HW1 ORIE 5250

Chenyang Yan cy477

Yutong Wang yw2364

Hongyi Nie hn327

In [1]:

```
import pandas as pd
from math import cos, sin, asin, sqrt, pi
```

In [2]:

```
data1 = pd.read_csv('data1.txt', header = None)
data1.head()
```

Out[2]:

	0	1	2	3	4	5	6	7
0	B041C2	2009-09-01 08:00:17	114.05119	22.52883	12.0	0	0	0
1	B041D7	2009-09-01 08:00:54	114.08298	22.57267	42.0	90	1	31
2	B044B1	2009-09-01 08:00:26	114.04295	22.53100	16.0	0	1	31
3	B046B7	2009-09-01 08:00:50	114.04504	22.52823	0.0	0	0	0
4	B046C5	2009-09-01 08:00:10	113.88145	22.57775	63.0	0	1	31

In [3]:

```
data2 = pd.read_csv('data2.txt', header = None)
data2.head()
```

Out[3]:

	0	1	2	3	4	5	6	7
0	B04F23	2009-09-01 08:00:43	114.06882	22.54150	5.0	112	0	0
1	B04F40	2009-09-01 08:00:09	114.09938	22.55532	0.0	67	1	31
2	B04G03	2009-09-01 08:00:03	114.08207	22.55993	27.0	0	1	31
3	B04G35	2009-09-01 08:00:00	113.95005	22.58903	26.0	0	0	0
4	B04G85	2009-09-01 08:00:28	114.06045	22.54933	1.0	0	1	31

In [4]:
```
data3 = pd.read_csv('data3.txt', header = None)
data3.head()
```

Out[4]:

	0	1	2	3	4	5	6	7
0	B041C2	2009-09-01 07:55:17	114.05417	22.52590	34.0	0	0	0
1	B041C2	2009-09-01 07:55:57	114.05138	22.52607	18.0	0	0	0
2	B041C2	2009-09-01 07:56:17	114.05140	22.52780	31.0	0	0	0
3	B041C2	2009-09-01 07:56:37	114.05150	22.52878	17.0	0	0	0
4	B041C2	2009-09-01 07:56:57	114.05154	22.52945	11.0	0	0	0

Problem 1

We wrote a Haversine distance function to calculate the sphere distance.

In [5]:
```
def distance(d1, d2):
    r = 6371000
    lon1 = d1[2]
    lat1 = d1[3]
    lon2 = d2[2]
    lat2 = d2[3]
    return 2 * r * asin(sqrt(sin(pi * (lat2 - lat1) / 360)**2 + cos(pi
```

Problem 2

a)

Firstly, we created a distance matrix d, then formulated the problem by the following linear programing:

$$MinZ$$

$$s.t : z \geq \sum_{i=1}^{n} d_{ij} y_{ij}$$

$$y_{ij} \leq x_i$$

$$\sum_{i=1}^{n} y_{ij} = 1$$

$$\sum_{i=1}^{n} x_i \leq k$$

$$x_i, y_{ij} = 0 or 1$$

For K = 5, we have the objective value of 11647 and open at the location of index 19, 43, 50, 61 ,76. For K = 10, we have the objective value of 6226.0 and open at the location of index 2, 10, 12, 19, 20, 25, 37, 49, 63, 93. We found that the higher the K is, the objective value can be smaller as we are offering more options for the possible locations for the emergency supply kits.

In [6]:

```
from ortools.linear_solver import pywraplp
```

In [7]:
```
def K_center(df, k):

    n = df.shape[0]
    m = df.shape[0]
    x = [0 for i in range(n)]
    y = [[0 for i in range(m)] for j in range(n)]
    d = [[0 for i in range(m)] for j in range(n)]


    for i in range(n):
        for j in range(m):
            d[i][j] = distance(df.iloc[i], df.iloc[j])

    solver = pywraplp.Solver('K centers',
                            pywraplp.Solver.SAT_INTEGER_PROGRAMMING)

    z = solver.NumVar(0, solver.infinity(), 'z')

    for i in range(n):
        x[i] = solver.IntVar(0, 1, 'x' + str(i))
        for j in range(m):
            y[i][j] = solver.IntVar(0, 1, 'y' + str(i) + str(j))
            solver.Add(y[i][j] <= x[i])

    constraint1 = [0 for i in range(m)]
    for j in range(m):
        constraint1[j] = solver.Constraint(0, solver.infinity())
        constraint1[j].SetCoefficient(z, 1)
        for i in range(n):
            constraint1[j].SetCoefficient(y[i][j], -d[i][j])

    constraint2 = [0 for i in range(m)]

    constraint3 = [0]
    constraint3 = solver.Constraint(0, k)

    for i in range(n):
        constraint3.SetCoefficient(x[i], 1)

    for j in range(m):
        constraint2[j] = solver.Constraint(1, 1)
        for i in range(n):
            constraint2[j].SetCoefficient(y[i][j], 1)

    objective = solver.Objective()
    objective.SetCoefficient(z, 1)
    objective.SetMinimization()

    status = solver.Solve()
    if status == solver.OPTIMAL:
        print('Problem solved in %f milliseconds' %solver.wall_time())
    elif status == solver.FEASIBLE:
        print('Solver claims feasibility but not optimality')
    else:
        print('Solver ran to completion but did not find an optimal sol
    print('The objective value is ', objective.Value())
```

```
    print("The kits should be placed at locations: ")
    for i in range(n):
        if x[i].solution_value() > 0:
            print(i)
```

In [8]:
```
K_center(data2, 5)
```

```
Problem solved in 9812.000000 milliseconds
The objective value is  11647.0
The kits should be placed at locations:
19
20
43
62
93
```

In [9]:
```
K_center(data2,10)
```

```
Problem solved in 11923.000000 milliseconds
The objective value is  6226.0
The kits should be placed at locations:
2
10
19
25
37
48
49
76
93
97
```

b)

(i)

Firstly, we calculate the Integer programing result for K = 20:

In [40]:
```
K_center(data1,20)
```

```
Problem solved in 10829691.000000 millisecond
The objective value is  5872.0
The kits should be placed at locations:
19
22
110
187
210
285
306
348
350
375
487
695
790
875
887
936
939
947
952
954
```

(ii)

Next, we formulate gready algorithm for K = 20

In [18]:
```
def findMax(dist, n):
    cur = 0
    for i in range(n):
        if (dist[i] > dist[cur]):
            cur = i
    return cur

def selectKCenter(n, d, k):
    dist = [0 for i in range(n)]
    centers = []
    for i in range(n):
        dist[i] = 10**9
    # Choose the start point as 0, need to change
    curmax = 19
    for i in range(k):
        centers.append(curmax)
        for j in range(n):
            dist[j] = min(dist[j], d[curmax][j])
        curmax = findMax(dist, n)
    return centers
```

In [12]:
```
dForData1 = [[0 for i in range(1000)] for j in range(1000)]
for i in range(1000):
    for j in range(1000):
        dForData1[i][j] = distance(data1.iloc[i], data1.iloc[j])
```

In [13]:
```
import time
start = time.time()
centers = selectKCenter(1000, dForData1, 20)
end = time.time()
print((end - start)*1000)
print(centers)

```

```
9.089946746826172
[19, 187, 358, 954, 283, 939, 200, 328, 101, 946, 936, 861, 306, 776, 88
2, 462, 964, 691, 89, 22]
```

In [14]:
```
maxV = 0
for i in range(1000):
    minV = 10**9
    for j in centers:
        minV = min(minV, dForData1[i][j])
    maxV = max(minV, maxV)
print(maxV)
```

```
7339.954910070642
```

(iii)

We found that the time solving for integer program is much higher than greedy algorithm, which is 3 hours compare to 18 seconds, but the objective value of integer program is better than greedy, which is 5872 compare to 7340. The location is slightly different because the begining location has a huge impact on the greedy solution, we have to campare multiple results of greedy solution in order to reach the results of integer programming, and that's part of the reason why greedy solution has larger objective value.

Problem 3

a)

Firstly, we created a distance matrix d, then formulated the problem by the following linear programing:

$$Min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} y_{ij}$$

$$s.t: y_{ij} \leq x_i$$

$$\sum_{i=1}^{n} y_{ij} = 1$$

$$\sum_{i=1}^{n} x_i \leq k$$

$$x_i, y_{ij} = 0 or 1$$

For K = 5, we have the objective value of 408419.5235825479 and open at the location of index 19, 39, 46, 65, 90. For K = 10, we have the objective value of 224647.33084365726 and open at the location of index 19, 39, 49, 59, 69, 71, 76, 78, 90, 97. We found that the higher the K is, the objective value can be smaller as we are offering more options for the possible locations for the emergency supply kits.

In [15]:
```
def K_median(df, k):

    n = df.shape[0]
    m = df.shape[0]
    x = [0 for i in range(n)]
    y = [[0 for i in range(m)] for j in range(n)]
    d = [[0 for i in range(m)] for j in range(n)]


    for i in range(n):
        for j in range(m):
            d[i][j] = distance(df.iloc[i], df.iloc[j])

    solver = pywraplp.Solver('Problem 3',
                             pywraplp.Solver.SAT_INTEGER_PROGRAMMING)

    for i in range(n):
        x[i] = solver.IntVar(0, 1, 'x' + str(i))
        for j in range(m):
            y[i][j] = solver.IntVar(0, 1, 'y' + str(i) + str(j))
            solver.Add(y[i][j] <= x[i])

    constraint1 = [0 for i in range(m)]
    for j in range(m):
        constraint1[j] = solver.Constraint(1, 1)
        for i in range(n):
            constraint1[j].SetCoefficient(y[i][j], 1)

    constraint2 = [0]
    constraint2 = solver.Constraint(0, k)

    for i in range(n):
        constraint2.SetCoefficient(x[i], 1)



    objective = solver.Objective()
    for j in range(m):
        for i in range(n):
            objective.SetCoefficient(y[i][j], d[i][j])
    objective.SetMinimization()

    status = solver.Solve()
    if status == solver.OPTIMAL:
        print('Problem solved in %f milliseconds' %solver.wall_time())
    elif status == solver.FEASIBLE:
        print('Solver claims feasibility but not optimality')
    else:
        print('Solver ran to completion but did not find an optimal sol
    print('The objective value is ', objective.Value())

    print("The kits should be placed at locations: ")
    for i in range(n):
        if x[i].solution_value() > 0:
            print(i)
```

In [16]:
```
K_median(data2, 5)
```

```
Problem solved in 7809.000000 milliseconds
The objective value is  408419.5235825479
The kits should be placed at locations:
19
39
46
65
90
```

In [17]:
```
K_median(data2, 10)
```

```
Problem solved in 4265.000000 milliseconds
The objective value is  224647.33084365726
The kits should be placed at locations:
19
39
49
59
69
71
76
78
90
97
```

Problem 3(b)

(i)

Firstly, we calculate the Integer programing result for K = 20:

In [42]:
```
K_median(data1, 20)
```

```
Problem solved in 9720341.000000 millisecond
The objective value is  1537892.0
The kits should be placed at locations:
12
39
198
267
274
352
389
394
430
519
549
601
643
750
804
860
920
937
967
983
```

(ii)

Next, we formulate gready algorithm for K = 20

In [19]:
```
import numpy as np
```

In [20]:
```
def findMin(dist, centers, n):
    cost = [10**9 for i in range(n)]
    for j in range(n):
        if j in centers:
            continue
        else:
            centers.append(j)
            s = 0
            for i in range(n):
                s += findMinDis(i, centers, dist)
            cost[j] = s
            centers.remove(j)
    return np.argmin(cost)

def findMinDis(i, centers, dist):
    res = 10**9
    for j in centers:
        if dist[i][j] < res:
            res = dist[i][j]
    return res


def selectKMedian(n, d, k):
    centers = []
    # Choose the start point as 0, need to change
    curmax = 39
    for i in range(k):
        centers.append(curmax)
        curmax = findMin(d, centers, n)
    return centers
```

In [21]:
```
start = time.time()
centers = selectKMedian(1000, dForData1, 20)
end = time.time()
print((end - start)*1000)
print(centers)
sumV = 0
for i in range(1000):
    minV = 10**9
    for j in centers:
        minV = min(minV, dForData1[i][j])
    sumV += minV
print(sumV)
```

```
39718.693256378174
[39, 379, 302, 543, 14, 601, 190, 519, 187, 747, 804, 907, 529, 967, 267,
903, 151, 364, 920, 38]
1672890.2795380945
```

(iii)

We found that the time solving for integer program is much higher than greedy algorithm, which is 2.5 hours compare to 39941 seconds, but the objective value of integer program is slightly better than greedy, which is 1537892 compare to 1672890. The location is slightly different because the begining location has a huge impact on the greedy solution, we have to campare multiple results of greedy solution in order to reach the results of integer programming.

Problem 4

In [22]:
```
import numpy as np
```

In [23]:
```
d = [[0 for i in range(100)] for i in range(100)]
for i in range(100):
    for j in range(100):
        d[i][j] = distance(data2.iloc[i], data2.iloc[j])
averageDistance = np.mean(d)
costPerGas = averageDistance * sqrt(100)
```

In [24]:
```
print("We have the average distance d of " + str(averageDistance))
print("We have the number of taxi driver of " + str(100))
```

```
We have the average distance d of 21498.859784159537
We have the number of taxi driver of 100
```

In [25]:
```
solver = pywraplp.Solver('Problem 4',
                         pywraplp.Solver.SAT_INTEGER_PROGRAMMING)
k = solver.IntVar(0, 100, 'k')
x = [0 for i in range(100)]
y = [[0 for i in range(100)] for i in range(100)]
for i in range(100):
    x[i] = solver.IntVar(0, 1, 'x' + str(i))
    for j in range(100):
        y[i][j] = solver.IntVar(0, 1, 'y' + str(i) + str(j))
        solver.Add(y[i][j] <= x[i])

constraint1 = [0 for i in range(100)]
for j in range(100):
    constraint1[j] = solver.Constraint(1, 1)
    for i in range(100):
        constraint1[j].SetCoefficient(y[i][j], 1)

constraint2 = [0]
constraint2 = solver.Constraint(0, 0)
constraint2.SetCoefficient(k, -1)
for i in range(100):
    constraint2.SetCoefficient(x[i], 1)
objective = solver.Objective()
objective.SetCoefficient(k, costPerGas)
for j in range(100):
    for i in range(100):
        objective.SetCoefficient(y[i][j], d[i][j])
objective.SetMinimization()
```

In [26]:

```
status = solver.Solve()
if status == solver.OPTIMAL:
    print('Problem solved in %f milliseconds' %solver.wall_time())
elif status == solver.FEASIBLE:
    print('Solver claims feasibility but not optimality')
else:
    print('Solver ran to completion but did not find an optimal solutic
print(objective.Value())
print(k.solution_value())
```

```
Problem solved in 8256.000000 milliseconds
1263750.3279368877
3.0
```

Now we notice that openning 3 locations can minimize the sum of the total cost of building gas stations plus the total distances from all taxi drivers to their respective assigned gas stations, which is now 1263750.

Problem 5

(a)

Firstly, split the dataset 3 to 7:55 - 8:00 to 8:00 to 8:05, use the first half to construct driver location and second half for rider location.

Then according to the question, we identify when the column “Occupied” switches from 0 to 1 to the set of rider locations, and we consider each rider location and its prior 5 minutes window, and get the driver location accordingly.

(b)

In [19]:
```
import pandas as pd
from math import cos, sin, asin, sqrt, pi
```

In [20]:
```
data3 = pd.read_csv('data3.txt', header = None)
data3.head()
```

Out[20]:

	0	1	2	3	4	5	6	7
0	B041C2	2009-09-01 07:55:17	114.05417	22.52590	34.0	0	0	0
1	B041C2	2009-09-01 07:55:57	114.05138	22.52607	18.0	0	0	0
2	B041C2	2009-09-01 07:56:17	114.05140	22.52780	31.0	0	0	0
3	B041C2	2009-09-01 07:56:37	114.05150	22.52878	17.0	0	0	0
4	B041C2	2009-09-01 07:56:57	114.05154	22.52945	11.0	0	0	0

In [21]:
```
data3['hour'] = pd.to_datetime(data3[1]).dt.hour
a = data3.loc[data3['hour'] == 8]
```

R is saved as rider.

In [22]:
```python
unique_id = list(a[0].unique())
rider = []
for i in unique_id:
    total = a.loc[a[0] == i]
    for j in range(total.shape[0]-1):
        if total.iloc[j,6] == 0 and total.iloc[j+1,6] == 1:
            rider.append(total.index[j+1])
            break
print(rider)
```

```
[240, 333, 22016, 404, 637, 22275, 703, 845, 889, 966, 22660, 1059, 2270
9, 1118, 22766, 22795, 1179, 1238, 1268, 1283, 1350, 22992, 1373, 1388, 2
3148, 23191, 1573, 23233, 23251, 23283, 1708, 23571, 23592, 1983, 2143, 2
155, 23896, 2304, 2341, 2566, 2618, 24270, 24283, 24317, 24354, 2811, 293
0, 24581, 3037, 3061, 24729, 3120, 24769, 3410, 25247, 3691, 3717, 3773,
3846, 25510, 3895, 3904, 3934, 3947, 4000, 25752, 25796, 4215, 4282, 441
9, 26114, 4498, 4503, 26199, 4594, 4674, 26382, 4801, 4919, 4961, 4979, 4
995, 5076, 26733, 5184, 26839, 5304, 26957, 5358, 5732, 5879, 5949, 2760
1, 6186, 6209, 27975, 27997, 6387, 6595, 6608, 6675, 28328, 6719, 7049, 2
8704, 28866, 7276, 7296, 29352, 29546, 8178, 8477, 8527, 8552, 30226, 302
74, 30322, 8806, 30486, 8954, 30664, 30675, 30716, 9128, 30786, 30828, 30
841, 30885, 9275, 31198, 9627, 9657, 9689, 9847, 31561, 9941, 9988, 1006
2, 10101, 10153, 31825, 10349, 32073, 10473, 10522, 32199, 10579, 32227,
10632, 10672, 10688, 32363, 10742, 10771, 10782, 10796, 10808, 32469, 108
62, 32517, 32527, 10935, 10988, 11057, 32697, 32780, 11182, 32873, 32883,
32893, 11301, 11322, 33081, 11464, 33231, 11643, 11668, 11682, 11700, 118
03, 11814, 11840, 33614, 33697, 12136, 12149, 12183, 33828, 12207, 33865,
33906, 12336, 12352, 34006, 34020, 12441, 34135, 34147, 12554, 34195, 342
24, 12606, 12694, 34349, 34384, 12777, 12787, 12865, 12878, 13022, 34776,
13176, 13475, 35308, 35372, 13817, 35461, 35484, 35586, 13973, 14019, 356
59, 35676, 35681, 35716, 14104, 14131, 14139, 14159, 35824, 14223, 14289,
35948, 35976, 35994, 14410, 14423, 36147, 36166, 14570, 14735, 36448, 364
70, 14864, 36572, 14967, 14989, 15002, 36654, 36689, 15086, 15141, 36784,
36808, 36820, 15222, 15255, 15265, 15289, 15337, 37033, 15413, 15507, 371
59, 37170, 37210, 15587, 15713, 37363, 15745, 37405, 15812, 37496, 37519,
15912, 16104, 37844, 16229, 37881, 16288, 16345, 16617, 16756, 16791, 168
31, 16868, 17074, 17121, 17133, 17229, 38990, 17398, 17405, 17490, 39151,
17548, 39216, 17685, 39367, 39416, 17794, 17814, 17883, 17903, 39555, 396
18, 39648, 18049, 39687, 18219, 18234, 18428, 18625, 18682, 18779, 18826,
18884, 18904, 19038, 19259, 19407, 19605, 19695, 19721, 19887, 19937, 200
15, 20074, 20114, 20145, 20157, 20201, 20234, 20287, 20435, 20453, 20911,
20985, 21077, 21093, 21266, 21320, 21357, 21381, 21394, 21429, 21631, 397
57, 39938, 39985, 40010, 40289, 40896, 41019, 41102, 41133, 41181, 41196,
41217, 41230, 41281, 41305, 41366, 41399, 41488, 41512, 41566, 41745, 423
19, 42396, 42409, 42474, 42511, 42570, 42628, 42746, 42877, 42910, 43133,
43661, 43690, 43703, 43730, 43808, 43845, 43979, 44064, 44134, 44228, 442
70, 44277, 44547, 44572, 44679, 44875, 44936, 44991, 45149, 45198, 45313,
45364, 45412, 45734, 45753, 45840, 46009, 46093, 46178, 46331, 46399, 466
11, 46690, 46793, 46827, 46933, 46990, 47087, 47127, 47364, 47374, 47395,
47984, 48003, 48009, 48115, 48479, 48518, 48638, 48649, 48721, 48786, 489
17, 48988, 49010, 49025, 49297, 49482, 49976, 49996, 50037, 50092, 50103,
50117, 50190, 50225, 50265, 50302, 51965, 52074, 52135, 52156, 52987, 533
42, 54079, 54103, 54179, 54192, 54956, 54978, 55073, 55137, 55549, 55690,
55715, 55730, 55772, 55903, 55915, 55972, 56797, 58036, 58844, 58857, 590
09, 59101, 59638, 59840, 62312, 62365, 62423, 62508, 62857, 62873, 63316,
63515, 63568, 63813, 63854, 63911, 63946, 63978, 64028, 64082, 64508, 646
39, 64697, 64738, 64811, 64893, 65168, 65252, 65978, 66025, 66103, 66160,
```

66312, 66400, 66451, 66484, 66786, 68223, 68234, 68299, 68406, 68490, 686
14, 68763, 68804, 69274, 69298, 69661, 69701, 69738, 69836, 69895, 69909,
69935, 70030, 70055, 70514, 70665, 70703, 70727, 70803, 70883, 72867, 730
57, 73122, 73236, 73679, 73820, 73889, 73929, 74076, 74240, 74276, 74343,
74499, 74708, 74845, 75137, 75163, 75476, 75520, 75755, 75820, 75849, 759
09, 75962, 75975, 75984, 76074, 76326, 76675, 76745, 76942, 76970, 77020,
77152, 77324, 77377, 77448, 77516, 77637, 77643, 77667, 77775, 77898, 779
57, 78235, 79142, 79197, 79466, 79811, 80295, 80354, 80393, 80477, 80493,
80505, 80518, 81002, 81313, 81396, 81458, 81575, 81650, 81845, 81876, 819
66, 82032, 82233, 82311, 82421, 82470, 82491, 82717, 83242, 83511, 83759,
84096, 84221, 85505, 85530, 85561, 85568, 85576, 85804, 85889, 86811, 869
38, 87883, 87942, 89099, 91042, 91099, 91165, 91346, 91477, 91747, 91899,
92077, 92352, 92471, 92512, 92545, 92913, 93031, 93062, 93079, 93092, 931
08, 93152, 93258, 93296, 93463, 93503, 93671, 93929, 93964, 94083, 94692,
94722, 94771, 94793, 94920, 94934, 95022, 95043, 95115, 95300, 96196, 962
09, 96273, 96296, 96307, 96399, 96487, 96609, 96637, 96670, 96709, 96802,
96828, 96880, 96911, 96978, 97008, 97346, 97958, 98033, 98258, 98478, 985
02, 98686, 98701, 99060, 99129, 99222, 99382, 99507, 100052, 100994, 1010
09, 101043, 101052, 101064, 101130, 101149, 101191, 101200, 101208, 10125
4, 101833, 102272, 102374, 102395, 102555, 102583, 102643, 102712, 10272
5, 102738, 102795, 102837, 102866, 102995, 103070, 103196, 104474, 10452
4, 104532, 104563, 104699, 104783, 105166, 105790, 105821, 106044, 10606
5, 106073, 106151, 106394, 106430, 106601, 106783, 106950, 107144, 10725
0, 107268, 107367, 107385, 107459, 108008, 108031, 108164, 108546, 10903
4, 110462, 111228, 111246, 111356, 111555, 112100, 113176, 115111, 11547
9, 115496, 115506, 115529, 115537, 115620, 115650, 115657, 115719, 11573
8, 115783, 115810, 115825, 115902, 116146, 116272, 117648, 117670, 11781
1, 117916, 117935, 117961, 118051, 118068, 118078, 118112, 118129, 11816
8, 118217, 118293, 118317, 118326, 118459, 118526, 118544, 118608, 11878
7, 118833, 119101, 119272, 119290, 119787, 119800, 119849, 120063, 12038
1, 120405, 120520, 120644, 120801, 120964, 121050, 121269, 121317, 12135
7, 121369, 121430, 121620, 122083, 122180, 122907, 122961, 122996, 12303
4, 123052, 123084, 123244, 123256, 123270, 123349, 123646, 123867, 12421
0, 124325, 125577, 126188, 126221, 126289, 126608, 126635, 126899, 12707
1, 127116, 127148, 127224, 127310, 127322, 127398, 127502, 127517, 12809
7, 128553, 128709, 129352, 129661, 129673, 129723, 130628, 130960, 13135
8, 131396, 131409, 131416, 131453, 131513, 131636, 131643, 131999, 13226
0, 132368, 132382, 132469, 132570, 132870, 133480, 133589, 133676, 13469
0, 134763, 134771, 134829, 134913, 135304, 135330, 135342, 135436, 13545
8, 135944, 136017, 136302, 136536, 136581, 136617, 136709, 136718, 13678
6, 137076, 137164, 137559, 137580, 137656, 137876, 137975, 138014, 13809
3, 138326, 138467, 138716, 138844, 138917, 138942, 139132, 139400, 13941
6, 139462, 139477, 139500, 139544, 140433, 140539, 140595, 140726, 14076
3, 140782, 140831, 141890, 142010, 143149, 143269, 143282, 143530, 14353
7, 143672, 151616, 151636, 151772, 151778, 151934, 152014, 152199, 15253
2, 152778, 162624, 162634, 153058, 162780, 153138, 153177, 153207, 16291
9, 153366, 153459, 153524, 163277, 153647, 153885, 153978, 154004, 15425
0, 164132, 164353, 164393, 154748, 154773, 154832, 164697, 155052, 15514
7, 164870, 155250, 155277, 155297, 165013, 165120, 165150, 155678, 15571
9, 161952, 162002, 171727, 162226, 172015, 172237, 172290, 172375, 17280
7, 172998, 173020, 173106, 173217, 173359, 173413, 173467, 173478, 17349
3, 173519, 173561, 173585, 173592, 173633, 173816, 173917, 174412, 17443
0, 174585, 174662, 174870, 174929, 174961, 175077, 175358, 175558, 17563
7, 175729, 175923, 175997, 176171, 176315, 176368, 176578, 176599, 17670
8, 176740, 176823, 177178, 177258, 177268, 177342, 177373, 177601, 17789
1, 177908, 177947, 178105, 178122, 178965, 179003, 179063, 179212, 17949
4, 179585, 179724, 180102, 180157, 180281, 180316, 180340, 180381, 18042

```
2, 180553, 180865, 181057, 181169, 181356, 181399, 181506, 181579, 18174
9, 181879, 182274, 182381, 182454, 182601, 183254, 183372, 183377, 18349
0, 183544, 183571, 183639, 183881, 184096, 184599, 184616, 185134, 18536
0, 185648, 185763, 185849, 185946, 185959, 186078, 186105, 186158, 18675
8, 186838, 186950, 187016, 187168, 187192, 187209, 187242, 187330, 18734
6, 187393, 187644, 187678, 187822, 187906, 187984, 188038, 188473, 18854
0, 188573, 188663, 188721, 188815, 188832, 188935, 189142, 189347, 18955
6, 189672, 189695, 189778, 189798, 190249, 190359, 190801, 190877, 19089
6, 190927, 190972, 191007, 191368, 191392, 191703, 192071, 192161, 19217
2, 192311, 192336, 192352, 192388, 192431, 192519, 192534, 192561, 19256
4, 192587, 192635, 193008, 193188, 193424, 193487, 193510, 193553, 19361
5, 193653, 193871, 194320, 194429, 194499, 194698, 194773, 194801, 19481
4, 195008, 195066, 195095, 195129, 195159, 195167, 195184, 195210, 19526
7, 195352, 195518, 195704, 195845, 195925, 196018, 196111, 196179, 19727
4]
```

T is saved as driver

In [23]:

```
data3['minute'] = pd.to_datetime(data3[1]).dt.minute
data3['minute'] = data3['minute'].apply(lambda x : x+60 if x <= 5 else
driver = []
for i in rider:
    driver_id = data3.iloc[i,0]
    time = data3.iloc[i,9]
    time_list = list(range(time-5,time+1))
    #now only look at the same id in the above time duration
    newdata = data3[(data3['minute'].isin(time_list)) & (data3[0] == dr
    occupied = newdata[6].tolist()
#     print(occupied)
#     print(newdata.index[0])
    if len(occupied) > 0:
        if sum(occupied) == 0:
            driver.append(newdata.index[0])
        else:
            ls = [i for i, e in enumerate(occupied) if e == 1]
            driver.append(newdata.index[ls[-1]])

```

In [24]:

```
def distance(d1, d2):
    r = 6371000
    lon1 = d1[2]
    lat1 = d1[3]
    lon2 = d2[2]
    lat2 = d2[3]
    return 2 * r * asin(sqrt(sin(pi * (lat2 - lat1) / 360)**2 + cos(pi
```

The haversin distance is saved in D.

In [*]:

```
D = {}
for i in driver:
    for j in rider:
        l1 = data3.iloc[i]
        l2 = data3.iloc[j]
        location = (i,j)
        D[location] = distance(l1,l2)
```

Here we construct a bipartite graph using NetworkX algorithm.

In [11]:

```
import networkx as nx
from networkx.algorithms import bipartite
```

In [13]:

```
B = nx.Graph()
# Add nodes with the node attribute "bipartite"
B.add_nodes_from(rider, bipartite=0)
B.add_nodes_from(driver, bipartite=1)
```

In [27]:

```
# Add edges only between nodes of opposite node set
for i in driver:
    for j in rider:
        l1 = data3.iloc[i]
        l2 = data3.iloc[j]
        location = (i,j)
        weight = distance(l1,l2)
        B.add_edge(i, j, weight = weight)
```

In [29]:

```
my_matching = bipartite.matching.minimum_weight_full_matching(B, rider,
```

In [30]:

```
my_matching
```

```
41181: 20986,
10473: 32106,
14570: 165153,
139500: 91166,
240: 21873,
176368: 98687,
6387: 28229,
84221: 118788,
2304: 180158,
172290: 172293,
12554: 34187,
135436: 89100,
41230: 41231,
69909: 111362,
106783: 106784,
153885: 163561,
164132: 32816,
2341: 23974,
37159: 37160,
```

In [34]:
```
total_cost = 0
for key, value in my_matching.items():
    l1 = data3.iloc[key]
    l2 = data3.iloc[value]
    weight = distance(l1,l2)
    total_cost += weight
```

The cost of our matching.

In [35]:
```
total_cost
```

Out[35]: 75341.22312759975

In [ ]:
```

```