Final Project Report

519021911325 刘畅

Rough Introduction:

The main purpose of this GUI is to analyze and visualize Online Ride-Hailing order data, and in order to achieve this, functions of my GUI include the following parts:

1.Load data (order information and grid information);

2.Display table of data loaded;

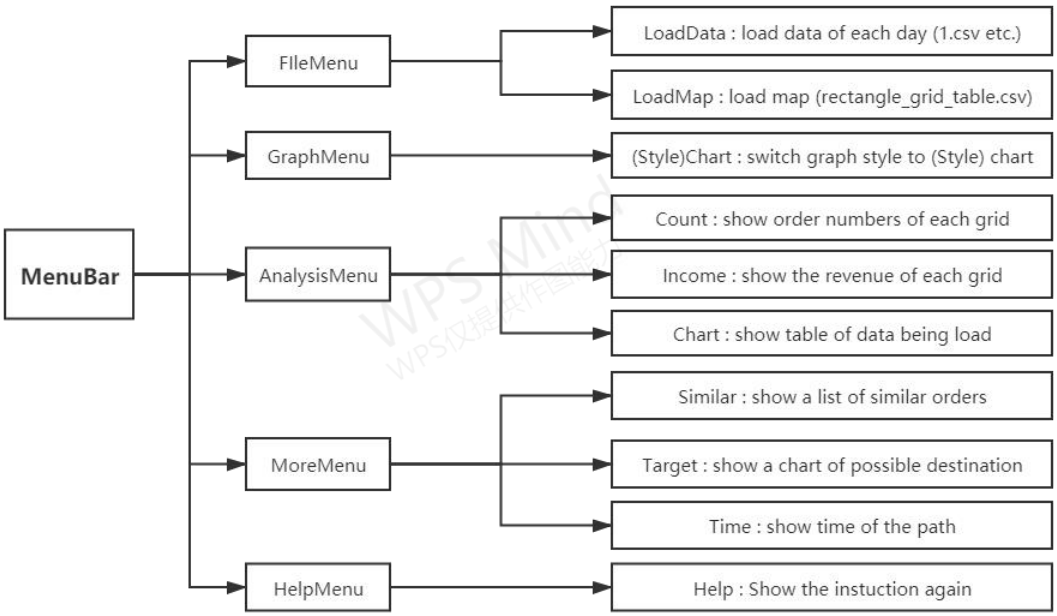
3.Draw graph (line graph, bar graph, pie chart);

4.Show possible destinations from a certain grid;

5.Calculate time it may take from one place to another;

6.Find a set of similar orders.

And here is a mind-map to show the rough structure of my GUI.



Implement Details:

The First problem, of course, is to design a rough structure. And to avoid complicated problems, I choose to use three sub-windows in exactly the same place, to show table, chart and help text. As the user chooses different functions, corresponding widget will be shown, and the other two will be hidden. And as a user, you will not see any strange actions.

Then my explanations will focus on the functions said above.

1. Data Loading (myThread.cpp & mainwindow.cpp Line 79-178)

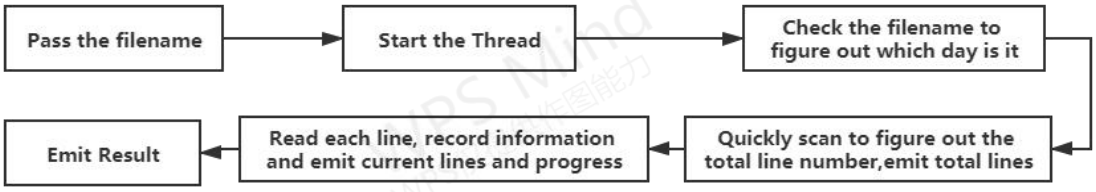
Click **LoadData** and choose the **.csv** file you would like to load. And a thread will start, which is initialized with the filename of your file and will emit messages of current lines read, total lines, date of your data, and final result. There are slots in **MainWindow** that will receive those message and display progress in a bar. In this action, multi-thread is applied.

LoadMap is similar to **LoadData**, but because the file is very small, and can finish nearly instantly, I didn't use multi-thread, but the result looks exactly the same.

Of course you shouldn't load the map data in **LoadData** or load data of each day in

LoadMap, which is illegal. So before the load action starts, it will check whether the file is valid, if not, it will end and pop out a warning window.

Below is a mind-map that roughly explains the working process of **myThread**.



Work of Load Thread

2. Pre-processing Data (counterthread.cpp & mainwindow.cpp Line 1315-1360)

As in the future actions, several statistics, such as total order number, total revenue, and starting and ending grid ID for each order are needed, so there will be another thread to handle those data and thus save much time in the future.

After you load your data and press the **PushButton** to finish the load action, you are required to choose the range of your data, either using the **Slider** or the **SpinBox** is OK.

For each order in the list, it will first figure out the grid ID of starting and ending place, to be specific, it will traverse the map and find a suit ID that includes this place in both latitude and longitude range. And the result is recorded in a list and being emit.

For count and income, after finding the ID, the corresponding element will be added by 1. For Example, if the order we are looking into starts from grid 55, then the action would be : result[55]+=1, total_income+=fee_of_this_order.

Of course, similar to load action, the process data is being transmitted, so there will still be a progress bar showing how much work has been done.

Below are the functions that judge **Starting Grid** and **Ending Grid**.

```
int CountThread::judgeNum(QStringList list)
{
    // lng : data 3 -> map 2,4 (2<4)
    // lat : data 4 -> map 3,5 (5<3)
    double lng = list[3].toDouble();
    double lat = list[4].toDouble();
    //qDebug()<<lng<<lat;
    for(int i=0;i<100;++i)
    {
        QStringList tmp = map[i];
        double v2 = tmp[2].toDouble();
        double v3 = tmp[3].toDouble();
        double v4 = tmp[4].toDouble();
        double v5 = tmp[5].toDouble();
        if(lng<v2) continue;
        if(lng>v4) continue;
        if(lat<v5) continue;
        if(lat>v3) continue;
        return i;
    }
    return -1;
}
```

```
int CountThread::judgeArrive(QStringList list)
{
    // lng : data 5 -> map 2,4 (2<4)
    // lat : data 6 -> map 3,5 (5<3)
    double lng = list[5].toDouble();
    double lat = list[6].toDouble();
    //qDebug()<<lng<<lat;
    for(int i=0;i<100;++i)
    {
        QStringList tmp = map[i];
        double v2 = tmp[2].toDouble();
        double v3 = tmp[3].toDouble();
        double v4 = tmp[4].toDouble();
        double v5 = tmp[5].toDouble();
        if(lng<v2) continue;
        if(lng>v4) continue;
        if(lat<v5) continue;
        if(lat>v3) continue;
        return i;
    }
    return -1;
}
```

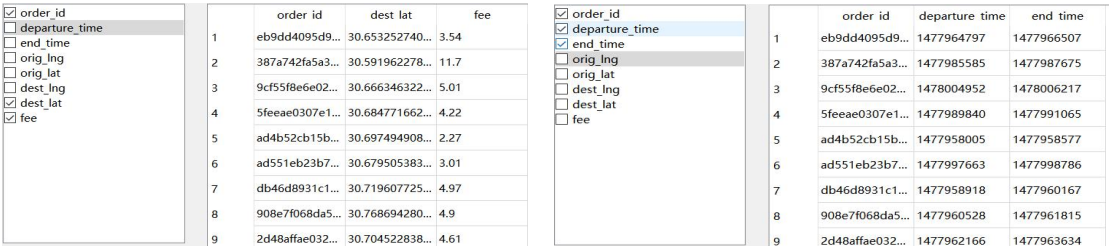
3. Table Displaying (mainwindow.cpp Line 180-229)

After you choose the range of data, a **Table** and a **ChooseBox** will be displayed.Even the progress above haven't stopped, you are still allowed to operate the table, such as rolling the mouse wheel to look around because this will not disturb the work of thread.

The **ChooseBox** contains the name of each column, which is initialized by your data. You

can click those options to change the display in the **Table**, the corresponding column will disappear or appear in the **Table**.

Realization of this function is through **QListWidget**, **QCheckBox** and **QTableWidget**. The **ChooseBox** gains titles of each column from the first row of your data. And as the state of one option is changed, a signal will emit and the **Table** will show or hide the information of corresponding column.



4. Graph Drawing (mainwindow.cpp Line 255-563)

In **Analysis,** you can choose **Count** or **Income** to show graph of statistics, but you need to wait till the progress is finished. The default graph style is line chart, and you can choose different styles in **Choose**, including bar chart and pie chart. Style information of the chart is recorded by an integer in main thread and used when drawing a graph. The original data comes from the thread, which emit answer that is being recorded by **MainWindow**.

And as required, above the graph widget will be a label, showing the total number of the order or the total revenue of the company during this period of time.

You can also see two sliders and two spinboxes, and you can choose the range of grids you are interested in, press **"**Update**"** below and you can see the change. When the button is clicked, the graph will be drawn again, only data inside this range will be displayed. If the range is too small and the style is line chart, additional points will be inserted to make the chart better. The inserted points depends on left and right value of this range, if left and right value are (k,a) and (k+1,b), the inserted point would be (k+0.5,(max(a,b)+2*min(a,b))/3).

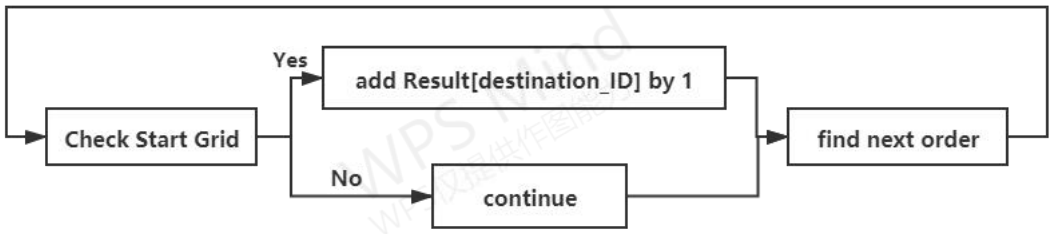
Here is an example when the grid in the chosen range is little, which only contains 5 points, after inserting points, it looks more smooth.

5. Target Analysis (targetthread.cpp & mainwindow.cpp Line 610-652)

The thread won't start immediately because the **Starting Grid** and **Ending Grid** haven't been input. It's similar to the range box above, but there will be only one **Slider** and one **SpinBox** that are connected, either way can change the number. After choosing the preferred range, push the button below and the thread will be ready to get started.

Initialization of this thread includes passing **data**, **map**, **grid_list** it have previously calculated, and grid information you input. The result emit by thread will be received and draw a pie chart by **MainWindow**.

Below is a flow chart of target_thread.



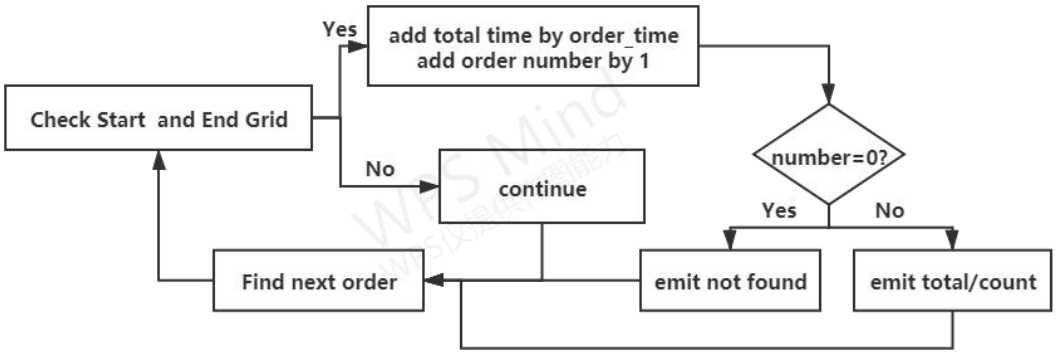
Work of Target Thread

6. Time Analysis (TimeThread.cpp & MainWindow.cpp Line 654-720)

Similar to **Target** action, there will be a **Box** containing two **Sliders** and **SpinBoxes**, representing **Starting Grid** ID and **Ending Grid** ID.

Initialization of this thread includes passing **data**, **map**, **grid_list** it have previously calculated, and grid information you input. The result is an integer, which is average time it takes from one grid to another. And it will be shown in a label on the left.

If there is no relative data being found, the label will show "No Samples Found".



Work of Time Thread

7. Similar Analysis(SimilarThread.cpp & MainWindow.cpp Line 565-608)

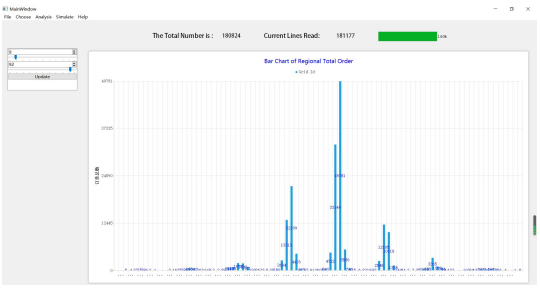
This thread doesn't need you to input anything, but you should switch back to table (**Chart** in **Analysis**) and press the row of data you are interested in before click **Similar.** And you will see a list of similar orders in the **Table**.

The way it judge whether two orders are similar is by a function, it will return a value by weighing differences in latitude, longitude and time spent. Due to data characteristic, the first two enjoy a higher priority and the power will be bigger. When the value returned by this function is lower than a certain threshold value, the thread will reckon the two orders are similar, and add the order into result list.

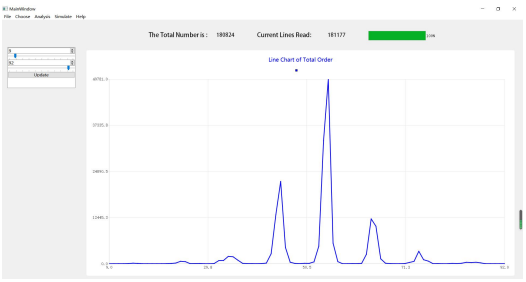
Result:

Due to the limited space here, it's impossible to show all the results, but all the relative screenshot have been submitted along with this document in the project.

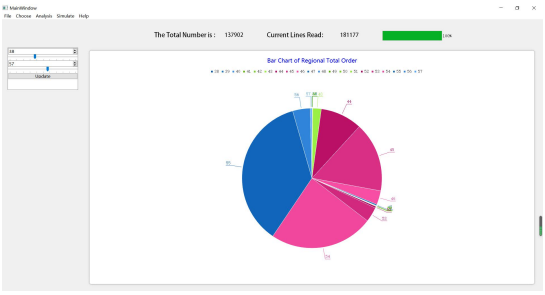
For a more detailed result, please refer to the video demo or see the screenshot in the project.



Bar Chart



Line Chart



Pie Chart

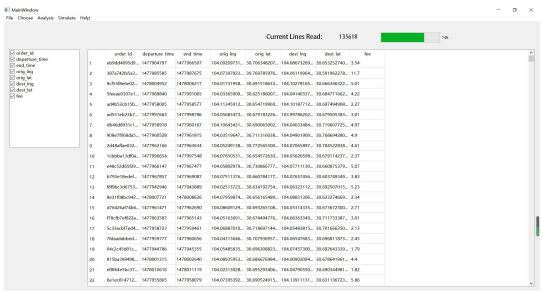


Table View

Discussions:

1. Performance of My Application

Originally, there are 5 .csv file for each day, which is kind of troublesome in loading. So I have merged them into 15 files, one for each day. As the dataset is quite huge, loading takes a while. At first, I try to load data without using multi-thread. And as a result, the program jammed severely. And after using another thread to help, the whole progress becomes much more smooth, so I wonder whether the main thread is considerably slower than the sub threads, and really understand why we are recommended not to do anything that is very time-consuming in main thread.

When using my application, especially when you finish choosing the date range, you may find the progress speed incredibly slow. But it's not a problem, it's slow just because it is processing loads of data and doing pre-processing. If I remove this part and let the program process data whenever it meets the need, it will still cost a lot of time every time. Of course that it's not worth it. So wait patiently till it's done, and enjoy the smooth experience later.

2. Interesting Results Revealed From the Data

After analyzing the possible targets of each grid, I found something very interesting, in nearly every grid, wherever it is, almost all the most possible target is grid 55. Though I'm not from Chengdu and know little about it, I'm still curious why residents considerably far from downtown order ride-hailing mainly to go to center of Chengdu. I guess maybe back in 2016 when it's not that popular, they prefer to call taxi by traditional ways(waving hands and etc.) But when they want to go to downtown, it's hard to get a taxi that is willing to carry you so far away, they turn to ride-hailing for help.

After analyzing incomes from each grid and total order numbers in each grid, here are some points. Apart from the facts above that most orders comes from or comes around grid 55, what amaze me is the low price of each order. Take grid 55 for example, average order fee is only about 3.6 RMB! As far as I know, calling a taxi now takes about 10 RMB at least. After searching, I find out 2016 was the time Didi and Uber were in a price war. Then no wonder.