Sharing Bike transportation problem

Name: 张若舟 116010296

丁 隽 116010033

朱 航 116010331

Problem

To live a healthy life, the sharing bike is getting more and more popular among the world. However, to our disappointment, sometimes we cannot find accessible sharing bike near some of the parking spots. The main reason is that there is bike surplus for some spots and shortage for other spots. Thus, to balance the number of bike for each spot to meet its demand, the bike company should regularly move bikes from spots which sharing bikes are surplus to that are insufficient. The purpose of this project is to find the optimal way for the company to transport the bike between different parking spots.

Preparation

- ## Get raw data

  The target city chosen by our team is San Francisco, and the largest bike company in San Francisco is Ford GoBike. The data of bike renting records are free to get on the official website of Ford GoBike, including start point, end point, duration time, bike id and so on. The website is www.fordgobike.com/system-data and our team take the records in February 2018 into consideration.

- ## Cleaning data

  There are 106718 records and 342 spots being used in our row data. This amount of data is too large for us to handle. Therefore, we decided to reduce the data sample. Firstly, a python program “data_get.py” was built to read raw data and calculate the average daily change (the number of bike entering the spot – the number of bike leaves the spot) of each spot, some were positive, and some were negative. Based on the calculation, some spots with tiny daily changes (absolute value less than 5) were ignored because we assume these spots need no transportation. Finally, 34 spots are selected. These 34 spots were going to be filtered again in the next step.

  Secondly, another python program “graph_get.py” was built to show the position visually on a coordinate system, in order to avoid the situation that two spots were too close. The longitude and latitude value in raw number were recalculated by normalization method and result graph of “graph_get.py” is shown in Figure 1. We also consider the changing value of each points to balance the sum of all the changing close to 0. After all of these operations, there is 29 spots left, which would be taken into our mathematic model. To make the data easier to calculate

and group (will be illustrated more in later pater), the absolute value of the changes between 5 and 10 were all regarded as 5, the absolute value between 10 and 15 were regarded as 10, and the absolute value larger than 15 were unchanged. The results of this step were shown in Table 1.

Finally, the python program “distance_get.py” was built to calculate the distance from each spot to all other spots”.

Data

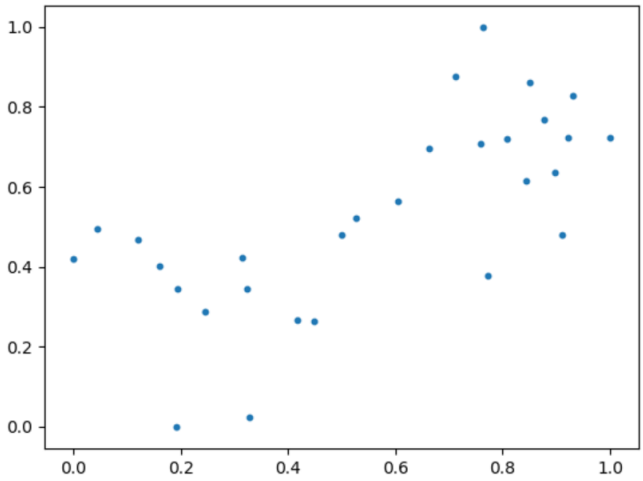

Figure 1. Bike Spots in Coordinate System

Spot Id	Daily Change	Simplified Change	Latitude	Longitude
1	-5	-5	37.750506	-122.4339496
2	-7	-5	37.7518194	-122.4266139
3	5	5	37.76476522	-122.420091
4	5	5	37.765052	-122.4218661
5	-6	-5	37.7662185	-122.4310597
6	-8	-5	37.7692005	-122.4338119
7	-6	-5	37.7693053	-122.4268256
8	6	5	37.771058	-122.402717
9	-8	-5	37.772406	-122.4356498
10	-9	-5	37.77331088	-122.4442926
11	-16	-16	37.77341397	-122.4273169
12	-8	-5	37.775946	-122.4377775

13	24	24	37.776598	-122.395282
14	-7	-5	37.776619	-122.417385
15	-8	-5	37.7774157	-122.4418376
16	-5	-5	37.7787677	-122.4159292
17	11	10	37.7810737	-122.4117382
18	10	10	37.78383	-122.39887
19	-19	-19	37.78499973	-122.3959356
20	-9	-5	37.78829998	-122.4085307
21	6	5	37.788975	-122.403452
22	22	22	37.7896254	-122.400811
23	5	5	37.7896767	-122.3904285
24	-17	-17	37.789756	-122.394643
25	-9	-5	37.792251	-122.397086
26	12	10	37.795392	-122.394203
27	6	5	37.79728	-122.398436
28	-5	-5	37.79801364	-122.4059504
29	16	16	37.80477	-122.403234

Table 1. Time vs. Angle (Large Amplitude)

Modeling

- **Assumption:**

1. Consider the distance between any two parking spots as the linear distance.
2. Assume that each spot has a path to any other spot, although some of the paths are unreasonable.
3. Sub tour is not permitted.
4. The daily net change of the number of bike of every spot is equivalent to its monthly average value, so that the daily net change of every spot is a constant.
5. The transporting truck has infinitely large carrying capacity.
6. Along the optimal route, the truck always has enough bikes for parking spots whose daily change is negative, which means that the number of bikes left on the truck is always nonnegative.
7. The transporter starts at the spots with largest positive net change.

- **Variables:**

N: number of nodes, positive integer

X_{ij}: indicator of moving from node i to node j, binary

$DIST_{ij}$: the distance between node i and node j, positive number

U_i: the order in which node i is visited, positive integer

C_i: the average change of number of sharing bikes of node i per day, integer

R_i: the number of bikes left on the truck after moving to node i, nonnegative integer

- **Formulation**

Objective function: $\sum_{i=1}^{N}\sum_{j=1}^{N} X_{ij} * DIST_{ij}$

Constraints: 1. $\sum_{i=1}^{N} X_{ij} = 1, \forall i \epsilon \{1,2, \dots \dots, N\}$

2. $\sum_{j=1}^{N} X_{ij} = 1, \forall j \epsilon \{1,2, \dots \dots, N\}$
3. $U_i - U_j + N * X_{ij} \leq N - 1, \forall i \neq j \; and \; i \neq 1 \; and \; j \neq 1$
4. $R_i = C_i + \sum_{j \neq i} R_j * X_{ji}$
5. $X_{ij} \in \{0,1\}, U_i \in \mathbb{N}^+, R_i \in \mathbb{N}^+$

This is a modified version of the general Traveling Salesman Problem (TSP). The objective is to minimize the total distance that the transporter need to move. Constraint 1 and 2 ensure that for each spot, the transporter coming in once and leaving once. Constraint 3 ensures that there is no cycle. Constraint 4 means that the remining bike in the transporter after leaving spot i equals the remaining bike of the previous spot plus the change.

Result

The result after applying our model to GoBike data is showing in figure 2, the left subfigure is the result with constraint 4, the right subfigure is the result without constraints 4:

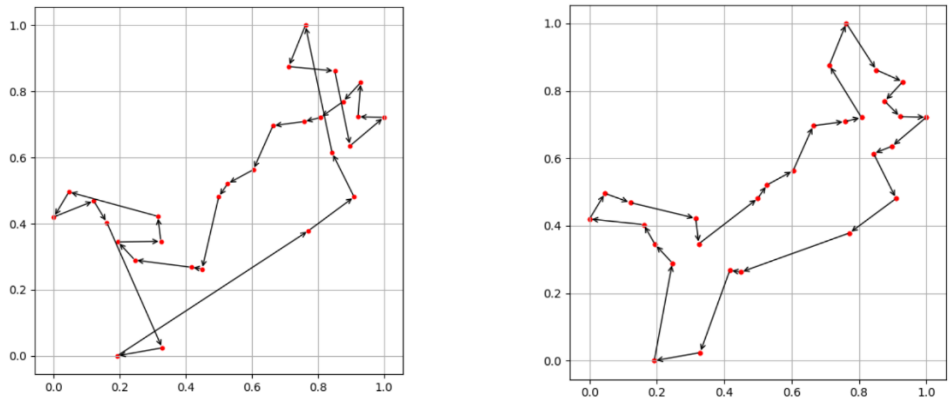

Figure 2. Route with/without constraint 4

We find that under constraint 4, there are several routes crossing each other. This is because the transporter does not have enough bike to go to the next "optimal" spot. Thus, the bike must go to a the "surplus spot" to get bike, which causes the procedure to be inefficient. By observation, we find that the daily change for some spot is very small. However, these kinds of spots require the transporter to reach and cause extra distance every day. Therefore, we propose an additional schedule for the company.

We divide these thirty spots into three groups based on their daily change. The first group corresponds to the spots with daily change greater than 5 and less than 10. The second group corresponds to the spots with daily change greater than 10 and less than 15. The remaining spots are grouped as group 3. For group 1, we set that for every three days there is one transportation. For group 2, we set that for every two days there is one transportation. For group 3, the company need to transport these bikes every day. Table 2 shows this schedule.

DAY	Transport Group
1	3
2	3, 2
3	3, 1
4	3, 2
5	3
6	3, 2, 1

Table 2. Group schedule

Improvement

The routes for the new plan is showed in Figure 3:

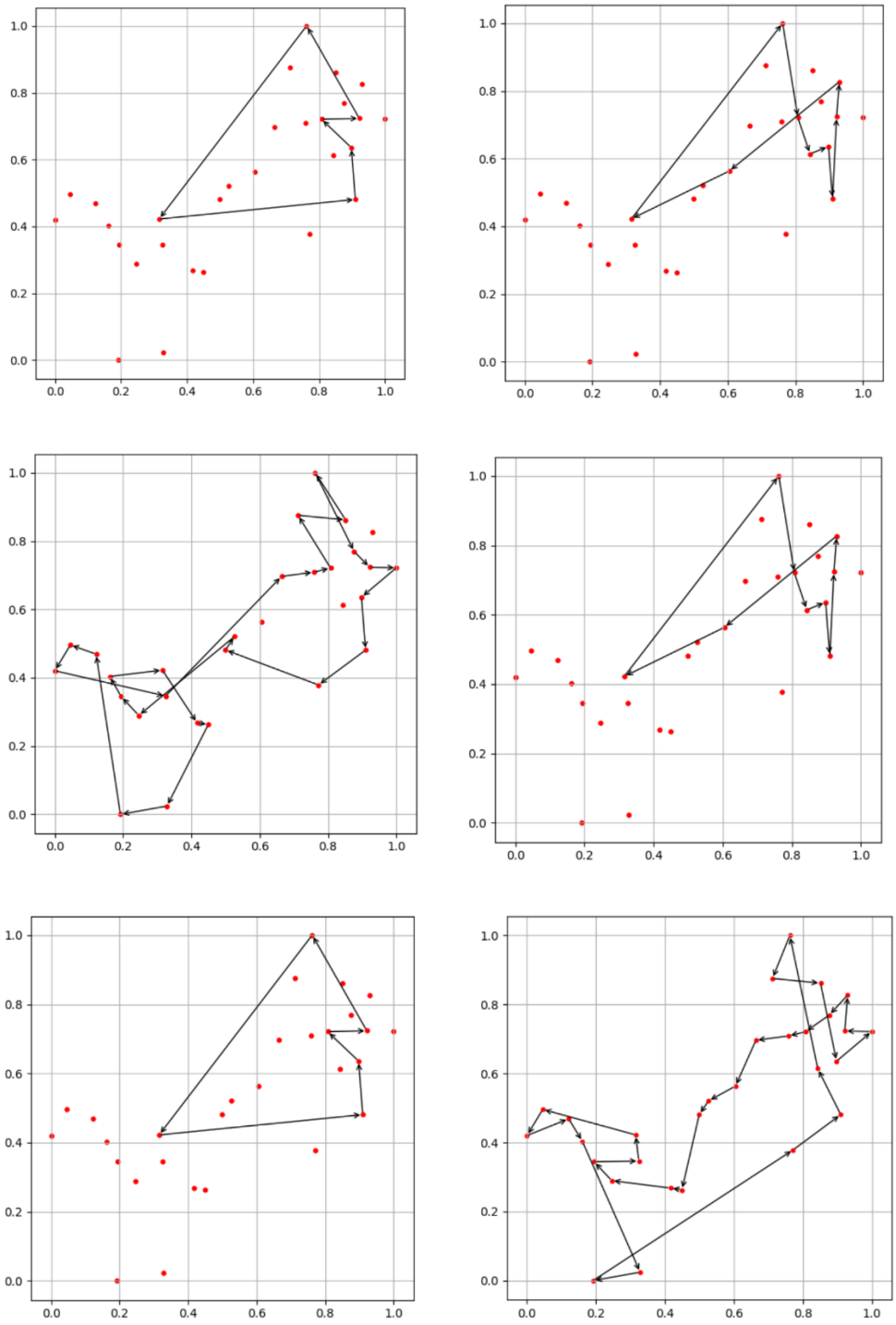

Figure 3. Route for every day

Although in day 3 and day 6, the route is still not perfect, the average daily distance has been reduced significantly. The specific route is summarized in Table 3:

Day	Route
1	11>13>19>22>24>29
2	11>29>22>18>19>13>24>26>17
3	1>12>15>10>7>20>21>22>28>27>29>25>24>23>19>13>8>14>16>5>6>9>11>4>3>2
4	11>29>22>18>19>13>24>26>17
5	11>13>19>22>24>29
6	1>8>13>18>29>28>27>19>23>24>26>25>22>21>20>17>16>14>3>4>5>6>7>11>15>10>12>9>2

Table 3. Specific route for every day

Conclusion

In this project, we propose a possible solution for the sharing bike transportation problem. The route for the transporter should be as short as possible under the constraints that the remaining bike on the transporter is nonnegative. In addition, we propose a periodic schedule, which can reduce the average daily distance for the transporter. However, due to the limitation of the solver (LINGO) for this Mixed Integer Non-linear Programming, if we use this model to a more complicated problem (for example, our raw data contains more than 300 bike spots), the solver may fail to find the optimal solution (even in our problem, some of the solution is not global optimal).

Appendix

LINGO code used to solve the model:

```
MODEL:
SETS:
CITY/1..29/:U,C,R;
LINK(CITY,CITY):DIST,X;
ENDSETS
DATA:
DIST=@OLE('C:\distance_result.csv','DIST');
C=@OLE('C:\change_of_num_29result.csv','C');
ENDDATA
N=@SIZE(CITY);
MIN=@SUM(LINK:DIST*X);
@FOR(CITY(K):@SUM(CITY(I):X(I,K))=1;);
@FOR(CITY(K):@SUM(CITY(J):X(K,J))=1;);
@FOR(CITY(K):@FOR(CITY(J)|J#GT#1#AND#K#GT#1:U(J)-U(K)+N*X(J,K)<N-
1;));
@FOR(CITY(K)|K#GT#1:@SUM(CITY(I)|I#NE#K:R(I)*X(I,K))+C(K)=R(K););
@FOR(LINK:@BIN(X););
@FOR(CITY:@GIN(U););
END
```

Python code used to clean the data and visualize the results:

```
import csv

csv_file = open("201802-fordgobike-tripdata.csv", "r")
reader = csv.reader(csv_file)
station_id_list = []
result = []
for i in range (0,343):
    station_id_list.append(i)
    result.append(0)
for row in reader:
    if row[2] != "start_station_id":
        start_id = int(row[2])
        end_id = int(row[6])
        result[start_id] -= 1
        result[end_id] += 1
for i in range (0,343):
    result[i] = int(result[i]/28)
csv_file.close()

csv_result_file = open("change_of_num_result.csv", "w",newline="")
```

```
fileheader = ["station_id", "delta_num"]
writer = csv.writer(csv_result_file)
writer.writerow(fileheader)
rows = zip(station_id_list,result)
for row in rows:
    writer.writerow(row)
csv_result_file.close()
```

```
import csv
import math

csv_file = open("change_of_num_29result_13.csv", "r")
reader = csv.reader(csv_file)
latitude_list = []
longtitude_list = []
final_result = []

for row in reader:
    if row[3] != "latitude":
        latitude = (float(row[3])-37.750506)/(37.80477-37.750506)
        longtitude = (float(row[4])+122.4442926)/(-
122.3904285+122.4442926)
        latitude_list.append(latitude)
        longtitude_list.append(longtitude)

csv_file.close()
for i in range (0,len(latitude_list)):
    result_for_i = [i+1]
    for j in range (0,len(latitude_list)):
        if j == i:
            result_for_i.append(100)
        if j != i:
            result_for_i.append(math.sqrt((latitude_list[i]-
latitude_list[j])*(latitude_list[i]-
latitude_list[j])+(longtitude_list[i]-
longtitude_list[j])*(longtitude_list[i]-longtitude_list[j])))
    final_result.append(result_for_i)
csv_result_file = open("distance_result_13.csv", "w")
fileheader = ['']
for i in range (1,30):
    fileheader.append(i)
writer = csv.writer(csv_result_file, lineterminator='\n')
writer.writerow(fileheader)
rows = final_result
for row in rows:
```

```
    writer.writerow(row)
csv_result_file.close()
```

```
import csv
import matplotlib.pyplot as plt

csv_file = open("change_of_num_11result.csv", "r")
reader = csv.reader(csv_file)
latitude_list = []
longtitude_list = []

for row in reader:
    if row[3] != "latitude":
        latitude = (float(row[3])-37.750506)/(37.80477-37.750506)
        longtitude = (float(row[4])+122.4442926)/(-
122.3904285+122.4442926)
        latitude_list.append(latitude)
        longtitude_list.append(longtitude)

plt.scatter(longtitude_list, latitude_list, s=10)
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt

csv_file = open("change_of_num_29result.csv", "r")
reader = csv.reader(csv_file)
latitude_list = []
longtitude_list = []
change_value = []

for row in reader:
    if row[3] == "latitude":
        latitude_list.append(0)
        longtitude_list.append(0)
        change_value.append(' ')
    elif row[3] != "latitude":
        latitude = (float(row[3])-37.750506)/(37.80477-37.750506)
        longtitude = (float(row[4])+122.4442926)/(-
122.3904285+122.4442926)
        latitude_list.append(latitude)
        longtitude_list.append(longtitude)
        change_value.append(str(row[2]))

def drawArrow1(A, B):
    ax = plt.subplot(122)
```

```
    ax.annotate("", xy=(B[0], B[1]), xytext=(A[0],
A[1]),arrowprops=dict(arrowstyle="->"))
    ax.grid()
    ax.set_aspect('equal') #x轴y轴等比例
drawArrow1(np.array([longtitude_list[11],latitude_list[11]]),np.arr
ay([longtitude_list[13],latitude_list[13]]))
drawArrow1(np.array([longtitude_list[13],latitude_list[13]]),np.arr
ay([longtitude_list[19],latitude_list[19]]))
drawArrow1(np.array([longtitude_list[19],latitude_list[19]]),np.arr
ay([longtitude_list[22],latitude_list[22]]))
drawArrow1(np.array([longtitude_list[22],latitude_list[22]]),np.arr
ay([longtitude_list[24],latitude_list[24]]))
drawArrow1(np.array([longtitude_list[24],latitude_list[24]]),np.arr
ay([longtitude_list[29],latitude_list[29]]))
drawArrow1(np.array([longtitude_list[29],latitude_list[29]]),np.arr
ay([longtitude_list[11],latitude_list[11]]))
plt.scatter(longtitude_list[1:], latitude_list[1:], s=10,color="r")
for i in range (1,len(longtitude_list)):
    ax = plt.subplot(122)
    ax.grid()
    ax.scatter(longtitude_list[i],latitude_list[i], c='r', s=10)

#ax.annotate("%s" %change_value[i],xy=(longtitude_list[i],latitude_
list[i]),xycoords='data',xytext=(3, 3),textcoords = 'offset
points',fontsize=9,color = 'blue')
plt.show()
```