Taxi-RS: Taxi-Hunting Recommendation System Based on Taxi GPS Data

Xiujuan Xu, Jianyu Zhou, Yu Liu, Zhenzhen Xu, and Xiaowei Zhao

***Abstract*—Recommender systems are constructed to search the content of interest from overloaded information by acquiring useful knowledge from massive and complex data. Since the amount of information and the complexity of the data structure grow, it has become a more interesting and challenging topic to find an efficient way to process, model, and analyze the information. Due to the Global Positioning System (GPS) data recording the taxi's driving time and location, the GPS-equipped taxi can be regarded as the detector of an urban transport system. This paper proposes a Taxi-hunting Recommendation System (Taxi-RS) processing the large-scale taxi trajectory data, in order to provide passengers with a waiting time to get a taxi ride in a particular location. We formulated the data offline processing system based on HotSpotScan and Preference Trajectory Scan algorithms. We also proposed a new data structure for frequent trajectory graph. Finally, we provided an optimized online querying subsystem to calculate the probability and the waiting time of getting a taxi. Taxi-RS is built based on the real-world trajectory data set generated by 12 000 taxis in one month. Under the condition of guaranteeing the accuracy, the experimental results show that our system can provide more accurate waiting time in a given location compared with a naïve algorithm.**

***Index Terms*—Big data, frequent trajectory graph (FTG), recommendation algorithm, taxi Global Positioning System (GPS) data, Taxi-hunting Recommendation System (Taxi-RS).**

I. Introduction

TRAJECTORY can be regarded as the trace of mobile objects in space as times change. The convergence of trajectory data allows the easy acquisition of information about the trajectories of users using mobile devices [1]. Global Positioning System (GPS), one of the growing technologies of geolocation, is widely used on taxis and makes the GPS-equipped taxi be regarded as the detector of urban transport system. It becomes possible to access the location information of taxis of a whole city at any time. Since the amount of trajectory data [3] and the complexity of data structure grow, it has become a more interesting and challenging topic to find an efficient way to process, model, and analyze the mass data of movement.

A taxi is an important tool, for people, to travel in the city, but sometimes it is common, for people, to face an awkward position [2]. Most people had such experience: you had to wait for more than 10 min to get a taxi ride only because you were 2 min late in the specific place. Another example: you had been standing by the street waiting for a cab for over 15 min, but your neighbor just came out of his home, crossed the street, walked a few meters, and got a taxi ride immediately. It seems like people have to collect more information to get a taxi ride instead of longer hopeless waiting. Along with the increase in the amount of taxis, it becomes urgent to solve recommended location information and calculate waiting time by means of large-scale taxi GPS data.

In the information age, it is very easy to collect information. For the mentioned taxi problem, it becomes a very interesting question to recommend location information and calculate waiting time by means of large-scale taxi GPS data.

In this paper, we propose a processing method based on big data environment for the taxi GPS historical data, including offline and online processing. We implement a recommendation system called Taxi-hunting Recommendation System (Taxi-RS), and our system can calculate the probability and time of getting a taxi ride when giving information of time and point. The following are the major contents of this paper.

1) We design data offline processing. First, we design an algorithm called HotSpotScan (HSS) algorithm to scan hotspots in a city. Then, the taxi preference trajectory is recognized by a new algorithm called Preference Trajectory Scan (PTScan) algorithm.
2) We propose an offline graph model by taxi preference trajectory and store offline compressed data by using a multiple adjacent table.
3) We construct a probability model by using the offline model to compute the probability and the waiting time of getting a taxi ride and analyze the complexity of online processing.

The rest of this paper is organized as follows. Section II gives a brief review on related literature. In Section III, some conception is formally introduced. The overview of the Taxi-RS is given in Section IV. In Section V, we introduce the key algorithms of offline processing. In Section VI, we present the frequent trajectory graph (FTG) model. In Section VII, we propose a calculation model to get a taxi ride. The results of the experimental evaluation are given in Section VIII. Remarks are stated under the conclusion in Section IX.

Manuscript received March 4, 2014; revised June 21, 2014 and August 21, 2014; accepted November 12, 2014. Date of publication December 31, 2014; date of current version July 31, 2015. This work was supported by the Natural Science Foundation of China and the Civil Aviation Administration of China through jointly funded projects under Grant U1233110, by the Natural Science Foundation of China under Grants 61202441 and 51209036, and by the Fundamental Research Funds for the Central Universities under Grants DUT13JR01 and 1600-852018. The Associate Editor for this paper was Q. Zhang. *(Corresponding author: Xiaowei Zhao.)*

The authors are with the Software School, Dalian University of Technology, Dalian 116620, China (e-mail: xjxu@dlut.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TITS.2014.2371815

II. Related Work

Urban traffic flow is an important tool to analyze the real-time traffic conditions of the city [4], [12]. In recent years, there has been some corresponding work in the field of analysis and processing technology of GPS traffic data. Some works focused on the route planning, and others are committed to help taxi drivers to find the nearest passenger or maximize drivers' revenue.

A. Trajectory Mining

To make a passenger-hunting recommendation, it is an important application to leverage the knowledge discovered from historical trajectories. The traditional methods for this problem are to identify the place of interests. Zheng *et al.* [7] proposed several new ways of learning the traffic patterns from GPS data. A T-Drive [6] system is an intelligent route planning system, which perceives traffic flow using taxis equipped with GPS sensors and designs the fastest line for ordinary users. T-Drive [6] proposed a routing algorithm based on time-dependent landmark graph. According to the taxi trajectory data, T-Drive can learn the minor time cost by the virtual edge. An improved version of T-Drive [7] takes more factors into account, including the weather, individual driving habits, skills, and other factors. A T-Finder system [8], [9] recommends the most possible place where a taxi might pick up passengers. Clustering algorithm is used to find the location where drivers can get higher income from taxi trajectories. Then, it uses the probabilistic model to analyze costs and risks when drivers and passengers choose different strategies. Another passenger-hunting system is HUNTS [21]. The HUNTS system makes hunting trajectory recommendations for taxi drivers to increase their profits.

B. Route Planning

Route planning can be divided to personalized route planning and dynamic route planning. From the view of recommendation [11], route planning includes taxi-hunting recommendation, energy-efficient driving route recommendation, fast driving route recommendation, and so on. A route-planning algorithm can be divided to global and local algorithms, according to the scope of considered trajectories. Local algorithms are also called incremental algorithms, which match each track point to the appropriate sections by using a greedy strategy. Such algorithms usually find a local optimum section to match each track point based on the similarity between distance and angle [18]–[20]. On the other hand, the considered object of a global algorithm is usually the whole trajectories. Its goal is to find the road sequence that is the closest to the track in some sense. In recent years, some works have used Frechet distance [16] and weak Frechet distance to compare the similarity between tracks [17].

However, previous works usually consider the recommendation system from the drivers' view. They obviously ignore the requirement of passengers. We consider that it is worthy to answer the following questions when a person is waiting to get a taxi ride: Is it easy to get a taxi ride at the current location? How long will it take until the first vacant taxi arrives at the current location? Where is the nearest "best place" to get a taxi? How long will it be there? In this paper, we introduce a new technology, called taxi hunting, to answer the question mentioned.

III. Problem Definition

In this paper, we solve the following two problems by using large-scale data generated by a taxi GPS device: 1) calculate the probability of getting a taxi in a particular place at a certain time; 2) calculate the waiting time until the first vacant taxi hits the requirement. Here are some basic definitions.

A. Basic Definitions

Suppose that there are a collection of *n* taxis, i.e., $Taxi = \{Taxi_1, \ldots, Taxi_n\}$. Every piece of GPS data is denoted as a GPS data point. Here, we give a more detailed definition [5].

Definition 1. GPS Data Point: A piece of GPS data point records the $longtitude_{i,j}$, $latitude_{i,j}$, speed $v_{i,j}$, the direction $dir_{i,j}$ and running state $State - run_{i,j}$, and so on, which is written as $gps_{i,j}$ that generated by a taxi $Taxi_i$ with the GPS equipment at time t_j. That is

$$gps_{i,j} = \{Taxi_i, t_j, longtitude_{i,j}, latitude_{i,j}, v_{i,j}, dir_{i,j}, State - run_{i,j}\}. \quad (1)$$

A map at a time consists of the GPS data on that time and the corresponding taxi's speed vector. Map difference can distinguish the differences between maps on different times, to store big data efficiently. Therefore, the precision of map difference only affects the number of file blocks. The degree of precision has no effect on final results and just has a weak impact on file block.

Definition 2. Map Difference: Given a time t_i, Map_i represents the map on t_i, and R_i represents the eigenvalue of Map_i. Then, the difference $\Delta Map(t_2, t_1)$ of Map_1 and Map_2 can be defined as

$$\Delta Map(t_2, t_1) = Map_2 - Map_1. \quad (2)$$

Similarly, the eigenvalue difference between Map_1 and Map_2 can be defined as follows (the specific calculation method of R_i in Section V-A):

$$\Delta R(t_2, t_1) = R_2 - R_1. \quad (3)$$

Definition 3. Frequent Surface: The frequent surface is 3-D curved surfaces, defined as $TCM(x, y, z)$, where x represents the longitude, and y represents the latitude. Then, we defined z as $z = count(x, y)$, which means the number of taxi flows on the $Point(x, y)$. Some locations have a large number of taxi flows, and we call these points as hotspots. Meanwhile, some areas have a large number of taxi flows, which are called hotspot zones. Each hot point is the smallest indivisible hotspot zone. One way to define hotspots is as follows.

Definition 4. Hotspot: For each GPS $coordinates(x, y)$ on the map, x represents the longitude, and y means the latitude. If the number of taxis through the point within one day is greater than the threshold K, we called the point a hotspot, denoted by $HotSpot(x, y)$. According to definition 3, hotspot can be defined by a collection represented as

$$(x, y)|z = count(x, y) \geq K. \quad (4)$$

Definition 5. Hotspot Zone: The hotspot zone gets the adjacent hotspots together; at the same time, the $count(x, y)$ of the adjacent hotspots is similar. A hotspot $HotSpot(x, y)$ is selected to represent the hotspot zone, denoted by $HotSpotZone(x, y)$. We can calculate the running track of a taxi by using GPS data. Therefore, it is accurate and feasible to get the taxi trajectory from GPS data. Hence, we make the following abstract. First, distinguish taxi trajectories according to the identification of each car, i.e., every taxi has its own path, and those trajectories are independent of each other. Second, trajectories are the historical data of taxis running. If there is a trajectory, a taxi must go through this route at some moment to form the trajectory. In the following, the definition of trajectory is introduced, and the related notations are given.

Definition 6. Trajectory: A taxi trajectory Tr is the running path of a $Taxi_i$ during the period of time (from t_j to t_k) and is represented by the GPS data $(gps_{i,j}, \ldots, gps_{i,k})$. We use $gps_{i,j}, \ldots, gps_{i,k}$ to display the trajectory of $Taxi_i$ during the period of time. Therefore, all of the trajectories consist of a collection of trajectory. Using trajectory notation, a formal definition of frequent trajectory is easily given as follows.

Definition 7. Frequent Trajectory: If the appearance number of a trajectory is recorded as $count(Tr)$, frequent trajectories are trajectories whose appearance number is greater than the specified threshold FT. That is

$$FrequentTrajectory|Tr : count(Tr) > FT. \tag{5}$$

Definition 8. FTG: Suppose that we have a set of hotspots $hs_1, \ldots, hs_k$. A hotspot database is denoted by a directed graph $G = V, E$. The set of hotspots V denotes all hotspots, i.e., $V = hs_1, \ldots, hs_k$. E is a set of all edges. It is stipulated that an edge e_{ij} from hs_i to hs_j means that hotspot hs_j is the successor of hotspot hs_i.

B. Problem Definition

As a whole, the taxi-hunting problem of this paper is a local-optimal trajectory retrieving problem, which is to calculate the waiting time to get a taxi ride successfully at a certain time in a certain place. The formal definition of the problem is shown in problem definition 1.

1) Problem Definition 1: Arrival time of the first taxi (ATFC): Given large-scale GPS data of a city, find out the waiting time $T(x, y, t, m)$ until the first vacant taxi arrived at $Point(x, y)$ in m minutes at time t, where x represents the longitude, y represents the latitude, and m represents the waiting time.

The ATFC is the waiting time for a passenger to get a taxi ride successfully at a certain time in a certain place. For computing the ATFC, we have to get the probability of getting a taxi ride (PGTR); hence, we define the probability problem of getting a taxi ride, as shown in problem definition 2.

2) Problem Definition 2: PGTR: Given large-scale GPS data of a city, find the probability $P(x, y, t, m)$ that a taxi could reach a particular place $Point(x, y)$ in m minutes at a certain time t.

To solve the two questions, we give the notation of effective trajectory. Effective trajectory refers to a trajectory that there is at least one running taxi for the current query times corresponding historical time.

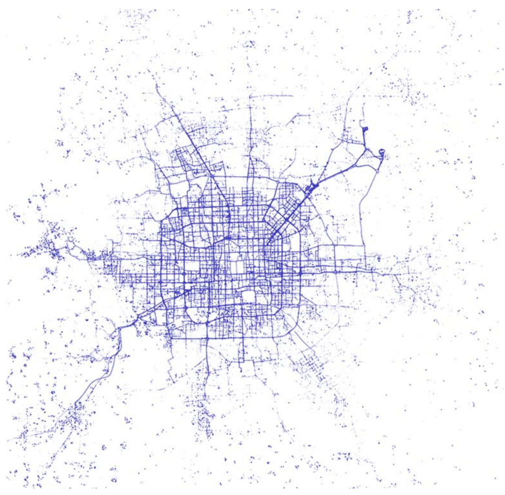


Fig. 1. $G(x, y, n)$ on November 30, 2012.

Definition 9. Effective Trajectory: Suppose that the set of taxis through the point in historical time is denoted by $Taxi_i, \ldots, Taxi_j$. Effective trajectories are the set of taxi trajectories that those taxis had running through the $Point(x, y)$ during the historical period of $[t, t + m]$, as

$$Tr_{\text{valid}} : \left\{gps_{(u,t)} \rightarrow gps_{(u,(t+1))} \rightarrow \ldots \rightarrow gps_{(u,(t+m))}, u \in [i, j]\right\}. \tag{6}$$

There is an important characteristic of effective trajectory. If an effective trajectory covers the place of getting a taxi ride for query times corresponding historical time, it will be high for the success PGTR at this point on query time. We can give the definition of invalid trajectory by using the definition of effective trajectory. In the process of dynamic query, invalid trajectory is meaningless for query.

Definition 10. ETG: Effective trajectory graph (ETG) is a graph composed of taxi effective trajectories that those taxis had running through the $Point(x, y)$ during historical $time[t, t + m]$. ETG $G(V, E)$ is a subgraph of FTG. For example, we can generate an FTG, as in Fig. 1.

IV. Overview of Taxi-RS

The core idea of this paper is to compress and converse of large-scale GPS data, to obtain an offline model for dynamic query. At each dynamic query processing, we calculate the waiting time and the PGTR, using the offline model. The Taxi-RS system is divided into three modules: the offline preprocessing model, the FTG model, and the online processing model, as shown in Fig. 2.

1) Offline preprocessing part obtains offline model training based on GPS big data.
2) The second part is to obtain the frequent trajectory model, which is used to query online.
3) For each query input, search the FTG model, to calculate the probability and the waiting time.

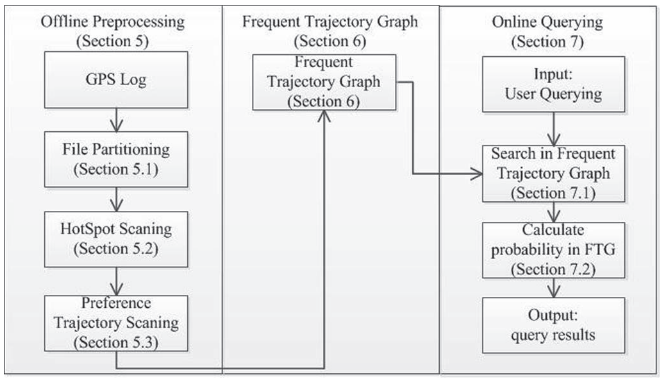

Offline Preprocessing (Section 5)
GPS Log
File Partitioning (Section 5.1)
HotSpot Scaning (Section 5.2)
Preference Trajectory Scaning (Section 5.3)
Frequent Trajectory Graph (Section 6)
Frequent Trajectory Graph (Section 6)
Online Querying (Section 7)
Input: User Querying
Search in Frequent Trajectory Graph (Section 7.1)
Calculate probability in FTG (Section 7.2)
Output: query results

Fig. 2. Taxi-RS system overview.

V. Offline Preprocessing Algorithm

This section introduces offline preprocessing, which consists of three stages: file partitioning, hotspot scanning, and preference trajectory scanning (see in Section V-A–C). The processing procedures are as follows.

A. File Partition

First, we read the original GPS data and divide it by days into seven parts since a week has seven days. Taking into account the continuity of taxi running, the data difference between adjacent time slots is smaller compared with all data in the entire time slot.

However, the file of one day is too big to process. We solve this problem by the following two steps. First, we will store the whole GPS data during a period of time, for instance, 1 min. Then, we store the data of the next time slot through a difference approach. Therefore, we think that the appropriate time period to divide the file is 24 h. To calculate the map difference between time t_1 and t_2, we need to calculate a map eigenvalue by (7). Assume that $Taxi = \{Taxi_1, \ldots, Taxi_n\}$ is the set obtained from the GPS data. To define R_i of the map Map_i features, two essential features should be considered: 1) the GPS location $Point(x, y)$ of each taxi at time t_i; 2) the speed of each taxi at t_i. Before giving the formal definition of R_i, we will define several intermediate variables.

1) Lo_{xi} (in meters) represents the longitude of the location of $Taxi_x$ at time t_i, which is converted into the lateral distance of the reference point (see Section V-B about conversion processing).
2) La_{xi} (in meters) represents the latitude of the location of $Taxi_x$ at time t_i, which is converted into the longitudinal distance of the reference point (see Section V-B about conversion processing).
3) S_{xi} denotes the relative distance between the location of $Taxi_x$ at time t_i and the reference point, which is defined as $S_{xi} = \sqrt[n]{\Delta Lo_{xi}^2 + \Delta La_{xi}^2}$.
4) v_{xi} denotes the speed of $Taxi_x$ at time t_i corresponding to the GPS data (in meters per second).
5) SET_i denotes the set of all GPS data at time t_i. The definition of R_i is given by

$$R_i = \frac{\Sigma_{gps_{xi} \in SET_i}(S_{xi} \times v_{xi})}{(|SET_i|)}. \quad (7)$$

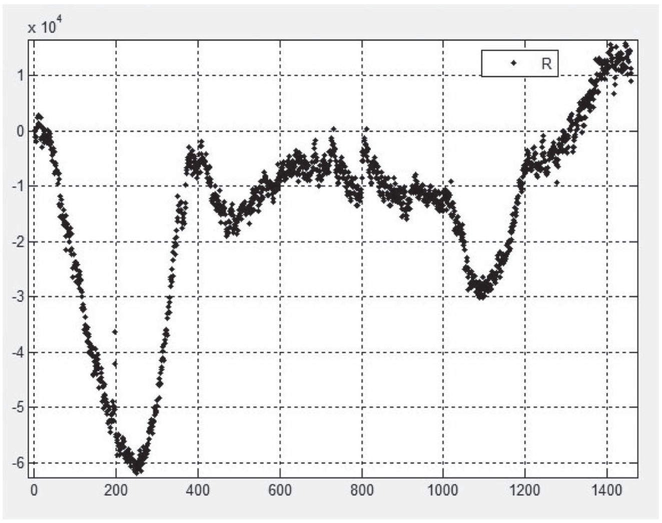


Fig. 3. Graph of the map eigenvalue with time, on November 30, 2012.

R_i can be considered as the sum of the relative distance between the locations of the driving taxis and the reference location at time t_i. R_i could be used to represent the traffic situation of the taxis in a city at the moment t_i. Fig. 3 shows the map eigenvalue with every minute of November 30, 2012. Then, the file of a day is divided into four parts.

B. Hotspot Scanning

Since a taxi trajectory can be abstracted into a pattern string spliced by the time and place of a taxi, we can compress pattern strings by selecting hotspots. It is necessary to find out hotspot zones. Therefore, we can get all pattern strings of trajectories when data points of every taxi map to hotspots.

We scan every hotspot from each file block to generate frequent surface $TCM(x, y, z)$, as mentioned in definition 3. Hence, we can accelerate the access speed by simple Hash function. We can use Algorithm 1 to get the frequent surface (as shown in Fig. 1).

Algorithm 1. Generate Frequent Surface.

```
Input: All GPS data $gps_1, \ldots, gps_n$ of a day.
Output: $z = count(x, y)$ in $TCM(x, y, z)$.
1) For $i = 0$ to NumberOf(GPS-Data)
    $count[i] = 0$;
2) For $i = 0$ to NumberOf(GPS-Data)
    $x = gps.latitude[i] \times 111.7 \times 1000/10$
    $y = gps.longtitude[i] \times 111.7 \times cos(39.5/180) \times 1000/10$
    $hash = x \times 100000 + y$
    $count[hash] = count[hash] + 1$
3) $z = count$;
4) Return $z$.
```

Then, we will find out that appeared points constitute the main roads. Some roads are particularly intense, and some places are very sparse. Therefore, hotspots and hotspot zones exist in reality. Hence, we propose the HSS algorithm to simplify the computation by polymerizing hotspots. At the same time, to ensure the computation accuracy, we would give the threshold k. For the surface $z = count(x, y) = K$, the map is divided into multiple hotspots. Similarly, we can get multiple hotspot zones by $k = 0$, i.e., $z = 0$. Generally, $count(x, y) = K$ can guarantee that the zone belonging in the $x-y$ plane is a closed curve. All points within the closed curve are frequent points of GPS data. Therefore, we have completed the preliminary selection of hotspots and hotspot zones.

We adopt the idea of adaptive partition to implement the hotspot scanning algorithm. We define a variable F as the threshold K, which is greater than $(count(x, y))$.

Algorithm 2. HSS Algorithm.

```
Input: Frequent Surface $TCM(x,y,z)$, $F = (count(x,y))$, thread lock Lock.
Output: The number of hotspot zones, representative point for each hotspot, the range of each hotspot zone.
1) If ($N$ greater than $10^5$) or ($F$ equals 0) return;
2) $sub\_TCM[4]$ = simple-division(TCM);//4 sub-block is divided by quadrant number.
3) $F = F/4$;
4) For $i = 0$ to 3
     $C_i = sum_{count}(sub\_TCM_i)$
5) remove $TCM$;
6) $N = N - 1$
7) For $i = 0$ to 3
     if $C_i$ not far less than $F$
         add $sub\_TCM_i$
         hot-spot-scan($sub\_TCM_i$, $F$, Lock)
         $N = N + 1$.
```

The HSS algorithm is a recursive partition process; at the same time, it also contains the idea of breadth-first search. In the process of recursion, we maintain the value of N, which is the number of hotspots divided. Therefore, a recursive boundary condition is that the number of hotspots is greater than 10^5 or that the current number of hotspots is greater than $F = \sum(count(x, y)) = 0$ (line 1).

In every step of the recursion, the current frequency surface is congruently divided into four parts: upper left, upper right, lower left, and lower right (line 2); at the same time, the current threshold F is set to the original 1/4 (line 3). We get $(count(x, y))$ for every subsurface (line 4), and the SUM_count procedure is optimized by the classic data structure, i.e., interval tree. Consequently, we remove the step of TCM and N minus 1 (there would be a smaller range TCM to replace the original TCM; we prove its validity in the Appendix). Then, we traverse the four subblocks to save $count(x, y)$ of TCM, which is not less than F. N is added by 1, and the algorithm enters the lower recursion. Simultaneously, we will compute sub_TCM recursively, so that the algorithm runs until all subsurface has been partitioned. Similarly, we can prove the correctness of the HSS algorithm, namely, its reachability of recursive boundary.

C. Preference Trajectory Scanning

The scale of the initial GPS data will be greatly reduced after the hotspot scanning, but the selected data are still too large to the final solution. We note the following phenomenon.

1) A taxi driver often finds passengers in a few fixed streets.
2) Few trajectories happened by accident. For example, a passenger's destination is strange, and the driver has to go this route when he returns.
3) Sometimes, a taxi driver goes through certain routes due to some special factors.

The three cases mentioned earlier are very normal and frequent in real life. However, these cases have a great influence on the results of these taxis recommended. The recommendation algorithm should be based on more frequent and more general trajectories. Thus, we should pay attention to the trajectories mentioned in case 1. Meanwhile, we try to get rid of the trajectories mentioned in cases 2 and 3. In real life, some drivers prefer some routes; in other words, the selection of routes is preferred. A driving trajectory of a taxi can be viewed as a sequence related to time. Thus, trajectories of preferred routes must be frequent. There are many available taxis on frequent trajectories; hence, we can ensure that an available taxi would arrive at some location within a reasonable time. To identify preference trajectories, we need to identify taxi trajectories at key points.

Due to the use of historical data in offline processing, we scan GPS historical data by using the idea of PrefixSpan algorithm [10] to find a substring that appears frequently. We call the algorithm PTScan. PTScan algorithm selects frequent sequences from candidate sequences, by filtering out infrequent sequences. According to our experiments, the threshold value of frequent trajectory is 10.

Here, we give some symbols for the PTScan algorithm. The symbol ω is denoted as a taxi trajectory within a time slot, which can be viewed as a sequence. The symbol S represents a substring of ω. The symbol $S|\omega$ means the projection of ω on S. In particular, if ω is empty, $S|\omega$ is S.

Algorithm 3. PTScan Algorithm.

```
Input: GPS historical data, $\omega$.
Output: Frequent substring.
1) Scan $S|\omega$, find the sequence b with $length(b) = 1$;
2) Sequence $b$ is added to the end of $\omega$ and become a new sequence.
3) $< b >$ can be used as the last element of $\omega$ and become a sequence.
4) For each sequence $b$, adding $b$ to $\omega$ so as to form a sequence $\omega$, and output $\omega$.
5) For each $\omega$, construct its projection database $S|\omega$, and call the algorithm $PTScan(\omega, length(\omega) + 1, S|\omega)$.
```

Basically, the PTScan algorithm works as follows. Initially, we scan the taxi trajectory sequence database to generate all sequences of length 1. Then, the corresponding projected database is formed according to the sequence whose length is 1. Consequently, we repeat the previous steps on the corresponding projection database until no more sequence of length 1 is generated. Finally, we repeat the process for each different projection database until no new sequence of length 1 is generated.

The PTScan Algorithm continues to produce multiple smaller projected databases of the trajectory sequence database by using the divide and conquer idea. Then, the algorithm mines sequential patterns on each project database to obtain the taxi driving preference trajectory. The PTScan algorithm focuses on sequence patterns of available taxis.

VI. FTG

This part introduces how to design the FTG model and how to construct the storage and related data structure of the frequent trajectory model. We obtain frequent pattern substrings of taxi trajectories according to the offline preprocessing results. Since the key of the paper is to solve the probability that can get a taxi ride at a query point in m minutes, we need to analyze and construct the factors that can affect the probability of the final solution.

If an available taxi arrived at a designated location in m minutes, we must ensure the following conditions.

1) To ensure that an available taxi can reach a location in m minutes, we need to analyze the speed and the distance from historical data.
2) To ensure that other people would not get the car ride when it comes on its way, we need to analyze the trigger conditions in historical data, and calculate the PGTR by others on its way in each direction.
3) To ensure that query location is on the preferred trajectories. Therefore, we construct the FTG.

For the query time t_0 in the FTG $FTG = (V, E)$, two symbols $w(V)$ and $w(E_{i,j})$ are proposed. $w(V_i) = n$ denotes the total number of available taxis in an aggregation point V_i during the moment $[t_0 - m, t_0]$. $w(E_{i,j}) = (P, T)$ shows two tuples of P and T. P represents the probability that a taxi is running from $Point_i$ to $Point_j$ in m minutes. T represents the time of using by a taxi from $Point_i$ to $Point_j$.

It is not complicated to construct the FTG. However, *FTG* will have $|V| = N \approx 10^5$ nodes, and it is approximate to a complete graph; hence, $|E| \approx |V|^2 \approx 10^{10}$. Therefore, it will produce too enormous space cost to acceptable if *FTG* is merely stored in classic adjacency matrix and adjacency table. For instance, the complexity of the adjacency matrix space is $O(|V|^2)$. The complexity of the adjacent table space is $O(E)$. We consider the conception of effective trajectory when querying, so that the new data structure can be stratified by the time. Hence, we design the new data structure shown in Fig. 4, which can reduce the space. We use n to represent a 10 m × 10 m hotspot statistics frequency in the new data structure, from the sparsest area to the densest areas by an ascending order. In fact, there exists a lot of the repeated frequency of hotspot zones; hence, there are 10^5 different values. We use t to represent time from 0 to 24 by stratifying per minute. It would be convenient for querying.

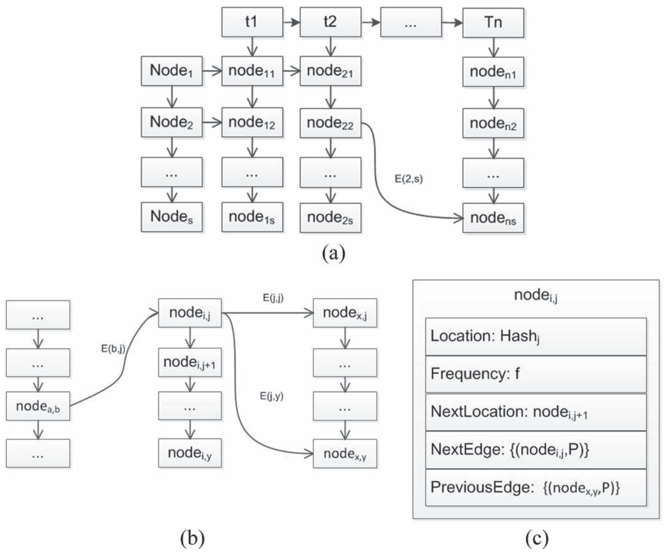


Fig. 4. Sketch map of the main data structure. (a) TMAT. (b) $node_{i,j}$, precursor node: $node_{a,b}$, and successor node: $node_x$. (c) Data structure graph of node $node_{i,j}$.

In Fig. 4(a), t_i points to the situation of all the divided hotspots in the current time and divides the map of per minute into about $N \approx 10^5$ hotspots. Hash value of the hotspot is uniquely identified. Each hotspot in data structure is expressed as $node_{i,j}$. $node_{i,j}$ represents the hotspot on a time t_i of $Hash = j$. This data structure is called a time-stratified multiple adjacent table (TMAT).

Suppose that $node_{i,j}$ represents node V_j in the time of t_i. $node_{x,y}$ represents node V_y in the time of t_x. As shown in Fig. 4(b), $E(j, y)$ represents the edge relationship of $node_{i,j}$ and $node_{x,y}$ in the graph model, $w(E_{j,y})$ denotes the two tuples (P, T), and the two values have been calculated. In addition, we need not store $w(E_{j,y}).T$ alone because it can be calculated by the data structure directly $(w(E_{j,y}).T = t_x - t_i)$. We only need to store $w(E_{j,y}).P$.

Therefore, it can be mapped into a multiple adjacent table for hotspot trajectory strings obtained from the PTScan algorithm. Meanwhile, we update the edge information in the TMAT. For example, for an edge $node_{i,j} \to node_{x,j}$, if the location does not move as the time goes, then it reflects the parking state of the taxis.

The data structure is shown for each node $node_{i,j}$, in Fig. 4(c). The following is the explanation of corresponding data of $node_{i,j}$.

Location: $Hash_j$ denotes the specific hotspot location represented by $node_{i,j}$. It is represented by the Hash value calculated earlier.

Frequency: f denotes the vehicle frequency of $node_{i,j}$ in moment t_i.

NextLocation: $node_{i,j+1}$ denotes the next adjacent node $node_{i,j+1}$ of $node_{i,j}$ in moment t_i.

NextEdge: $node_{x,y}, P$ denotes the subsequent nodes in the frequent pattern substring of the vehicle $node_{i,j}$ (edge relationship in Fig. 5) and corresponding weight P after the moment t_i.

PreviousTime: $node_{a,b}, P$ denotes the precursor nodes in the frequent pattern substring of the vehicle $node_{i,j}$ (edge

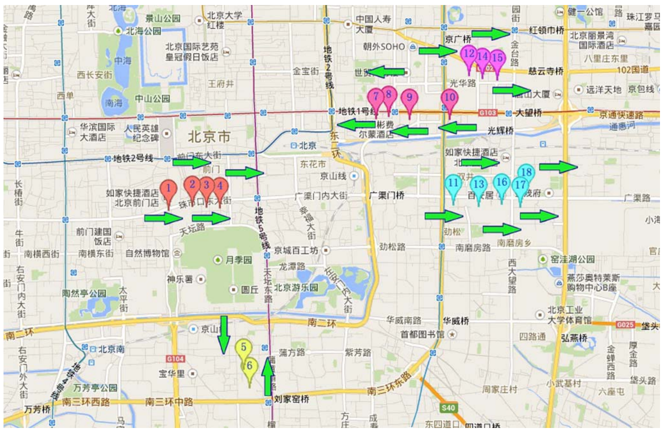


Fig. 5. Test locations.

relationship in Fig. 5) and corresponding weight P before the moment t_i.

For TMAT, we calculated and stored two files (weekday offline model and weekend offline model), respectively. In addition, in the subsequent query, we can read the corresponding data structure file for a specified date.

VII. Online Querying Algorithm

Here, we introduce a dynamic online processing algorithm based on the offline model. First, we can get the following information for an inquiry:

1) the location (x, y) where we want to get a taxi ride, represented by GPS coordinates;
2) time of the query;
3) inquiry acceptable waiting time m.

For an inquiry point, we find out suspicious points, which would affect the PGTR. Meanwhile, we also have an FTG by training according to the existing large-scale data. The problems can be solved dynamically. First, we locate the hotspots on the offline graph model based on GPS coordinates. Next, we find suspicious hotspots around the hotspot zones that can affect the PGTR. The core idea is the searching process. The online query part is as follows.

1) Locate the query location $Point(x, y)$ to the corresponding hotspot zone.
2) Find out the number of the trajectories that may arrive at the query point $Point(x, y)$ according to the time information t (see in Section VII-A).
3) Estimate parameters in the probability modeling of getting a taxi ride according to the trajectory number in m minutes. Calculate the probability and waiting time of getting a taxi ride (see in Section VII-B).

After getting the offline model, we make a quick search by index of TMAT table. First, we locate the user's query location $Point(x, y)$ to the corresponding hotspot zone in the offline model. Second, according to the query time t, we find the number of trajectories that pass by $Point(x, y)$ during $[t, t+m]$. Then, the probability is in accordance with the Poisson distribution after we analyze; hence, we can estimate the parameters of Poisson distribution according to the trajectory number. Finally, we calculate the PGTR through the Poisson distribution and calculate the waiting time according to the geometric distribution. In summary, online query speed is very fast.

A. *Solution for Two Problems*

Here, P represents the probability of a taxi from the point $Point_i$ toward $Point_j$ within m minutes. T represents the time of a taxi from the point $Point_i$ toward $Point_j$. The method to solve P and T is as follows.

1) The method to solve P. The impact factors of P have taxi driving preferences and the condition of getting a taxi ride during the way.
   a) For taxi driving preferences: For all the hot string, we statistically obtain the number n_i of occurrences of V_i'. The number n_i of occurrences of V_j' is a direct successor of V_i', as shown in
   $$P_{i,j(\text{preference})} = \frac{n_i}{n_j}. \tag{8}$$
   b) For the condition of getting a taxi ride during the way: The events of getting a taxi ride are independent of each other; hence, all events that the relevant taxis are being ridden are independent and identically distributed. Suppose that p_k represents the PGTR on its way. p_k can be calculated using
   $$P_{i,j(\text{non-get})} = 1 - \Pi_{k \in E_{i,j}}(1 - p_k). \tag{9}$$
   P could be calculated by
   $$P_{i,j} = P_{i,j(\text{preference})} \times P_{i,j(\text{non-get})}. \tag{10}$$
2) The methods to solve T.

Suppose that there are two hotspots V_i and V_j. Suppose that S denotes the average traveling distance from V_i to V_j. Suppose that H denotes the average speed from V_i to V_j. Then, we could calculate the driving time from V_i to V_j by the equation $T = S/H$.

B. *Optimization Process*

All historical data have been mapped on the new data structure, i.e., TMAT graph; then, the problems are solved by searching the strategies of the graph. For each querying point (denoted by the coordinate of the location), we should search the corresponding hotspot. Then, we search it in the opposite direction for all edges into the hotspot. Consequently, we could get a querying tree during searching the graph [13]–[15]. The querying location is the root of the tree, and the related hotspots on the TMAT graph are twigs of the leaves.

We maintain two values in the search process. For the starting point, we define a pruning optimization parameter θ and a searching boundary μ.

Initially, $\mu = m$. When entering a new layer of searching, μ minus $w(E).T(\mu = \mu - w(E).T)$. At the same time, the search boundary will be determined by μ. (Obviously, the boundary should be $\mu = 0$. We could consider that a taxi is driving from the query location to around locations within the time limit of m minutes.) On the other side, θ is obtained during the merging process of rising from leaf nodes to the root on

TABLE I
NUMBER OF STATISTICAL TEST POINT IN EACH DIRECTION

Direction	Number
from east to west	90
from west to east	20
from north to south	15
from south to north	300
summary	425

the searching tree. For the father–son relationship parameter θ, we use the multiplication operation. The parameter of the brotherhood is calculated by

$$\theta = 1 - \prod_{i=0}^{m} (1 - w(E).P). \tag{11}$$

At the same time, we use the pruning optimization parameters β_i to prune the search optimization. In search of the ith layer, the side of all probability for the current layer is the collection $PS_i = \{w(E)P_{i0}, w(E).P_{i1}, \ldots, w(E).P_{im}\}$, and we define the magnitude as β_i. Then, β_i is calculated by

$$\beta_i = lg\left(\frac{\sum_{p\in PS_i} lg\,(w(E).P)}{|PS_i|}\right). \tag{12}$$

The meaning of β_i is the average digits of probability on the ith layer. For example, if $\beta_i = -2$, it indicates that the average probability level of the current layer is 0.01. When searching to the jth layer, we will find that the probability is the product of the relationship between layer and layer, and the probability $p \in [0, 1]$. Therefore, when search depth is deep enough, the probability of this layer has slight effect on the overall probability. Although μ has not reached its boundary in the process of search, such searching is meaningless, and hence, we need to prune it. When searching on the ith layer, α_i will be updated by

$$\alpha_i = \sum_{i=0}^{n} \beta_i. \tag{13}$$

Equation (13) represents the magnitude of the current probability when the searching is getting to the current layer. Obviously, it should be pruned if $\alpha_i \ll \beta_i$. If $\alpha_i - \beta_i < -5$, it can be considered as $\alpha_i \ll \beta_i$ because we take the logarithm in the process.

In the searching process, we found that the two searching methods are probably similar or even identical. Therefore, we record the search results and use the method of the memorial searching, to speed up the searching process.

Meanwhile, we would like to add a new offline file. Given a query location, we first query the offline files to check if it is queried in the previous query. If the query location has not been queried, we would not search it on the tree.

Algorithm 4. Algorithm for Searching and Pruning for Optimization [SPO Algorithm (μ, β)]

Input: (1) a taxi location (x, y), expressed by GPS coordinates and conversed of query to the corresponding hotspots; (2) the query time t; (3) the acceptable waiting time m for query; (4) pruning optimization parameter λ and the remaining time μ in searching process; (5) the global parameter α and β.

Output: For this query, the total probability and total number of taxis which pass by the query location. (In each layer of searching, we store the probability and time-consuming on each edge.)

```
1) if $\mu$ not greater than 0
    a) save $(\beta, m - \mu)$
    b) $N = N + M \times \beta // N$: represents the total number of taxis that could arrive the querying location, and $M$ represents the total number of taxis that could be searched from the query location
    c) return 0
2) if $\beta$ less than $\alpha - 5$
    a) save $(\beta, m - \mu)$
    b) $N = N + M \times \beta$
    c) return 0
3) $p = 1$
4) for $e$ in adjective $E$
    a) $p = p \times (1 - SPO(\mu - w(e).T, \beta * w(e).P))$
5) return $1 - p$
```

In summary, the SPO algorithm includes the pruning optimization during the searching process. It improves the efficiency of searching by using the memorial search.

C. Computation Process for Two Problems

Using the SPO algorithm, we can calculate the number of taxis coming from all possible points, which is equal to the number of expected taxis. For this problem, the PGTR is consistent with the Poisson probability distribution. For a particular time and a particular location, the probability distribution function is

$$P(x = k) = e^{-\lambda \cdot \lambda^k / k!}. \tag{14}$$

The parameter λ in (14) is the average probability of occurrence per unit time (or per unit area) of random events. Then, Poisson distribution is suitable for the description of the number of times that random events occur per unit time.

Generally, the mathematical expectation of Poisson distribution is represented by $E(P(\lambda)) = \lambda$, for the current question $P(\lambda)$. We could obtain the coefficient of Poisson distribution. Hence, the probability of being able to take a taxi is

$$P = P(x \neq 0) = \lim_{(n\to\infty)} \sum_{i=1}^{m} P(x = i) = 1 - e^{-\lambda}. \tag{15}$$

Meanwhile, the time when the first taxi comes obeys the geometric distribution; hence, we can calculate the time by the following procedure.

First of all, for all adjoining sides of the query point, we could obtain the average time, which is spent on coming to corresponding sides. Then, we can obtain a series of tuple $(P_1, Time_1), (P_2, Time_2), \ldots, (P_n, Time_n)$, by sorting the average time $Time_i$ in ascending order. ($Time_i$ is sorted in ascending order, and we can obtain the series of tuple by the

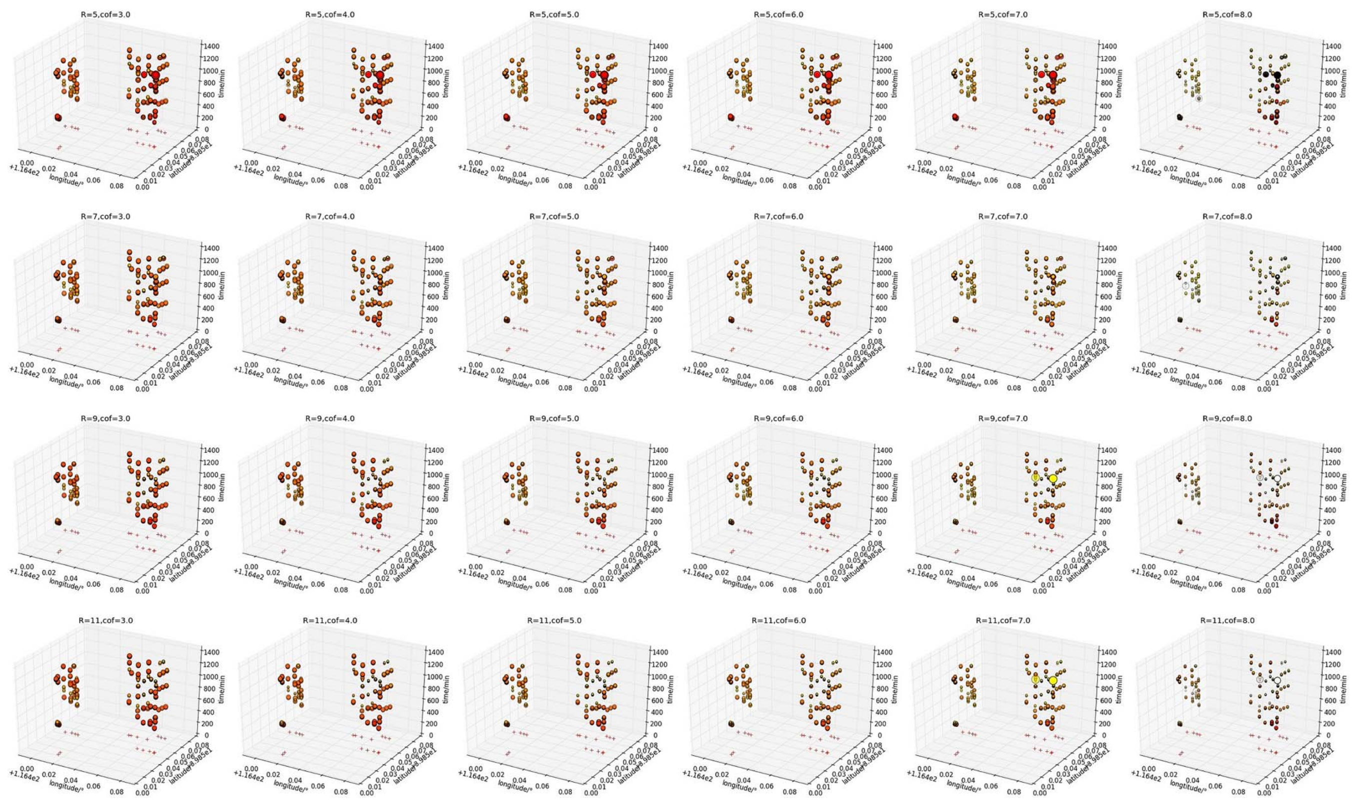


Fig. 6. Different results of different parameters for R and cof.

earlier searching algorithm.) To calculate the time used to wait, we can use

$$\begin{aligned} T = &P_1 Time_1 + (1-P_1)P_2 Time_2 \\ &+ (1-P_1)(1-P_2)P_3 Time_3 + \cdots \\ &+ \prod_{i=1}^{n-1}(1-P_i)P_n Time_n. \end{aligned} \tag{16}$$

The algorithm of online processing is completed.

In summary, for a particular location (x, y), within a certain time t_i, the PGTR within m minutes is affected by the parameter λ that is the number of taxis coming within m minutes. The wait time is affected by a tuple, which is built up by P and T in (16).

VIII. Experiments

Here, the results of the experimental evaluation using a real database will be shown.

A. *Experimental Data Set and Experimental Platform*

We build our system based on a real taxi data set generated by over 12 000 taxis in a month, in Beijing City. There are about 1.1 billion GPS data. There are 30 million with the size of 2-G data for each day. The data set recorded the taxi trajectories. Offline preprocessing and frequent trajectories of this paper are constructed on top of Hadoop platforms.

1) Offline Processing Analysis: Most of the time for offline processing is consumed during the method of information processing. Offline processing costs only 5 h in total. Since, for the given data, a recursive operation (10^9) is $O(n)$, the maximum complexity of offline processing is $O(n \log n)$. The other algorithms are $O(\varepsilon n)$. For offline processing, we do a good optimization.

2) Online Processing Analysis

It costs 20 min to construct frequent trajectories for one day. Online processing uses the FTG, which includes information of hotspot trajectories and frequent statistic. Such storage method can save more space than general graph models (such as adjacent table). Although the final storage space for offline results need 5.5 g, this is inevitable to ensure the accuracy.

3) Time-Consuming Analysis of the Model

Each inquiry online can be completed in about 0.5 s because the relevant data have been compressed to store. This is due to the process of pruning the search optimization (SPO algorithm) and the original Hash treatment in the algorithm 1. After the mentioned processing, the search depths are small. The search depths are 3 during computing a PGTR successfully and 15 during computing waiting time. When a probability is smaller than 0.001 ($p < 0.001$), we prune the search tree of an inquiry point. Therefore, the mean search depth is 5. Coupled with the memory technology (dynamic programming), the time complexity approximates to linear computing. The number of tests in each direction is the statistics in Table I.

B. *Experimental Results*

We use a set of specific locations in some specific time as the test data. We selected 18 locations, and each location is tested

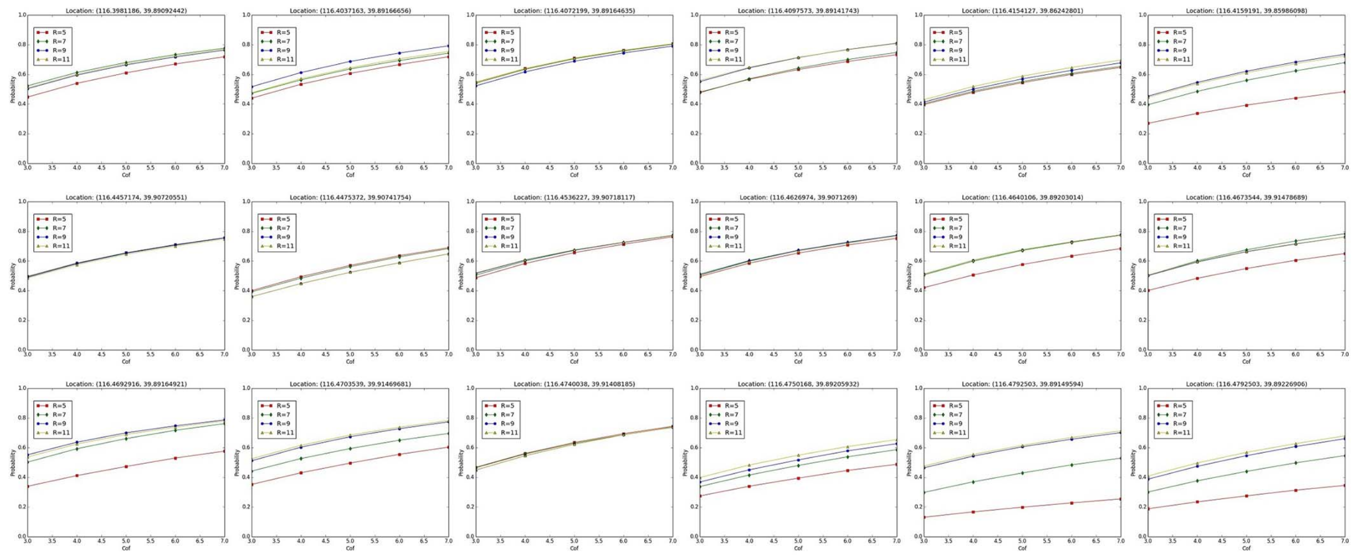


Fig. 7. Probability distribution with the change of parameters R and cof.

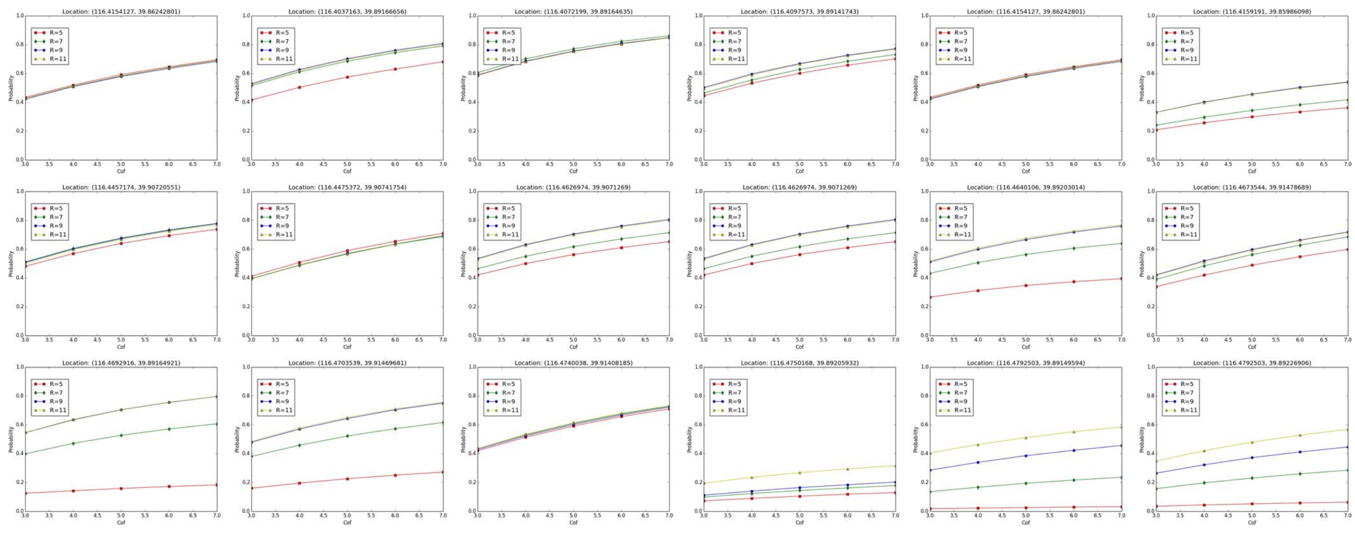


Fig. 8. Probability distribution with the change of R and cof (naïve).

from 5 to 35 times for evaluation. Then, there are a total of 425 test data. The average test time for each location is about 23 for testing. Time was distributed between September 9, 2013 and September 15, 2013. Each test location is tested only in one direction. The direction of the test for each test location is selected from one of the following four directions: from east to west, from west to east, from north to south, or from south to north. Fig. 5 shows the distribution of these locations.

Fig. 6 (from $a1$ to $d6$) shows the different results of different parameters for R and cof. In order to make the probability more visible, it is a good way to magnify the expectation of the Poisson distribution (λ). This would keep the difference among the probability and give a better user experience. Then, we define parameter cof to show the magnification times of λ.

In these figures, the x-axis represents the latitude, the y-axis represents the longitude, and the z-axis denotes the minutes passed from 00:00 of the day. Each circle represents a sample data, the location of the sample illustrated, the time, and the real location (the projection of circles on the plane $z = 0$; the symbol is $+$) of the sample. The colors of the circles range from red to yellow, which show the probabilities of catching a taxi for the sample data. When the color is more yellow, the probabilities are higher. The size of the circles shows the waiting time of the sample, the bigger the circle is, the longer the time is. It can be obviously discovered that the color of the bigger circle is much more yellow; this proves the inversely proportional relationship between the probability and the waiting time, effectively.

Fig. 7 (from $a1$ to $c6$) shows the probability distribution with the change of parameters R and cof at the specific location. In Fig. 7, the probability increases by the increase of either of the parameters R and cof. It can be explained that the R is the radius of a searching scope. When the searching scope extended, the more hotspots would be included to calculate the probability, and the value probability would be much higher (Since the sample data are not selected at the same time, some figures might show that R is high but the probability is low). Similarly, the parameter cof is the coefficient of λ in the Poisson distribution; there is a direct proportion between the probability and the cof. There is an interesting phenomenon: if some locations have a similar probability distribution, they shall be near to others in reality, and the traffic conditions are similar too.

In Fig. 8, we compare our algorithms with a naïve algorithm. In the naïve algorithm, we compute the waiting time by the average value of getting a taxi ride in a month, and calculate the PGTR by the average probability of getting a vacant taxi ride in a month.

IX. Conclusion

Our contributions are the following. According to the taxi GPS historical data, our paper proposes a taxi-hunting system of data processing, i.e., Taxi-RS, in the environment of big data. Taxi-RS includes two phases: offline processing and fast online processing. The first phase includes offline data preprocessing and an offline graph model constructor. First, we calculated eigenvalues between the image data from the GPS data. We divided the big data file into seven parts, according to the difference degree between the eigenvalues. Then, we scan hot points on compressed data to identify the preference trajectories. According to the preference trajectories, we can construct the offline graph trajectory model. When inquiring online, we can search quickly in the trajectory graph to calculate query results by using the probability model based on the query input time and location. Finally, we output query results.

There are several aspects in extending this current work further. In real life, a city's taxi system is a dynamic system changing gradually with time passing. In addition, the system is not closed and dramatically affected by external factors (such as road construction, traffic control, etc.). Therefore, one-month GPS data are not enough to support the continuous outputs. After a long period of time, the old GPS data will not match the real-time taxi system; hence, it is necessary to timely update the model based on the latest GPS data. To deal with this problem, we have the following suggestions.

1) We could check the time validity of the data set and discard invalid data that are past their expiry date. Therefore, we need to cover the offline model with the latest data. This approach is relatively simple, but it is possible to abandon some of the old data, which are still valuable and may result in some error.
2) On this basis, we could replace the old data more gradually. For example, we can iteratively update old data by $data\ in\ the\ model = the\ latest\ data \times 0.8 + the\ old\ data \times 0.2$. The advantage is that we could retain the reference of the old data during discard.

However, if we use the two methods mentioned, we will take a long time when updating the offline model every time. We could optimize the updating time by using the dynamic graph or dynamic programming algorithm, so that we can construct the offline model to update new data.

Acknowledgment

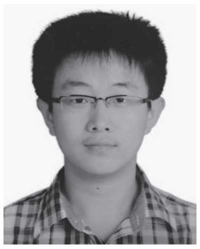
The authors would like to thank the anonymous referees for their helpful comments and suggestions, which led to this revised version.

References

[1] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *Proc. ACM SIGKDD*, 2011, pp. 316–324.
[2] J. Yuan, Y. Zheng, L. Zhang, X. Xie, and G. Sun, "Where to find my next passenger?" in *Proc. UbiComp*, 2011, pp. 109–118.
[3] Q. He, K. Chang, and E.-P. Lim, "Analyzing feature trajectories for event detection," in *Proc. ACM SIGIR*, 2007, pp. 207–214.
[4] S. Liu, C. Liu, Q. Luo, L. Ni, and R. Krishnan, "Calibrating large scale vehicle trajectory data," in *Proc. IEEE MDM*, 2012, pp. 222–231.
[5] J. Yuan *et al.*, "T-Drive: Driving directions based on taxi trajectories," in *Proc. 18th SIGSPATIAL Int. Conf. GIS*, 2010, pp. 99–108.
[6] J. Yuan, Y. Zheng, X. Xie, and G. Z. Sun, "T-Drive: Enhancing driving directions with taxi drivers intelligence," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 1, pp. 220–232, Jan. 2013.
[7] Y. Zheng, Y. Chen, Q. Li, X. Xie, and W. Ma, "Understanding transportation modes based on GPS data for Web applications," *ACM Trans. Web*, vol. 4, no. 1, pp. 1–36, Jan. 2010.
[8] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W. Ma, "Understanding mobility based on GPS data," in *Proc. Ubicom*, 2008, pp. 312–321.
[9] Y. Zheng, L. Liu, L. Wang, and X. Xie, "Learning transportation mode from raw GPS data for geographic applications on the web," in *Proc. WWW*, 2008, pp. 247–256.
[10] J. Pei *et al.*, "PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth," in *Proc. 17th Int. Conf. Data Eng.*, 2001, pp. 215–224.
[11] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutirrez, "Recommender systems survey," *Knowl.-Based Syst.*, vol. 46, pp. 109–132, Jul. 2013.
[12] Y. Zheng and X. Zhou, Eds., *Computing With Spatial Trajectories*. New York, NY, USA: Springer-Verlag, 2011.
[13] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. ACM SIGMOD Int. Conf. Oil Manag. Data*, NewYork, NY, USA, 1984, pp. 47–57.
[14] H. Samet, "The quadtree and related hierarchical data structures," *ACM Comput. Surv.*, vol. 16, no. 2, pp. 187–260, Jun. 1984.
[15] C. Shahabi, M. R. Kolahdouzan, and M. Sharifzadeh, "A road network embedding technique for k-nearest neighbor search in moving object databases," in *Proc. ACM GIS*, 2002, pp. 94–100.
[16] H. Hu, D. L. Lee, and V. C. S. Lee, "Distance indexing on road networks," in *Proc. 32nd Int. Conf. VLDB Endowment*, 2006, pp. 894–905.
[17] G. Z. Sun, Z. Zhang, and J. Yuan, "An efficient pre-computation technique for approximation KNN search in road networks," in *Proc. Int. Workshop Location Based Social Netw.*, Seatle, WA, USA, 2009, pp. 41–44.
[18] J. Greenfeld, "Matching GPS observations to locations on a digital map," in *Proc. 81th Annu. Meet. Transp. Res. Board*, 2002, vol. 1, pp. 1–13.
[19] A. Civilis, C. Jensen, J. Nenortaite, and S. Pakalnis, "Efficient tracking of moving objects with precision guarantees," in *Proc. 1st Annu. Int. Conf. MOBIQUITOUS*, 2004, pp. 164–173, IEEE.
[20] A. Cvilis, C. Jensen, and S. Pakalnis, "Techniques for efficient road-network-based tracking of moving objects," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 5, pp. 698–712, May 2005.
[21] Y. Ding, S. Liu, J. Pu, and L. M. Ni, "HUNTS: A trajectory recommendation system for effective and efficient hunting of taxi passengers," in *Proc. IEEE 14th Int. Conf. Mobile Data Manag.*, 2013, pp. 107–116.

Xiujuan Xu received the Ph.D. degree in computer science and technology from Jilin University, Changchun, China, in 2008.

She is an Associate Professor with the Software School, Dalian University of Technology, Dalian, China. Her research interests include applications of data mining, recommend systems, and social network.

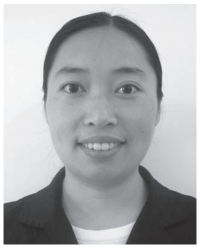


Jianyu Zhou is working toward the B.S. degree from Dalian University of Technology, Dalian, China.

He is a Senior with Dalian University of Technology and is recommended for admission for the Ph.D. degree at Tsinghua University, Beijing, China. His research interests include applications of big data and taxi trajectory flow.

Yu Liu received the Ph.D. degree from Xi'an Jiaotong University, Xi'an, China, in 2008.

He is a Professor with the Software School, Dalian University of Technology, Dalian, China. His research interests include swarm intelligence, evolutionary computation, computational intelligence, and web data mining.



Zhenzhen Xu received the Ph.D. degree from Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang, China, in 2009.

She is an Associate Professor with the Software School, Dalian University of Technology, Dalian, China. Her research interests include scheduling algorithms and intelligent optimization algorithms.

Xiaowei Zhao received the M.S. degree from Queen Mary and Westfield College, University of London, London, U.K., in 2005. She is currently working toward the doctorate degree with the Faculty of Management and Economics, Dalian University of Technology, Dalian, China.

She is currently an Associate Professor with the Software School, Dalian University of Technology. Her research interests involve complex systems, recommender systems, and gaming theory.