6.850 Project Report
David Xiong

This project is focused on using Voronoi diagrams to determine a most isolated point in Google maps. There are several applications for using this program. This isolated point can be used to determine the best location to place a new establishment. Many establishments benefit from being further away to similar establishments. For example, fast food restaurant chains benefit from being spread out from similar restaurants to prevent competition. Also, public services such as police stations and fire departments should be spread out to have quick access to all areas. In a similar vein, the isolated point is also the one which would take the longest to receive help, in case of emergencies.

Given an input of several sites, the Voronoi diagram splits the plane up into several different areas, based on which points are closest to each site. The boundaries of these areas are formed by the perpendicular bisectors between two sites. One interesting property of the Voronoi diagram is the circle event: Voronoi vertices are formed when three or more perpendicular bisectors meet, and the vertex is the circumcircle of three or more sites. Furthermore, if the circumcircle is drawn, there are no other sites within the area of the circle. This means that the sites the circumcircle passes through are the sites that are closest to the vertex, and are also all equidistant from the vertex at a distance of the radius length. The most isolated point on a plane can be described as the point whose nearest site is the furthest away. In other words, we want to find the maximum distance of the minimum distance between a point and a site. Due to the circle event, this point must be one of the vertices of the Voronoi diagram.

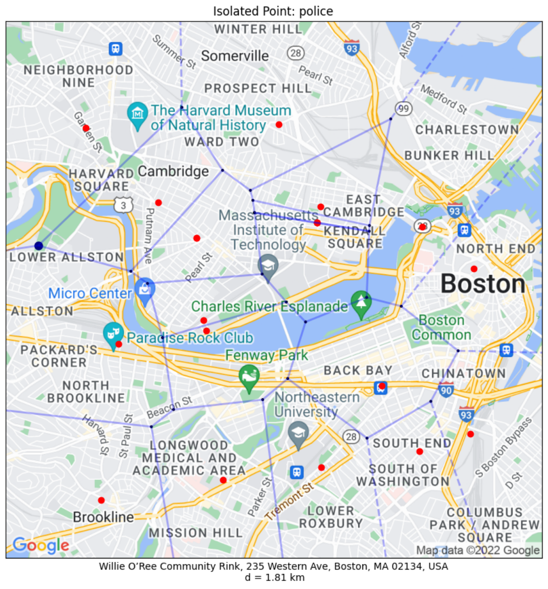
The basic procedure of the algorithm is outlined here: Google maps API is used to extract location data from the area of interest. This includes a static image of the map, as well as the coordinates of the locations of interest. These coordinates will be used as the coordinates of the sites of the Voronoi diagram. Python’s Scipy will be used to create the Voronoi diagram, and Matplotlib is used to visualize the map and diagram. The Voronoi vertices within the bounding box will be looped through to determine the vertex which has the maximal distance to any site. The address of this isolated point is determined, as well as the distance to the nearest site, and the point is drawn on the map.

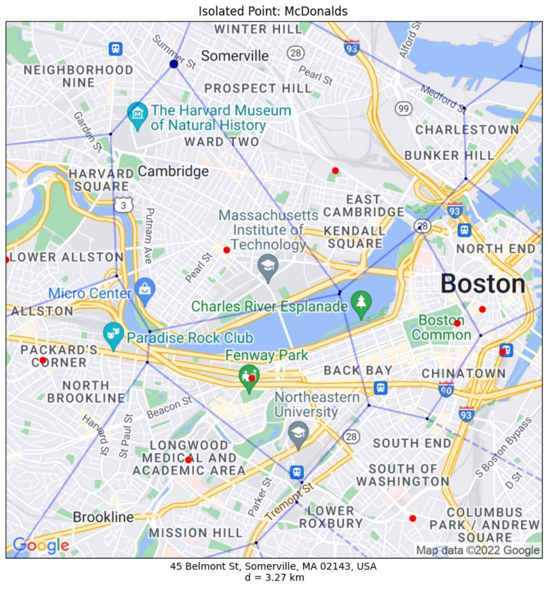
Three parts of the Google Maps API are used. These are the Maps Static API, Places API, and Geocoding API. All can be accessed with a Python library given a valid API key. The Maps Static API is able to return a picture of the map centered around a location, with a designated zoom level. The Places API is able to search for locations of interest around a certain point. This search allows keywords as well as type filters. Possible type filters include "police", "restaurant", "gas_station", etc. The Geocoding API is able to return an address given a location, and vice versa.

The Google Maps API also uses several coordinate systems. The most common is the latitude-longitude coordinate system, with the latitude being a number from -90 to 90, and longitude from -180 to 180. These numbers represent the angle from the center of the earth to the point of interest. The Places and Geocoding APIs use these coordinates as inputs and outputs. A second coordinate system is the World coordinates. These coordinates have x and y values between 0 and 256, and depend on the horizontal and vertical displacement in the Mercator projection. The Mercator projection is a projection of the Earth's surface onto a flat plane, and is distorted at low and high y. The Maps Static API uses the world coordinates to produce a flat image of the area around our desired location. In our project, functions are used to convert between the coordinate systems. Anytime something is drawn on the map, it must be done in world coordinates, so the latitude/longitude coordinates extracted from the Places API are converted to world coordinates. The Voronoi diagram is drawn using the world coordinates of the sites.

To compute the Voronoi diagram, Scipy's "Voronoi" class is used, and the voronoi_plot_2d function is used to graph the diagram. The Voronoi class is able to compute the vertices, ridges (segments connecting vertices), and the site and vertex coordinates. Our bounding box is contained by the edges of the graph, but the area of the computed Voronoi diagram is larger than this by an O(n) factor (roughly 9). This ensures that even if the isolated point is close to the boundary, it is still accurately calculated to the nearest sites. To calculate the isolated point, the program loops through all vertices which are inside of the bounding box. The connecting ridges determine which sites define the vertex (lie on the circumcircle), and the distance is calculated to one of these sites. To calculate distance between the vertex and the sites, the coordinates of each are first converted into latitude/longitude, then the distance is calculated in kilometers. The conversion from world coordinates directly to kilometers is difficult, as it depends on the latitude of the location. The vertex and distance corresponding to the maximum distance is recorded. These points and lines are graphed using Matplotlib.

Shown below are two results of the program. The first picture finds the point most isolated from police stations in Cambridge. The second picture finds the point most isolated


Isolated Point: police
Willie O'Ree Community Rink, 235 Western Ave, Boston, MA 02134, USA
d = 1.81 km


Isolated Point: McDonalds
45 Belmont St, Somerville, MA 02143, USA
d = 3.27 km

from McDonalds in Cambridge. The map of the Cambridge/Boston area is shown in the background, centered on MIT. In red, are the graphed locations of the existing sites. The

Voronoi diagram is overlaid on the map in blue, and the most isolated point is shown by a blue dot. The most isolated point in our first example is in Lower Allston, with the nearest address at Willie O'Ree Community Rink, 235 Western Ave, Boston, MA 02134, USA. The distance to the nearest police stations is roughly 1.81 km. This location would be the location which is best fit for a new police station, in order to obtain maximum coverage. Similarly, the optimal location for a franchise owner to establish a new McDonalds would be in Somerville.

The runtime of this algorithm is O(n log n), where *n* is the number of sites found nearby by Google Maps. The Voronoi diagram is constructed in O(n log n) time, as determined by Fortune's algorithm. Furthermore, the calculation of the isolated point is O(n), given that there are O(n) vertices, with each calculation taking O(1) time.

There are some limitations to the algorithm. The algorithm works best in small scale maps, such as those on the scale of a city. For larger scale maps such as those of countries, the distances would be distorted, due to the world coordinates based on the Mercator Projection. The larger the change in y, the larger the distortion. The distance also cannot be calculated in the latitude/longitude coordinate system using euclidean distances. The most accurate way would be to discard the Scipy algorithm, and create a new Voronoi implementation which is able to take into account the non-euclidean distances, and then convert back to world coordinates to display on the map. The displayed map may not look as accurate, though the calculated distances would be. Another limitation lies in the Google Maps API search. Since many locations are family-owned businesses, they are placed on the map by the owner, and their categories are determined by the submitter. Thus, some locations do not show up depending on the searched keywords and types. When trying to find the fire stations in Cambridge, I was unable to find a result which contained an exhaustive list of all fire stations.

This project fuses geometric computing with Google maps, and the result is an algorithm that can find the most isolated place in a map. This program may be used by franchise owners, city planners, and government officials in analyzing optimal areas to place new stores and services.

APPENDIX

This is the code used in the project, including the functions for converting coordinates, calculating the Voronoi diagram, and graphing the results. A valid Google Maps API key is needed.

```
import math
from matplotlib import pyplot as plt
import numpy as np
from scipy.spatial import Voronoi, voronoi_plot_2d
import googlemaps
import requests

api_key = 'YOUR_API_KEY'
gmaps = googlemaps.Client(key=api_key)


#Convert Lat/Lng to World Coordinates
def latlng_to_world(latlng):
    TILE_SIZE = 256
    siny = math.sin((latlng[0] * math.pi) / 180)
    siny = min(max(siny, -0.9999), 0.9999)
    worldx = TILE_SIZE * (0.5 + latlng[1] / 360)
    worldy = - TILE_SIZE * (0.5 - math.log((1 + siny) / (1 - siny)) / (4 * math.pi))
    return np.array([worldx, worldy])

def world_to_latlng(world):
    worldx, worldy = world
    TILE_SIZE = 256

    lng = (worldx / TILE_SIZE - 0.5) * 360
    f = math.e**((worldy / TILE_SIZE + 0.5) * (4 * math.pi))
    siny = (f-1)/(f + 1)
    lat = math.asin(siny) * 180 / math.pi
    return np.array([lat, lng])

def latlng_to_km(lat_dist):
    #360 degrees is 2pi r
    earth_radius = 6371 #in km
    distance = 2*math.pi*earth_radius/360 * lat_dist
    return distance

def world_to_km(world_dist):
    #2048 is 2pi r
    earth_radius = 6371 #in km
    distance = 2*math.pi*earth_radius/2048*world_dist
    return distance

mit_lng_coordinates = (42.35923382978076, -71.09311966503593)
mit_world_coordinates = latlng_to_world(mit_lng_coordinates)
print(world_to_latlng(mit_world_coordinates))
place_name = "McDonalds"
place_type = "restaurant"
place_query = gmaps.places(place_name, mit_lng_coordinates, 4000, type = place_type)
places = place_query['results']
num_places = len(places)
places_coords = []
for place in places:
    loc = place['geometry']['location']
    latlng_coord = np.array([loc['lat'], loc['lng']])
    coord = latlng_to_world(latlng_coord)
    places_coords.append(coord)
places_coords = np.array(places_coords)

#plotting
fig = plt.figure()
ax = fig.gca()
ax.axes.xaxis.set_ticks([])
ax.axes.yaxis.set_visible(False)
ax.figure.set_size_inches(10, 10)
ax.scatter(places_coords[:,0], places_coords[:,1], c = 'red')
ax.set_title("Isolated Point: " + place_name)


#Get Image

url = "https://maps.googleapis.com/maps/api/staticmap?"
center = "Cambridge, Massachusetts"
zoom = 13
r = requests.get(url + "center=" + str(mit_lng_coordinates)[1:-1] + "&zoom=" +
                   str(zoom) + "&size=512x512&scale=2&key=" +
                             api_key)

f = open('map.png', 'wb')
f.write(r.content)
f.close()

#plot image
img = plt.imread("map.png")
center_world = mit_world_coordinates
width = 256/2**(zoom)
xlim = (center_world[0] - width, center_world[0] + width)
ylim = (center_world[1] - width, center_world[1] + width)
ax.imshow(img, extent=[xlim[0], xlim[1], ylim[0], ylim[1]])
```

```python
        # Calculate Voronoi Polygons
        vor = Voronoi(places_coords)

        def simple_voronoi(vor, saveas=None, xlim=None, ylim = None):
            # Make Voronoi Diagram
               fig  =  voronoi_plot_2d(vor,  show_points=False,  show_vertices=False,  s=4,  ax  =  ax,
line_colors = "blue", line_width=2, line_alpha=0.3, point_size=2)

            if xlim and ylim:
                ax.set_xlim(*xlim)
                ax.set_ylim(*ylim)
        simple_voronoi(vor, xlim = xlim, ylim = ylim)

        points = vor.points
        vertices = vor.vertices
        ridge_points = vor.ridge_points
        ridge_vertices = vor.ridge_vertices
        num_vertices = len(vertices)
        num_ridge = len(ridge_points)
        max_dist = 0
        max_vertex = 0
        max_point = 0

        for i in range(num_ridge):
            ridge_point = ridge_points[i][0]
            ridge_vertex = ridge_vertices[i][0]
            if ridge_vertex != -1:
                coord = vertices[ridge_vertex]
                if xlim[0] < coord[0]  and coord[0] < xlim[1] and ylim[0] < coord[1]  and coord[1] <
ylim[1]:
                    dist = math.dist(points[ridge_point], vertices[ridge_vertex])
                    if dist > max_dist:
                        max_dist = dist
                        max_vertex = ridge_vertex
                        max_point = ridge_point
            ridge_vertex = ridge_vertices[i][1]
            coord = vertices[ridge_vertex]
            if ridge_vertex != -1:
                if xlim[0] < coord[0]  and coord[0] < xlim[1] and ylim[0] < coord[1]  and coord[1] <
ylim[1]:
                    dist = math.dist(points[ridge_point], vertices[ridge_vertex])
                    if dist > max_dist:
                        max_dist = dist
                        max_vertex = ridge_vertex
                        max_point = ridge_point
        max_vertex_coord = vertices[max_vertex]
        max_point_coord = points[max_point]
        reverse_geocode_result = gmaps.reverse_geocode(world_to_latlng(max_vertex_coord))
        print(reverse_geocode_result[0]['formatted_address'])
        print(max_dist)
        max_dist_km = latlng_to_km(math.dist(world_to_latlng(max_vertex_coord),
world_to_latlng(max_point_coord)))
        #max_dist_km = world_to_km(max_dist)
        ax.scatter(vertices[:,0], vertices[:,1], s = (2)**2*np.ones(num_vertices), c = 'black')
        ax.scatter(max_vertex_coord[0], max_vertex_coord[1], s = 8**2, c = "navy")
        ax.set_xlabel(reverse_geocode_result[0]['formatted_address'] + "\n d = " +
str(np.round(max_dist_km, 2)) + " km")
        plt.show()
```