The Ride-Hailing Analyzer

BY XUXIAN

Shanghai Jiao Tong University

Abstract

The Ride-Hailing Analyzer is an efficient, reliable application to visualize and analyze the the Ride-Hailing order data of Chengdu City. It provides user a easy way to access to infomation like order frequecy (of certain grid area of Chengdu), travel time and fee distribution. The analyzer also provides clear interface for users to load certain dataset, tune the parameter and query orders similar to their input. The app works with SQLite and QThread to achieve a higher performance and a more fluent user experience.

1 Introduction

The main goal of this application is to get users access to ride hailing order data, display the data in a reasonable way and furthermore, help users predict travel time, fee or other useful data for their own journey. This report aims to fully introduce the project from low-level implementation to high-level user interface and will be organised to three part:

- **Implementation Details:** Section 2 focuses on how the app realize its function, including management of analysis event, approach of analysis and UI design.
- **Analyze Result:** Section 3 mainly demostrate tht analyze result.
- **Performance Discussion:** Section 4 discuss on the efficiency, scalability and robustness of the application.

2 Implementation Details

2.1 Management of Analysis Event

The design to manage the analysis events forms from the idea of servers.

The project use a *manager*, which inherits from QThread, to manage all analysis events. Once open the application, the manager is started and using QEventLoop to running in the background. All the analysis event in the project is a QThread which can emit signal like *process* and *success*. Therefore, the manageris designed to be a queue of thread and highly connect with the main thread and other child thread using signals and slots.

Whenever the main thread(the client) start a analysis, it sends the thread(a request) to the manager(the server). If the queue of request is empty, the manager start the event at once. Otherwise, the manager first check if the queue has an identical request, if exists, it would be cancelled and removed. Finally, the manager add the request to the end of the queue.

This design can make the app enable to perform the following behavior:

- Two irrelevant analysis events triggerred continuously would be organised to be processed one by one.
- When two identical analysis events are triggerred continuously, the former one would be cancelled and replaced by the latter one.

- A read database event would stop all other analysis and be processed at once.

2.2 Analyze Approach

2.2.1 Plot Chart

When plotting chart, the application firstly user *select* sentences to choose the corresponding orders and sort them by departure time, travel time or fee. Then the app iterates through the selected orders and continuously push stastics result to the end of lineseries. The order per hour is defined as:

$$O_{\text{hour}} = \frac{O_{\text{timestep}} \cdot t_{\text{hour}}}{t_{\text{timestep}}}$$

2.2.2 Related Order Search

When search the orders related to the users' input, the app firstly insert a column *relativity* into the database and calculate it for the order whose departure time is close to users' input(±20min). Use d(a,b) to denote distance between pos a and b, o_{order} to denote origin pos of order, and e_{input} to denote the end pos of input. The value of *relativity* is defined as:

$$r = d(o_{\text{order}}, o_{\text{input}}) + d(e_{\text{order}}, e_{\text{input}})$$

The app selected the orders with least value of r, and the expected travel time and fee is derivated from the average result of the most related order.

2.3 UI Design

The core idea of the UI design is to provide more efficient service and give users a more smooth experience. Some methods or tricks is implemented to balance the trade off betweent the analysis duration and the GUI fluency.

- To plot the number of order over time costs relatively longer time and would disturb the user. Instead of process the data and plot the chart sequentially, the analyzer synchronize the two step by putting the lineseries into the QThread. The chart is drawn from left to right as processing the data. During process, the chartview iterates through the time span and draws each-day's chart. After process, users can scroll the chart to saw everyday's chart. The design slightly increase the analysis duration, as a compensation, users get a better experiance and feel it consumes less time.

- To have a indirect interface in the related order searching part, the application uses QWebEngineView and QWebChannel to interact with amap API and javascript. The widget load html file through URI. When click on map or select the related order, js object *marker* would be created on the map to provide a insightful visualization of the order infomation.

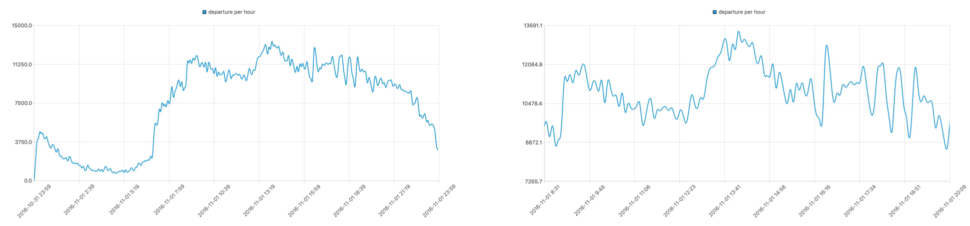
- The frequency of the emition of signal progress is tuned to balance the trade off between the fluency and consumed time.

2.4 Others

The project works highly with Sqlite in-memory database to provide high-efficiency service and simply the implement of sorting and selecting. A static mutex is used to promise exclusive access to the database and the technique of *transaction* and *rollback* is used to protect the database when an event is cancelled.

3 Analyze Results

3.1 Spatio-temporal Demand Pattern



(a)daily data (b)rubber band selected data

Figure 1. order per hour for all gird in 2016-11-1

As Fig.1(a) shows, the number of order takes off around 7 a.m. and drops gradually from 9:30 p.m to 11:30 p.m.. During the most busy phase, there are around 13000 orders per hour. Fig.1(b) selects the relatively busy phase of a day, from about 8:30 a.m. to 8:00 p.m., in the rubber band mode. It shows in the relatively busy hours there are several certain peaks like 9:00 a.m. and 2:00 p.m., which may be the timepoint people start to work.

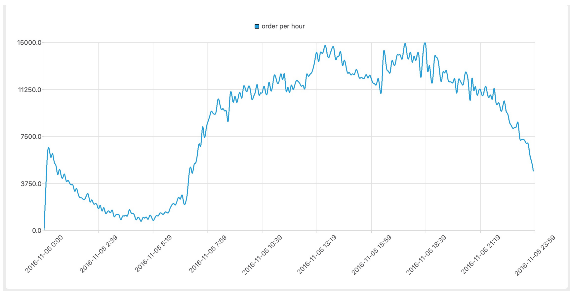
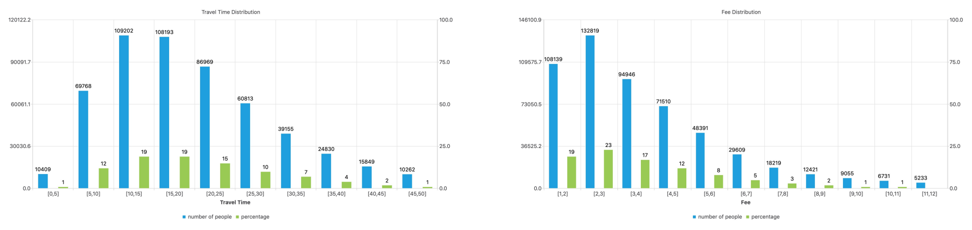


Figure 2. order per hour for all grid in 2016-11-5

Comparing Fig.1(a) with Fig.2, it's also easy to find that the peak around 8:30 a.m. disappeared in 2016-11-5. A reasonable explaination can be 2016-11-5 is Saturday and people do not need to work.

3.2 Travel-time and Fee Distribution

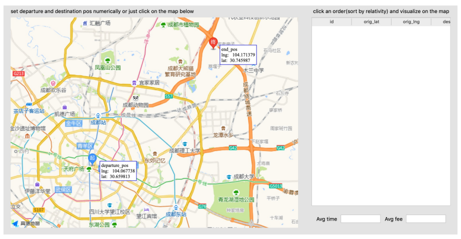


(a)travel time histogram (b)fee histogram

Figure 3. travel time and fee distribution from 2016-11-1 to 2016-11-3

Fig.3 demonstrates both number of people and the corresponding percentage of each interval. The travel time mainly distributes in the range of 5 minutes to 30 minutes while the fee mostly falls into the range of 1 yuan to 6 yuan. The pattern of the two distrubutions fit with each other. They both increase quickly in the lower range and drops slowly in the higher, which shows the longer travel time correspond to a higher fee.

3.3 Related Order Search

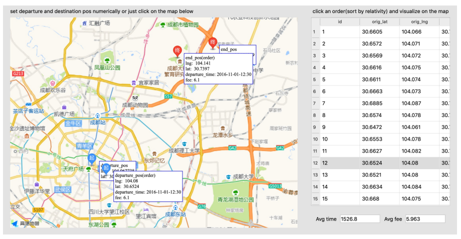

set departure and destination pos numerically or just click on the map below

click an order(sort by relativity) and visualize on the map

id	orig_lat	orig_lng	des

Avg time | Avg fee

(a)set query information


set departure and destination pos numerically or just click on the map below

click an order(sort by relativity) and visualize on the map

	id	orig_lat	orig_lng	
1	1	30.6605	104.066	30.
2	2	30.6572	104.071	30.
3	3	30.6569	104.072	30.
4	4	30.6616	104.075	30.
5	5	30.6611	104.074	30.
6	6	30.6663	104.073	30.
7	7	30.6885	104.087	30.
8	8	30.6574	104.078	30.
9	9	30.6472	104.061	30.
10	10	30.6553	104.078	30.
11	11	30.6627	104.082	30.
12	12	30.6524	104.08	30.
13	13	30.6521	104.08	30.
14	14	30.6634	104.084	30.
15	15	30.668	104.075	30.

Avg time 1526.8 Avg fee 5.963

(b)search and visualize related order

Figure 4. related order searching

Fig.4 shows the graphic user interface of related order searching module. Users can set the departure and destination position on map (by click) or input the longitude and latitude numerically. The searched related order is shown in the table besides (in Fig.4(b)). Users can click the order to visualize the order on map and compare with their input. The expected travel time and fee is below the table.

4 Performance Discussion

This part dicuss the performance of the application maily about its efficiency and robustness.

4.1 Efficiency

The test is done on MacBook Pro(13-inch) with 2.4 GHz Intel Core i5 processor and 8 GB 2133 MHz memory. The test uses QElapsedTimer to record time consumed. The profile result is derivated from the average of five rounds of test.

Test Event	Time Consuming(ms)
Build Database(1 day)	1348.0
Build Database(15 day)	21378.2
Plot order-time chart(all grid, 1 day)	594.4
Plot order-time chart(all grid, 15 day)	17836.8
Distribution Analysis(travel time, 1 day)	768.5
Distribution Analysis(fee, 15 day)	11439.8
Related Order Searching(search in one-day dataset)	93
Related Order Searching(search in all dataset)	785

Table 1. Efficiency Profile

Since a pause is added when the former-day's chart is plotted and the next is going to be processed, the plot chart(15 day) consumes more than 15 times of the time of plotting chart(1 day). However, the users experience way be a little better. To event like distribution analysis and related order searching, the application shows a relatively nice performance in the regard of efficiency.

4.2 Robustness

The application is robustness in showing the following performance.

- The data shared between different analyse event is highly protected. For example, both the database and the queue in the manager is protected by a mutex.
- The app would not be influenced by users irregular operation. For example, to click buttons repeatedly, to change the input during procession, or to trigger a event without necessary data would not cause abnormal operation of the app.