Optimizing the Ambulance Deployment in Ithaca

Jonathan Pang, Hong-Seok Choe, Sam Boardman

November 11th, 2018

Abstract

This paper aims to optimize the possible ambulance locations in the City of Ithaca. Using the well-established model of maximal covering location problem, we obtain the optimal ambulance locations among the potential ambulance locations that are selected from our model assumptions. Furthermore, we propose the possible assigning ratios for each optimal location that can be used for the appropriate allocation of the ambulances.

Contents

Bangs Ambulance Inc.,

We are delighted to have worked with you in improving the response rate of emergency medical services within the City of Ithaca. As you know, seconds count, so our approach was inherently mathematical.

We recommend that half of the ambulances be placed at Belle Sherman School and the other half at Mobil Gas Station. Particularly, 77% of ambulance call locations can be reached from the school within 6 minutes; 83% from the gas station. Since these locations are quite distanced within Ithaca, this placement of ambulances should ensure complete coverage of the city within 6 minutes. While we were unable to give the exact number of ambulances, the optimized location does not change based on how many ambulances are at disposal. Furthermore, since the number of ambulance calls takes on four distinct patterns over the day: one for each 6-hour interval (which dominates the change in demand over the weekend), we recommend experimentation with the exact number of ambulances during these four times, conserving ambulances more as desired results are achieved at higher numbers.

We hope that this information will be invaluable to your operations, and, ultimately, the citizens of Ithaca.

Sincerely,

Jonathan Pang, Hong-Seok Choe, and Sam Boardman

1 Introduction

The issue of optimization permeates all aspects of society, including the vitally important task of ensuring that EMS services can render help to individuals as quickly as possible. We consider this task with respect to the position and number of ambulances within the City of Ithaca. This entails factors intrinsic to Ithaca itself–such as the locations that can accommodate an ambulance–and more universal factors–such as the distribution of ambulance calls throughout the day and maximizing the proportion of ambulance calls covered subject to constraints on the ambulances. Combining this information and methodology, we apply broader mathematical principles to the nuances of the City of Ithaca, seeking to best suit their EMS needs. The report begins with preliminary assumptions and a precise statement of the components that will take part in the analysis. Next, it employs regression to predict the number of ambulance calls expected at any given time. Finally, it synthesizes all aspects of the model to solve an integer linear problem, thereby obtaining recommended placement and relative quantities of ambulances.

2 Model Assumptions

We use the maximal covering location problem (MCLP) originally proposed by Church and ReVelle (1974) [3]. In order to optimize the location of the ambulances in the city of Ithaca, we based our model on to the following information:

1. Potential Ambulance Locations: Possible locations in the city of Ithaca that consist of public or private parking spaces
2. Demand Points: Partitioned the city into a grid and weighted each point according to population density
3. Average Travel Time: Pairwise travel times computed between the potential ambulance locations and the demand points
4. Demand Scenarios: 4 scenarios represent demand variations due to the time of the day (mornings, afternoons, evenings or nights)

2.1 Potential Ambulance Locations

We hand-picked a total of 20 potential ambulance locations in the city using Google Maps [6]. This includes shopping malls, schools, and government offices in Ithaca. Potential bases must have proper parking space for ambulances along with other features such as access to electrical sockets and WC.

2.2 Demand Points

Demand points are areas where we expect a call. We evenly distributed these points across a 5x8 coordinate grid with 1/2 square miles in between (Unit in

decimal degrees (lat/long coordinates)). We scored each demand point by its population density. We found population density using the Federal Communications Commission API [1]. This API takes as input latitude and longitude coordinates and returns the population of the corresponding block.

2.3 Average Travel Times

After locating the potential ambulance bases and the demand points, we used the MapQuest API [7] to calculate the pairwise average travel times between potential ambulance locations and demand points. Although the values we obtain from the API are average travel times of cars without considering the complex factors such as traffic flow and weather conditions, we assume that the traveling behavior of ambulances resembles that of cars.

2.4 Demand Scenarios

Since we were only allowed with the data of total EMS calls of Ithaca, we interpolated a piece-wise sinusoidal function according to the national data. Using the function, EMS calls where further classified based on the time frames(from 12 AM to 6 AM, morning, from 6 AM to 12 PM, afternoon, from 12 PM to 6 PM, and evening, from 6 PM to 12 AM, night). Although studies show that the demand distribution patterns on Fridays, Saturdays, and Sundays are significantly different from those for Mondays to Thursdays, but these differences are associated with an increased number of trauma cases around midnight on Fridays and Saturdays and a decreased number of medical cases Saturdays and Sundays.[2] In our case, we assume that ambulances are dispatched in both cases, regardless of severity. Also since the deviation is small relative to the deviations in calls throughout the day, we use the same regression model for each scenario.

3 Model Parameters

The static ambulance location problem is modeled on a graph $G = (V \cup W, E)$. The set of demand points is denoted by V, and the set of potential ambulance locations is denoted by W. E, a set of edges $\{(i, j) : i \in V, j \in W\}$, represents the travel time t_{ij} [5] between ambulance site j and demand point i. Given a preset coverage standard $r > 0$, or simply the coverage time of EMS, a demand point $i \in V$ is said to be *covered* by the ambulance location $j \in W$ if $t_{ij} \leq r$. Let $W_i = \{j \in W : t_{ij} \leq r\}$ be the set of all ambulance locations covering the demand point i within time "radius" r.

We denote each scenario discussed in Section 2.4 by the letter $s \in S = \{1, 2, 3, 4\}$, morning $= 1$, afternoon $= 2$, evening $= 3$, and night $= 4$. Each demand point in each scenario can be associated with a demand $d_i^s \geq 0$, $s \in S$, $i \in V$, which in our case $d_i^s = \mu_s \cdot \lambda_i$, where μ_s is the expected number of calls in scenario s, and

λ_i is the population density of vertex i (i.e., the relative population density of i^{th} grid in the partition). $\mu_s = \int_{t_0}^{t_1} \rho(t)\ dt$ is computed via the regression function $\rho(t)$, and $\lambda_i = \frac{p_i}{p_0}$ is computed from the census data of Ithaca city[CITE] where p_i is the population of vertex i and p_0 is the total population.

4 Model Implementation

4.1 Solving MCLP

The main objective of MCLP is to maximize the demand that is covered at least once within a given time standard r. Defined on a graph G, a mathematical formulation of the problem can be stated as follows [3]:

Maximize

$$Z = \sum_{i \in V} d_i^s y_i$$

subject to

$$\sum_{j \in W_i} x_j \geq y_i \quad (i \in V)$$

$$\sum_{j \in W} x_j = p$$

$$x_j \in \{0, 1\} \quad (j \in W)$$

$$y_i \in \{0, 1\} \quad (i \in V)$$

where

$V =$ denotes the set of demand points

$W =$ denotes the set of potential ambulance points

$d_i^s =$ the demand of vertex i

$p =$ the total number of ambulances

$W_i = \{j \in W : t_{ij} \leq r\}$

$$x_j = \begin{cases} 1 & \text{if an ambulance is located at vertex j} \\ 0 & \text{otherwise} \end{cases}$$

$$y_i = \begin{cases} 1 & \text{if vertex i is covered by at least one ambulance} \\ 0 & \text{otherwise} \end{cases}$$

4.2 Estimating Expected Calls

The figures use a finite number of points to report a number (or fraction) of cases; we instead use the shape of the graphs to give a continuous time density of ambulance calls, which is proportional to percent of overall demand

and number of cases. The following four points are used for sinusoidal regression of percent of overall demand vs. hour of day in the interval [0, 12] (in hours), based on the mean value of the 7 curves for each day of the week: $(1, .42); (5, .26); (8, .46); (12, .83)$ Sinusoidal regression, with period of 14 hours, yields the following function:

$$\rho(t) = .256 \sin(.449t + 2.76) + .423, \ 0 < t < 12$$

For the remaining intervals, the curves for the 7 days are unwieldy, so the second plot is used for regression, which contains the average tendency of the time density for each day of the week. While the authors of the Melbourne paper note that there is a statistically significant difference in the number of calls per day of the week, the deviation is small relative to the deviations in calls throughout the day. Thus, we use the same model for each day of the week. With that in mind, the following two points are used for linear regression of number of cases vs. hour of day in the interval $(12, 16.5]$ (in hours): $(12, 69000)$; $(16.5, 61000)$ Using the factor $.83 : 69000$ to convert from number of cases to percent of overall demand, linear regression yields the following function:

$$\rho(t) = -.0184t + .935, \ 12 < t < 16.5$$

Lastly, the following four points are used for sinusoidal regression of number of cases vs. hour of day in the interval $(16.5, 24)$ (in hours): $(16.5, 61000)$; $(20, 59000)$; $(22, 53000)$; $(23, 45000)$ Using the factor $.83 : 69000$ to convert from number of cases to percent of overall demand, sinusoidal regression, with period of 21 hours, yields the following function:

$$\rho(t) = .193 \sin(.299t + 2.45) + .456, \ 16.5 < t < 24$$

Combining these functions, we obtain a piece-wise function the describes the rate of percent of overall demand per hour of day. Integrating this function over 0 to 24 yields 14.1; we want the integral to result in the number of ambulance calls per day which, using our above assumptions, is $4415/365 = 12.1$. Using the factor $12.1 : 14.1$, we divide both sides of this equality and obtain the revised function $\rho(t)$, which gives ambulance calls/time in terms of calls per hour. Applying this function, we can integrate over any period of time throughout the day to calculate the number of ambulance calls expected throughout that time interval. This is invaluable for determining the quantity of ambulances that should be stationed at different times in the day, which in turn will affect the locations of ambulances; having fewer ambulances requires that each ambulance cover a greater area.

5 Results and Discussion

Note that MCLP is a integer linear problem, so we can optimize via CVXPY [4] module in Python. As in the prompt, we set

- $r = 6\text{min}$

Furthermore, since no value for r2 is given, we set the possible number of ambulances p from 3 to 10 to study the optimal locations.

- $3 \leq p \leq 10$

We use the function obtained from the regression to compute μ_s for each scenario s.

$$\text{Estimated Expected Calls in Scenario 1} = \mu_1 = \int_0^6 \rho(t)\ dt \approx 1.62$$

$$\text{Estimated Expected Calls in Scenario 2} = \mu_2 = \int_6^{12} \rho(t)\ dt \approx 3.09$$

$$\text{Estimated Expected Calls in Scenario 3} = \mu_3 = \int_{12}^{18} \rho(t)\ dt \approx 3.99$$

$$\text{Estimated Expected Calls in Scenario 4} = \mu_4 = \int_{18}^{24} \rho(t)\ dt \approx 3.38$$

Once we have values of μ_s, we computed d_i^s from the population distribution λ_i. The set of potential bases W and the set of demand point V were calculated in coordinates as described earlier.

5.1 Summary of Results

For each value of $3 \leq p \leq 10$, we obtain the same set of optimal locations $(42.438026, -76.478207)$ and $(42.43955, -76.508107)$, which are Belle Sherman School and Mobil(gas station) It should be noted that given any scenario s, the solution remains unchanged. For the simple sanity check, Belle Sherman School has 29 demand points reached with the total sum of proportions 0.77(i.e. 77 percent of demands is covered). Also, Mobil gas Station reaches 35 demand points with the total sum of proportions 0.83 (i.e. 83 percent of the demand is covered). This implies that both optimal ambulance locations cover the entire demand points regardless of the time frame.

5.2 Further Suggestions

Given the two optimal ambulance points A and B, it suffices to assign adequate number of ambulances. Since the availability of *Bangs Ambulance Inc.* within the city of Ithaca is unknown, we suggest the ratio of allocation instead. We find the nearest neighbor(A or B) for each demand point by comparing the average travel times. Let $V_A = \{i \in V : t_{iA} \leq t_{iB}\}$ and $V_B = \{i \in V : t_{iB} \leq t_{iA}\}$ be the set of nearest neighbors of A and B, respectively. Since V_A and V_B partitions V, we can compute the relative coverage ratio $\lambda_A = \sum_{i \in V_A} \lambda_i \approx 0.4915$ and $\lambda_B = \sum_{i \in V_B} \lambda_i \approx 0.5085$. If the total number of ambulances available in the

city of Ithaca is p, we suggest that $p \cdot \lambda_A$ number of ambulances in location A(i.e., Belle Sherman School) and $p \cdot \lambda_B$ number of ambulances in location B(i.e., Mobil gas station).

5.3 Discussion

The proposed model is computationally-driven and can be easily modified according to the different demand scenarios. However, as in [5] , density points should be carefully constructed. Since we lack the actual calling locations, we were not able to cluster potential density points according to the actual call data. It should be noted that one of the reasons of obtaining only two optimal ambulance points is due to our naïve assumptions for density points.

6 Conclusion

Since MCLP finds the optimal ambulance locations given the total number of ambulances, we had to solve for each possible p, which fortunately yielded the same optimal locations. However, this does not guarantee the optimal number of ambulances that should be assigned to each optimal location. Thus, we further computed the ratio that can be used for assigning the adequate ambulances to the two locations. Given the number of ambulances that *Bangs Ambulance Inc* can utilize, we conclude that assign half of them to Belle Sherman School and the other half to Mobil gas station.

7 Appendix

Listing 1: Code for Solving Optimization via CVXPY

```
import cvxpy
import numpy as np

# The data for the ambulance deployment problem

# The variable we are solving for
amb_points = cvxpy.Variable(20, boolean = True)
y = cvxpy.Variable(40, boolean = True)
expected_calls = [1.62, 3.09, 3.99, 3.38] #Corresponds to Scenarios 1,2,3,4
d = expected_calls[i] * #Assumes Scenario i
    np.array([0.015399422521655439,0,0,0.0012832852101379532,0.004812319538017324,
    0.013795316008982997,0.05710619185113892,0.09496310555020854,0.015399422521655439,
    0.017645171639396856,0,0.0016041065126724415, 0.034327879371190244,0.005774783445620789,
    0,0,0.004170676932948348,0.005453962143086301,0.0003208213025344883,0.03144048764837985,
    0,0.07988450433108758,0.03144048764837985,0,0.02630734680782804,0.007699711260827719,
```

```
    0.030798845043310877,0.02791145332050048,0.0028873917228103944,0.059351940968880336,
    0.03657362848893166,0.021495027269810715,0.005774783445620789,0.030798845043310877,
    0.030798845043310877,0.015720243824189926,0.0038498556304138597,0.0792428617260186,
    0.10073788899582932,0.10522938723131216]) #the list of population densities(lambda_i's)

# w_i for each demand point i in V should be greater than or equal to y_i
w0_indices = [ 2, 4, 5, 6, 10, 12, 13, 14, 15, 17, 18 ]
constraint0 = sum([amb_points[i] for i in w0_indices]) >= y[0]
w1_indices = [ 2, 3, 4, 5, 6, 7, 10, 12, 13, 14, 15, 17, 18 ]
constraint1 = sum([amb_points[i] for i in w1_indices]) >= y[1]
w2_indices = [ 2, 5, 6, 13, 14, 15, 17, 18 ]
constraint2 = sum([amb_points[i] for i in w2_indices]) >= y[2]
w3_indices =  [ 3, 10, 11, 12, 13, 14, 15 ]
constraint3 = sum([amb_points[i] for i in w3_indices]) >= y[3]
w4_indices =  [ 1, 2, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 ]
constraint4 = sum([amb_points[i] for i in w4_indices]) >= y[4]
w5_indices =  [ 1, 2, 3, 7, 8, 9, 12, 14 ]
constraint5 = sum([amb_points[i] for i in w5_indices]) >= y[5]
w6_indices =  [ 1, 9 ]
constraint6 = sum([amb_points[i] for i in w6_indices]) >= y[6]
w7_indices =  [ 1, 19 ]
constraint7 = sum([amb_points[i] for i in w7_indices]) >= y[7]
w8_indices =  [ 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 14, 15, 17, 18 ]
constraint8 = sum([amb_points[i] for i in w8_indices]) >= y[8]
w9_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 14, 15, 16, 17, 18 ]
constraint9 = sum([amb_points[i] for i in w9_indices]) >= y[9]
w10_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18 ]
constraint10 = sum([amb_points[i] for i in w10_indices]) >= y[10]
w11_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18 ]
constraint11 = sum([amb_points[i] for i in w11_indices]) >= y[11]
w12_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18 ]
constraint12 = sum([amb_points[i] for i in w12_indices]) >= y[12]
w13_indices =  [ 1, 2, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19 ]
constraint13 = sum([amb_points[i] for i in w13_indices]) >= y[13]
w14_indices = [ 1, 2, 3, 7, 8, 9, 12, 14, 19 ]
constraint14 = sum([amb_points[i] for i in w14_indices]) >= y[14]
w15_indices =  [ 1, 9, 19 ]
constraint15 = sum([amb_points[i] for i in w15_indices]) >= y[15]
w16_indices =  [ 5, 6, 13, 15, 17, 18 ]
constraint16 = sum([amb_points[i] for i in w16_indices]) >= y[16]
w17_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 14, 15, 16, 17, 18 ]
constraint17 = sum([amb_points[i] for i in w17_indices]) >= y[17]
w18_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18 ]
constraint18 = sum([amb_points[i] for i in w18_indices]) >= y[18]
w19_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 ]
constraint19 = sum([amb_points[i] for i in w19_indices]) >= y[19]
```

```
w20_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 ]
constraint20 = sum([amb_points[i] for i in w20_indices]) >= y[20]
w21_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19 ]
constraint21 = sum([amb_points[i] for i in w21_indices]) >= y[21]
w22_indices =  [ 1, 2, 3, 6, 7, 8, 9, 12, 13, 15, 16, 19 ]
constraint22 = sum([amb_points[i] for i in w22_indices]) >= y[22]
w23_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 19 ]
constraint23 = sum([amb_points[i] for i in w23_indices]) >= y[23]
w24_indices =  [ 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 14, 15, 16, 17, 18 ]
constraint24 = sum([amb_points[i] for i in w24_indices]) >= y[24]
w25_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18 ]
constraint25 = sum([amb_points[i] for i in w25_indices]) >= y[25]
w26_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 14, 15, 16, 17, 18 ]
constraint26 = sum([amb_points[i] for i in w26_indices]) >= y[26]
w27_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 ]
constraint27 = sum([amb_points[i] for i in w27_indices]) >= y[27]
w28_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19 ]
constraint28 = sum([amb_points[i] for i in w28_indices]) >= y[28]
w29_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19 ]
constraint29 = sum([amb_points[i] for i in w29_indices]) >= y[29]
w30_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19 ]
constraint30 = sum([amb_points[i] for i in w30_indices]) >= y[30]
w31_indices =  [ 1, 2, 3, 6, 7, 8, 9, 12, 13, 15, 16, 19 ]
constraint31 = sum([amb_points[i] for i in w31_indices]) >= y[31]
w32_indices =  [ 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 14, 15, 16, 17, 18 ]
constraint32 = sum([amb_points[i] for i in w32_indices]) >= y[32]
w33_indices =  [ 2, 4, 5, 6, 7, 13, 15, 17 ]
constraint33 = sum([amb_points[i] for i in w33_indices]) >= y[33]
w34_indices =  [ 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 14, 15, 16, 17, 18 ]
constraint34 = sum([amb_points[i] for i in w34_indices]) >= y[34]
w35_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18 ]
constraint35 = sum([amb_points[i] for i in w35_indices]) >= y[35]
w36_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19 ]
constraint36 = sum([amb_points[i] for i in w36_indices]) >= y[36]
w37_indices =  [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19 ]
constraint37 = sum([amb_points[i] for i in w37_indices]) >= y[37]
w38_indices =  [ 2, 3, 6, 7, 8, 12, 15, 16 ]
constraint38 = sum([amb_points[i] for i in w38_indices]) >= y[38]
w39_indices =  [ 1, 2, 3, 6, 7, 8, 9, 12, 13, 15, 16, 19 ]
constraint39 = sum([amb_points[i] for i in w39_indices]) >= y[39]

constraint40 = sum(amb_points) <= p #p is the total number of ambulances


constraints = [constraint0, constraint1, constraint2, constraint3, constraint4,
               constraint5, constraint6, constraint7, constraint8, constraint9,
```

```
                constraint10, constraint11, constraint12,constraint13, constraint14,
                constraint15, constraint16, constraint17, constraint18, constraint19,
                constraint20, constraint21, constraint22, constraint23, constraint24,
                constraint25, constraint26, constraint27, constraint28, constraint29,
                constraint30, constraint31, constraint32, constraint33, constraint34,
                constraint35, constraint36, constraint37, constraint38, constraint39,
                constraint40]

# Our obejctive function is the sum of d*y's
obj = cvxpy.sum(cvxpy.multiply(d,y))


# We tell cvxpy that we want to maximize objective
# subject to vertices_constraint.
amb_problem = cvxpy.Problem(cvxpy.Maximize(obj), constraints)

# Solving the problem
amb_problem.solve()
print(y.value)
print (amb_points.value)
```

Listing 2: Code for Demand Points and Potential Ambulance Locations

```
const fetch = require("isomorphic-fetch");
const fccUrl = "https://geo.fcc.gov/api/census/area?";
const API_KEY = "";
const Url = "http://www.mapquestapi.com/directions/v2/routematrix?key="+API_KEY;

const WEST_BOUND = -76.526182;
const NORTH_BOUND = 42.456826;
const HALF_MILE = 0.00725;

var schools = [];
var malls = [];
var garages = [];
var gasStations = [];
var govtOffices = [];
var bangs = new Ambu("Bang's","Bang's",42.4453215,-76.5168851,0);
var demandPoints = [];
var ambulances = [bangs];

/**
 * @constructor
 * Possible ambulance location in Ithaca, NY
 *
 * @param {string} type
```

```
 *      The category that the spot falls under
 * @param {string} title
 *      Parking spot nickname for identification
 * @param {float} lat
 *      Latitude in decimal degrees
 * @param {float} lng
 *      Longitude in decimal degrees
 * @param {int} index
 *      Integer id for ambulance point
 * @property {float} scoreNoPop
 *      The calculated "value" of the ambulance point without taking population into account
 * @property {float} scorePop
 *      The calculated "value" of the ambulance point taking into account population proport
 * @property {float} nearestProp
 *      Only applies to ambulances at indices 1 and 15. Sum of population density that ambul
 *      "Responsible" means closest to.
 */
function Ambu(type,title,lat,lng,index) {
    this.type = type;
    this.title = title;
    this.lat = lat;
    this.lng = lng;
    this.index = index;
    this.scoreNoPop;
    this.scorePop;
    this.nearestProp = 0;
}

/**
 * @constructor
 * Demand point within Ithaca, NY
 *
 * @param {float} lat
 *      Latitude in decimal degrees
 * @param {float} lng
 *      Longitude in decimal degrees
 * @property {Ambu Array} ambuReach
 *      Set of ambulance locations with <6 min travel time in between
 * @param {int} index
 *      Integer id for ambulance point
 * @property {int} population
 *      Population at the block of the coordinate
 * @property {float} proportion
 *      Population at the block of the coordinate divided by the total calculated block popu
 */
function Demand(lat,lng,index) {
```

```
    this.lat = lat;
    this.lng = lng;
    this.ambuReach = [];
    this.index = index;
    this.population;
    this.proportion;
}

//MAIN FUNCTION, function to execute all of the async functions in the correct order
async function main() {
    initAmbulanceData();
    initDemandData();
    await fetchFCC();
    calculateProportion();
    await executeCalls();
    scoreAmbulance(false,1);
    updateScores();

    console.log(ambulances[1].title);
    console.log(ambulances[1].lat);
    console.log(ambulances[1].lng);
    console.log(ambulances[1].scoreNoPop);
    scoreAmbulance(true,1);
    console.log(ambulances[1].scorePop);
    console.log(ambulances[1].nearestProp);
    console.log(ambulances[15].title);
    console.log(ambulances[15].lat);
    console.log(ambulances[15].lng);
    scoreAmbulance(false,15);
    console.log(ambulances[15].scoreNoPop);
    scoreAmbulance(true,15);
    console.log(ambulances[15].scorePop);
    console.log(ambulances[15].nearestProp);
}

main();



/**
 * ------------------------------ HELPER FUNCTIONS --------------------------
 * |                    For calculation and scoring/weighting nodes          |
 * ---------------------------------------------------------------------------
 */

//Calculate the total population and divide to find the proportion
```

```
//Precondition: Execute AFTER calling fetchFCC()
function calculateProportion() {
    var totalPop = 0;
    for (let i=0; i<demandPoints.length; i++) {
        totalPop += demandPoints[i].population;
    }
    for (let i=0; i<demandPoints.length; i++) {
        demandPoints[i].proportion = (demandPoints[i].population)/totalPop;
    }
}

/**
 * Scores one ambulance node (specify by index) by counting how many demand points the ambu
 * reaches in under 6 minutes
 *
 * @param {boolean} weighted
 *      State whether the demand points should be weighted
 * @param {int} index
 *      Index of the ambulance node you wish to score
 * */
function scoreAmbulance(weighted, index) {
    var ambu;
    for(let i=0; i<ambulances.length; i++){
        if (ambulances[i].index == index) {
            ambu = ambulances[i];
        }
    }
    if (weighted) {
        ambu.scorePop = 0;
        for(let i=0;i<demandPoints.length;i++){
            if (compareAD(demandPoints[i].ambuReach, ambu.lat, ambu.lng)) {
                ambu.scorePop+=demandPoints[i].proportion;
            }
        }
    } else {
        ambu.scoreNoPop = 0;
        for(let i=0;i<demandPoints.length;i++){
            if (compareAD(demandPoints[i].ambuReach, ambu.lat, ambu.lng)) {
                ambu.scoreNoPop++;

            }
        }
    }
}

/**
```

```
 * Updates scores of our preferred ambulance locations, location 1 (Belle Sherman School)
 * and location 15 (Mobil Gas Station)
 */
function updateScores() {
    var ambu1lat = 42.438026;
    var ambu1lng = -76.478207;
    var ambu15lat = 42.43955;
    var ambu15lng = -76.508107;
    var latLng;
    var proportion;
    var ambu1Time=0;
    var ambu15Time=0;
    var ambu1;
    var ambu15;

    for (let i=0; i<ambulances.length; i++) {
        if(ambulances[i].index == 1) {
            ambu1=ambulances[i];
        }
        else if(ambulances[i].index == 15) {
            ambu15=ambulances[i];
        }
    }

    for (let i=0; i<demandPoints.length; i++) {
        latLng = demandPoints[i].ambuReach;
        proportion = demandPoints[i].proportion;
        for (let n=0; n<latLng.length; n++) {
            if((latLng[n].lat == ambu1lat)&&(latLng[n].lng == ambu1lng)) {
                ambu1Time = latLng[n].time;
            }
            else if((latLng[n].lat == ambu15lat)&&(latLng[n].lng == ambu15lng)) {
                ambu15Time = latLng[n].time;
            }
        }
        if (ambu1Time < ambu15Time) {
            ambu1.nearestProp += proportion;
        }
        else if (ambu15Time < ambu1Time) {
            ambu15.nearestProp += proportion;
        }
    }
}

/**
 * ------------------------------ HELPER FUNCTIONS --------------------------
```

```
 * |                    Functions that initialize coordinate data            |
 * ------------------------------------------------------------------------------
 */

/**
 * Push into arrays data about each ambulance point that we handpicked
 */
function initAmbulanceData() {

    var schoolTitles = ["Belle Sherman School", "Beverly J Martin School", "New Roots Charte
    var schoolLats = [42.438026, 42.441562, 42.440160];
    var schoolLongs = [-76.478207, -76.502234, -76.49934];
    var numSchools = schoolTitles.length;

    var mallTitles = ["Ithaca Shopping Plaza", "Tops Plaza", "Press Bay Alley"];
    var mallLats = [42.430012, 42.431831, 42.438846];
    var mallLongs = [-76.506928, -76.510317, -76.500102];
    var numMalls = mallTitles.length;

    var garageTitles = ["Green Street Garage", "Seneca Street Garage", "Dryden Road Garage"]
    var garageLats = [42.439177, 42.440974, 42.442192];
    var garageLongs = [-76.496639, -76.496649, -76.486123];
    var numGarages = garageTitles.length;

    var gasTitles = ["Sunoco Central", "Fastrac", "Sunny's Convenience", "Mobil", "Sunoco So
    var gasLats = [42.441052, 42.443714, 42.450600, 42.439550, 42.430664, 42.431172, 42.4448
    var gasLongs = [-76.499685, -76.508439, -76.504390, -76.508107, -76.497114, -76.508954,
    var numGas = gasTitles.length;

    var govTitles = ["DMV", "Ithaca Public Works"];
    var govLats = [42.447190, 42.453438];
    var govLongs = [-76.504649, -76.503932];
    var numGov = govTitles.length;

    // Create Ambu objects
    for (let i=0; i<numSchools; i++) {
        var school = new Ambu(
            "School",
            schoolTitles[i],
            schoolLats[i],
            schoolLongs[i],
            i+1
        );
        schools.push(school);
        ambulances.push(school);
    }
```

```
    for (let i=0; i<numMalls; i++) {
        var mall = new Ambu(
            "Mall",
            mallTitles[i],
            mallLats[i],
            mallLongs[i],
            i+1+numSchools
        );
        malls.push(mall);
        ambulances.push(mall);
    }
    for (let i=0; i<numGarages; i++) {
        var garage = new Ambu(
            "Garage",
            garageTitles[i],
            garageLats[i],
            garageLongs[i],
            i+1+numSchools+numMalls
        );
        garages.push(garage);
        ambulances.push(garage);
    }
    for (let i=0; i<numGov; i++) {
        var gov = new Ambu(
            "Government Office",
            govTitles[i],
            govLats[i],
            govLongs[i],
            i+1+numSchools+numMalls+numGarages
        );
        govtOffices.push(gov);
        ambulances.push(gov);
    }
    for (let i=0; i<numGas; i++) {
        var gas = new Ambu(
            "Gas Station",
            gasTitles[i],
            gasLats[i],
            gasLongs[i],
            i+1+numSchools+numMalls+numGarages+numGov
        );
        gasStations.push(gas);
        ambulances.push(gas);
    }
}
```

```
/**
 * Starting from the most north-west coordinate we designate, we calculate the
 * rest of the points in the 5x8 coordinate graph (separated by 1/2 sq mi)
 */
function initDemandData() {
    for (let m=0; m<5; m++) {
        for(let n=0; n<8; n++) {
            demandPoints.push(
                new Demand(
                    NORTH_BOUND - m*HALF_MILE, //subtract b/c we iterate southward
                    WEST_BOUND + n*HALF_MILE, //add b/c iterate eastward
                    m*8 + n
                )
            );
        }
    }
}

/**
 * ------------------------------ HELPER FUNCTIONS --------------------------
 * |                    Functions that make http reqs (Mapquest, FCC)         |
 * ------------------------------------------------------------------------------
 */


/**
 * Mapquest calculates the time distance between a demand point (first lat-lng coordinate)
 * and each of the 20 ambulance points (next 20 lat-lng coordinates)
 *
 * Then we add to the demand point the coordinates of the ambulance points where the distan
 * is less than or equal to 6 min
 */
async function executeCalls() {
    //Preparing mapquest API input data
    for (x=0; x<demandPoints.length; x++) {
        var Data = {
            "locations": []
        };
        Data.locations.push(
            {
                "latLng": {
                    "lat": demandPoints[x].lat,
                    "lng": demandPoints[x].lng
                }
            }
        );
```

```
        for (y=0; y<ambulances.length; y++) {
            Data.locations.push(
                {
                    "latLng": {
                        "lat": ambulances[y].lat,
                        "lng": ambulances[y].lng
                    }
                }
            );
        };
        //execute the fetch
        response = await fetch(Url, {
            method: 'post',
            body: JSON.stringify(Data)
        });
        var json = await response.json();
        console.log(json);
        timeArray = json.time;
        locationArray = json.locations;

        //filter by time distance
        for(let i=0; i<timeArray.length; i++) {
            if (((timeArray[i]/60)<= 6) && (timeArray[i] != 0)) { //conditional-- exclude fr
                console.log("Found a match");
                demandPoints[x].ambuReach.push(
                    {
                        "lat": locationArray[i].latLng.lat,"lng": locationArray[i].latLng.ln
                    }
                );
            };
        };
    };
};

//HTTP GET request to FCC server, which returns population
//in block corresponding to lat-lng coordinates specified in url
async function fetchFCC() {
    for (let i=0; i<demandPoints.length; i++) {
        url = fccUrl + "lat=" + demandPoints[i].lat + "&lon=" + demandPoints[i].lng + "&form
        var response = await fetch(url, {
            method: 'get'
        });
        var json = await response.json();
        var result = await json.results[0];
        var blockPop = await result.block_pop_2015;
        demandPoints[i].population = blockPop;
```

```
    }
}

/**
 *  ------------------------------ HELPER FUNCTIONS --------------------------
 *  |               For visibility and formatting data for other code         |
 *  ---------------------------------------------------------------------------
 */

/**
 * Compare latitude and longitude between ambulance coordinate and elements of reached ambu

 * @param {Array} latLng
 *      Array of ambulances that can access the demand point
 * @param {float} lat
 *      Ambulance latitude
 * @param {float} lng
 *      Ambulance longitude
 * @returns {boolean}
 *      True if demand coordinate reachable by ambulance
*/
function compareAD(latLng, lat, lng) {
    for(let i=0; i<latLng.length; i++) {
        if ((lat == latLng[i].lat)&&(lng == latLng[i].lng)) {
            return true;
        }
    }
    return false;
}


 // Prints demand point coordinates along with the ambulance coordinates with <=6 min of dr
function printPairwise() {
    for(let i=0; i<demandPoints.length; i++) {
        console.log("Demand point coordinates: "+demandPoints[i].lat+", "+demandPoints[i].ln
        reachedAmbuPoints = demandPoints[i].ambuReach;
        console.log("Accessible ambulance points:");
        for(let m=0; m<reachedAmbuPoints.length; m++) {
            console.log("Point "+m+": "+reachedAmbuPoints[m].lat+", "+reachedAmbuPoints[m].l
        };
    };
}

//See printPairwise-- does the same thing except prints ambulance index (id number) in place
function printPairwiseIndex() {
    for(let i=0; i<demandPoints.length; i++) {
```

```
        console.log("Demand point coordinates: "+demandPoints[i].lat+", "+demandPoints[i].ln
        reachedAmbuPoints = demandPoints[i].ambuReach;
        console.log("Accessible ambulance points: ");
        var tempArr = [];
        for (let m=0; m<reachedAmbuPoints.length; m++) {
            for(let n=0; n<ambulances.length; n++) {
                if ((reachedAmbuPoints[m].lat == ambulances[n].lat) && (reachedAmbuPoints[m]
                    tempArr.push(ambulances[n].index);
                }
            }
        }
        console.log(tempArr);
    }
}

//Print same data as printPairwise except formatted in separate arrays for latitude, longitu
//along with following a template for plotly.js
function printPlotData() {
    for(let i=0; i<demandPoints.length; i++){
        var xCoords = [];
        var yCoords = [];
        xCoords.push(demandPoints[i].lng);
        yCoords.push(demandPoints[i].lat);
        reachedAmbuPoints = demandPoints[i].ambuReach;
        for(let m=0; m<reachedAmbuPoints.length; m++) {
            xCoords.push(reachedAmbuPoints[m].lng);
            xCoords.push(demandPoints[i].lng);
            yCoords.push(reachedAmbuPoints[m].lat);
            yCoords.push(demandPoints[i].lat);
        }
        console.log("var demandPoint"+i+"={")
        console.log("x: " + "[" + xCoords + "],");
        console.log("y: " + "[" + yCoords + "],");
        console.log("mode: 'lines+markers', name: 'demandPoint"+i+"', type: 'scatter'};");
    }
}

//Print all data inside ambulance data points
function printAllAmbulanceData() {
    for(let i=0; i<ambulances.length; i++) {
        console.log(ambulances[i].index);
        console.log(ambulances[i].lat);
        console.log(ambulances[i].lng);
    }
}
```

```
//Print all data inside demand point, formatted for calculations in python
function printAllDemandData() {
    var tempArr = [];
    for(let i=0; i<demandPoints.length; i++) {
        console.log("Demand point "+demandPoints[i].index+":");
        console.log(demandPoints[i].lat+" , "+demandPoints[i].lng);
        console.log(demandPoints[i].population);
        console.log(demandPoints[i].proportion);
        tempArr.push(demandPoints[i].proportion);
    }
    console.log('['+tempArr+']');
}
```

References

[1] Federal communications commission api, 2018.

[2] Kate Cantwell, Amee Morgans, Karen Smith, Michael Livingston, Tim Spelman, and Paul Dietze. Time of day and day of week trends in ems demand. *Prehospital Emergency Care*, 19(3):425–431, 2015. PMID: 25664379.

[3] Richard Church and Charles ReVelle. The maximal covering location problem. In *Papers of the Regional Science Association*, volume 32, pages 101–118. Springer, 1974.

[4] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[5] Juan Carlos Dibene, Yazmin Maldonado, Carlos Vera, Mauricio de Oliveira, Leonardo Trujillo, and Oliver Schütze. Optimizing the location of ambulances in tijuana, mexico. *Computers in Biology and Medicine*, 80:107 – 115, 2017.

[6] Google. Google places api, 2018.

[7] Map Quest. Map quest api, 2018.