TripCube: A Trip-oriented vehicle trajectory data indexing structure

Tao Xu, Xihui Zhang, Christophe Claramunt, Li Xiang

TripCube: A Trip-oriented vehicle trajectory data indexing structure

Tao Xu [a,b], Xihui Zhang [c], Christophe Claramunt [d], Xiang Li [a,*]

[a] Key Laboratory of Geographical Information Science, Ministry of Education, East China Normal University, 500 Dongchuan Road, Shanghai 200241, China
[b] Key Laboratory of Analysis and Processing on Big Data of Henan Province, Henan University, 85 Minglun Road, Kaifeng, Henan 475001, China
[c] Department of Computer Science & Information Systems, College of Business University of North Alabama, Florence, Alabama, United States
[d] Naval Academy Research Institute, Brest, France

ARTICLE INFO

Article history:
Received 19 November 2016
Received in revised form 10 August 2017
Accepted 12 August 2017
Available online 9 September 2017

Keywords:
Spatio-temporal data management
Indexing structure
Vehicle trip
Vehicle trajectory data

ABSTRACT

With the dramatic development of location-based services, a large amount of vehicle trajectory data are available and applied to different areas, while there are still many research challenges left, one of them being data access issues. Most of existing tree-shape indexing schemes cannot facilitate maintenance and management of very large vehicle trajectory data. How to retrieve vehicle trajectory information efficiently requires more efforts. Accordingly, this paper presents a trip-oriented data indexing scheme, named *TripCube*, for massive vehicle trajectory data. Its principle is to represent vehicle trajectory data as trip information records and develop a three-dimensional cube-shape indexing structure to achieve trip-oriented trajectory data retrieval. In particular, the approach is implemented and applied to vehicle trajectory data in the city of Shanghai including >100 million locational records per day collected from about 13,000 taxis. *TripCube* is compared to two existing trajectory data indexing structures in our experiments, and the result exhibits that *TripCube* outperforms others.

1. Introduction

With the dramatic development of location-based services, vast amount of vehicle trajectory data can be easily gathered with GPS receivers equipped on vehicles. The abundance of trajectory data presents a valuable opportunity for scholars to discover previously unknown but potentially valuable information about vehicle movements and traffic situations, such as developing trajectory data mining methods (Dodge, Weibel, and Forootan, 2009; Izakian, Mesgari, and Abraham, 2016; Liu and Karimi, 2006; Pfoser and Theodoridis, 2003; Zhou et al., 2015), inferring residents travel characteristics and patterns (Hu, Miller, and Li, 2014; Kang, Liu, and Wu, 2015; Liu, Wang, Xiao, and Gao, 2012; Torrens et al., 2012), discovering spatio-temporal features of traffic flow (Ge et al., 2010; Liu and Ban, 2012; Wang, Wang, Song, and Raghavan, 2017; Wei, Zheng, and Peng, 2012; Zheng, Liu, Yuan, and Xie, 2011), and predicting travel time (Chen and Rakha, 2014; Jiang and Li, 2013). In the meantime, such data analysis avenues bring novel challenges, a crucial one being the development of appropriate solutions for the most efficient management of vehicle trajectory data (Jiang and Li, 2013; Kwan, 2016). Efficient data structures and algorithms need to be tailored for vehicle trajectory data (Jiang and Yao, 2006; Katal, Wazid, and Goudar, 2013).

As the moving of vehicles is usually constrained by road network, the spatial distribution of vehicle trajectory data is linear along with road segments. With predefined spatio-temporal granularity, original trajectory data can be divided into trajectory segments. Many indexing structures based on trajectory segments have been proposed. Most of them employ R-tree-based indexing structure and set some spatial attributes (e.g., longitude and latitude) as keywords to index vehicle trajectory points or segments. The principle of R-tree (Guttman, 1984) is to group nearby objects based on their spatial locations and to represent them with their Minimum Bounding Rectangles (MBRs) stored in tree nodes. However, in urban road network, long-term and massive vehicle trajectory data must generate a great number of overlapping or redundant MBRs and the corresponding indexing structure must be a bloated multilevel R-tree. Such a R-tree indexing structure is difficult to maintain, which dramatically increases operational cost and reduces query efficiency.

Different from tree-shape indexing structure, the size of cube-shape indexing structure is controllable by predefining spatio-temporal dimensions, whereas the depth and breadth of tree-shape indexing structures extend continuously. Most of cube-shape indexing approaches divide trajectory data into segments by fixed spatial granularity or fixed distance, and aggregate trajectory segments with the same spatio-temporal features into cells of cube. Trajectory segment retrieval can be made from the cube with given query conditions. However, such trajectory segmentation with fixed granularity is only beneficial to trajectory retrieval under given granularity. Moreover, it splits the semantic integrity of vehicle trajectory that brings obstacle to trajectory retrieval by given origin and destination.

In this paper, we attempt to explore a flexible indexing scheme for vehicle trajectory data with cube structure. Generally, a vehicle trajectory indicates vehicle driving process and can be divided by meaningful origin-destination pairs (e.g., the pick-up and drop-off points of taxicab, the entrances and exits of express road system, etc.) to represent vehicle mobility. Therefore, instead of fixed spatial granularity, we define vehicle

* Corresponding author.
E-mail addresses: txucn@hotmail.com (T. Xu), xli@geo.ecnu.edu.cn (X. Li).

http://dx.doi.org/10.1016/j.compenvurbsys.2017.08.005

trip to represent a travel case with an origin-destination pair and to make flexible trajectory segmentation. Moreover, we design a trip-oriented cube structure with three dimensions, namely origin, destination, and departure time, to index vehicle trajectory data and achieve efficient trip information retrieval.

Accordingly, this research introduces a vehicle trajectory data indexing scheme for trip information retrieval. A trip-oriented vehicle trajectory data indexing structure, named *TripCube*, is developed and implemented. *TripCube* consists of a three dimensional index cube and a set of trip information record. It is designed to evaluate queries on trip information between any pair of origin and destination locations at any time. The approach is applied to a large taxi trajectory data set in the city of Shanghai, and performance tables and figures as well as comparison to two existing methods are presented. Our contributions can be summarised as follows:

- *TripCube* is developed to address the challenge of organizing and indexing massive vehicle trajectory data.
- Different from most existing tree-shape indexing structures, *TripCube* is based on a cube-shape indexing structure and allocates indexing storage space before inserting new entries in order to facilitate maintaining complicated indexing structure when data volume dramatically increases.
- *TripCube* is especially applicable to long-term and massive vehicle trajectory data confined to a given road network.
- Compared with two existing trajectory-segment-based indexing structures, *TripCube* exhibits more excellent query efficiency.

The rest of this paper is organized as follows. The next section gives a detailed review of related works. *TripCube* is developed in Section 3 and is validated in Section 4 through a series of experiments. The last section concludes the paper.

2. Related works

In past few years, many research works for vehicle trajectory data indexing have been made. R-tree (Guttman, 1984) is the most common one. R-Tree and its variants, e.g., R + tree (Sellis, Roussopoulos, and Faloutsos, 1987), R*-tree (Beckmann, 1990), X-tree (Berchtold, Keim, and Kriegel, 1996), RT-tree (Xu, Han, and Lu, 1990), HR-tree (Nanopoulos, Theodoridis, and Manolopoulos, 2006), B dual-Tree (Yiu, Tao, and Mamoulis, 2008),R k-d Tree (Anandhakumar, Priyadarshini, Monisha, Sugirtha, and Raghavan, 2010),Vor-Tree (Sharifzadeh and Shahabi, 2010),HTPR*-Tree (Fang, Cao, Wang, Peng, and Song, 2012),and HBSTR-Tree (Ke et al., 2014), are often retained as the indexing structure of vehicle trajectory data. They use the Minimum Bounding Rectangle (MBR) to cluster spatial objects and create the height-balanced tree to index spatial objects. The spatial range retrieval based on MBR can be made with logarithmic query performance. However, maintaining such indexing structure is considerably complex, and insertion, update, and deletion operations, which might change the correlation and the volume of tree nodes, are time-consuming compared to data retrieval queries (Guttman, 1984). Although vehicle trajectory data are usually managed as a sort of "append-only record set" with only insertion operations but no update or deletion operations, the rapid growth of data volume still leads to a significant challenge for tree index maintenance, which not only expands the index space, but also lengthens the index query execution time.

As such, some existing solutions about trajectory segmentation have already tried to deal with the data volume challenges of vehicle trajectory data to some extent. Brakatsoulas, Pfoser, and Tryfona (2004) proposes a trajectory data management scheme to model, store, and mine moving object database. In this schema, vehicle trajectory is divided into segments corresponding to the edge of road network. Leonardi et al., (2014) deigns a spatio-temporal hierarchies framework, and decomposes a complete trajectory into trajectory segments with respect to given distance and time. In addition, trajectory segment can be delineated according to spatial region and time (Leonardi, Marketos, Frentzos, and Giatrakos, 2010; Masciari, 2012; Masciari, 2015; Pelekis and Theodoridis, 2014; Surya Prakash, 2014). With predefined spatio-temporal granularities, these approaches split trajectory data as trajectory segments with fixed distance or fixed range and index trajectory segments with tree-shape indexing structure (B-tree, or R-tree). This can reduce the complexity of tree indexing structure to some extent and facilitate trajectory segment retrieval under the defined spatio-temporal granularity. However, the fixed trajectory segmentation may break the semantic integrity of vehicle trajectory and cause data errors and losses inevitably (Masciari, 2015). The challenges caused by the large vehicle trajectory data have not been solved yet.

As the description of a travel case, vehicle trip from an origin to a destination contains complete semantic information. Using vehicle trip information from vehicle trajectory data, many research works have been made to predict travel time (Jiang and Li, 2013; Xu, Li, and Claramunt, 2017; Xu, Xu, Hu, and Li, 2017), analyze driving behavior (Ren, Tao, and Xiang, 2014), and reveal traffic patterns (Dai, Yang, Guo, and Ding, 2015; Izakian et al., 2016; Liu and Ban, 2012), etc. How to retrieval vehicle trip information efficiently has become a critical issue. Therefore, it is very valuable to trajectory segmentation based on vehicle trip for the management and application of vehicle trajectory data.

Since vehicle trip can be easily identified by origin, destination, and departure time, a controllable indexing space can be designed with spatial features (origin and destination points) and temporal features (departure time). A cube structure can directly map the relationship of spatio-temporal data, and some research results (Cao et al., 2015; Lins, Klosowski, and Scheidegger, 2013; Pelekis and Theodoridis, 2014; Surya Prakash, 2014) about cube structures for the management of spatio-temporal data have been presented. Based on them, we attempt to develop a novel trip-oriented trajectory data indexing scheme, named *TripCube*, to support the maintenance and retrieval of vehicle trajectory data. The *TripCube* consists of a three-dimensional indexing cube and a set of vehicle trip information records. Vehicle trajectory data are organized as vehicle trip information records and indexed by the cube structure on its attributes (i.e., origin, destination, and departure time).

3. Methodology

3.1. Principles

TripCube is a two-level indexing structure, i.e., a trip level and a cube level. At the trip level, we define trip information record as *vts* to represent trip information from raw vehicle trajectory data, where *vts* is a composite structure that includes vehicle ID, origin, destination, departure time, travel route, travel time, and vehicle trajectory, etc. At the cube level, a three dimensional index cube, consisting of one origin dimension, one destination dimension, and one time dimension, is generated to manage *vts* with specific origin, destination, and departure time. The origin and destination of a trip represent the travel starting and ending positions, which can be determined by application purposes, e.g., the pick-up/drop-off points of taxicab, the entrances and exits of express road system, and commuter destinations, etc. *TripCube* converts vehicle trajectory data to *vts* and indexes it by origin, destination, and departure time. By this means, *TripCube* is not a typical spatial indexing structure. Instead of performing a spatial search for vehicle trajectory points, *TripCube* retrieves the set of vehicle trips with explicitly encoded spatial and temporal query conditions (e.g., origin, destination, and departure time). The structure of *TripCube*, its initialization procedure, and retrieval algorithms are presented in the remaining part of this section.

3.2. Vehicle trip structure

To extract trip information from raw vehicle trajectory data, we define *vehicle trip structure* to represent trip information and corresponding locational sample points.

Assume that raw vehicle trajectory data consist of massive sample points and each sample point contains vehicle ID, timestamp, longitude, latitude, speed, and matched location in road network. Raw vehicle trajectory data are stored as a file on disk, and all sample points are sorted by vehicle ID and timestamp. Let $traj(s, num)$ defined by Eq. (1) represent a trajectory of a vehicle with length num (the number of sample points) from ith to $(s + num\text{-}1)$th sample points in a raw data file, where $\mathbb{P}$ is the raw vehicle trajectory data set, and p_i is a sample point from $\mathbb{P}$.

$$traj(s, num) = \bigcup_{i=s}^{s+num-1} p_i, p_i \in \mathbb{P} \tag{1}$$

Let *vehicle trip structure* (in short, *vts*), defined as Eq. (2), represent a vehicle traveling case from an origin to a destination, where o is the origin of the trip, d is the destination of the trip, t is departure time, tt is travel time, r is travel route, and s and num are position parameters of vehicle trajectory data in $\mathbb{P}$ defined by Eq. (1).

$$vts = \{vehicle\ ID, o, d, t, tt, r, s, num\} \tag{2}$$

vts is a composite structure with rich trip information and is a lightweight structure since *vts* does not contain massive sample points, while the position parameters, s and num, indicate corresponding trajectory data. Moreover, as shown in Eq. (3), let $\mathbb{VT}$ be the complete set of *vts* to represent all travel cases from $\mathbb{P}$. Therefore, raw trajectory data are organized as trip information set, and detailed trip information and corresponding trajectory data are retrieved by $\mathbb{VT}$.

$$\mathbb{VT} = \cup vts_i \tag{3}$$

The creation of $\mathbb{VT}$ is shown in Algorithm 1. Its input includes the raw vehicle trajectory data set ($\mathbb{P}$) and the predefined origin and destination set (*ODs*), and the output is the complete set $\mathbb{VT}$. The time complexity of Algorithm 1 is $O(n)$, and its running time is directly related to the volume of $\mathbb{P}$. Algorithm 1 traverses $\mathbb{P}$ to search every time-continuous or position-continuous trajectory series (*traj*), creates *vts* and inserts it in $\mathbb{VT}$. The key step is to determine whether the sample point p_i is in the current trajectory series (*traj*) by the continuity of time or position between p_i and the last sample point of *traj* (line 5). If two points are continuous, the length of *traj* plus 1 in line 6, and otherwise, *vts* is generated from *traj* and added in $\mathbb{VT}$ in line 9 to 10. Origin (o) and destination (d) of *vts* can be matched by the predefined origin and destination set (*ODs*). Departure time (t) is the timestamp of the fist sample point of *traj*. Travel time (tt) is the difference between the timestamp of the first sample point and the last sample point of *traj*. Travel route (r) is the sorted set of matched road segments of *traj*.

Algorithm 1. Create$\mathbb{VT}$ ($\mathbb{P}$, *ODs*).

```
1.  traj.s = 0; traj.num = 0;
2.  VT = ∅;
3.  for each p_i ∈ P
4.  {
5.    if (p_i ∈ traj)
6.    {  traj.num++;  }
7.    else
8.    {
9.      generate vts from traj;
10.     VT = VT ∪ vts;
11.     traj.s = i + 1;  traj.num = 0;
12.   }
13. }
14. return VT;
```

With *vts*, raw vehicle trajectory data $\mathbb{P}$ are organized as trip cases from origin node to destination node in road network.

3.3. Three dimensional index cube

In order to achieve a fast access to *vts* from the *vts* set ($\mathbb{VT}$), trip index is created on o, d, and t of *vts*. Then, using the quadruple group (o_i, d_j, t_s, t_e), trip information can be retrieved with the i^{th} origin, the j^{th} destination, and the departure time range $[t_s, t_e)$. Accordingly, we define a three dimensional index cube structure to index $\mathbb{VT}$ and retrieve trip information with query conditions (o_i, d_j, t_s, t_e), i.e. origin, destination, and departure time range.

As shown in Eq. (4), $\mathbb{C}[\mathbb{O}][\mathbb{D}][\mathbb{T}]$ denotes a three dimensional cube structure consisting of one origin dimension ($\mathbb{O}$), one destination dimension ($\mathbb{D}$), and one time dimension ($\mathbb{T}$), where $\mathbb{O}.count$ and $\mathbb{D}.count$ are the number of origin nodes and destination nodes, respectively, and $\mathbb{T}.count$ is determined by the time granularity and the time span of raw vehicle trajectory data. c_{ijk} is defined as a cell with the $\mathbb{C}$ coordinates of i, j, k. Since the size of the cube $\mathbb{C}$ is the product of $\mathbb{O}.count$, $\mathbb{D}.count$, and $\mathbb{T}.count$, the volume of $\mathbb{C}$ is affected by the size of $\mathbb{O}$, $\mathbb{D}$, and $\mathbb{T}$.

$$\mathbb{C}[\mathbb{O}][\mathbb{D}][\mathbb{T}] = \{c_{ijk} \mid 0 \le i \le \mathbb{O}.count, 0 \le i \le \mathbb{D}.count, 0 \le i \le \mathbb{T}.count, \} \tag{4}$$

As shown in Eq. (5), each cell (c_{ijk}) of $\mathbb{C}$ contains some pointers (g_w^{ijk}) to *vts*, where g_w^{ijk} denotes the wth pointer in c_{ijk}. The asterisk (*) symbol is a pointer operator, and vts_v is the vth *vts* in $\mathbb{VT}$. It means that, g_w^{ijk} points to vts_v with origin node (o_i), destination node (d_j), and departure time range $[t_k, t_{k+1})$

$$c_{ijk} = \left\{ g_w^{ijk} \mid w \ge 0, *g_w^{ijk} = vts_v \right\} \tag{5}$$

The creation of three dimensional index cube $\mathbb{C}$ is given in Algorithm 2. Its inputs include the *vts* set $\mathbb{VT}$, the predefined origin and destination set (*ODs*), and time granularity t^g, and its output is the three dimensional index cube $\mathbb{C}$. In line 1 to 3, Algorithm 2 generates origin dimension ($\mathbb{O}$) and destination dimension ($\mathbb{D}$) by *ODs*; set the time dimension $\mathbb{T}$ with t^g and the time span of $\mathbb{P}$; and then, create a blank cube structure $\mathbb{C}$. In line 4 to 11, for each *vts*, get the address pointer (g) with the fetch (&) symbol, the origin (o_i), the destination (d_j), and the departure time (t_k). Then, insert g into the specific cube cell c_{ijk}. The time complexity of Algorithm 2 is $O(n^3)$, and its running time is directly related to the size of $\mathbb{C}$.

Algorithm 2. Create$\mathbb{C}$ ($\mathbb{VT}$, *ODs*, t^g).

```
1.  generate O, D by ODs.
2.  generate time dimension T with t^g and the time span of P.
3.  generate a blank C with O, D, and T;
4.  for each vts ∈ VT
5.  {
6.    g = &vts;
7.    o_i = vts.o;
8.    d_j = vts.d;
9.    t_k = vts.t;
10.   insert g into c_ijk of C ;
11. }
12. return C;
```

The origin dimension ($\mathbb{O}$) and the destination dimension ($\mathbb{D}$) of $\mathbb{C}$ are from the predefined origin-destination pairs (*ODs*). The time dimension $\mathbb{T}$ is determined by t^g. With the fixed time span of $\mathbb{P}$, t^g is related to the capacity of elements of $\mathbb{C}$, and negatively related to the number of elements.

Using the complete index cube $\mathbb{C}$, we can directly access specific trip information with the query conditions, i.e. the ith entrance (o_i), the jth exit (d_j), and departure time range $[t_s, t_e)$.

3.4. TripCube structure

Based on the above mentioned *vts* set ($\mathbb{VT}$) and three dimensional index cube ($\mathbb{C}$), we propose a vehicle trajectory data indexing structure,

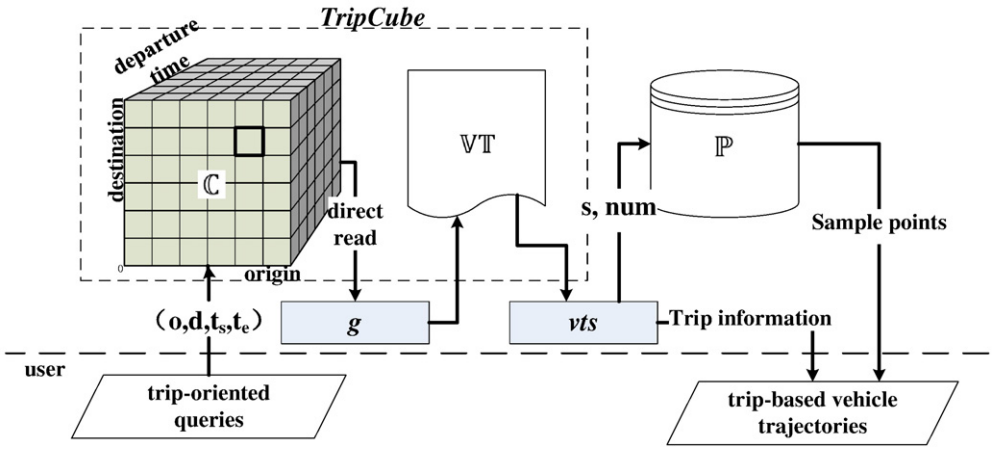

Fig. 1. Schematic overview of *TripCube*.

named *TripCube*. As shown in Eq. (6), *TripCube* denotes the proposed indexing structure including a *vts* set ($\mathbb{VT}$) and a cube structure ($\mathbb{C}$).

$$TripCube = <\mathbb{VT}, \mathbb{C}> \tag{6}$$

TripCube is a trip-oriented two level indexing structure including a trip level and a cube level. The trip level is a *vts* set ($\mathbb{VT}$), and the cube level is a three dimensional cube structure ($\mathbb{C}$) which consists of one origin dimension ($\mathbb{O}$), one destination dimension ($\mathbb{D}$), and one time dimension ($\mathbb{T}$). By recognizing vehicle trajectory series from origin to destination in road network, trip information are organized by *vts* and raw vehicle trajectory data set ($\mathbb{P}$) can be indexed by the *vts* set ($\mathbb{VT}$). Moreover, *TripCube* employs index pointer (g_w) to mark the location of *vts* in $\mathbb{VT}$ and these pointers are stored in specific cells of $\mathbb{C}$ with constraint conditions of origin, destination, and departure time.

By integrating Algorithms 1 and 2, *TripCube* initialization, as shown in Algorithm 3, creates the *vts* set ($\mathbb{VT}$) and the cube indexing structure ($\mathbb{C}$), and inserts all *vts* and its pointers (g_w) into them, respectively. The inputs of Algorithm 3 are the raw vehicle trajectory data set ($\mathbb{P}$), the predefined origin and destination set (*ODs*), and the time granularity (t^g). The output is the complete *TripCube*. Inheriting from Algorithm 2, the time complexity of Algorithm 3 is also $O(n^3)$, and its running time is directly related to the size of $\mathbb{C}$.

Algorithm 3. *TripCube* initialization ($\mathbb{P}$, *ODs*, t^g).

1. $\mathbb{VT}$= Create$\mathbb{VT}$ ($\mathbb{P}$, *ODs*); //Algorithm 1.
2. $\mathbb{C}$= Create$\mathbb{C}$ ($\mathbb{VT}$, *ODs*, t^g); //Algorithm 2.
3. *TripCube* = $< \mathbb{C}, \mathbb{VT} >$;
4. return *TripCube*;

In addition, *TripCube* has good scalability for the continuously increasing raw vehicle trajectory data. Theoretically, the structure of *TripCube* is extensible, and its volume is only limited by memory capacity. Therefore, for trajectory data newly added in $\mathbb{P}$, $\mathbb{C}$ needs to be expanded to create more cells, and new *vts* and its pointer *g* are generated to update $\mathbb{VT}$ and $\mathbb{C}$.

To create *TripCube*, all vehicle trip information and trip index pointers are sorted in *TripCube* with the form of *vts* and pointer (g_w), and *TripCube* is ready for retrieval. A schematic overview of *TripCube* is shown in Fig. 1. The retrieval process is divided into three steps. First, based on query conditions (o_i, d_j, t_s, t_e), read pointer g_w in c_{ijk} from cube structure $\mathbb{C}$ directly; then, fetch *vts* from $\mathbb{VT}$; finally, use *s* and *num* to extract trajectory sample points from $\mathbb{P}$, and then integrate trip information and sample points to answer trip-oriented vehicle trajectory retrieval.

Based on Fig. 1, trip-oriented queries can be answered with the algorithm of trip retrieval shown in Algorithm 4. The inputs of Algorithm 4 are retrieval conditions, i.e. the *i*th entrance (o_i), the *j*th exit (d_j), and departure time range [t_s, t_e). The outputs are the retrieval results, i.e. trip information set, named *TRIP*, and corresponding vehicle trajectory series set, named *TRAJ*. First, generate retrieved cells set *Cs* from *ts*, *te*,and t^g (line 1). Then, traverse every pointer g_w of every *cell* of *Cs* to extract trip information *TRIP* from $\mathbb{VT}$ and extract sample points *TRAJ* from $\mathbb{P}$ (line 2 to 11).

Algorithm 4. TripCube retrieval (o_i, d_j, t_s, t_e).

1. generate the retrieved set, named *Cs*, of cells of *TripCube* from t_s, t_e,and t^g;
2. for each *cell* ∈ *Cs*
3. {
4. for each g_w ∈ *cell*
5. {
6. $vts = * g_w$;
7. $TRIP = TRIP \cup vts$;
8. $traj = \cup_{l=vts.s}^{vts.s+vts.num-1} p_l$;
9. $TRAJ = TRAJ \cup traj$;
10. }
11. }
12. return *TRIP* and *TRAJ*;

Theoretically, as long as t^g is short enough, the volume of c_{ijk} will be 1 and the time complexity of Algorithm 4 will be $O(n \times m)$. *n* is the number of cells of *Cs*, and *m* is the number of pointer of cell. It indicates that the retrieval performance of *TripCube* is directly related to the span of query

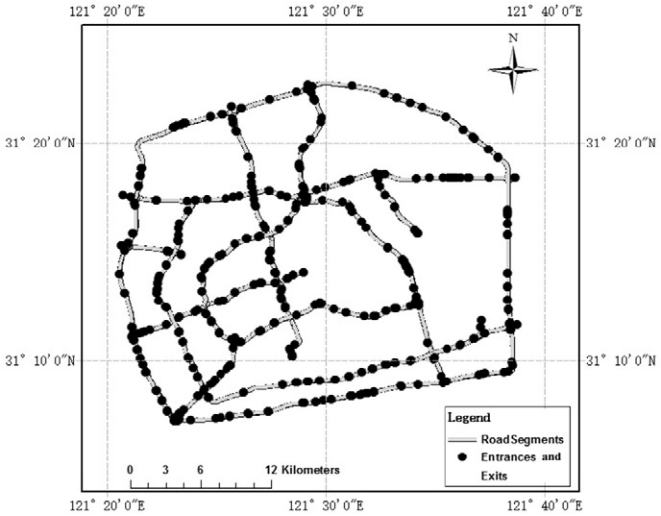

Fig. 2. The express road system of central Shanghai.

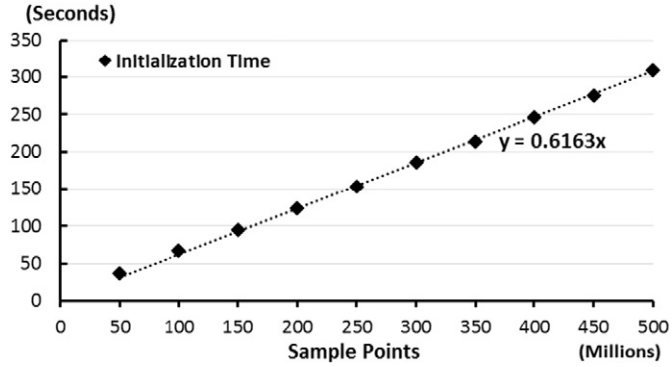

Fig. 3. Time for generating *TripCube*.

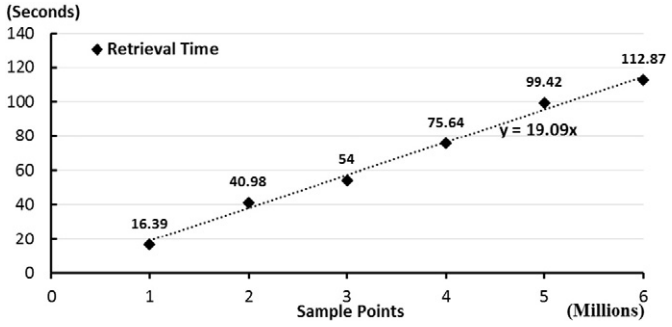

Fig. 5. Retrieval time of *TripCube*.

time and the number of cells of *TripCube*. However, with the space complexity $O(n^3)$, such t^g is unacceptable and unfeasible because of a dramatic increase in the volume of $\mathbb{C}$ and limited memory usage. Moreover, same to Algorithm 2, the space complexity of Algorithm 4 is $O(n^3)$.

The main idea of the *TripCube* is to generate and maintain the three-dimensional index cube ($\mathbb{C}$) and the *vts* set ($\mathbb{VT}$) to realize the storage and retrieval of trip information from raw vehicle trajectory data. Moreover, *TripCube* is a flexible and extensible data indexing scheme fitting for massive trajectory data, while its volume is limited by memory capacity. Finally, *TripCube* can achieve good retrieval performance because it uses cube structure and *vts* set to read trip information and vehicle trajectory sample points directly without traversing. Moreover, query based on specific vehicles is also supported by *TripCube*. Based on the retrieval mechanism of *TripCube*, vehicle information between a specified OD pair can be retrieved, where vehicles' identifications (e.g., ID) have been included. For example, "how many vehicles passed by this OD pair last month?", "how many times did the vehicle with *vehicle ID* '12345' travel between this OD pair last year?" and so on.

4. Experiments

4.1. Data

The express road system of Shanghai central city, China, provides the spatial layout of experiments. As shown in Fig. 2, the express road system covers about 680 km^2 including 240 entrances and 246 exits. The total length of road segments in the system is about 370 km. We define the entrances and exits as the origins and destinations of vehicle trips and use them to fill the *ODs*.

Vehicle trajectory data are from about 13,000 taxies covering the express road system for one month. In order to extract specific vehicle trajectory data of the express road system, map matching of vehicle trajectory data has been previously performed (Li, Li, Tang, and Xu, X., 2010). Each sample point of vehicle trajectory consists of vehicle ID, timestamp, longitude, latitude, speed, and matched location in the express road system. The average sampling rate of records is 0.1 Hz, and the average number of records is about 14 million per day.

4.2. Implementation

All experiments are conducted with a computer equipped with Intel i5, 3.4 GHz CPU, 4GB RAM, and Windows 10 64-bit operation system.

As mentioned in Section 4.1, $\mathbb{P}$ is a set of locational sample points matched on road segments. It is a continuously growing dataset, with very large amount of taxi trajectory data streaming into it per day. Part of $\mathbb{P}$ (30 days) from April 1, 2015 to April 30, 2015 is used in the following experiments. In addition, to avoid the interference from commercial database inherent indexing structures, $\mathbb{P}$ is stored in a hard drive as text files. The total capacity of $\mathbb{P}$ file is about 12.9 GB, and the number of locational sample points is about 500 million.

To generate *TripCube*, its size is confirmed first. Because of the daily periodicity change of taxi services, one whole day is a proper time granularity for $\mathbb{T}$ of *TripCube*, and $\mathbb{T} = \{t_i | April\ 1, 2015 \le t_i \le April\ 30, 2015,\ 1 \le i \le 30\}$. Moreover, the origins and destinations of vehicle trip are supplied by the entrances and exits of the express road system, where $\mathbb{O} = \{o_i |\ 1 \le i \le 240\}$, $\mathbb{D} = \{d_i |\ 1 \le i \le 246\}$. Therefore, the size of *TripCube* used for the following experiments is about 1.77 million ($30 \times 240 \times 246 = 1771200$).

4.3. Performance of TripCube

Based on the given t^g (one whole day) and the raw vehicle trajectory data set $\mathbb{P}$, Algorithm 3 (see Section 4.4) generates $\mathbb{VT}$ from $\mathbb{P}$, creates a blank cube structure $\mathbb{C}$, and stores each index pointer g_w into $\mathbb{C}$. Fig. 3 shows the time consumption of *TripCube* initialization. The vertical axis represents running time (unit: seconds), while the horizontal axis

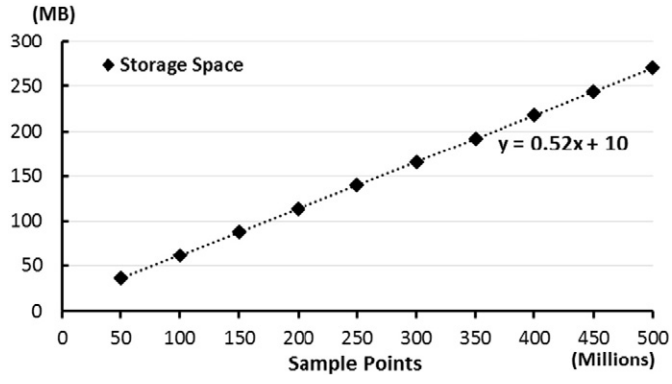

Fig. 4. Storage space of *TripCube*.

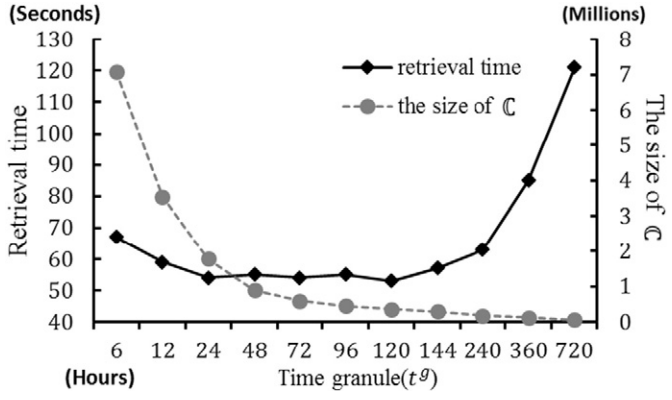

Fig. 6. The relationship between time granularity (t^g) and retrieval time.

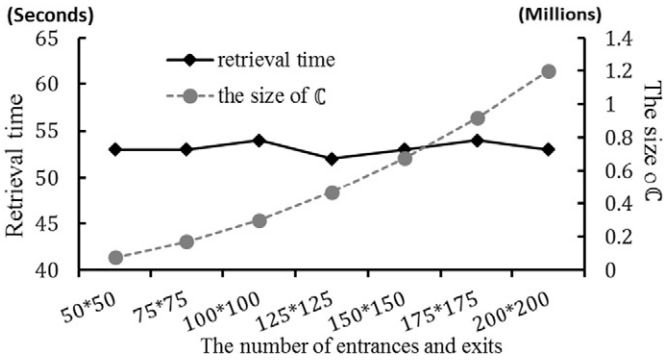

Fig. 7. The relationship between the number of entrances and exits and retrieval time.

represents the number of vehicle trajectory sample points (unit: millions). Black points are the time of *TripCube* initialization. It is observed that the trend of initialization time is linear, and the formula of trend line is $y = 0.6163x$, which indicates the time of *TripCube* initialization is directly related to the volume of $\mathbb{P}$ and about 0.6163 s per million sample points.

Fig. 4 shows the storage space of *TripCube* with different data volumes. It takes about 0.52 MB storage space for each million sample points, and the blank cube structure ($\mathbb{C}$) occupies about 10 MB storage space. When the number of sample points is about 500 million, the number of vehicle trips in $\mathbb{VT}$ is 2,848,289, and the number of cells in the cube structure ($\mathbb{C}$) is about 1,771,200.

With query conditions (origin, destination, and departure time range), vehicle trip information and trajectory sample points are retrieved by Algorithm 4. The time consumption is given in Fig. 5. Retrieval time is changing in a near linear fashion. It takes about 19.09 s per million sample points.

Since the size of $\mathbb{C}$ in *TripCube* is varied with time granularity (t^g) and the number of entrances and exits, the relationship between the size of $\mathbb{C}$ and retrieval performance is explored.

Fig. 6 illustrates the relationship between time granularity (t^g) (from 6 to 720 h) and retrieval time. Black points represent consuming time for retrieving three millions sample points and corresponding trip information with random retrieval conditions. Gray points represent the size of $\mathbb{C}$ with different time granularity (t^g). The trend of retrieval time can be divided into three stages. The first stage has smaller t^g in 6 or 12 h. With the increase of t^g, the size of $\mathbb{C}$ becomes smaller and the retrieval time is reduced slowly. It means smaller t^g produces larger cube structure ($\mathbb{C}$) and larger retrieved cell set *Cs* (see Algorithm 4) for massive trajectory data retrieval. Therefore, smaller t^g (6 or 12 h) leads to larger overhead for cell operation. The second stage has moderate t^g from 24 to 144 h (6 days) and has the best retrieval performance than other stages. The third stage has larger t^g from 240 (10 days) to 720 h (30 days). The larger t^g reduces the number of cells of $\mathbb{C}$ and increases the volume of cells, which leads to the increase of the time to traverse pointers of *vts* in cells. Especially, when t^g is 720 h, all pointers of *vts* in each entrance exit pair are stored in one cell, and $\mathbb{C}$ degenerates into a two-dimensional structure. Only specific *vts* with departure time range $[t_s, t_e)$ can be obtained by traversing all time pointers with o_i and d_j. The retrieval time is maximum and 121 s. As a result, we suggest that a moderate t^g is necessary for high performance *TripCube*.

Fig. 7 shows the relationship between the number of origin-destination pairs and retrieval time. Similar with Fig. 6, black points represent consuming time for retrieving three millions sample points and corresponding trip information with 24-hour granularity t^g and random retrieval conditions. Gray points represent the size of $\mathbb{C}$ with seven orders of magnitude of entrance-exit pairs. With the increase of the number of entrance-exit pairs, the size of $\mathbb{C}$ becomes larger, and the retrieval time is stationary on about 54 s. It indicates that the spatial distribution of vehicle trajectory data (the number of entrance-exit pairs) does not affect the retrieval performance of *TripCube* if the storage space is sufficient to operate *TripCube*.

4.4. Comparative analysis

In view of the application targets of managing and maintaining vehicle trajectory data, *TripCube* takes vehicle trip as a unit to create *vts*, and uses origin, destination, and departure time to index trip information and vehicle trajectory sample points. To evaluate the performance of *TripCube*, two of trajectory data indexing schemes based on trajectory segments are employed, and comparative analysis between *TripCube* and them are made. They are proposed by Brakatsoulas et al. (2004), named *Brakatsoulas method*, and Leonardi et al. (2014), named *Leonardi method*, respectively.

Brakatsoulas method is a scheme of managing vehicle trajectory data, in which, vehicle trajectory is divided into trajectory segments to store and index. A trajectory segment corresponds to an edge of road network and is defined as Eq. (7), where *ts* represents trajectory segments, *trajectory_id* and *edge_id* are the identification of vehicle trajectory and the edge of road network, *time*1 and *time*2 are the time vehicle enters the edge and leaves the edge. The initialization of *Brakatsoulas method* is to create the set of *ts* from $\mathbb{P}$ and to store it as table using Oracle™ 11 DBMS suite with B-tree index on *edge_id*, *time*1, and *time*2. Trip information retrieval with *Brakatsoulas method* is to traverse the table of *ts* with a given route and departure time range to get a set of *ts*, and then, integrate *ts* with the same *trajectory_id* to generate output results, i.e. vehicle trip information and trajectory data.

$$ts = \{trajectory\_id, edge\_id, time1, time2\} \tag{7}$$

Similar to *Brakatsoulas method*, *Leonardi method* treats vehicle trajectory as a series of trajectory segments to build a framework for modeling trajectory data. The difference is that *Leonardi method* defines a flexible spatio-temporal granularity and hierarchies, and decomposes a complete trajectory into trajectory segments with designated spatio-temporal granularity. Then, set the edge of road network as the spatial granularity and 24 h as the temporal granularity to model $\mathbb{P}$. The initialization of *Leonardi method* is to create four tables using Oracle™ 11 DBMS suite. They are: POINT_T storing raw sample points, SPACIAL_T storing the position of each trajectory segment in POINT_T and indexed by spatial granularity (the edge of the express road system), TEMPORAL _T storing the position of each trajectory segment in POINT_T and indexed by the temporal granularity (24 h), and FACT_T storing trajectory segment information (e.g. start time, end time, speed, etc.). With respect to a given route and departure time range, trip information retrieval with *Leonardi method* is to traverse SPACIAL_T and TEMPORAL_T to get the set of trajectory segments, and then, generates trip information from FACT_T and integrates vehicle trajectory from POINT_T.

Table 1
Features of the three indexing approaches.

Scheme	Indexed object	Indexed domain	Retrieval conditions
TripCube	Trajectory segments of multiple edges	Space and time	Origin, destination, and departure time.
Brakatsoulas method	Trajectory segments of one edge	Space	Route and departure time
Leonardi method	Trajectory segments of one edge	Space and time	Route and departure time

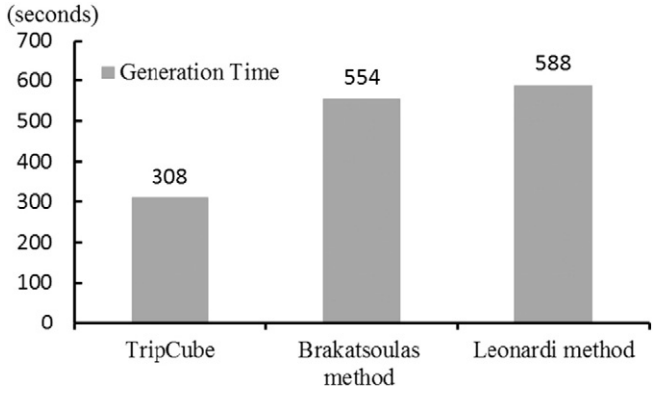

Fig. 8. Generation time comparison.

Features of the three indexing approaches are given in Table 1. Comparisons are made from three perspectives, i.e. generation, storage space, and trip retrieval.

- Generation

Fig. 8 demonstrates the comparison results of generation time. *Brakatsoulas method* needs to split a vehicle trajectory to edge-based trajectory segments with B-tree index in *edge_id*, *time*1, and *time*2 (see Eq. 7). *Leonardi method* uses spatio-temporal granularity to generate trajectory segments from $\mathbb{P}$ and extracts trajectory segment information to fill FACT_T. *TripCube* creates a *vts* set ($\mathbb{VT}$) and a cube structure ($\mathbb{C}$). As indicated in Fig. 8, the generation time of *TripCube* is the shortest. Note that *Brakatsoulas method* and *Leonardi method* are based on Oracle™ 11 DBMS. Additional time consumption caused by Oracle's own mechanics has been eliminated (e.g. time consumption of creating table space, creating data file, etc.). Time values of the two methods shown in Fig. 8 are actual generation time (including CPU time, and I/O time, etc.).

- Storage space

Comparison results of storage space are given in Fig. 9. Note that the storage space involved in the comparison does not include the storage space of raw trajectory sample points. *Brakatsoulas method* and *Leonardi method* store raw vehicle trajectory data as tables in Oracle, and $\mathbb{P}$ of *TripCube* are text files. *Brakatsoulas method* takes the largest space to store B-tree structure on trajectory segments. *Leonardi method* needs 738 MB to create three tables to index trajectory segments. *TripCube*'s storage space is the smallest than others.

- Trip retrieval

As shown in Table 1, retrieval conditions of *TripCube* are the origin and destination of a trip, and departure time, whereas *Brakatsoulas method* and *Leonardi method* are the route of a trip and departure time. As

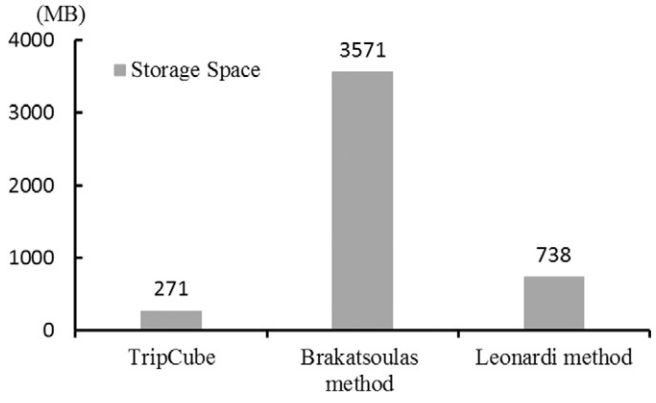

Fig. 9. Storage space comparison.

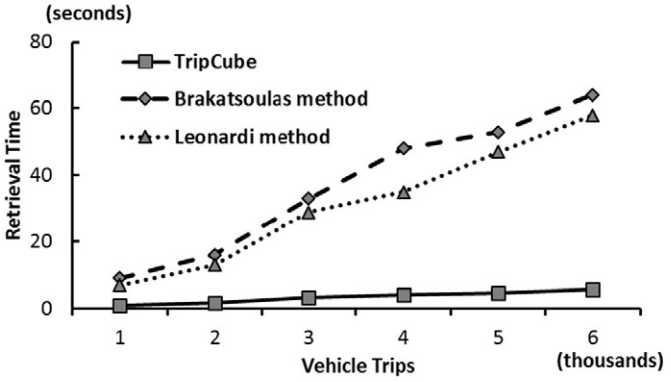

Fig. 10. Trip retrieval comparisons.

indicated in Fig. 10, thousands of vehicle trips are retrieved, *TripCube* is the best than others. Furthermore, the retrieval time of *TripCube* increases much less slowly with the increase of the number of trips than *Brakatsoulas method* and *Leonardi method*. This is because *TripCube* can directly read trip information, whereas *Brakatsoulas method* and *Leonardi method* need to decompose the route to the edges set, search trajectory segments of each edge, and then, compose trajectory segments to generate trip information.

5. Discussion and conclusion

This paper introduces a novel indexing structure, *TripCube*, to maintain and retrieve very large vehicle trajectory data. *TripCube* uses a three-dimensional cube structure and a *vts* set to initialize and index vehicle trajectory data. Efficient vehicle trip retrieval can be achieved by specific query conditions including origin, destination, and departure time range. The proposed approach is applied to a large vehicle trajectory sample points set covering Shanghai express road system. A series of experiments are conducted to evaluate the performance of *TripCube*. Two other existing approaches are implemented and compared to the proposed one. According to the comparison results, *TripCube* outperforms others in terms of generation, storage space, and trip retrieval.

When applying it to vehicle trajectory data, *TripCube* makes it possible to analyze traffic volumes and commuting patterns between any pair of origin and destination in large-scale road networks. Future efforts will be made in the following three directions. First, the spatio-temporal scalability of *TripCube* will be examined with larger datasets. Second, the size of three-dimensional cube cannot grow unlimitedly; as such, a partitioned index may be an alternative. Third, the fundamental structure of *TripCube* may be revisited to explore the possibility of further improving its performance.

Acknowledgments

This research was sponsored by the National Natural Science Foundations of China (Grant No.41271441 and 41771410). The authors also would like to show their gratitude to editor and anonymous reviewers for their constructive comments that greatly improved the manuscript.

References

Anandhakumar, P., Priyadarshini, J., Monisha, C., Sugirtha, K., & Raghavan, S. (2010). Location based hybrid indexing structure — R k-d tree. *Proceedings of the first international conference on integrated intelligent computing*, 140–145.

Beckmann, N. (1990). The R*-tree: An efficient and robust access method for points and rectangles. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, *19*(2), 322–331.

Berchtold, S., Keim, D. A., & Kriegel, H. -P. (1996). The X-tree: An indexing structure for high-dimensional data. *Proceedings of the 22th international conference on very large data bases* (pp. 28–39).

Brakatsoulas, S., Pfoser, D., & Tryfona, N. (2004). Modeling, storing and mining moving object databases. *Proceedings of the 2004 engineering and applications symposium* (pp. 68–77).

Cao, G., Wang, S., Hwang, M., Padmanabhan, A., Zhang, Z., & Soltani, K. (2015). A scalable framework for spatiotemporal analysis of location-based social media data. *Comput Environ Urban Syst*, *51*, 70–82.

Chen, H., & Rakha, H. A. (2014). Real-time travel time prediction using particle filtering with a non-explicit state-transition model (Part 1, Special Issue on Short-term Traffic Flow Forecasting). *Transportation Research Part C: Emerging Technologies*, *43*, 112–126.

Dai, J., Yang, B., Guo, C., & Ding, Z. (2015). Personalized route recommendation using big trajectory data. *IEEE, International Conference on Data Engineering*, 543–554.

Dodge, S., Weibel, R., & Forootan, E. (2009). Revealing the physics of movement: Comparing the similarity of movement characteristics of different types of moving objects. *Comput Environ Urban Syst*, *33*(6), 419–434.

Fang, Y., Cao, J., Wang, J., Peng, Y., & Song, W. (2012). HTPR*-Tree: An efficient index for moving objects to support predictive query and partial history query. *International conference on web-age information management. Vol.7142*. (pp. 26–39).

Ge, Y., Xiong, H., Tuzhilin, A., Xiao, K., Gruteser, M., & Pazzani, M. (2010). An energy-efficient mobile recommender system. *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 899–908).

Guttman, A. (1984). R-trees: A dynamic indexing structure for spatial searching. *ACM SIGMOD Rec*, *14*(2), 47–57.

Hu, Y., Miller, H. J., & Li, X. (2014). Detecting and analyzing mobility hotspots using surface networks. *Trans GIS*, *18*(6), 911–935.

Izakian, Z., Mesgari, M. S., & Abraham, A. (2016). Automated clustering of trajectory data using a particle swarm optimization. *Comput Environ Urban Syst*, *55*, 55–65.

Jiang, B., & Yao, X. (2006). Location-based services and gis in perspective. *Comput Environ Urban Syst*, *30*(6), 712–725.

Jiang, Y., & Li, X. (2013). Travel time prediction based on historical trajectory data. *Ann GIS*, *19*(1), 27–35.

Kang, C., Liu, Y., & Wu, L. (2015). Delineating Intra-urban spatial connectivity patterns by travel-activities: a case study of Beijing, China. *Proceedings of the 23th International Conference on GeoInformatics* (pp. 1–7).

Katal, A., Wazid, M., & Goudar, R. H. (2013). Big data: Issues, challenges, tools and good practices. *Proceedings of the sixth international conference on contemporary computing* (pp. 404–409).

Ke, S., Gong, J., Li, S., Zhu, Q., Liu, X., & Zhang, Y. (2014). A hybrid spatio-temporal data indexing method for trajectory databases. *Sensors*, *14*(7), 12990–13005.

Kwan, M. -P. (2016). Algorithmic geographies: Big data, algorithmic uncertainty, and the production of geographic knowledge. *Annals of the American Association of Geographers*, *106*(2), 274–282.

Leonardi, L., Marketos, G., Frentzos, E., & Giatrakos, N. (2010). T-Warehouse: Visual OLAP analysis on trajectory data. *International conference on data engineering, ICDE 2010, March 1–6, 2010. Vol.41*. (pp. 1141–1144). Long Beach, California, USA: DBLP.

Leonardi, L., Orlando, S., Raffaetà, A., Roncato, A., Silvestri, C., Andrienko, G., et al. (2014). A general framework for trajectory data warehousing and visual olap. *GeoInformatica*, *18*(2), 273–312.

Li, X., Li, X., Tang, D., & Xu, X. (2010). Deriving features of traffic flow around an intersection from trajectories of vehicles. *International conference on Geoinformatics, 2* (pp. 1–5).

Lins, L., Klosowski, J. T., & Scheidegger, C. (2013). Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Transactions on Visualization & Computer Graphics*, *19*(12), 2456–2465.

Liu, X., & Ban, Y. (2012). Uncovering urban mobility patterns with massive floating car data. *Comput Environ Urban Syst*.

Liu, X., & Karimi, H. A. (2006). Location awareness through trajectory prediction. *Comput Environ Urban Syst*, *30*(6), 741–756.

Liu, Y., Wang, F., Xiao, Y., & Gao, S. (2012). Urban land uses and traffic 'source-sink areas': Evidence from GPS-enabled taxi data in Shanghai. *Landsc Urban Plan*, *106*(1), 73–87.

Masciari, E. (2012). Warehousing and querying trajectory data streams with error estimation. *Fifteenth international workshop on data warehousing and Olap* (pp. 113–120). ACM.

Masciari, E. (2015). An end to end framework for building data cubes over trajectory data streams. *J Intell Inf Syst*, *45*(2), 1–34.

Nanopoulos, A., Theodoridis, Y., & Manolopoulos, Y. (2006). Indexed-based density biased sampling for clustering applications. *Data Knowl Eng*, *57*(1), 37–63.

Pelekis, N., & Theodoridis, Y. (2014). Preparing for mobility data exploration. *Mobility data management and exploration* (pp. 121–141). Springer New York.

Pfoser, D., & Theodoridis, Y. (2003). Generating semantics-based trajectories of moving objects. *Comput Environ Urban Syst*, *27*(3), 243–263.

Ren, H., Tao, X. U., & Xiang, L. I. (2014). Driving behavior analysis based on trajectory data collected with vehicle-mounted gps receivers. *Wuhan Daxue Xuebao*, *39*(6), 739–744.

Sellis, T. K., Roussopoulos, N., & Faloutsos, C. (1987). The R+−tree: A dynamic index for multi-dimensional objects. *Proceedings of the 13th international conference on very large data bases* (pp. 507–518).

Sharifzadeh, M., & Shahabi, C. (2010). Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries. *Proceedings of the Vldb Endowment*, *3*(1–2), 1231–1242.

Surya Prakash, K. (2014). Data mining and warehousing for temporal data objects. *IJRCCT*, *3*(1), 123–127.

Torrens, P. M., Nara, A., Li, X., Zhu, H., Griffin, W. A., & Brown, S. B. (2012). An extensible simulation environment and movement metrics for testing walking behavior in agent-based models. *Comput Environ Urban Syst*, *36*(1), 1–17.

Wang, J., Wang, C., Song, X., & Raghavan, V. (2017). Automatic intersection and traffic rule detection by mining motor-vehicle gps trajectories. *Comput Environ Urban Syst*, *64*, 19–29.

Wei, L. -Y., Zheng, Y., & Peng, W. -C. (2012). Constructing popular routes from uncertain trajectories. *Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 195–203).

Xu, T., Li, X., & Claramunt, C. (2017). Trip-oriented travel time prediction (TOTTP) with historical vehicle trajectories. *Front Earth Sci*. http://dx.doi.org/10.1007/s11707-016-0634-8.

Xu, T., Xu, X., Hu, Y., & Li, X. (2017). An entropy-based approach for evaluating travel time predictability based on vehicle trajectory data. *Entropy*, *19*(4), 165.

Xu, X., Han, J., & Lu, W. (1990). RT-tree: An improved R-tree indexing structure for spatiotemporal data. *Proceedings of the 4th international symposium on spatial data handling* (pp. 1040–1049).

Yiu, M. L., Tao, Y., & Mamoulis, N. (2008). The b, dual, -tree: Indexing moving objects by space filling curves in the dual space. *VLDB J*, *17*(3), 379–400.

Zheng, Y., Liu, Y., Yuan, J., & Xie, X. (2011). Urban computing with taxicabs. *Proceedings of the 13th international conference on ubiquitous computing* (pp. 89–98).

Zhou, Y., Zhang, Y., Ge, Y., Xue, Z., Fu, Y., Guo, D., et al. (2015). An efficient data processing framework for mining the massive trajectory of moving objects. *Comput Environ Urban Syst*, *61*, 129–140.