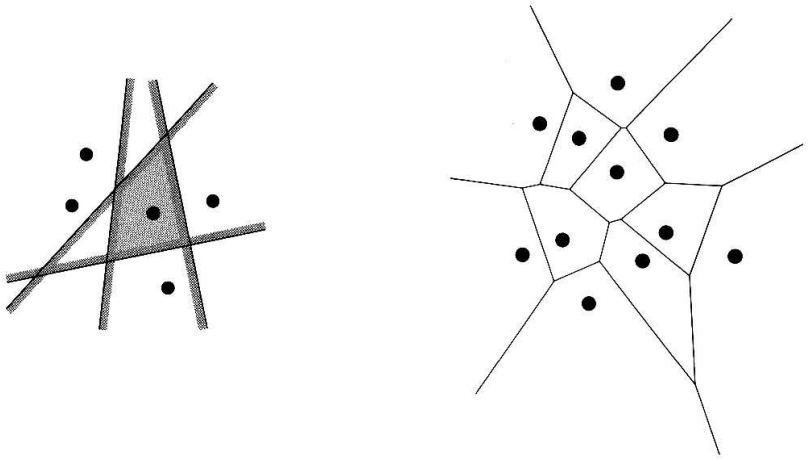


Figure 1: Left: constructing a Voronoi cell by intersecting half-planes. The shaded cell is the intersection of all half planes generated from the perpendicular bisectors of the point and the other points in the plane. Right: A Voronoi Diagram for a set of points.

CS 4733, Class Notes - Voronoi Path Planning

1 Voronoi Diagrams

- Visibility graph path planning has the problem that we navigate along the vertices of grown polygons. If we are not careful, we can come very close to actual obstacles if our growing is incorrect, or more realistically, mobile robot control has some errors in it.
- A better idea may be to try to find paths that are not close to obstacles, but in fact as far away as possible from obstacles. This will create a maximal safe path, in that we never come closer to obstacles than we need.
- To do this, we will look at the *Voronoi Diagram* in the plane. Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of points in the plane called $sites$. The Voronoi diagram is the sub-division of the plane into n distinct cells, one for each site. Each cell has the property that a point q corresponds to a site p_i iff $dist(q, p_i) < dist(q, p_j)$ for each $p_j \in P$ with $j \neq i$
- A Voronoi diagram can be used to find out the trading area of a region . If you are planning to open a new Home Depot store, you want to know what region it will serve, and how this will relate to exisitng Home Depot's in the region. Assuming people go to the nearest Home Depot, you can create a Voronoi diagram which shows how a region will be subdivided by adding more stores in the region (note: this assumes a Euclidean distance metric which may not be applicable with travel by car on existing roads).
- How do we compute such a diagram? The straightforward method (see fig. 1) is to find the perpendicular bisector between a site p and a site q. This separates the plane into two half-

planes, one with points closer to p and one with points closer to q. If we intersect all the half planes that separate a site p_i from the other sites p_j, we can find the Voronoi cell determined by p_i. Doing this for all the points creates the Voronoi diagram.

- The above brute force procedure can be sped up appreciably, and there are many known algorithms to create a Voronoi Diagram, with a lower bound of $O(n\ log\ n)$.

2 Path Planning Using Voronoi Diagrams

- Assuming we have point-based obstacles and a point robot, we can use the Voronoi diagram to navigate. You can think of the Voronoi diagram as a Voronoi graph, made up of edges and vertices. To go from a point P_{start} to a point P_{goal}, we simply find the nearest points on the Voronoi graph to $(P_{start}\ ,\ P_{goal})$: $(P^*start\ ,\ P^*_{goal})$. Wethen use a standard graph search type algorithm (e.g Dijkstra's algorithm) to traverse the vertices and edges of the graph from $P^*start$to P^*_{goal}.

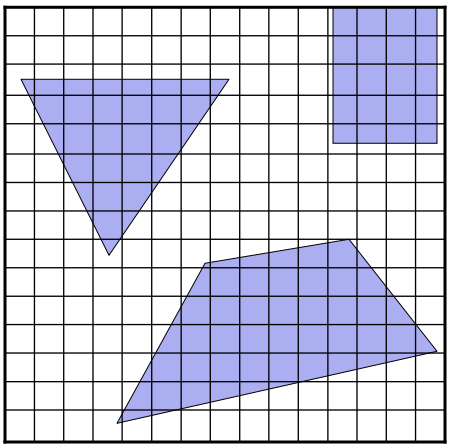
- Suppose the robot isn't a point and the obstacles aren't a point? The two-dimensional region in which the robot moves will contain buildings and other types of barriers, each of which can be represented by a convex or concave polygonal obstacle. To find the generalized Voronoi diagram for this collection of polygons, we can use an approximation based on the simpler problem of computing the Voronoi diagram for a set of discrete points. We estimate the polygonal obstacle boundaries by discrete points.

  Method:

  1. Approximate the boundaries of the polygonal obstacles with the large number of points that result from subdividing each side of the original polygon into small segments.
  2. Compute the Voronoi diagram for this collection of approximating points.
  3. Once this complicated Voronoi diagram is constructed, eliminate those Voronoi edges which have one or both endpoints lying inside any of the obstacles.
  4. The remaining Voronoi edges form a good approximation of the generalized Voronoi diagram for the original obstacles in the map.

- To take into account a robot which isn't point size, we need to find critical points and critical lines in the Voronoi diagram. These are places where the Voronoi path has a local minima. At these points we can see if the robot's diameter will fit through the space at the critical point: is the diameter greater than the critical line length. Note that this assumes a fixed robot orientation, or we can use the maximum diameter of the robot over all rotations.

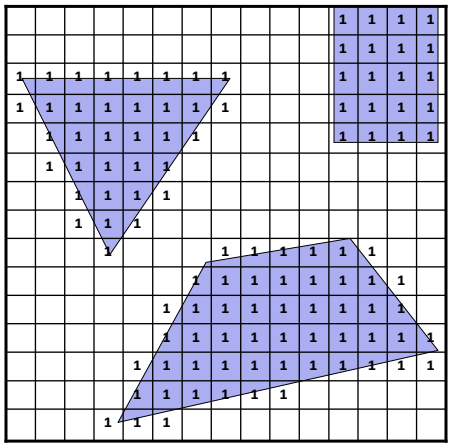
Practical Algorithm 2: Environment Model



Voronoi Diagrams: Practical Algorithm 2

Brushfire GVG Generation

- Start with an empty grid with obstacles = 1



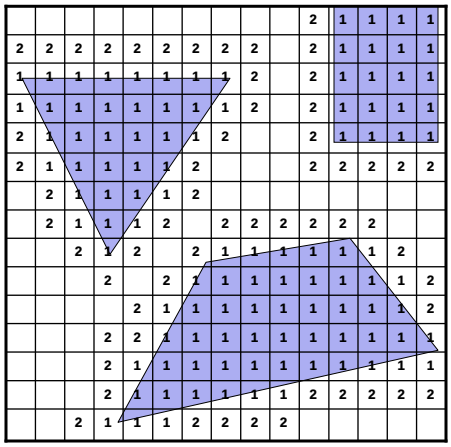
Voronoi Diagrams: Practical Algorithm 2

Brushfire GVG Generation

- "Wavefront"

- Start with an empty grid with obstacles = 1
- Expand all cells (i) to (i+1)

- If a cell is expanded twice, label as a GVG edge and do not expand further.



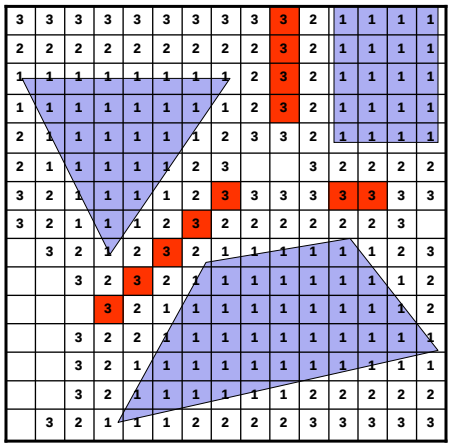
Voronoi Diagrams: Practical Algorithm 2

Brushfire GVG Generation

- "Wavefront"

- Start with an empty grid with obstacles = 1
- Expand all cells (i) to (i+1)

- If a cell is expanded twice, label as a GVG edge and do not expand further.



Voronoi Diagrams: Practical Algorithm 2

Brushfire GVG Generation

- "Wavefront"

- Start with an empty grid with obstacles = 1
- Expand all cells (i) to (i+1)

- If a cell is expanded twice, label as a GVG edge.