Università
degli Studi
di Padova

The Euclidean MST: a rough journey

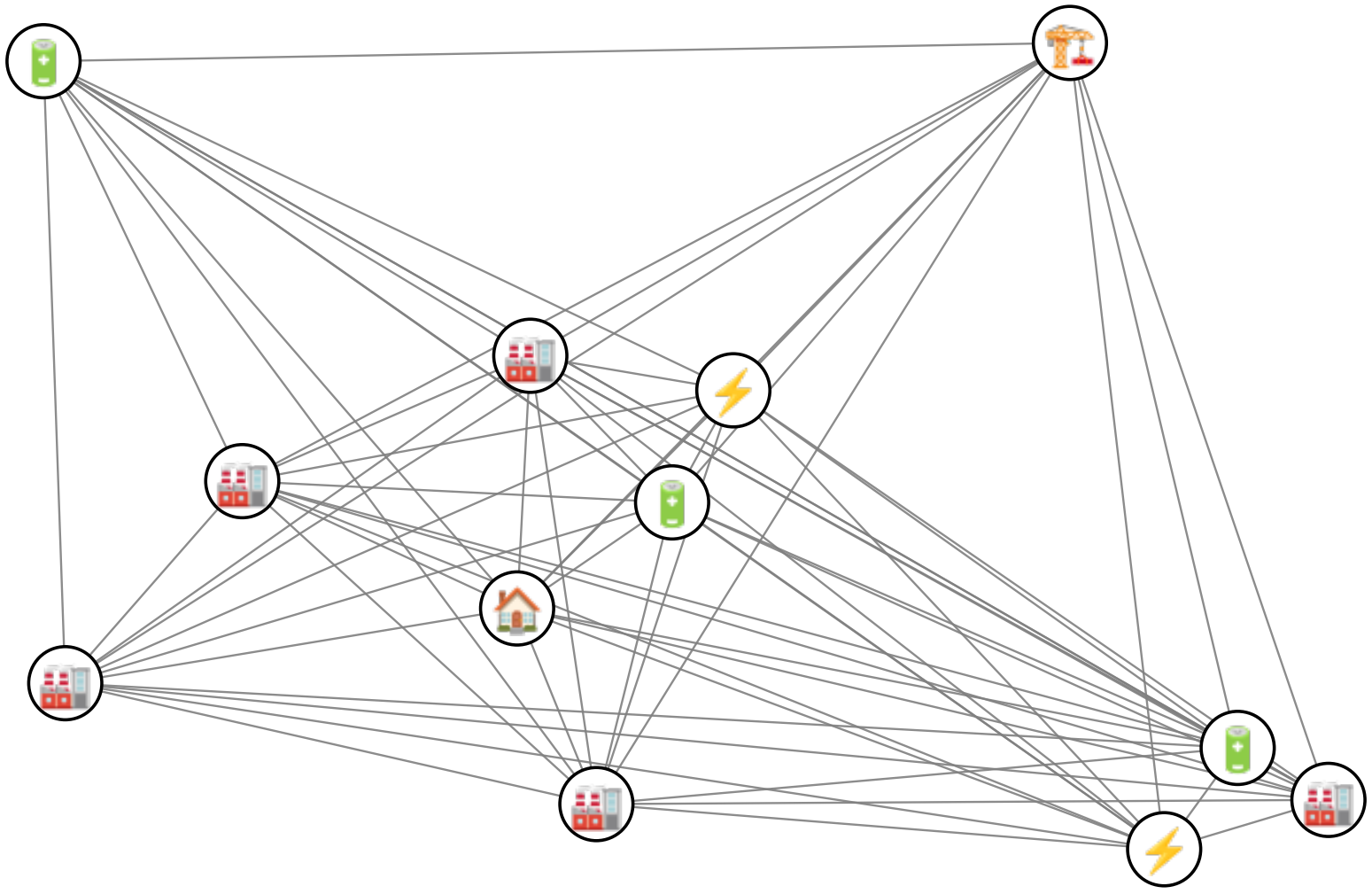
Alessandro Dario
Mentor: prof. Andrea Pietracaprina

Bertinoro – 3° weekend ortogonale

Index

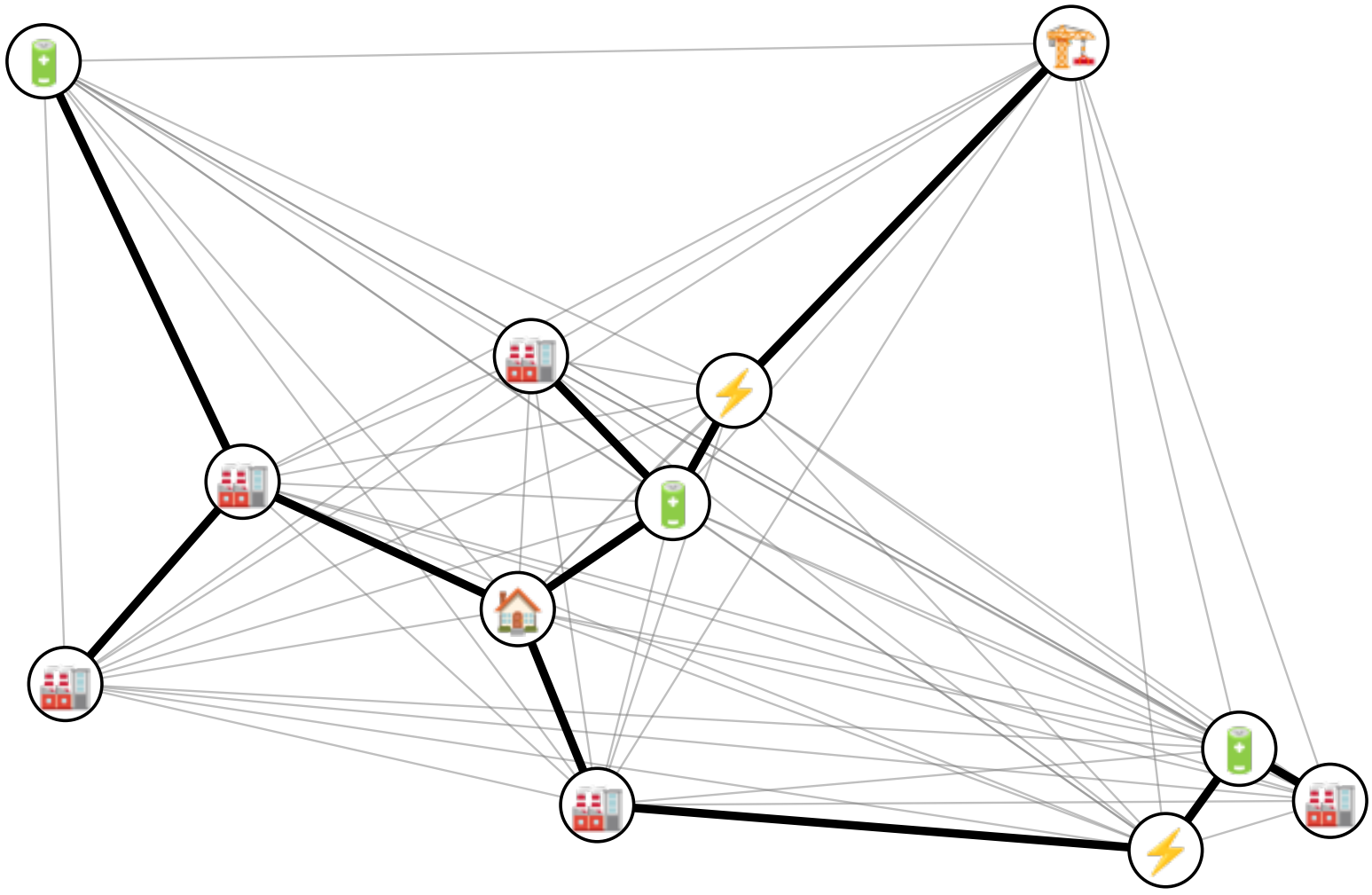
Motivating Example

Task: *build a **connected** electrical grid with the **least amount of wire** by connecting **pairs** of sites.*



Motivating Example

Task: *build a **connected** electrical grid with the **least amount of wire** by connecting **pairs** of sites.*



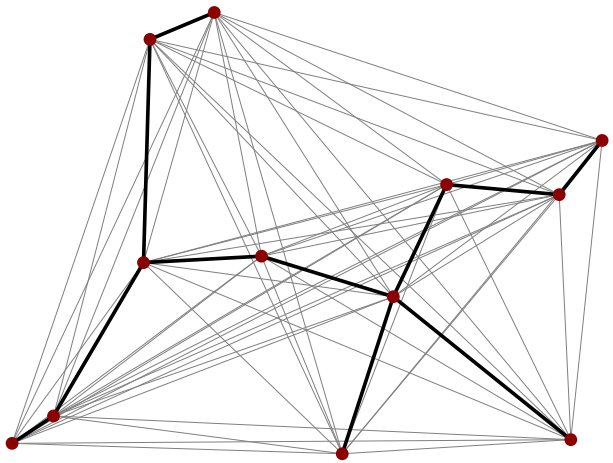
Problem statement

Given a set of points in the **Euclidean** space $\mathbb{R}^d$, let $G = (V, E)$ be the **complete graph** of the points, with weight:

$$w(e = (x, y)) = d(x, y).$$

The **minimum spanning tree** T^* of G is the spanning tree of G with minimum weight:

$$w(T^*) = \sum_{e \in T^*} w(e).$$



Applications

Optimization problems:

- Wire layout problems
- Wireless network connectivity problems
- Facility location problems
- Graph analysis, e.g. in cosmology

Important primitive:

- Basis for hierarchical clustering
- Approximation of the traveling salesman problem

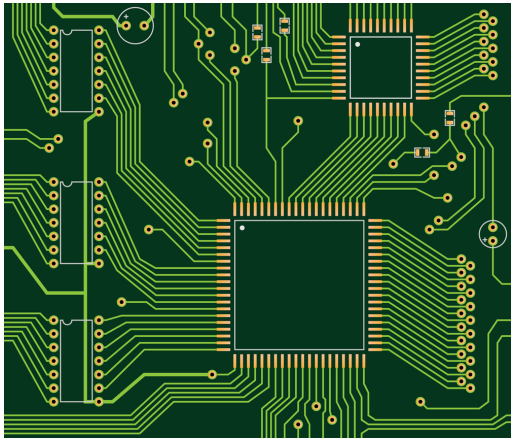
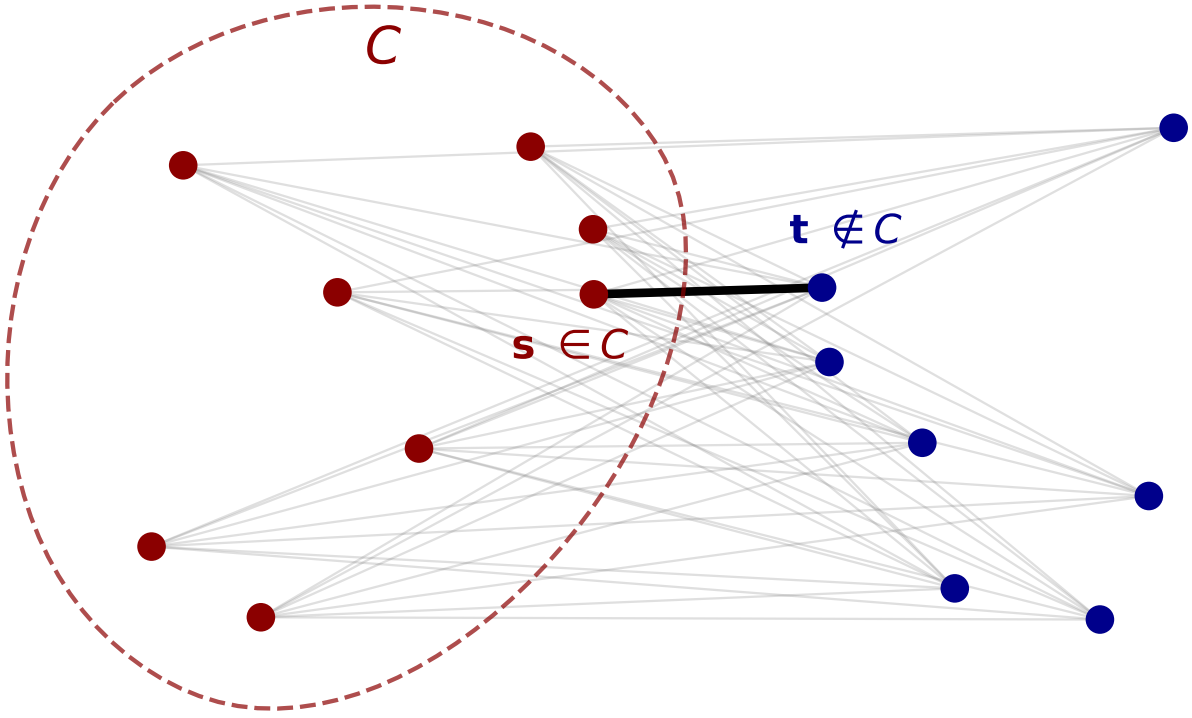


Figure: A printed circuit board.

MST Cut Property

Fundamental property of the MST:

- The **minimum** edge **crossing a cut** is always in the MST.
- Base for efficient **greedy** algorithms: Prim, Kruskal, Borůvka.



Borůvka's Algorithm

Borůvka's greedy algorithm for the generic graph (1926):

1: Each vertex $v \in V$ is an isolated component
2: **repeat**
3: **for all** component C **do**
4: Find the **minimum weight edge** that separates C from other components
5: **end for**
6: Add all found edges to the MST, excluding duplicates
7: **Merge** the connected **components**
8: **until** only one component remains

Borůvka's Algorithm

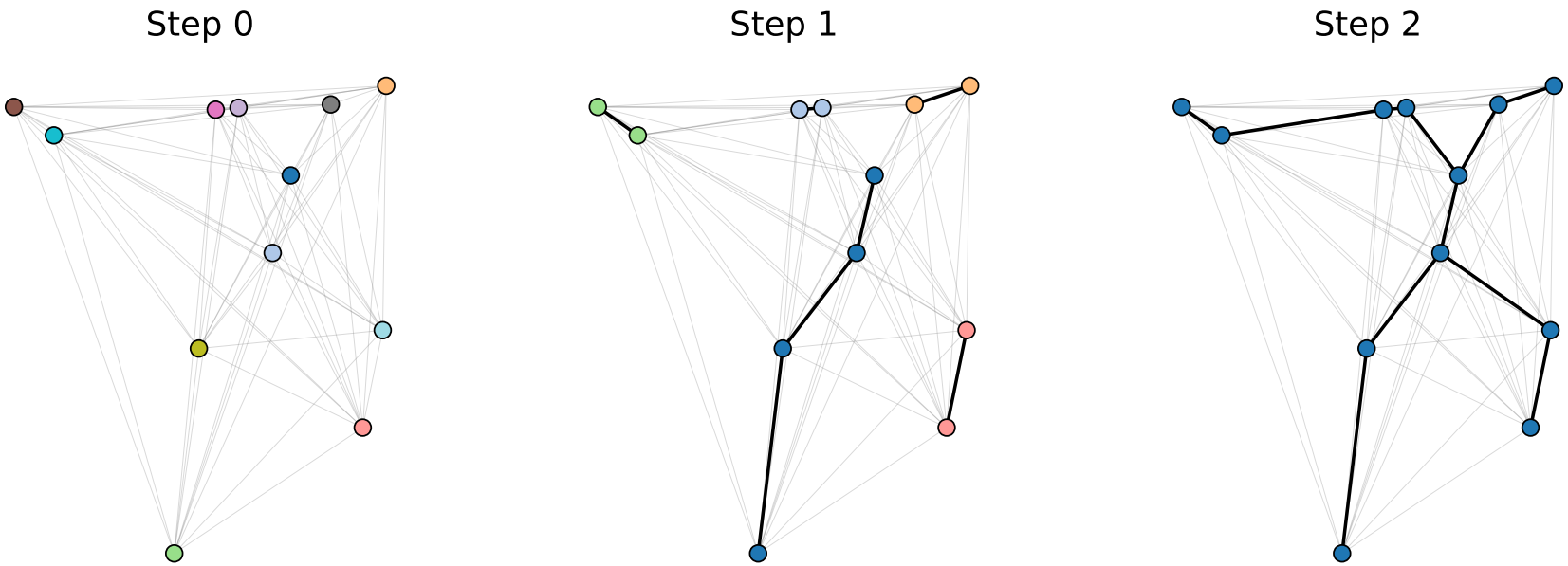


Figure: Execution of Borůvka's algorithm.

Complexity of Borůvka's Algorithm

Complexity:

- Each iteration must consider $O(|E|)$ edges.
- Each iteration **halves** the number of **components**.
- In total, at most $O(\log |V|)$ iterations.

Total complexity: $O(|E| \log |V|)$.

For the **complete graph** with $|E| = \binom{n}{2}$: $O(n^2 \log n)$.

Complexity of Borůvka's Algorithm

Complexity:

- Each iteration must consider $O(|E|)$ edges.
- Each iteration **halves** the number of **components**.
- In total, at most $O(\log |V|)$ iterations.

Total complexity: $O(|E| \log |V|)$.

For the **complete graph** with $|E| = \binom{n}{2}$: $O(n^2 \log n)$.

Complexity of the EMST

Enumerating all **edges** is expensive. Can we get the complexity of the EMST to $o(n^2)$?

- The fundamental primitive is **nearest neighbor** search.
- We would like to **exclude** *a-priori* **distant pairs**.
- We'll start with a simpler case: the EMST on the **plane**.

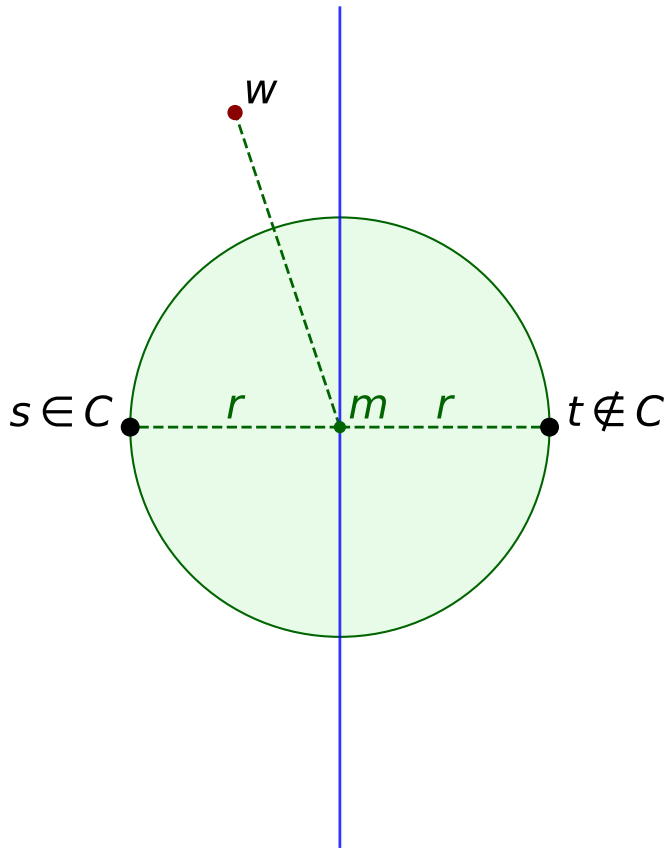


Figure: Can we leverage the cut property on the plane?

Index

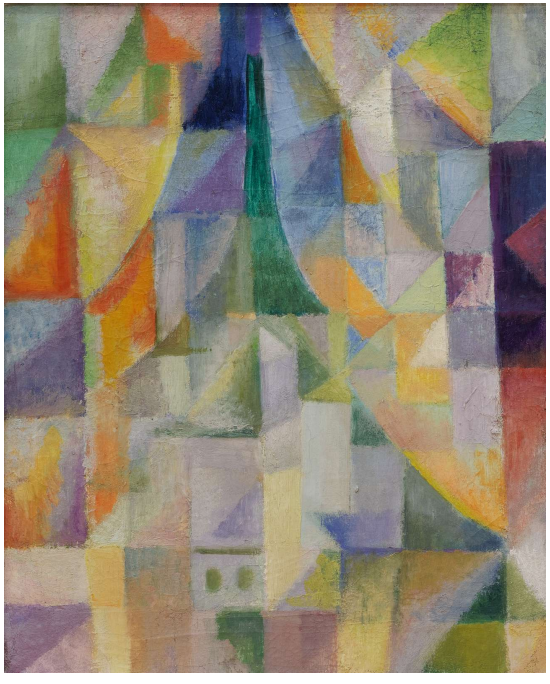
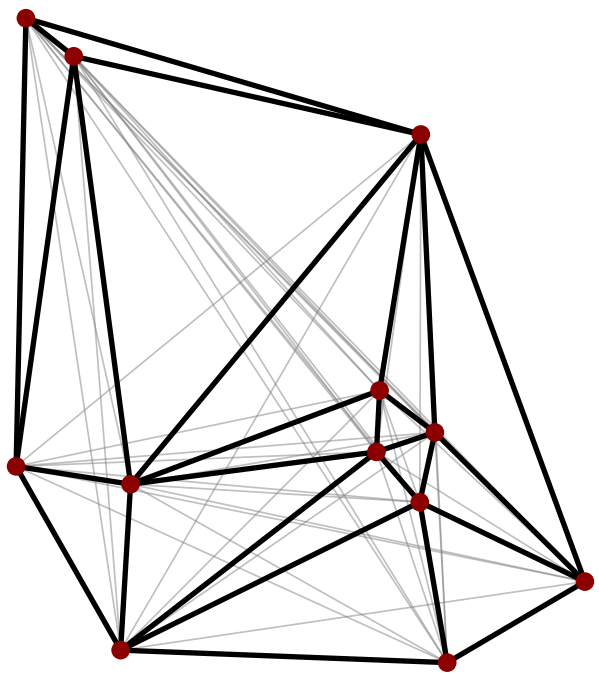


Figure: A painting by Robert Delaunay. Unrelated to Charles-Eugène Delaunay.

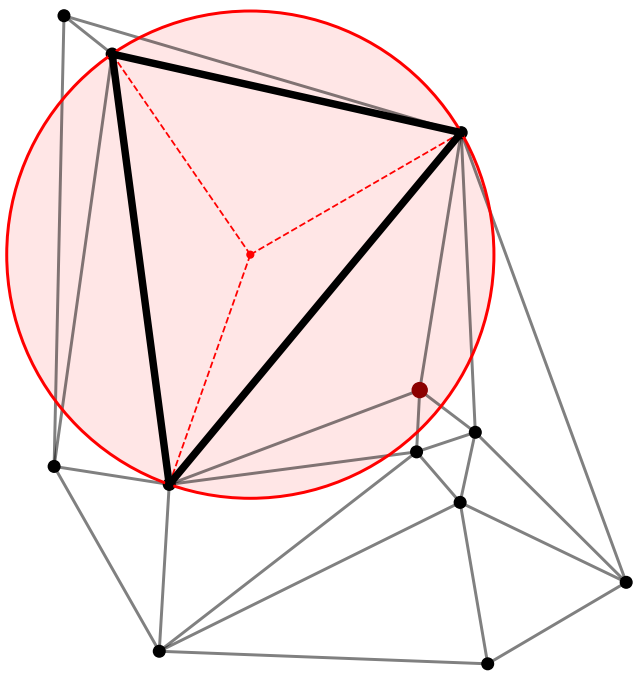
Point Triangulation

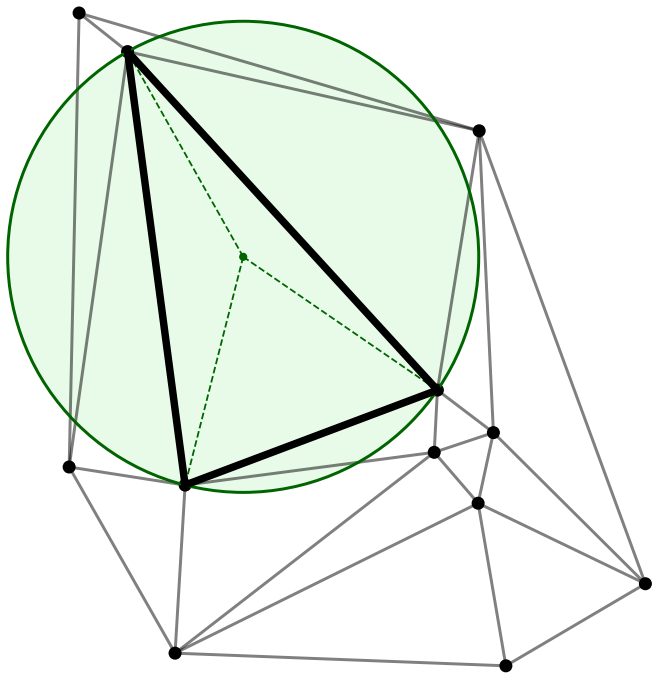
Let P be a set of points in the Euclidean space $\mathbb{R}^2$.
A **triangulation** of P is a set of non-intersecting **edges** connecting **all and only** the points of P, such that they **partition** the space enclosed by P into **triangles**.



Delaunay Triangulation

A triangulation T of P is the **Delaunay Triangulation** (DT) if it satisfies the **empty circle property**: for every triangle $\Delta \in T$, its **circumcircle** contains no other point of P in its interior except for its three vertices.





Delaunay Properties

Delaunay properties:

- Maximizes the **minimum angle** of triangles.
- **Unique** in the absence of degeneracies.
- **Fundamental property**:

$$MST \subseteq DT.$$

Index

The Voronoi Diagram

The Voronoi diagram of points $P \subset \mathbb{R}^2$ is the partition of the plane into convex **cells** $\{V(p) \mid p \in P\}$ where

$$V(p) = \{x : |x - p| \leq |x - q| \ \forall q \in P\}.$$

Each cell contains **all points** of the plane **closer to a site p than to any other** point in P.

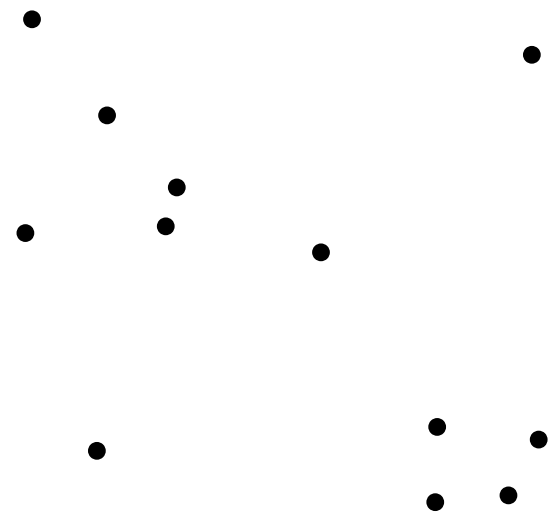


Figure: Voronoi diagram of a set of points.

The Voronoi Diagram

The Voronoi diagram of points $P \subset \mathbb{R}^2$ is the partition of the plane into convex **cells** $\{V(p) \mid p \in P\}$ where

$$V(p) = \{x : |x - p| \leq |x - q| \ \forall q \in P\}.$$

Each cell contains **all points** of the plane **closer to a site p than to any other** point in P.

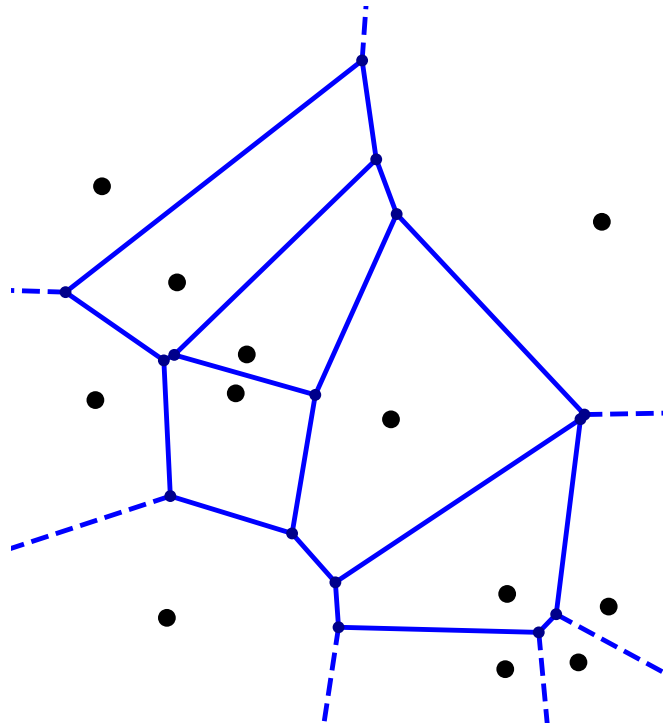


Figure: Voronoi diagram of a set of points.

Voronoi Properties

Properties of the Voronoi diagram:

- Cell intersections form Voronoi **edges**, segments equidistant from the **two** closest sites.
- Edges meet at Voronoi **vertices**, each equidistant from **three** sites.
- A Voronoi diagram for n sites has $O(n)$ **vertices** and $O(n)$ **edges**.
- Useful for several closeness problems on the plane: *closest point, furthest point, all closest points, largest empty circle, convex hull...*

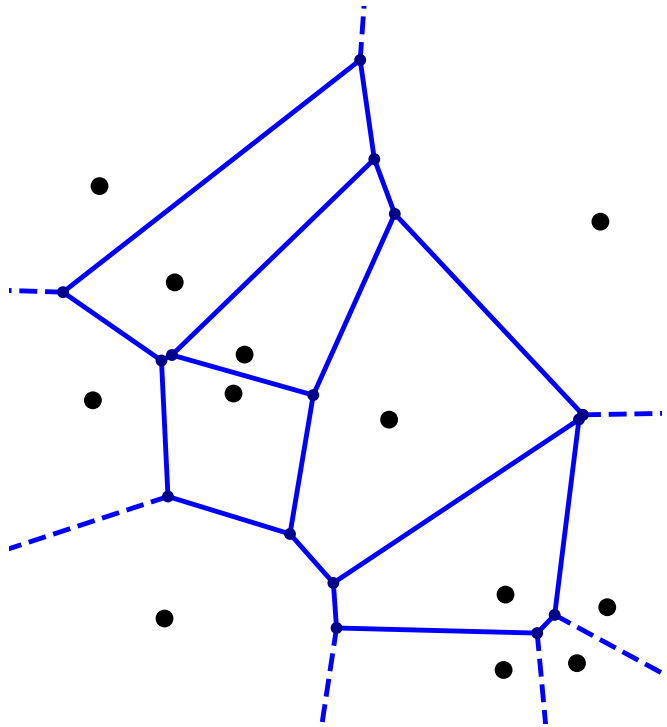
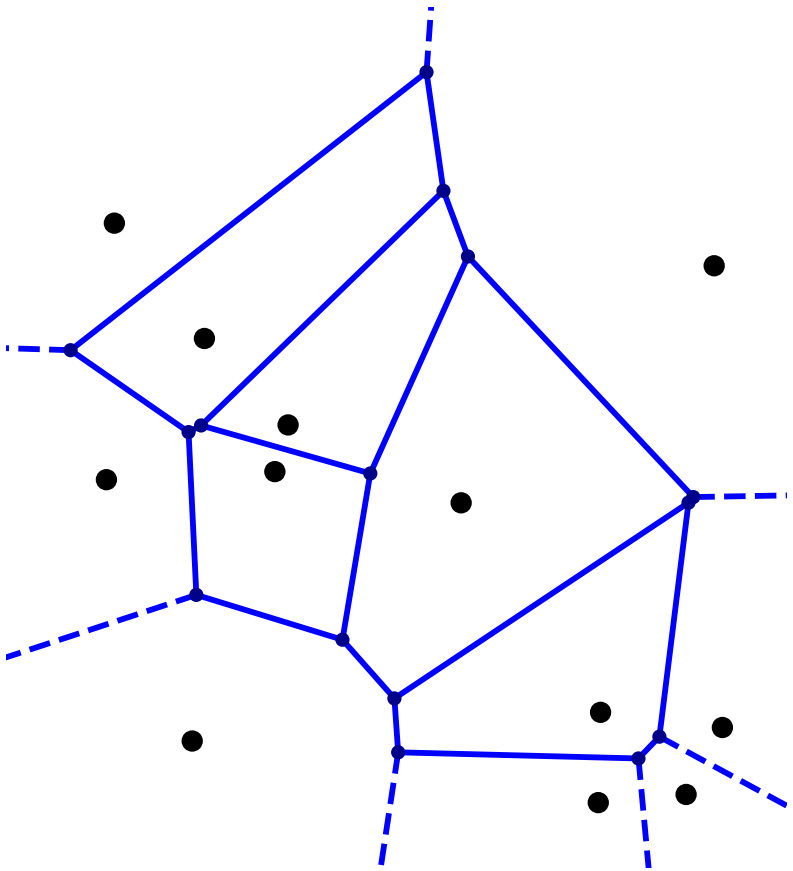


Figure: Voronoi diagram of a set of points.

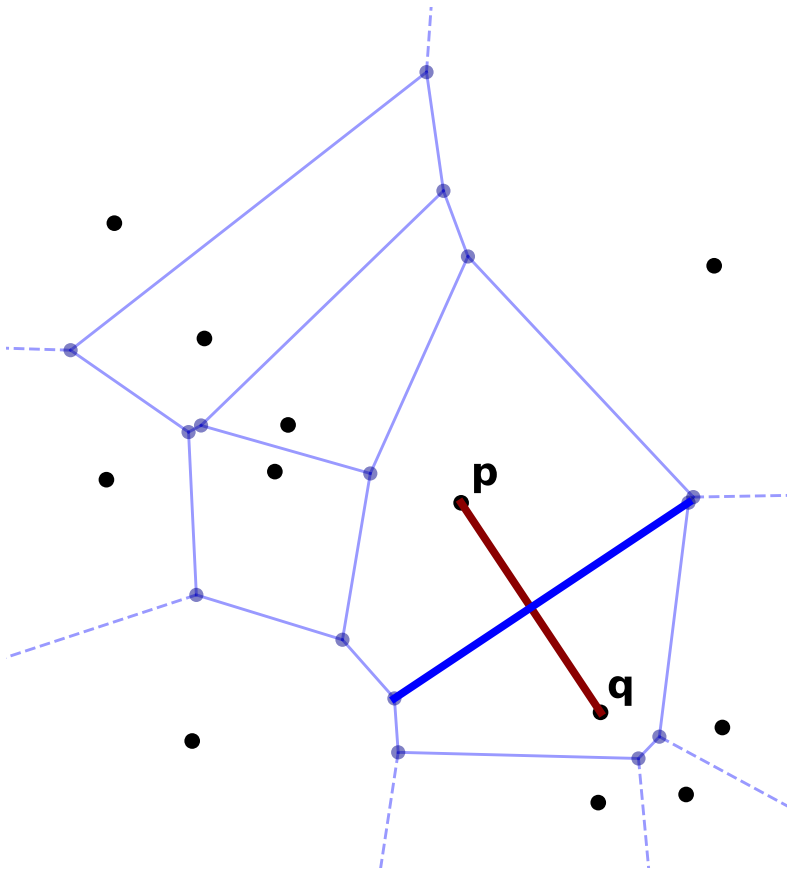
Geometric Duality with Delaunay

The **Voronoi** diagram is the **dual** of the **Delaunay** triangulation: two sites p, q share a Voronoi edge if and only if (p, q) is a Delaunay edge.



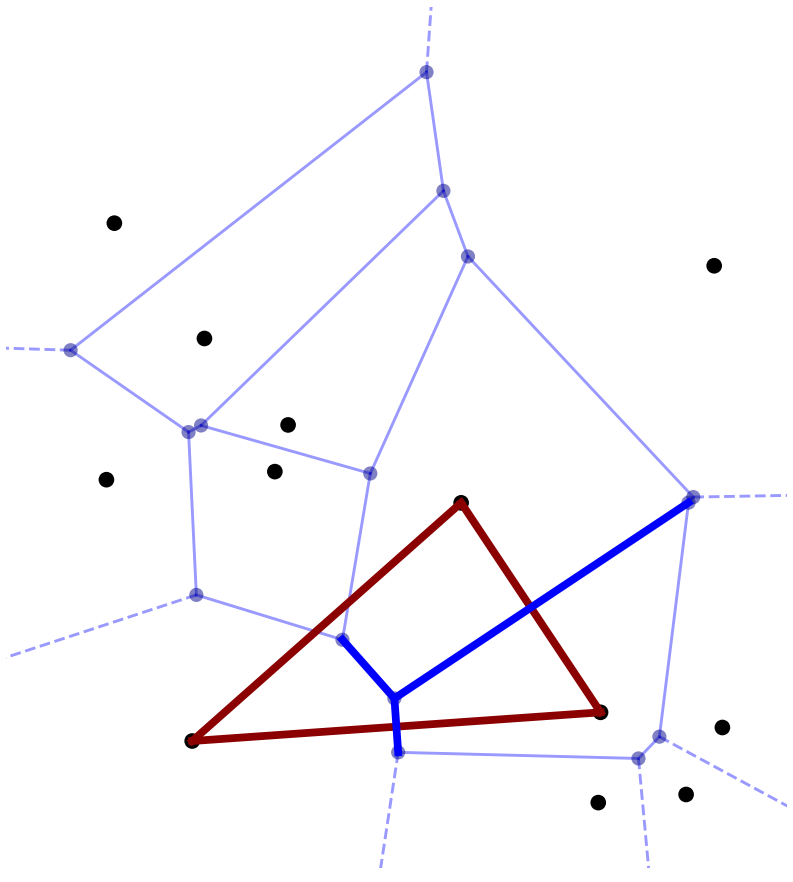
Geometric Duality with Delaunay

The **Voronoi** diagram is the **dual** of the **Delaunay** triangulation: two sites p, q share a Voronoi edge if and only if (p, q) is a Delaunay edge.



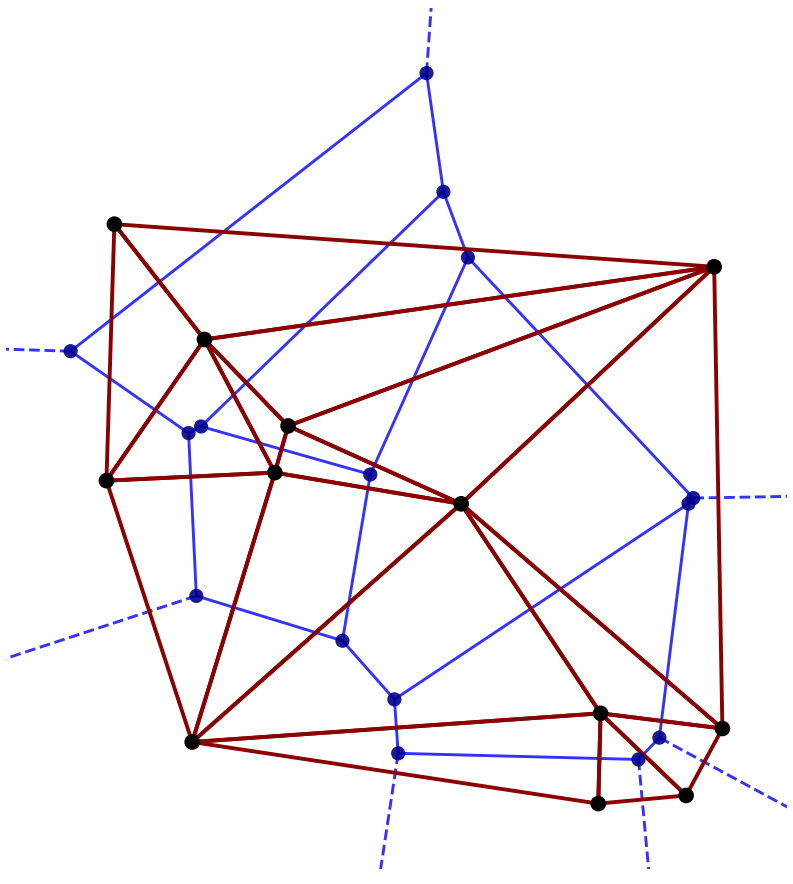
Geometric Duality with Delaunay

The **Voronoi** diagram is the **dual** of the **Delaunay** triangulation: two sites p, q share a Voronoi edge if and only if (p, q) is a Delaunay edge.



Geometric Duality with Delaunay

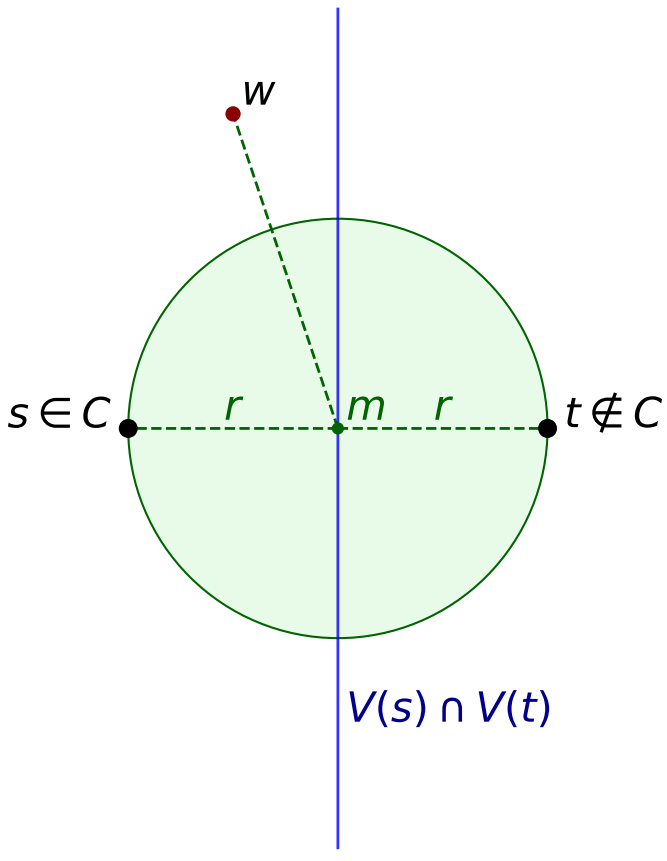
The **Voronoi** diagram is the **dual** of the **Delaunay** triangulation: two sites p, q share a Voronoi edge if and only if (p, q) is a Delaunay edge.



Delaunay Contains MST

Let $st \in MST$ be the **minimum length edge** crossing a cut C selected by Borůvka's algorithm.

1. $\nexists w \in P$ inside the circle having st as diameter, otherwise st is not of minimum length, as both $sw \leq st \wedge wt \leq st$ and one of the two crosses the cut C.
2. s, t are the two sites closest to midpoint m.
3. There **exists a Voronoi edge** separating s and t.
4. Thus $st \in DT$.



Voronoi substructure

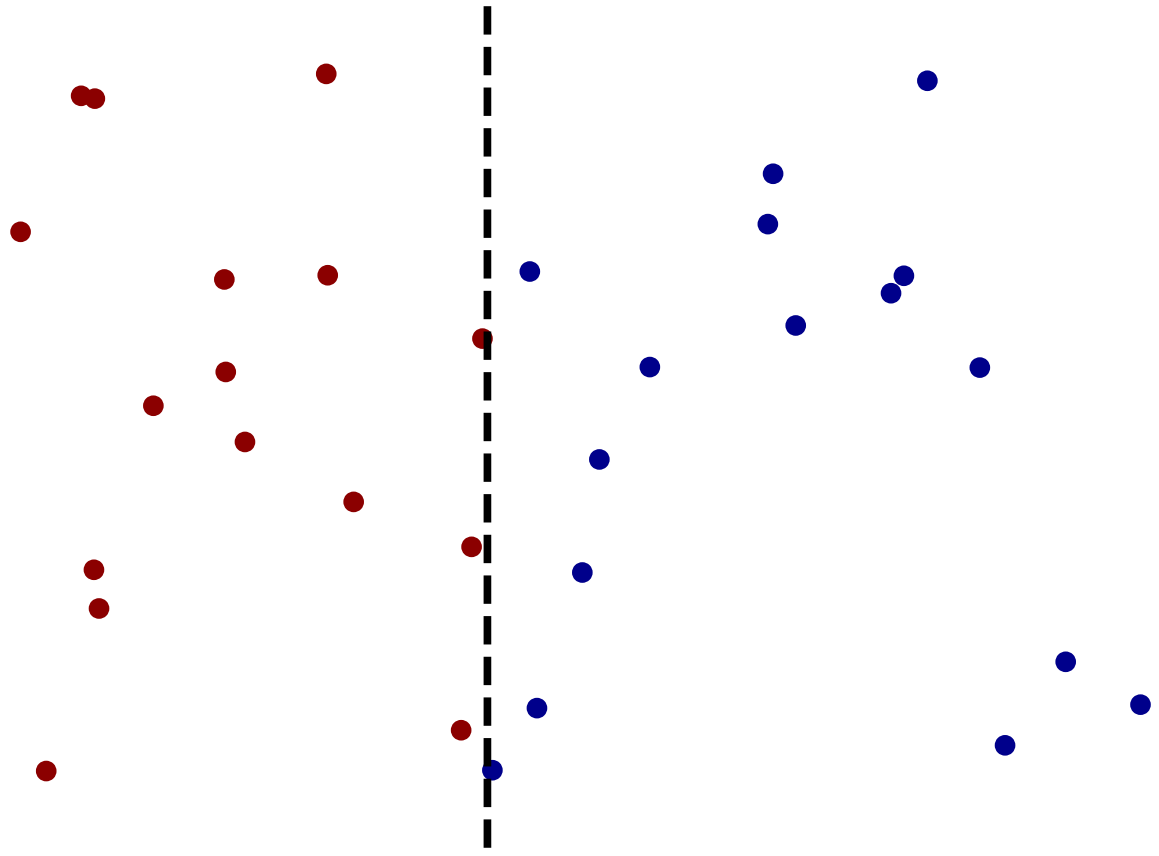


Figure: Voronoi substructure. **Partition** the points along an axis: the whole Voronoi diagram is the **union** of the two half Voronoi diagrams, properly intersected with the **boundary line**.

Voronoi substructure

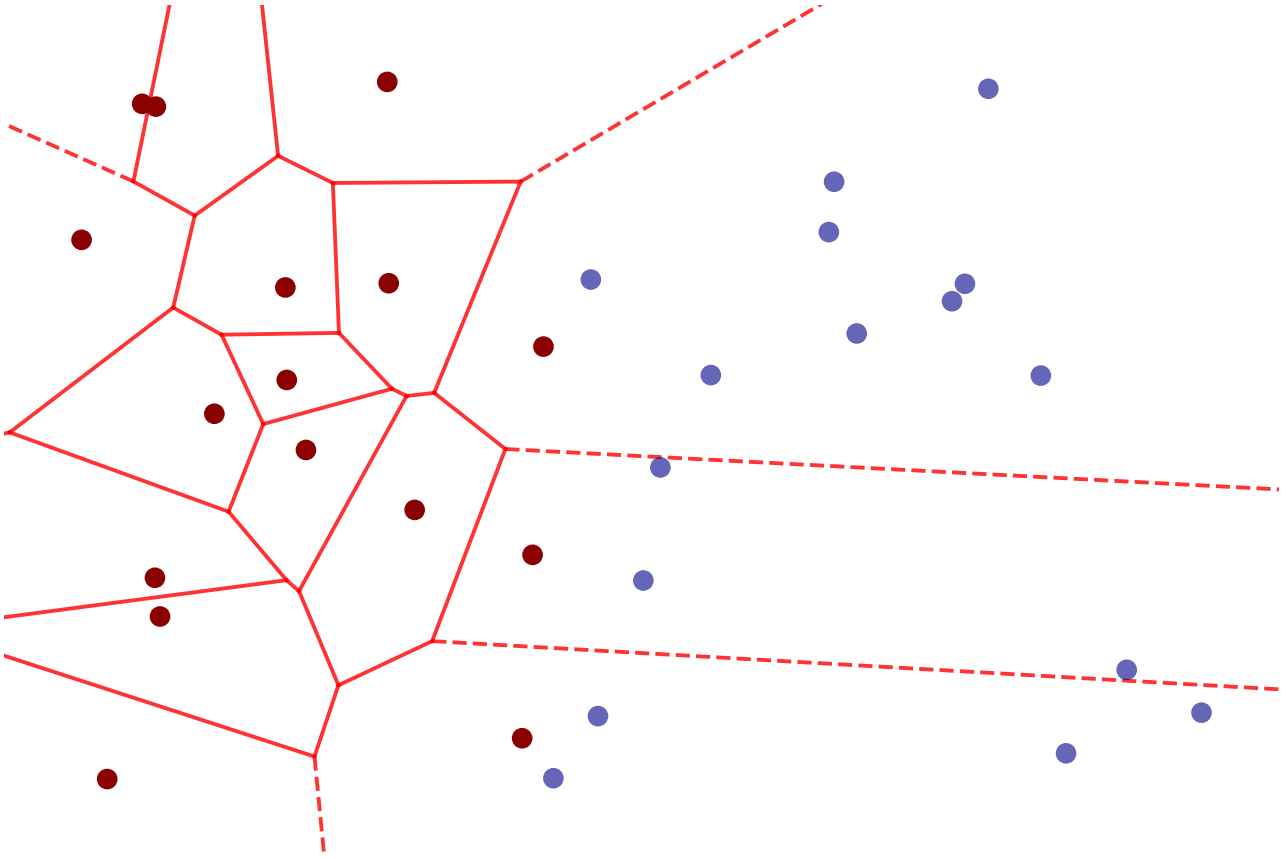


Figure: Voronoi substructure. **Partition** the points along an axis: the whole Voronoi diagram is the **union** of the two half Voronoi diagrams, properly intersected with the **boundary line**.

Voronoi substructure

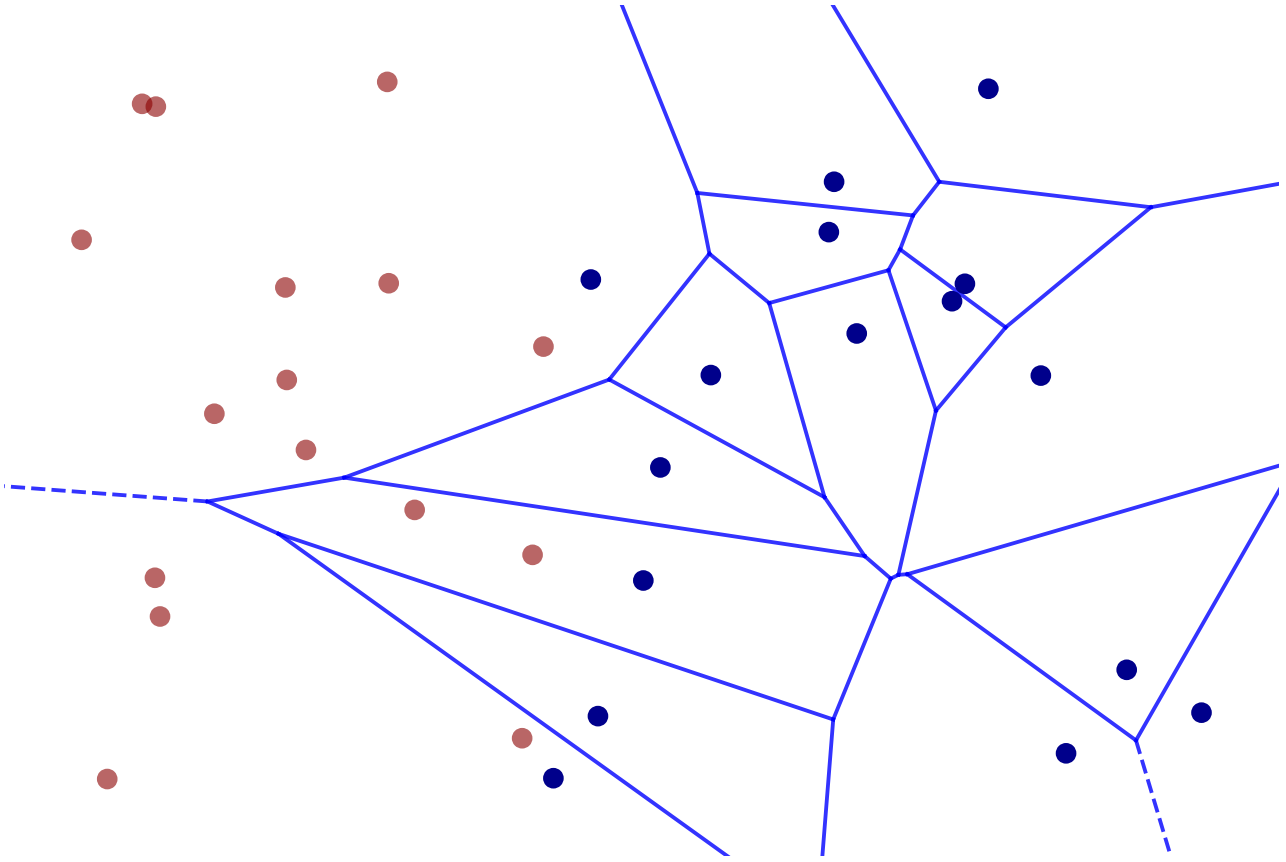


Figure: Voronoi substructure. **Partition** the points along an axis: the whole Voronoi diagram is the **union** of the two half Voronoi diagrams, properly intersected with the **boundary line**.

Voronoi substructure

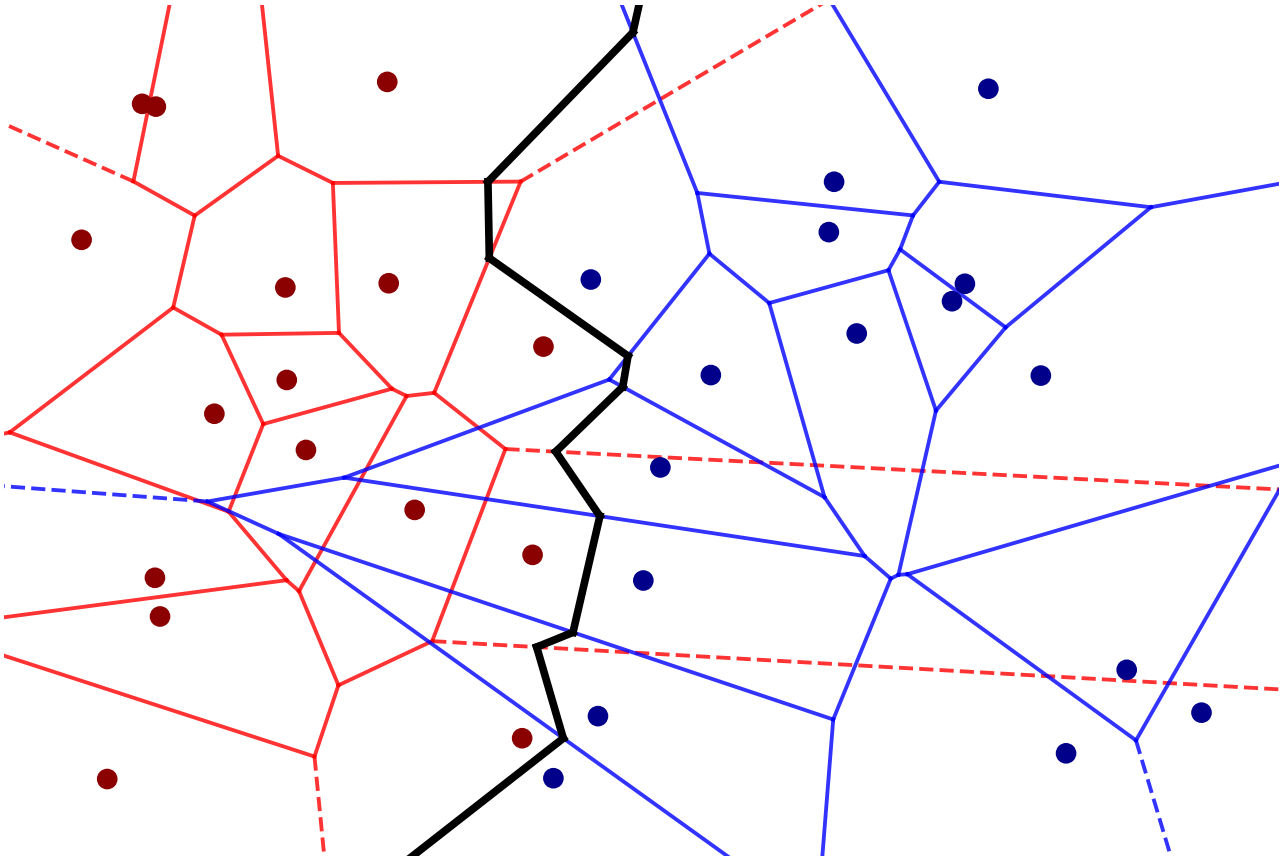
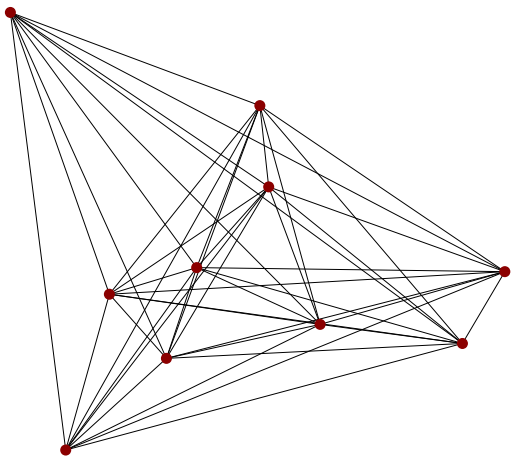


Figure: Voronoi substructure. **Partition** the points along an axis: the whole Voronoi diagram is the **union** of the two half Voronoi diagrams, properly intersected with the **boundary line**.

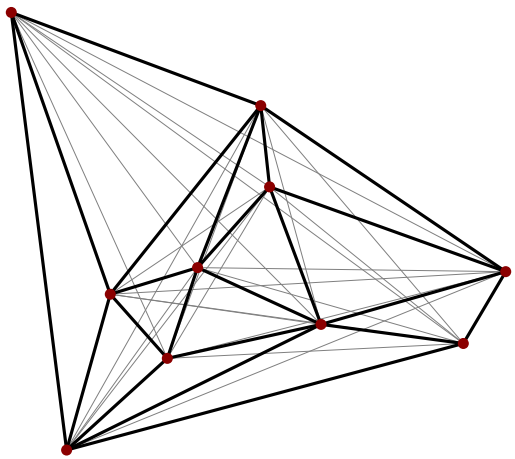
Algorithm for the EMST

An algorithm for the MST of P follows immediately:

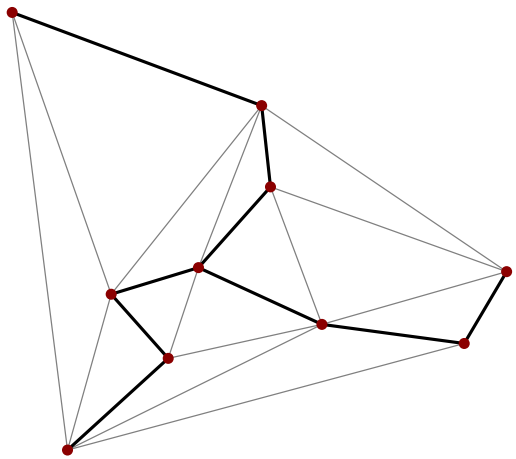
1. Compute the **Voronoi diagram**, in $O(n \log n)$.
2. Use **duality** to compute the **Delaunay** triangulation, in $O(n)$.
3. Apply **Borůvka**'s algorithm to the graph, in $O(n \log n)$.



All Edges



Delaunay



MST

Index

The Curse of Dimensionality

How to solve the Euclidean MST problem in **high dimensions**?

- Voronoi cells are practical only in 2D or 3D.
- In d dimensions, Voronoi cells are polyhedra with a number of faces **exponential** in d.
- Construction requires $\Omega(n^{\frac{1}{2}d})$ operations.

Dual-Tree Borůvka Algorithm

March et al. present the **Dual-Tree Borůvka Algorithm** in 2010.

- Keeps the **outer iterations** of the Borůvka algorithm.
- Solves the *all-points nearest-neighbor* primitive using **data structures** typical of *NN-search*.

Fast Euclidean Minimum Spanning Tree: Algorithm, Analysis, and Applications

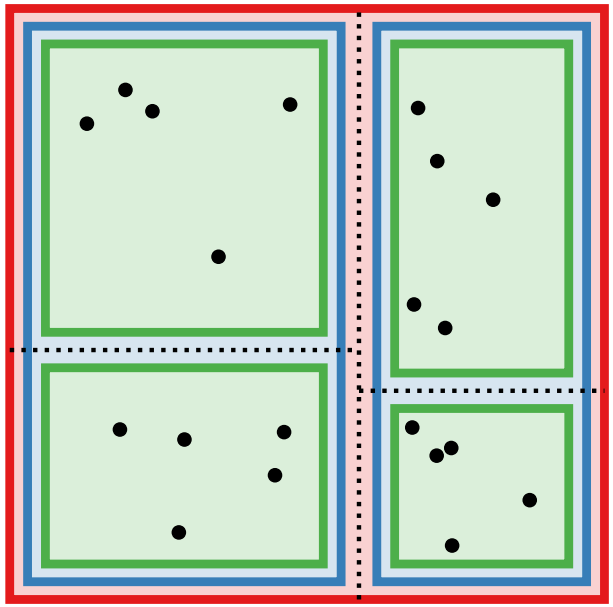
William B. March Parikshit Ram Alexander G. Gray
School of Computational Science & Engineering, Georgia Institute of Technology
266 Ferst Dr., Atlanta, GA 30332
{march@, p.ram@, agray@cc.}gatech.edu

ABSTRACT

The Euclidean Minimum Spanning Tree problem has applications in a wide range of fields, and many efficient algorithms have been developed to solve it. We present a new, fast, general EMST algorithm, motivated by the clustering and analysis of astronomical data. Large-scale astronomical surveys, including the Sloan Digital Sky Survey, and large simulations of the early universe, such as the Millennium Simulation, can contain millions of points and fill terabytes

this long-standing theoretical and algorithmic interest, the MST is useful for many practical data analysis problems. Many optimization problems can be posed as the search for the MST in a network [36]. The MST is also used as an approximation for the traveling salesman problem [24], in document clustering [48], analysis of gene expression data [15], wireless network connectivity [46], percolation analyses [8], and modeling of turbulent flows [44], among other areas. These problems are commonly solved in the Euclidean set-

Dual-Tree Borůvka Algorithm



Partition the space recursively, for example with a kd-tree.

- Searching the nearest neighbors individually for each point would be inefficient.
- Instead, we traverse the tree **recursively**. Using pairs of **nodes** of the tree we can compare many pairs of points **in "parallel"**.
- We can **prune** comparisons between distant pairs of nodes.

Figure: The first 3 levels of a kd-tree in 2D.

Dual-Tree Borůvka Algorithm

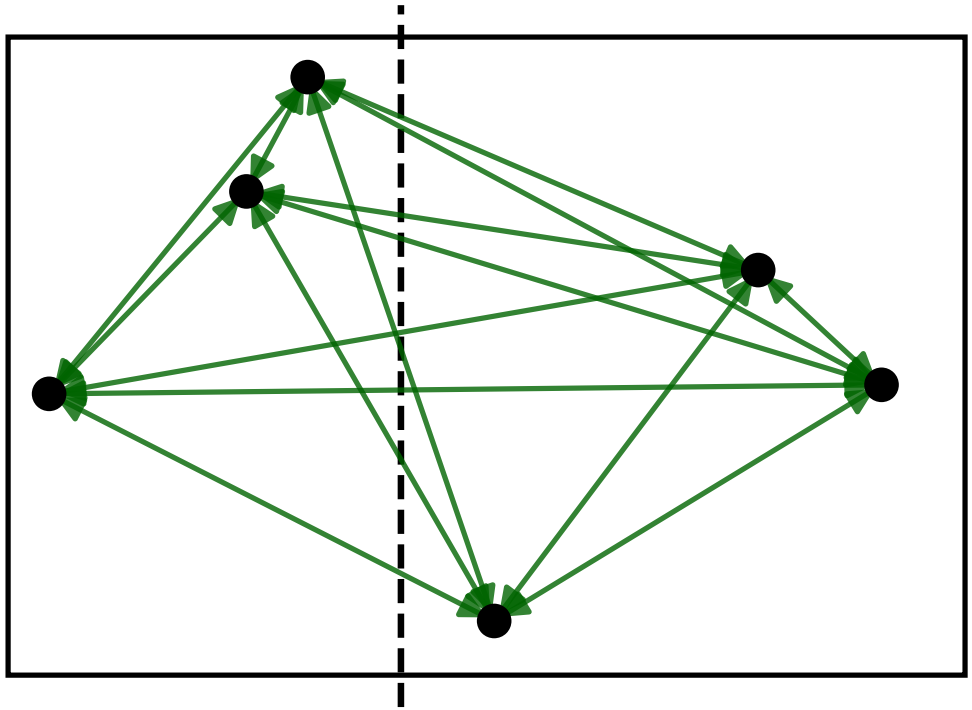


Figure: How the DTB algorithm splits pairs recursively.

Dual-Tree Borůvka Algorithm

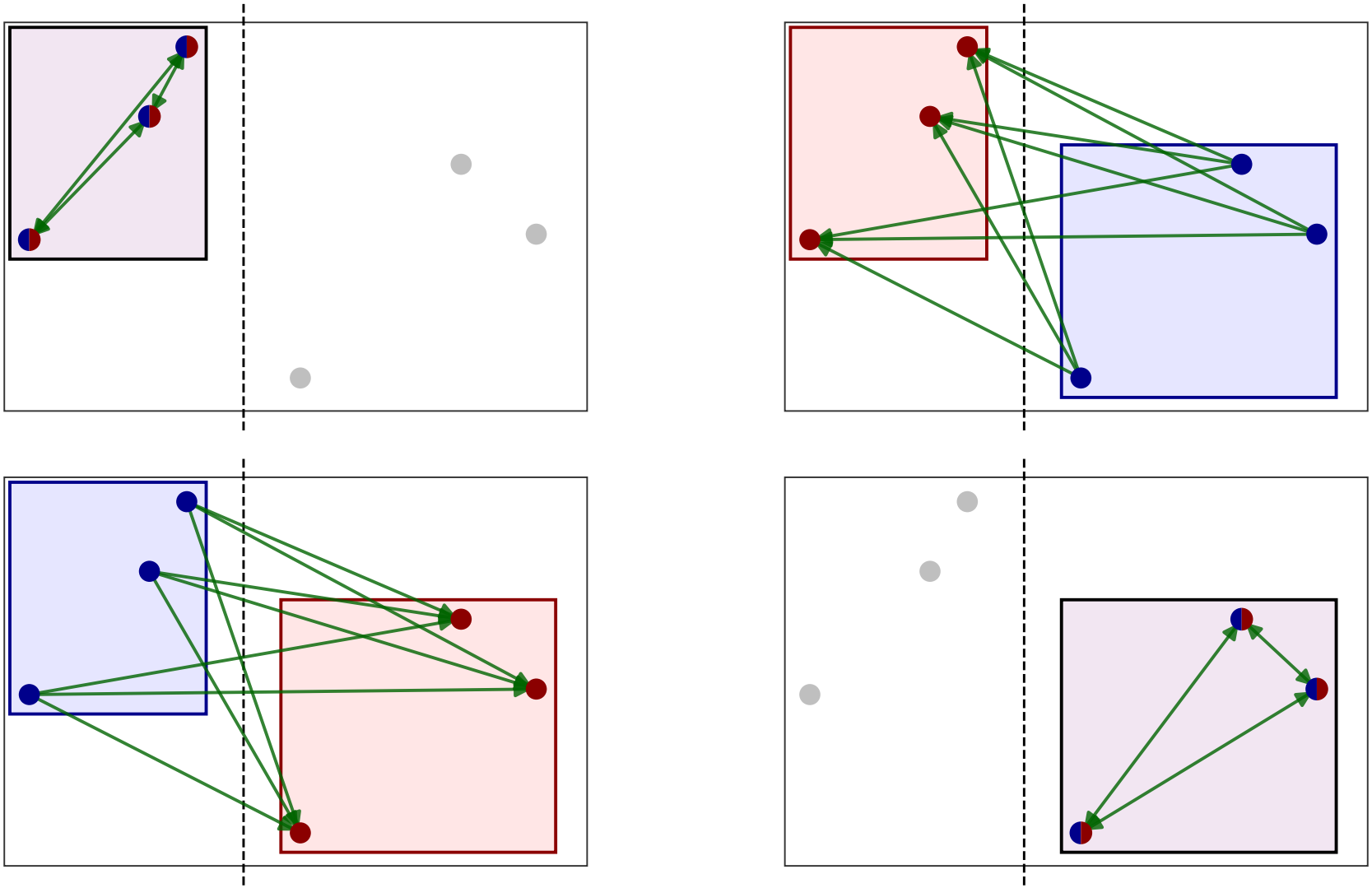


Figure: How the DTB algorithm splits pairs recursively.

Dual-Tree Borůvka Algorithm

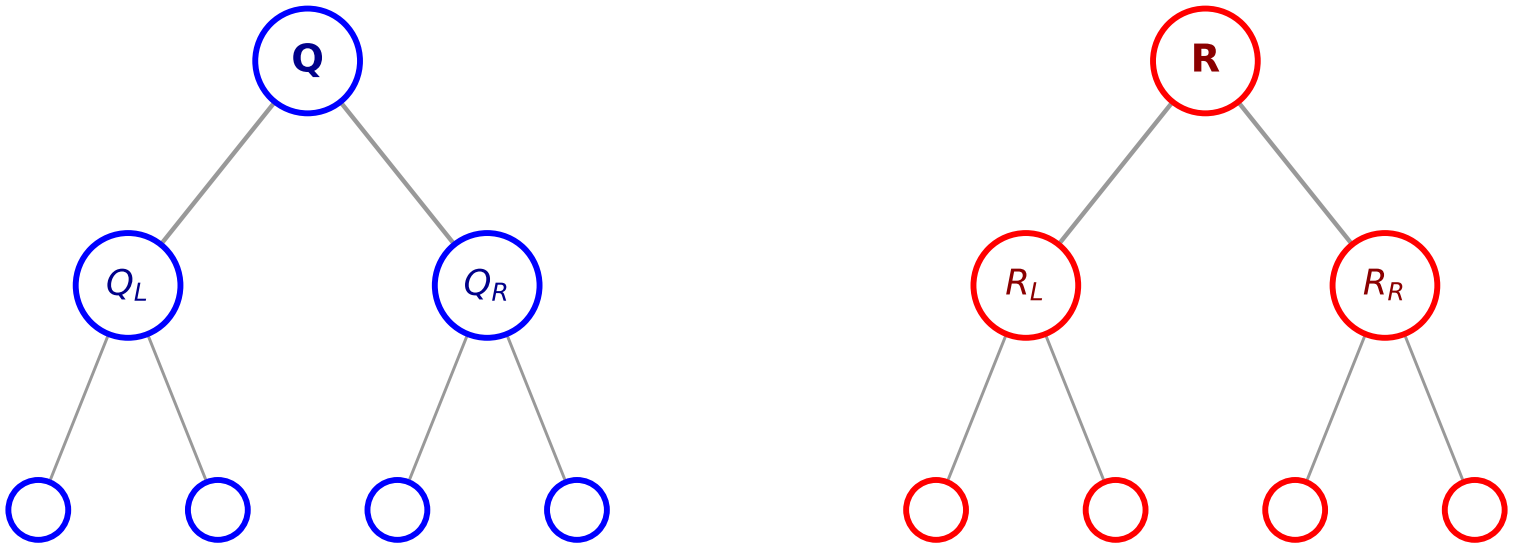


Figure: We traverse **in parallel** two pointers to the tree same kd-tree: the **query** tree Q and the **reference** tree R. We want to find for each $q \in Q$ the nearest $r \in R$, $r \neq q$.

Dual-Tree Borůvka Algorithm

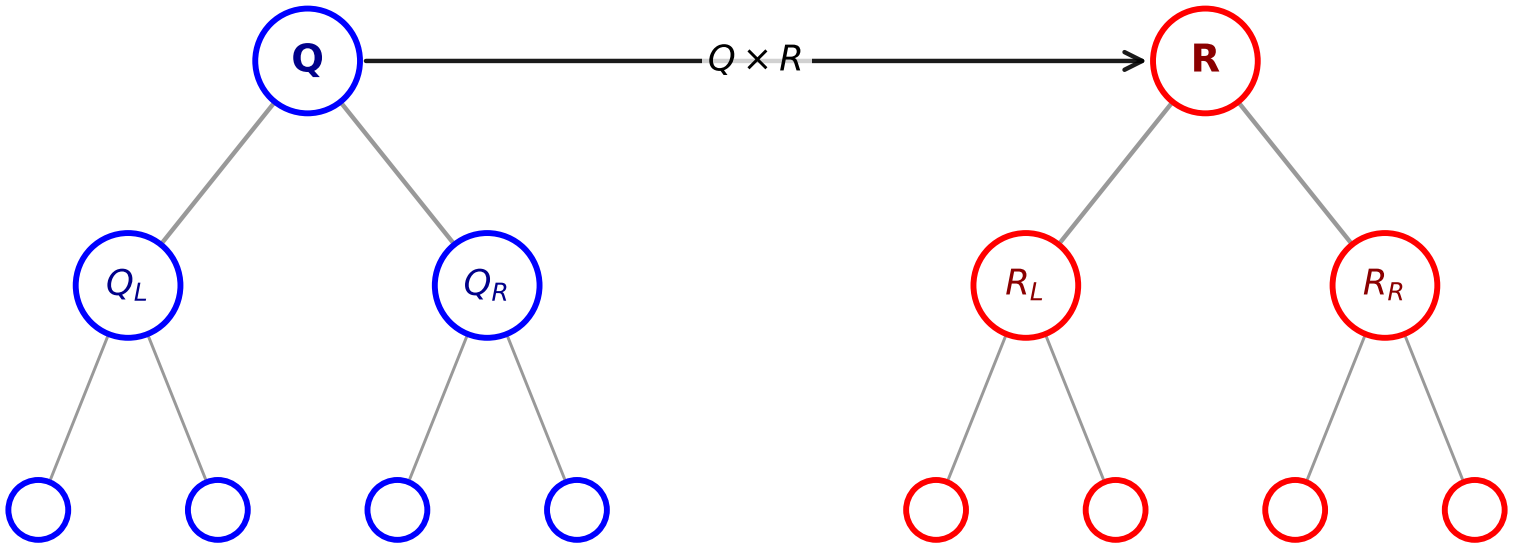


Figure: We traverse **in parallel** two pointers to the tree same kd-tree: the **query** tree Q and the **reference** tree R. We want to find for each $q \in Q$ the nearest $r \in R$, $r \neq q$.

Dual-Tree Borůvka Algorithm

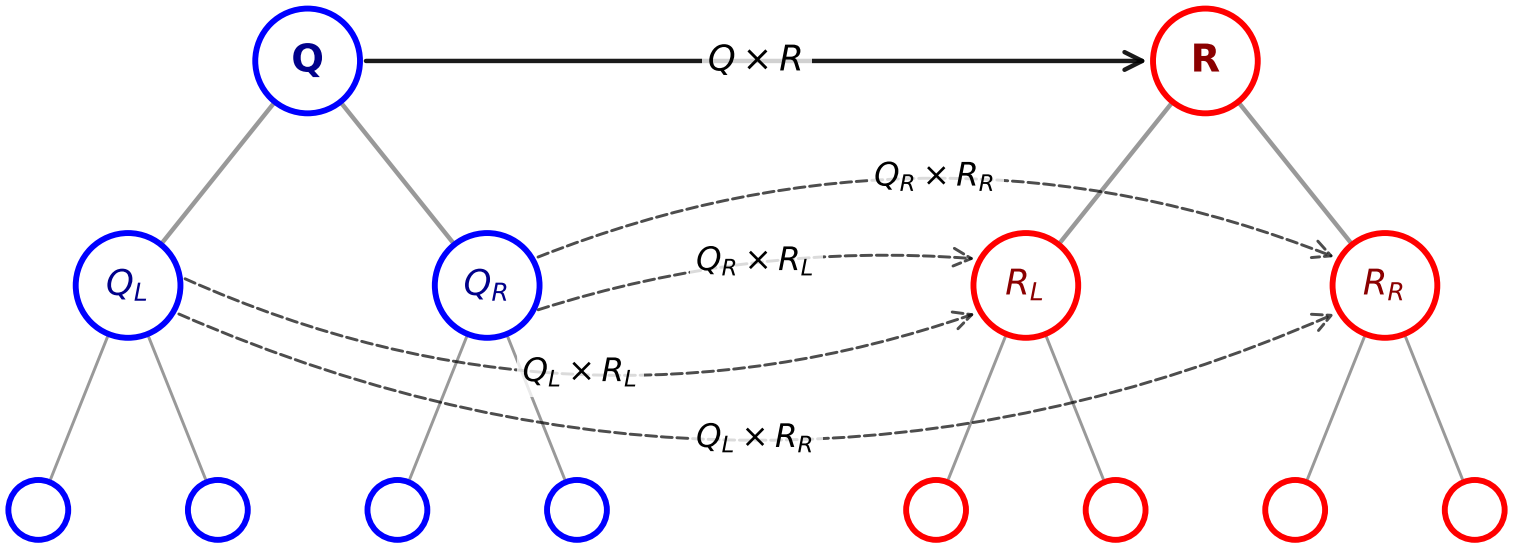


Figure: We traverse **in parallel** two pointers to the tree same kd-tree: the **query** tree Q and the **reference** tree R. We want to find for each $q \in Q$ the nearest $r \in R$, $r \neq q$.

Dual-Tree Borůvka Algorithm

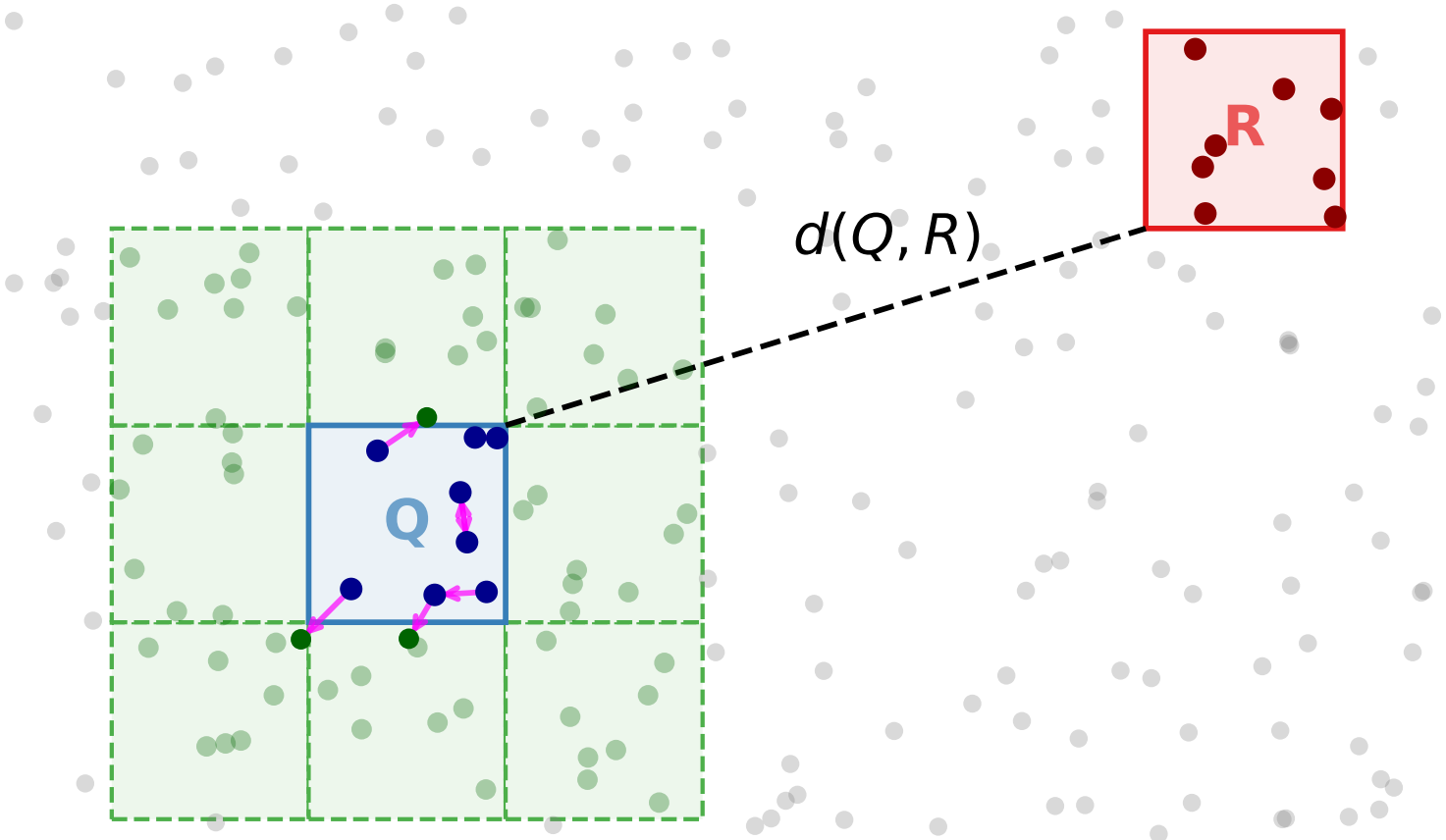


Figure: Points of Q will find their nearest neighbors in Q or in its adjacent nodes. Distant nodes such as R can be **pruned**.

Algorithm for all nearest neighbors

An algorithm for the *first iteration* of Dual-Tree Borůvka:

Algorithm AllNearestNeighbors(kd-tree nodes Q, R)

1: **if** $d(Q, R) > d(Q)$ **then** # prune all pairs in $Q \times R$
2: **return**
3: **else if** $Q = \{q\}$ and $R = \{r\}$ are leaves **then**
4: **if** q $\neq$ r and $d(q, r) < d(Q)$ **then**
5: $d(Q) \leftarrow d(q, r)$
6: $NN(q) \leftarrow (q, r)$
7: **end if**
8: **else**
9: AllNearestNeighbors($Q.left, R.left$)
10: AllNearestNeighbors($Q.left, R.right$)
11: AllNearestNeighbors($Q.right, R.left$)
12: AllNearestNeighbors($Q.right, R.right$)
13: # track the worst distance of the best neighbor for all points in Q
14: $d(Q) \leftarrow \max\{d(Q.left), d(Q.right)\}$
15: **end if**

Complexity of DTB

The **dual-tree** is a **practical approach** for many problems in space. Using more sophisticated data structures:

- During tree descent, each query node Q is compared with a **constant number** of reference nodes R.
- The number of neighbors depends on d but not on n.
- Each **iteration** can be completed in $O(n \cdot \alpha(n))$ operations.

  $\alpha(n)$ *is the inverse Ackermann function, which is nearly constant.*
- Borůvka requires $O(\log n)$ iterations.

Total complexity for spaces of fixed expansion constant:

$$O(\alpha(n) \cdot n \log n).$$

The Curse of Dimensionality

DTB does not scale well with d.

- As d increases, DTB must visit an **exponentially larger adjacency set** for each query Q.
- For very large d, this set is large enough to include all n points, pruning is minimal and the algorithm **checks all pairs**.

Complexity as a function of dimension is:

$$O(c \cdot \alpha(n) n \log n), \quad c \in O(1)^d.$$

Approximate EMST

Practical solution for high dimensional problems:

- We accept an ε-**approximation**, i.e. an MST T with weight

$$w(T) \leq (1+\varepsilon)w(T^*).$$

- State of the art *(Arya and Mount)* requires

$$O(n \log n + n\varepsilon^{-2} \log^2 \tfrac{1}{\varepsilon}).$$

- Theoretical analysis does *not* eliminate a factor $O(1)^d$.

References

- M. I. Shamos and D. Hoey. Closest-point problems. In *Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science*, pages 151–162, 1975.
- W. B. March, P. Ram, and A. G. Gray. Fast Euclidean Minimum Spanning Tree: Algorithm, Analysis, and Applications. In *Proceedings of KDD'10*, 2010.
- S. Arya and D. M. Mount. A Fast and Simple Algorithm for Computing Approximate Euclidean Minimum Spanning Trees. In *SIAM-SODA*, 2016.

Università
degli Studi
di Padova

Thanks for your attention.

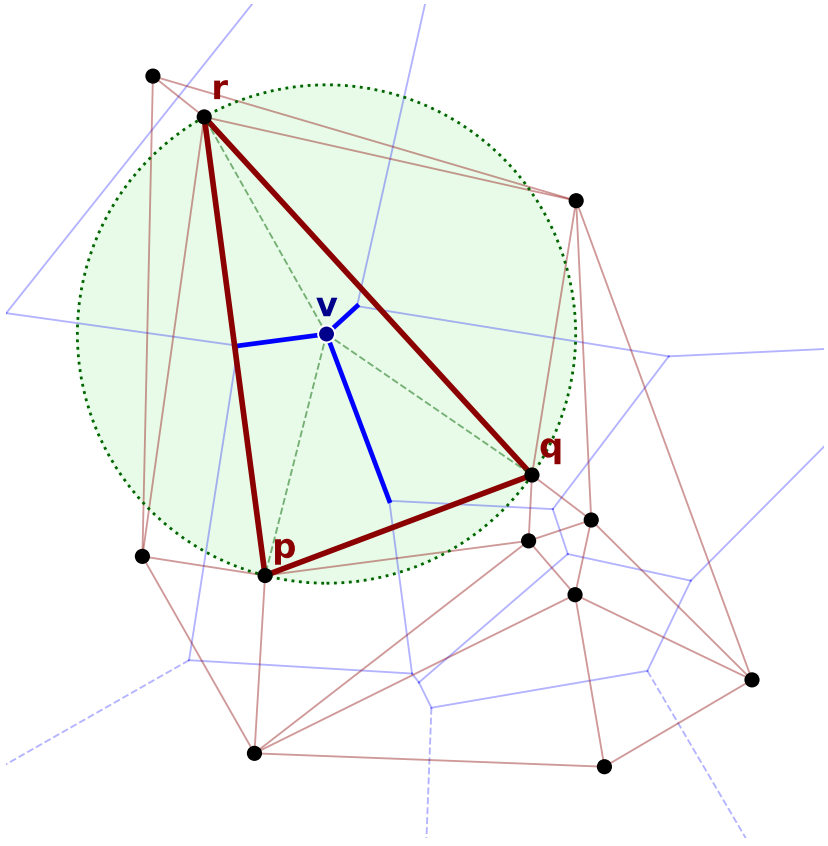
Alessandro Dario

Bertinoro – 3° weekend ortogonale

Appendix: Geometric Duality with Delaunay

Why?

- A Voronoi vertex v is equidistant to three sites p, q, r, making v the **circumcenter** of the triangle Δpqr.
- No other site can be closer to v than p, q, r thus the circumcircle contains no other site, satisfying the **empty circle property**.



Appendix: Naive Algorithm for Voronoi

Given the points $P \subset \mathbb{R}^2$:

1. Consider a point $p \in P$.
2. Calculate the **bisector** with every other point $q \in P$.
3. **Intersect** all **half-planes** on the side of p.
4. Repeat for all other points in P.

Complexity: $O(n^2 \log n)$.

Appendix: Divide-and-conquer Algorithm for Voronoi

Recursive algorithm by *Shamos and Hoey*:

- **Base**: when $n = 2$ the bisector is the only edge.
- **Divide**: split the set P along the x-axis into two subsets L, R with $|L| = |R|$.
- **Recurse**: compute the Voronoi cells of L and R.
- **Conquer**:
  1. determine the boundary line between the two partitions;
  2. merge the two Voronoi diagrams by intersecting each with the boundary line.

Complexity: $O(n \log n)$, proven to be optimal.

Appendix: KD-trees

KD-trees are a data structure for partitioning points $P \subset \mathbb{R}^d$.

- The root contains all points.
- Each node splits its points in half along one dimension.
- The leaves of the tree contain a single point.
- Construction requires $O(d \cdot n \log n)$ operations.

Appendix: Approximate Algorithm

Arya and Mount's algorithm, given points $P \subset \mathbb{R}^d$ and $\varepsilon \in (0, 1)$:

1. Constructs a compressed quad-tree of P.
2. Partitions P into $O(n)$ *well-separated pairs*, i.e., pairs of rectangles $\{(A_1, B_1), \dots\}$ satisfying:
   - Covering: $\forall p, q \in P,\ p \neq q\ \exists!(A_i, B_i)\ s.t.\ p \in A_i, q \in B_i$;
   - Separation: $\forall (A_i, B_i) :\ d(A_i, B_i) \geq 2 \cdot \max\{diam(A_i), diam(B_i)\}$.
3. For each pair (A_i, B_i), only the shortest connecting edge can be part of the MST.
4. Not all pairs $A_i \times B_i$ are considered: quad-tree nodes are no longer expanded when smaller than $\varepsilon \cdot \max\{diam(A_i), diam(B_i)\}$.
5. The MST is solved on this set of $O(n)$ edges.