Caregiver Routing for Riverside PACE

Ben Weaver and Brian Goldman

December 17, 2014

CSCI 698

Abstract

Riverside PACE employs hundreds of caregivers to visit their patients' homes each day. In order to reduce transportation costs and allow caregivers more time with each patient, we devise an algorithm that assigns each caregiver to a group of patients who live close to one another. We find that grouping patients geographically may reduce mileage reimbursement to a quarter of the current cost. Finally, we discuss how new patients can join PACE without disrupting established patient-caregiver relationships or increasing mileage reimbursement costs.

1 Introduction

The Riverside Program of All-Inclusive Care for the Elderly (PACE) offers comprehensive in-home care to Virginia seniors who would otherwise need to live in a nursing home. As a subsidiary of the non-profit Riverside Health System, PACE strives to provide patients better care and greater independence than they would receive in a nursing home, but at a comparable cost. Many patients receive Medicare or Medicaid benefits to help pay for the program.

Trained caregivers pay regularly-scheduled visits to patients' homes, attend to the patients' routine needs, and help determine when a patient needs additional care. Not all caregiver visits are scheduled; if a patient has a pressing need, he or she can call PACE and request assistance.

Patients also schedule visits to their nearest PACE day center. There, patients may consult with their medical team, exercise, participate in planned social activities, or simply enjoy the company of other seniors. Patients come and go from the day centers via shuttle buses.

PACE permanently employs 250 in-home caregivers who serve approximately 650 patients in six locations: Charlottesville, Hampton, Newport News, Petersburg, MacTavish (Richmond), and Manchester (Richmond). PACE has also hired temporary workers to supplement their permanent staff. Permanent employees receive 56 nontaxable cents per mile driven between patients' homes— the standard rate determined by the IRS.

When a new patient joins PACE, it is a challenge to assign them to a caregiver. Most caregivers do not have time to care for a new patient, and the few caregivers who do have time may have to drive a considerable distance to reach the new patient's home. As a result, some caregivers now need to drive across town several times each day in order to tend to their patients. PACE believes that patients benefit from building relationships with individual caregivers, so the program's management has tried to keep patients with the caregivers they know already, regardless of where the patients live.

2 Problem Statement

PACE would like to shorten each caregiver's daily drive in order to reduce mileage reimbursement costs and allow caregivers more time with each patient. Current patients may need to switch caregivers now, but PACE would like these new assignments to be stable into the future.

Since caregivers may need to visit their patients in a different order each day, we will group patients into *clusters*. Each caregiver will serve a single cluster, and we wish to minimize the mean distance between all pairs of patients who share the same caregiver (we call this measure the *mean intra-cluster distance*). Grouping patients into such clusters will offer caregivers considerable flexibility to adjust to day-to-day needs. Because all patients in a single cluster will live close to one another, any path between those patients should be short. If a patient needs immediate, unscheduled help, his or her caregiver will not be far away. And if a caregiver cannot attend to all of his or her clients on a given day, PACE management can easily determine which other caregivers are nearby.

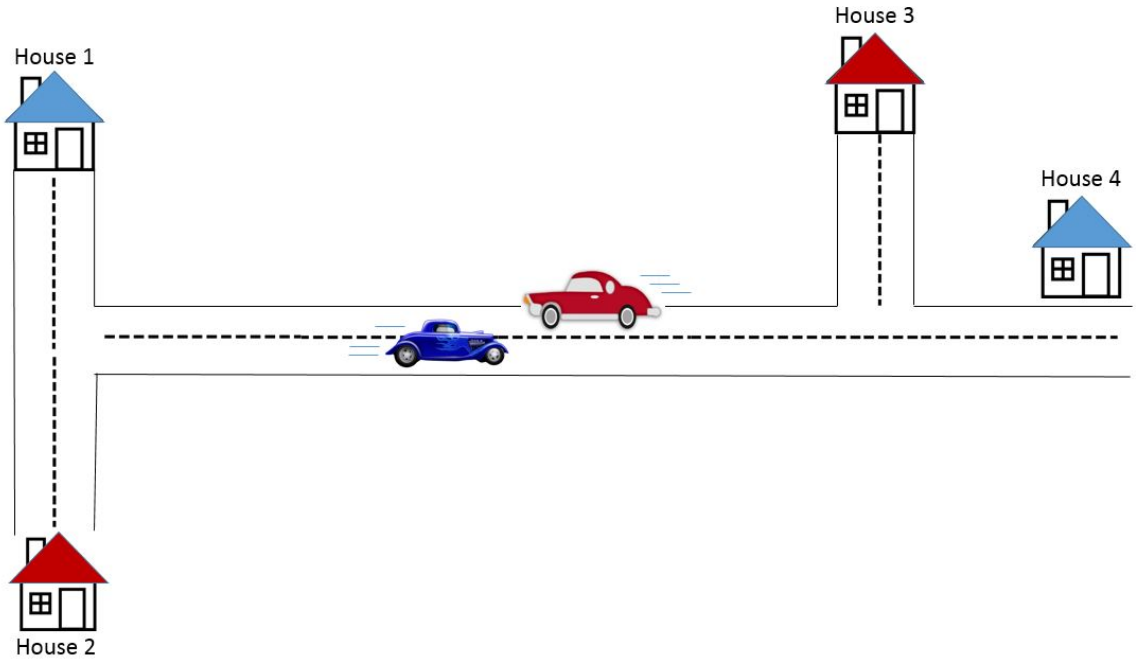

Figure 1: Poorly Clustered Patient Homes— A System Diagram

3 Methods

We will begin by considering how to group patients into clusters initially, then later discuss how to add and remove patients over time.

Although assigning patients to caregivers may resemble a standard clustering problem, we require that each caregiver serve approximately the same number of patients— a caveat that thwarts many known clustering algorithms. We also recognize that we cannot calculate the mean distance between patients sharing a caregiver for every possible clustering and then return the best result. For example, if we have 99 patients and 33 nurses, there are $\frac{99!}{3!^{33}33!} \approx 2 \times 10^{95}$ possible clusterings— many more than even the fastest computer could ever consider.

So, we designed several candidate clustering algorithms in Python and tested each to determine which would work best. Here, we summarize a few methods:

- **Spectral k-means:** Given n data points (patient addresses), the k-means algorithm seeks k hub points so that the average distance between each data point and its nearest hub point is minimized. Once the k-means algorithm finds these hub points, it places each data point in the same cluster as its nearest hub.

The spectral k-means algorithm refines this method by clustering the rows of the k largest eigenvectors of a Laplacian matrix defined using the distance between each pair of data points. Ng et al. discuss some advantages of the spectral k-means algorithm compared to the original k-means algorithm.

As we have suggested, neither the k-means nor the spectral k-means algorithm can guarantee that each cluster will be the same size. Indeed, if we try to use one of these methods to find a set of patients for each caregiver, just a few caregivers might find themselves responsible for most of the patients. So, we generate fewer clusters than we have caregivers, assign caregivers to clusters so that each cluster has approximately the same number of caregivers per patient, then use one of our other algorithms to decide which patients within a cluster will share a caregiver.

- **A Greedy Algorithm:** We begin with k hub points chosen to be far away from each other. Each hub point is given a cluster. Then, the hub points take turns adding data points to their cluster, always choosing the nearest data point still remaining. Since no hub point adds a second data point to its cluster until all other hub points have chosen a first, the data points are divided among the k clusters as evenly as possible.

- **Monte Carlo Clustering:** We randomly group n data points into k clusters, each containing $\frac{n}{k}$ points.[1] Then, we calculate the mean distance between all pairs of data points in the same cluster. We repeat this process thousands of times and return the clustering with the least mean intracluster distance.

- **Clustering Refinement:** Given any clustering of n points into k clusters, we ask for each pair of points x and y in different clusters whether swapping x into y's cluster and y into x's cluster would reduce the overall mean intracluster distance.[2] We continue switching points until no swap remains that would improve our clustering.

Several of these methods can be combined with each other. For example, we could use the greedy algorithm to generate an initial clustering to improve upon using cluster refinement. Or, we could use the spectral k-means algorithm to split the patients into a few large clusters and then use Monte Carlo methods to determine which patients within these large clusters should share a caregiver.

Since we do not have complete data on PACE's patients, nor do we know from where future patients will come, we tested our methods on 10 sets of 100 randomly generated addresses— approximately the number of patients at each of PACE's locations. We used a random address generator to find addresses in the postal code surrounding Leicester, England— a city of size comparable to those

[1] If n data points cannot be divided evenly into k clusters, then some clusters will have an extra data point.

[2] Note that switching x and y has no effect on any other cluster, so we need only calculate the total distance between x and the remaining points in y's cluster and the total distance between y and the remaining points in x's cluster.

that PACE serves.[3]

For each list, we calculated the distance between each pair of addresses using GPS coordinates. Unfortunately, it was not feasible to calculate the true drive-time between each pair of addresses. Online routing software (such as MapQuest or Google Maps) limit the number of drive-times any user can calculate per day.

In order to test our algorithms fairly, we allow each 10 seconds to split a given address set into 30 clusters as well as possible. Using the same address sets for each algorithm reduces the variance due to random address generation and allows us to directly compare two algorithms' effectiveness on each address set. We judge the clustering each algorithm produces quantitatively according to its mean intracluster distance, as well as graphically, using a map to display each cluster.

4 Results

We found we could discard a few candidate algorithms immediately. In Figure 2, we illustrate a clustering found using just Monte Carlo methods. Points of the same color share a cluster.[4]

[3] We would have preferred to use addresses in Richmond or Newport News, but we could not find a reliable random address generator for these localities.

[4] Because we have 30 clusters, the difference between some colors may appear a bit subtle.

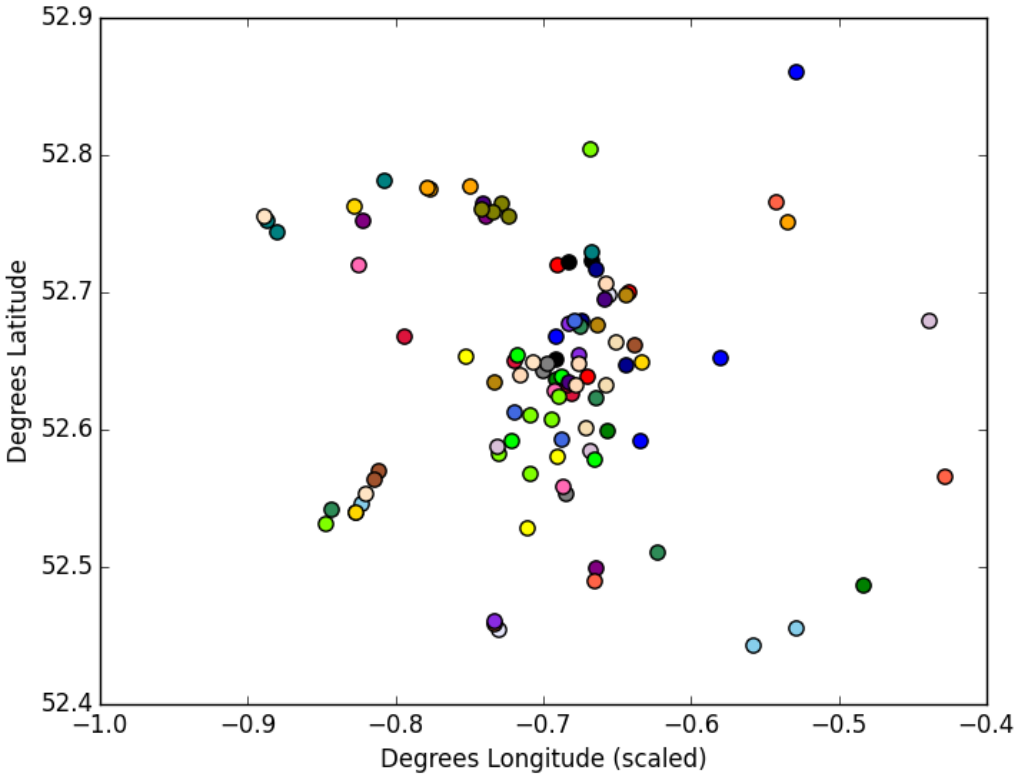

Figure 2: Monte Carlo Clustering.

The greedy algorithm did not perform much better, but the spectral k-means algorithm showed promise. In Figure 3, we show a clustering found first by dividing 100 patient addresses into large clusters, then using Monte Carlo simulation to decide which patients in the same cluster would share a caregiver.

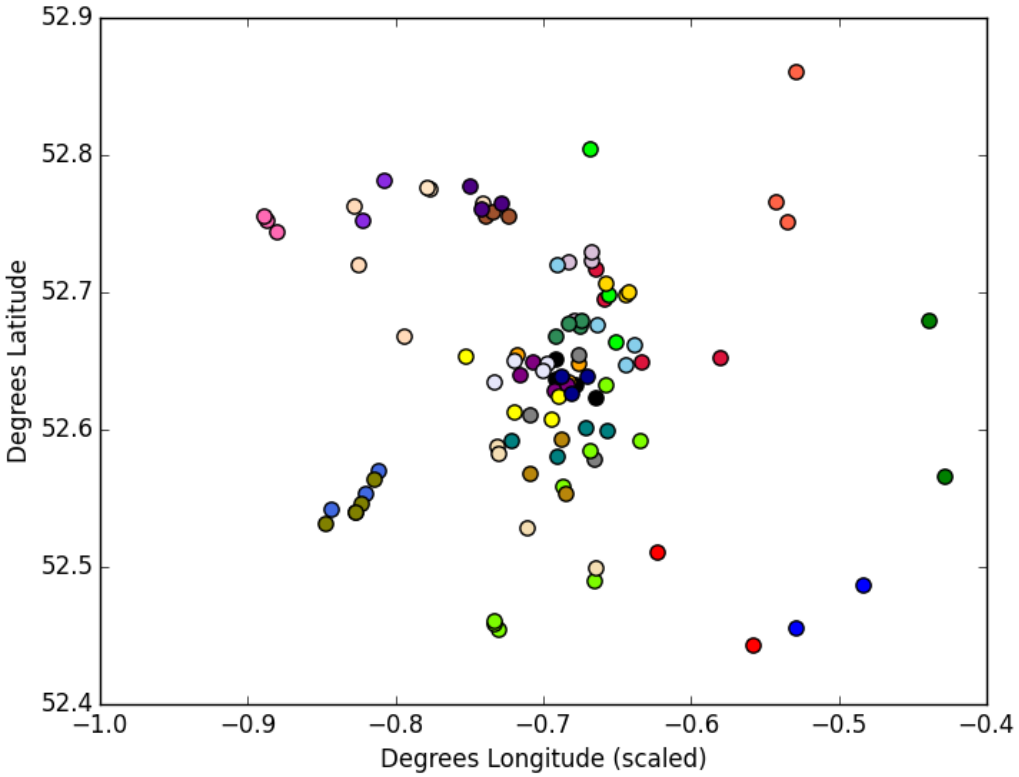

Figure 3: Spectral k-Means and Monte Carlo Clustering.

While points of the same color appear more closely clustered together, there is still substantial room for improvement. For example, the blue and dark green clusters in the bottom-left of Figure 3 should not overlap. Instead, the bottom-left three points could form one cluster, and the four points above them and to their right could form another cluster.

We designed the clustering refinement algorithm to deal with such concerns and quickly found that it outperforms all of our other methods. Figure 4 shows a clustering found using refinement.

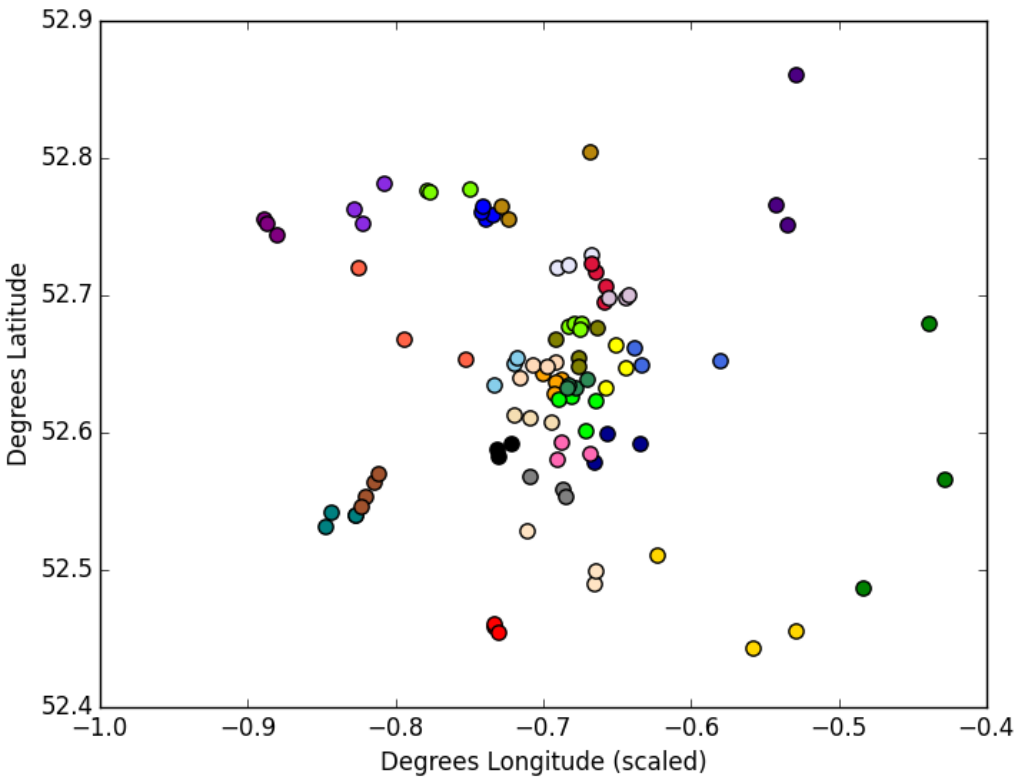

Figure 4: Clustering Refinement.

We were surprised to find that refining different initial clusterings often led to different results. So we asked how and whether we should combine refinement with our other methods. Should we generate lots of random clusterings and refine the best one? Or should we generate fewer random clusterings and refine each?[5] We addressed this question using a non-parametric hypothesis test.[6] If the mean intracluster distance produced by each method follows the same distribution, then the number of address lists where the second method outperforms the first should be a binomial random variable with sample size $n = 10$ and probability $p = 0.5$.

[5]Recall that we will devote the same amount of runtime to each method. Refinement is much slower than random clustering generation. We could randomly generate about $50,000$ random clusterings in 10 seconds, but we could only refine about 20 clusterings in that time.

[6]We make no assumptions about the distributions of data returned by each algorithm.

Table 1: Mean Intracluster Distances— Test 1

Address List	Refining Best Random Clustering	Refining All Random Clusterings
1	1.95	1.73
2	1.92	1.75
3	1.97	1.99
4	2.39	2.08
5	2.18	1.99
6	2.04	1.79
7	2.03	1.77
8	2.34	2.01
9	2.15	2.17
10	1.85	1.59

But, the second method outperformed the first on eight of our ten address lists; the second method should only have a 9.9% chance of performing so well if the two methods are truly equivalent. We infer that it is likely better to refine as many random clustering as possible in a given period of time rather than to refine only the best random clustering which could be generated in the same length of time.

We then asked whether it would help to begin by splitting patients into large group using spectral k-means, randomly assign patients in the same group to clusters, then refine the resulting clustering. Since we would only need to use the spectral k-means method once, we could still generate and refine about as many clusterings as before. Moreover, each of these clusterings should have a lower mean intracluster distance then a completely random clustering.

Table 2: Mean Intracluster Distances— Test 2

Address List	Refining All Spectral K-Means Clusterings	Refining All Random Clusterings
1	1.87	1.73
2	1.80	1.75
3	1.95	1.99
4	1.96	2.08
5	1.97	1.99
6	1.68	1.79
7	1.76	1.77
8	2.02	2.01
9	2.09	2.17
10	1.90	1.59

We performed the same non-parametric test and found that the algorithm which refined spectral k-means clusterings performed better than the algorithm which refined completely random clusterings on six of our ten address lists. There is a 37.7% chance that the first algorithm would perform at least this well if neither algorithm was any better than the other; so, we conclude that there is

insufficient evidence to conclude that the first algorithm performs any better than the second.

These results led us to question whether there is any advantage to refining a good initial clustering rather than a bad one. We tested an algorithm which generates thousands of random clusterings, find the one with the least mean intracluster distance, then refines it, against another algorithm which generates and refines a single random clustering. Although we would expect the first algorithm to perform better than the second algorithm, each algorithm performed better on five of our ten address lists.

Table 3: Mean Intracluster Distances—Test 3

Address List	Refining First Random Clustering	Refining Best Random Clustering
1	2.16	1.95
2	2.16	1.92
3	2.17	1.97
4	2.24	2.39
5	2.15	2.18
6	2.46	2.04
7	2.01	2.03
8	2.25	2.34
9	2.43	2.15
10	1.63	1.85

5 Recommendations

Since we do not find any clear benefit to refining a good initial clustering instead of a bad one, we recommend that PACE generate several random initial patient clusters, refine each, and keep the refined clustering with the least mean intracluster distance; we show Python code for this process in Section 6 (b). We cannot completely eliminate the possibility that refining initial clusterings generated using spectral k-means produces better final clusterings than refining randomly generated initial clusterings. Still, we recommend against this approach, since it leads to greater variation in cluster size.

We suggest that each PACE location input their current patient list and the number of caregivers they keep on permanent staff into this algorithm. Once the algorithm groups patients into clusters, management could try to match each permanent caregiver to a cluster with patients the caregiver already knows. Or, caregivers could be assigned to clusters near their own homes.

When a new patient joins PACE, management could check if a nearby caregiver has time to add the patient to their schedule. If not, we suggest PACE assign the patient to a caregiver hired to a temporary contract. As more patients join over time, PACE should find that a few patients still served by temporary caregivers live close to one another; at this point, we recommend management hire a new permanent caregiver to tend to these patients. If, in time, management finds that too many patients are being served by temporary caretakers, management could enter an updated patient list into the given algorithm and find a new clustering.

Assigning permanent caretakers to groups of patients who all live close to each another should reduce PACE's reimbursement costs considerably. To estimate how much money PACE might save, we used ARENA to simulate both a sample of routes generated using clustering refinement and a sample of routes generated randomly.

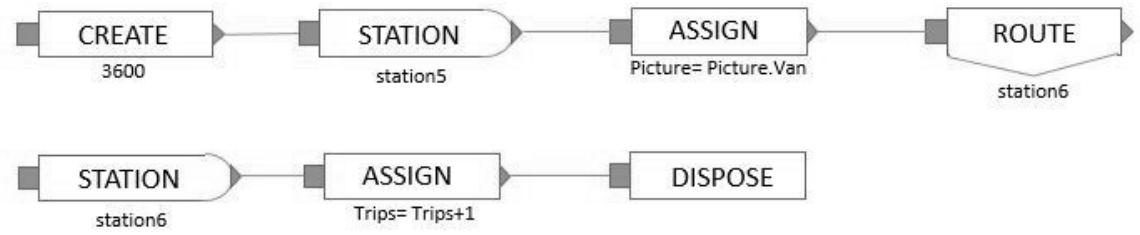

Figure 5: SIMAN Block Diagram for a Route

Savings should vary considerably depending on location and the current routes caregivers drive. But in our simulation, approximately one dollar was spent on mileage reimbursement after clustering refinement for every four dollars spent beforehand. Since the mileage reimbursement rate reflects both the cost of gasoline and fixed costs associated with automobile ownership, PACE could track actual savings to determine whether caregivers will need additional assistance to defray the cost of maintaining their vehicles. However, we believe such compensation would not be tax-deductible.

6 Appendix

a. Input Statements

(i) Model statements

```
BEGIN;


R1              CREATE          :3750;
                STATION,        station1;
                ASSIGN:         Picture = Picture.Van;
                ROUTE:          1000,station2;
                STATION,        station2;
                ASSIGN:         Distance = Distance + 5;
                ROUTE:          1000,station3;
                STATION,        station3;
                ASSIGN:         Distance = Distance + 5;
                ROUTE:          1800,station4;
                STATION,        station4;
                ASSIGN:         Distance = Distance + 9;
                ASSIGN:         Trips = Trips + 1;
                DISPOSE;

R2              CREATE          :2000;
                STATION,        station5;
                ASSIGN:         Picture = Picture.Van;
                ROUTE:          2000,station6;
                STATION,        station6;
                ASSIGN:         Distance = Distance + 10;
                ASSIGN:         Trips = Trips + 1;
                DISPOSE;

R3              CREATE          :400;
                STATION,        station7;
                ASSIGN:         Picture = Picture.Van;
                ROUTE:          400,station8;
                STATION,        station8;
                ASSIGN:         Distance = Distance + 2;
                ASSIGN:         Trips = Trips + 1;
                DISPOSE;

R4              CREATE          :400;
                STATION,        station9;
                ASSIGN:         Picture = Picture.Van;
                ROUTE:          400,station10;
                STATION,        station10;
```

```
ASSIGN:         Distance = Distance + 2;
ASSIGN:         Trips = Trips + 1;
DISPOSE;

CREATE          :400;
STATION,        station11;
ASSIGN:         Picture = Picture.Van;
ROUTE:          400,station12;
STATION,        station12;
ASSIGN:         Distance = Distance + 2;
ASSIGN:         Trips = Trips + 1;
DISPOSE;

CREATE          :1600;
STATION,        station13;
ASSIGN:         Picture = Picture.Van;
ROUTE:          1600,station14;
STATION,        station14;
ASSIGN:         Distance = Distance + 8;
ASSIGN:         Trips = Trips + 1;
DISPOSE;

CREATE          :600;
STATION,        station15;
ASSIGN:         Picture = Picture.Van;
ROUTE:          600,station16;
STATION,        station16;
ASSIGN:         Distance = Distance + 3;
ASSIGN:         Trips = Trips + 1;
DISPOSE;

CREATE          :400;
STATION,        station17;
ASSIGN:         Picture = Picture.Van;
ROUTE:          400,station18;
STATION,        station18;
ASSIGN:         Distance = Distance + 2;
ASSIGN:         Trips = Trips + 1;
DISPOSE;

CREATE          :600;
STATION,        station19;
ASSIGN:         Picture = Picture.Van;
ROUTE:          600,station20;
STATION,        station20;
ASSIGN:         Distance = Distance + 3;
```

```
                ASSIGN:         Trips = Trips + 1;
                DISPOSE;

END;
```

(ii) Experiment statements

```
BEGIN;
        PROJECT,        PACE Project, Ben and Brian;
        VARIABLES:      Distance, 0:
                        Trips, 0;
        PICTURES:       Picture.Van;
        ENTITIES:       Entity 1, Picture.Van;
        STATIONS:       station1:
                        station2:
                        station3:
                        station4:
                        station5:
                        station6:
                        station7:
                        station8:
                        station9:
                        station10:
                        station11:
                        station12:
                        station13:
                        station14:
                        station15:
                        station16:
                        station17:
                        station18:
                        station19:
                        station20;
        REPLICATE,      1,0,2000000,,,,,,Seconds;

END;
```

b. Python code

```
import numpy
import scipy
import scipy.linalg
import scipy.cluster.vq
import math
import numpy.random
import random
import matplotlib
import matplotlib.pyplot as plt
import time

start = time.clock()
current = start
TIME_ALLOWED = 10
NUM_NURSES = 30

#arrays to hold latitudes and longitudes of address GPS coordinates
lat = []
lng = []
addrs = []

#open file and parse gps coordinates into lists. this block will need to be
# changed if the input file for gps coordinates is different that ours. lat
#and lng should be a python list of floating point numbers.
#addrs is a list of strings

file = open('input.txt','r')
for line in file:
    lat.append(float(line.split("\",")[1].split(",")[0].strip()))
    lng.append(float(line.split("\",")[1].split(",")[1].strip()))
    addrs.append(line.split(", USA")[0][1:])

#length of this list == the number of patient addresses
length = len(lat)

#create distance matrix for addresses, scaled to reflect actual distances
#between points because latitude degrees vary in size
dist = numpy.empty([length,length])
for i in range(0,length):
    for j in range(0,length):
        dist[i][j] = math.sqrt((4761.0*(lat[i]-lat[j])**2) + (3025.0*(lng[i]-lng[j])**2))

#python list of integers [0,1,2,...,99]
indices = range(0,length)
```

```
#keep track of the best distance and grouping
min_distance = 9999999.9
min_clusters = []

#main loop of algorithm *********************************************
while (current-start < TIME_ALLOWED):
    my_cluster = indices[:] #deep copy
    random.shuffle(my_cluster) #shuffle in place

    #partition into NUM_NURSES lists of indices, guaranteed to be equal
    #length(at most off by one if not divisible evenly)
    division = len(my_cluster) / float(NUM_NURSES)
    base_list = [my_cluster[int(round(division * i)) : \
        int(round(division * (i + 1)))] for i in xrange(NUM_NURSES)]
        # e.g. [[3,83,17],[4,94,56],.....]

    pg_dist = []
    for i in range(0,NUM_NURSES):
        temp_dist = numpy.zeros([len(base_list[i])])
        for j in range(0,len(base_list[i])):
            for k in range(0,len(base_list[i])):
                if j != k:
                    temp_dist[j] += dist[base_list[i][j],base_list[i][k]]
        pg_dist.append(temp_dist)

    is_swapped = 1 #flag for swapping. if no swaps occur, loop will end
    while is_swapped == 1:
        is_swapped = 0
        for i in range(0,NUM_NURSES):
            for j in range(0,i):
                for k in range(0,len(base_list[i])):
                   for w in range(0, len(base_list[j])):
                        total1=-1.0*dist[base_list[i][k],base_list[j][w]]
                        total2=-1.0*dist[base_list[i][k],base_list[j][w]]
                        for g in range(0,len(base_list[i])):
                            total1+=dist[base_list[i][g],base_list[j][w]]
                        for g in range(0,len(base_list[j])):
                            total2+=dist[base_list[i][k],base_list[j][g]]
                        if (total1+total2<pg_dist[i][k]+pg_dist[j][w]):
                            is_swapped = 1
                            pg_dist[i][k],pg_dist[j][w]=total1,total2
                            base_list[i][k],base_list[j][w]=base_list[j][w],base_list[i][k]

    #check if this run outperformed previous runs
    average_distance = numpy.zeros([NUM_NURSES])
    pairs = 0
```

```
    for i in range(0,NUM_NURSES):
        route_size = int(len(base_list[i]))
        for j in range(0,route_size):
            for k in range(0,j):
                average_distance[i]+=dist[base_list[i][j],base_list[i][k]]
                pairs+=1
    if (sum(average_distance)/float(pairs)) < min_distance:
        min_distance = (sum(average_distance)/float(pairs))
        min_clusters = base_list

    current=time.clock()
#end of main loop ****************************************************
#ouput to file
target = open("clusters.txt", "w")
target.write("Mean Intracluster Distance: " + str(min_distance))
target.write("\n")
target.write("Clusters: ")
target.write("\n")
for item in min_clusters:
    for index in item:
        target.write(str(index) + " "*(6-len(str(index))) + "-    ")
        target.write(str(addrs[index]))
        target.write("\n")
    target.write("\n")
target.close()

#output graph of best results. list of colors for graphing clusters
colors = ['red','blue','green','black','orange','purple','grey','teal', \
        'yellow','darkblue','chartreuse','darkgoldenrod','lavender','crimson', \
        'tomato','thistle','lime','skyblue','gold','seagreen','hotpink','bisque', \
        'sienna','blueviolet','indigo','peachpuff', 'royalblue','lawngreen', \
        'olive','wheat','saddlebrown']
for i in range(0,length):
    lng[i] = lng[i] / 1.255 #scale so graph is proportional

#add points to graph
for i in range(len(min_clusters)):
    y = [lat[ind] for ind in min_clusters[i]]
    x = [lng[ind] for ind in min_clusters[i]]
    plt.scatter(x, y, s = 55,c = colors[i%len(colors)])
    plt.ylabel('Degrees Latitude')
    plt.xlabel('Degrees Longitude (scaled)')
    plt.gca().invert_xaxis()
plt.savefig('clusters.png')

print "Done! Browse to clusters.txt and clusters.png"
```

c. Bibliography

"Batch Geocoding." Batch Geocode. Web. 15 Dec. 2014. <http://www.findlatitudeandlongitude.com/batch-geocode>.

Leary, Dianne P. Scientific Computing with Case Studies. Philadelphia: Society for Industrial and Applied Mathematics, 2009. Print.

Ng, Andrew, Michael Jordan, and Yair Weiss. "On Spectral Clustering: Analysis and an Algorithm." Web. 15 Dec. 2014. <http://ai.stanford.edu/~ang/papers/nips01-spectral.pdf>.

Pegden, Claude Dennis, and Robert E. Shannon. Introduction to Simulation Using SIMAN. 2nd ed. New York: McGraw-Hill, 1995. Print.

"Random Addresses Generator." Random Addresses Generator. Web. 15 Dec. 2014. <http://www.doogal.co.uk/RandomAddresses.php>.

"Stack Overflow." Stack Overflow. Web. 15 Dec. 2014. <http://stackoverflow.com/>.

"Welcome to Python.org." Python.org. Web. 15 Dec. 2014. <https://www.python.org/>.

Von Luxburg, Ulrike. "A Tutorial on Spectral Clustering." Web. 15 Dec. 2014. <http://www.kyb.mpg.de/fileadmin/user_upload/files/publications/attachments/luxburg06_TR_v2_4139[1].pdf>.