Voronoi Diagrams
The Post Office Problem

Min-Te Sun, Ph.D.

Service Area of Each Post Office

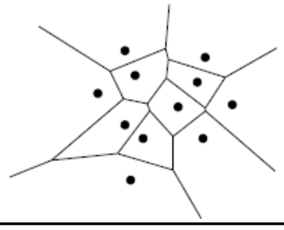
- Assumptions
  - The price of a particular good or service is the same at every site;
  - The cost of acquiring the good or service is equal to the price plus the cost of transportation to the site;
  - The cost of transportation to a site equals the Euclidean distance to the site times a fixed price per unit distance;
  - Consumers try to minimize the cost of acquiring the good or service.

Voronoi Assignment Model

- The model where every point is assigned to the nearest site is called the *Voronoi assignment model*.
- The subdivision induced by this model is called the *Voronoi diagram* of the set of sites.
- Vast applications - physics, astronomy, robotics, and other geometry structures (e.g., Delaunay Triangulation in Chap 9)

Definition of Voronoi Diagram

- Let $P := \{p1, p2, . . . , pn\}$ be a set of n distinct points in the plane (i.e., sites). We define the Voronoi diagram of P as the subdivision of the plane into n cells, one for each site in P, with the property that a point q lies in the cell corresponding to a site p_i if and only if $\text{dist}(q, p_i) < \text{dist}(q, p_j)$ for each $p_j \in P$ with $j \neq i$.
  - We denote the Voronoi diagram of P by Vor(P).
  - The cell of Vor(P) that corresponds to a site p_i is denoted V(p_i).



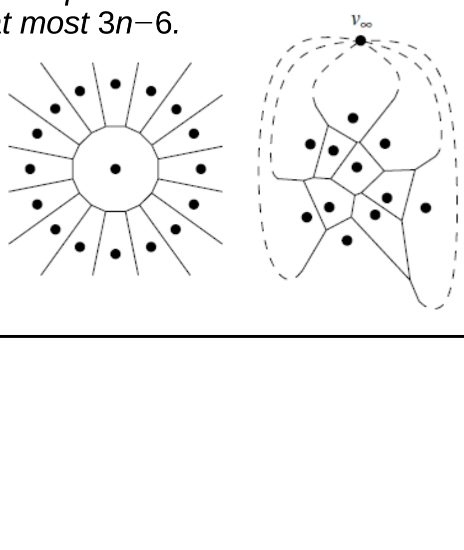
The Case of Two Sites and Its Generalization

- For two points *p* and *q* in the plane we define the *bisector of p and q* as the perpendicular bisector of the line segment *pq*.
  - This bisector splits the plane into two half-planes.
  - We denote the open half-plane that contains *p* by $h(p,q)$ and the open half-plane that contains *q* by $h(q, p)$.
  - $r \in h(p,q)$ if and only if $\text{dist}(r, p) < \text{dist}(r,q)$.
- We observe that $\mathcal{V}(p_i) = \bigcap_{1 \leqslant j \leqslant n, j \neq i} h(p_i, p_j)$.

Vertices and Edges of Vor(P)

- *Let P be a set of n point sites in the plane. If all the sites are collinear then* Vor(*P*) *consists of* $n-1$ *parallel lines. Otherwise,* Vor(*P*) *is connected and its edges are either segments or half-lines.*
- *For n* $\geqq$ 3*, the number of vertices in the Voronoi diagram of a set of n point sites in the plane is at most* $2n-5$ *and the number of edges is at most* $3n-6$*.*
  - $m_v - m_e + m_f = 2$

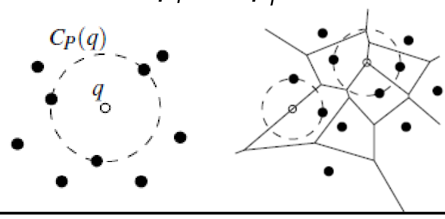
  => $(n_v+1) - n_e + n = 2$
  - Deg: $2n_e \geqq 3(n_v+1)$



C_p(q) and Vertex/Edge

- *For the Voronoi diagram* Vor(*P*) *of a set of points P the following holds:*

1. *A point q is a vertex of* Vor(*P*) *if and only if its largest empty circle $C_P(q)$ contains three or more sites on its boundary.*
2. *The bisector between sites p_i and p_j defines an edge of* Vor(*P*) *if and only if there is a point q on the bisector such that $C_P(q)$ contains both p_i and p_j on its boundary but no other site.*

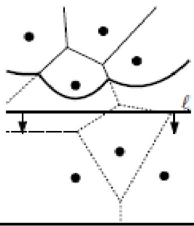


Computing Voronoi Diagram

- Brute force: Computing the common intersection of half planes $h(p_i, p_j)$, with $j \neq i$
  - $O(n^2 \log n)$ complexity => not good enough!
- Fortune's algorithm
  - $\Omega(n \log n)$
  - This algorithm is optimal since sorting n real numbers is reducible to the problem of computing Voronoi diagram
  - A "variation" of plane sweep

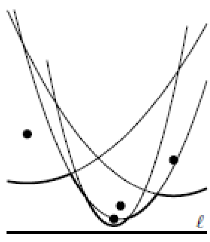
“Beach Line”

- Regular sweep line does not work because the incoming point will affect part of the diagram above the line
- Beach line is the boundary of the Voronoi diagram that will not change after the sweep line moves down



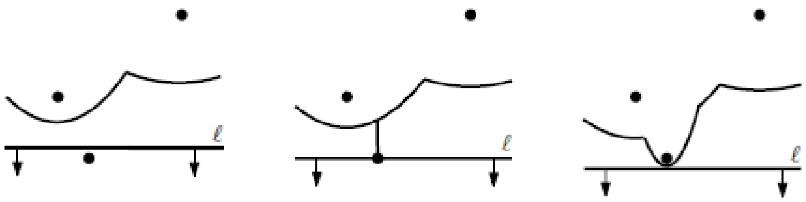
Definition of beach line

- The part of the Voronoi diagram above that cannot be changed anymore if points $q \in l^+$ to any site below is greater than the distance of q to l itself.
  - The nearest site of q cannot lie below if q is at least as near to some site $p_i \in l^+$ as q is to l.
  - The locus of points that are closer to some site $p_i \in l^+$ than to l is bounded by a parabola.
  - *The beach line is x-monotone, that is, every vertical line intersects it in exactly one point.*



Maintain Beach Line 1/2

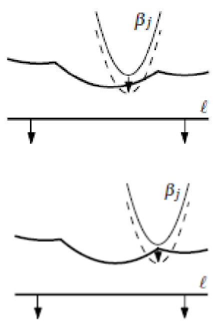
- Where a new arc appears on the beach line?
  - When the sweep line reaches a new site – called “site event”
  - *The only way in which a new arc can appear on the beach line is through a site event.*

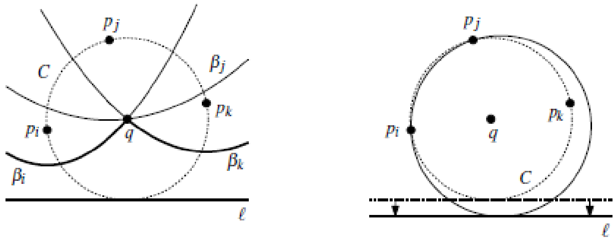


Only Through Site Event Will a New Arc Appear!

1. Solving quadratic formulas and we will find that there will never be a single solution => a single tangent point is never possible.
2. There will be a circle passing through the exact point where new arc breaks through => impossible to happen!

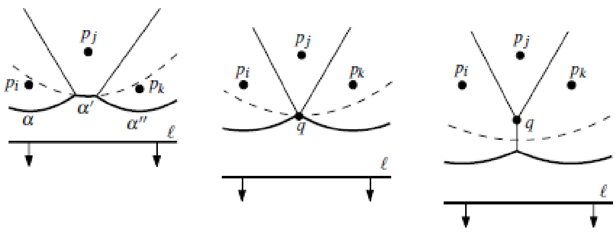
- This fact suggests that # of arcs $\leqq$ 2n - 1





When Will Arc Disappear?

- The 2nd type of event in line sweep is when arc disappear => called circle event
- *The only way in which an existing arc can disappear from the beach line is through a circle event.*



Fortune’s Algorithm

Algorithm VORONOIDIAGRAM(P)
Input. A set $P := \{p_1, \ldots, p_n\}$ of point sites in the plane.
Output. The Voronoi diagram Vor(P) given inside a bounding box in a doubly-connected edge list $\mathcal{D}$.

1. Initialize the event queue $\mathcal{Q}$ with all site events, initialize an empty status structure $\mathcal{T}$ and an empty doubly-connected edge list $\mathcal{D}$.
2. **while** $\mathcal{Q}$ is not empty
3.   **do** Remove the event with largest y-coordinate from $\mathcal{Q}$.
4.   **if** the event is a site event, occurring at site p_i
5.   **then** HANDLESITEEVENT(p_i)
6.   **else** HANDLECIRCLEEVENT(γ), where γ is the leaf of $\mathcal{T}$ representing the arc that will disappear
7. The internal nodes still present in $\mathcal{T}$ correspond to the half-infinite edges of the Voronoi diagram. Compute a bounding box that contains all vertices of the Voronoi diagram in its interior, and attach the half-infinite edges to the bounding box by updating the doubly-connected edge list appropriately.
8. Traverse the half-edges of the doubly-connected edge list to add the cell records and the pointers to and from them.

Handle Site Event

HANDLESITEEVENT(p_i)

1. If $\mathcal{T}$ is empty, insert p_i into it (so that $\mathcal{T}$ consists of a single leaf storing p_i) and return. Otherwise, continue with steps 2– 5.
2. Search in $\mathcal{T}$ for the arc α vertically above p_i. If the leaf representing α has a pointer to a circle event in $\mathcal{Q}$, then this circle event is a false alarm and it must be deleted from $\mathcal{Q}$.
3. Replace the leaf of $\mathcal{T}$ that represents α with a subtree having three leaves. The middle leaf stores the new site p_i and the other two leaves store the site p_j that was originally stored with α. Store the tuples $\langle p_j, p_i \rangle$ and $\langle p_i, p_j \rangle$ representing the new breakpoints at the two new internal nodes. Perform rebalancing operations on $\mathcal{T}$ if necessary.
4. Create new half-edge records in the Voronoi diagram structure for the edge separating $\mathcal{V}(p_i)$ and $\mathcal{V}(p_j)$, which will be traced out by the two new breakpoints.
5. Check the triple of consecutive arcs where the new arc for p_i is the left arc to see if the breakpoints converge. If so, insert the circle event into $\mathcal{Q}$ and add pointers between the node in $\mathcal{T}$ and the node in $\mathcal{Q}$. Do the same for the triple where the new arc is the right arc.

Handle Circle Event

HANDLECIRCLEEVENT(γ)

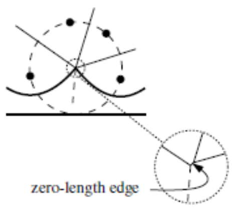
1. Delete the leaf γ that represents the disappearing arc α from $\mathcal{T}$. Update the tuples representing the breakpoints at the internal nodes. Perform rebalancing operations on $\mathcal{T}$ if necessary. Delete all circle events involving α from $\mathcal{Q}$; these can be found using the pointers from the predecessor and the successor of γ in $\mathcal{T}$. (The circle event where α is the middle arc is currently being handled, and has already been deleted from $\mathcal{Q}$.)
2. Add the center of the circle causing the event as a vertex record to the doubly-connected edge list $\mathcal{D}$ storing the Voronoi diagram under construction. Create two half-edge records corresponding to the new breakpoint of the beach line. Set the pointers between them appropriately. Attach the three new records to the half-edge records that end at the vertex.
3. Check the new triple of consecutive arcs that has the former left neighbor of α as its middle arc to see if the two breakpoints of the triple converge. If so, insert the corresponding circle event into $\mathcal{Q}$. and set pointers between the new circle event in $\mathcal{Q}$ and the corresponding leaf of $\mathcal{T}$. Do the same for the triple where the former right neighbor is the middle arc.

What Data Structure to Maintain?

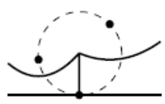
- Voronoi diagram is stored as a subdivision with a large bounding box
- The beach line is represented by a balanced binary search tree T
- The event queue Q is implemented as a priority queue, where the priority of an event is its *y*-coordinate

Degenerate Cases

1. If more than 3 points are on the same circle?
   - The algorithm just deals with 3-point circle event so multiple circle events will be created, which result in points at the same location in the Voronoi diagram and "zero length edge" between them
   - Such case can be removed at later time



2. If a site p_i happens to be located exactly below the breakpoint between two arcs on the beach line?
   - the algorithm splits either of two arcs and inserts the arc for p_i in between the two pieces, one of which has zero length. This piece of zero length now is the middle arc of a triple that defines a circle event.

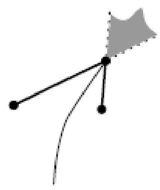


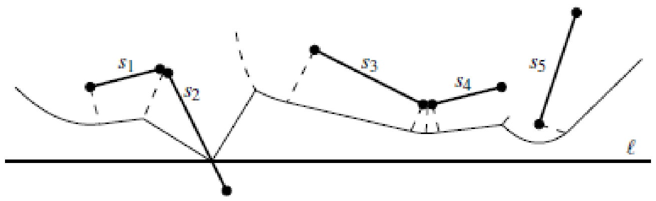
Complexity of Fortune's Algorithm

- *The Voronoi diagram of a set of n point sites in the plane can be computed with a sweep line algorithm in O*(*n*log*n*) *time using O*(*n*) *storage*

Voronoi Diagram for a Set of Points and Line Segments

- The beach line now may contain line segments
- Note that we do not deal with line segments that share the same endpoint => avoid ambiguity





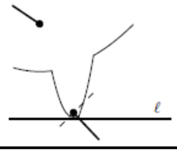
Five Situations New Arc Appear

1. If a point *p* is closest to two site endpoints while being equidistant from them and , then *p* is a breakpoint that traces a line segment
2. If a point *p* is closest to two site interiors while being equidistant from them and , then *p* is a breakpoint that traces a line segment.
3. If a point *p* is closest to a site endpoint and a site interior of different sites while being equidistant from them and , then *p* is a breakpoint that traces a parabolic arc.
4. If a point *p* is closest to a site endpoint, the shortest distance is realized by a segment that is perpendicular to the line segment site, and *p* has the same distance from , then *p* is a breakpoint that traces a line segment.
5. If a site interior intersects the sweep line, then the intersection is a breakpoint that traces a line segment (the site interior).

- Note that in case 4 and 5, the breakpoint does not actually trace an arc of the Voronoi diagram because only one site is involved.

Upper and Lower Endpoint

- Site events at upper endpoints should be handled differently from site events at lower endpoints
  - At an upper endpoint, an arc of the beach line is split into two
  - In between, four new arcs appear. The breakpoints between these four arcs are of the last two types.
  - At a lower endpoint, the breakpoint that is the intersection of the site interior with the sweep line is replaced by two breakpoints of the fourth type, with a parabolic arc in between

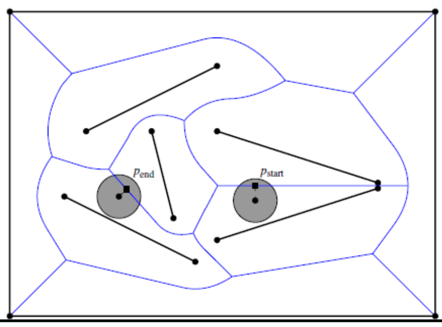


How About Algorithm?

- The algorithm remains the same, except that more types of site events and circle events need to be handled.
- Each parabola in Voronoi diagram will just be treated the same as an “edge” in the algorithm (as long as we know its endpoints and corresponding faces.
- The algorithm will still have the same time and storage complexity

Another Application of Voronoi Diagram for Line Segments)

- Robot motion planning - *retraction*
  - A number of walls modeled as line segments
  - The arcs of the Voronoi diagram define the middle between the line segments, and therefore define a path with the most clearance.



Retraction Algorithm

Algorithm RETRACTION(S, q_{start}, q_{end}, r)

Input. A set $S := \{s_1, \dots, s_n\}$ of disjoint line segments in the plane, and two discs D_{start} and D_{end} centered at q_{start} and q_{end} with radius r. The two disc positions do not intersect any line segment of S.

Output. A path that connects q_{start} to q_{end} such that no disc of radius r with its center on the path intersects any line segment of S. If no such path exists, this is reported.

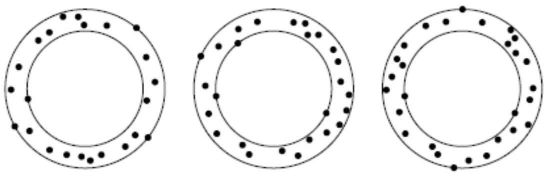
1. Compute the Voronoi diagram $\text{Vor}(S)$ of S inside a sufficiently large bounding box.
2. Locate the cells of $\text{Vor}(P)$ that contain q_{start} and q_{end}.
3. Determine the point p_{start} on $\text{Vor}(S)$ by moving q_{start} away from the nearest line segment in S. Similarly, determine the point p_{end} on $\text{Vor}(S)$ by moving q_{end} away from the nearest line segment in S. Add p_{start} and p_{end} as vertices to $\text{Vor}(S)$, splitting the arcs on which they lie into two.
4. Let $\mathcal{G}$ be the graph corresponding to the vertices and edges of the Voronoi diagram. Remove all edges from $\mathcal{G}$ for which the smallest distance to the nearest sites is smaller than or equal to r.
5. Determine with depth-first search whether a path exists from p_{start} to p_{end} in $\mathcal{G}$. If so, report the line segment from q_{start} to p_{start}, the path in $\mathcal{G}$ from p_{start} to p_{end}, and the line segment from p_{end} to q_{end} as the path. Otherwise, report that no path exists.

Robot Motion Planning

- *Given n disjoint line segment obstacles and a disc-shaped robot, the existence of a collision-free path between two positions of the robot can be determined in O(*$n$log$n$*) time using O(*n*) storage.*

Coordinate Measurement

- Measure "Roundness"
  - sample points on the surface of the object
  - The *roundness* of a set of points P is defined as the width of the smallest-width annulus that contains the points
  - Three cases



Farthest-Point Voronoi Diagrams

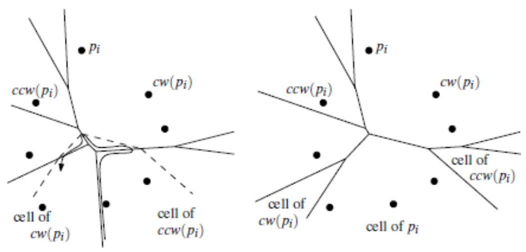
- Finding the smallest-width annulus is equivalent to finding its center point.
  - Once the center point *q* is fixed, the annulus is determined by the points of *P* that are closest to and farthest from *q*.
- If we have the Voronoi diagram of *P*, then the closest point is the one in whose cell *q* lies.
- A similar structure exists for the farthest point, namely the *farthest-point Voronoi diagram*.
  - The intersection of the "other half planes" forms the farthest-point Voronoi cell for a point

Important Observations

- *Given a set P of points in the plane, a point of P has a cell in the farthest-point Voronoi diagram if and only if it is a vertex of the convex hull of P.*
  - *Not all points have cells in farthest-point Voronoi diagram!*
  - The vertices and edges of the farthest-point Voronoi diagram form a tree-like structure, because the diagram is connected and does not have cycles.
    - A cycle would imply a bounded cell.
  - the farthest-point Voronoi diagram of n points has $O(n)$ vertices, edges, and cells.

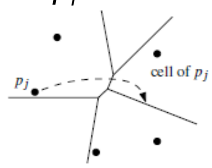
Incremental Algorithm for Farthest-Point Voronoi Diagram

- We first compute the convex hull of P, take its vertices, and put them in random order. Let this random order be $p_1, \ldots, p_h$.
  - We remove the points $p_h, \ldots, p_4$ one by one from the cyclic order, and when removing p_i, store its clockwise neighbor $cw(p_i)$ and counterclockwise neighbor $ccw(p_i)$



Incremental Algorithm (cont.)

- Given the farthest-point Voronoi diagram of $\{p_1, \ldots, p_{i-1}\}$, we maintain a pointer for each point p_j, $1 \leq j < i$, to the half-infinite half-edge of the doubly-connected edge list that is most counterclockwise in a traversal of the boundary of the farthest-point Voronoi cell of p_j.



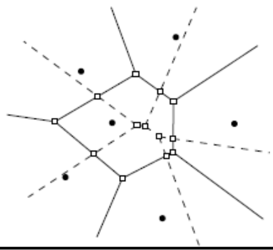
- Then we add p_i one by one
- The cell of p_i will come "in between" the cells of $cw(p_i)$ and $ccw(p_i)$.
  - Just before p_i is added, $cw(p_i)$ and $ccw(p_i)$ are each other's neighbors on the convex hull of $\{p_1, \ldots, p_{i-1}\}$, so their cells are separated by a half-infinite edge that is part of their bisector.
    - The point $ccw(p_i)$ has a pointer to this edge.

Incremental Algorithm (cont.)

- The bisector of *pi* and $ccw(p_i)$ will give a new half-infinite edge that lies in the farthest-point Voronoi cell of $ccw(p_i)$, and is part of the boundary of the farthest-point Voronoi cell of p_i.
- We traverse the cell of $ccw(p_i)$ in the clockwise direction to see which edge the bisector intersects. On the other side of this edge is the farthest-point Voronoi cell of another point p_j from $\{p_1, \ldots, p_{i-1}\}$, and the bisector of p_j and p_i will also give an edge of the farthest-point Voronoi cell of p_i.
- We again traverse the cell of p_j in the clockwise direction to determine where the other insertion of the cell boundary and the bisector is located.
- By tracing cell boundaries in clockwise order, we trace the farthest-point Voronoi cell in counterclockwise order.
- *Given a set of n points in the plane, its farthest-point Voronoi diagram can be computed in* $O(n\log n)$ *expected time using* $O(n)$ *storage*

Smallest-Width Annulus

- We generate the vertices of the *overlay* of the Voronoi diagram and the farthest-point Voronoi diagram.
- The vertices of the overlay are exactly the candidate centers of the smallest-width annulus, covering all three cases
- *Given a set P of n points in the plane, the smallest-width annulus (and the roundness) can be determined in* $O(n^2)$ *time using* $O(n)$ *storage.*



Homework Assignment 7

Page 170

- 7.1
- 7.5