CSCI-SHU 240 Project Report

Optimization of Citi Bike System Rebalance Route

Mengjie Shen and Yuchen Wang

May 2020

1. Introduction

Nowadays, bike sharing has become a popular commuting way in urban cities. It is a convenient and environmentally-friendly alternative besides other existing public transportation.

However, users may conduct a one-way trip leading to the unbalance of the bike system, leading to a low quality service and user dissatisfaction. A key to success for a bike sharing system is the effectiveness of rebalancing operations. Therefore, the system operator needs to use a fleet of vehicles to move the bikes along the stations for rebalancing.

In our case, we use citi bike company for case study, considering the station-based rebalancing problem. We mainly focus on Static Complete Rebalancing Problem, which means that the rebalance is done when users intervention is negligible and each rebalance will meet the target inventory level for each station in the network. We construct a model to seek a route for rebalancing the bicycles efficiently. To this end, we want to minimize the total travel distance of the service vehicle for rebalancing.

2. Methodology

We first introduce some preliminaries, and then formally define the problem of rebalance route optimization.

A. Preliminaries

1) *Station Network:* The bike stations network is represented by a directed graph $G = (V, E)$. With each station $v \in V$ as a vertex, the edges in E are directed connections of bike stations $e_{ij} = (v_i, v_j) \in E$. Each node and edge have several attributes.
2) *Feasible path:* We denotes the possible path as T, the capacity per vehicle as C. The path is a feasible path if the number of bikes on the vehicle θ satisfy the constraint that: $0 \le \theta \le C$ and T contains all stations.

B. Literatures

Our model is inspired by the Traveling Salesman Problem (TSP). TSP is a famous optimization problem which aims to find the shortest possible route that visits each city and returns to the origin city. A notable model to solve the TSP problem is the Miller–Tucker–Zemlin formulation. Our model is built on Miller–Tucker–Zemlin formulation with some constraints to ensure the rebalance target.

C. Problem Formulation

In this model, we intend to minimize the total bike shipping distances for the vehicle. Z denotes the total travel distance of vehicles for rebalancing. w_{ij} denotes the distance between bike stations. x_{ij} is a binary variable. If $x_{ij} = 1$, then vehicle will be traveled directly from v_i to v_j , otherwise $x_{ij} = 0$. d_i denotes the station demand. If d_i is positive, then station i has extra bikes. If d_i is negative, then station i lacks bikes. θ_{ij} denotes the number of bicycles on the vehicles when the vehicle is moving from v_i to v_j . U_i defines the order in which each vertex i visited on a tour.

Our model is a mixed integer linear program. A mathematical expression for the optimization statement is as follows:

$$\min Z = \sum_i \sum_j w_{ij} x_{ij}$$

s.t.

$$\sum_{j \in V} \theta_{ij} - \sum_{k \in V} \theta_{ki} = d_i, \ \forall i \in V \tag{1}$$

$$\theta_{ij} \geq 0, \text{and } \theta_{ij} \leq C \cdot x_{ij}, \ \forall (i,j) \in e_{ij} \tag{2}$$

$$U_i - U_j + n x_{ij} \leq n - 1, \ i,j = 2...n \tag{3}$$

$$1 \leq U_i \leq n - 1, \ i = 2....n \tag{4}$$

$$\sum_{j \in V} x_{ij} = \sum_{j \in V} x_{ji}, \forall i \in V \tag{5}$$

$$\sum_{j \in V} x_{ij} = 1, \ \ \forall i \in V \tag{6}$$

$$x_{ij} \in \{0, 1\}, \ \ \ \forall (i,j) \in e_{ij} \tag{7}$$

The first constraint ensures that the balance operation will meet the target inventory level for each station in the network. The second constraint ensures that the number of bikes on the vehicles is within the vehicle capacity. The third constraint and fourth constraints are Miller-Tucker-Zemlin Subtour Elimination constraints, which ensures that the route for a vehicle covers all the stations and constructs a cycle, rather than a combination of subtours. The fifth constraint ensures that the number of visits to the node equals the number of exit from the node. The sixth constraint ensures that each node is visited exactly once.

3. Data Analysis

We use real-life data from Citi Bike open data source. In particular, we investigate the data of January 2020. The raw data records all 26,020 transactions in the month in chronological order. Note that all the transactions are generated by users, which means that the data has been processed to remove trips that are taken by staff as they service and inspect the system. Each transaction has the following attributes:

- Trip Duration (seconds)
- Start Time and Date
- Stop Time and Date
- Start Station ID
- Start Station Name
- Start Station Latitude
- Start Station Longitude
- End Station ID
- End Station Name
- End Station Latitude
- End Station Longitude
- Bike ID
- User Type (Customer = 24-hour pass or 3-day pass user; Subscriber = Annual Member)
- Gender (Zero=unknown; 1=male; 2=female)

- Year of Birth

	tripduration	starttime	stoptime	start station id	start station name	start station latitude	start station longitude	end station id	end station name	end station latitude	end station longitude	bikeid	usertype	birth year	gender
0	226	2020-01-01 00:04:50.1920	2020-01-01 00:08:37.0370	3186	Grove St PATH	40.719586	-74.043117	3211	Newark Ave	40.721525	-74.046305	29444	Subscriber	1984	2
1	377	2020-01-01 00:16:01.6700	2020-01-01 00:22:19.0800	3186	Grove St PATH	40.719586	-74.043117	3269	Brunswick & 6th	40.726012	-74.050389	26305	Subscriber	1989	2
2	288	2020-01-01 00:17:33.8770	2020-01-01 00:22:22.4420	3186	Grove St PATH	40.719586	-74.043117	3269	Brunswick & 6th	40.726012	-74.050389	29268	Customer	1989	1
3	435	2020-01-01 00:32:05.9020	2020-01-01 00:39:21.0660	3195	Sip Ave	40.730897	-74.063913	3280	Astor Place	40.719282	-74.071262	29278	Customer	1969	0
4	231	2020-01-01 00:46:19.6780	2020-01-01 00:50:11.3440	3186	Grove St PATH	40.719586	-74.043117	3276	Marin Light Rail	40.714584	-74.042817	29276	Subscriber	1983	2

Fig 3.1. Head of the raw data

According to our needs, we only retain the underscored attributes and index each transaction by its start time. The processed data is shown below.

starttime	start station id	start station latitude	start station longitude	end station id
2020-01-01 00:04:50.192	3186	40.719586	-74.043117	3211
2020-01-01 00:16:01.670	3186	40.719586	-74.043117	3269
2020-01-01 00:17:33.877	3186	40.719586	-74.043117	3269
2020-01-01 00:32:05.902	3195	40.730897	-74.063913	3280
2020-01-01 00:46:19.678	3186	40.719586	-74.043117	3276
...	...	...	...	...
2020-01-31 23:29:29.391	3213	40.718489	-74.047727	3194
2020-01-31 23:30:59.367	3792	40.716870	-74.032810	3639
2020-01-31 23:42:34.846	3273	40.721651	-74.042884	3209
2020-01-31 23:45:00.680	3185	40.717733	-74.043845	3267
2020-01-31 23:48:35.170	3206	40.731169	-74.057574	3202

Fig 3.2. Processed data

Picking out unique station ids, we figure out that there are in total 51 different start stations and 54 end stations in the dataset. The set of start stations is a subset of the set of end stations, and the latter set has three more stations. We only study the 51 stations included in the start stations, each of those has an exact position denoted by a latitude and a longitude (see Fig 3.3).

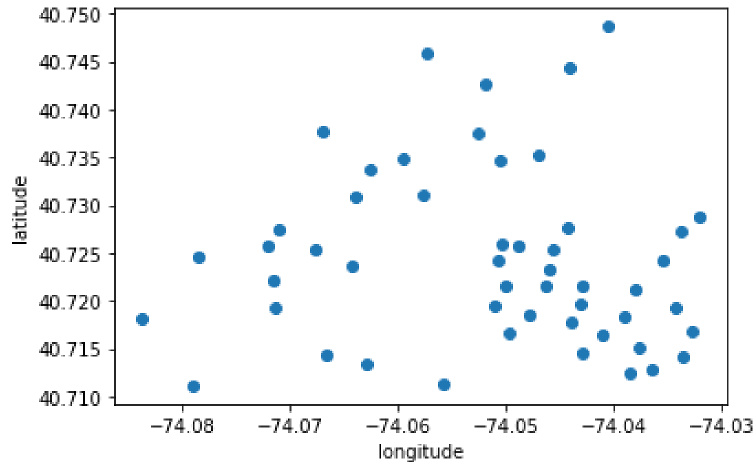


Fig 3.3. Stations distribution graph

In order to reduce computational complexity, we apply the *k*-means algorithm to aggregate the stations into 10 clusters (see Fig 3.4.) and regard each cluster as a new generated station.

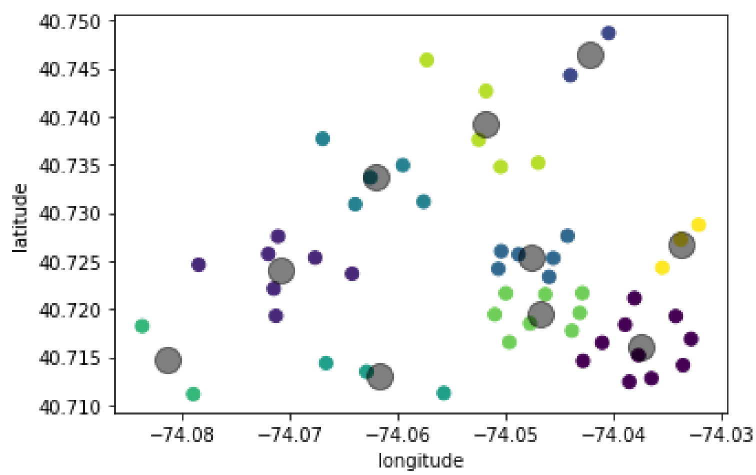


Fig 3.4. *K*-means result with 10 clusters

So far, each one of the 51 stations belongs to one of the 10 clusters. We only consider inter-cluster transactions, ignoring the intra-cluster ones. Next, we count the changes in the number of bikes in each cluster. Because the exact numbers of bikes at each station are not given in the dataset, we set 00:00 a.m. as a benchmark, where the numbers of bikes at all clusters equal to zero. Then, we iterate through all transactions and -1 (+1) if there is a pick-up (drop-off) action.

The graph below Fig 3.5.a. and Fig.3.5.b. show how the number of bikes at each cluster fluctuate on Wednesday 15th January and Saturday 18th January respectively. (without any rebalancing operations).

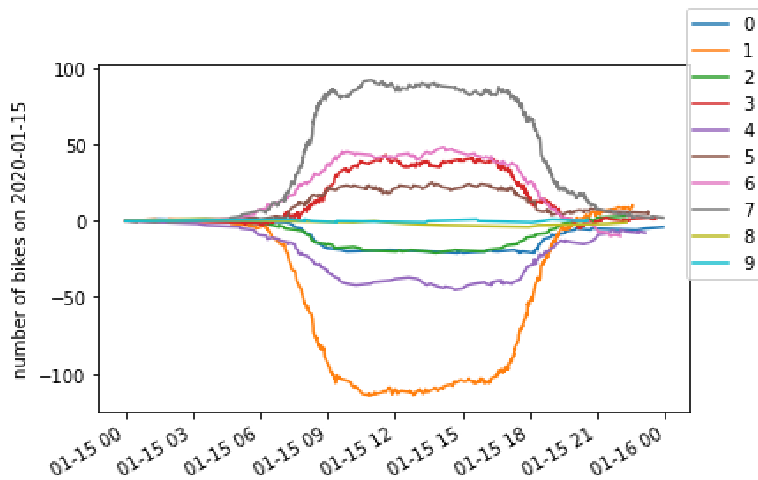


Fig 3.5.a. Relative number of bikes at each cluster (Wednesday 15th January)

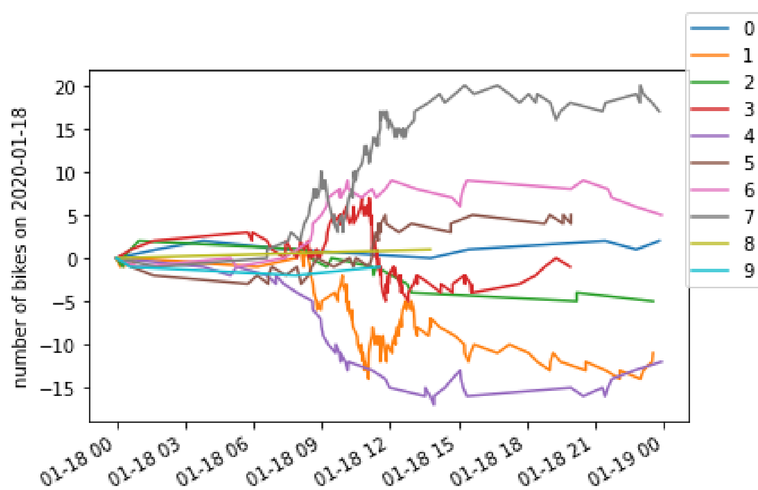


Fig 3.5.b. Relative number of bikes at each cluster (Saturday 18th January)

According to the graphs, we figure out that the law of data fluctuation is significantly different between weekdays and weekends. On weekdays, peak traffic is between 6 a.m. and 9 a.m. and between 5 p.m. and 8 p.m. People commute between where they live and where they work. For stations near the business district, the figures rise in the morning and fall almost symmetrically in the afternoon, while for residential areas the reverse is true. However, on weekends, people's travel needs are no longer concentrated in the rush hour and people's destinations are more diversified.

Since the dataset is not detailed enough, we do not know how many docks are there in each station. It is difficult for us to calculate the rebalancing demand. However, to illustrate our model, we simply calculate the changes in the number of bicycles at each station throughout 15th January and use the result as our rebalancing demand. After a day of operation, we reset the whole system back to where it was that morning.

4. Result and Discussion

We input our optimization model into GAMS to get final results. In particular, we used the Cplex MIP solver to solve our problem and did a sensitivity analysis by setting different vehicle capacity.

```
----     66 VARIABLE x.L  whether there are bikes transfered from i to j

             node1       node2       node3       node4       node5       node6

node3        1.000
node4                                                                    1.000
node5                                1.000
node7                    1.000
node8                                            1.000
node9                                                        1.000

     +       node7       node8       node9      node10

node1                    1.000
node2                                1.000
node6                                            1.000
node10       1.000


----     66 VARIABLE z.L                   =        0.200  total travel distance
```

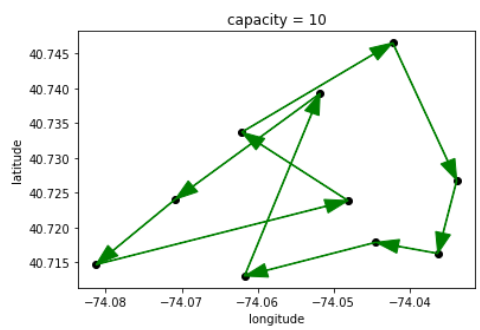


Fig 4.1. optimal solution with vehicle capacity = 10

```
----     66 VARIABLE x.L  whether there are bikes transfered from i to j

             node1       node2       node3       node4       node5       node6

node1                                            1.000
node2                                                        1.000
node4                                                                    1.000
node7                    1.000
node8        1.000
node10                               1.000

     +       node7       node8       node9      node10

node3        1.000
node5                                1.000
node6                                            1.000
node9                    1.000


----     66 VARIABLE z.L                   =        0.190  total travel distance
```

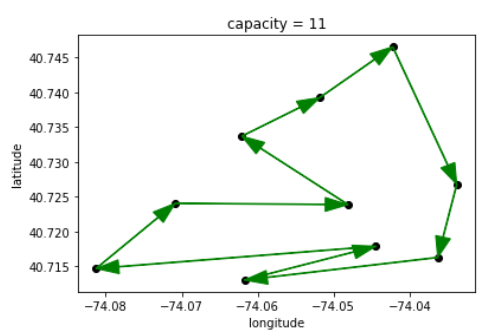


Fig 4.2. optimal solution with vehicle capacity = 11

```
----     67 VARIABLE x.L  whether there are bikes transfered from i to j

            node1       node2       node3       node4       node5       node6

node4       1.000
node5                   1.000
node6                                           1.000
node7                               1.000
node9                                                       1.000
node10                                                                  1.000

    +       node7       node8       node9      node10

node1                   1.000
node2       1.000
node3                                           1.000
node8                               1.000


----     67 VARIABLE z.L                   =        0.190  total travel distance
```

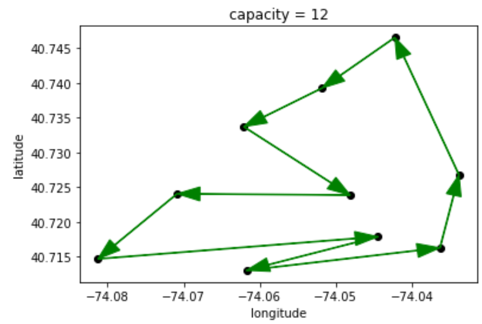


Fig 4.3.optimal solution with vehicle capacity = 12

```
----     67 VARIABLE x.L  whether there are bikes transfered from i to j

            node1       node2       node3       node4       node5       node6

node5                   1.000
node6                                           1.000
node7                               1.000
node8       1.000
node9                                                       1.000
node10                                                                  1.000

    +       node7       node8       node9      node10

node1                               1.000
node2       1.000
node3                                           1.000
node4                   1.000


----     67 VARIABLE z.L                   =        0.156  total travel distance
```

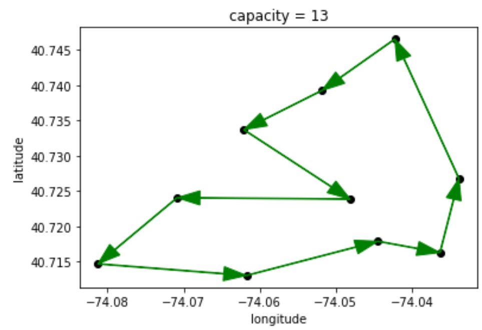


Fig 4.4.optimal solution with vehicle capacity = 13

```
----     67 VARIABLE x.L  whether there are bikes transfered from i to j

            node1       node2       node3       node4       node5       node6

node2                                                       1.000
node4                                                                   1.000
node7                   1.000
node8                                           1.000
node9       1.000
node10                              1.000

    +       node7       node8       node9      node10

node1                   1.000
node3       1.000
node5                               1.000
node6                                           1.000


----     67 VARIABLE z.L                   =        0.156  total travel distance
```

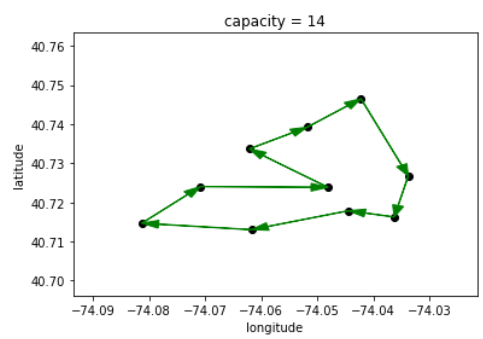


Fig 4.5.optimal solution with vehicle capacity =14

The above output indicates the optimal strategy to design the shortest path to rebalance the whole station network at different vehicle capacity.

5. Conclusion

In this project we constructed a model to generate an optimal route for the station-based bikes rebalancing problem, inspired by the Traveling Salesman Problem model. We used the data from the citi bike company as a case study and simplified the problem by clustering the citi bike bike stations according to geographical locations. We then used GAMS and applied the Cplex Mixed Integer Problem, and generated the optimal route for rebalancing.

Our model also has some limitations and space to improve in the future. Currently, it can only be used in the case where there is only one vehicle used for rebalancing instead of the fleet and each vehicle can only visit each station once. In addition, because the data is not detailed enough, we cannot get the accurate demand on each station. To improve that, we may run the model on a more sophisticated dataset and get a more constructive result.

6. Appendix

A. GAMS

```
Sets
        i       clusters of stations    /node1*node10 /;
Alias (i,j)

Parameters

        d(i) demands for each cluster
        /       node1   -4
                node2   10
                node3   3
                node4   2
                node5   -8
                node6   5
                node7   -10
                node8   2
                node9   -1
                node10  1/;

Table w(i,j) distance between bike stations v(i) and v(j)


         node1       node2         node3       node4       node5       node6       node7       node8       node9       node10
node1    0.          0.01740656    0.02803472  0.02564747  0.01430677  0.03113276  0.02066321  0.01795033  0.01961556  0.03876175
node2  0.01740656    0.            0.01582217  0.01404104  0.02275546  0.01460469  0.01709441  0.00701318  0.03442013  0.02341575
node3  0.02803472    0.01582217    0.          0.02771651  0.02437968  0.02190971  0.01169642  0.02259806  0.03837457  0.01202611
node4  0.02564747    0.01404104    0.02771651  0.          0.0354326   0.01078307  0.03112872  0.00831374  0.04501595  0.03082944
node5  0.01430677    0.02275546    0.02437968  0.0354326   0.          0.03716745  0.01303437  0.02712287  0.0140105   0.03640369
node6  0.03113276    0.01460469    0.02190971  0.01078307  0.03716745  0.          0.0291344   0.01387568  0.04899975  0.02148753
node7  0.02066321    0.01709441    0.01169642  0.03112872  0.01303437  0.0291344   0.          0.0236926   0.02701077  0.02365105
node8  0.01795033    0.00701318    0.02259806  0.00831374  0.02712287  0.01387568  0.0236926   0.          0.03696844  0.02873597
node9  0.01961556    0.03442013    0.03837457  0.04501595  0.0140105   0.04899975  0.02701077  0.03696844  0.          0.05039163
node10 0.03876175    0.02341575    0.01202611  0.03082944  0.03640369  0.02148753  0.02365105  0.02873597  0.05039163  0.
```

```
scalar C capacity for vheicle    /12/;

variable x(i,j)  whether there are bikes transfered from i to j
         q(i,j)  quantity of bikes transfered from i to j
         u(i)    MTZ subtour elimination
         z       total travel distance;

binary variable x;
positive variable q,u;
Equations
         distance        define objective fuction
         c1(i)      make sure rebalanced
         c2(i) make sure enter only once
         c3(i) make sure exit only once
         c4(i,j)    make sure under vheicle capacity
         c5   MTZ subtour elimination_1
         c6   MTZ subtour elimination_2
         c7   MTZ subtour elimination_3;

distance..    z=e=sum((i,j),w(i,j)*x(i,j));
c1(i)..       sum(j,q(i,j))-sum(j,q(j,i)) =e= d(i);
c2(i)..       sum(j,x(i,j))=e= 1;
c3(i)..       sum(j,x(j,i))=e= 1;
c4(i,j)..     q(i,j) =l= C*x(i,j);
c5(i,j)$(ord(i)>1 and ord(j)>1)..        u(i)-u(j)+10*x(i,j) =l=9;
c6(i,j)$(ord(i)>1)..      u(i)=l=9;
c7(i,j)$(ord(i)>1)..      u(i)=g=1;

Model travel_distance /all/;
option MIP = Cplex;
Solve travel_distance using MIP minimizing z;
Display x.l, z.l;
```

B. Python Data Processing

In [1]:

```python
import numpy as np
import pandas as pd
df = pd.read_csv('JC-202001-citibike-tripdata.csv')
```

In [2]:

```python
df = pd.read_csv('JC-202001-citibike-tripdata.csv')
df['starttime'] = pd.to_datetime(df['starttime'], format="%Y-%m-%d %H:%M:%S")
df['stoptime'] = pd.to_datetime(df['stoptime'], format="%Y-%m-%d %H:%M:%S")
df.set_index('starttime', inplace=True)
df.loc['2020-01-02':'2020-01-03',:]
df.drop(['tripduration', 'stoptime', 'start station name', 'end station name', 'end station latitude', 'end station longitude', 'bikeid', 'use
df
```

Out[2]:

	start station id	start station latitude	start station longitude	end station id
starttime				
2020-01-01 00:04:50.192	3186	40.719586	-74.043117	3211
2020-01-01 00:16:01.670	3186	40.719586	-74.043117	3269
2020-01-01 00:17:33.877	3186	40.719586	-74.043117	3269
2020-01-01 00:32:05.902	3195	40.730897	-74.063913	3280
2020-01-01 00:46:19.678	3186	40.719586	-74.043117	3276
...	...	...	...	...
2020-01-31 23:29:29.391	3213	40.718489	-74.047727	3194
2020-01-31 23:30:59.367	3792	40.716870	-74.032810	3639
2020-01-31 23:42:34.846	3273	40.721651	-74.042884	3209
2020-01-31 23:45:00.680	3185	40.717733	-74.043845	3267
2020-01-31 23:48:35.170	3206	40.731169	-74.057574	3202

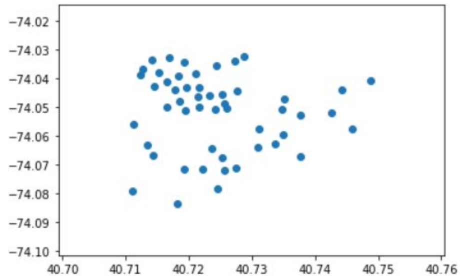
26020 rows × 4 columns

In [3]:
```
cd = df.groupby('start station id').mean()
cd = cd.loc[:,['start station latitude', 'start station longitude']]
cd
```

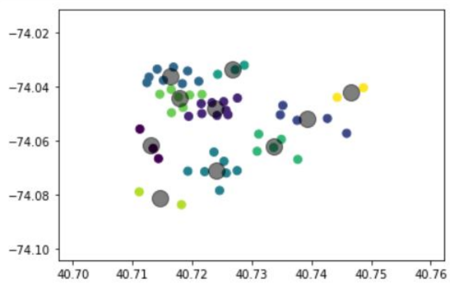
Out[3]:

start station id	start station latitude	start station longitude
3184	40.714145	-74.033552
3185	40.717733	-74.043845
3186	40.719586	-74.043117
3187	40.721124	-74.038051
3191	40.718211	-74.083639
3192	40.711242	-74.055701
3193	40.724605	-74.078406
3194	40.725340	-74.067622
3195	40.730897	-74.063913
3196	40.744319	-74.043991
3198	40.748716	-74.040443
3199	40.728745	-74.032108
3201	40.737711	-74.066921
3202	40.727223	-74.033759
3203	40.727596	-74.044247
3205	40.716540	-74.049638
3206	40.731169	-74.057574
3207	40.737604	-74.052478
3209	40.724176	-74.050656
3210	40.742677	-74.051789
3211	40.721525	-74.046305

In [4]:
```
%matplotlib inline
import matplotlib.pyplot as plt
lats = cd['start station latitude'].values
longs = cd['start station longitude'].values
plt.scatter(lats,longs)
plt.show()
```



In [5]:
```
from sklearn.cluster import KMeans
X = list(zip(lats,longs))
n_clusters = 10
kmeans = KMeans(n_clusters, random_state=0).fit(X)
y_kmeans = kmeans.predict(X)
y_kmeans
centers = kmeans.cluster_centers_
plt.scatter(lats, longs, c=y_kmeans, s=50, cmap='viridis')
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
plt.show()
```



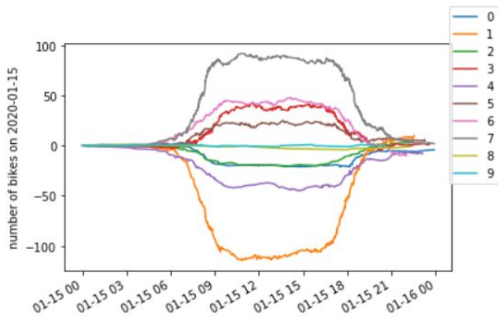
In [6]:
```
cd['cluster'] = y_kmeans
cd
```

Out[6]:

start station id	start station latitude	start station longitude	cluster
3184	40.714145	-74.033552	3
3185	40.717733	-74.043845	7
3186	40.719586	-74.043117	7
3187	40.721124	-74.038051	3
3191	40.718211	-74.083639	8
3192	40.711242	-74.055701	0
3193	40.724605	-74.078406	4
3194	40.725340	-74.067622	4
3195	40.730897	-74.063913	6
3196	40.744319	-74.043991	9
3198	40.748716	-74.040443	9
3199	40.728745	-74.032108	5
3201	40.737711	-74.066921	6
3202	40.727223	-74.033759	5
3203	40.727596	-74.044247	1
3205	40.716540	-74.049638	7
3206	40.731169	-74.057574	6
3207	40.737604	-74.052478	2
3209	40.724176	-74.050656	1
3210	40.742677	-74.051789	2
3211	40.721525	-74.046305	1
3212	40.734786	-74.050444	2
3213	40.718489	-74.047727	7
3214	40.712774	-74.036486	3
3220	40.734961	-74.059503	6
3225	40.723659	-74.064194	4
3267	40.712419	-74.038526	3
3268	40.713464	-74.062859	0
3269	40.726012	-74.050389	1
3270	40.725289	-74.045572	1
3272	40.723332	-74.045953	1

In [8]:
```
import datetime
import random
day = datetime.datetime(2020, 1, 15)
df_of_period = df.loc[:, '2020-01-15']
df_of_period
nbikes = {i:[0] for i in range(n_clusters)}
times = {i:[day] for i in range(n_clusters)}
for trans in df_of_period.iterrows():
    index = trans[0]
    series = trans[1]
    start_id = int(series['start station id'])
    end_id = int(series['end station id'])
    start_cluster = int(cd.loc[start_id, 'cluster'])
    times[start_cluster].append(index.to_pydatetime())
    nbikes[start_cluster].append(nbikes[start_cluster][-1]-1)
    try:
        end_cluster = int(cd.loc[end_id,'cluster'])
        times[end_cluster].append(index.to_pydatetime())
        nbikes[end_cluster].append(nbikes[end_cluster][-1]+1)
    except KeyError:
        pass
```

In [9]:
```
#plot the results
fig, ax = plt.subplots()
for i in range(n_clusters):
    ax.plot(times[i], nbikes[i])
fig.legend([str(i) for i in range(n_clusters)])
plt.ylabel('number of bikes on '+str(day)[:10])
fig.autofmt_xdate()
```



In [10]:
```
demands = np.zeros(n_clusters)
for i in range(n_clusters):
    demands[i] = nbikes[i][-1]
demands
```

Out[10]:
```
array([ -4.,  10.,   3.,   2.,  -8.,   5., -10.,   2.,  -1.,   1.])
```

In [11]:
```
c = kmeans.cluster_centers_
c
```

Out[11]:
```
array([[ 40.7130215 , -74.06172358],
       [ 40.72385174, -74.04809662],
       [ 40.739237  , -74.05178916],
       [ 40.71626457, -74.03628198],
       [ 40.72403822, -74.07085132],
       [ 40.72675413, -74.03378323],
       [ 40.7336816 , -74.06208206],
       [ 40.71786744, -74.04443974],
       [ 40.71467065, -74.0812697 ],
       [ 40.74651732, -74.0422171 ]])
```

In [12]:
```
distances = np.zeros((n_clusters,n_clusters))
for i in range(n_clusters):
    for j in range(n_clusters):
        distances[i, j] = ((c[i, 0]-c[j, 0])**2 + (c[i, 1]-c[j, 1])**2)**0.5
distances
```

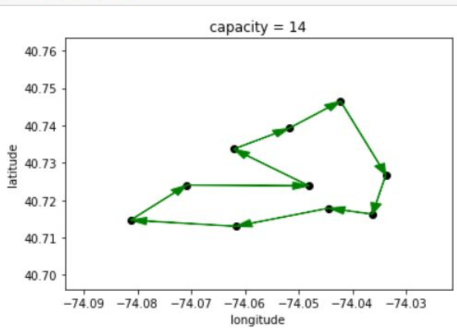
Out[12]:
```
array([[0.        , 0.01740656, 0.02803472, 0.02564747, 0.01430677,
        0.03113276, 0.02066321, 0.01795033, 0.01961556, 0.03876175],
       [0.01740656, 0.        , 0.01582217, 0.01404104, 0.02275546,
        0.01460469, 0.01709441, 0.00701318, 0.03442013, 0.02341575],
       [0.02803472, 0.01582217, 0.        , 0.02771651, 0.02437968,
        0.02190971, 0.01169642, 0.02259806, 0.03837457, 0.01202611],
       [0.02564747, 0.01404104, 0.02771651, 0.        , 0.0354326 ,
        0.01078307, 0.03112872, 0.00831374, 0.04501595, 0.03082944],
       [0.01430677, 0.02275546, 0.02437968, 0.0354326 , 0.        ,
        0.03716745, 0.01303437, 0.02712287, 0.0140105 , 0.03640369],
       [0.03113276, 0.01460469, 0.02190971, 0.01078307, 0.03716745,
        0.        , 0.0291344 , 0.01387568, 0.04899975, 0.02148753],
       [0.02066321, 0.01709441, 0.01169642, 0.03112872, 0.01303437,
        0.0291344 , 0.        , 0.0236926 , 0.02701077, 0.02365105],
       [0.01795033, 0.00701318, 0.02259806, 0.00831374, 0.02712287,
        0.01387568, 0.0236926 , 0.        , 0.03696844, 0.02873597],
       [0.01961556, 0.03442013, 0.03837457, 0.04501595, 0.0140105 ,
        0.04899975, 0.02701077, 0.03696844, 0.        , 0.05039163],
       [0.03876175, 0.02341575, 0.01202611, 0.03082944, 0.03640369,
        0.02148753, 0.02365105, 0.02873597, 0.05039163, 0.        ]])
```

visulization

In [17]:
```
def drawArrow(A, B):
    plt.arrow(A[0], A[1], B[0]-A[0], B[1]-A[1], width=0.0001, head_width=0.002,length_includes_head=True, color='g')

rt = [  1 ,9, 5, 2, 7, 3, 10, 6, 4, 8 ]
rt = [i-1 for i in rt]
def plotRoute(route,cap):
    route.append(route[0])
    plt.scatter(c[:,1],c[:,0],color='black')
    for i in range(len(route)-1):
        drawArrow([c[route[i],1], c[route[i],0]], [c[route[i+1],1], c[route[i+1],0]])
    plt.title('capacity = ' + str(cap))
    plt.xlabel('longitude')
    plt.ylabel('latitude')
    plt.show()

plotRoute(rt,14)
```



7. Reference

Duan, Yubin, Jie Wu, and Huanyang Zheng. "A greedy approach for vehicle routing when rebalancing bike sharing systems." *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018.

Pal, Aritra, and Yu Zhang. "Free-floating bike sharing: Solving real-life large-scale static rebalancing problems." *Transportation Research Part C: Emerging Technologies* 80 (2017): 92-116.