CS241 Final Project Report

Pieyu Chen 518021910489

I. Introduction

Overcrowding in metro systems is a common problem in many cities,by analyzing and visulizing related data can help us understand this issue.

Main questions of analyzing the traffic data obtained from Hangzhou metro system and visualizing it are shown below:

- How to load and store the data-set;
- How to organize those data to implement data analysis and visualize data;
- How to give feasible path suggestions to users.

For the first question,as database is an organized and unified data collection,it's easy to add,delete and query the data,and after completely load for the first time,there's no need to load again,so it's suitable for loading and storing data.In this project I create a database and table to load and store the large amount of data.

For the second question,as all needed data are stored in the database in an orderly manner,they're easy to do statistics.By setting data range and use SQL statement to count and get the required data,then I can visualize the data.

For the third question,all possible paths can be obtained by breadth first search or depth first search algorithm.In this project,I choose breadth first search algorithm.Though metro map are stored in a separate file,I can still combine it with the Hangzhou AFC data-set to broaden user options.

Details will be shown in part **Implementation Details**.

Analysis will be shown in part **Results**.

II. Implementation Details

A. Overview of GUI

1) GUI of 3 tasks:

The data-set is clearly classified in 7 fields,I found that fields of user-ID and device-ID are not used in subsequent operations and it's useless for users.In order to speed up data loading and save storage space,I dropped these 2 fields.Yet I found time,station-ID and status fields are necessary in task 3.1.2),so I set the checkboxes of the 3 fields checked and change the checkboxes uncheckable to force user to load those data.

To plot the flow trend,I use 2 QDateTimeEdit and 2 QSpinBox to get the restrictive condition needed when drawing the trend. Also,3 checkboxes of options *Inflow Trend*,*Outflow Trend* and *Total Trend* are used to make sure users can selectively draw the trend.

In *Routes Recommendation* part,I use 2 QSpinBox to get the station numbers and provide users 4 options include *Shortest Route*,*Least Crowded Route*,*Recommended Route*,*All Routes*.In order to ensure users only select one option every time,options are all QRadioButton.And the routes are shown in the below read-only QPlainTextEdit.

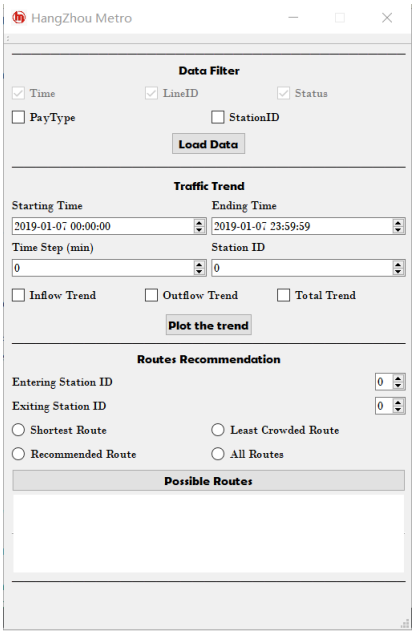

Fig. 1. Graphic User Interface

2) GUI limitations:

To ensure the data that user input are in data range,I simply set limitations of those Input Widgets:

In *Traffic Trend*:

- DateTime is limited from 2019-01-07 00:00:00 to 2019-01-13 23:59:59;
- Station ID are limited from 0 to 80;
- Time Step is limited to more than 0.

In *Routes Recommendation*:

- Station ID are limited from 0 to 80.

3) GUI default options:

To ensure users at least select an option in *Traffic Trend* and *Routes Recommendation*,I simply set default option in GUI:

In *Traffic Trend*:

- Default option is *Total Trend*.

In *Routes Recommendation*:

- Default option is *Recommended Route*.

B. Implementation Details of each function:

1) Data Load Filter:

Data loading actually starts when the application is opened,for during implement progress I found that create a database and load all data in the table can greatly improve speed in following operation.Thus,the *Data Filter* just serves as a explicit progress prompt function.In implement progress,I found that *MainWindow* occupied a thread,if another thread is not created and started to load the data,the whole application will crash.So I created MyThread class inherited from QThread class and override the *run()* function to load data.Another advantage from doing this is users can still use the *Routes Recommendation*function while loading data. The *run()* function consists of 3 parts. First, get the names of all data csv files through the absolute path of the folder where the data csv files are stored. Then create a database with a table named *record*.The *record* table consists of 5 columns corresponding with the 5 fields of the data-set.

TABLE I
EXAMPLE OF TABLE **RECORD**

time	lineID	stationID	status	payType
2019-01-07 06:41:33	C	63	1	1
...	...	...	...	...

Last, the data in each file is inserted into the corresponding position in the table through SQL statement.

One key point is,considering the following tasks, I need to filter data fields time,status and stationID and I found that it's very slow to filter the three data fields separately.Also I found that there's no need to add data after all csv file are inserted into *record* table.Under this circumstance,create index on *record*(time,stationID,status)can greatly improve the speed of data retrieval.

Another key point is that I need to pass the load progress to *MainWindow* and present it to user.To implement this function,an int array *file_flags[211]* is created with all elements' initial value are 0.If the i_th file is inserted,set the value of *file_flags[i]* equal 1 and the 211^{th} element represents whether the index is created.As the csv files are inserted in table one by one,the sum of each element's value in *file_flags[211]* can be used to present the loaded data percentage.

Define loaded data percentage $= P_l$,value of *file_flags[i]* $= v_i$,P_l is calculated by equation:

$$P_l = \frac{\sum_{i=0}^{210} v_i}{210} \times 100\% \tag{1}$$

As the database is stored on disk,there's no need to reload the data-set again if all data are in table.So a token QProgressDialog will be shown if the data are complete.Every time the application is opened,a check function runs in the background to verify the integrity of the database.If the data is complete,the application won't load the data again.

2) Traffic Trend:

To ensure the result is accurate and the program will not crash abnormally,the *Traffic Trend* function cannot be used until all data is loaded.So if data are not complete,a QMessage Box will be shown to inform users.Slot function is used to make connection when user click the *Plot the trend* button.There are 7 Widgets in GUI to accept input.The number of points in trend picture is determined by the starting time,ending time and time step user input.Convert starting and ending times to time stamps and time step to seconds.A QMessageBox will be shown if the ending time is earlier than stating time to force users input again.

Define number of points $= N$,the time stamps of starting time $= T_s$ and ending times$= T_e$,time step in seconds $= t_s$,N is calculated by equation:

$$N = \lceil \frac{T_e - T_s}{t_s} \rceil \tag{2}$$

After the number of points is calculated,count the number of people in each time step by *for loop*.SQL statement *select* is used to count the number of people in required data range.For example,to count the number of people in station-ID n,time between t_1 and t_2 with status i ,the SQL statement is "select count(*) from record table where stationID $= n$ and time between t_1 and t_2 and status $= i$".Then I can draw the flow trend picture with the number of people per time step.The outflow trend is the data with status 0,the inflow trend is the data with status 1 and total trend is the sum of inflow and outflow.

Define the number of people of the n^{th} time step $= P_n$ and time of the n^{th} time step $= T_n$,the n^{th} point is(P_n,T_n).Set the X axis as time and in format(MM-dd hh:mm),Y axis as number of people.The flow trend picture is drawn with the help of QLineSeries,QChart and QChartView classes.In the implement progress I found that QChart class won't automatically set the Y axis which will lead to error of only a part of picture is shown.By setting the Y axis according to the maximum number of people manually,pictures can be shown completely. When there isn't enough points,interpolation is used to smooth the line.I set the critical value of points number$= 10$.If the points number is less than 10,QSplineSeries class which can smooth the line by interpolation is used instead of QLineSeries class.

3) Routes Recommendation:

Using slot functions to make connections first.When the *Possible Routes* is clicked,get the user input of entering station-ID and exiting station-ID and import the metro-map csv file,then call function *possible_routes(...)*.Breadth first search algorithm is used to get all routes in this task.After getting all the routes and store them in a two-dimensional vector,the recommended path is provided according to the user's choice.

For the shortest route,get the route with the smallest corresponding *QVector::size()* and output the route.

For the least crowded route,measurement index of congestion degree are counted in advance and stored in array *congestion_degree[81]*,every element in it is the passenger number of corresponding station which reflects the congestion of a station to some extent.Calculate the score of each route,the least crowded route is the one with the smallest score.

Define score of *route[i]=L_i,congestion degree of station[j]=C_j*.L_i is calculated by equation:

$$L_i = \sum_{j=1}^{n} C_j \tag{3}$$

By counting the total congestion score of each route,the least crowded route with the smallest score is found and output it as the least crowded route.

For the recommended route,it's a combination of the shortest route,the least crowded route and the most continuous route,under the assumption that the smaller the difference value between 2 station-ID,the closer they are or the higher continuity the route has.Normalize the congestion degree and calculate the score of each route,the recommended route is the one with the smallest score.

Define Score of *route[i]=L_i,congestion degree of station[j]=C_j,ID of station[j]* in *route[i]=L_{ij}*,Route length l;L_i is calculated by equation:

$$L_i = \sum_{j=1}^{n} \left(\frac{C_j}{100000}\right) + \sum_{j=1}^{n-1} \left|L_{ij} - L_{i(j+1)}\right| + l \tag{4}$$

By count the total score of each route,the recommended route with the smallest score is found and output it as the recommended route.

For all routes,simply traverse the two-dimensional vector and output every route.

III. Result

A. Data Filter

Although the use of database makes the file load a little slower, the data structure is clearer.The use of array *file_flags[211]* realizes the real-time monitoring of progress.

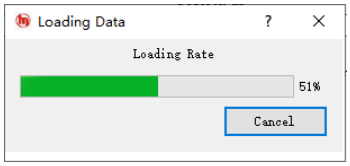

Fig. 2. Progress hints

B. Traffic Trend

A example figure of traffic inflow,outflow and total flow is shown below: From this figure and the date,it's clear that:

- The morning peak is around 8 o'clock and the evening peak is around 18 o'clock every working day;
- This station is in the residential area for inflow peak value is in the morning peak and outflow peak value is in the evening;
- Peak value in weekends is less than weekdays,maybe it's because people like to rest at home on weekends.

Fig. 3. Traffic Trend of Station 0

C. Routes Recommendation

A result of four options of routes recommendation from station25 to station80 are shown below:

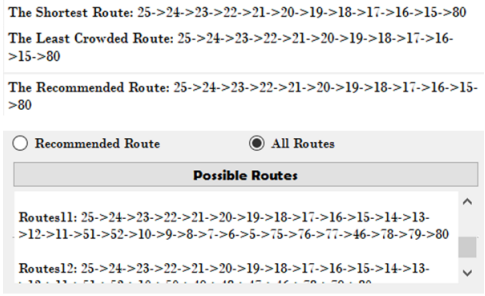

Fig. 4. Routes Recommendation

D. Other Analysis

1) Routes Recommendation:

To give feasible path suggestions to users,I counted the traffic of each station and consider it as a measure of station congestion.And I made assumptions about the less the difference value between 2 station-ID,the closer they are.Thus,the recommended route can be given by route continuity, length and congestion.Refer to equation (4) for calculation method.Figures of congestion degree and a recommended route from station 0 to station 80 are shown below:

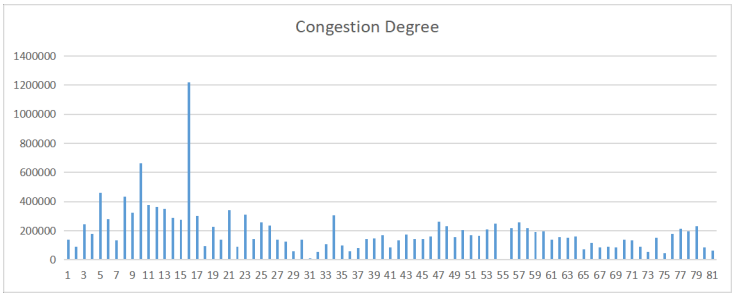

Fig. 5. Congestion Degree

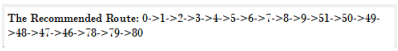

Fig. 6. Recommended Route

2) PayType Preference:

By count all the PayType data,the frequency of the four PayType are shown below.

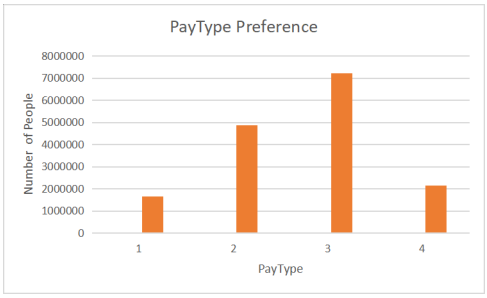

Fig. 7. PayType Preference

Through this result,I find that people prefer PayType 3.Thus,metro managers can increase the number of devices supporting the corresponding PayType.

Though Device-ID and User-ID are kind of useless for basic application,they can be used to implement the following two functions.

3) Equipment Statistics:

The device number in one station can be count through SQL statement "select count(distinct deviceID) from record where stationID = i"

And the flow through a specific device can be count through SQL statement"select count(*) from record where stationID=i and deviceID =j"

By Doing this,I found the device number of station0 is 18 and deveice load is shown below:

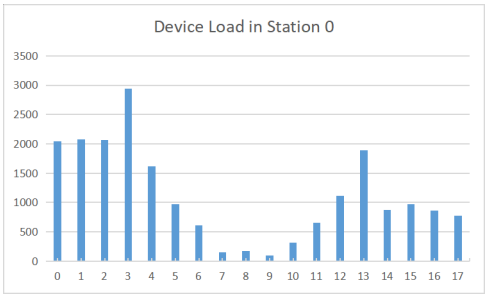

Fig. 8. Device Load in Station 0

From the result,I find that some devices is under heavy-duty service while others are not often used,this may lead to reducing service life of some devices.Metro stations can guide people flow to average device load to avoid this and reduce congestion during peak hours.

Also,with this,Metro managers can learn the load of each equipment and decide whether to add or reduce device to balance cost and effectiveness.

4) Time estimation:

By calculate the closest time interval with 2 different status of a specific user-ID and get his entering station ID and exiting station ID,we can get the estimated travel time between the 2 station.With the whole data-set,we can count the estimated time travel between 2 stations.However,I failed to implement this function due to my limited ability.

IV. Discussions

A. Performance of my application in my PC

1) Data Filter:

TABLE II
DATA LOADING

Average time per file	1076ms
Minimum time per file	976ms
Maximum time per file	1298ms
Index create time	19.8s
Total time	245.76s
Total time	245.76s
Database size(with index)	1029568kb
Database size(without index)	537172kb

- Data load is slower than memory database or direct load in memory,but the data structure is better and does not need to be loaded repeatedly;
- Index greatly improves the speed of data search;
- Poor portability, users need to set the absolute path of data storage.

2) Traffic Trend:

TABLE III
TREND PLOT(TAKE STATION 0 AS AN EXAMPLE)

Time Step	1	5	10	30	60
Plot time per day	0.6985s	0.5214s	0.4914s	0.4728s	0.4728s

- When time step gets smaller, image aliasing becomes more obvious;

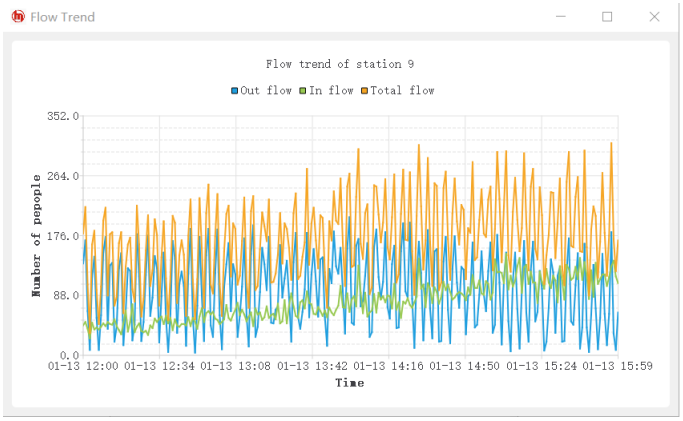

Fig. 9. Image Aliasing

- The flow trend is not obvious in the image of interpolation fitting.

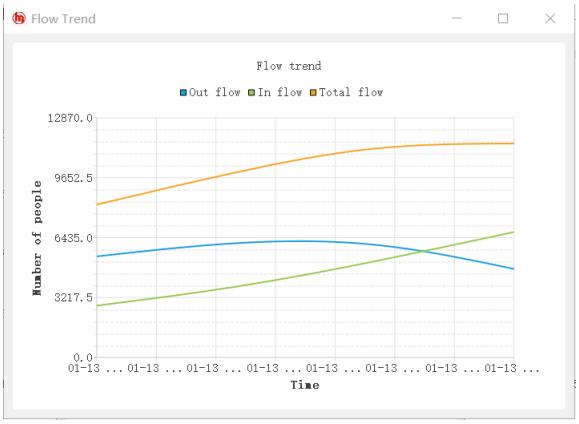

Fig. 10. Image of Interpolation Fitting

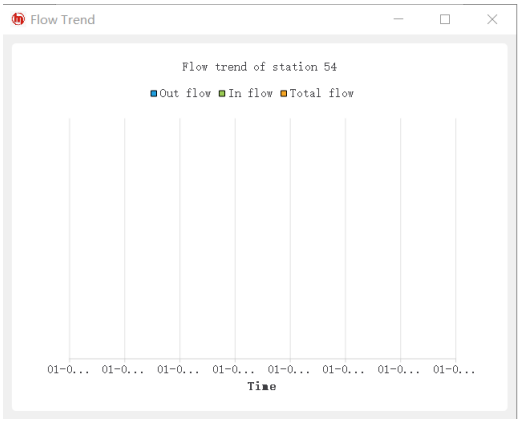

Fig. 12. Traffic Trend of Station 54

3) Routes Recommendation:

- Real-time output;
- The normalization of recommended path score calculation is not rigorous enough;
- The output is not beautiful enough.

4) GUI Limitation:

- Input restriction effectively prevents possible logic errors caused by input;
- Input widgets such as QSpinBox make user input more convenient.

B. Interest Result Revealed

The area of a station can be seen from the flow trend at different stations.For example,different from station 0, the peak flow of station 9 in a week occurs on weekends,it can be inferred that station 9 may not be in residential area but in tourist area or commercial area.

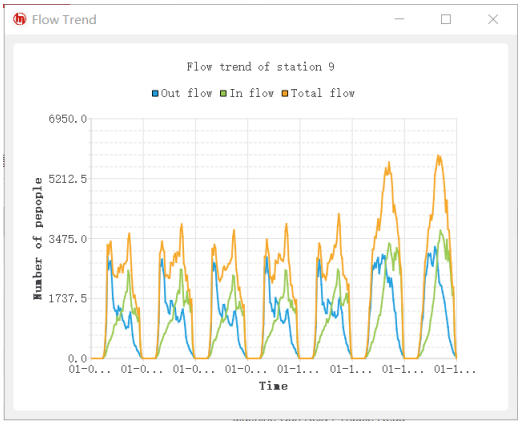

Fig. 11. Traffic Trend of Station 9

Station 54 has no data at all,maybe it's an under-construction station.

By drawing a heat map for Metro roadMap csv file,we can see the continuity between stations and make assumptions:The points in orange color that appears in the upper right part or the lower left part may be the transfer station and the less the difference value between 2 station-ID,the closer they are.

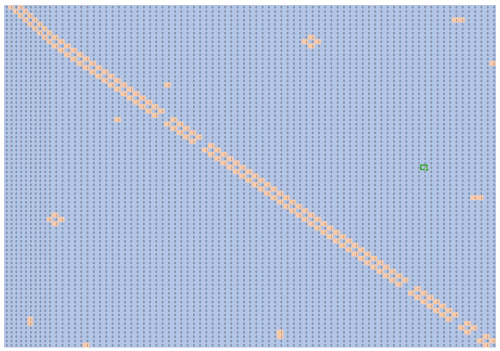

Fig. 13. Heatmap of Metro roadMap

V. Acknowledgement

I would like to express my gratitude to all those who helped me in this courese.

Great thanks to Dr.Ling and Dr.Jin for I have learned various methods of analyzing and solving problems from them and have a deeper understanding of computer science.

Great thanks to TAs(Miss.Zhu Wei and Mr.Sun Jiahui) for their help in learning this semester.

Great thanks to my classmates Chi Zhang and Haotian Xue for their help and company during the development process.