DR598 Shenzhen Taxicab Traffic Visualization

Xinyu Dai, Jie Lou

In this directed research, we tried to analyze the degree of traffic congestion of the Chinese city Shenzhen. We would use the Taxicab GPS data to count on each taxi's influence on the traffic and use different color to visualize the degree and plot the road system on map.

Data Generation

Here we will discuss how we analyze and process the raw data then finally get a JSON file that can be accepted by d3 to draw a map.

- Get raw data from OpenStreetMap

First we export an xml file of all the road information data of the Chinese city Shenzhen from the OpenStreetMap[1] of the bounding box <bounds minlat="22.45" minlon="113.767" maxlat="22.8667" maxlon="114.617"/>, which is the administrative area of the city.

Inside the xml file, there are three key elements that are important to us: <node>(Figure1), <way>(Figure2) and <relation>(Figure3). In the nodes elements, we select three parts for future use: "id","lat" and "lon". Nodes are the composition parts of ways, where node_id would be the reference in the <nd> elements of way. The latitude and longitude attributes are the coordinates for map plotting.

```
<node id="2614443564" lat="22.4629625" lon="114.0033363" version="1" timestamp="2014-01-
08T07:05:59Z" changeset="19876697" uid="169827" user="hlaw"/>
```

Figure 1: The snapshot of the detailed information of node element

The way elements include information about the nodes which make up the road area. In the tag element, it includes the type and usage of the way. For instance, the figure 2a is a snapshot of a parking area which is kind of an amenity area. While plotting about the road system, we need the highway tag which is used for identifying any kind of road, street or path[2]. Thus we select the highway type of "motorway", "trunk", "primary", "secondary", "tertiary", "unclassified", "residential" and "service" (Figure 2b), which indicates the importance of the highway within the road network as a whole. To be detailed, the motorway is the fast and restricted access road, trunk way is the most important in standard road network, while unclassified is the least important in standard road network. "Residential" and "service" roads are smaller road for access. When drawing the map using the Taxicab GPS data[3] we use all these types for analysis.

[1] Available at: http://www.openstreetmap.org/export#map=13/22.5750/114.1050
[2] Available at: http://wiki.openstreetmap.org/wiki/Key:highway
[3] Thanks to the data provider: http://www-users.cs.umn.edu/~zhang/data.html (for research only)

```
</way>
<way id="232420288" version="1" timestamp="2013-08-05T10:09:07Z" changeset="17225618" uid="44514
" user="MarsmanRom">
  <nd ref="2407726273"/>
  <nd ref="2407726240"/>
  <nd ref="2407726216"/>
  <nd ref="2407726286"/>
  <nd ref="2407726273"/>
  <tag k="amenity" v="parking"/>
</way>
```

Figure 2a: The snapshot of the detailed information of way element (not selected)

```
<way id="232420289" version="1" timestamp="2013-08-05T10:09:07Z" changeset="17225618" uid="44514
" user="MarsmanRom">
  <nd ref="2407726258"/>
  <nd ref="2407726214"/>
  <nd ref="2407726289"/>
  <tag k="highway" v="service"/>
</way>
```

Figure 2b: The snapshot of the detailed information of node element(selected)

Having the separate roads at hand, we need to link the roads with points into major highway or a cycle route or a bus route (which would be used for future analysis on bus station). We refer to the relation element(Figure 3) in the xml dataset for this information which includes tags stating relationship types "route", "turn restriction (meaning you can't turn from one way into another way)" and "multipolygon (an area with holes)"[4]. The member elements are which for the relationship. We pick out the routes which are roads to link all the ways. There are 21268 roads in total in the city of Shenzhen.

```
<member type="way" ref="358454527" role=""/>
<member type="way" ref="358454526" role=""/>
<member type="way" ref="358454567" role=""/>
<member type="way" ref="358454568" role=""/>
<member type="way" ref="358454569" role=""/>
<tag k="name" v="京港澳高速"/>
<tag k="name:en" v="G4 Jinggang'ao Expressway"/>
<tag k="network" v="CN-expressways"/>
<tag k="ref" v="G4"/>
<tag k="route" v="road"/>
<tag k="type" v="route"/>
</relation>
```

Figure 3: The snapshot of the detailed information of relation element

In order to find the number of taxis occupied and their influence in each part of the street, we split the above major roads into one-mile-long paths and defined an expansion area to gather the count number for heat map color reference. We use the following distance function to calculate the distance of two points with longitude and latitude. Then we split the linked roads to find the one-mile-long stoppoints from the start road point of each road.

[4] Detailed information at: http://wiki.openstreetmap.org/wiki/Elements#Relation

```
def dist(pointa,pointb):
    radlat1=math.radians(pointa.lat)
    radlat2=math.radians(pointb.lat)
    a=radlat1-radlat2
    b=math.radians(pointa.lon)-math.radians(pointb.lon)
    s=2*math.asin(math.sqrt(math.pow(math.sin(a/2),2)+math.cos(radlat1)*math.cos(radlat2)*math.pow(math.sin(b/2),2)))
    earth_radius=3958.7583
    s=s*earth_radius
```

Figure 4: The snapshot of the distance function to convert the distance of two road nodes from their longitude and latitude

- Data Preprocessing

After that we load in the Taxicab data to join with each expanded area of the one-mile-long path. In this 1.88G dataset collected with attributes taxi id, time, latitude and longitude, we have around 47 million of single taxi transaction. (We tried to use the whole dataset for accurate analysis and ran that on MapReduce jobs. Due to the uniformed partition method, it failed because of load unbalancing. Further partition way of data-driven would be introduced for the whole dataset.)

We now used sample dataset of 200M to analyze. For each one-mile-long path, we expand it bounding area as a square around the path and find spatial join of the two dataset. In order to draw the map and store the points data, we convert the output dataset into a JSON file with the following format using the python function json.dumps()(Figure 5):

```
In [54]: roadsToDraw[0]

Out[54]: {'path': [{'count': 10,
    'points': [{'lat': 22.5844326, 'lon': 114.0988493}
     {'lat': 22.5835283, 'lon': 114.0995459},
     {'lat': 22.5830763, 'lon': 114.0998375},
     {'lat': 22.5828435, 'lon': 114.0999287},
     {'lat': 22.5826912, 'lon': 114.0999643},
     {'lat': 22.5821414, 'lon': 114.1000394},
     {'lat': 22.5818677, 'lon': 114.1001379},
     {'lat': 22.5816708, 'lon': 114.1003023},
     {'lat': 22.581552, 'lon': 114.100474},
     {'lat': 22.5814819, 'lon': 114.1008474},
     {'lat': 22.5814692, 'lon': 114.1009591},
     {'lat': 22.58148, 'lon': 114.1039239},
     {'lat': 22.5815003, 'lon': 114.1048932},
     {'lat': 22.5815197, 'lon': 114.1058157},
     {'lat': 22.5815399, 'lon': 114.1067121},
     {'lat': 22.5815623, 'lon': 114.1077037},
     {'lat': 22.5815839, 'lon': 114.1086634},
     {'lat': 22.5816052, 'lon': 114.1096215}]}],
  'type': 'tertiary'}

In [60]: import json
```

Figure 5: The output JSON file example

Draw the traffic road system on map using d3.js

Here we will go through how to use our data to draw on the web page using d3.js. We will put out solutions that how to project the real latitude and longitude to svg.

- Read data from the JSON file

Previously, we got a JSON file that contains all the data information that we need. So the

basic idea is to parse the data via d3.js and then draw every path on the map. d3.js provides us with a function d3.json() to read data from a JSON file asynchronously. Since the data set is a little bit large, the page will be frozen for a while. In the later chapter, we will show how we address this problem.

So we organize the data like the following JSON object.

```
[{" type": " tertiary",
        "path": [
                {" points": [
                                {"lon": 114.0988493 , "lat": 22.5844326},
                                {"lon": 114.0995459 ,  "lat": 22.5835283} ,
                                {"lon": 114.0998375 ,  "lat": 22.5830763} ,
                                {"lon": 114.0999287 ,  "lat": 22.5828435}
                               ],
                        "count": 10}
]  }, ..... // more path objects
]
```

The ***"type"*** keyword contains the type of the path, it can be motorway, secondary, etc. The ***"points"*** keyword contains an array of longitude and latitude. The ***"count"*** keyword contains the number of taxis on that path.

- Compute the size of the map

d3.js will draw on a svg element. So first of all, we need to compute the size of the map. We need to make a projection of the length of the latitude and longitude to our web page. For example, we set the width of our svg element to be 200px, the height of the element and all the path of the map should be computed accordingly. The height of the svg element will be

Height = (width of the svg element) * (difference of max and min longitude) / (difference of max and min latitude);

For the max and min latitude and longitude, we put the data in a JavaScript object, so we can use it everywhere. In this way, we can scale the map based on our web page correctly.

```
var maxminLatLon = {minLat: 22.433297, maxLat: 22.9765329, minLon: 1 13.6932913,
maxLon: 114.7095769} ;
```

- Use the latitude and longitude data on svg

Also, we could not use the coordinates from the raw data directly. Following pseudo code are the way we compute the coordinates on our web page.

To get the appropriate latitude of the webpage:

```
width / (maxminLatLon. maxLat -  m axminLatLon. minLat ) * ( lat -  maxminLatLon.
```

minLat) ;

To get the appropriate longitude of the webpage:

height / (maxminLatLon. **maxLon** - maxminLatLon. **m inLon**) * (lon - maxminLatLon. **m inLon**) ;

In this way, we are able to transform the raw data to the data that we can draw on our svg element correctly.

- Draw a path

According to d3, if we want to draw a path-like object on the web page, we can use the path tag from d3. The path tag requires to provides a line generator as data to for drawing lines. The line generator will return the coordinate of the start and end points to d3, and then d3 is able to draw a line. We will use the method we mentioned above to compute the start and end nodes of the line.

- Set different colors

Since we have an attribute "count", we can use this attribute to draw the path in different color.

Features

- Deal with page freezing when loading data

We should not allow user to operate on the page when it is loading data. So we make an on loading page when the app is loading data. The following screenshot is what we have implemented. Later we can add more features to this page, such as a progress bar.

After the app finished loading all the data, this page(Figure6) will disappear and users can begin to use other features.

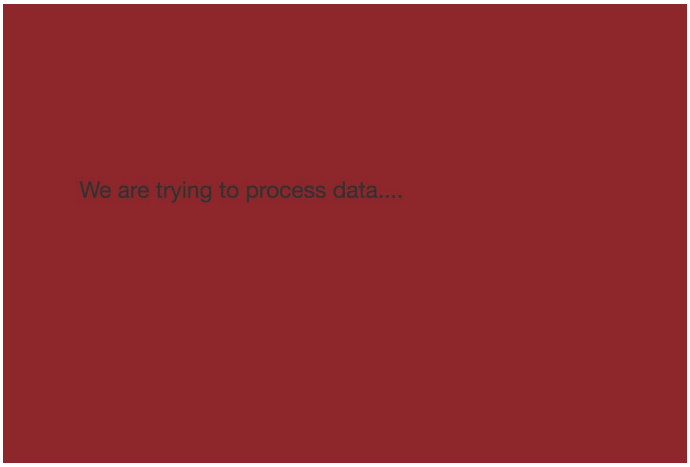

Figure 6: The loading page while plotting the map

- Legend

The code for creating a Legend is located in *public/js/d3.legend.js*. We provided the legend data at the start of the html file. “data-legend” attribute will create a title and “data-legend-color” attribute will create the corresponding color for the title. d3.legend.js will gather these data and draw the legend accordingly.

- Export

We provide an export functionality to save the d3 graph we have generated. We need to transform the svg element to a png or jpeg so that the user is able to save them locally. The basic idea is to transform the svg content into a canvas element and then transform the canvas to a png. After the user click the export button, a png will be inserted into the bottom of the web page.

Node.js Server

We used node.js as the implementation for the server side.

- Express 4.0

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. Express also helps us do the backend router. Usually, the front end will send rest-API calls to the backend to require certain kinds of resources. All the business logics are implemented in the server side via Node.js.

From now we have only one page for drawing map and downloading. So we just need to routes the index.js in the server. Later we will have multiple pages, we can either do a server side rendering or client side rendering, which is a single page application (SPA).

- Serve the static resources

To serve static files such as images, CSS files, and JavaScript files, use the express.static built-in middleware function in Express.

Pass the name of the directory that contains the static assets to the express.static middleware function to start serving the files directly. For example, use the following code to serve images, CSS files, and JavaScript files in a directory named public.

- Using Node Cluster to accelerate

A single instance of Node.js runs in a single thread. To take advantage of multi-core systems the we can launch a cluster of Node.js processes to handle the load. The cluster module allows we to easily create child processes that all share server ports. We have not implemented it yet but definitely we can use this technology to process a JSON file faster.

Future Work

For now, we are using a python program to transform the raw data into a JSON file. We should be able to automate this process in our Node.js server. However, the data may be too large for users to download to local and then upload to our web server, so we may use some web service to request raw data and transform it in the backend, which will provide a better user experience.

The functionality for drawing a map works fine now, but we can add more useful features like filters to get different information of the map. We have a sidebar in our html page, which is a placeholder for the future sidebar.