Computational Geometry

Nearest neighbour decision boundary

Problem definition

- Optimisation problem
- **R**ed points, **B**lue points, together = set **S** of **n** points
- New point **q**

Voronoi condensing

- Not all points are useful for solving the problem, only **k** ≤ **n**
- Create Voronoi diagram, delete points surrounded by cells of the same colour
- O(**n** log **n**) (ex. Fortune's algorithm)

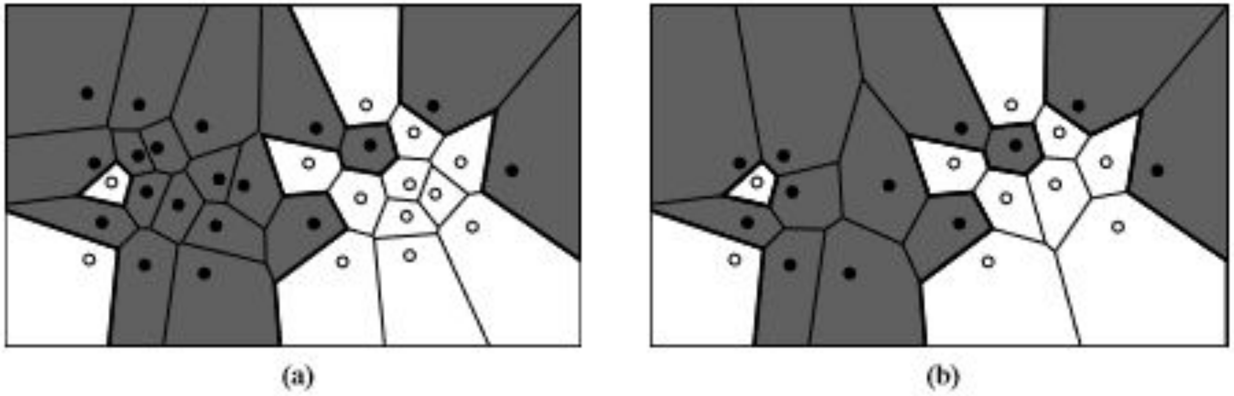


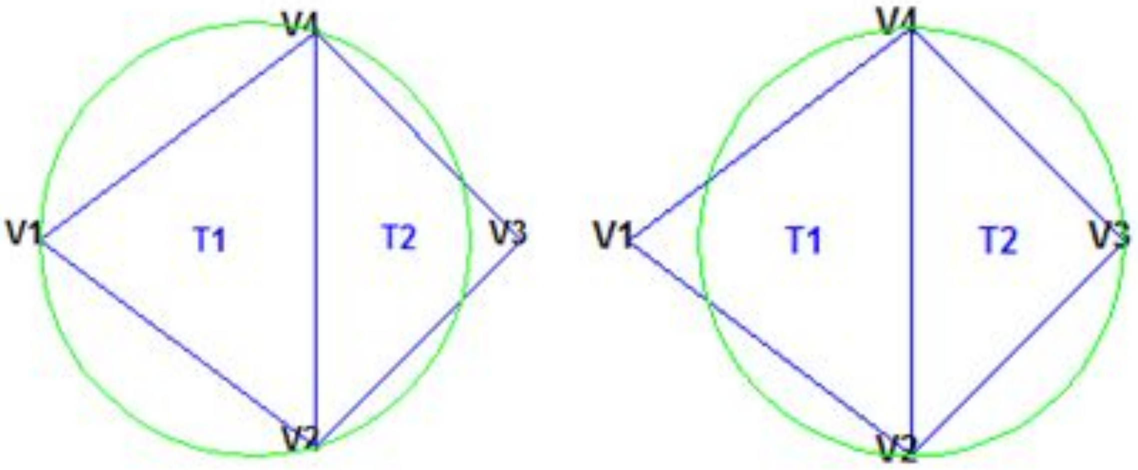
Fig. 1. The Voronoï diagram (a) before Voronoï condensing and (b) after Voronoï condensing. Note that the decision boundary (in bold) is unaffected by Voronoï condensing. Note: In this figure, and all other figures, red points are denoted by white circles and blue points are denoted by black disks.

Our proposition

- O(**n** log **k**) algorithm for Voronoi Condensing
- Output-sensitive algorithm via **k**

Delaunay Triangulations

A **Delaunay triangle** in S is a triangle whose vertices *v1, v2, v3* are in S and such that the circle with *v1*, *v2* and *v3* on its boundary does not contain any point from S in it.



Delaunay triangulations

A **Delaunay triangulation** of S is a partitioning of the convex hull of S into Delaunay triangles.

A **Delaunay edge** is a line segment whose vertices *v1, v2* are in S and such that there exists a circle with *v1* and *v2* on its boundary such that it does not contain any point from S in its interior.

A **bichromatic Delaunay edge** is a Delaunay edge whose two vertices have different colours, i.e. one is red and the other one is blue.

A **bichromatic Delaunay triangle** is a Delaunay triangle whose set of defining vertices contain two vertices of different colours, i.e. there is at least one blue vertex and at least one red vertex.

Delaunay triangulation = unique = contains all Delaunay edges = contains all bichromatic Delaunay edges = solves the problem

Complexity explanations

- $\mathbf{K} \geq \mathbf{k}$
- Find all bichromatic edges in $O((\mathbf{K}^2 + \mathbf{n}) \log \mathbf{K})$
- For $\mathbf{K} \leq \sqrt{\mathbf{n}} \rightarrow O(\mathbf{n} \log \mathbf{K})$
- Ideally $O(\mathbf{n} \log \mathbf{k})$

- $i = 0 \ldots\ldots \log \log \mathbf{n}$: $\mathbf{K} = 2^{2^i}$
- Launch the algorithm with current **K** value : - Finds entire decision boundary

  - Determines $\mathbf{K} < \mathbf{k} \rightarrow$ next value

- If now $\mathbf{K} > \sqrt{\mathbf{n}}$: stop and launch traditional $O(\mathbf{n} \log \mathbf{n})$ algorithm

Total cost

- Algorithm will terminate if **K** > **k** and **K** = 2^{2^i}
- Worst case : log log **k** launches
- One run : O(**n** log 2^{2^i})

Total : $$\sum_{i=0}^{\lceil \log \log k \rceil} O(n \log 2^{2^i}) = \sum_{i=0}^{\lceil \log \log k \rceil} O(n 2^i) = O(n \log k)$$

Pivots

Given a ray, the pivot operations returns the biggest circle whose center is on the ray, has the origin of the ray on its boundary and has no point of the set S in its interior.

Complexity : construction O(**n** log **K**);

query O((**n/K**) log **K**)

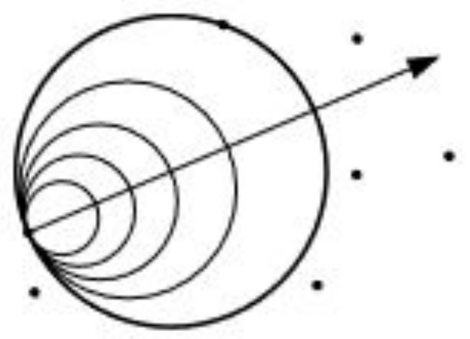


Fig. 3. A pivot operation.

Finding the first edge

Goal : finding 1 correct bichromatic edge

- Link random **R** point **r** to random **B** point **b** . Create ray **RAY**
- First pivot : ray = **RAY,** S = **B**, origin = **r**. Found point = **b’**, found circle = **C**
- Second pivot : ray = from **b’** to center of C, S = **A**, origin = **b’** .Found point = **r’**
- by construction, the two points that we have found are on the boundary of a circle with no other point in its inside, and are thus relevant to the decision boundary

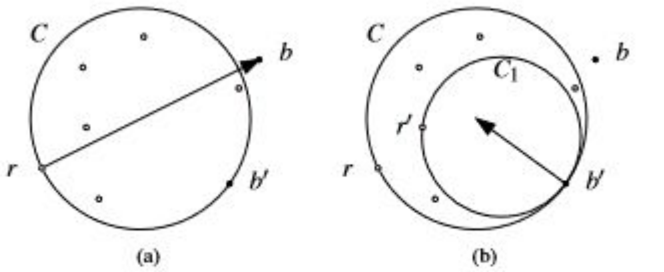


Fig. 4. The (a) first and (b) second pivot used to find a bichromatic edge (r', b').

Augmented triangulation

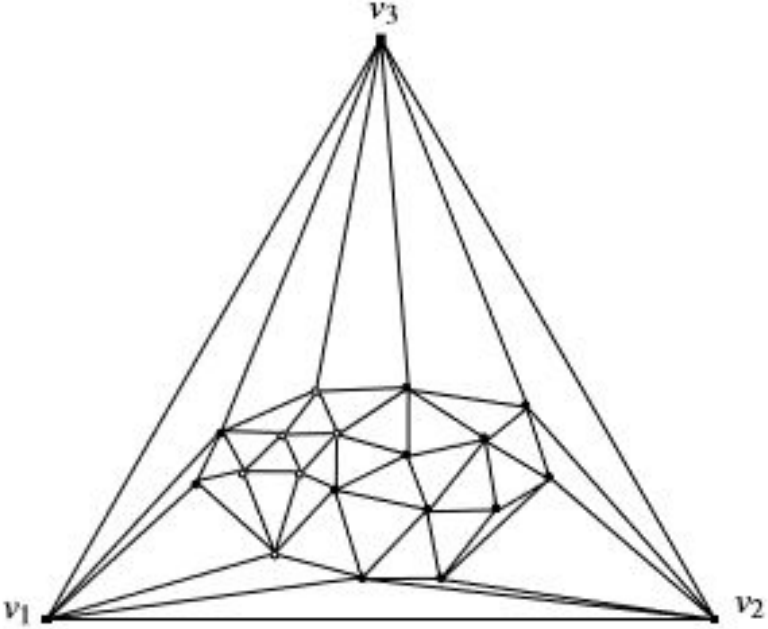


Fig. 5. The augmented Delaunay triangulation of S.

Finding additional points

$P \subseteq Q \subseteq S$, Q = relevant subset, P = currently found subset

For any triangle t, C(t) = the circle with the 3 vertices of t on its boundary

Lemma :

For every triangle t in the augmented Delaunay triangulation of *P*, if t has a blue (resp. red) vertex then *C(t)* does not have a red (resp. blue) point of S in its interior

$\leftrightarrow$ P=Q

Finding additional points

If t contains a blue vertex **b**, pivot in **R** along the ray originating at **b** and passing through C(t).

Result = C(t) → move on

Else : add found point **r** to **Q** and continue

Do the same in **B** along the ray originating at **r**.

Repeat for all triangles.