New York taxi problem stat222 capstone project

Cheng Lin 3035266742

Abstract:

Our project mainly include three part: first, the comparison of trip duration between taxi and bike in Manhattan based on bootstrap method. second, trip time prediction based on ensemble tree models(LightGBM and XGBoost), we combined these two part together and make a bokeh app which can give an interactive guide to when taxis are faster or slower than Citi Bikes. Third, based on a certain assumption, we compare the greedy solution and optimal solution in different situations for taxi order dispatch problem.

1. Problem Description

Our project mainly concentrates on New York taxi problem. Our data sources include:

1. Yellow Taxi Trip Data from TLC (https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page)
2. Manhattan Bike Data from Citi Bike (https://www.citibikenyc.com/system-data)
3. Manhattan Daily Weather Data from NOAA (https://w2.weather.gov/climate/xmacis.php?wfo=okx)

For the comparison of trip duration between taxi and bike in Manhattan based on bootstrap method, we used yellow taxi data from 2018 to 2019 and Manhattan Bike Data from Citi Bike in 2019.
For the prediction of trip duration of taxi and citi bike, we used yellow taxi data from 2018 to 2019, Manhattan Bike Data from Citi Bike from 2018 to 2019 and Manhattan Daily Weather Data from NOAA.
For the comparison between the greedy solution and optimal solution in different situations for taxi order dispatch problem, we used yellow taxi data in 2016 (with latitude and longitude) and XGBoost model.

In our Bokeh app, we want to show zone-level comparison between the travel time of taxis and bikes in Manhattan and give predicted trip time by both bike and taxi, given location and time of the trip. This can give people a good visual traffic situation in New York, and can provide decision-making for people's way of travel.

In our order allocation simulation, we compare the dispatch time of greedy algorithm and optimal algorithm in two scenarios. In one scenario, we used the XGBoost trained model and the data extracted from the test set to conduct a global reality simulation. In another scenario, we construct a simple model with clear physical meaning to simulate.

The results of these two simulations show that the optimal algorithm has little effect on the reduction of dispatch time compared with greedy algorithm in the case of a large global scope, while in the case of high-density passengers and drivers in a small area, the optimal algorithm has a great effect on the reduction of dispatch time compared with greedy algorithm.

2. EDA

For the 2018 and 2019 taxi datasets, the dataset itself has no latitude and longitude but only zones. For the 2018 and 2019 datasets, we removed the outliers of fare, travel distance and travel time. We also extracted year, month, day and week from datetime. We took logarithm of trip distance and trip time to reduce skewness and did a cyclic transformation on time-related predictors to preserve the continuity of them.

For the 2016 taxi dataset, because it have the features latitude and longitude rather than zones, we have more thing to do with latitude and longitude features. First, we used K-means cluster on latitude and longitude and used elbow method to get 7 pick up latitude centroids, 7 pick up longitude centroids, 7 drop off latitude centroids, 7 drop off longitude centroids. Second, we computed haversine distance, manhattan distance and Bearing from start to end features based on latitude and longitude. Third, we did a PCA transformation on latitude and longitude to get the relevant PCA latitude and longitude. We also dropped the observations where latitude and longitude of pick up location or drop off location are not in New York city.

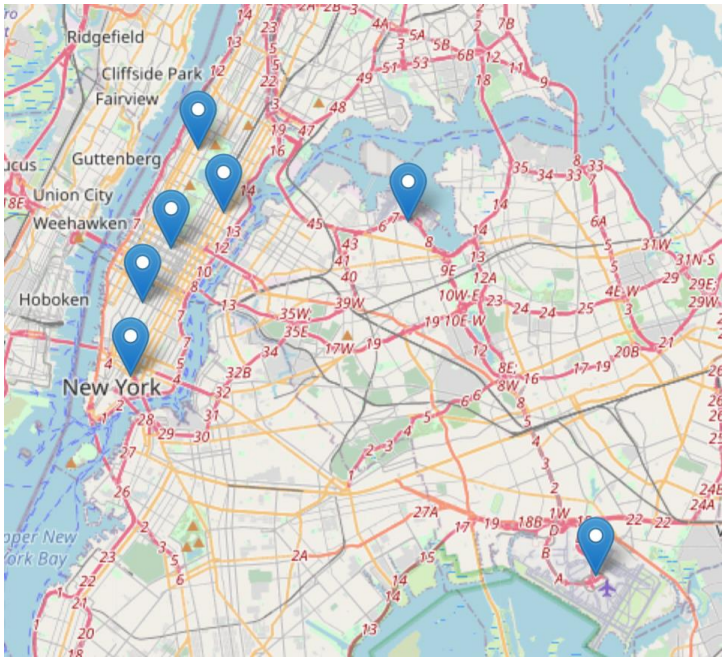

Figure 1: 7 pick_up centroids

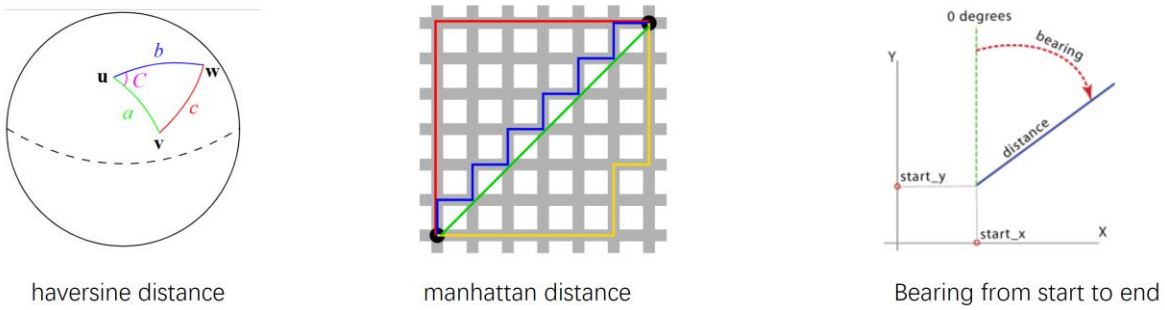

Figure 2: An intuitive image of haversine distance, manhattan distance and Bearing

For the citi bike, the features engineering is similar to the 2016 taxi dataset because it also have latitude and longitude features.

Apart from feature engineering, we also have a preliminary impression on the relationship between order counts and time (hour, week), the relationship between trip time and time (hour) in exploratory data analysis. We also observed that weather has a significant impact on order counts and trip time. The impact of weather on the order counts and trip time motivated us to add weather features to our prediction model.

3. Bokeh app

Our Bokeh app can give an interactive guide to when taxis are faster or slower than Citi Bikes and trip time prediction given user's input. We can see from the interactive interface of Bokeh app that for comparison, the inputs are start zone, end zone, time block, selected weekend or weekday, the output are the probability of bike beats taxi, taxi median trip time and bike median trip time . For prediction, the inputs are pick up address, drop off address, trip start time, the outputs are the prediction of trip distance, trip time of yellow cab, trip time of citi bike.

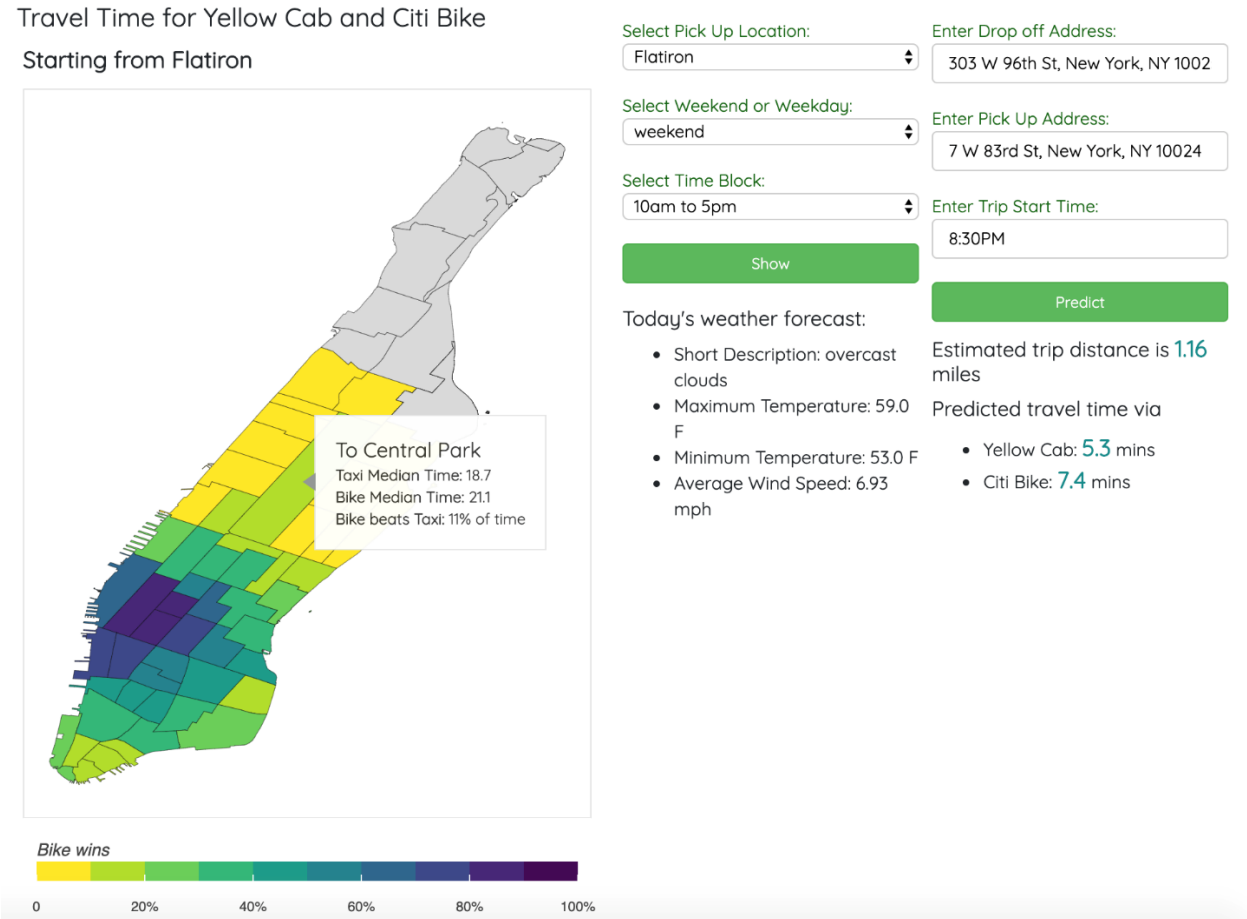

Figure 3: Interactive interface of Bokeh app

Our time blocks for comparison are determined by the following Figure.

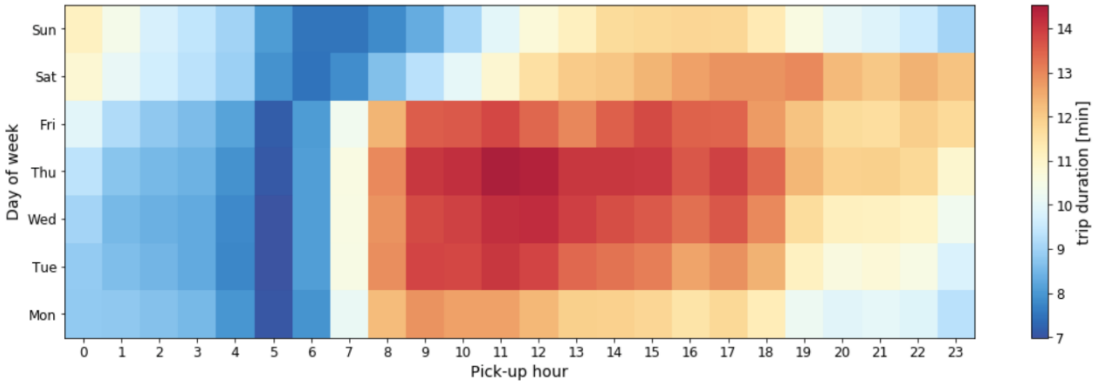

Figure 4

We divide the time of a day into several time blocks according to its periodic regularity as the user's input.

In comparison part, as we can see in the Interactive interface of Bokeh app, the inputs and outputs are based on zones. So we have to map the latitude and longitude of all the stations of the bike dataset into the corresponding zone id in the taxi dataset. In order to create a corresponding zone id for each latitude and longitude of bike station, we need to solve the **point-in-polygon** (**PIP**) problem. In computational geometry, the point-in-polygon (PIP) problem asks whether a given point in the plane lies inside, outside, or on the boundary of a polygon. We used the shapefile of the New York taxi data and convert the latitude and longitude of all stations of citi bike into shapefile, then used **Ray Casting algorithm** to solve the PIP problem. The function conducting the Point in Polygon (PIP) query can be found in geometry.within() in the Shapely package in python. Then, we grouped by start zone, end zone, time block, week block and used bootstrap method to store the result (the probability of bike beats taxi) for our Bokeh app.

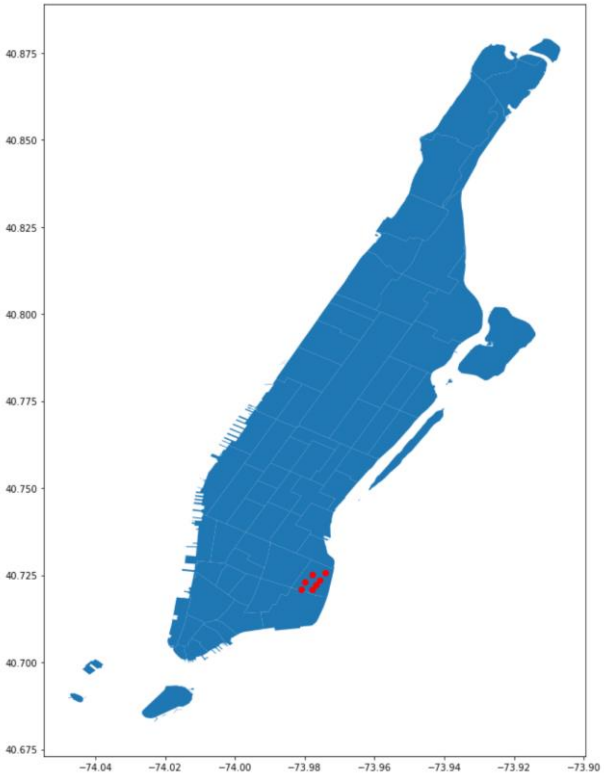

Figure 5: citi bike stations in a given zone

In trip prediction part, we used LightGBM for both taxi and bike datasets. We choose LightGBM because it has the following advantages: Faster training speed, lower memory usage, Higher accuracy and Easily handle categorical variables with many levels. The mainly difference of these two datasets are that taxi dataset only have zone id for pick up location and drop off location but bike dataset have latitude, longitude and station id of pick up location and drop off location. So their differences in feature engineering are mainly related to latitude and longitude, which is shown in EDA part. Both prediction used Manhattan Daily Weather Data from NOAA, which increased the model performance a lot.

The final model for both taxi and bike based on the following model choice:
Train Test Split: use 2018 data for prediction, 2019 data for testing and validation
hyperparameters: learning rate, max depth, number of leaves, feature fraction, bagging fraction, max bin
Parameter Tuning: Use randomized CV to find the best set of hyperparameters.

The taxi model performance:
42% improvement of RMSE: LightGBM 3.66mins vs baseline model (linear regression) 6.30mins
34% improvement of R squares: LightGBM 77% vs baseline model 50%.

The citi bike model performance:
RMSE: 3.58mins
R squares: 57%

4. Order allocation simulation

Order allocation problem and assumption:

Question description: Suppose there are n passengers, n drivers.
Define matrix $\boldsymbol{C} \in \boldsymbol{R}^{n \times n}$
$\boldsymbol{C_{ij}}$ **:** the driving time of driver i to pick up passenger j.
Assumption:
·Assume for all $\boldsymbol{i, j}$ $\boldsymbol{C_{ij}}$ is fixed.
·Assume for all $\boldsymbol{i, j}$ we know the value of $\boldsymbol{C_{ij}}$.
·Assume all allocations will be accepted.
The goal: allocate orders to minimize the total pick-up time.

Define $\boldsymbol{X}$ be a boolean matrix where $\boldsymbol{X_{ij}} = 1$ if row i is assigned to column j.
So, based on our assumption, the order allocation problem can be describe as following:
Finding i, j that minimize $\sum_i \sum_j \boldsymbol{C_{ij}} \boldsymbol{X_{ij}}$
Such that each row is assignment to at most one column, and each column to at most one row.

Therefore the order allocation problem is converted into the linear sum assignment problem. The linear sum assignment problem is also known as minimum weight matching in bipartite graphs.

We can solve the linear sum assignment problem using the Hungarian algorithm, also known as the Munkres or Kuhn-Munkres algorithm. The Hungarian algorithm has an O(n3) running time. The detail of the Hungarian algorithm is in appendix.
I used scipy.optimize.linear_sum_assignment() in python to solve the linear sum assignment, the parameter is the cost matrix of the bipartite graph and the returns are an array of row indices and one of corresponding column indices giving the optimal assignment.

Situation 1: mimic the real world situation in the whole New York city area.
First, I used XGBoost to fit taxi data in 2016 (after feature engineering, which is shown in EDA part). The target value is trip time duration. The feature importance of 43 features will be shown in the appendix.

The final XGBoost model performance:
R square in test set: 0.833
test RMSE: 253s

In this situation, I selected n observations from test set and split it into two tables. One only contains pick up latitude and pick up longitude, the other contains all the 41 remain features. Then I cross join these two table and used XGBoost to predict n^2 trip time results to construct the cost matrix ***C***. The ***C*** matrix has the real physical meaning of the whole New York city.

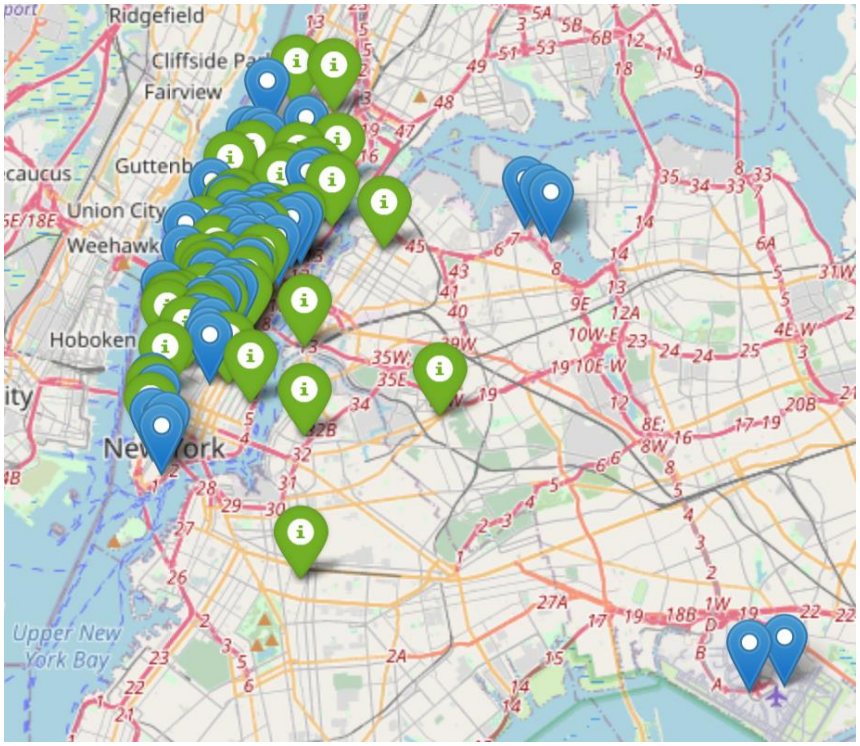

Figure 6: n^2 pairs of pick up and drop off locations where n = 50

I used drop off locations represent the locations of drivers, and pick up locations represent the locations of passengers. In Figure 6, blue represents the driver, green represents the passenger.

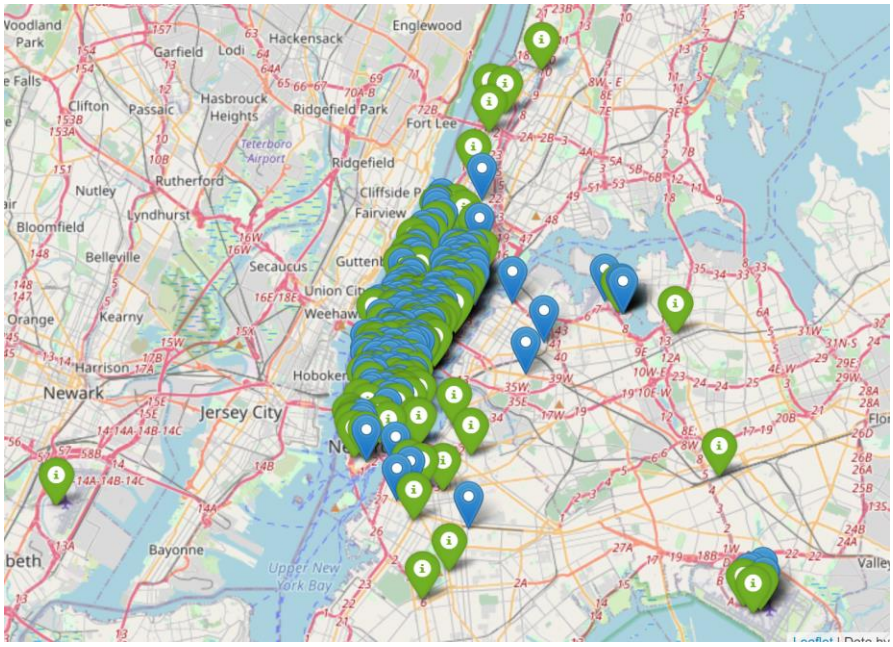

Figure 7: n^2 pairs of pick up and drop off locations where n = 200

For n = 50 and n = 200, the total pick up time/average pick up time of greedy solutions are roughly 1.04 times of that of optimal solutions.

n_50

	greedy_res	opt_res	ratio_greedy_to_opt
0	36342.538284	34888.968750	1.041663
1	43841.589371	41922.285156	1.045782
2	37423.853287	35938.179688	1.041340
3	43886.080032	42148.074219	1.041236
4	35896.432144	34171.863281	1.050468
5	36718.543289	35531.621094	1.033405
6	36304.156662	34785.671875	1.043653
7	41280.282516	39597.449219	1.042499
8	37301.845581	35990.058594	1.036449
9	41802.124146	40100.089844	1.042445

Figure 8: the greedy results and optimal results and their ratio for n = 50

n_200

	greedy_res	opt_res	ratio_greedy_to_opt
0	181015.945038	175011.062500	1.034311
1	178483.036606	171928.062500	1.038126
2	164097.616135	157729.671875	1.040373
3	166523.272133	160723.500000	1.036085
4	163163.926025	157167.375000	1.038154
5	167783.629257	162023.593750	1.035551
6	172602.586746	166465.562500	1.036867
7	160477.376297	155299.812500	1.033339
8	172307.142349	166969.156250	1.031970
9	165175.923691	159474.390625	1.035752

Figure 9: the greedy results and optimal results and their ratio for n = 200

We can see that in this simulation, optimal algorithm has little advantage compared with greedy

algorithm.

Situation 2: simulation on a simple model with clear physical meaning.

Define (x_{di}, y_{di}) as the location of driver i, where i = 1, …, n.
Where $x_{di} \sim Unif(-10,10)$ $y_{di} \sim Unif(-10,10)$
Define (x_{pj}, y_{pj}) as the location of passenger j, where j = 1, …, n
Where $x_{pj} \sim Unif(-10,10)$ $y_{pj} \sim Unif(-10,10)$
Define the pick up time for driver i and passenger j as $|x_{di} - x_{pj}| + |y_{di} - y_{pj}| = \boldsymbol{C_{ij}}$.
So we have the cost matrix $\boldsymbol{C}$, we can use the Hungarian algorithm to get the optimal solution.

sn5

	greedy_res	opt_res	ratio_greedy_to_opt
0	52.375553	52.375553	1.000000
1	37.022746	37.008285	1.000391
2	53.857689	51.220108	1.051495
3	41.968943	38.835588	1.080683
4	60.617834	51.896385	1.168055
5	45.895057	43.536754	1.054168
6	34.417168	26.886803	1.280077
7	54.252275	46.210942	1.174014
8	57.968333	40.044457	1.447599
9	46.019661	45.490647	1.011629

sn20

	greedy_res	opt_res	ratio_greedy_to_opt
0	106.070691	88.576879	1.197499
1	111.474563	97.782068	1.140031
2	133.288096	113.626176	1.173040
3	109.110234	92.846373	1.175170
4	117.554622	95.802984	1.227046
5	163.484628	125.206410	1.305721
6	139.487181	98.185986	1.420642
7	115.092646	98.122888	1.172944
8	127.526556	107.208951	1.189514
9	109.856134	86.464273	1.270538

sn50

	greedy_res	opt_res	ratio_greedy_to_opt
0	224.554393	171.077794	1.312586
1	221.178188	160.047110	1.381957
2	183.756711	138.327703	1.328416
3	222.166473	150.430425	1.476872
4	263.607124	216.850927	1.215614
5	207.926316	152.597546	1.362580
6	212.264056	175.582615	1.208913
7	250.319572	167.651946	1.493091
8	158.101041	135.705214	1.165033
9	208.340256	157.329861	1.324226

sn200

	greedy_res	opt_res	ratio_greedy_to_opt
0	598.838359	425.707637	1.406689
1	535.222194	364.894214	1.466787
2	512.111802	382.221010	1.339832
3	456.107003	332.887965	1.370152
4	469.131810	329.226557	1.424951
5	549.361187	351.046057	1.564926
6	513.849178	393.898994	1.304520
7	499.955859	343.501591	1.455469
8	561.502030	389.558552	1.441380
9	561.374426	408.371181	1.374667

Figure 10: the greedy results, optimal results and their ratio for different n

As we can see, with the increase of n, the ratios of greedy solution to optimal solution also increase. When n = 200, the ratio is roughly about 1.40. Therefore this simulation gives us an inspiration: perhaps in the small area accompanied by high-density order demand and empty cars, compared with greedy algorithm, the optimal algorithm can reduce the total pick-up time/average pick-up time greatly.

References

Shimrat, M., "Algorithm 112: Position of point relative to polygon" 1962, Communications of the ACM Volume 5 Issue 8, Aug. 1962
Harold W. Kuhn. The Hungarian Method for the assignment problem. Naval Research Logistics Quarterly, 2:83-97, 1955.
Harold W. Kuhn. Variants of the Hungarian method for assignment problems. Naval Research Logistics Quarterly, 3: 253-258, 1956.

Appendix I: The features importance of XGBoost model for taxi data in 2016.

{'total_travel_time': 5389,
'total_distance': 5360,
'dropout_center_6': 725,
'pickup_latitude': 5430,
'center_latitude': 5955,
'pick_month': 1149,
'center_longitude': 6537,
'hour': 4382,
'sin_week': 1668,
'cos_week': 1130,
'pickup_center_4': 524,
'pickup_center_2': 127,
'pickup_center_6': 860,
'pickup_longitude': 6355,
'pickup_center_5': 520,
'sin_hour': 2170,
'day_of_year_y': 2508,
'dropout_center_2': 134,
'cos_hour': 2570,
'snow depth': 628,
'dropout_center_4': 612,
'week_of_year': 2577,
'hvsine_pick_drop': 4959,
'pickup_center_3': 323,
'average temperature': 1789,
'maximum temerature': 2617,
'pickup_center_1': 710,
'dropout_center_0': 473,
'dropoff_latitude': 6487,
'bearing': 6317,
'minimum temperature': 1963,
'pickup_center_0': 310,
'passenger_count': 507,
'dropout_center_5': 523,
'day_of_year_x': 3823,
'precipitation': 964,
'vendor_id': 373,
'dropout_center_1': 492,
'day_of_week': 1984,
'dropout_center_3': 203,
'manhtn_pick_drop': 3370,
'dropoff_longitude': 5939,
'snow fall': 107}

Appendix II: The Hungarian algorithm

The following 6-step algorithm is a modified form of the original Munkres' Assignment Algorithm (sometimes referred to as the Hungarian Algorithm). This algorithm describes to the manual manipulation of a two-dimensional matrix by starring and priming zeros and by covering and uncovering rows and columns. This is because, at the time of publication (1957), few people had access to a computer and the algorithm was exercised by hand.

Step 0: Create an n by m matrix called the cost matrix in which each element represents the cost of assigning one of n workers to one of m jobs. Rotate the matrix so that there are at least as many columns as rows and let k=min(n,m).

Step 1: For each row of the matrix, find the smallest element and subtract it from every element in its row. Go to Step 2.

Step 2: Find a zero (Z) in the resulting matrix. If there is no starred zero in its row or column, star Z. Repeat for each element in the matrix. Go to Step 3.

Step 3: Cover each column containing a starred zero. If K columns are covered, the starred zeros describe a complete set of unique assignments. In this case, Go to DONE, otherwise, Go to Step 4.

Step 4: Find a noncovered zero and prime it. If there is no starred zero in the row containing this primed zero, Go to Step 5. Otherwise, cover this row and uncover the column containing the starred zero. Continue in this manner until there are no uncovered zeros left. Save the smallest uncovered value and Go to Step 6.

Step 5: Construct a series of alternating primed and starred zeros as follows. Let Z_0 represent the uncovered primed zero found in Step 4. Let Z_1 denote the starred zero in the column of Z_0 (if any). Let Z_2 denote the primed zero in the row of Z_1 (there will always be one). Continue until the series terminates at a primed zero that has no starred zero in its column. Unstar each starred zero of the series, star each primed zero of the series, erase all primes and uncover every line in the matrix. Return to Step 3.

Step 6: Add the value found in Step 4 to every element of each covered row, and subtract it from every element of each uncovered column. Return to Step 4 without altering any stars, primes, or covered lines.

DONE: Assignment pairs are indicated by the positions of the starred zeros in the cost matrix. If C(i,j) is a starred zero, then the element associated with row i is assigned to the element associated with column j.