Algorithms Course Project

UML Shuttle Service

12/11/2020

1 Team Info

Name: UML Shuttle Service
Members:

1. Parisa Hajibabaee, parisa_hajibabaee@student.uml.edu
2. Naman Mody, naman_mody@student.uml.edu

2 Problem Description

(A description of your problem and its relation to a real-world application (10 pts).)

With the massive spread of COVID-19, some private companies asked their employees to take a PCR test every day before coming to work and starting their daily routine. In order to do that they should make contracts with some private laboratories in their vicinity offering a Covid-19 test. We plan to develop a shuttle system that can pick up the staff from their homes (origins) every morning on weekdays and drop them to the lab (destination) to take the test in a very short time.
In this problem, we use the word "patient" referring to "potential infected staff".

2.1 Mathematical Modeling

Two important subproblems must be considered: Patient-to-Lab Assignment problem and Shuttle Routing Problem.

2.1.1 Patient-to-Lab Assignment

In the patient-to-lab assignment problem, patients should be assigned to district labs having some known test services and capacities such that total patient-to-lab distance is minimized. We will use distance as a surrogate measure for

shuttle riding time, since in this phase, the direct distance between each patient's home and his/her lab will be used in the model without any consideration of shuttle routing. Even though vehicle routing is not included in this model, the result of this phase will help us gauge how much improvement may be possible under "smarter" assignment decisions. The patient-to-lab assignment would be a mixed-integer model formulated to minimize the total direct distance that all patients would travel in a straight line (without considering routing) from each of their origins to reach their designated labs.

Here, following notations should be introduced.

L	Set of labratories, indexed by i
P	Set of patients, indexed by j
d_{ij}	Distance from patient j place of residence to lab i (miles)

In this subproblem, we want to determine the patient-to-lab assignments that minimize the total direct distance between patient homes and labs. This decision is captured via the decision variable x_{ij} which equals 1 if patient j is assigned to lab i, otherwise, $x_{ij} = 0$. So, our objective function and constraints would be as follows:

$$Minimize \sum_i \sum_j x_{ij} d_{ij} \tag{1}$$

$$S.t. \sum_i x_{ij} = 1 \qquad j \in P, \tag{2}$$

$$x_{ij} = \{0, 1\} \qquad i \in L, j \in P. \tag{3}$$

2.1.2 Shuttle Routing Problem

The vehicle routing problem (VRP) model would be a mixed-integer programming model that minimizes the total travel distance driven by all the shuttles while picking up all patients and dropping them to their intended labs/destinations. forasmuch as a graph consisting of nodes and links between them is a fair representative of most problems in computer science, we will use it in our model. Patients and labs should be considered as nodes, and edges would be the different routes between patients and labs. In this network, shuttles start their travel from an origin node/depot.

We should note that these two problems are dependent because the optimal assignment of patients to labs, and its output will, in turn, be used as an input parameter in the shuttle routing problem model.

The first subproblem which was presented earlier recommends the optimal assignment of patients to labs. The output of first subproblem will be used as an input parameter in the second subproblem. Based on the given description,

we define five sets for our second model:

L	Set of labratories, indexed by i and j
P	Set of patients, indexed by i and j
D	Set of depots, indexed by i and j
N	Set of nodes, which is union of L, P, and D, indexed by i and j
S	Set of shuttles, indexed by k
d_{ij}	Distance from node i to node j
c_k	Capacity of shuttle k
a_{ij}	1 if patient i is assigned to lab j, 0 otherwise

The primary decision variable in this problem is x_{ijk}, which equals 1 if node i is immediately followed by node j on shuttle route k, otherwise $x_{ijk} = 0$. In order to formulate the model, some bookkeeping variables are required. First, we introduce bookkeeping variable y_{ik} to record which patient is served by which shuttle: $y_{ik} = 1$ if patient i is served by shuttle k; otherwise, $y_{ik}= 0$. Bookkeeping variable z_{ik} keeps track of which lab is visited by which shuttle: $z_{ik} = 1$ if lab i is visited by shuttle k; otherwise, $z_{ik} = 0$. Also, bookkeeping variablew_{ik} shows the position of each node on each shuttle route. For example, if patient A is the third patient visited by shuttle Z and shuttle Z has not visited any lab yet, then $w_{AZ} = 4$, as the shuttle depot is always the first node to be visited by any shuttle. In addition to keeping track of the position of the nodes visited by each shuttle, bookkeeping variable w_{ik} also serves the purpose of eliminating any possible sub-tours traveled by each w_{ik} (sub-tour elimination requirement). Here, we can formulate our problem with the goal of minimization of total distances travelled by all shuttles, the objective function for shuttle routing problem would be as follow:

$$Minimize \sum_{i \in N} \sum_{j \in N} \sum_{k \in S} x_{ijk} d_{ij} + \frac{1}{M} \sum_{i \in N} \sum_{k \in S} w_{ik} \tag{4}$$

This objective function has two terms. The first term is dealing with the total distance traveled by all shuttles, whereas the second term makes sure that bookkeeping variable w_{ik} is not unnecessarily inflated. Here, we should use a very small constant multiplier on our second objective function term so as to not adversely impact the value of the overall objective function.

Now, constraint sets must be introduced for this subproblem.

$$\sum_{j \in N} \sum_{k \in S} x_{ijk} = 1 \qquad\qquad i \in P \tag{5}$$

This constraint forces each shuttle to visit exactly one node immediately after visiting a patient node. This is necessary, as it is not possible to pick up a patient at the end of a shuttle's routing plan travel at a minimum, this patient must be delivered to lab. Therefore, it follows that there should be either another patient or a lab visited after any patient visit.

$$\sum_{i \in N} \sum_{k \in S} x_{ijk} = 1 \qquad j \in P \tag{6}$$

Constraint (6) makes sure that exactly one node is visited before each patient visit. This constraint, in concert with constraint set (5), forces each patient origin to be visited once, with exactly one arc going into and exactly one arc going out of each patient node.

$$\sum_{i \in N} \sum_{k \in S} x_{ijk} \geq 1 \qquad j \in L \tag{7}$$

Constraint set (7) guarantees that there is at least one patient visited before any lab is visited. This insures that a lab is not the first place to be visited in any shuttle route, as there would be no patients onboard to be dropped off at the lab. It should be noted that this constraint set allows more than one shuttle to visit each lab.

$$\sum_{j \in N} x_{ijk} \leq 1 \qquad i \in L, k \in S \tag{8}$$

Constraint set (8) insures that at most one node is visited immediately after each lab visit by any shuttle. The visited node can be either a patient node, another lab node, or the final depot destination node when all the patients are dropped off.

$$\sum_{i \in P} \sum_{j \in N} x_{ijk} \leq c_k \qquad k \in S \tag{9}$$

Constraint set (9) makes sure that capacity of each shuttle is not exceeded:

$$\sum_{j \in N} x_{ijk} = y_i k \qquad i \in P, k \in S \tag{10}$$

$$\sum_{j \in N} x_{jik} = y_i k \qquad i \in P, k \in S \tag{11}$$

Constraint sets (10) and (11) are valid equalities that update bookkeeping variable y_{ik} by relating it to the main decision variable, x_{ijk}.

$$\sum_{i \in D} \sum_{j \in P} x_{ijk} = 1 \qquad k \in S \tag{12}$$

Constraint set (12) forces all shuttles to start their daily trips from the origin depot node.

$$\sum_{i \in L} \sum_{j \in D} x_{ijk} = 1 \qquad k \in S \tag{13}$$

Constraint set (13) insures all shuttles end their daily trips at the depot.

$$\sum_{j \in L} \sum_{k \in S} x_{ijk} = 0 \qquad i \in D \tag{14}$$

Constraint set (14) guarantees that no shuttle goes directly from the depot to a lab, as no patient(s) would have been picked up.

$$\sum_{i \in P} \sum_{k \in S} x_{ijk} = 0 \qquad j \in D \tag{15}$$

Constraint set (15) verifies that no shuttle goes directly from a patient node to the depot, as no lab drop-off would have occurred.

$$\sum_{j \in N} x_{ijk} = z_{ik} \qquad i \in L, k \in S \tag{16}$$

$$\sum_{j \in N} x_{jik} = z_{ik} \qquad i \in L, k \in S \tag{17}$$

Constraint sets (16) and (17) update bookkeeping variable z_{ik} by relating it to the main decision variable x_{ijk}.

$$x_{iik} = 0 \qquad i \in N, k \in S \tag{18}$$

Constraint set (18) ensures that there is no return travel from a node to back itself.

$$w_{ik} = 1 \qquad i \in D, k \in S \tag{19}$$

Constraint set (19) sets bookkeeping variable z_{ik} for the origin depot node = 1, thereby forcing the depot to be the first node visited by each shuttle.

$$w_{ik} \geq w_{jk} + 1 - (1 - x_{jik}) * M \qquad i \in L \cap P, j \in N, k \in S \tag{20}$$

$$a_{ji} w_{ik} \leq w_{jk} \qquad i \in P, j \in L, k \in S \tag{21}$$

Constraint sets (20) and (21) update bookkeeping variable w_{ik} and together, disallow sub-tours in the subproblem 2 routing model. Constraint set (21) guarantees that patients are picked up by a shuttle before that same shuttle visits their destination lab.

$$\sum_{j \in P} a_{ij} y_{ik} \geq z_{ik} \qquad i \in L, k \in S \tag{22}$$

Constraint sets (22) is the last valid inequality that insures no patient is on a shuttle that does not visit his/her destination lab.

2.2 Greedy Algorithm/Heuristic Solution Development

We successfully formulated our problems which is categorized as NP-hard problem. in this section, we are going to use greedy algorithm to construct a feasible solution that is definitely done under polynomial time constraint. Following input must be considered in our proposed algorithm:

1. Number of patients
2. Number of labs
3. Number of shuttles
4. Shuttle capacities
5. Distance matrix (between all patients and labs)

Following procedure should be taken to guarantee that a feasible solution will obtain. At fist step, we need to assign patients to the labs, and then each shuttle is needed to visit all required labs for patients drop off.

1. Patient-to-Lab Assignment:
   (a) Let P denote the set of all patients assigned to a lab. Initially, P is empty.
   (b) Let P' denote all patients not yet assigned to a lab. Initially, P' contains all students.
   (c) Let N_s denote the last visited network node of shuttle s. Initially, N_s is set to the depot for all shuttles.
   (d) If P' is empty, go to Step 2. Otherwise,
      i. Find the patient p in P' that lives closest to any node N_s (the current location of each shuttle s) for shuttle s that has remaining capacity to take on more patients.
      ii. Assign patient p to shuttle s. Update N_s to reflect the network node associated with patient p's house. Remove p from P'. Add p to P. Go to Step 1d.
2. Shuttle Routing Problem:
   (a) For each shuttle, determine the labs which need to be visited for dropping off each patient assigned to the shuttle. Let L_s denote the set of destination labs to be visited by shuttle s. Initially, L_s contains all labs attended by the patients on shuttle s.
   (b) If L_s is empty for all shuttles, STOP. Otherwise,
      i. Find the lab l in L_s that is closest to any node N_s (the current location of each shuttle s) for each shuttle s.

ii. Assign shuttle s to travel to lab l by updating N_s to reflect the network node associated with lab l. Remove e from L_s. Go to Step 2b.

Our heuristic solution will give us two main results: the total distance that all shuttles traveled and the order of visited nodes by each shuttle. Let's consider an example problem including seven patients(p_1,p_2,...p_7), two labs (l_1, l_2), and three shuttles(s_1,s_2,s_3). Following table shows the output of our proposed/-constructive greedy approach. It should be noted that each shuttle starts from origin node "Origin" and all seven patients are picked up and dropped off to the designated labs.
Route for first shuttle (s_1): Origin-p_1-p_2-p_5- l_2-l_1
Route for second shuttle (s_2): Origin-p_7-p_6-l_1
Route for third shuttle (s_3): Origin-p_4-p_3-l_2
****(Just first shuttle needs to meet both labs in its route.)

There is a high probability that this greedy approach might become inefficient in finding the minimal total distances between patients and labs. We can improve this greedy solution by focusing on the way patients are assigned to shuttles from unassigned pool P' and the placement of labs in the shuttle route. In our greedy algorithm, a patient was assigned to each shuttle in each iteration of 1d. Now, we can assign patients to only one single shuttle at a time. When the number of patients on the shuttle reaches capacity, the shuttle is removed from further consideration, and the next empty shuttle is used for patients assignment.

Considering the step 1 of greedy algorithm, we also want to improve the placement of lab visits on each shuttle's route. In the greedy heuristic, labs are visited at the end of each shuttle's route, regardless of when and where the last patient is picked up—this could lead to a missed opportunity for earlier patient drop-off. For example, consider the case of 20 patients and two labs (l_1 and l_2) being served by a single shuttle. If only three patients are destined for lab l1, and these same patients are picked up at the beginning of the route, there might be a chance that lab l1 can be visited at some point earlier in the route in a way that reduces the total distance traveled. In order to identify this opportunity, we need to perform an additional step after assigning all patients to labs which identifies the earliest position that each lab can be assigned in each shuttle's route. Then, when performing the main shuttle assignment loop, we can assess lab placement in each shuttle route from this earliest point to the end of the shuttle's route. So, we had two improvements on our first greedy algorithm.

3 Algorithms

(A description of the algorithms you will use to solve the problem. You must use one or more algorithms learn from this course. More is preferred (20 pts).)

Here, we briefly describe the algorithms that we used to solve the problems. In the previous section, we fully explained the algorithms and steps that we took.

1. Greedy Algorithms

   Due to the NP-hard complexity of vehicle routing models, several heuristics and approximation algorithms that quickly find good solutions, have been devised. we focused on developing practically doable heuristic solution based on greedy algorithms.

2. Vehicle Routing Problem (VRP)

   VRP is a combinational optimization and integer programming problem which can help us to answer the question of our project "What is the optimal set of routes for shuttles to traverse in order to pick up patients from their homes/origins and drop them to a given set of labs/destinations?". VRP is known as a technique that generalises the popular travelling salesman problem (TSP).

3. Mixed Integer Linear Programming (MILP)

   To formulate both subproblem mentioned earlier, we used Integer Linear Programming with minimization objective functions. Linear optimization problems that urge some of the variables to be integers are called Mixed Integer Linear Programming (MILP).

4 Time Complexity

A theoretical analysis of the time complexity of your algorithms (5 bonus points for providing space complexity analysis) (10 pts) .

VRP belongs to the class of NP hard optimization problem that can be exactly solved only for small instances of the problem. Although the heuristic methods does not guarantee optimal solutions, it yields best results in practice within a reasonable time.

4.1 Optimization problem-MILP

Our first solution for this problem was to devise an exact algorithm that utilizes mixed integer linear programming. Due to the nature of the algorithm on trying all permutations (ordered combinations) and seeing which one is the shortest, the running time using this approach lies with a polynomial factor of O(n!) (this is the factorial of the number of locations).

The locations based on the longitudes and latitudes of the location is converted to a distance matrix of size n^2, where n is the number of locations. Using this matrix it then computes the routing for these locations. Therefore, this is the maximum amount of space that gets utilized by the algorithm, which is O(n^2).

4.2 Heuristic/approximation algorithms

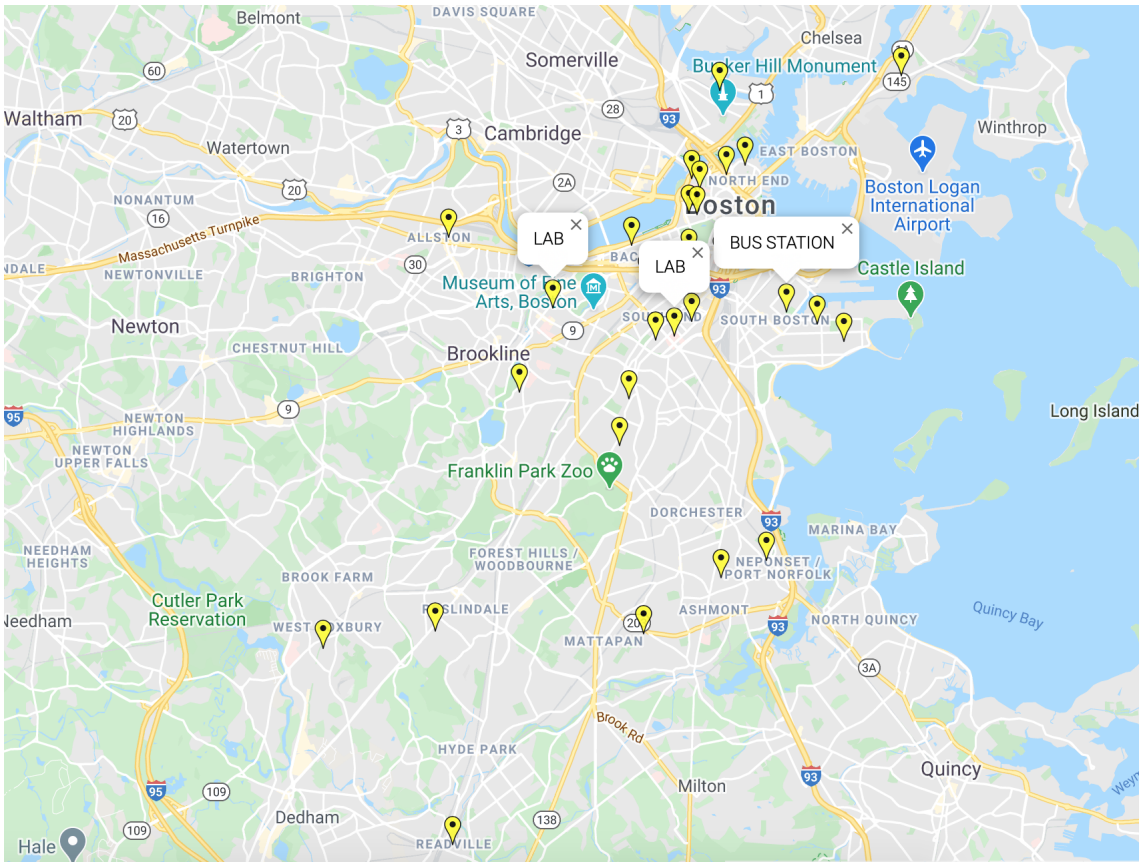
Constructive heuristics is one of the main categories of heuristics. For example,the nearest neighbour (NN) algorithm (a greedy algorithm) lets the salesman choose the nearest unvisited city as his next move. This algorithm quickly yields an effectively short route. the NN algorithm has the worst-case time complexity of $\Theta(n^2)$ and the worst-case space complexity of $\Theta(n)$.

5 Dataset

A description of your dataset (5 pts).

Since the problem is pertaining to the current affairs we thought of simulating the routes for one of the cities in America and chose Boston. We also took into consideration the vehicular guidelines as per the mandated rules of the city. Based on the guidelines we found that normal shuttle a normal shuttle in Boston can hold about 83 passengers per ride. However, due to the pandemic they have restricted the number to 27 passengers in order to maintain the required social distancing norms. Therefore, we chose the number of shuttles and the number of patients based on these constructs. The dataset was also dependent on whether the route could be formulated within a maximum run time limit of 1 hour to ensure a reliable fast heuristic.

Based on these conditions we found a dataset that comprises of 28 locations within Boston city in which, 25 are where patients reside, 2 are Labs and one shuttle depot. Below is an image of the locations mapped with labels for the Labs and shuttle depot.



The area covered by these data markers is 53.64 miles/ 86.33 kilometers.

We chose the patient's data points based on the 23 neighborhoods of Boston and these were randomly picked. In contrast, the Labs and shuttle are actual covid testing labs and a shuttle depot, respectively. Using these coordinates we produced a distance matrix and simulate the routing for these points.

6 Python Implentation

(A successful implementation of the algorithms using a high-level language of your choice (C, C++, Java, or Python)(40 points))

The source code of our solution implemented in Python is provided with this pdf file.

Distance Matrix

This creates a distance matrix for the dataset from locational latitudes and longitudes, before getting fed to the routing algorithm.
The latitudes and longitudes are separated in two lists 'lat' and 'lon', respectively. Additionally, the values are placed in the order of Depot first, Lab/s second and following that with random placement of patients residences.

```
cities_df = pd.DataFrame({
    'lat':[42.339561,
           42.334964, 42.340213,
           42.365575, 42.364762, 42.358015, 42.349070, 42.352120,
    42.334305, 42.33393, 42.292385, 42.314153, 42.324212,
    42.278374, 42.278848, 42.275617, 42.238242, 42.381620,
    42.353696, 42.367374, 42.362756, 42.358296, 42.349981,
    42.345365, 42.337827, 42.337150, 42.289079, 42.322991],
    'lon':[-71.044389,
           -71.073309,-71.104623,
           -71.059846, -71.068748, -71.067547, -71.061297,
    -71.084282, -71.078024, -71.029584, -71.04947, -71.087541,
    -71.113242,-71.081160, -71.134715, -71.163523, -71.130200,
    -71.061717, -71.131247, -71.055093, -71.066741, -71.069387,
    -71.069446, -71.080479, -71.068844, -71.036429, -71.061430,
    -71.084816]
                          })

cities_df['lat'] = np.radians(cities_df['lat'])
cities_df['lon'] = np.radians(cities_df['lon'])

dist1 = DistanceMetric.get_metric('haversine')
dist = dist1.pairwise(cities_df [['lat','lon']].to_numpy())*6373
```

Routing Algorithm

This has been programmed using Gurobi, which is an advanced optimization solver for mathematical programming one of them being the mixed integer linear program, which we have used to solve this problem. The way the algorithm works is that the distance matrix gets split in to its components before being fed to the different constraints, which are the formulated constraints seen in the earlier section and the objective function, i.e., to minimize the total distances travelled by all shuttles, is programmed using the Gurobi framework. The optimizer once finished outputs the patients being assigned to which shuttle, the labs visited by which shuttle, each node being followed by which other node that can then be retraced to show the route taken and the elimination of sub-tours.

```
#!/usr/bin/env python
# coding: utf-8

# In[2]:


import numpy as np
import random
```

```
import gurobipy as gp
import time
from gurobipy import GRB
from gurobipy import quicksum
from sklearn.neighbors import DistanceMetric
from math import radians
import pandas as pd
import csv

with open('/Users/naman/Desktop/lat_lon.csv') as f:
    lat_lon = [[],[]]
    reader = csv.reader(f)
    for row in reader:
        for col in range(2):
            lat_lon[col].append(row[col])


lat_dataset     = [42.339561, 42.334964, 42.340213]
lon_dataset     = [-71.044389, -71.073309, -71.104623]
r = random.SystemRandom()

list1 = [[0,1,1,0,1,0,1,0,0,1,1,0,1,1,0,1,0,0,0,1,1,0,1,1,0],
         [1,0,0,1,0,1,0,1,1,0,0,1,0,0,1,0,1,1,1,0,0,1,0,0,1]]

print("Below is a 2 step selection process,")
print("1) Select your preference to choose between default (
    original 28 datapoint set) or choose the number of patients (
    algorithm will randomly pick patient locations from the
    multiple locations in the csv file)")
print("2) If you have chosen the number of patients, there will be
    a prompt that will ask you to select a number between 2 and 25"
    )

input_select = int(input("Please select your prefrence to choose
    the number of patients (1) or default (0):"))
while(1):
    if(input_select > 1 or input_select < 0):
        input_select = int(input("Please select your prefrence to
    choose the number of patients (1) or default (0):"))
    else:
        break

if(input_select == 1):

    p = int(input("Please enter the number of patients (between 2
    and 25):"))         #Number of patients
    while(1):
      if(p > 25 or p < 2):
          p = int(input("Please enter the number of patients (
    between 2 and 25):"))
      else:
          break

    for i in range(p):
        n = r.randint(0,len(lat_lon[0])-1)
        if lat_lon[0][n] not in lat_dataset:
            lat_dataset.append(float(lat_lon[0][n]))
```

```
            lon_dataset.append(float(lat_lon[1][n]))

    cities_df = pd.DataFrame({'lat':lat_dataset, 'lon':lon_dataset})
    print(cities_df)

    list2 = [[],[]]
    for i in range(len(list1)):
        for j in range(p):
            list2[i].append(list1[i][j])
    a = np.array(list2)

elif(input_select == 0):
    p = 25
    cities_df = pd.DataFrame({
    'lat':[
           42.339561,
           42.334964, 42.340213,
           42.365575, 42.364762, 42.358015, 42.349070, 42.352120, 42.334305, 42.33393, 42.292385, 42.314153, 42.324212,
           42.278374, 42.278848, 42.275617, 42.238242, 42.381620, 42.353696, 42.367374, 42.362756, 42.358296, 42.349981,
           42.345365, 42.337827, 42.337150, 42.289079, 42.322991],
    'lon':[-71.044389,
           -71.073309,-71.104623,
           -71.059846, -71.068748, -71.067547, -71.061297, -71.084282, -71.078024, -71.029584, -71.04947, -71.087541, -71.113242,
           -71.081160, -71.134715, -71.163523, -71.130200, -71.061717, -71.131247, -71.055093, -71.066741, -71.069387, -71.069446,
           -71.080479, -71.068844, -71.036429, -71.061430, -71.084816]
                           })
    a = np.array(list1)

l = 2             # Number of labs
M = 1000          # Big M
s = 2             # Number of shuttles

cities_df['lat'] = np.radians(cities_df['lat'])
cities_df['lon'] = np.radians(cities_df['lon'])

dist1 = DistanceMetric.get_metric('haversine')
dist = dist1.pairwise(cities_df [['lat','lon']].to_numpy())*6373

C = [27,27]                                     # capacity of shuttles
D = [0]                                         # Depot
L = [i for i in np.arange(1,l+1,1)]             # Set of Labs
P = [i for i in np.arange(l+1,l+1+p,1)]         # Set of Patients

LP = set(L)|set(P)
N  = set(D)|set(L)|set(P)               # set of all the nodes including depot
n  = len(N)                             # the number of nodes
S  = [i for i in np.arange(1,s+1,1)]    # Set of shuttles
```

```
104
105 m = gp.Model()
106
107 # Add variables
108
109 y={}
110 z={}
111 x={}
112 w={}
113
114 # To record which patient is served by which shuttle: yik = 1 if
    patient i is served by shuttle k; otherwise,yik = 0.
115 for i in P:
116     for k in S:
117         y[(i,k)] = m.addVar(lb=0, vtype=GRB.BINARY, name="patient:
    %d, shuttle: %d" % (i,k))
118
119 # To record which lab is visited by which shuttle: zik = 1 if lab i
     is visited by shuttle k; otherwise,zik = 0.
120 for i in L:
121     for k in S:
122         z[(i,k)] = m.addVar(lb=0, vtype=GRB.BINARY, name="lab: %d,
    shuttle: %d" % (i,k))
123
124 # The  primary  decision  variable. node i is immediately followed
    by node j on shuttle route k, otherwise 0.
125 for i in N:
126     for j in N:
127         for k in S:
128             x[(i,j,k)] = m.addVar(lb=0, vtype=GRB.BINARY, name="
    first_node: %d, next_node: %d, shuttle: %d" % (i,j,k))
129
130 # serves the purpose of eliminating any possible sub-tours traveled
     (sub-tour elimination requirement)
131 for i in N:
132     for k in S:
133             w[(i,k)] = m.addVar(lb=0, vtype=GRB.INTEGER, name="w%d
    ,%d" % (i,k))
134
135
136 m.update()
137
138 # Add constraint set 5
139 for i in P:
140     m.addConstr(quicksum(x[(i,j,k)] for j in N for k in S) == 1,
    name="c1-%d" % i)
141
142 # Add constraint set 6
143 for j in P:
144     m.addConstr(quicksum(x[(i,j,k)] for i in N for k in S) == 1,
    name="c2-%d" % j)
145
146 # Add constraint set 7
147 for j in L:
148     m.addConstr(quicksum(x[(i,j,k)] for i in N for k in S) >= 1,
    name="c3-%d" % j)
149
```

```
# Add constraint set 8
for i in L:
    for k in S:
        m.addConstr(quicksum(x[(j,i,k)] for j in N) <= 1,name="c4.%
    d,%d" % (i,k))

# Add constraint set 9
for k in S:
    m.addConstr(quicksum(x[(i,j,k)] for i in P for j in N) <= C[k
    -1],name="c5-%d" % k)

# Add constraint set 10
for i in P:
    for k in S:
        m.addConstr(quicksum(x[(i,j,k)] for j in N) == y[(i,k)],
    name="c6-%d,%d" % (i,k))

# Add constraint set 11
for i in P:
    for k in S:
        m.addConstr(quicksum(x[(j,i,k)] for j in N) == y[(i,k)],
    name="c7-.%d,%d" % (i,k))

# Add constraint set 12
for k in S:
    m.addConstr(quicksum(x[(i,j,k)] for i in D for j in P) == 1,
    name="c9-%d" % k)

# Add constraint set 13
for k in S:
    m.addConstr(quicksum(x[(i,j,k)] for i in L for j in D) == 1,
    name="c10-%d" % k)

# Add constraint set 14
for i in D:
    m.addConstr(quicksum(x[(i,j,k)] for j in L for k in S) == 0,
    name="c11-%d" % i)

# Add constraint set 15
for j in D:
    m.addConstr(quicksum(x[(i,j,k)] for i in P for k in S) == 0,
    name="c12-%d" % j)

# Add constraint set 16
for i in L:
    for k in S:
        m.addConstr(quicksum(x[(i,j,k)] for j in N) == z[(i,k)],
    name="c8.%d,%d" % (i,k))

# Add constraint set 17
for i in L:
    for k in S:
        m.addConstr(quicksum(x[(j,i,k)] for j in N) == z[(i,k)],
    name="c9.%d,%d" % (i,k))


# Add constraint set 18
```

```
for i in N:
    for k in S:
        m.addConstr(x[(i,i,k)] == 0, name="c10.%d,%d" % (i,k))

# Add constraint set 19
for i in D:
    for k in S:
        m.addConstr(w[(i,k)] == 1, name="c11.%d,%d" % (i,k))

# Add constraint set 20
for i in LP:
    for j in N:
        for k in S:
            m.addConstr(w[(i,k)]>=w[(j,k)]+1-M*(1-x[(j,i,k)]), name
    ="c12.%d,%d,%d" % (i,j,k))

# Add constraint set 21
for i in P:
    for j in L:
        for k in S:
            m.addConstr(a[(j-1,i-s-1)]*w[(i,k)] <= w[(j,k)], name="
    c14.%d,%d,%d" % (i,j,k))

# Add constraint set 22
for i in L:
    for k in S:
        m.addConstr(quicksum(a[(i-1,j-s-1)]*y[(j,k)] for j in P) >=
     z[(i,k)], name="c14.%d,%d" % (i,k))

# Objective Function

m.setObjective(quicksum(dist[(i,j)]*x[(i,j,k)] for i in N for j in
    N for k in S)+1/M*quicksum(w[(i,k)] for i in N for k in S),GRB.
    MINIMIZE)

m.optimize()

for v in m.getVars():
    if v.X != 0.0:
        print('%s %g' % (v.VarName, v.X))


# In[ ]:









# In[ ]:
```

7 Running Samples

Running Samples: inputs and outputs (5 pts).

The inputs are:

1. Number of patients (p) = 25

2. Number of labs (l) = 2

3. Number of shuttles (s) = 2

4. Latitude ('lat') = [42.339561, 42.334964, 42.340213, 42.365575, 42.364762, 42.358015, 42.349070, 42.352120, 42.334305, 42.33393, 42.292385, 42.314153, 42.324212, 42.278374, 42.278848, 42.275617, 42.238242, 42.381620, 42.353696, 42.367374, 42.362756, 42.358296, 42.349981, 42.345365, 42.337827, 42.337150, 42.289079, 42.322991]

5. Longitude ('lon') = [-71.044389, -71.073309,-71.104623, -71.059846, -71.068748, -71.067547, -71.061297, -71.084282, -71.078024, -71.029584, -71.04947, -71.087541, -71.113242,-71.081160, -71.134715, -71.163523, -71.130200, -71.061717, -71.131247, -71.055093, -71.066741, -71.069387, -71.069446, -71.080479, -71.068844, -71.036429, -71.061430, -71.084816]

The output is:

The distance matrix once it gets processed looks like the following image.

0	2.432	4.953	3.16	3.445	2.8	1.747	3.565	2.827	1.369	5.264	4.536	5.913	7.448	10.04	12.109	13.299	4.89	7.311	3.216	3.167	2.927	2.364	3.037	2.02	0.707	5.787	3.801
2.432	0	2.64	3.58	3.336	2.61	1.854	2.111	0.395	3.597	5.126	2.594	3.495	6.328	8.0296	9.9322	11.733	5.276	5.199	3.904	3.138	2.615	1.7	1.298	0.486	3.042	5.196	1.6337
4.953	2.64	0	4.637	4.019	3.63	3.696	2.133	2.284	6.209	6.991	3.221	1.916	7.144	7.2606	8.6659	11.536	5.801	2.653	5.07	3.998	3.526	3.089	2.066	2.954	5.617	6.706	2.5144
3.16	3.58	4.637	0	0.737	1.05	1.84	2.505	3.786	4.31	8.186	6.156	6.359	9.856	11.444	13.146	15.3	1.791	6.015	0.439	0.648	1.127	1.906	2.816	3.174	3.702	8.51	5.1623
3.445	3.336	4.019	0.737	0	0.76	1.85	1.899	3.472	4.704	8.205	5.837	5.807	9.663	10.989	12.612	14.953	1.962	5.282	1.159	0.277	0.721	1.645	2.363	2.996	4.061	8.44	4.8303
2.8	2.607	3.635	1.052	0.757	0	1.12	1.524	2.774	4.113	7.45	5.148	5.315	8.929	10.395	12.096	14.285	2.669	5.258	1.46	0.531	0.154	0.907	1.763	2.248	3.454	7.684	4.1464
1.747	1.854	3.696	1.84	1.85	1.12	0	1.92	2.142	3.104	6.38	4.443	5.088	8.031	9.8729	11.724	13.568	3.621	5.773	2.099	1.587	1.223	0.678	1.63	1.396	2.437	6.673	3.4862
3.565	2.111	2.133	2.505	1.899	1.52	1.92	0	2.047	4.931	7.235	4.232	3.912	8.207	9.1449	10.719	13.218	3.769	3.864	2.938	1.865	1.404	1.243	0.814	2.034	4.272	7.26	3.2403
2.827	0.395	2.284	3.786	3.472	2.77	2.142	2.047	0	3.983	5.221	2.374	3.106	6.227	7.7329	9.5959	11.515	5.431	4.878	4.133	3.298	2.761	1.881	1.247	0.85	3.435	5.212	1.3768
1.369	3.597	6.209	4.31	4.704	4.11	3.104	4.931	3.983	0	4.902	5.249	6.964	7.496	10.598	12.786	13.484	5.926	8.642	4.27	4.428	4.249	3.732	4.373	3.257	0.667	5.635	4.702
5.264	5.126	6.991	8.186	8.205	7.45	6.38	7.235	5.221	4.902	0	3.959	6.329	3.038	7.1744	9.5689	8.9681	9.977	9.578	8.354	7.955	7.512	6.614	6.421	5.3	5.093	1.051	4.4769
4.536	2.594	3.221	6.156	5.837	5.15	4.443	4.232	2.374	5.249	3.959	0	2.392	4.014	5.5213	7.5798	9.1444	7.799	5.68	6.493	5.67	5.132	4.254	3.52	3.049	4.92	3.52	1.0083
5.913	3.495	1.916	6.359	5.807	5.32	5.088	3.912	3.106	6.964	6.329	2.392	0	5.741	5.3461	6.8064	9.6637	7.662	3.598	6.775	5.744	5.232	4.602	3.577	3.952	6.478	5.782	2.3416
7.448	6.328	7.144	9.856	9.663	8.93	8.031	8.207	6.227	7.496	3.038	4.014	5.741	0	4.4077	6.7853	6.0187	11.59	9.336	10.13	9.46	8.942	8.023	7.452	6.69	7.502	2.013	4.9718
10.04	8.03	7.261	11.44	10.99	10.4	9.873	9.145	7.733	10.6	7.174	5.521	5.346	4.408	0	2.3979	4.5319	12.91	8.33	11.83	10.88	10.34	9.561	8.64	8.509	10.36	6.137	6.4
12.11	9.932	8.666	13.15	12.61	12.1	11.72	10.72	9.596	12.79	9.569	7.58	6.806	6.785	2.3979	0	4.9808	14.46	9.081	13.55	12.54	12.02	11.33	10.34	10.42	12.5	8.534	8.3484
13.3	11.73	11.54	15.3	14.95	14.3	13.57	13.22	11.52	13.48	8.968	9.144	9.664	6.019	4.5319	4.9808	0	16.91	12.84	15.64	14.8	14.26	13.4	12.6	12.17	13.44	8.001	10.14
4.89	5.276	5.801	1.791	1.962	2.67	3.621	3.769	5.431	5.926	9.977	7.799	7.662	11.59	12.911	14.46	16.914	0	6.504	1.675	2.138	2.67	3.576	4.317	4.906	5.365	10.29	6.7921
7.311	5.199	2.653	6.015	5.282	5.26	5.773	3.864	4.878	8.642	9.578	5.68	3.598	9.336	8.3302	9.0814	12.842	6.504	0	6.441	5.397	5.11	5.097	4.275	5.425	8.009	9.199	5.1223
3.216	3.904	5.07	0.439	1.159	1.46	2.099	2.938	4.133	4.27	8.354	6.493	6.775	10.13	11.825	13.553	15.636	1.675	6.441	0	1.086	1.549	2.266	3.217	3.475	3.695	8.724	5.5084
3.167	3.138	3.998	0.648	0.277	0.53	1.587	1.865	3.298	4.428	7.955	5.67	5.744	9.46	10.879	12.542	14.801	2.138	5.397	1.086	0	0.542	1.438	2.24	2.778	3.784	8.207	4.666
2.927	2.615	3.526	1.127	0.721	0.15	1.223	1.404	2.761	4.249	7.512	5.132	5.232	8.942	10.342	12.022	14.26	2.67	5.11	1.549	0.542	0	0.925	1.703	2.277	3.588	7.727	4.1268
2.364	1.7	3.089	1.906	1.645	0.91	0.678	1.243	1.881	3.732	6.614	4.254	4.602	8.023	9.5614	11.327	13.396	3.576	5.097	2.266	1.438	0.925	0	1.042	1.353	3.067	6.806	3.2572
3.037	1.298	2.066	2.816	2.363	1.76	1.63	0.814	1.247	4.373	6.421	3.52	3.577	7.452	8.6395	10.337	12.598	4.317	4.275	3.217	2.24	1.703	1.042	0	1.272	3.735	6.454	2.5141
2.02	0.486	2.954	3.174	2.996	2.25	1.396	2.034	0.85	3.257	5.3	3.049	3.952	6.69	8.5086	10.418	12.173	4.906	5.425	3.475	2.778	2.277	1.353	1.272	0	2.666	5.456	2.1091
0.707	3.042	5.617	3.702	4.061	3.45	2.437	4.272	3.435	0.667	5.093	4.92	6.478	7.502	10.364	12.496	13.438	5.365	8.009	3.695	3.784	3.588	3.067	3.735	2.666	0	5.729	4.2792
5.787	5.196	6.706	8.51	8.44	7.68	6.673	7.26	5.212	5.635	1.051	3.52	5.782	2.013	6.1371	8.5338	8.0013	10.29	9.199	8.724	8.207	7.727	6.806	6.454	5.456	5.729	0	4.2343
3.801	1.634	2.514	5.162	4.83	4.15	3.486	3.24	1.377	4.702	4.477	1.008	2.342	4.972	6.4	8.3484	10.14	6.792	5.122	5.508	4.666	4.127	3.257	2.514	2.109	4.279	4.234	0

Figure: Distance Matrix

NOTE: For the following figures the 1's at the end of each line are to be ignored.

```
patient: 3, shuttle: 2 1
patient: 4, shuttle: 2 1
patient: 5, shuttle: 2 1
patient: 6, shuttle: 2 1
patient: 7, shuttle: 2 1
patient: 8, shuttle: 1 1
patient: 9, shuttle: 1 1
patient: 10, shuttle: 1 1
patient: 11, shuttle: 1 1
patient: 12, shuttle: 1 1
patient: 13, shuttle: 1 1
patient: 14, shuttle: 1 1
patient: 15, shuttle: 1 1
patient: 16, shuttle: 1 1
patient: 17, shuttle: 2 1
patient: 18, shuttle: 2 1
patient: 19, shuttle: 2 1
patient: 20, shuttle: 2 1
patient: 21, shuttle: 2 1
patient: 22, shuttle: 2 1
patient: 23, shuttle: 2 1
patient: 24, shuttle: 1 1
patient: 25, shuttle: 1 1
patient: 26, shuttle: 1 1
patient: 27, shuttle: 1 1
lab: 1, shuttle: 1 1
lab: 2, shuttle: 2 1
```

Figure: This shows the patient to shuttle assignment and the shuttle to lab assignment

As you can see from the output each of the patients are accordingly placed in shuttle 1 and 2. These shuttles are then directed to any one of the labs based on the routes it take. Below we see the routes taken.

```
first_node: 0, next_node: 6, shuttle: 2 1
first_node: 0, next_node: 25, shuttle: 1 1
first_node: 1, next_node: 24, shuttle: 1 1
first_node: 2, next_node: 23, shuttle: 2 1
first_node: 3, next_node: 19, shuttle: 2 1
first_node: 4, next_node: 20, shuttle: 2 1
first_node: 5, next_node: 21, shuttle: 2 1
first_node: 6, next_node: 3, shuttle: 2 1
first_node: 7, next_node: 18, shuttle: 2 1
first_node: 8, next_node: 1, shuttle: 1 1
first_node: 9, next_node: 10, shuttle: 1 1
first_node: 10, next_node: 26, shuttle: 1 1
first_node: 11, next_node: 27, shuttle: 1 1
first_node: 12, next_node: 11, shuttle: 1 1
first_node: 13, next_node: 16, shuttle: 1 1
first_node: 14, next_node: 12, shuttle: 1 1
first_node: 15, next_node: 14, shuttle: 1 1
first_node: 16, next_node: 15, shuttle: 1 1
first_node: 17, next_node: 4, shuttle: 2 1
first_node: 18, next_node: 2, shuttle: 2 1
first_node: 19, next_node: 17, shuttle: 2 1
first_node: 20, next_node: 5, shuttle: 2 1
first_node: 21, next_node: 7, shuttle: 2 1
first_node: 22, next_node: 0, shuttle: 2 1
first_node: 23, next_node: 22, shuttle: 2 1
first_node: 24, next_node: 0, shuttle: 1 1
first_node: 25, next_node: 9, shuttle: 1 1
first_node: 26, next_node: 13, shuttle: 1 1
first_node: 27, next_node: 8, shuttle: 1 1
```

Figure: Routes taken by each of the shuttles

These routes are extrapolated on to the map we saw earlier and the following images provide each of the routes taken by the shuttles.

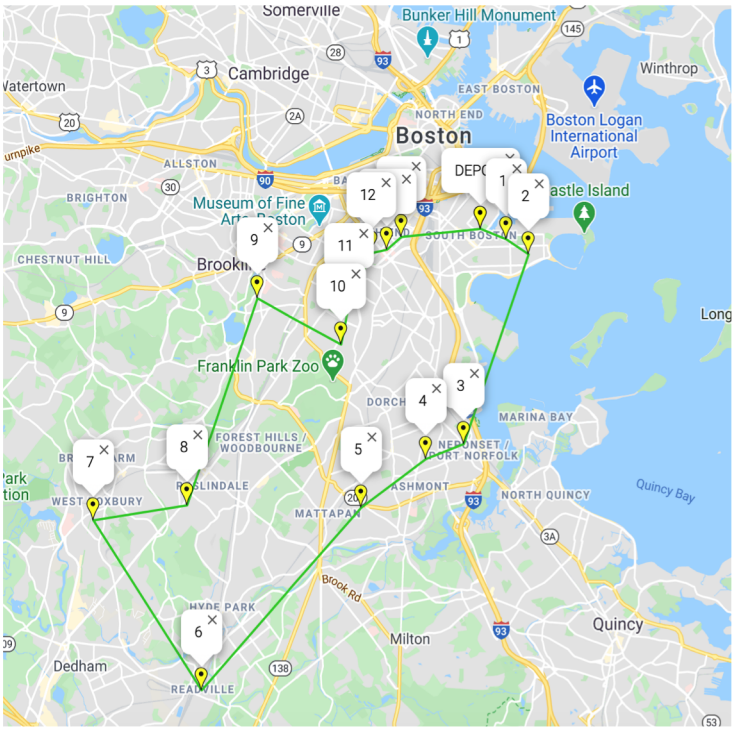


Figure: Route taken by shuttle 1

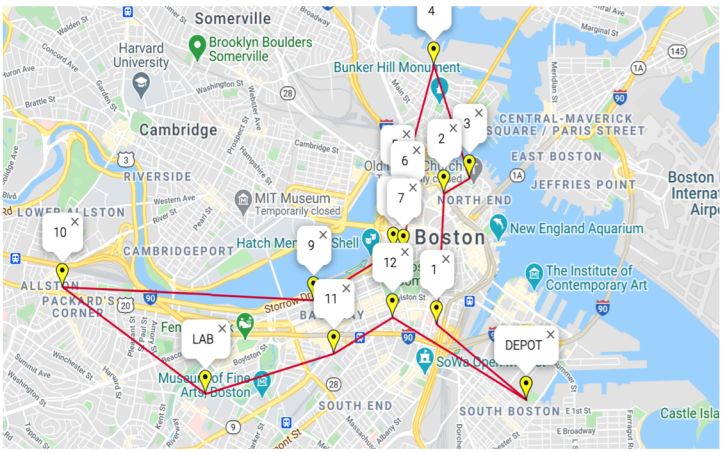


Figure: Route taken by shuttle 2

0 is the starting point and ending point of each route. And before each of the routes end, each of the shuttles drop the patients at the lab, which are nodes 1 and 2. However, we tried running this program on multiple data sets but each

time we didn't get the ideal output of the labs being the last stop before going back to the depot. This might be seen as an over optimization issue.

```
w0,1 1
w0,2 1
w1,1 14
w1,2 14
w2,1 15
w2,2 12
w3,2 3
w4,2 6
w5,2 8
w6,2 2
w7,2 10
w8,1 13
w9,1 3
w10,1 4
w11,1 11
w12,1 10
w13,1 6
w14,1 9
w15,1 8
w16,1 7
w17,2 5
w18,2 11
w19,2 4
w20,2 7
w21,2 9
w22,2 14
w23,2 13
w24,1 15
w25,1 2
w26,1 5
w27,1 12
```

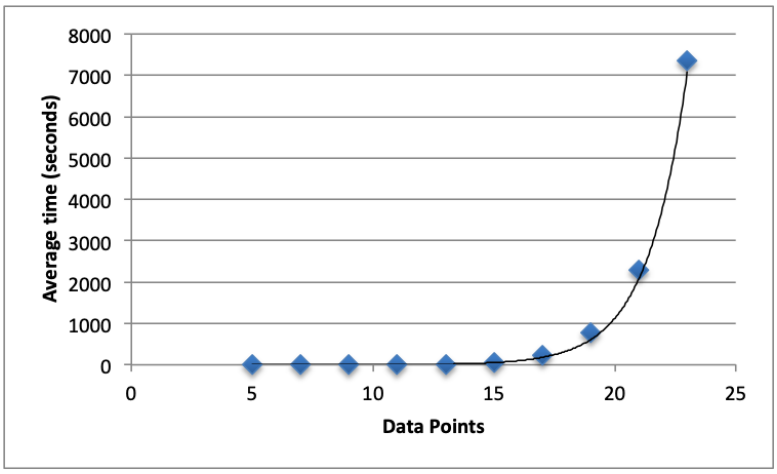
Figure: Sub-Tours Eliminated

8 Complexity Analysis of Running Samples

A line graph for time complexity analysis with the x-axis being the size of the inputs and the y-axis the actual running time incurred. Your must run your program on multiple randomly selected inputs of the same size and average the running time of that size. Please specify what computers you use (CPU and RAM) to run your programs (10 pts).

We used a 2.8GHz Intel Core i5 processor with 8GB RAM to run our program. To run our program on multiple randomly selected inputs we first collected a large number of coordinates (greater than 150 data points) from the Boston area in order to randomly produce smaller data sets. The number of data points ranged from 5 to 23 in increments of 2. The algorithm was then run for 10 cycles with each a cycle a new data set being generated. The start and end time for each of these runs were recorded before being averaged at the end of the 10 cycles. These averages were then plotted based on their respective

input sizes and below is the plot of what was observed.



From this graph we observe that as the number of data points increase even with increments of 2 affects the time complexity substantially. As like most vehicle routing problems we see that the trend is n!, which means the worst case time is O(n!).