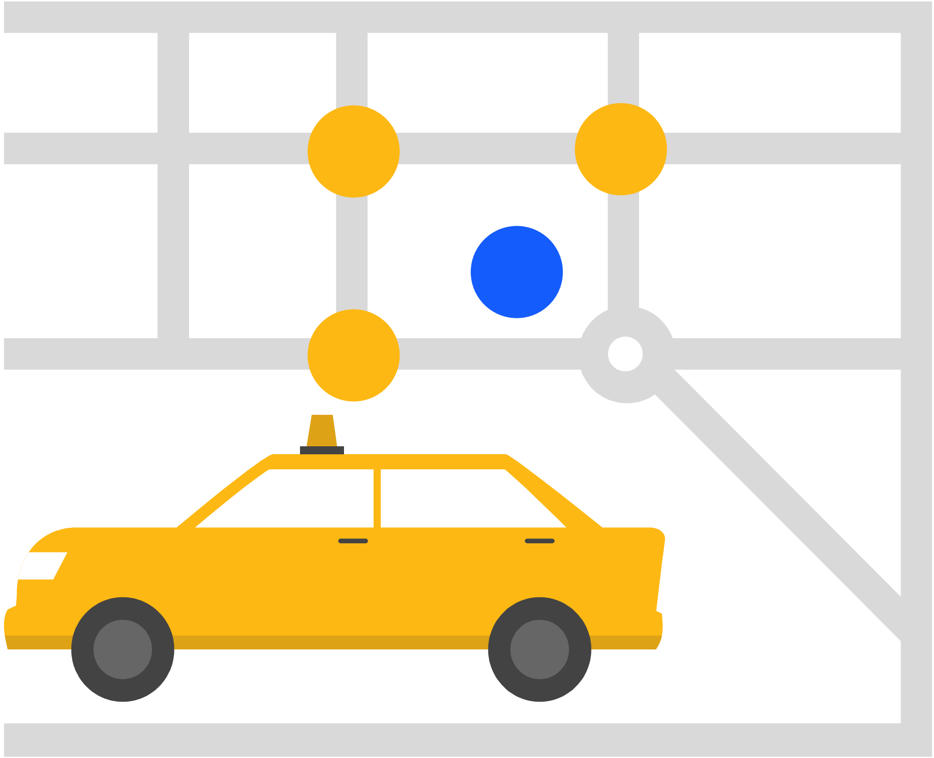
Taxi Trajectory Analysis

EDAA - G04

Diogo Rodrigues
Eduardo Correia
João Sousa

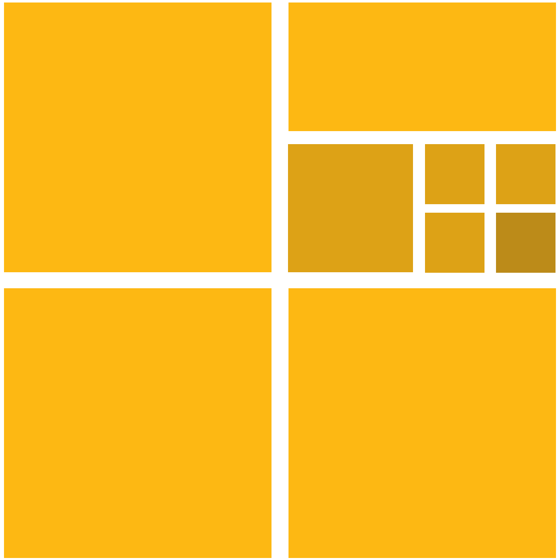
Problem recap

Problem of **mapping GPS coordinates** with **network nodes** that represent the real-life roads.



Quad-trees

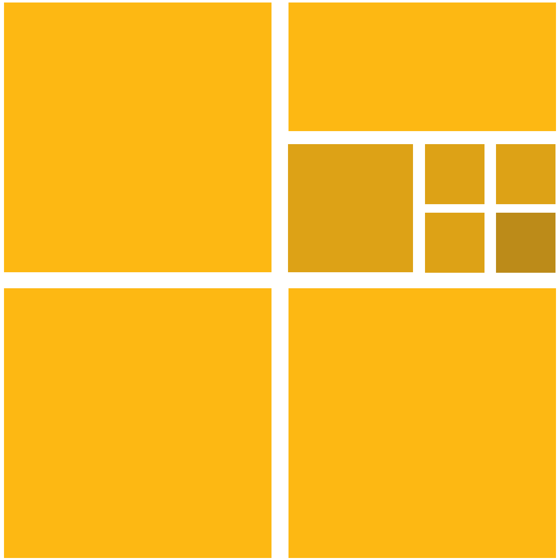
Search runs in $\Theta(\log n)$ with $O(n)$ space complexity.
Calculation of distance of leaves to the goal, determining the appropriate child to execute a search over.
Checks between the distance of the goal to the median, allow pruning of the search space.



Pseudocode

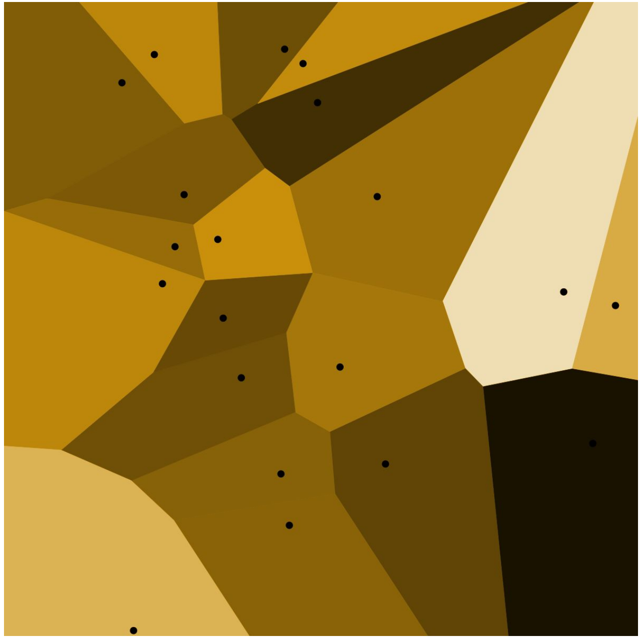
Quad-trees

```
dBest = INF
search(r):
    If(r is leaf):
        d = dist(r.point, p)
        If(d < dBest): dBest = d
    Else:
        v = (xAxisActive ? p.x : p.y)
        child      = (v < median ? r.lchild : r.rchild)
        otherChild = (v < median ? r.rchild : r.lchild)
        search(child)
        If(|v-median| < dBest):
            search(otherChild)
```



Voronoi Diagram Construction

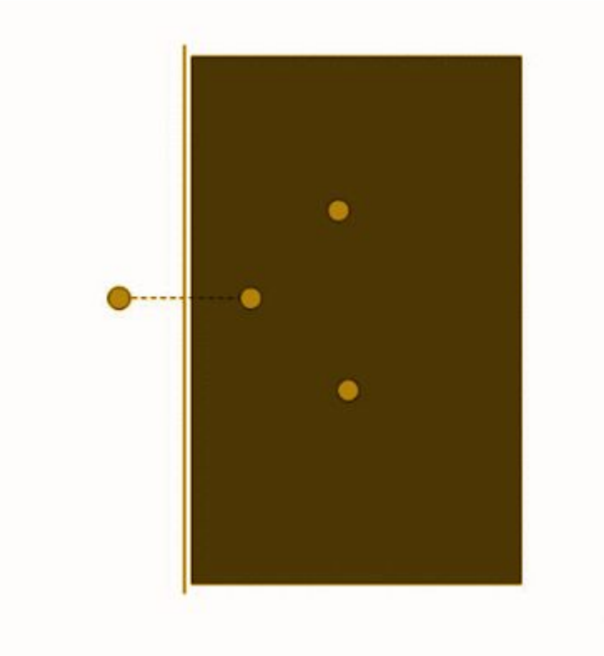
Start: **collection of points** (sites) in a plane.
Goal: **divide the plane in regions**, one per point, where each region (cell) corresponds to the area that is closer to the point of that region than any other point.



Voronoi Diagram Construction

The naïve approach: brute force

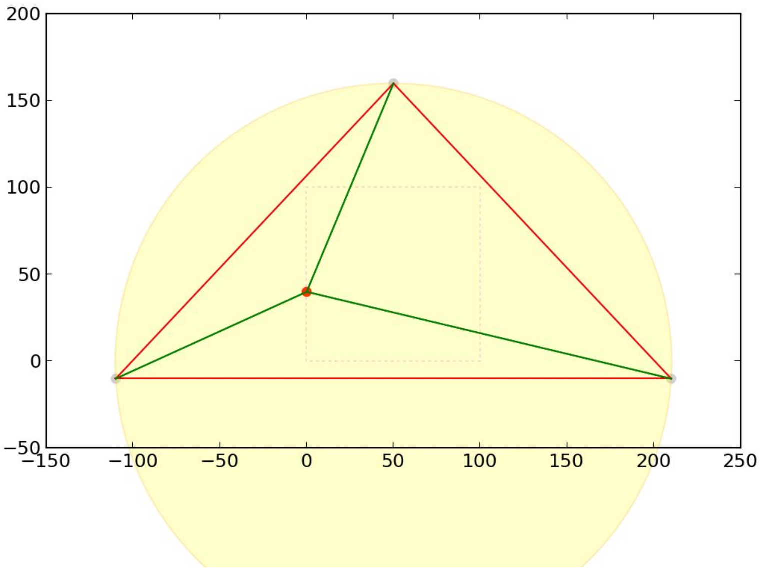
Define Voronoi cells as the intersection of all half planes between 2 sites, i.e., compute 1 cell by intersecting N-1 half planes: ***O(n* log^2 *n)*** time with half-plane intersection.
Therefore, ***O(n^2* log *n)*** time in total.



Voronoi Diagram Construction

An incremental approach: Bowyer-Watson

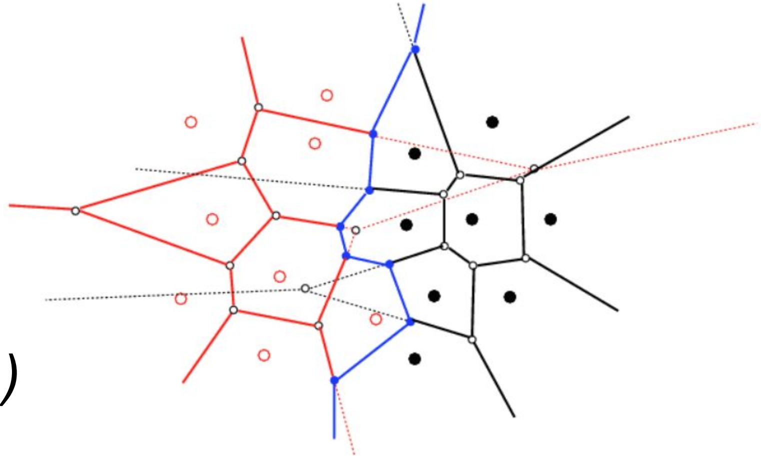
Method for computing **Delaunay triangulation**.
Incremental algorithm, adding points, one at a time.
Delete triangles whose circumcircles contain the new point.
Up to ***O(n²)*** operations needed.



Voronoi Diagram Construction

Divide and conquer approach (Shamos and Hoey)

- **Split** the set of sites **in two** with around the same size (left and right) using a dividing line.
- Construct diagrams **recursively**.
- **Merge** voronoi diagrams.



Recurrence relation: $T(n) = 2T(n/2) + O(n)$
Time complexity: $\boldsymbol{O(n \log n)}$

Voronoi Diagram Construction

Sweep line approach

Fortune's algorithm is a sweep line algorithm for generating a **Voronoi diagram** from a set of points in a plane using ***O(n* log *n)*** time and ***O(n)*** space.

Fortune's Algorithm

Concepts

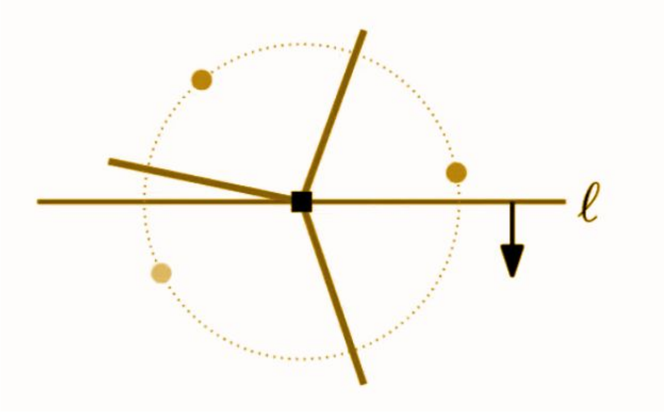
Beach line: piecewise curve, an amalgamation of parabolas.

Site event: sweep line reaches a new site, inserting a new parabolic arc into the beach line.

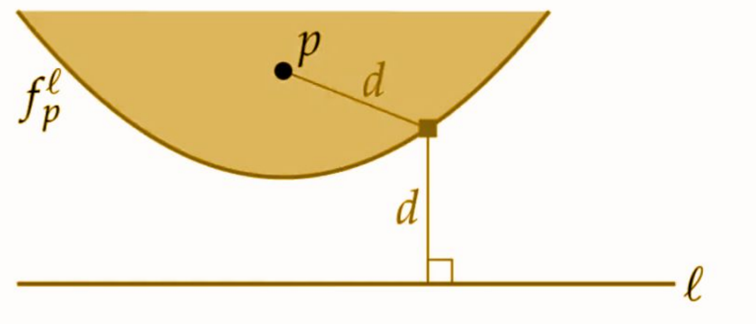
Circle / voronoi vertex event: a new vertex is created as a result of an arc from the beach line which length shrunk to 0.

Fortune’s Algorithm

Risk: backtracking when the sweep line reaches the next vertex.



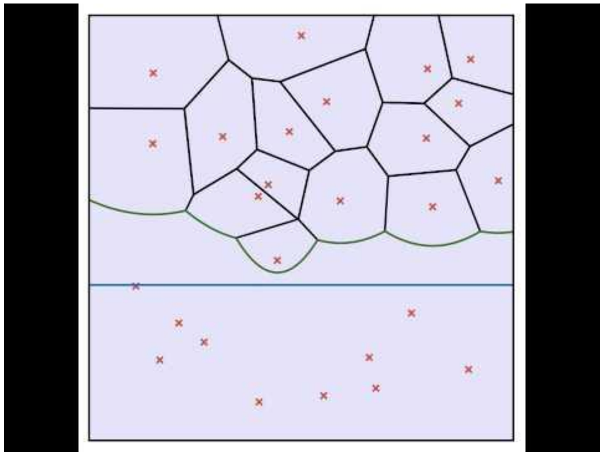
What area above the sweep line is already **fixed**?



Fortune’s Algorithm

The **beach line** is obtained by considering the lower envelope of all parabolas.

Regions **above** it will not suffer any changes with the appearance of new arcs.



Pseudocode

Fortune's algorithm

```
Fill the event queue with site events for each input site.
While the event queue still has items in it:
    If the next event on the queue is a site event:
        Add the new site to the beachline
    Otherwise it must be an edge-intersection event:
        Remove the squeezed cell from the beachline
Cleanup any remaining intermediate state
```

Slab decomposition

Only 2 binary searches are needed: one on each axis.

Allows **point location** in ***O*(log *n*)** while requiring $\boldsymbol{O(n^2)}$ or ***O*(*n*)** space if maintaining the segments that intersect in a persistent red-black tree.

Q&A