CS241 Final Project Report

Wendi Chen 519021910071

I. Introduction

Online ride-hailing systems like Didi and Uber have gained a huge number of users in recent years. However, for the distribution of orders by region and time is uneven, congestion and empty load often occur. So, it's natural that these companies need data-based analysis to help them arrange their taxi resources better.

In this course project, all data we have is the raw Didi orders in Chengdu, which means we need to dig up enough valuable information. And how to load, store and organize such large scale data affect the efficiency of the entire program. In my implementation, all data is loaded at the first time and organized through an array of structures, which proves effective in practice. Some simple information extraction is completed in the process of loading for later use which boosts run-time efficiency.

Also, considering users of this software are not professional data analysts, the visualization of data should be intuitive and easy to use. So, I have provided plenty of charts for users to choose from. At the same time, software needs to be robust enough to avoid run-time error when the user try to use the software from the wrong order. In the framework of the entire program, I have designed adequate exception handling mechanisms to guide users use the program correctly for the first time.

Besides, performance matters. For a GUI program, it usually gets stuck when processing time-consuming tasks like loading data and accessing Internet. As a result, I implemented certain parts of code via multi-threading, which ensures that the program can run smoothly.

The last point I want to mention is some unique features. I implemented a dynamic thermal diagram which can show the change of the order density of a certain period like an animation. In addition, I provide access to AMap so that users can get an authentic route with the route planning function.

Key Words: data organization, visualization, user-friendliness, robustness, multi-thread, thermal diagram, route planning.

II. Implementation Details

A. GUI with nearly One Window

There are three main functions—loading, drawing and analysis. To make the user experience like a natural flow, I use three *groupBoxs* to contain the widgets of these three parts respectively and put them on the *mainWindow* (Figure 1).

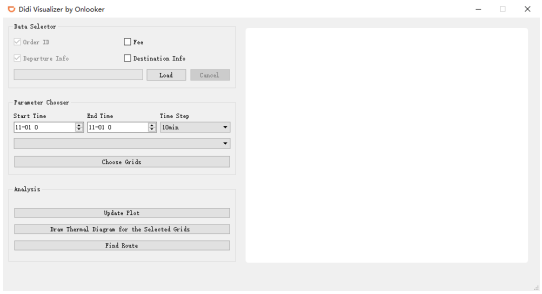

Fig. 1: Graphic User Interface

1) Data Selector: The order information has 8 fields. As the tasks we're going to finish relate to both time and location, I put the departure time together with the departure coordinates as *Departure Info*, and the same for the *Destination Info*. After the above processing, we now have only four fields, among which *Order ID* and *Departure Info* are necessary and set uncheckable. After that, users are allowed to choose the fields they like for demonstration and analysis. A *QProgressBar* is also provided to show the real-time progress of loading

2) Parameter Chooser: In the process of visualization, there are some hyper-parameters that require users to choose. For the time, I designed two *QDateTimeEdit* widgets and a *QComboBox* for users to select start time, end time and time step. For grids, I implemented it in a quite intuitive way—drag and release (Figure 2).

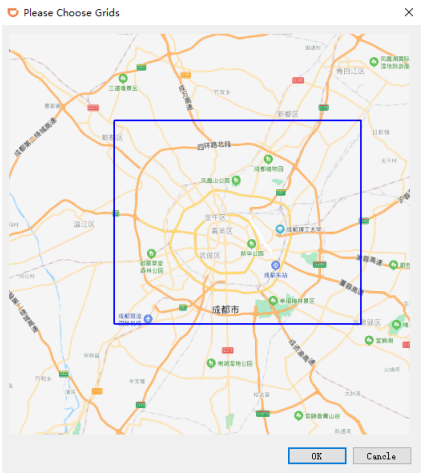

Fig. 2: Grids Choosing Window

3) Analysis: All analysis methods are contained in this *groupBox*, users can clicked one of these three *QPushButton* to activate the corresponding function.

For plot displaying, the GUI will draw certain kind of plot according to user's choice on the right of the *mainWindow*(Figure 3). That's because the user may frequently adjust the parameters and this way can avoid switching back and forth between multiple windows.

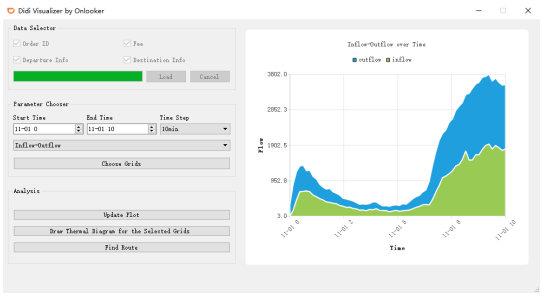

Fig. 3: Plot Drawing

As to thermal diagram and route planning, a new window is expected for displaying more details (Figure 4 and Figure 5). For example, in the thermal diagram window, users can drag the slider to change time and in route planning window, road sequence and time cost are shown on the right.

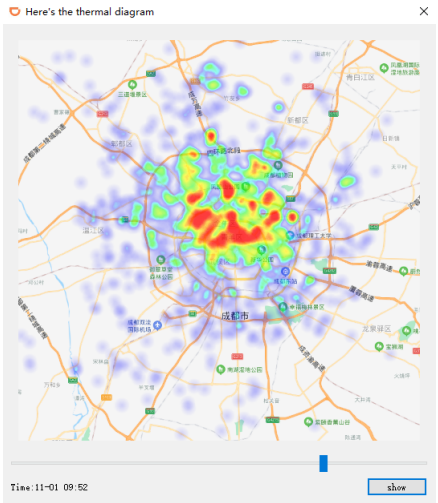

Fig. 4: Thermal Diagram

4) Limitations and Default Settings: In order to ensure the parameters users choose make sense, I set some limitations for the GUI.

- Start Time and End Time should be set between 2016-11-1 0:00 and 2016-11-15 23:00.
- Start Time should be earlier than End Time.
- Time Step is limited to 10min, 30min, 1 hour and 2 hour.

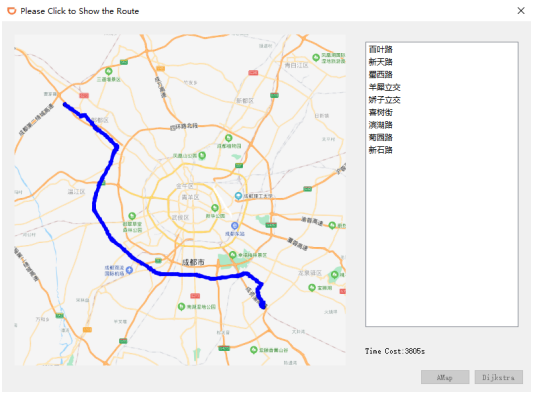

Fig. 5: Route Planning

- Data should be loaded first and then users can use any of the analysis functions.

Default settings are listed as below.

- Start Time is 2016-11-1 0:00 and End Time is 2016-11-1 1:00.
- Time Step is 10 min.
- Default plot is *Real-time Demand* plot.

B. Data Loading

As the order data is divided by day and each day has 5 batches, the number of files is still a lot. So it's a better idea to let users choose 'a folder' instead of 'files' (Figure 6).

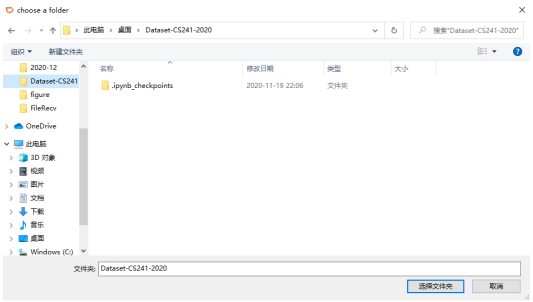

Fig. 6: Folder Choosing Window

After the folder path has been specified, the data loader will try to load *rectangle_grid_table.csv* first to get the geographic coordinates of the area. With the geographic information ready, the pre-processing work is also being carried out simultaneously, which classifies each order into their grid correspondingly. The range and value of progress bar is set according to the number of files in the folder.

All of the above processes are completed in another thread. This prevents congestion between the process of data loading and the GUI and users can cancel the loading at any time they want.

Considering that the overall data volume is about 100MB, which is not very large, it is possible to store all the data in memory instead of using a database. As mentioned in *Introduction*, I organized all the data in a vector of structure. The constituent elements of the structure are listed as follows. In addition to the raw data, the area is divided into 10×10, 100×100 and 500×500 grids, according to which each order has elements like *i, index_i* and *r_index_i* that mark the relative departure coordinates.

TABLE I: Elements of structure

order_id	dep_time	...	i	...	index_i	...	r_index_i
0cc44...	147...	...	5	...	249	..	49
...	...	...	...	...	...	...	...

Define the boundes of the map as x_{start} and x_{end}, we can simply calculate i in this way:

$$i = \frac{lng_{orig} - x_{start}}{x_{end} - x_{start}} \tag{1}$$

Although field checkers are provided, according to actual implementation, it is found that the main time consumption is file reading. So all fields are read at first to avoid re-loading and the response of field checkers is used to remove some plot choices to simplify the GUI.

C. Drawing Plots

I've provided 6 different kinds of plot for users to choose from including Real-time Demand, Inflow-Outflow, Inflow-Outflow Ratio, Travel Time Distribution, Fee Distribution and Total Revenue. The display forms includes line chart, pie chart, histogram and area chart, which can vividly display the characteristics of each group of data. Meanwhile, if users try to draw a plot without loading data, a message box will be shown to guide the user do in the right way.

An obvious problem is that when there are few data points, line chart will not be smooth enough. To solve this, I limited the fewest point number to 5, implemented Lagrange's interpolation and used *QSplineSeries* to smooth the curve. As we know, the polynomial obtained by Lagrange interpolation does not converge at the boundary, so I just put the interpolation points near the center into the *lineSeries*.

$$P_n(x) = \sum_{j=0}^{n} y_j l_j(x) \tag{2}$$

$$l_k = \frac{(x - x_0)...(x - x_n)}{(x_k - x_0)...(x_k - x_n)} \tag{3}$$

D. Drawing Thermal Diagram

Thermal Diagram is used to reflect the relative density distribution of orders within a certain period of time. Therefore, the main problem is how to convert density information into color information.

According to a reference article on the Internet[1], we can first convert the density information into gray information, and then use the color look-up table and draw a gradient circle to draw a heat map.

Define the number of one certain point is $count_i$ and max one as $count_{max}$, the gray information can be calculated in this way:

$$\alpha = \frac{count_i}{count_{max}} \times 255 \tag{4}$$

And then convert gray information to color:

$$color = colorList[\alpha] \tag{5}$$

E. Route Planning

As all the data we have are just order information, we can make a simple assumption that the density of vehicles in a certain area is proportional to the number of orders. Therefore, We need to plan a route to bypass the crowded area.

Divide the map into 100×100 grids. Like the gray calculation, the weight of each grid can be calculated in this way:

$$weight_i = \frac{count_i}{count_{max}} \times 20 + 1 \tag{6}$$

In order to make full use of data, I implemented Dijkstra algorithm which doesn't rely on real-world roads but finds the best route between 'pixels'. Of course, this is not enough. However, if we use an open source third-party map library, the entire software will become bloated, which is contrary to the original intention of lightweight development. So I finally chose to call the AMap API. By clicking on the GUI, the program can obtain the route information from AMap through Internet, and then decode the response and draw the route by *QPainter* (Figure 5). Also, for the network access is time-consuming, these piece of code is implemented via multi-threading.

III. Results

A. Data Loading

After testing, the loading progress bar can work normally, and the user can cancel the data loading at any time and re-select the folder (Figure 7).

[1]https://blog.csdn.net/gongjianbo1992/article/details/104566768

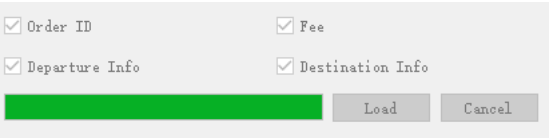

Fig. 7: Progress Bar

B. Drawing Plots

After interpolation, the available data points almost doubled, and there was no surge near boundary data (Figure 8).

All charts can be zoomed in by specifying a local area. Due to space limitations, only pie chart and histogram are shown below(Figure 9 and Figure 10).

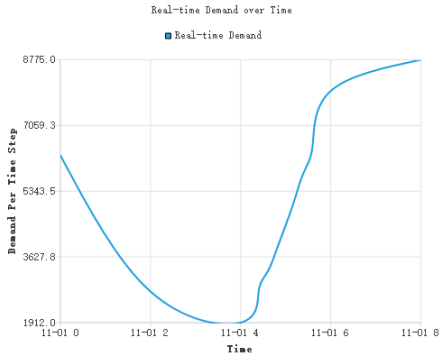

Fig. 8: Line Chart of Real-time Demand

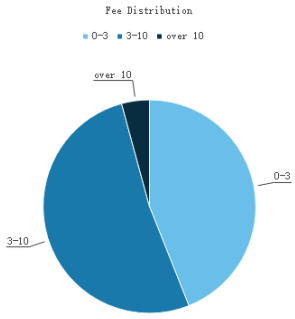

Fig. 9: Pie Chart of Fee Distribution

After we visualize the data, we can find the following conclusions.

- The number of orders changes cyclically on a daily basis with the peak at around 1 p.m.(Figure 12)
- Different areas have different ratio of Inflow-Outflow. At places like airports, outflow is obviously more than inflow.
- People who live in the suburbs tend to spend more commuting time and higher commuting expenses.
- The company's profit is basically proportional to the number of orders.

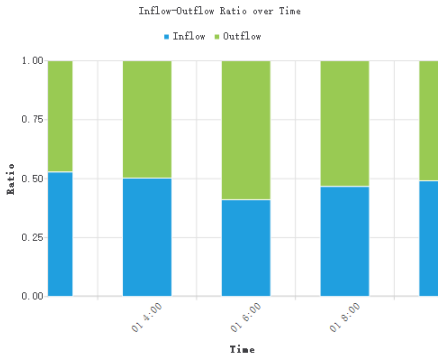

Fig. 10: Fee Distribution of Inflow-Outflow Ratio

C. Drawing Thermal Diagram

As what is shown in Figure 4, users can drag the slider to change start time or click 'show' to get an animation of density changes. At the same time, it can be found that the use of gradient colors can make adjacent circles merge with each other, making the image very natural. And then we have the following conclusions.

- Urban areas have more traffic than suburbs and are more prone to congestion.
- Large capacity often appears at intersections, schools, downtown and other places where large people flow.

D. Route Planning

In the Dijkstra mode, the route planning function will avoid those crowded areas, but this route is virtual for it doesn't consider real roads.The result of AMap route planning is shown in Figure 5.

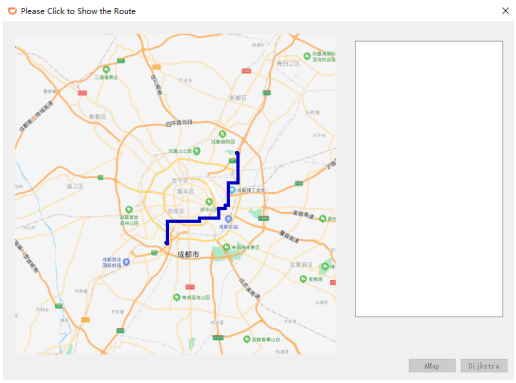

Fig. 11: Dijkstra Route Planning

IV. Discussions

A. Performance

The average time of data loading on my PC is about 11s, which is quite acceptable. The testing data is listed below.

TABLE II: Data Loading

Average Loading Time	10.67s
Maximum Loading Time	11.59s
Minimum Loading Time	9.83s

There're mainly two ways to speed up the loading process.

- Allocate more threads for data loading. Currently, we have only one thread for data loading, and it'll be helpful to use more threads to handle parallel tasks like file reading.
- In fact, we have never used the order_id field, we can use more refined operations to read the file to avoid the construction of *QString*. If we write all the parsing function ourselves, it'll definitely be faster.

During the run-time of the entire program, it often traverse the entire data set, because 10^6 is not a very large data. Moreover, usually we only traverse the data of a certain period of time. For the data of the 10^4 level, real-time calculation can be completely achieved. For example, in the process of drawing a heat map, it draws as it calculates. However, if the data level is larger, we had better extract key data and delete the redundant data to boost run-time efficiency. Or in another way, we can just write a 'cache' to avoid over-calculating.

B. Interesting Results

During the design of the project, I always wondered whether the data set has a lot of redundant data. Because there are no holidays or other special days from November 1 to November 15. From the image we can see that the changes in the daily order distribution are similar in terms of time and space.

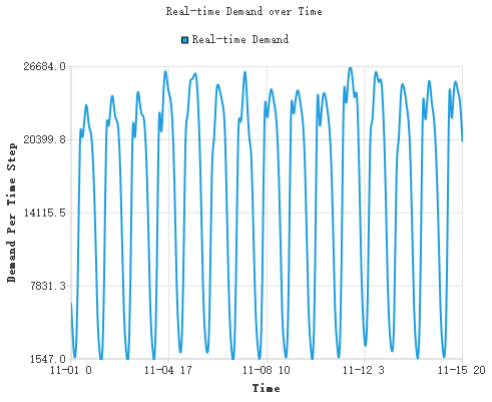

Fig. 12: Demand from Nov.1st to Nov.15th

After communicating with some classmates, I learned that we can actually use these data to do some machine learning algorithms, such as CNN. Using the network with feature extraction, we can conduct data mining more efficiently. However, due to the time limit of this project, I have not been able to truly implement this idea, which is a little regrettable.

V. Acknowledgement

Great thanks to Dr.Ling and Dr.Jin. From their lectures, I've learnt numerous useful methods and thoughts for solving problems and writing reliable programs.

Great thanks to TAs for their hard work and helpful guidance.

Great thanks to Peiyu Chen, Dept. of CS, SJTU. He helped me with some basic but important steps of designing program and writing essay.

Great thanks to Hanyu Ye, Wenhao Chen and other classmates. I benefited a lot from talking with them.