Analysis and Visualization of Online Ride-Hailing Order Data

Project for Principles and Practice of Problem Solving

1. Introduction

1.1 Goals

- The number one goal I set for the application is for it to able to accept any data-set as long as it is in the same format. The application should not have any assumption about the location or time range of the data-set.
- The application should also process at reasonable speed, long processes should have a progress bar and should not freeze the application.
- The application should also be as user friendly as possible, with easy to understand design and easy to use controls.
- The application should be able to plot graphs of demand over time, average travel time over time, revenue generate over time, etc. It should be able to generate bar charts of the average distribution of orders, travel time and revenue generate in a day, which is useful for identifying peak hours.
- The graphs should be able to be filtered by time and the starting location or the destination location, the location filter uses grid IDs given by the raw data.
- For the elective tasks, I have decided to implement map-related analysis and prediction-related analysis.

1.2 Data structure

To be able to plot the graphs required, the order data needs be summed together, like so:

Time interval	Grid ID	Number of orders	Average travel time	Revenue generated
2015-11-16 0:00	45	1234	12:34	123456

The table stores the summed data within one hour after the **Time interval** field. To store these data, it's easier to store them as `QList<QList<int>>`. For the time interval, the raw data already has it in the form of seconds since epoch (we just need to round it down to the nearest hour).

The bar charts should show the distribution throughout the day, so we can just take the table above and sum each time of the day together.

1.3 Filter by time

User should be to select starting time and ending time within the total given time range of the raw data.

1.4 Filter by location

The filter should show 10 by 10 translucent buttons on top of the map of the region in the data, the buttons should be snapped to the correct location on the map. The user should be able to toggle the buttons to include or exclude the data in the corresponding grid in the graph.

1.5 Map

There should be a map tab being able to display heat maps, users should also be able to select locations and retrieve information about the location selected.

Heat map can be generated by summing all the number of orders for each grid (from the table shown above) and mapping the values to an opacity value, then draw the colored squares with the corresponding opacity on to the map.

1.6 Prediction-related analysis

For the prediction-related analysis, I have decided to use the map and allow the user to select a grid. When a grid is selected, a heat map of the distribution of destinations is shown, thus an order from the grid selected is more likely to go to the more “heated” grids.

The user can also select 2 grids to get information such as the average time taken and the average fees of orders from the first grid to the second, which is predictive for a new order of the same starting location and destination.

2. Implementation Details

2.1 Loading the grid table

For better performance, only the most first grid is read the get the most south-west vertex and the width and height of one grid. Using these values, a `QGeoShape` is generated to represent the total region, which is used for reverse geo-coding, area calculation and to pass to QML to display the map. All of the values are wrapped in a `struct`.

2.2 Loading order data

This is my first time dealing with such large amount of data (about 2 millions lines of csv). Reading the files one by one is certainly not an option. After going through Qt documentations, To avoid dealing with low level API like mutexes and locks. I have decided to use `QtConcurrent::mappedReduced` function to concurrently read files and reduce it onto one table. The general idea is as such:

1) Generate a list of valid files names and use list as the sequence being mapped.
2) In the map function, read the file line by line, sum to data onto an empty table and return the table.
3) In the reduce function, merge the tables together.

Since we need to filter by starting *and* ending locations, we need 200 rows of data for each time interval, this gives $200grids \times 15days \times 24hours = 72000rows$, and we need to later look up these values by the grid ID and time interval, to avoid linear time look up, we can either ensure the table is sorted or use a hash table with grid ID and time interval as keys. The latter option is clearly better.

A `QString` of grid ID is generated where the data filtered by the starting locations have values of “000” to “099”, while the ending locations have values of “100” to “199”. The string is appended by the time interval to serve as the key, the value is stored as QList<int> where the list items are { number of orders, total travel time, total revenue }.

We also need to store data about destinations of each starting location for the predictive analysis. For this we only use grid ID as keys and QList<int> with 300 items as the value, the list stores number of orders, total travel time and total revenue for each destinations. This part of the data is used to generate a heat map of the destinations for each grid and to display information of orders between 2 grids.

2.3 SummedDataModel

The application implements Qt's model/view approach, where the data is manipulated by the *model* while the display is handled by the *view*. SummedDataModel is a subclass of QAbstractTabelModel as the model for QChartView. The model stores the hash table generated but dynamically generate the actual table for chart view based on the filters. The class provides methods to return heat map values based on the date stored in the hash table, and also methods to return number of orders, total travel time and total revenue between 2 grids given starting grid ID and ending grid ID.

2.4 BarProxyModel

BarProxyModel is a subclass of QIdentityProxyModel that serves as a proxy of SummedDataModel to sum the data onto 24 hours to show average distribution of data by time of day.

2.5 SummedChart

SummedChart is a class that manages the chart shown in the application. The class is responsible for updating the axis and swapping the type of chart to view.

2.6 Filters

The time filters are simply QDateTimeEdits, while the location filter is handled in QML.

The signals sent by these filters updates SummedDataModel which will adjust the outputs for the data() and rowCount() methods to update the chart.

2.7 Maps

Maps and location filters are implemented via QML maps,with Openstreetmap as the plugin used for the map API.

The filter view is locked to the QGeoRectangle sent from loading the grid table, and 100 toggle buttons are appended on to the filter. Toggling the buttons sends the signal with the corresponding grid ID to the c++ code.

The map view is located in second tab (the first tab is the graph view). It draws the grids using QML ListModel and MapItemView. The main difference between the map and the filters is that the grids are locked on to the coordinate of the map, so user can use gestures to drag or zoom in and out of the map. The function loadHeatMap recieves the signal from c++ with the list of values to draw the heat map. Similar to the filter, selection of grids also send signals with the corresponding grid ID to the c++ code.

2.8 Reverse Geo-coding

I have also implemented reverse Geo-coding for the application to show the city and country from the coordinates given by the data set. The coordinate of the center of the region given is sent to Openstreetmap's reverse Geo-coder and the signal to be sent back is connected to another slot to display the location information on the application.

3. Results

Please refer to the video for more detailed demonstration of the application.

3.1 Loading the data

User is able to open a file dialog, select the folder containing the data, and load it onto the application. The application is able to load the given data set under 5 seconds with a progress dialog showing the loading progress graphically.

3.2 Charts

The user can select different chart views including 4 types of line charts: demand(orders), average travel time, average revenue per hour and total revenue. The line charts are always plotted against time with one hour being the minimum duration. The user can also view the bar chart variations of demand, travel time and revenue per hour, the difference is that the data is summed to one day for easier analysis of the data pattern for an average day.

The graph can be filtered by time and location. The location filter is especially intuitive to use as user can see the area represented by each grid with the map underneath it. The user is also able to clear or select all of the grids via the push buttons below the filter.

3.3 Information Board

On the left side the application, general information about the data is displayed, including:

- The name of the city of the data and the area spanned by the grids.
- The filtered time range and the total time range.
- The total revenue and the average revenue per hour (filtered and selected).

3.4 Map

The map is displayed another tab, the grids are drawn on the map and the user is free to use gestures to navigate around the map. The user can view heat maps of orders the start from the locations or orders with the locations as destinations separately. The user can select any grid, and information about this grid is displayed, including the grid ID and the number of orders from/to this grid, and a heat map is displayed showing where the passenger is likely to go from the grid. Upon selecting another grid, information about the orders from the first grid to another is displayed, including the total number of orders, the average travel time and the average fees.

4. Discussions

In this section I will discuss about the development experience, the difficulties I faced and the improvements that can be made.

4.1 Performance

I am quite satisfied with the resulting performance of the application, considering I was firstly using `QtConcurrent::run` and reading the files one by one, I had to use a small fraction of the data to test as it would take minutes to finish loading. After discovering and implementing `QtConcurrent::mappedReduced`, I was able to load the entire data set in about 90 seconds. The problem was using `QList<QList<int> >` as the data structure, I had to use linear time look up to sum to data to the correct row. After switching to `QHash<QString, QList<int>>` as the data structure, the performance greatly improved to loading under 5 seconds. The hash table also allowed me to store other types of data like the total time range which can be conveniently accessed via the custom key I assigned. After summing the data on to tables of much smaller scale, the application had no issue processing them quickly and no multi-thread programming is needed.

4.2 Improvements to be made

- More testing is needed. The application went through basic tests and the bugs revealed during the testing are fixed. However the basic tests are not enough to conclude that the application is bug free.
- Better design. After running the application a few times, I realized that the data patterns for each day is almost identical, this made the bar charts unnecessary as their purpose is to average out the data patterns for every day.
- The default time edit widgets by Qt is not very easy to use, having to either type out the date and time or use arrow keys to move up and down between times, a custom QML based pop-up date time picker would be ideal.
- Speaking of QML, which is something I learned mid-way into development (I was trying to avoid using anything I did not know already, but map views can only be implemented via QML), QML just looks so much better than Qt widgets, I wish I had coded the entire front-end in QML and let the back-end heavy lifting be handled by c++.
- A heat map with higher resolution. Initially, I thought it was impossible to generate high resolution heat maps quickly as the number of data needed goes up by a square factor. After actually implementing the heat map, I realized that a heat map with at least 30 by 30 grids is plausible. The 10 by 10 grid heat map is clearly underwhelming and does not show the distribution of orders clearly enough.

4.3 Conclusion

This project is certainly among the most interesting tasks I've ever done in my academic life. I learned what developing an actual large scale application is like. Compared to solving problems with code, organization is much more important, otherwise you will lost in the sea of messy code you made and be stuck in debug hell, which is a trap I fell into many times. In the end, I wish there was more time for me to fine tune the GUI of the application, improve the features and tests for possible bugs, but I am proud of what I have made.