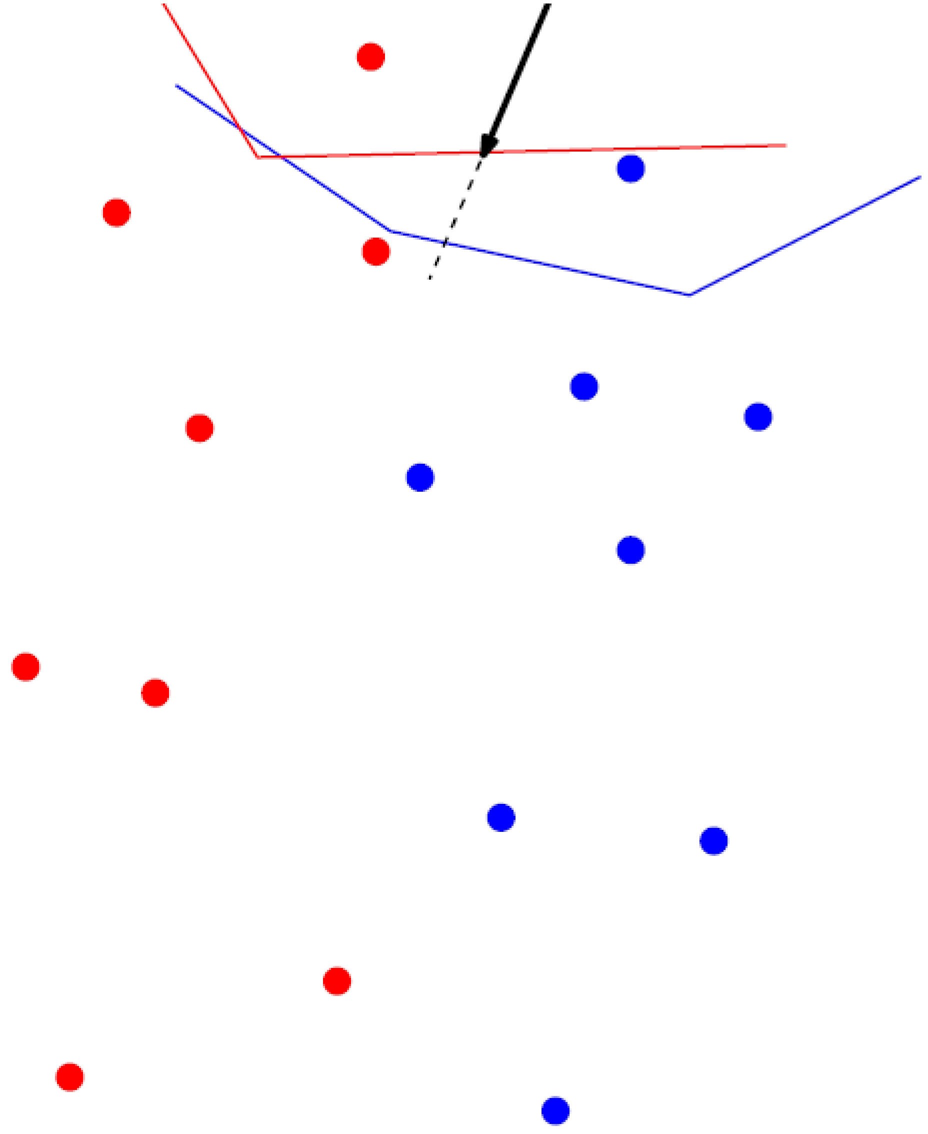


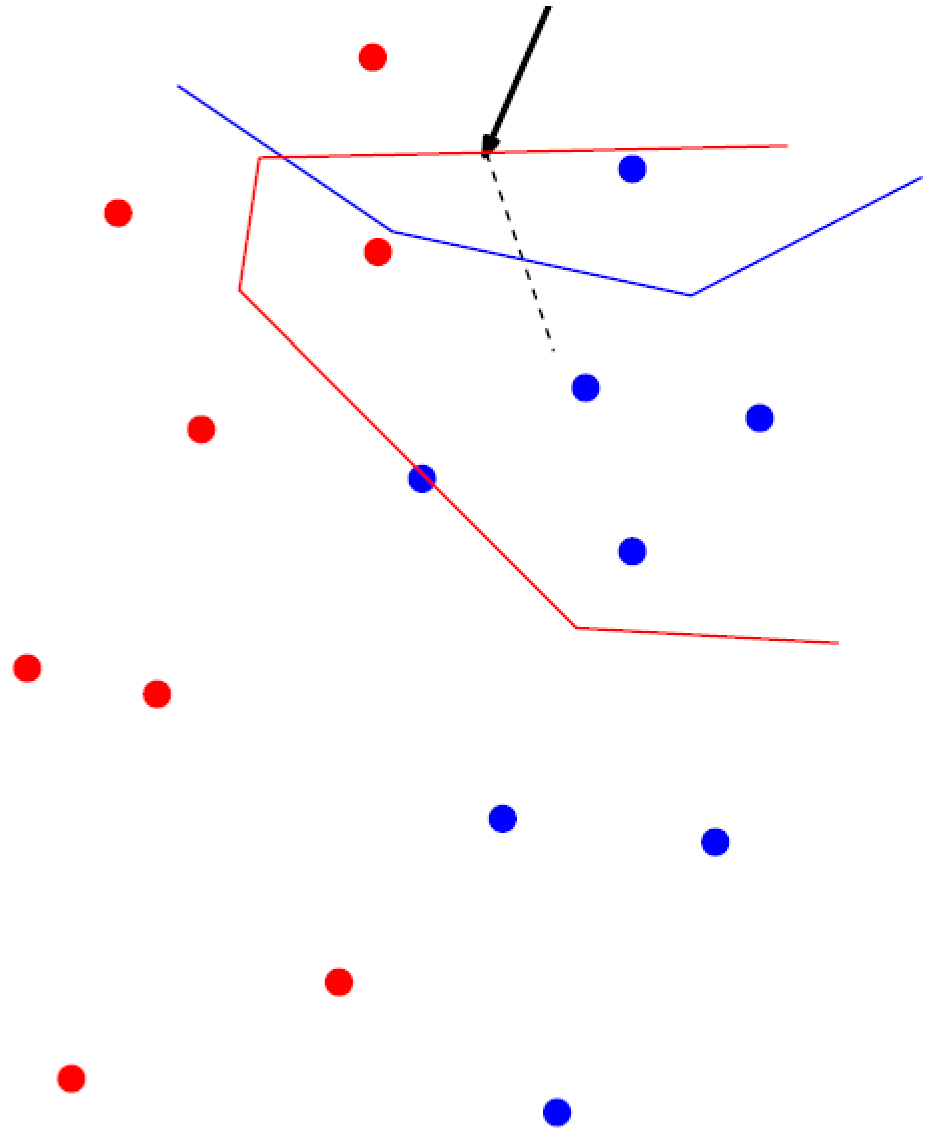
Divide and conquer

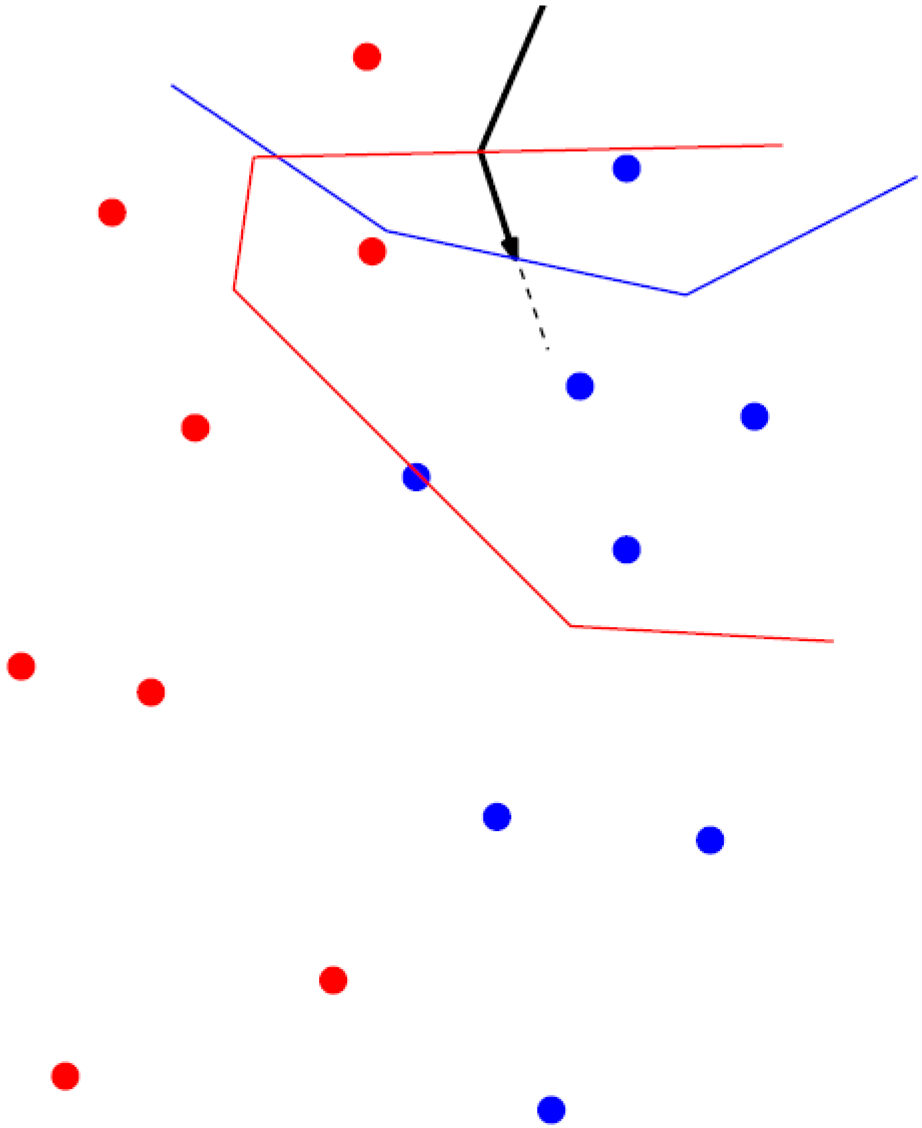
- We start from one of these half-lines and continue with the following procedure, until we reach the second half-line:
  - Always when there starts an edge $e \in b(R, B)$ for which $e \subset b_{ij}$, $p_i \in R$, $p_j \in B$:
    - Search for the intersection of edge *e* with $\mathrm{Vor}_R(p_i)$
    - Search for the intersection of edge *e* with $\mathrm{Vor}_B(p_j)$
    - Select one of these intersections
    - Determine p_k corresponding to a new starting region
    - Replace p_i or p_j (according to the selected point) by new p_k
    - Repeat this step with the new edge

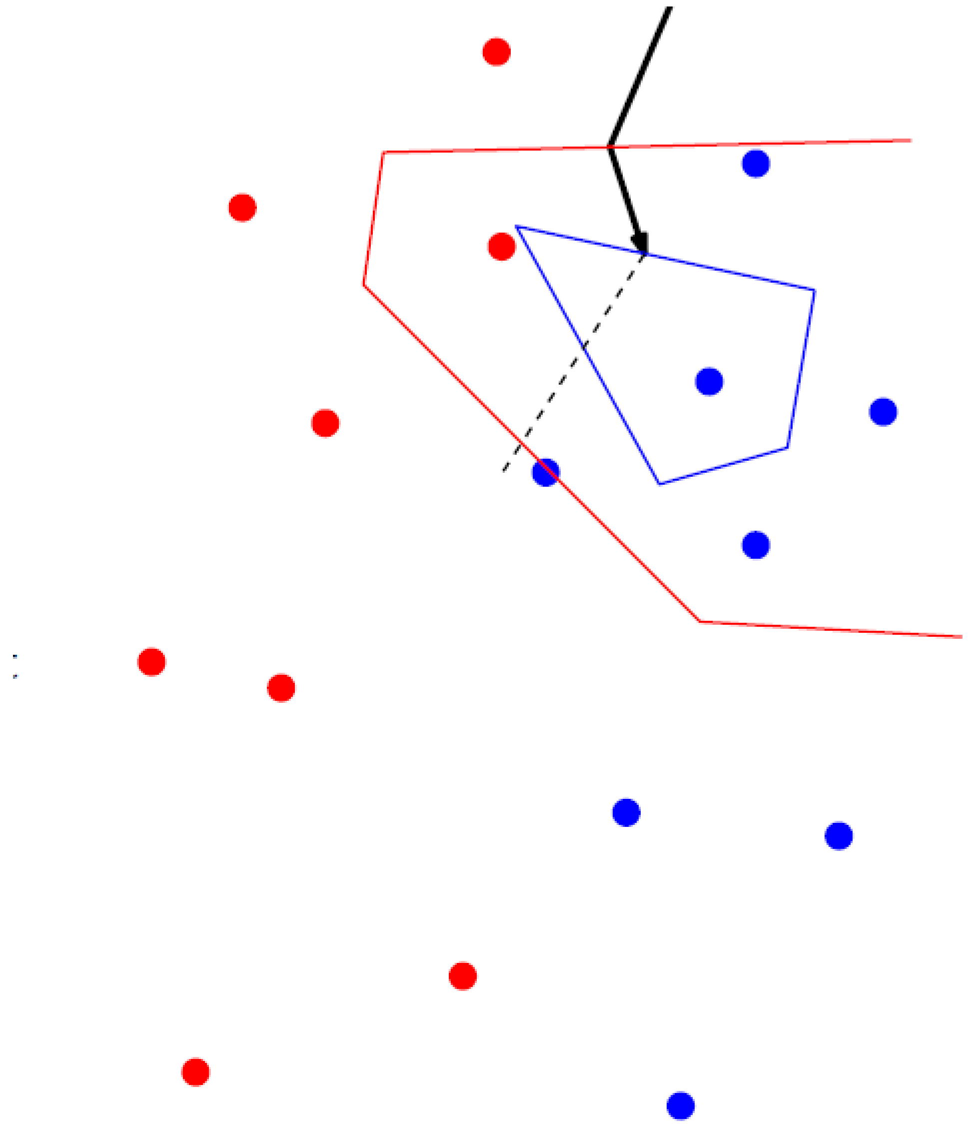


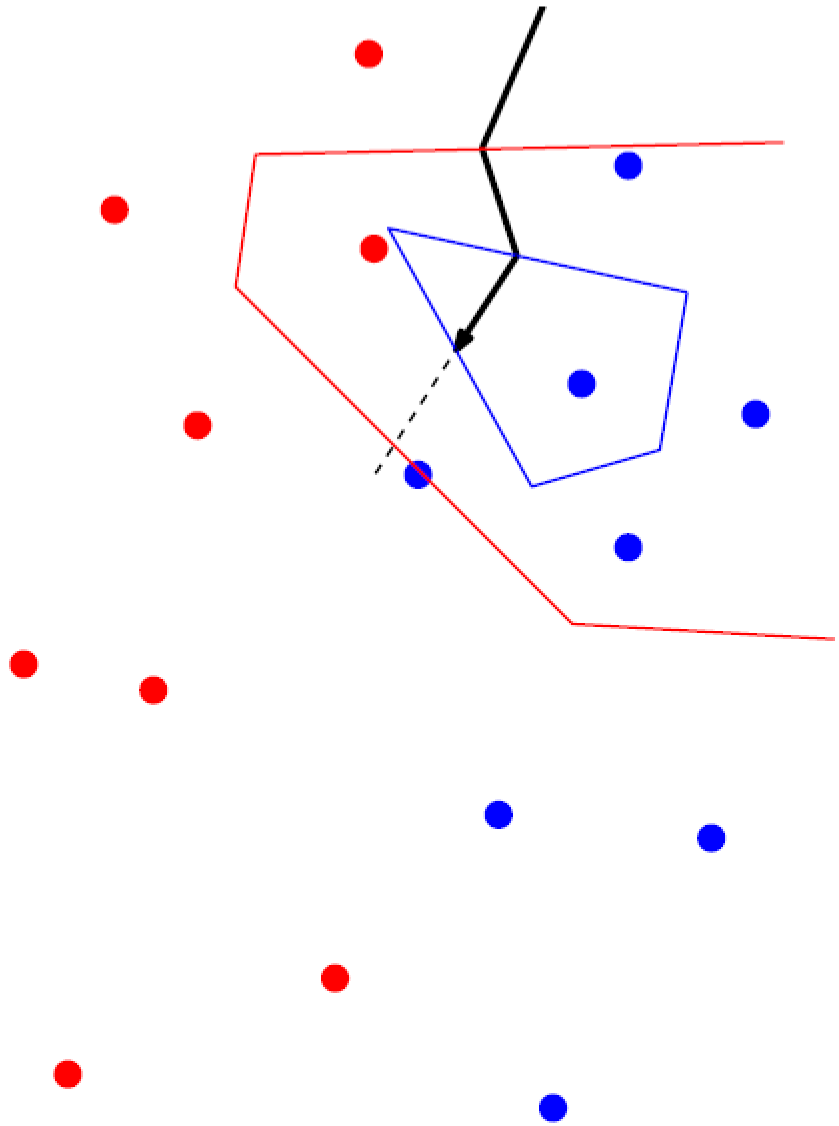


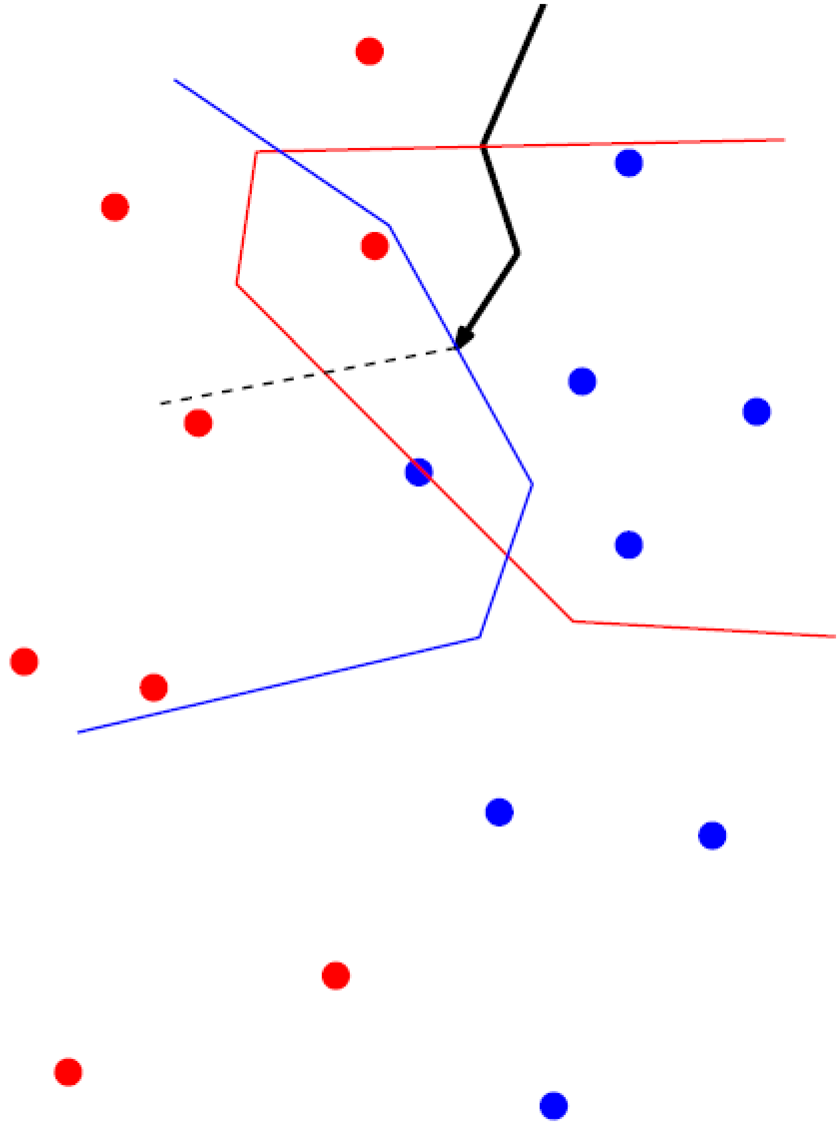


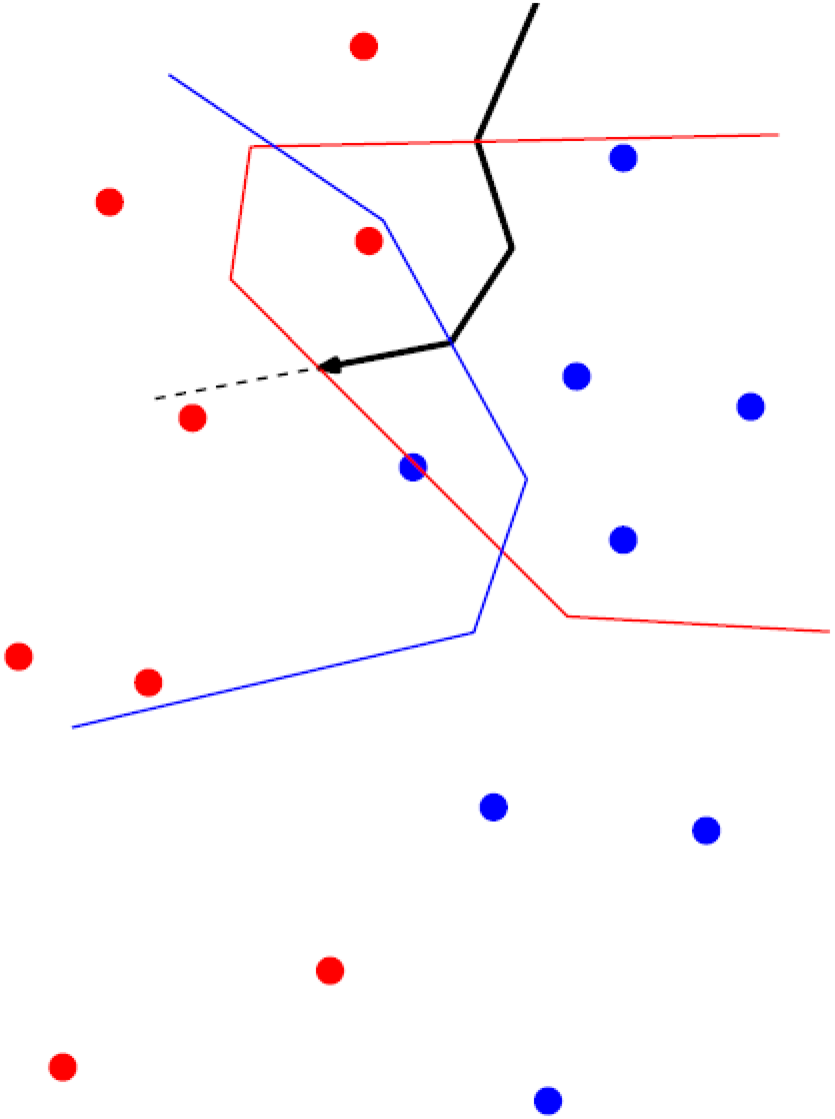


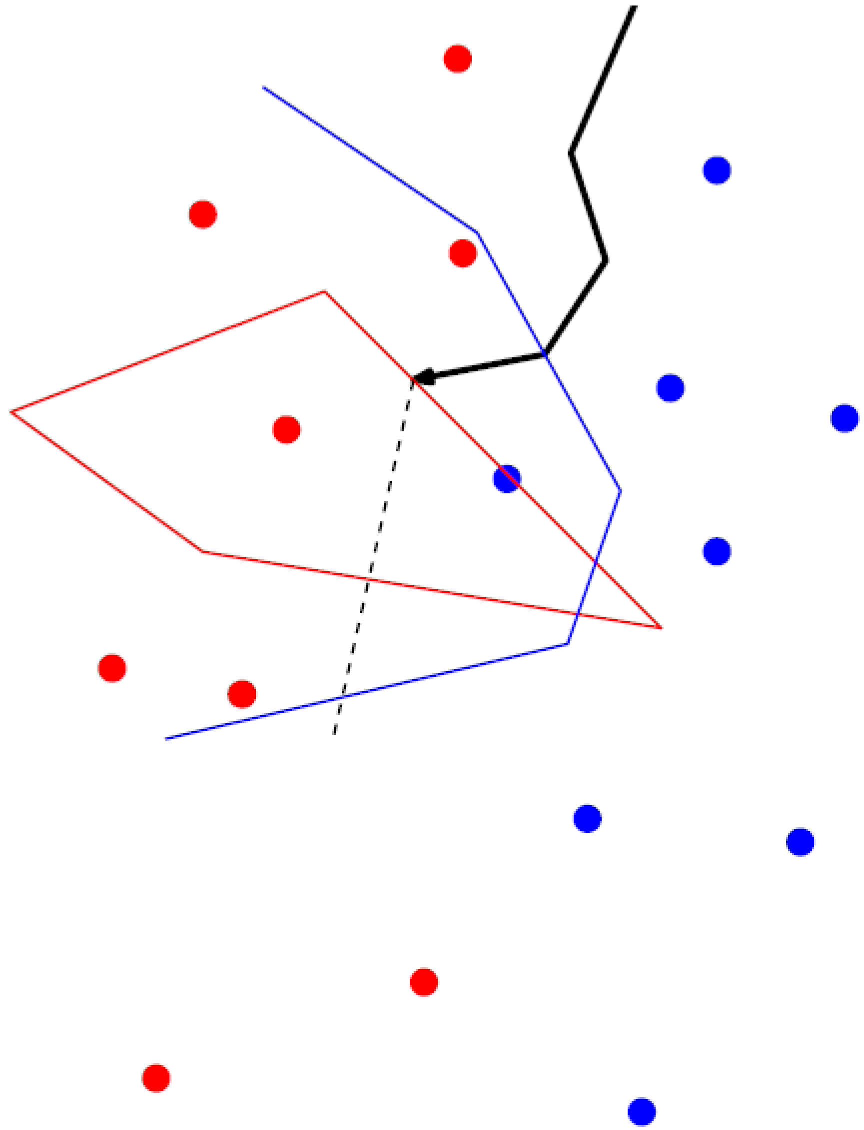


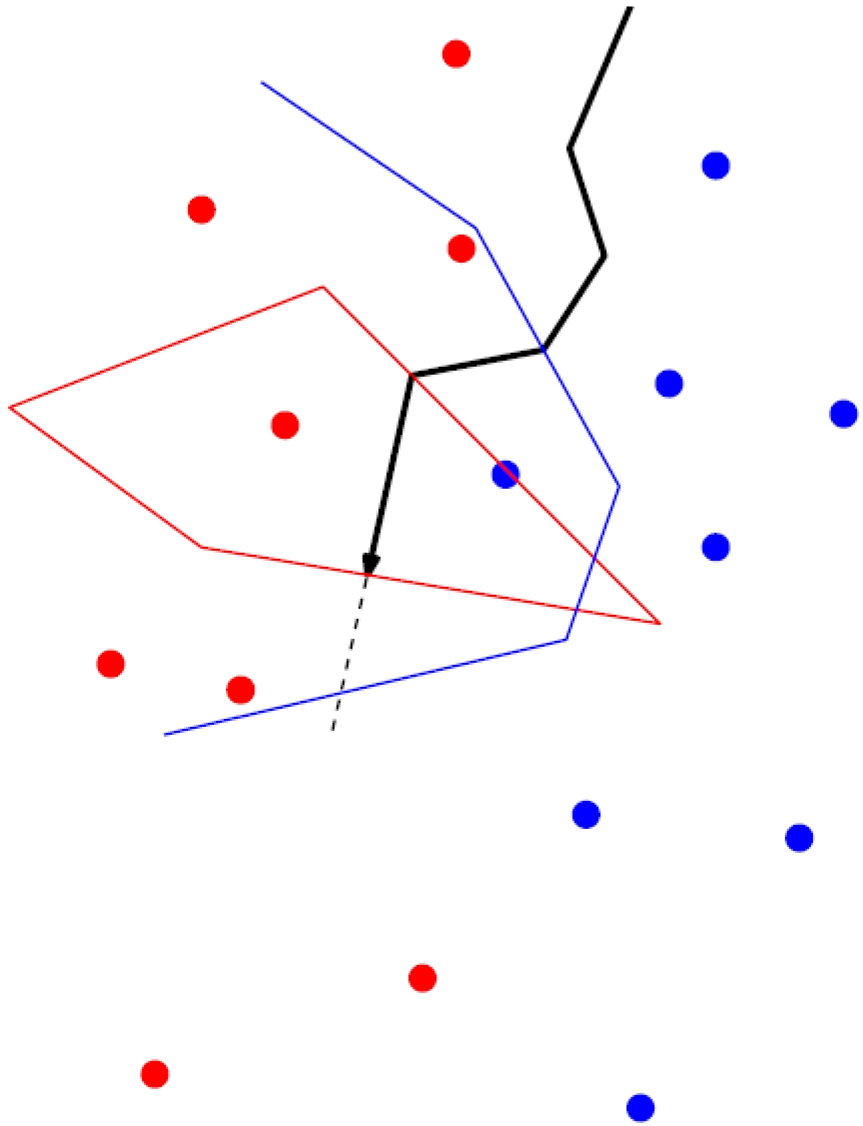


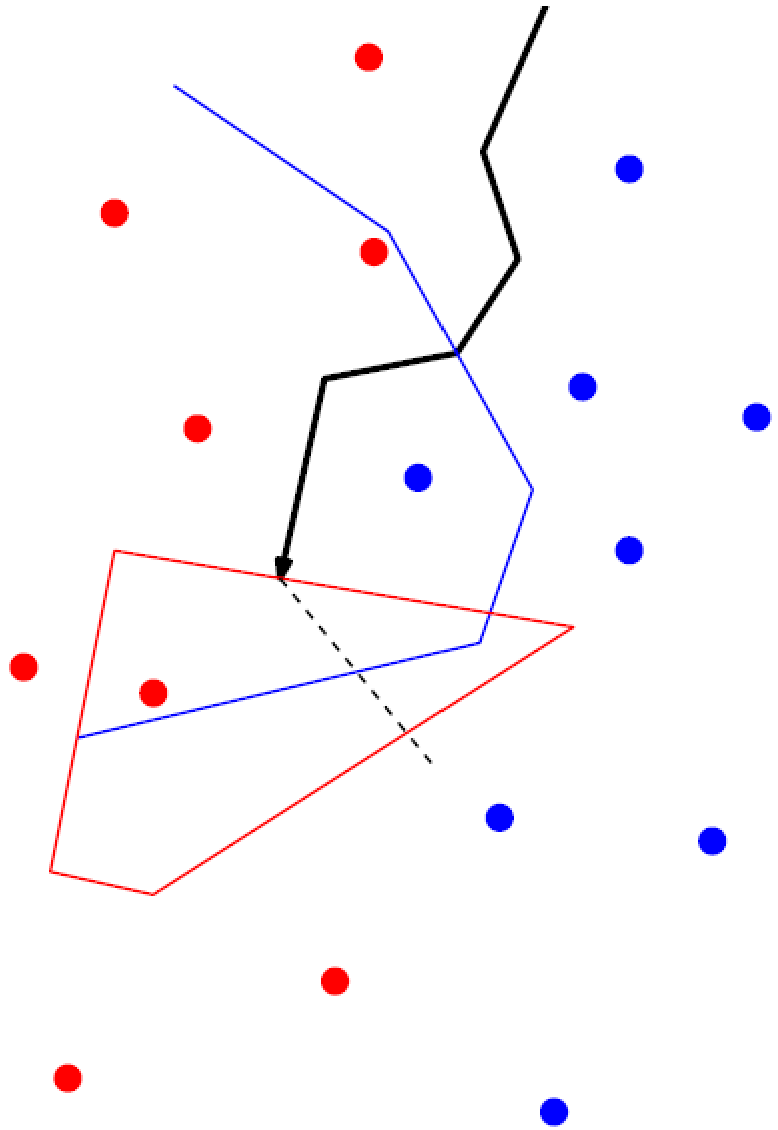


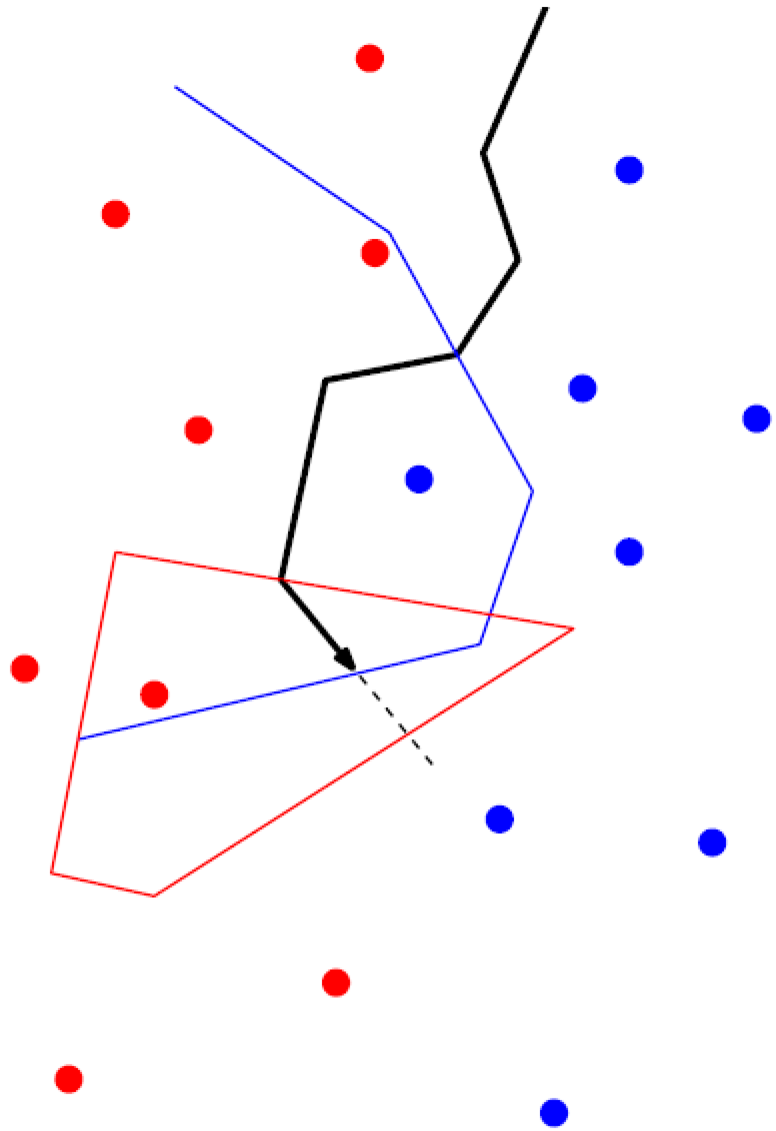


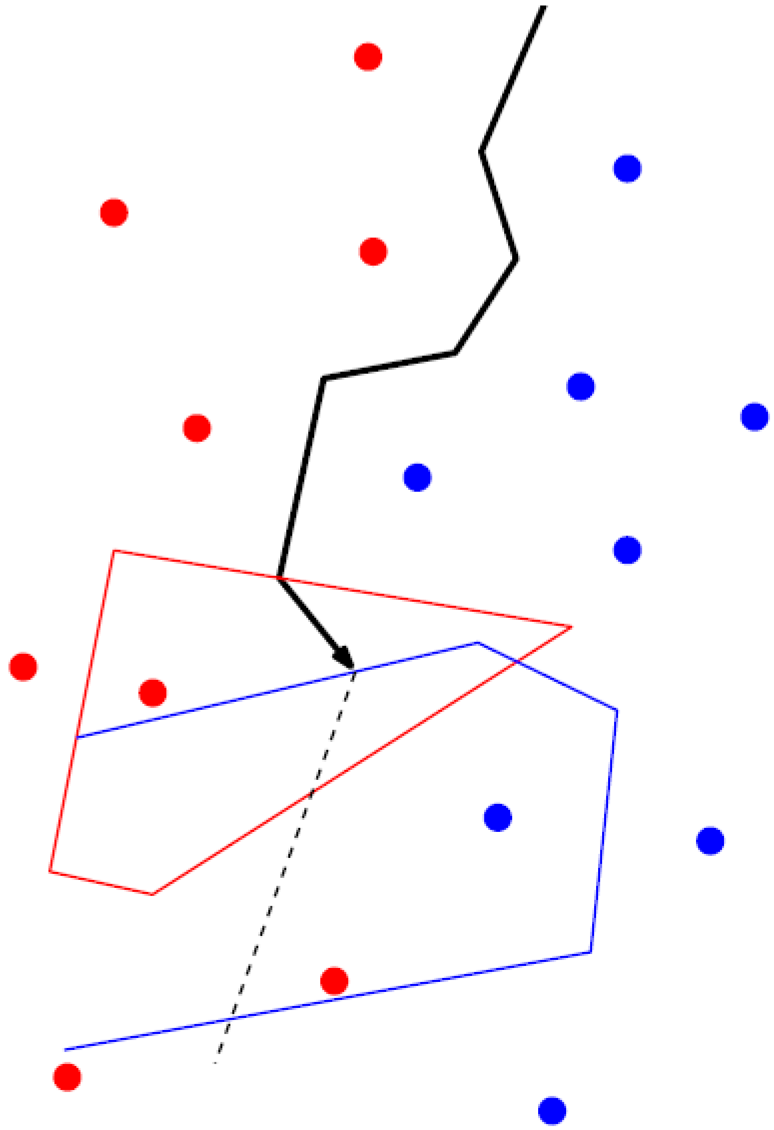


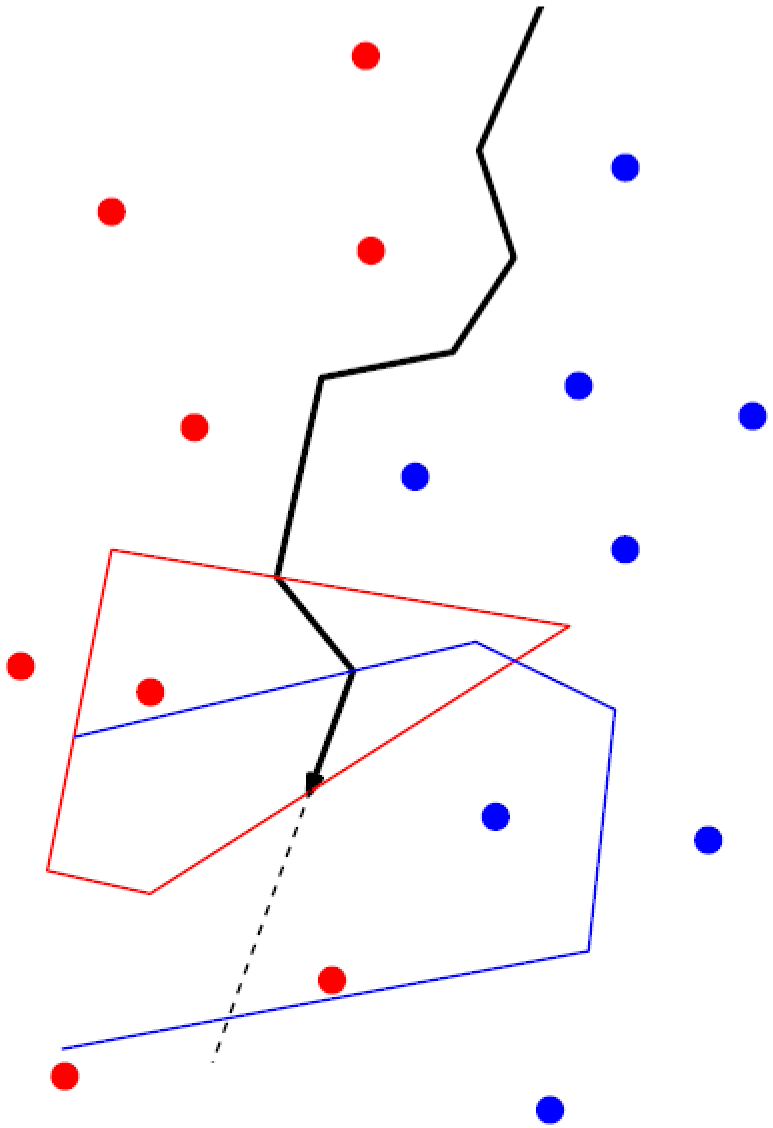


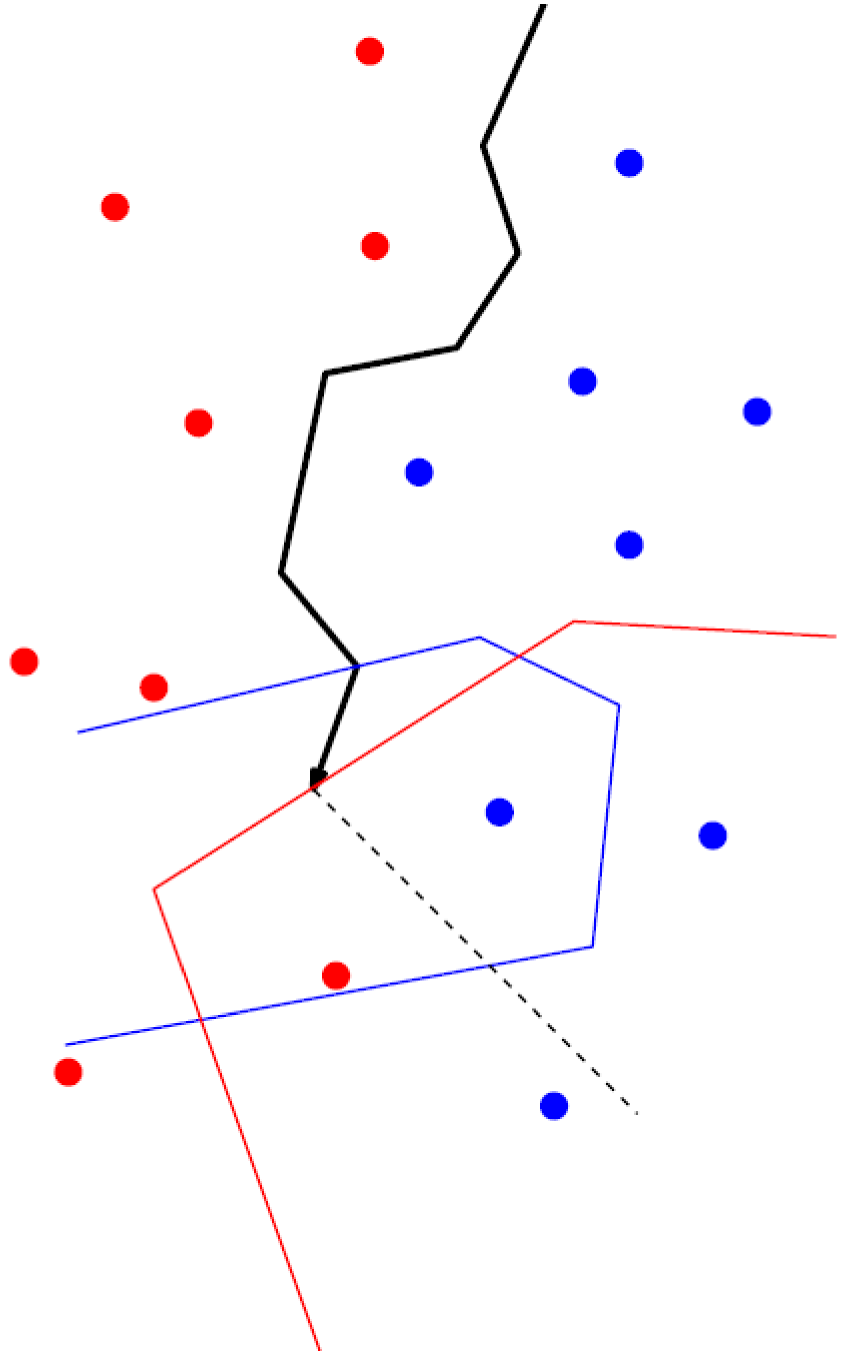


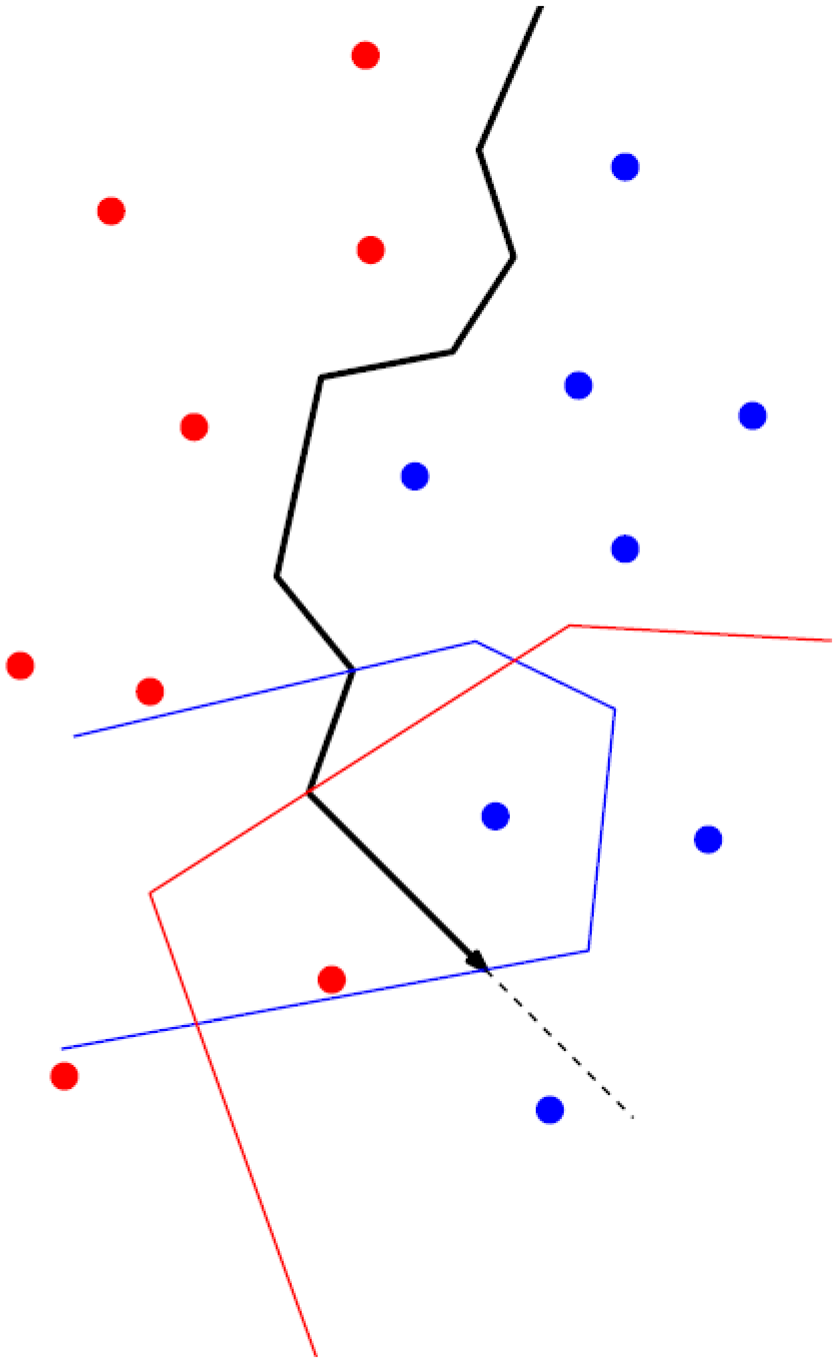


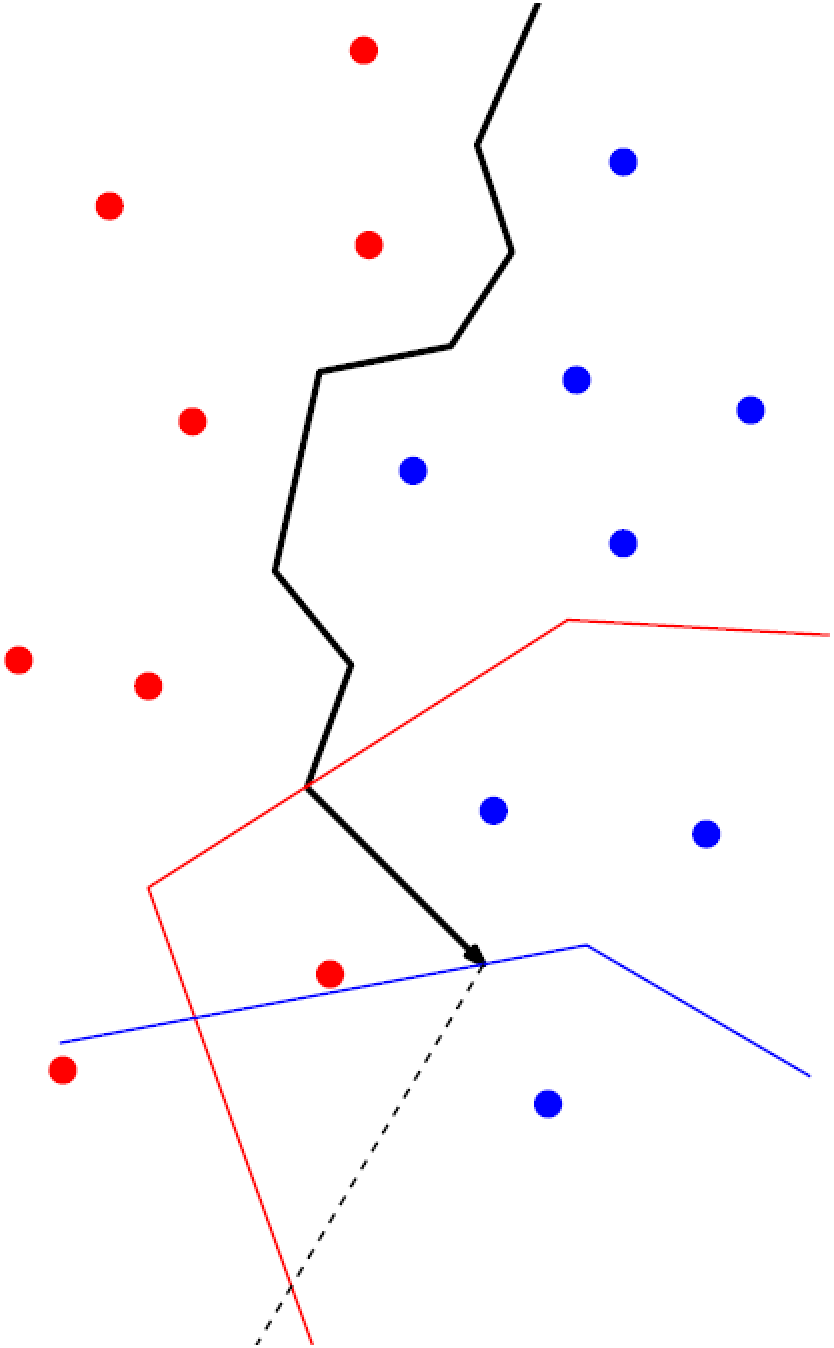


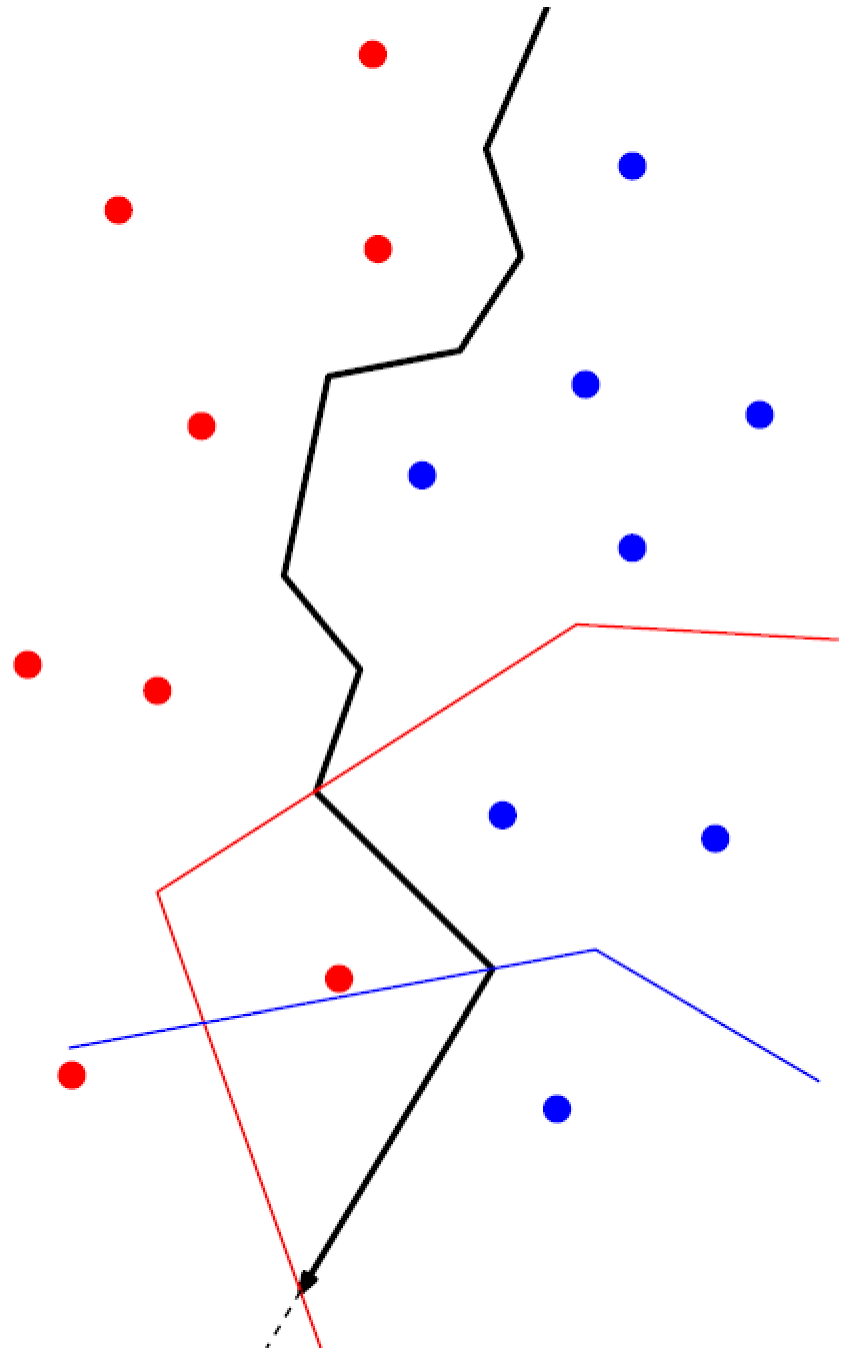


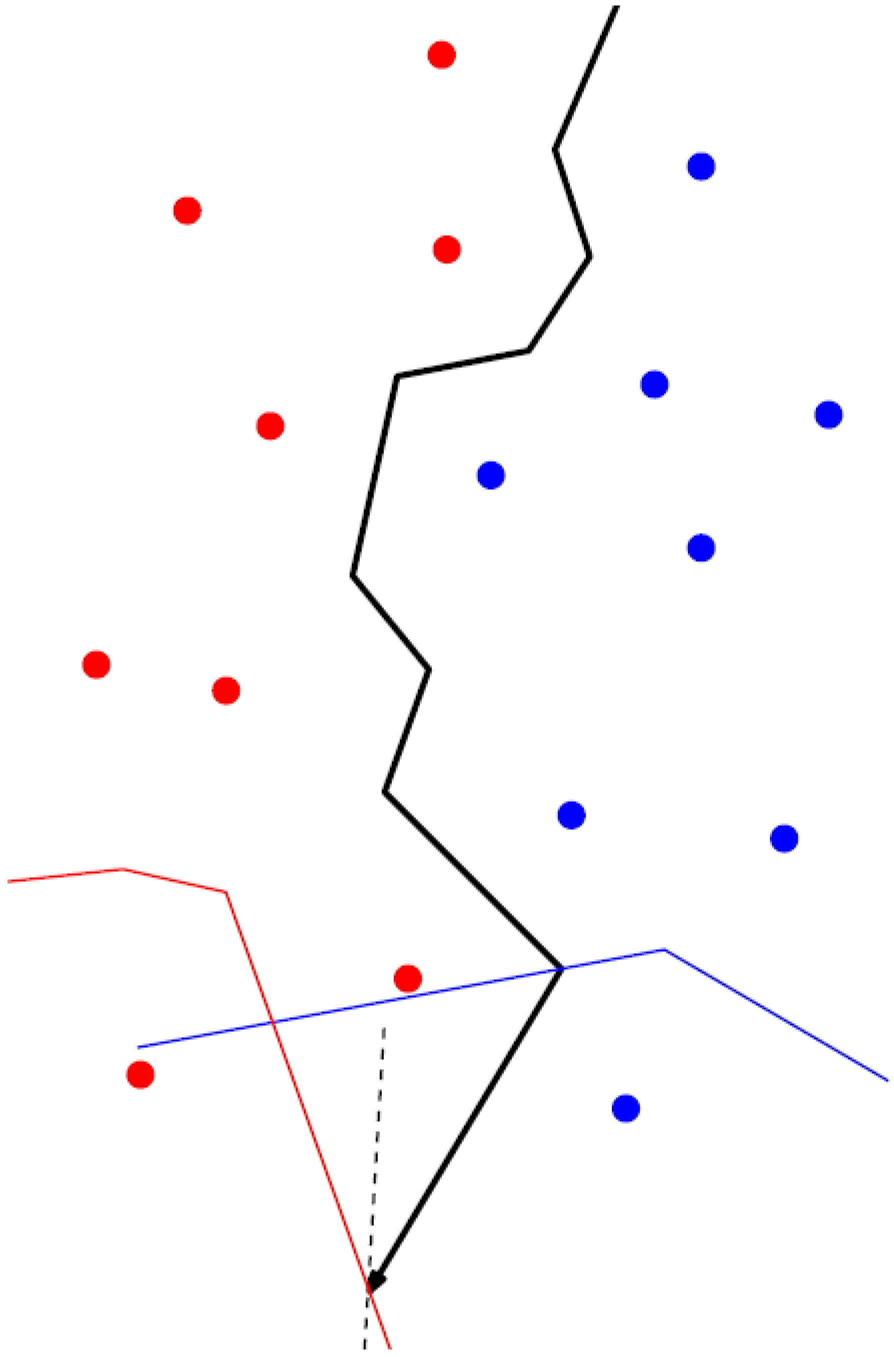


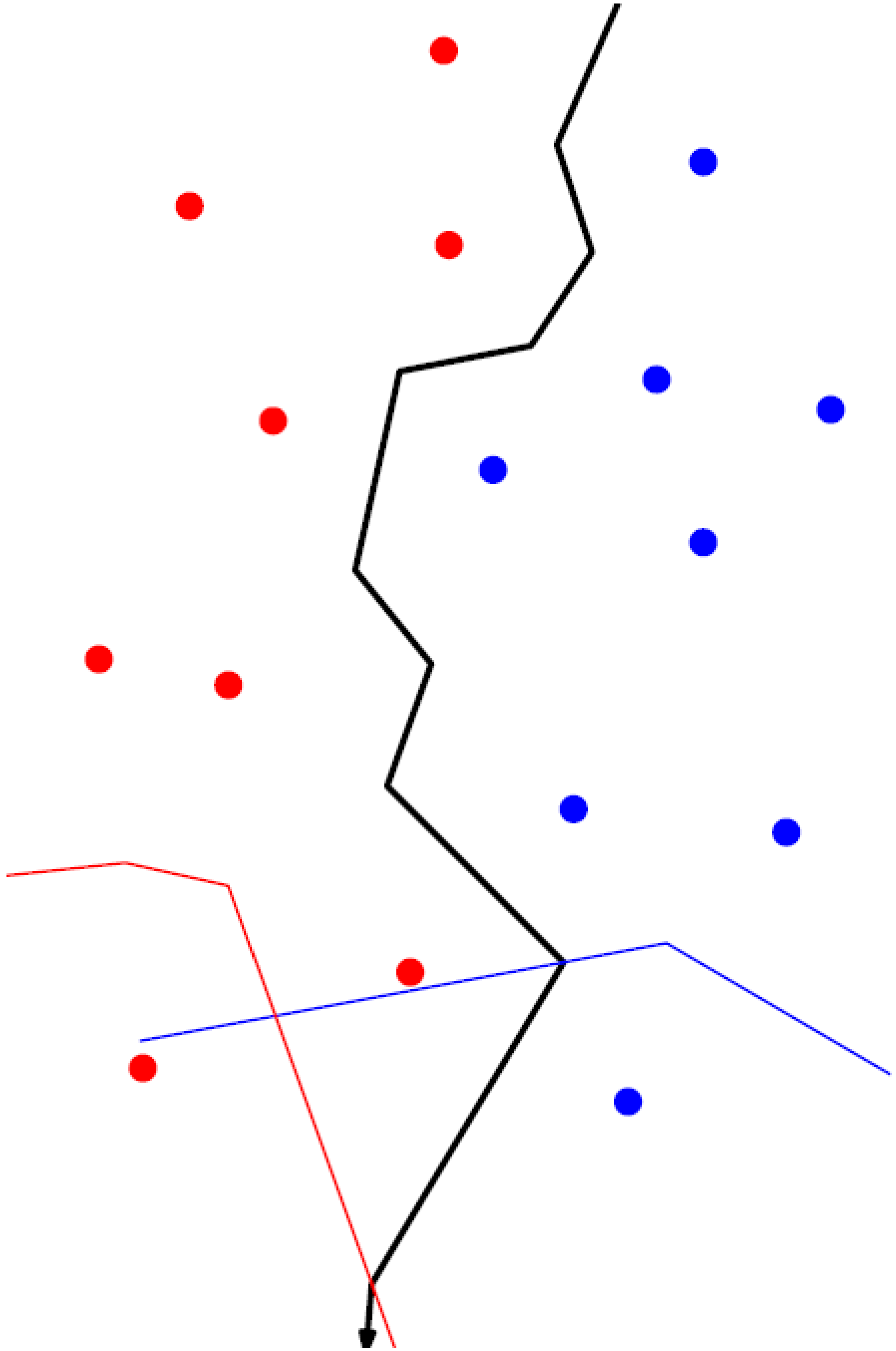


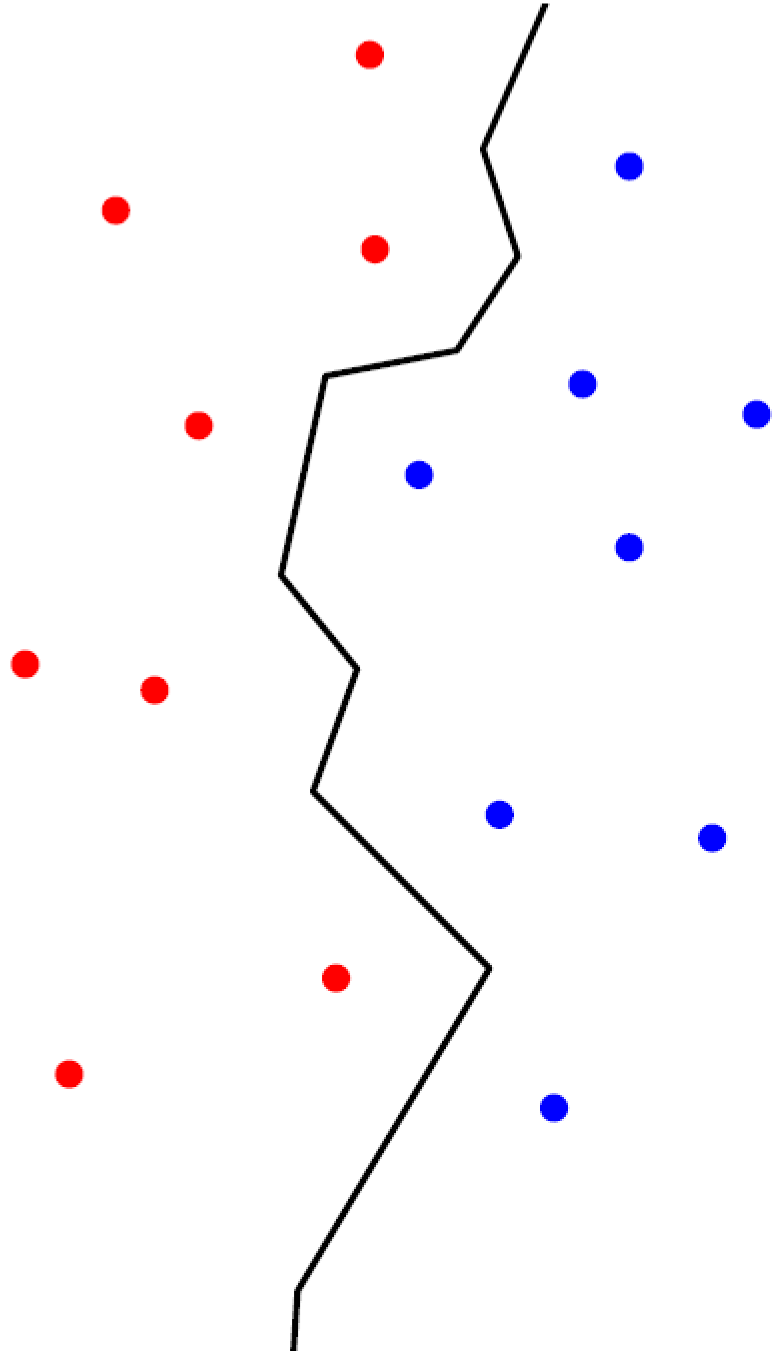












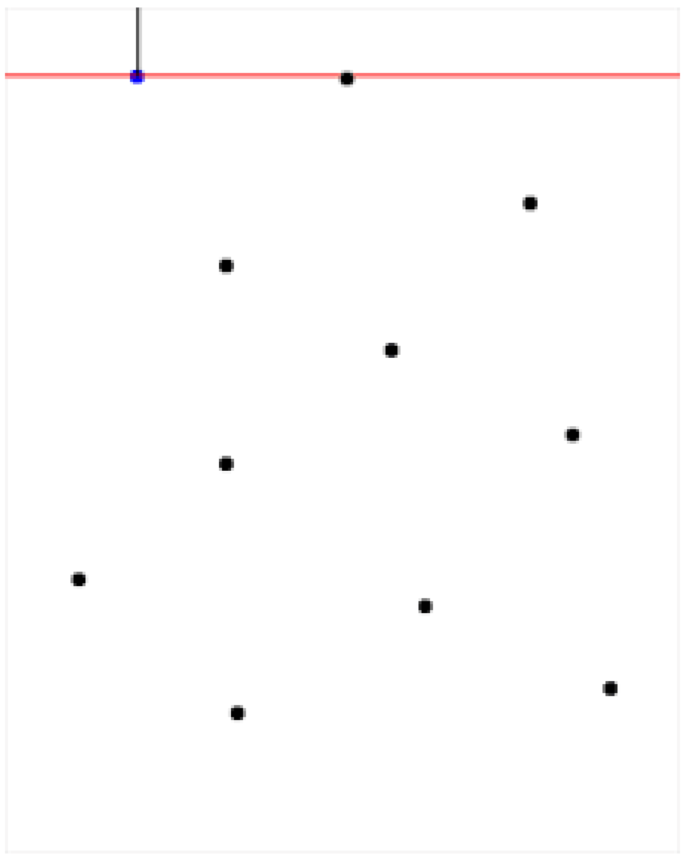
Sweep line (Fortune's algorithm)

- Algorithm uses so-called „sweep line“ and „beach line“, both of them traversing the space containing the input points
- The sweep line can be horizontal or vertical, heading from top to bottom or vice versa
- Invariant of the algorithm = for the input points already traversed by the sweep line we have already a correct VD constructed, the rest of the points was not processed yet

Sweep line (Fortune's algorithm)

- „Beach line“ is not in fact a line but a curve above the sweep line, consisting of parts of parabolas
- A set of all points being closer to some of the points above the sweep line than to the sweep line itself is delineated by parabolic arcs – their connection forms the beach line

Sweep line (Fortune's algorithm)



Sweep line (Fortune’s algorithm)

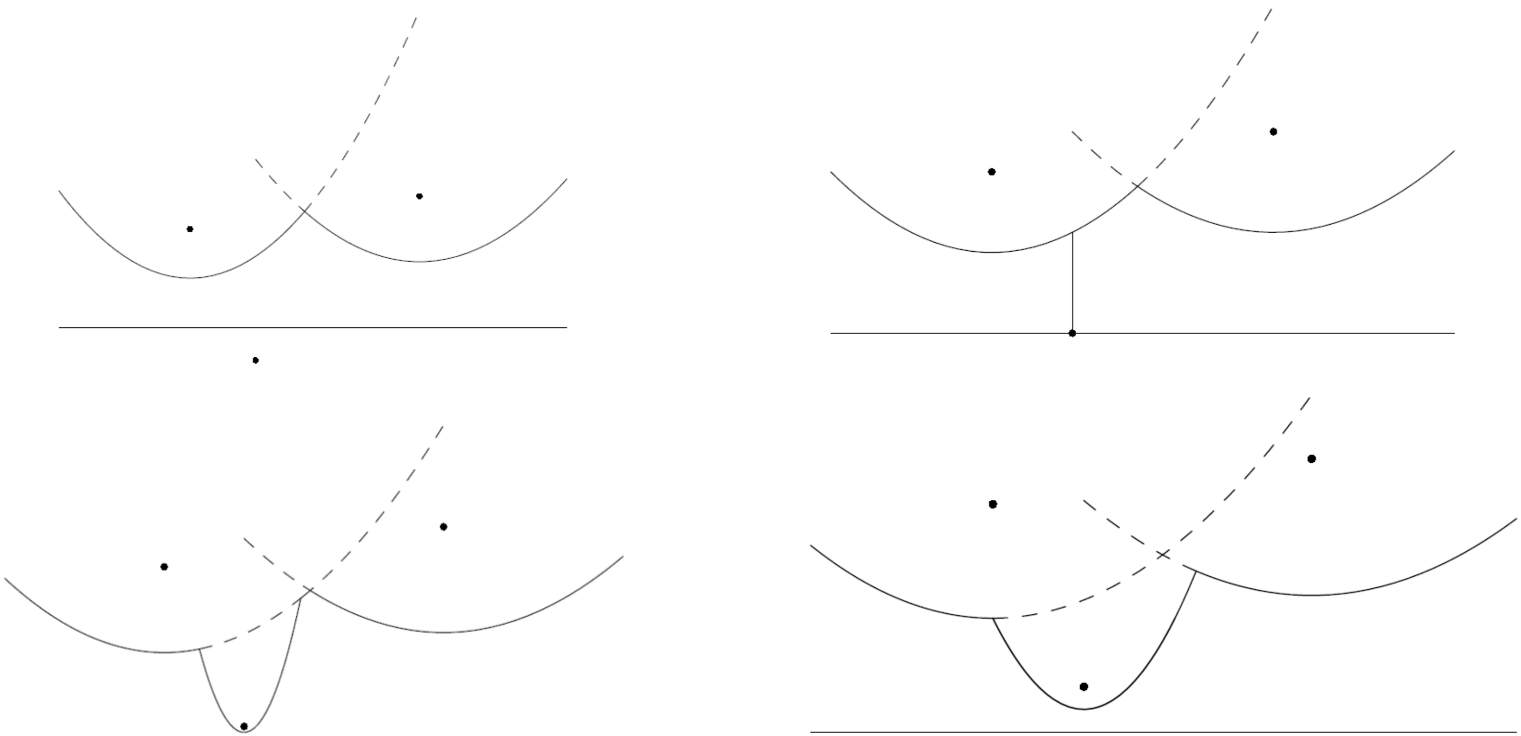
- The intersection of arcs lying on the beach line lie on the edges of the VD. With moving the sweep line, these intersections create the edges of VD Vor(P)
- The algorithm contains the following two operations:

Sweep line (Fortune’s algorithm)

- **Site event** – a new generating point emerges on the beach line, we have to add it to the VD structure
- **Circle event** – when one of the parabolic arcs is terminated

Site event

- This event generates a new parabolic arc on the beach line and its intersection with the current beach line starts to create a new VD edge



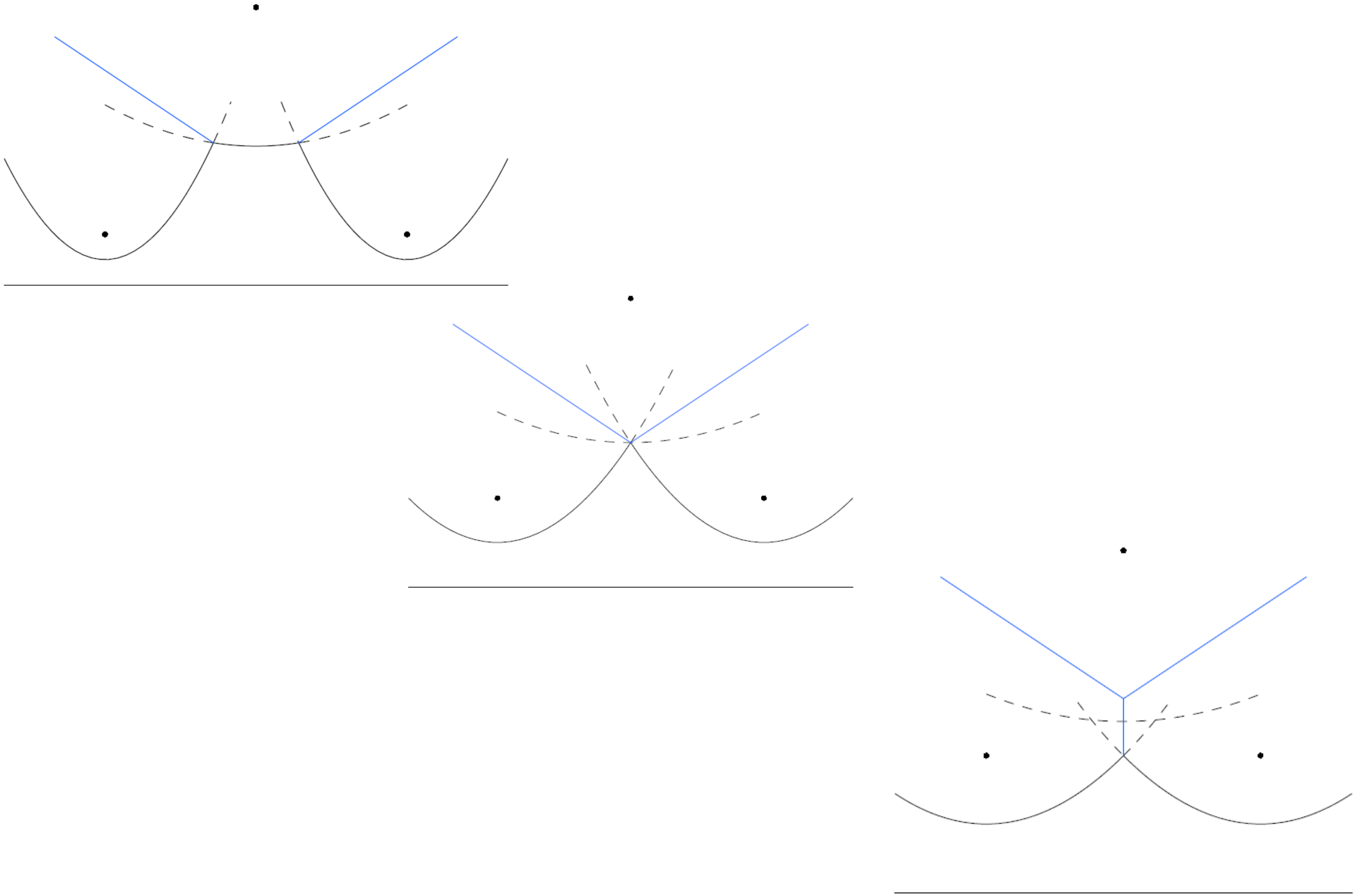
Site event

- Beach line consists of maximally $2n - 1$ parabolic arcs, because each generating point creates one parabola and divides maximally one existing parabolic arc to two parts

Circle event

- When some of the parabolic arcs is terminated
- This happens when three parabolas generated by points P_i, P_j, P_k all intersect in point Q – then this point Q forms the new Voronoi vertex

Circle event



Sweep line (Fortune's algorithm)

- More information, details for implementation:
  - http://blog.ivank.net/fortunes-algorithm-and-implementation.html

Weighted Voronoi diagrams

- One of possible generalizations of VD, when each generating point is assigned to a weight. This weight influences the size and shape of the VD cell.
- Let's assign weight $w_i \in R$ to point P_i. Then we define the corresponding metrics as

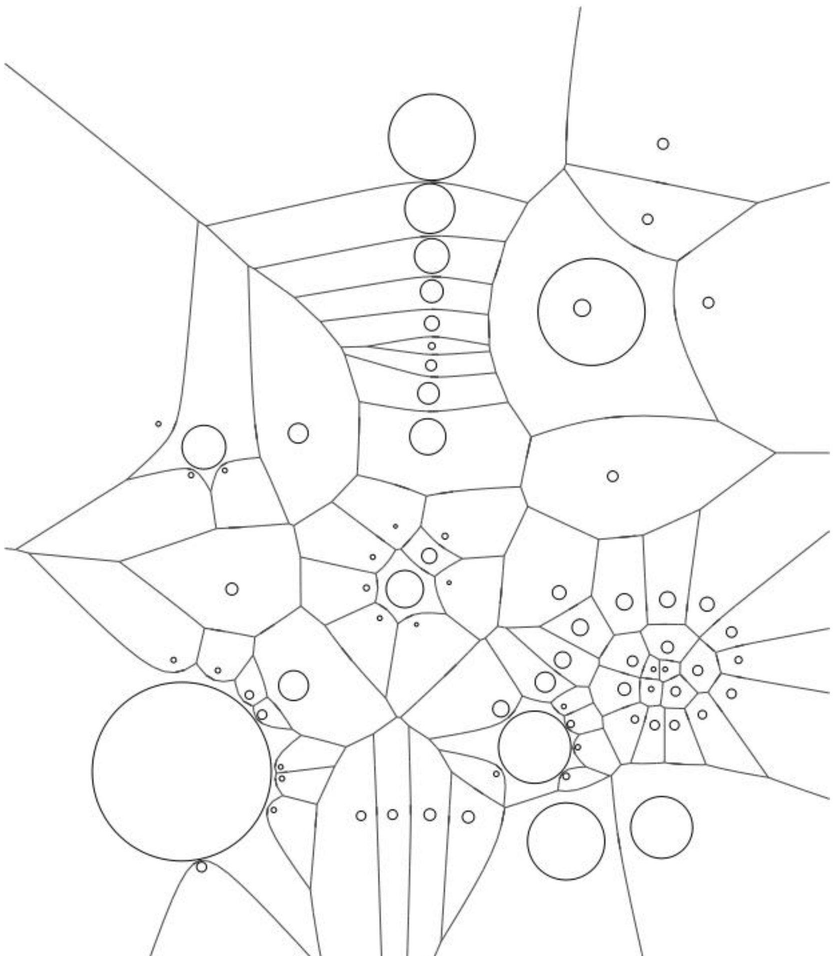
$$\mathrm{dist}_{\mathrm{WVD}}(P, Q) = \mathrm{dist}(P, Q) - w_i$$

where dist can be an arbitrary metrics

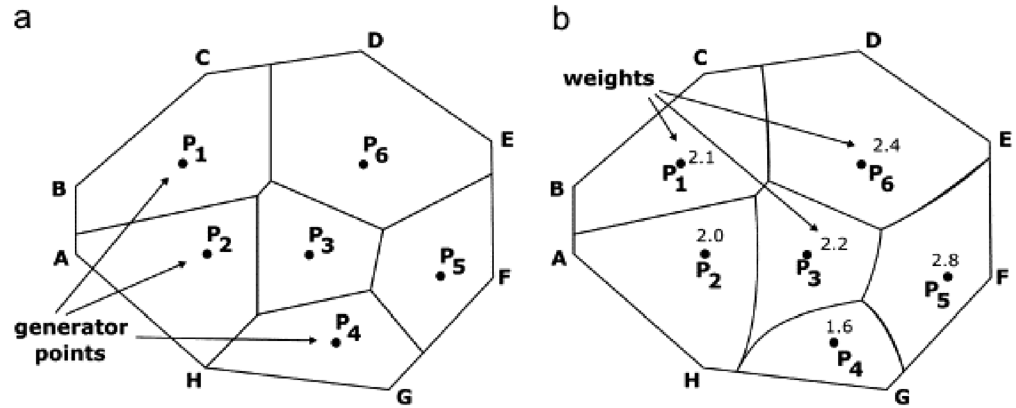
Weighted Voronoi diagrams

- When increasing the weight of a given point the corresponding VD cell is increasing which correspond to the given metric
- When the dist metric is the Euclidean distance, then $dist_{WVD}(P, P_i)$ can be interpreted as the distance of point P from a circle with center in P_i and radius w_i
- Voronoi edges are in this case parts of hyperbolas

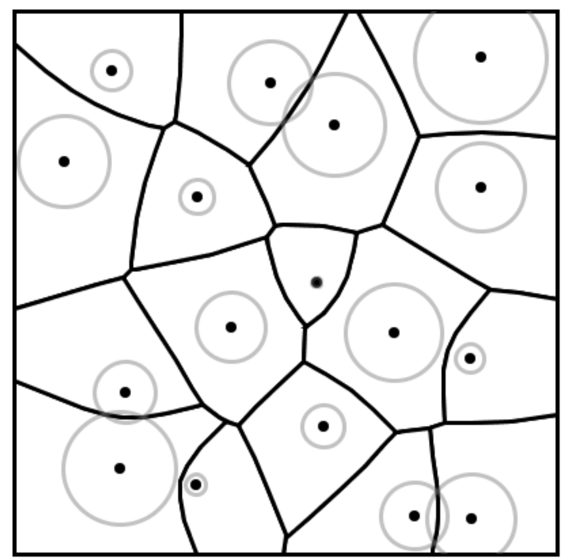
Weighted Voronoi diagrams



www.cgal.org



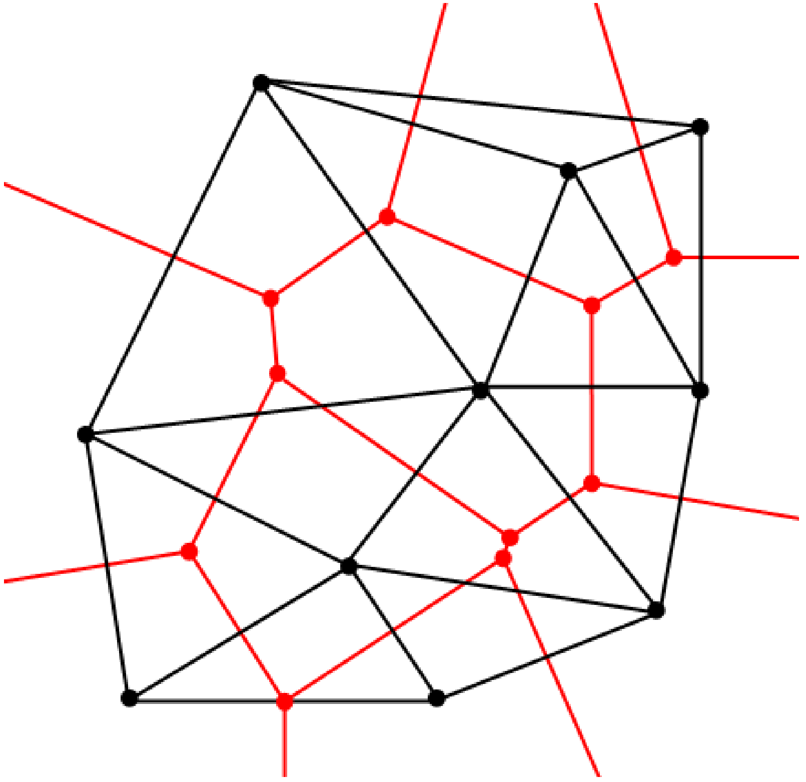
www.sciencedirect.com



d.hatena.ne.jp

Assignment

- Use the already constructed Delaunay triangulation for the construction of Voronoi diagram
- Visualize it



fr.wikipedia.org