CS241 Final Project Report

519021910913 黄喆敏

1. Introduction

There is a surge in online ride-hailing systems in the past couple years. These online services collect information about spatio-temporal demand, relying on a centralized dueling platform to match drivers with orders. Therefore, the purpose of order assignment is to find the best match between drivers and orders, which is particularly important. By analyzing and visualizing order-related data, we can solve the problem more easily. The aim of my project is promote efficient travel and improve order fulfillment rates in online system. Both **speed and reliability** are carefully considered in the project. Generally speaking, the function includes:

1） Load and store the dataset according to the user's wishes
2） Organize data smartly to implement data analysis and data visualization
3） Return similar orders to users
4） Provide time-prediction and location-prediction to users, and a QML map for visualization

2. Implementation

2.1 Filter and Dataset Loading

There are two main tables in the database, which will be discussed below.

Dataset table

ID	departure_time	end_time	orig_lng	orig_lat	dest_lng	dest_lat

ID table

order_id	departure_time	end_time

We notice that all the data can be stored into numerical type, except ID of order. So we create two table, one is specially prepared for ID, the other is for other data. To establish a consistent one-to-one match between two tables, we insert both Ride Start Time and Ride End Time into ID table, which can be seen as **'fingerprint'**. Besides, both Ride Start Time and Ride End Time are read in the form of **UNIX timestamp**. Actually, it is quite convenient to save and do some calculations in such form. So we transfer timestamp to specific time only when it is displayed to users.

To increase the loading speed, we use **lazy loading technique**. For the import fields, there are 8 check boxes on user interface, of which 5 can be chosen, and the other 3 are mandatory. The program will load the data according to the field that users choose. For the time span, there are two QDateTimeEdit widgets. Users can select time interval whatever they want. To ensure that the time interval is valid, the two widgets are connected by signals and slots. Starting time will always before the ending time. After users select the directory, the program will read all the file names in the directory, and then decide whether to read this file according to its name. So it's recommended that all the file names should not be changed.

In order to display the process of dataset loading, we use **multi-thread technique**. To set the value of progress bar, we count the rows that have been loaded in the thread. and send signals at intervals. After each .csv file is completely loaded, a signal will be sent to MainWindow, then the status bar will show the file path and name. If users are out of patience, they can press CANCEL whenever they want, and the thread will be terminated shortly. To prevent the memory leak, there will be garbage collection afterwards.

Sorting in advance can save a lot of time in the following process. So after the whole dataset is inserted into the table, the progress bar would be set to 95%, and tell users that the program starts sorting by status bar. And the working thread would then immediately sort the data in ascending order, according to the departure

time. If the user select the ID data, then ID would be sorted by using the 'fingerprint' above. We always assume that the user has little patience, so the sorting process is likewise run via multi-thread, preventing the program from being stuck.

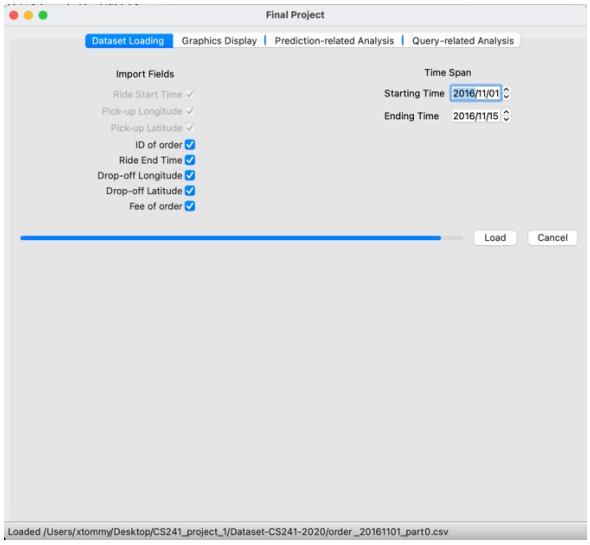


Fig.1 Interface of Dataset Loading

2.2 Graphics Display

We have used many types of different widgets in this part. Users can always make some mistakes when they input information, so we set lots of LIMITATIONS on Qt Widgets. **Signal and slots** are used widely in this part, ensuring that the range of time and grid is always valid.

To ensure the result is accurate and the program will not crash abnormally, all the functions cannot be used until data has been loaded completely. So we design **different types of QMessageBox** to inform users, whether you need to wait for data loading, or you need to import fields that have been chosen. Besides, we use **OpenGL** to produce some animation effects, displaying real-time dynamic graph.

We use **line chart** to display the spatio-temporal demand patterns, or the total revenue of the company in a certain period of time. Users can select time step, daily or hourly. If users do not satisfy with these two options, they can choose 'Customize'. Then a scroll bar will be activated for choosing time step. The number of data points can be calculated by equation below. N refers to the number of data points, T_1, T_2 refers to time span, and t refers to time step.

$$N = [\frac{T_2 - T_1}{t}]$$

If N is **less than 25**, we use QSplineSeries class to smooth the line **by interpolation**. Then a dot line will be displayed.

For other value information, like the distribution of travel time and order fees, we use **pie chart** instead. The program will first find the minimum and maximum value from the data. Then the data will be equally spaced into 10 parts. However, **for the travel time**, there are always some **outliers**, whose duration are incredibly long. To display useful information, the program will find the outliers, delete them, and then draw the chart.

2.3 Query-related Analysis

For the query-related analysis, there has not been agreement on the definition of similarity. So we allow users to **choose their own standards**. Users can select similar orders based on location, or travel time. Besides, we use **regular expression** to set limits on latitude and longitude. Users can only input numbers ranging from -180 to 180, with no more than 6 decimal places.

For location-related query, there are three patterns to choose. Users are allowed to use only pick-up location, only drop-off location, or both two locations to get the similarity. Then the program will calculate the

similarity according to **Manhattan Distance**, sorting them in ascending order. For time-related query, we use timestamp to compare both starting time and ending time.

When the program selects the **100 orders** with the highest degree of similarity, data of these orders will be displayed in a table. Remember that **data out of filter** will not be displayed in the table.

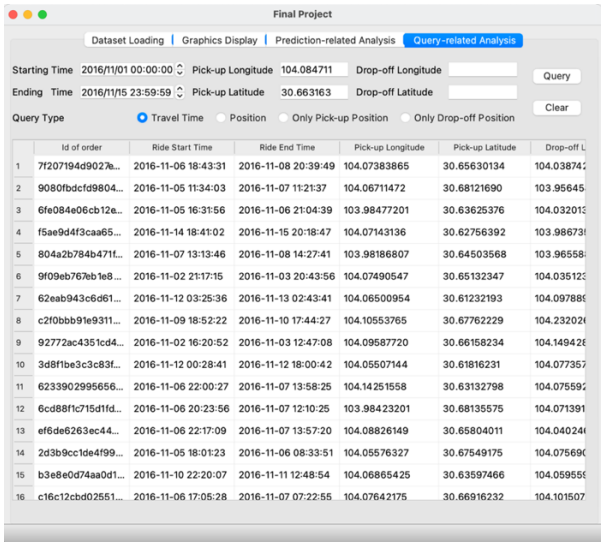


Fig.2 Interface of Query-related Analysis

2.4 Prediction-related Analysis

There are two problems in this part. One problem asks to predict the destination according to starting point. After reading the paper about Didi's taxi order dispatch model, I've found that **departure time and location** is of equal importance. Different starting time may lead to different destination, so time should also be considered as well.

The prediction is made based on **similar order**, which is comparable to **KNN(k nearest neighbors) algorithm**. We establish an evaluation function below to calculate the similarity. After several attempts, we use **Euclidean Distance** to calculate the difference of location. (Euclidean Distance is better than Manhattan Distance.) The unit of time is second, which means that time's change rate is much faster than location's. So there's **a large coefficient** in front of the location. Δ_{lng}, Δ_{lat} refers to the difference between the location input by users, and the location of data. Δ_{time} refers to the difference between the input time and the ride start time of data.

$$f(X_i) = 100000 \times \sqrt{\Delta_{lng}^{\ 2} + \Delta_{lat}^{\ 2}} + \Delta_{time}$$

After we calculate the evaluation function, we sort them in ascending order. Then we select top 10 order, calculate a weighted average position based on the following formula.

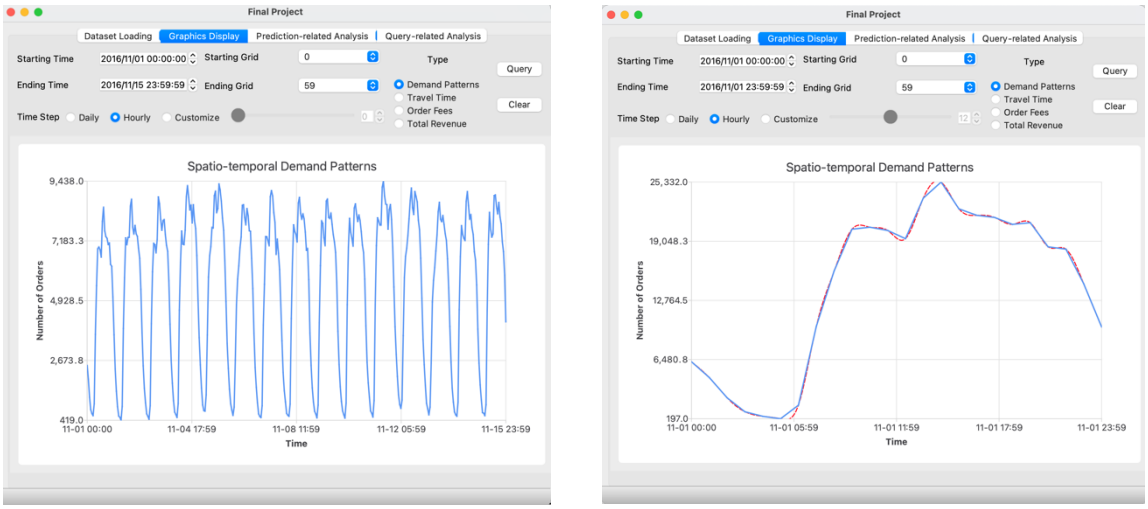
The principle of time prediction is quite similar, therefore it will not be mentioned.

$$pre_{pos} = \sum_{i=0}^{9} pos_i \times \frac{(10-i)^3}{\sum_{j=0}^{9} j^3}$$

The latitude and longitude are quite **abstract**. Many people may be confused where the predicted location is. **So we import a map plugin and use OpenStreetMap library to show the location visually.** The center of the map is set on the right side of Tianfu Square, which is located in the center of Chengdu, in the form and position similar to Beijing's Tiananmen Square. The nature of Qt's engine's integration with **Qt's meta object system** enables C++ functionality to be invoked directly from QML. Once the prediction is made, the C++ program will send a signal to QML map, updating the position on the map.

3. Analysis Result

3.1 Spatio-temporal Demand Analysis



(a) 15-day data (b) one-day data (with interpolation)

Fig.3 Spatio-temporal Demand Analysis

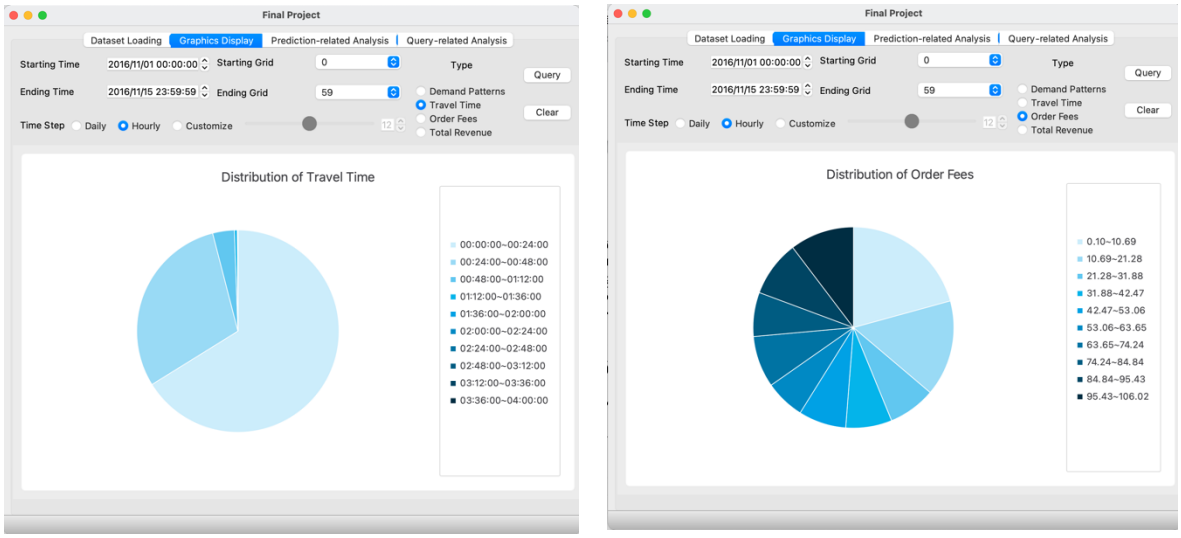
As shown in Figure 3(a), the peak number of orders per day is basically the same. The peak of Nov.11 is slightly higher than the others. Perhaps people are hurried to Double 11 Shopping Festival.

And in Figure 3 (b), we use interpolation to smooth the line. We can see that the order demand grows sharply in the morning, and then reaches its peak in the early afternoon. After that the demand gradually decreases. Besides, due to space limitation, we do not show the result of total revenue.

3.2 Travel Time & Order Fees

As shown in Figure 4(a), the travel time of a majority of orders falls into 24 minutes below, indicating that short-distance transportation occupies a large proportion. There is also some orders ranging from 24 minutes to 48 minutes.

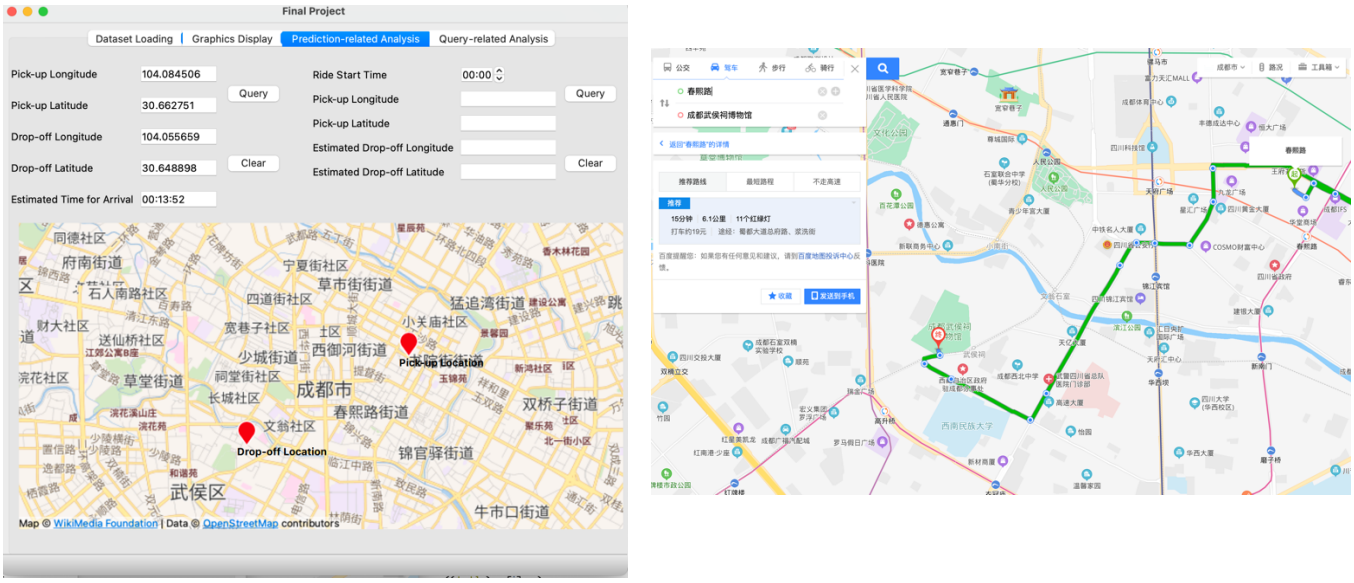
As shown in Figure 4(b), the order fee during the 15 days has a much more uniform distribution. There are a number of orders whose fees are below ¥10, and huge amounts of allowance from the platform can account for that.



(a) Travel Time (b) Order Fees

Fig.4 Pie Chart of Travel Time & Order Fees (15-day)

3.3 Prediction-related Analysis



(a) Results from GUI application (b) Results from Baidu Map

Fig.5 Prediction-related Analysis (According to 15-day Data)

We take time prediction as an example. As shown in Figure 5, we have chosen some landmarks in Chengdu to estimate time for arrival. In order to test the accuracy of the prediction, we compare the result with that from some mature map platform. The performance is quite good.

4. Evaluation and Discussion

Evaluation is done on MacBook Pro(15-inch, 2017). We run the program under Qt's Debug Mode for several times, and use QTime to obtain the evaluation result. Result is shown in Table 1. As you can see from the table, the result is quite satisfactory. Actually, the result is much better in Release Mode.

Table 1 Evaluation Result

Task	Average Time(ms)
Reading Dataset(1 day, including ID)	2030
Reading Dataset(15 days, including ID)	18970
Sorting(15 days)	19024
Graphics Display(15 days, all data)	6043
Prediction-related Analysis(15 days)	3058
Query-related Analysis(15 days)	2437

5. Acknowledgement

I would like to express my gratitude to all those who have helped me in this course.
I want to express my sincere gratitude to Dr. Ling and Dr. Jin, for sharing so much fascinating knowledge and passion for education with us. I feel so fortunate to have been placed in your class. I have learned programming with C++ for many years, but before the study of this course, it never occurred to me that we can use C++ to design such practical GUI application. During my process of programming, I've learned a mass of tricky, but interesting mechanisms, which gives me some enlightenment.
I would also want to say thank you to the marvelous teaching assistants in this course. During the whole class, both TAs have performed all their duties conscientiously, offering us a lot of help.