CS2309 Final Project Report

Ziqian Zhao 520021910171

January 2, 2022

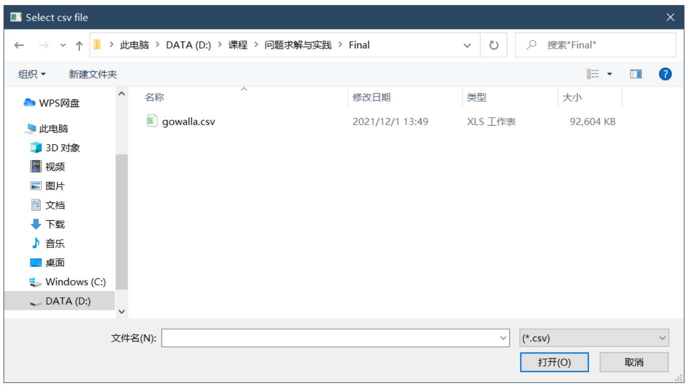
I. INTRODUCTION

Point-of-interest(POI) is an important concept in Geographic Information System(GIS). It has been widely used in lots of location-based applications. For these applications, the valuable information mined from POI data can help them processing various tasks, such as navigation, prediction, recommendation system, etc. In this course project, we are provided with the Gowalla dataset, which contains a total of 1,502,536 checking-in records of users who have shared their locations by checking-in on the Gowalla website over the period of Feb. 2009 - Oct. 2010. We are requested to write a program to analyze and visualize the dataset. With the 100 megabyte dataset, we have to consider how to load, store and organize such large scale data effectively and efficiently. Also, we should pay attention to making our GUI as concise and intuitive as possible for the users of the software who are not professional programmers. Finally, the form of how the data is presented largely determines the user experience. Therefore, rich charts and intuitive demonstration are requested. The main idea of the program is to allow users to mine valuable information underneath the data and gain intuitive impression from the raw data.

II. IMPLEMENTATION DETAILS

1. Loading

When first running the program, users will be asked to specify the *.csv* file which contains the data.



名称	修改日期	类型	大小
gowalla.csv	2021/12/1 13:49	XLS 工作表	92,604 KB

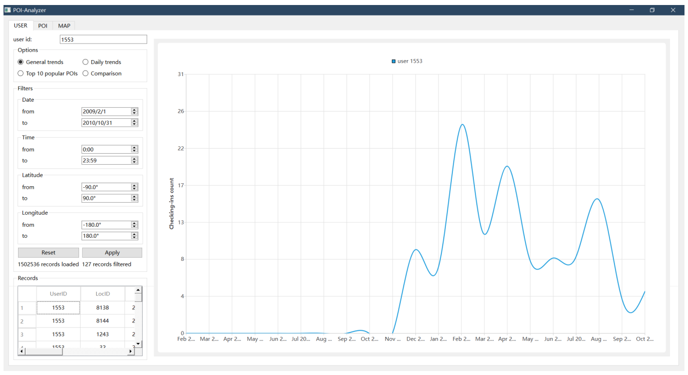
Figure 1: File Selecting Window

After the file path has been specified, the program will start load the data. A progress bar is provided to show the loading progress. The value of the progress bar is calculated according to the total bytes and bytes which have already been read. Loading progress and GUI are working in different threads, so users can cancel the loading at any time.

Considering that the data size is about 90MB, instead of using a local database, it is more reasonable to load all the data into memory directly. In this way, we can gain higher access performance. The program will parse the strings in *.csv* file into the *POI-Record* custom objects, which store six major types of metadata: userID, locationID, date, time, latitude and longitude. First, we will store all the *POI-Record* in a array. Considering that our analyses are mainly based on a specific user(or a set of users) or a specific POI(or a set of POIs), we will maintain two hashmaps. The key of one is userID and its value is a array which contains all the *POI-Records* related to the user specified by the userID. And the key of the other is locationID and its value is a array which contains all the *POI-Records* related to the POI specified by the locationID. In our implementation, we simply use two *QVectors* to replace the hashmaps. Because userID and locationID are both increasing from 0. Organizing data in this way, we can quickly find out all the records related to a specific user or a specific POI(or a set of them). After successfully loading all data, we will walk into the GUI part.

2. GUI Structure

GUI contains three tabs. The first tab is *USER*, the second is *POI*, and the last is *MAP*, where we can do analysis based on user, POI and map.



user id: 1553

Options

[✓ General trends] Daily trends

Top 10 popular POIs Comparison

Filters

Date from 2009/2/1 to 2010/10/31

Time from 0:00 to 23:59

Latitude from -90.0° to 90.0°

Longitude from -180.0° to 180.0°

1502536 records loaded 127 records filtered

Records

	UserID	LocID	
1	1553	8138	2
2	1553	8144	2
3	1553	1243	2

Figure 2: GUI Window

A. User Tab

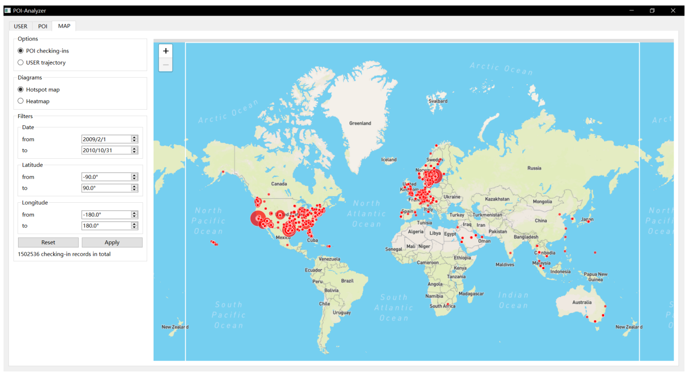
In the user tab, the toolbar is on the left side and the chart display area is on the right side. On the top of the toolbar is a *QLineEdit* widget which allows users to specify the userIDs they want to analyze. Below the input widget there is a *QGroupBox* where users can select the chart type they are interested in. After users select the chart type, the chart will display on the right of the GUI. Next there is a filter, which contains many *QSpinBox* to allows users to tune the parameters including starting time, ending time and GPS range. The GPS precision is one decimal place. Users can click the *Apply* button to apply the filter parameters they have chosen. And users can click the *Reset* button to reset parameters to the default values. (Default values are as follows: Date from 2009-2-1 to 2010-10-31, Time from 0:00 to 23:59, Latitude from $-90.0°$ to $90.0°$, Longitude from $-180.0°$ to $180.0°$) There are also two *QLabel*s to show how many records are loaded and how many records are filtered. Finally, we have a form to demonstrate all the filtered records. With the form, users can easily figure out the detailed information of filtered records.

B. POI Tab

The POI tab and the User tab are almost identical. Compared to User tab, the only difference of POI tab is that options do not contain *top 10 popular POIs* but contain *top 10 active users*.

C. Map Tab

In the map tab, the toolbar is also on the left side, and the map view is on the right side. In toolbar, users can select whether to show POI checking-ins or user's trajectory. If users choose to show user's trajectory, they will be asked to specify the userID which they are interested in.



Options

[✓ POI checking-ins]

USER trajectory

Diagrams

[✓ Hotspot map]

Heatmap

Filters

Date from 2009/2/1 to 2010/10/31

Latitude from -90.0° to 90.0°

Longitude from -180.0° to 180.0°

1502536 checking-in records in total

Figure 3: Map Tab Window

For POI checking-ins, users can choose hot spot map or heat map. For user's trajectory, in addition to hot spot map and heat map, animation display of trajectory is also provided.

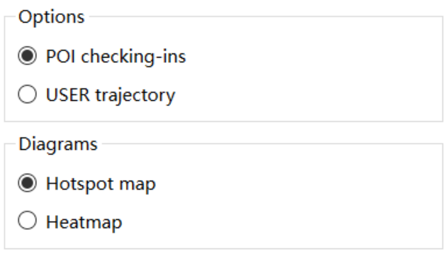

Options
[✓ POI checking-ins]
USER trajectory
Diagrams
[✓ Hotspot map]
Heatmap

Figure 4: POI Checking-ins

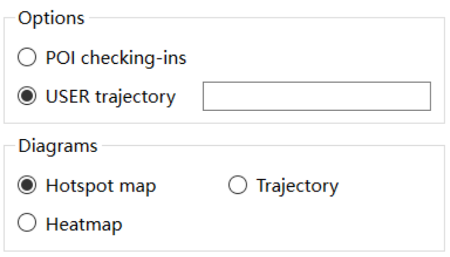

Options
POI checking-ins
[✓ USER trajectory]
Diagrams
[✓ Hotspot map]
Trajectory
Heatmap

Figure 5: User Trajectory

In map tab, a filter is also provided. Users can tune parameters including time range and GPS range. A *QLabel* is provided to show how many checking-in records are filtered with the given time range and GPS range.

In the map view, users can change the zoom level and observe the entire world or small areas freely.

3. Charts

For analysis based on users and POIs, we offer various kind of charts, including line chart, histogram, pie chart, etc. Users can gain valuable information from these charts, such as general checking-in trend, daily checking-in trend and top 10 popular POIs/active users of a specific user or POI(or a group of them, just split each ID with the comma). Also, users can compare the daily active period and popular POIs/active users of two different users/POIs.

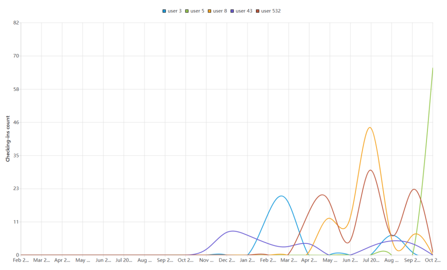


Figure 6: General Trend

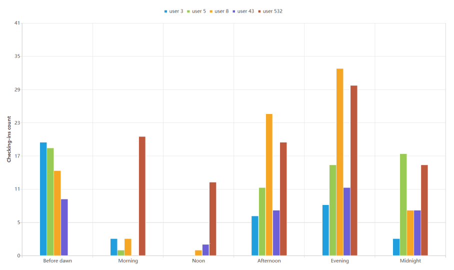


Figure 7: Daily Active Period

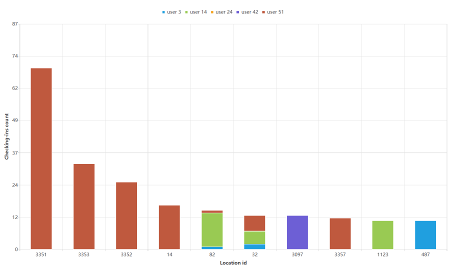


Figure 8: TOP10 Popular POIs

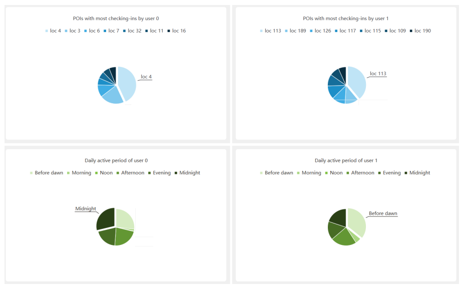


Figure 9: Comparison between two users

Charts implementation details are as follows:

- In general trend chart, we divided the total period by month. We use interpolation and the *QSplineSeries* to smooth the curve. Up to five curves can be displayed at the same time.
- In daily active period chart, we use the *QBarSeries*. A day is divided into six parts(0:00-6:00 before dawn, 6:00-11:00 morning, 11:00-13:00 noon, 13:00-18:00 afternoon, 18:00-21:00 evening, 21:00-0:00 midnight). The time difference is ignored in the division of daily time periods.
- In top 10 popular POIs/active users chart, we use the *QStackedBarSeries*. A priority queue is used to sort and find ten POIs/users with the most records related to the given user/POI(or group).
- In comparison chart, we use the *QPieSeries* to demonstrate the proportion information. Users can compare the proportion of daily activeness and popular POIs/active users of two given users/POIs.

4. Map-related Analysis

In the map tab, we implement the map view by *QWebEngineView*, which can demonstrate an html web page in the Qt program and support c++ and JavaScript to invoke each other. And we use the JavaScript library *Leaflet*[1] to interact with the map, such as drawing some points, lines, heat map, etc. For analysis about POI, we provide hot spot map and heat map. For analysis about user, we additionally provide an animation display of user's trajectory, which is based on time and location. We use a white border to mark the current selected GPS range.

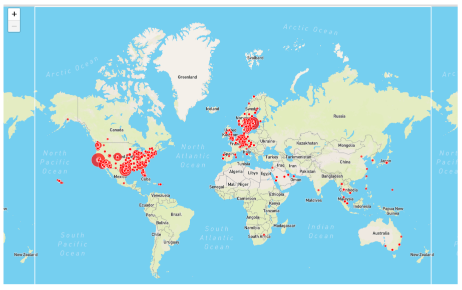


Figure 10: Hot Spot Map

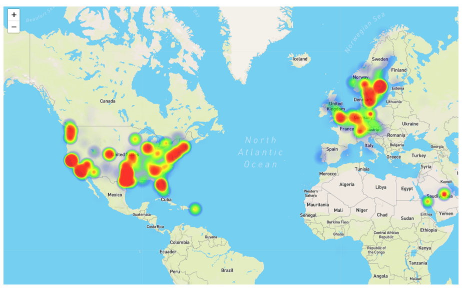


Figure 11: Heat Map

A. Hot Spot Map

In the implementation of Hot spot map, the first step is to calculate the checking-in counts of each POI in the given time range and GPS range. Then we draw the circle of each POI. The radius is calculated by the global max checking-in counts and current checking-in counts. Radius size in pixels.

$$radius_i = 15 \times \frac{count_i}{maxcount} + 1$$

In order to promote performance, POIs with too few records will be ignored(counts less than 1% of the max counts). Besides, we can click the points on the map to see detailed information of it, such as locationID and checking-in counts.

B. Heat Map

We use the JavaScript plugin *Heatmap.js*[2] to draw the heat map. In its implementation, the density information will be converted into gray scale information. And then map the gray scale information to different colors.

C. Trajectory Animation

In this diagram, we use the JavaScript plugin *Leaflet.Polyline.SnakeAnim*[3], which is able to display an animation of points and lines in order. We simply draw a straight line between the two locations visited by the user successively to show the user's trajectory. Lines are drawn in chronological order of user access.

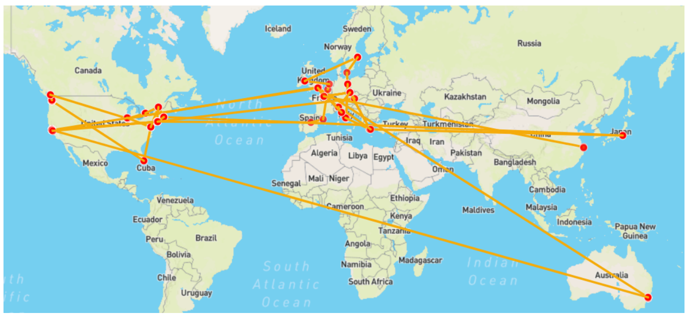


Figure 12: User Trajectory

Because the data size of the map is huge, we choose to load the map online. Therefore, for analysis based on map, networking is required.

[1] Leaflet : an open-source JavaScript library for mobile-friendly interactive maps. https://leafletjs.com/
[2] Heatmap.js : JavaScript Library for HTML5 canvas based heatmaps. https://github.com/pa7/heatmap.js
[3] Leaflet.Polyline.SnakeAnim. https://github.com/IvanSanchez/Leaflet.Polyline.SnakeAnim

III. RESULT

For almost all functions, the program can run successfully and smoothly.

1. The progress bar can work successfully. Users can interrupt loading at any time.
2. Users can easily tune the parameters in a graphical way to filter the records.
3. All the charts can be displayed correctly and quickly.
4. By using *QSplineSeries* and interpolation, the lines can be displayed smoothly when only a few data points are available.
5. For map-related analysis, the heat map, hot spot map and user trajectory can be displayed in a quite intuitive way. And it has user-friendly interactivity.

IV. DISCUSSION

1. Performance

The performance of the program is quite acceptable.

- The time of loading data on my computer is about 15s. Drawing charts is about 20ms. Drawing hot spot map/heat map is about 1s.
- In user and POI tab, memory usage is about 300MB. While calculating, the peak usage is about 350MB. In map tab, memory usage is about 500MB. While calculating and drawing, the peak usage is about 550MB.

For further improvement, we can use the pipeline model to speed up filtering records with multi-threads.

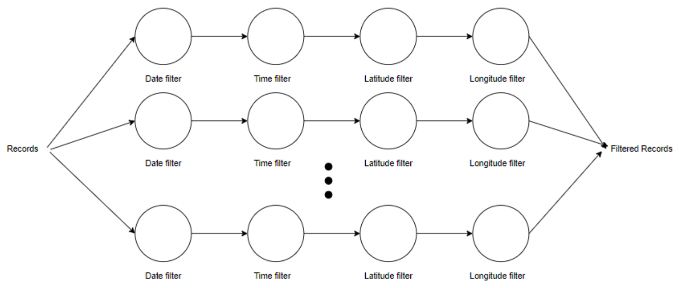


Figure 13: Multi-thread Filter

2. Interesting Results

After locating the most popular POIs on the map, we find an interesting result. It seems that people tend to sharing their locations when they are at the airport.

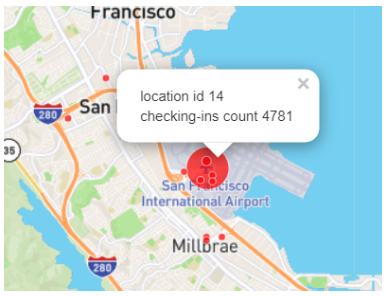


Figure 14: San Francisco International Airport

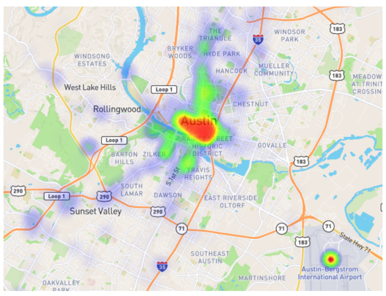


Figure 15: Austin-Bergstrom International Airport

Of the ten most popular POIs, seven are airports and one is a railway station