0 Member Information

Neng Shi (shi.1337@osu.edu)
Jingyi Shen (shen.1250@osu.edu)

1 Introduction

With the rapid development and popularization of mobile Internet, it is possible to collect trajectory data of moving objects by sensors with spatial location recording functions. People have generated massive amounts of microblog sign-in data, taxi GPS data, bus swiping data, and other spatiotemporal data using positioning technologies. These spatiotemporal data have become the research focus of geographic science, computer science, anthropology, urban planning, and other subjects. Movement trajectory data is widely used in the study of urban space. As the main body of urban mobility, urban residents' behavior patterns are of great significance to urban space study.

In our work, we use New York City taxi trips data as our data source. Overall, it contains three kinds of information:

- Spatial attributes (pickup and dropoff locations)
- Temporal attributes (pickup and dropoff time)
- Scalar attributes (e.g., the passenger numbers)

Usually, to analyze this kind of spatial-temporal data, people rely on spatial analysis software based on the spatial database, such as ArcGIS or QGIS. However, to achieve complicated analysis, users have to write SQL queries, which need database knowledge.

We build a visual analysis system that supports efficient visual queries on spatial, temporal, and scalar attributes to resolve the limitations. For spatial queries, users can use the mouse to draw a rectangle on the map, and our system will output the record whose location is inside the rectangle. We use the quadtree algorithm to speed up the spatial queries. For temporal and scalar queries, users can use the input box to type the range they want, and our system can return their desired records. Based on these basic query operations, we can design some analytical questions covering social scientists' and engineers' concerns. For example,

- where → where
- where, then when → what

We will discuss these analytical questions in detail (Sec. 4).

Our system provides multi-facet efficient visualization of query results. For every record, we render its pickup and dropoff locations on a map. We support level-of-detail rendering to resolve the visual clutter problem. We also provide two aggregation views:

- Time series plot: aggregation of spatial and scalar attributes
- Histogram: aggregation of temporal and spatial attributes

We will discuss the visualization views in detail (Sec. 5, 6).

2 Datasets

We use two spatial-temporal origin-destination (OD) datasets. Both are New York Taxi Trips data. One of the data is collected from Kaggle:
Dataset 1 (nyc-taxi-trip-duration, 2016):
https://www.kaggle.com/c/nyc-taxi-trip-duration

Dataset 2 (nyc-taxi-trip, 2013):
https://github.com/VIDA-NYU/TaxiVis/blob/master/data/rawData/sample_merged_1.csv

In both dataset, it contains temporal attributes such as pickup time and dropoff time and spatial attributes such as pickup and dropoff locations. It also has other scalar attributes such as trip duration in dataset 2 and distance in dataset 1.
Here is a screenshot of two datasets.
dataset 1:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	pickup_time	dropoff_time	id_taxi	pickup_long	pickup_lat	dropoff_long	dropoff_lat	distance	fare_amount	surcharge	mta_tax	tip_amount	tolls_amount	payment_type	passengers
2	1/1/13 9:19	1/1/13 9:21	333D72A3E9	-73.954796	40.59893	-73.950531	40.604225	0.5	4	0	0.5	0	0	CSH	1
3	1/1/13 14:02	1/1/13 14:27	8E189DABE2	-73.862709	40.769142	-73.982079	40.762295	10.2	31	0	0.5	0	4.8	CSH	2
4	1/1/13 15:11	1/1/13 15:18	89D227B655	-73.978165	40.757977	-73.989838	40.751171	1	6.5	0	0.5	0	0	CSH	4
5	1/1/13 16:06	1/1/13 16:22	333D72A3E9	-73.989059	40.750572	-73.973686	40.755997	1.8	11	0	0.5	0	0	CSH	2
6	1/1/13 16:46	1/1/13 16:56	E9BA494DC1	-73.959908	40.806396	-73.983292	40.775673	2.8	11	0	0.5	0	0	CSH	1
7	1/1/13 17:11	1/1/13 17:17	5C1E310A26	-73.98793	40.749542	-73.978981	40.766735	1.4	6.5	0	0.5	0	0	CSH	3
8	1/1/13 18:36	1/1/13 18:39	E9BA494DC1	-73.992172	40.749954	-73.99675	40.744553	0.5	4	0	0.5	0	0	CSH	1
9	1/1/13 18:45	1/1/13 18:48	1F03ADDC38	-73.982384	40.752102	-73.993332	40.736393	1.2	5.5	0	0.5	0	0	CSH	1
10	1/1/13 19:01	1/1/13 19:19	39FDC731D7	-73.8638	40.770416	-73.98484	40.732506	9.93	28	0	0.5	0	4.8	CSH	5
11	1/1/13 19:41	1/1/13 19:49	02BFD1B64C	-73.984734	40.76931	-73.997787	40.744205	2.1	8.5	0	0.5	0	0	CSH	2
12	1/1/13 23:26	1/1/13 23:49	0A3EF18FFD	-73.783318	40.648636	-73.964249	40.719543	18.9	50.5	0.5	0.5	0	0	CSH	2
13	1/1/13 23:33	1/1/13 23:36	1ABD0BB8B	-73.983711	40.756092	-73.98494	40.748222	0.8	5	0.5	0.5	0	0	CSH	2
14	1/2/13 6:58	1/2/13 7:24	DD98E2C3AF	-73.987022	40.759373	-73.862022	40.768017	3	2.5	0	0.5	0	4.8	CSH	1

dataset 2:

	A	B	C	D	E	F	G	H	I	J	K
1	id	vendor_id	pickup_time	dropoff_time	passenger_c	pickup_long	pickup_lat	dropoff_long	dropoff_lat	store_and_fv	trip_duration
2	id0190469	2	1/1/16 0:00	1/1/16 0:14	5	-73.981743	40.7191582	-73.938828	40.8291817	N	849
3	id1665586	1	1/1/16 0:00	1/1/16 0:22	1	-73.985085	40.7471657	-73.958038	40.7174912	N	1294
4	id1210365	2	1/1/16 0:01	1/1/16 0:07	5	-73.965279	40.8010407	-73.947479	40.8151703	N	408
5	id3888279	1	1/1/16 0:01	1/1/16 0:05	1	-73.982292	40.7513313	-73.991341	40.7503395	N	280
6	id0924227	1	1/1/16 0:01	1/1/16 0:13	1	-73.970108	40.7598	-73.989357	40.7429886	N	736
7	id1078247	2	1/1/16 0:01	1/1/16 0:03	1	-73.973335	40.7640724	-73.974854	40.761734	N	114
8	id3609443	1	1/1/16 0:01	1/1/16 0:21	2	-73.993103	40.7526321	-73.953903	40.8165398	N	1204
9	id2914314	2	1/1/16 0:02	1/1/16 0:23	1	-73.985443	40.7357101	-73.957489	40.8116112	N	1310
10	id3675972	2	1/1/16 0:02	1/1/16 0:10	1	-73.993736	40.7417603	-74.004669	40.7454681	N	445
11	id0846101	2	1/1/16 0:03	1/1/16 0:07	1	-73.97686	40.6833763	-73.983643	40.6888657	N	263
12	id0372520	1	1/1/16 0:03	1/1/16 0:16	1	-73.969788	40.7575836	-73.951576	40.7594452	N	770
13	id3968582	1	1/1/16 0:04	1/1/16 0:22	2	-73.993973	40.7408676	-73.988556	40.764782	N	1071

3 Data preprocessing

We have the following data pre-processing steps for both datasets:

- select attributes
  - for dataset 1:
    - 'pickup_time', 'dropoff_time', 'id_taxi', 'pickup_long', 'pickup_lat', 'dropoff_long', 'dropoff_lat', 'distance', 'fare_amount', 'surcharge', 'tip_amount', 'tolls_amount','payment_type', 'passengers'
  - for dataset 2:
    - 'id', 'pickup_datetime', 'dropoff_datetime', 'passenger_count', 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude', 'trip_duration', 'vendor_id',
- remove duplicates
- remove empty cell
- remove invalid data
  - zero longitude and zero latitude
- subset of of in a pre-defined NY area
  - bottom_lat = 40.535122
  - top_lat = 40.943897
  - right_long = -73.667663
  - left_long = -74.039955

After preprocessing we have 9709 data samples in dataset 1.
For dataset 2, we further select data samples between Jan 1st and Jan 3rd, we result in 9872 data samples.

4 analytical questions

Inspired by visual task abstraction, we designed our analytical questions based on the three WHs: what, where and when. We propose the following analytical questions.

Start with where:

- Where do the drop offs happen given a pick up area? (where → where); what if we further add a time constraint? (where + when → where)
- Compare the taxi trips in lower Manhattan and middle Manhattan for a given time (where + when → what)

Start with when:

- What time is the taxi busy/idle during a holiday time? (when → when)

Spatial query: the brute region query requires to scan the whole dataset, which is time-consuming. We build a quadtree index to reduce the query time from O(N) to O(logN), which makes our interaction more efficient.

Once we load the dataset, we will construct geometric features with each record's pickup location. We input all the geometric features into the root node. If the number of features exceeds our predefined capacity (200 in our program), we split the node into four sub-nodes, add each feature to the corresponding sub-node, and delete the current node's geometric feature record (i.e., all features only stored in the leaf node). If the number of the child node's geometric features is still greater than the predefined capacity, the sub-node is recursively split. We use a treemap to visualize the quadtree after the build.

During range query, we input the region *rect*, query the geometric features that intersect the region *rect*. If the region *rect* intersects the current node's bounding box, the quadtree is recursively traversed to query which node's bounding box intersects the query region. After obtaining the candidate geometric features that may be inside the region rect, we accurately determine whether the geometric feature is inside the region rect. In the visualization system, we can select the query area with the mouse to do the region query.

5 visualization requirements

R1: Our visual analytical system needs to support efficient spatial, temporal and scalar attribute query of the data. A projection of data into a 2D plane is required. To assist users for detailed spatial analysis, proper interactions are needed, such as hover, zoom in and spatial range selection.

R2: To fulfill the temporal analysis requirement, our system needs to enable temporal range query, such that users can input a temporal range and get a filtered output.

R3: To help users understand the statistical distribution of the selected data, our system needs to have an aggregated visualization. For example, aggregate by time for all selected data or aggregate by scalar attributes.

R4: Due to the limited screen space, we would encounter severe visual clutter problems if we want to show all the pickup and dropoff locations. To resolve that, our visualization system supports adaptive level-of-detail (LOD) mode.

- Our LOD strategy is to reduce the points rendered by subsampling. Since all the point features are stored in the quadtree leaf nodes, we only select *1/SR* points in each leaf node in LOD mode, where *SR* is the subsampling rate. If we zoom in or out, SR will scale proportionally to the map zoom level. In the original view, *SR* = 200.

6 Visualization design

For spatial query, we first build a 2D map to show the spatial distribution of the data (**R1**). Each data sample is colored by its class, blue for pick up locations and red for drop off locations.

At the beginning, there are two modes, users can switch between LOD mode and normal mode (**R4**). In LOD mode, once the data is loaded and the quadtree is built, a treemap will show up on the map. Since we are only visualizing subsamples in the view, it has much less visual clutter. When users zoom in an area on the map, more points will be rendered adaptively in the map view, which gives the user level-of-detail.

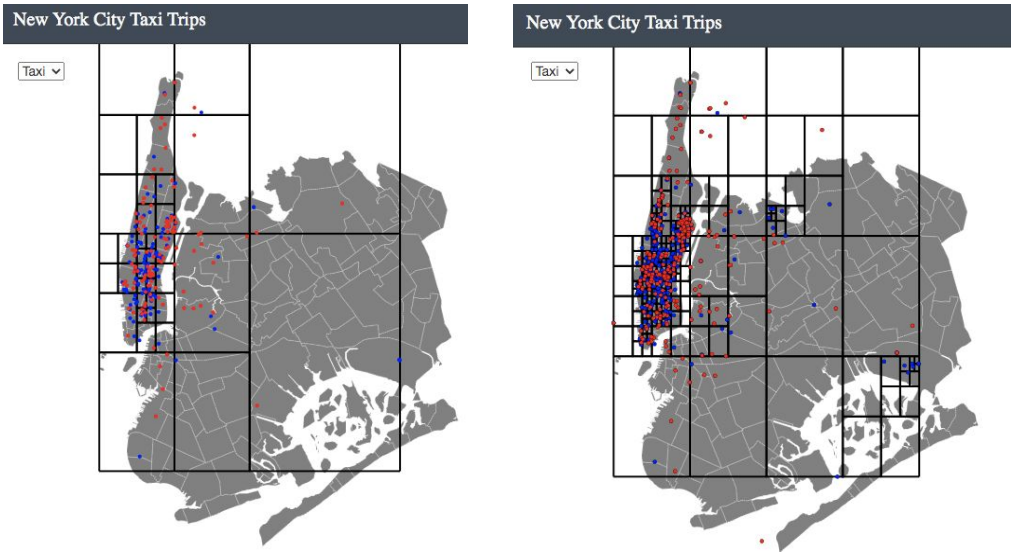


Fig 1: left: in LOD mode; right: in normal mode.

Our system allows spatial range query by drawing a rectangle on the map (**R1**). All the trips with pick up (or drop off) locations in this rectangle will be selected. After users make selections (for example, select by pick up locations) on the map, the selected data will be recolored into green for selected pick up locations and yellow for selected drop off locations. Users can zoom into the map to get a finer selection.

Our system also includes a date pick view which allows users to make temporal range queries on a calendar (**R2**). After temporal range selection, the spatial map will be updated to show only the filtered data.

We build a dropdown list to select attributes. After all the queries are done, users can click the 'Query' button to update histogram and time series plots. These two views will show the aggregated statistics for the selected data (**R3**). In the histogram view, users need to select an attribute. A predefined attribute is *distance* for dataset 1 and *passenger count* for dataset 2. Then the histogram of this attribute is computed. The x axis is the value for each bin of this attribute, the y axis is the frequency of instances in each bin. In

the time series view, we divide time into bins and the instances are aggregated by temporal bins. The x axis of time series plot view represents temporal bin and the y axis shows the frequency in each bin. We use a predefined bin = 24. Users can also slide on the time series plot to zoom in a time range.

7 Software

Frontend: javascript
Backend: flask
Libraries: D3.js, bootstrap, litepicker.js, d3-timeseries.js

8 Results and Evaluation

We use our system to answer the analytical questions we proposed in Sec. 4.

1. Where do the drop offs happen given a pick up area (where → where)?
   We found a small cluster on the lower right corner of the map. By hovering on the map, we found it is the John F. Kennedy International Airport. We select this region as shown in Fig. 2. The yellow points are the corresponding drop off locations. As we expected, drop off locations are far from the pick up locations and they are mostly in Manhattan (especially midtown) and Brooklyn.

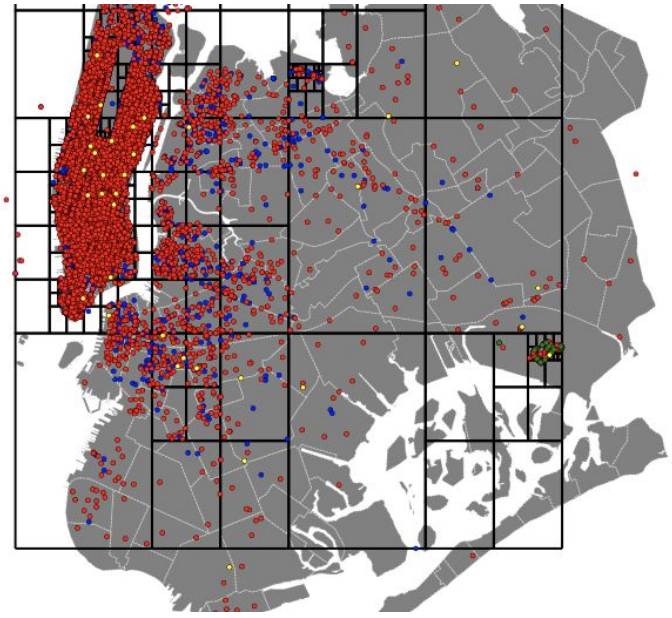


Fig 2: pick up locations (in green) and the corresponding drop off locations (in yellow).

2. What time is the taxi busy/idle during a holiday time? (when → when)
   To answer this question, we first select the whole city on the map. Then we select data points between Jan 1st 2016 and Jan 3rd, 2016, we have the following time series plot:

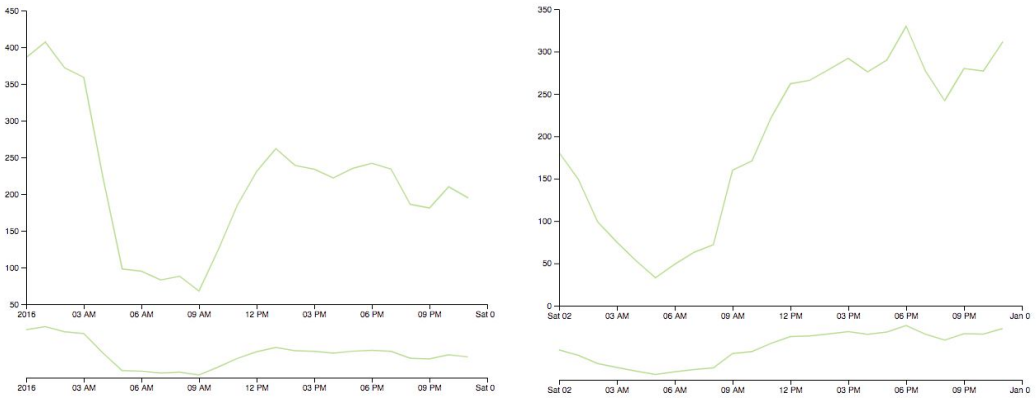


Fig 3: left: time series plot for Jan 1st 2016;
right: time series plot for Jan 2nd 2016

The x axis is time and the y axis is frequency (how many taxi trips happen in this temporal range). We noticed a peak after midnight. A possible reason is people went out to celebrate the new year and after the celebration, there is a clear peak in the number of taxi trips. Another finding is, we did not see a morning peak as usual, which is also what we expected.

3. Compare the taxi trips in lower Manhattan and middle Manhattan for a given time (where + when → what)?
   To answer this question, we select lower Manhattan and middle Manhattan separately and add the same temporal constraint. We have the following passenger count and trip distribution distribution:

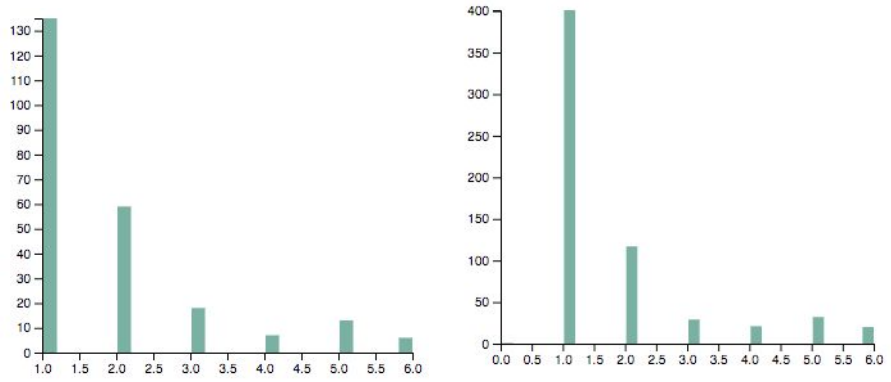


Fig 4: left: histogram plot of passenger count in lower Manhattan;
right: histogram plot of passenger count in middle Manhattan;

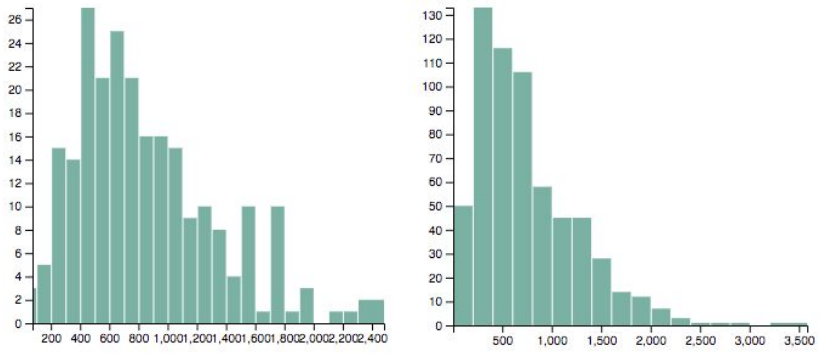


Fig 5: left: histogram plot of trip duration it in lower Manhattan;
right: histogram plot of trip duration in middle Manhattan;

The x axis is the value of each bin and the y axis is frequency (how many taxi trips happen in this attribute bin). From both plots in Fig 4, we can see people tend to take taxis alone. And compared with lower Manhattan, middle Manhattan has much more taxi trips in this time. In Fig 5, we can see the distance distribution of middle Manhattan is wider, meaning the taxi trip trajectories are spread out further than lower Manhattan.

Analyze the quadtree:

The parameter *capacity* determines its performance for the quadtree: the maximum number of each leaf node's geometric features. If the value is too large, the number of judgments in each leaf node is too large. Otherwise, the quadtree level is too high. The parameter *sampling rate* determines how many points we show in our view. If the value is too large, we will encounter visual clutter. Otherwise, the view will become too sparse. To tune these two parameters, when *capacity* changes, we calculate the quadtree's height, the number of internal and leaf nodes and draw the quadtree treemap to visualize the quadtree structure. In the LOD mode, we tune the parameter *sampling rate* for each *capacity* to sample the proper number of points. We show some quadtree statistical information and default LOD views under different *capacities* and *sampling rate* in the appendix. Finally, we use *capacity* = 200 and *sampling rate* = 200.

9 project log

Neng:
12. 2 write the QuadTree class.
12. 8 finish using real data to construct quadtree and do range query;
Add LOD visualization based on quadtree.

Jingyi:
12.1: get Geo data for map;
12.3: select proper dataset for our visualization; pre-process data
12.4: draw Geo map add temporal query
12.5: add select by brush on map
12.9: finish histogram and time series plot (Thanks a lot to Neng, for helping debugging!!)

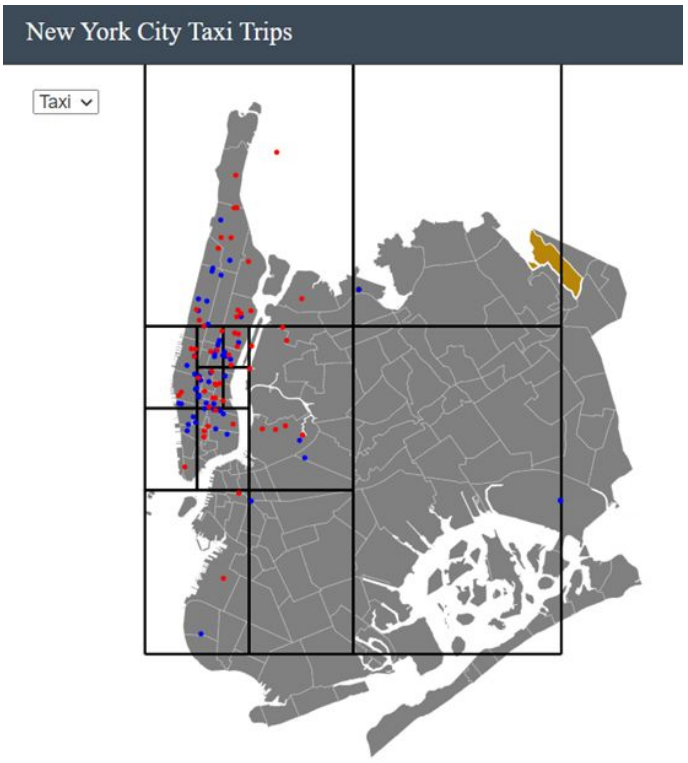
10 Teamwork

team members: Neng Shi, Jingyi Shen
Our work is listed in project log (Sec. 9).

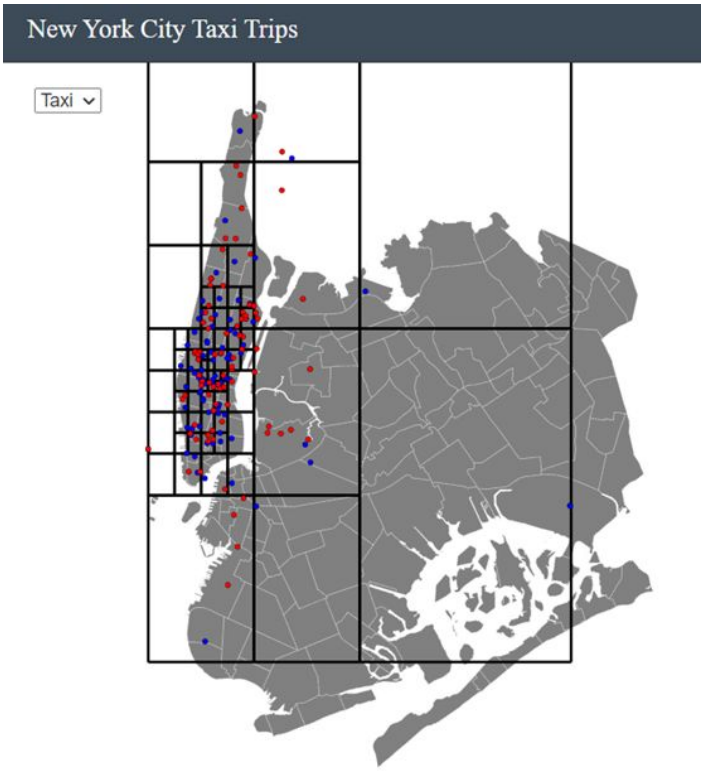
appendix:
A. quadtree analysis

Here, by setting different quadtree node capacity and sampling rate, we output the quadtree's height and the number of leaf nodes. We also show the quadtree structure and default LOD view.

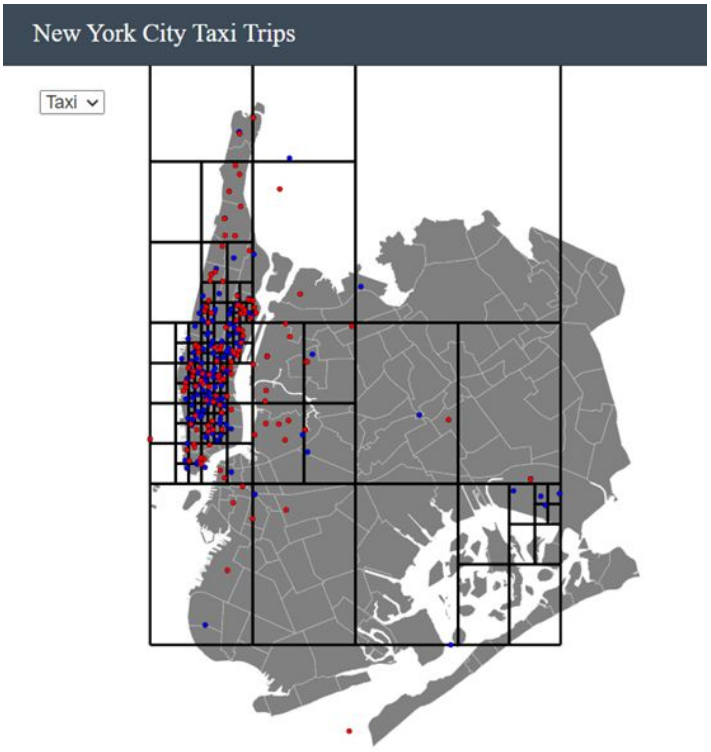
C=2000, SR=200
QuadTree height: 5
QuadTree nodes: {interiorNum: 4, leafNum: 13}



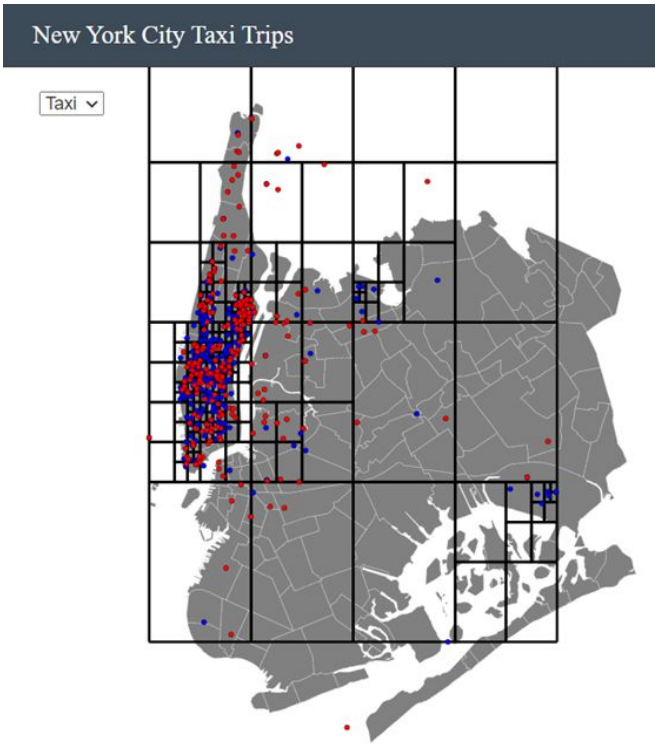
C=400, SR=200
QuadTree height: 7
QuadTree nodes: {interiorNum: 22, leafNum: 67}



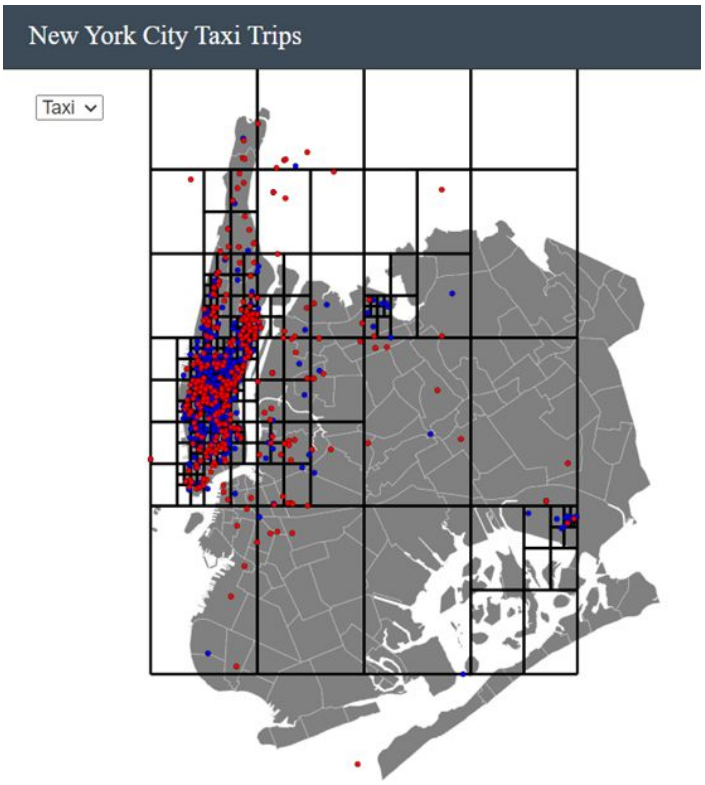
C=200, SR=200
QuadTree height: 8
QuadTree nodes: {interiorNum: 46, leafNum: 139}



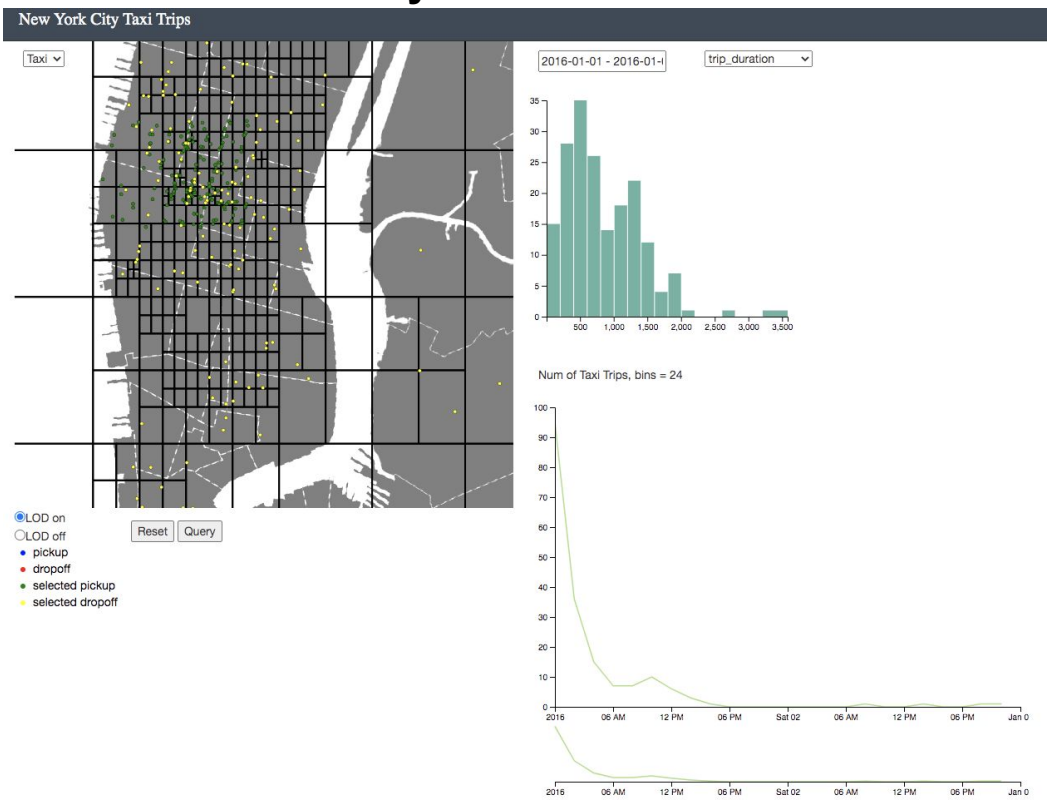
C=100, SR=100
QuadTree height: 9
QuadTree nodes: {interiorNum: 89, leafNum: 268}



C=70, SR=70
QuadTree height: 10
QuadTree nodes: {interiorNum: 131, leafNum: 394}



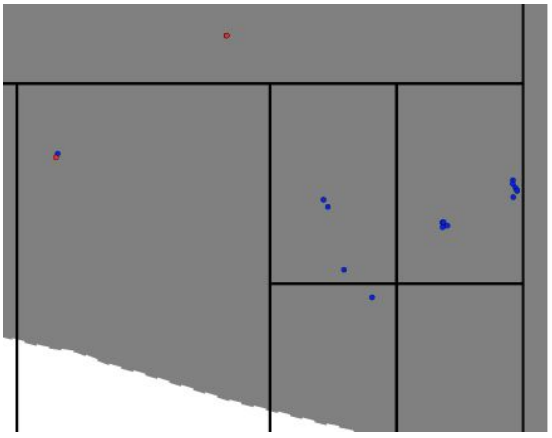
B. Overview of the system



C. LOD example

C=200, SR=200

⟶ zoom in airport area→



--> more points shown up