SOFTWARE: PRACTICE AND EXPERIENCE
*Softw. Pract. Exper.* 2017; **47**:1081–1094
Published online 24 August 2016 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.2435

# A cloud-based taxi trace mining framework for smart city

Jin Liu[1], Xiao Yu[1], Zheng Xu[2,*,†], Kim-Kwang Raymond Choo[3,4], Liang Hong[1,5] and Xiaohui Cui[6]

[1]*State Key Lab. of Software Engineering, Computer School, Wuhan University, Wuhan, China*
[2]*Third Research Institute of Ministry of Public Security, Shanghai, China*
[3]*Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, TX, USA*
[4]*School of Information Technology and Mathematical Science, University of South Australia, Adelaide, SA, Australia*
[5]*School of Information Management, Wuhan University, Wuhan, China*
[6]*International School of Software, Wuhan University, Wuhan, China*

SUMMARY

As a well-known field of big data applications, smart city takes advantage of massive data analysis to achieve efficient management and sustainable development in the current worldwide urbanization process. An important problem in smart city is how to discover frequent trajectory sequence pattern and cluster trajectory. To solve this problem, this paper proposes a cloud-based taxi trajectory pattern mining and trajectory clustering framework for smart city. Our work mainly includes (1) preprocessing raw Global Positioning System trace by calling the Baidu API Geocoding; (2) proposing a distributed trajectory pattern mining (DTPM) algorithm based on SPARK; and (3) proposing a distributed trajectory clustering (DTC) algorithm based on SPARK. The proposed DTPM algorithm and DTC algorithm can overcome the high input/output overhead and communication overhead by adopting in-memory computation. In addition, the proposed DTPM algorithm can avoid generating redundant local trajectory patterns to significantly improve the overall performance. The proposed DTC algorithm can enhance the performance of trajectory similarity computation by transforming the trajectory similarity calculation into AND and OR operators. Experimental results indicate that DTPM algorithm and DTC algorithm can significantly improve the overall performance and scalability of trajectory pattern mining and trajectory clustering on massive taxi trace data. Copyright © 2016 John Wiley & Sons, Ltd.

Received 28 February 2016; Revised 29 May 2016; Accepted 20 July 2016

KEY WORDS: big data application; smart city; distributed trajectory pattern mining; distributed trajectory clustering; SPARK

## 1. INTRODUCTION

### *1.1. Background*

Smart city as a kind of data-driven applied technology has attracted much attention. It usually takes advantage of massive data analysis to achieve efficient management and sustainable development in the current worldwide urbanization process. Especially, traffic trajectory data analysis is one of the hot spots in the field of smart city. Traffic trajectory data analysis makes use of data mining and machine learning techniques to provide the government, businessmen and citizens multiple services and enable them to clearly understand traces of their daily activities and social behaviors, as well as the developmental tendency in cities [1–4].

An important task in smart city is to discover frequent trajectory sequence pattern (hereinafter referred to as trajectory pattern) in taxi trajectories. The discovered trajectory pattern performs as

*Correspondence to: Zheng Xu, Third Research Institute of Ministry of Public Security, Shanghai, China.
†E-mail: xuzheng@shu.edu.cn

a kind of very important tool for governors to optimize traffic networks, relieve traffic congestion and improve the sound development of traffics [5, 6]. The trajectory pattern also contains a great deal of driving experience of taxi drivers, which can be used to give guidance to navigation [7, 8].

On the other hand, we can use previously discovered trajectory pattern as features describing taxi trajectory and cluster taxi trajectory. The discovered taxi trajectory clusters contain a set of taxi trajectories containing similar trajectory patterns. Trajectory clustering also has a wide range of location-based services. For example, the resulting clusters would help provide knowledge about traffic flows as well as dense areas in a road network [9]. Such knowledge is very useful for applications in traffic monitoring [10], public transit planning and location-based advertising on mobile devices [9].

The service of trajectory pattern mining and trajectory clustering for smart city deals with a huge amount of trace data, which is a practical big data application [11]. In practice, it is appropriate to deploy the service on cloud computing environment so that it can efficiently manage and analyze extensive quantity of taxi trace data. However, the deployment of the previously mentioned trajectory analysis service in the cloud platform should consider at least three issues as follows: (1) how to process raw Global Positioning System (GPS) trace data for trajectory analysis; (2) how to achieve the high efficiency and high scalability of distributed trajectory pattern mining algorithm; and (3) how to achieve the high efficiency and high scalability of distributed trajectory clustering algorithm.

For the first issue, it is impossible to discover trajectory pattern and cluster trajectory from raw trace data [12], because the domain of GPS position coordinates is continuous and the granularity of raw data is very low. So, we cannot directly apply traditional sequence pattern mining algorithm and cluster algorithm to process raw trace data. To handle this problem, some existing studies [12–14] adopt the space partitioning strategy, which discretizes the whole space into many small grids based on a prespecified granularity. Although simple and efficient, rigid space partitioning is not suitable for mining taxi trajectory patterns. It suffers from the sharp boundary problem, namely, the locations close to grid boundaries may be assigned to different grids [15].

For the second issue, Qiao *et al*. [16] have proposed plute algorithm, a distributed trajectory pattern mining algorithm based on HADOOP. Plute algorithm proceeds through $k$ rounds of MapReduce jobs to find all trajectory patterns. The HADOOP cloud framework is not appropriate for iterative MapReduce job, because in each round, a new MapReduce job needs to read the data from and write back to HADOOP Distributed File System (HDFS), which cause high input/output (I/O) overhead and communication overhead. Moreover, the trajectory pattern mining algorithm presented in our paper is derived from a well-known GSP algorithm [17]. However, it may need to generate a huge number of candidate trajectory sequences when mining a large amount of taxi trace data. The performance degradation for the trajectory pattern mining algorithm would be obvious.

For the third issue, the trajectory clustering algorithm presented in our paper is derived from $k$-means algorithm scheme, because $k$-means algorithm is a data-intensive algorithm that needs several iterations before completion, so that it would require several scans over datasets. In this way, the distributed clustering algorithms [18, 19] based on HADOOP perform several MapReduce jobs during iterations [20]. It causes high I/O overhead and communication overhead. Furthermore, a large amount of computation of trajectory clustering algorithms focuses on calculating the similarity degree of trajectory sequence. So, we expect an efficient approach to perform the similarity calculation.

### *1.2. Our work and contribution*

Accordingly, this paper proposes a cloud-based taxi trajectory pattern mining and trajectory clustering framework for smart city. Figure 1 illustrates the framework of our works.

To preprocess the raw GPS trace data, unlike that rigid space partitioning strategy, we transform GPS position coordinate into trajectories expressed on a road segment level by calling the Baidu API Geocoding [21] because a taxi typically moves along the road. By transforming the low-granularity GPS trace into the high-granularity trajectory represented by a set of passed road segments, it avoids the sharp boundary problem and guarantees the applicability of discovered trajectory patterns and trajectory clusters.



DOI: 10.1002/spe

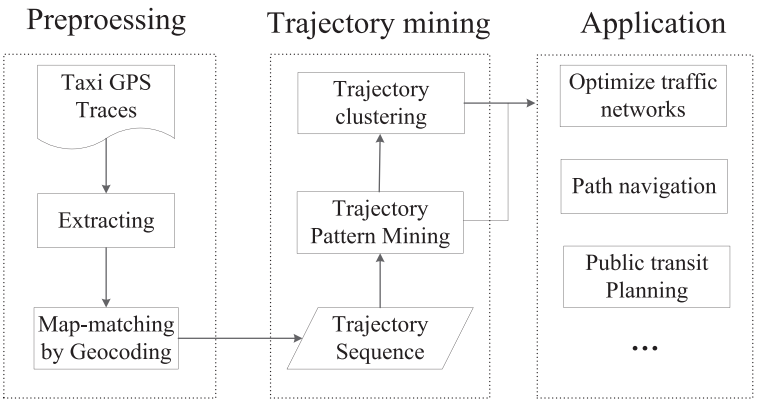


Figure 1. Overview of our framework.

To achieve the high efficiency and high scalability of distributed trajectory pattern mining algorithm, it is essential to minimize the I/O overhead and communication overhead. So, we propose an efficient and effective distributed trajectory pattern mining (DTPM) algorithm based on SPARK. DTPM algorithm adopts in-memory computation and loads trajectory sequence database and trajectory patterns generated at each iteration into resilient distributed datasets (RDDs) to overcome the high I/O overhead and communication overhead. Beyond that, we optimize the candidate pattern generation procedure of original GSP algorithm by taking advantage of the fact that two adjacent elements in a trajectory sequence $S$ are actually two adjacent road segments. So, each candidate trajectory sequence can only grow with its adjacent road segment. In this way, the procedure of the trajectory pattern mining can produce less candidate trajectory sequences and improve its performance.

To achieve the high efficiency and high scalability of distributed trajectory clustering algorithm, it is also essential to minimize the I/O overhead and communication overhead. So, we propose an efficient and effective distributed trajectory clustering (DTC) algorithm based on SPARK. DTC algorithm adopts in-memory computation and loads trajectory sequence database and the centers of the clusters at each iteration into RDD to overcome the high I/O overhead and communication overhead. In addition, DTC algorithm uses previously discovered trajectory pattern as features describing taxi trajectory and encodes each trajectory sequence with various lengths into an $n$-dimensional trajectory vector according to trajectory sequence bitmap. Because we have already transformed each trajectory sequence into an $n$-dimensional vector, this similarity calculation with $n$-dimensional vector can further be transformed into AND and OR operators.

Motivated by these observations, in this paper, we address these challenges through the following contributions:

1. The proposed taxi GPS trace preprocessing approach can avoid the sharp boundary problem and guarantees the applicability of discovered trajectory patterns and trajectory clusters.
2. The proposed DTPM algorithm can avoid generating redundant trajectory patterns to significantly improve the overall performance.
3. The proposed DTC algorithm can enhance the performance of trajectory similarity computation by transforming the trajectory similarity calculation into AND and OR operators.
4. The proposed DTPM algorithm and DTC algorithm can overcome the high I/O overhead and communication overhead of traditional distributed algorithms by adopting in-memory computation

### *1.3. The organization*

The remainder of this paper is organized as follows. Section 2 presents the related work. In Section 3, we present the problem definition. Sections 4 and 5 describe our proposed DTPM algorithm and trajectory cluster algorithm respectively. In Section 6, some experiments are demonstrated that show the scalability and efficiency of the proposed algorithms. Finally, Section 7 addresses the conclusion and future work.



DOI: 10.1002/spe

## 2. RELATED WORK

In this section, we will introduce the related works that can be categorized into two research directions including trajectory pattern mining and trajectory clustering.

### *2.1. Trajectory pattern mining*

Some existing studies [12–14] have investigated mining trajectory pattern in database of moving object locations. To handle spatial continuity, they adopt the space partitioning strategy. It suffers from the sharp boundary problem [15]. Chen *et al*. [22] aim to discover the most popular route between two locations by observing the traveling behaviors of many previous users. Wei *et al*. [23] present a route inference framework based on collective knowledge to construct the popular routes from uncertain trajectories. Luo *et al*. [24] propose a scalable method for finding the most frequent path during user-specified time periods in large-scale historical trajectory data. Zhang *et al*. [15] propose SPLITTER algorithm to mine what they call fine-grained sequential patterns, which must satisfy spatial compactness, semantic consistency and temporal continuity simultaneously. Jiang *et al*. [25] investigate the problem of discovering all movement patterns from semantic trajectory databases. Qiao *et al*. [26] propose a frequent trajectory pattern tree mining algorithm, which is applied to find a trajectory sequence with a series of frequently visited atomic roads. However, the existing serial algorithms for trajectory pattern mining struggle to dealing with massive trajectory data. In practice, it is appropriate to implement distributed trajectory pattern mining algorithms on cloud computing environment so that it can efficiently mine extensive quantity of taxi trajectory.

The distributed trajectory pattern mining algorithms on cloud computing environment have recently received increasing attention. Qiao *et al*. [16] propose an efficient and effective parallel trajectory pattern mining (plute) algorithm that includes three essential techniques: prefix projection, data parallel formulation and task parallel formulation. Plute algorithm utilizes an iterative MapReduce framework to implement trajectory pattern mining on HADOOP cloud computing environment. One major drawback of the algorithm is that it needs to proceed through numerous rounds of MapReduce jobs, which will easily cause high I/O overhead and communication overhead.

### *2.2. Trajectory clustering*

Some existing studies [27–34] have investigated clustering the trajectories of mobile objects in a road network. They adopt traditional *k*-means, hierarchical or density-based clustering algorithms to group similar trajectories. Nanni *et al*. [27] investigate group similar trajectories as a whole, while Lee *et al*. [28] aim to find similar subtrajectories. The trajectory similarity measure in the clustering algorithms is Euclidean distance. However, as it demands the length of two trajectories be equal by using Euclidean distance, its application scope is very limited [29]. Li *et al*. [32] aim to study online trajectory clustering. Lee *et al*. [31] use hierarchical region-based and trajectory-based clustering for trajectory classification. However, these density-based methods cluster free space trajectories, that is, without consider constrained road network [34].

Other works [29–31,34] consider the network constraint to derive similarity measures. But, they may suffer from expensive trajectory similarity computation. Our approach transforms trajectory similarity computation into AND and OR operators, thus avoiding the expensive trajectory similarity computation. On the other hand, the existing serial algorithms for trajectory clustering cannot be applied to deal with massive trajectory data. Zhao and Anchalia *et al*. [18, 19] propose a parallel *k*-means clustering algorithm based on HADOOP. It also causes high I/O overhead and communication overhead because of the iterative MapReduce job.

## 3. PRELIMINARIES

In this section, we first define some terms used in our paper. Then, we formalize the trajectory pattern mining problem and trajectory clustering problem.



DOI: 10.1002/spe

*Definition 1 (GPS log)*
As shown in the left part of Figure 2, a GPS log is a collection of GPS points $p_i$ generated by the taxi GPS device. Each GPS point $p_i$ contains the time stamp ($p_i$.T), the latitude ($p_i$.Lat), longitude ($p_i$.Lngt) and the operation status (i.e. 1 or 0; 1 stands for occupied, and 0 stands for vacant).

As vacant taxi trajectory does not contain passenger information, in this paper, we only consider the taxi trajectories where taxi's operation statue is occupied. A taxi typically moves along the road segment. Therefore, it is more useful and intuitive to represent a trajectory as a series of road segment name. For example, suppose that a occupied taxi has passed by '$p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}$'. As shown in the right part of Figure 2, it is much easier to understand to represent its trajectory as a road segment name of '$r_1$, $r_2$, $r_3$, $r_4$, $r_5$' rather than consecutive GPS points. With the development of technology, many Internet companies have provided their web map server interface. In our paper, we leverage API Geocoding [21] provided by Baidu to match a raw GPS point $p_i$ to road segment $r_i$ on Baidu Map.

*Definition 2 (Trajectory sequence)*
A taxi trajectory sequence $S$ is a sequence of road segment pertaining to an occupied taxi ride, which can be represented as $S=<r_1, r_2,\ldots, r_m>$ where $r_i$ is a road segment. The number of road segments contained in $S$ is defined as the length of a trajectory sequence. A trajectory sequence with length $l$ is called an $l$ trajectory sequence.

*Property 1*
The adjacent two elements $r_i$ and $r_{i+1}$ in a trajectory sequence $S$ are two adjacent road segments. For example, if an element in a trajectory sequence $S$ is $r_1$, the adjacent element of $r_1$ in $S$ should be $r_2$, $r_6$ or $r_{13}$.

*Definition 3 (Trajectory sequence database)*
A trajectory sequence database $D$ is a set of tuples (*Sid*, $S$), where *Sid* is a trajectory sequence id and $S$ is a trajectory sequence. The number of trajectory sequences in $D$ is defined as $|D|$.

*Definition 4*
A trajectory sequence $\alpha=<a_1, a_2,\ldots a_n>$ is contained in another trajectory sequence $\beta=<\beta_1, \beta_2,\ldots \beta_m>$ if there are integers $1\le j_1<j_2<\ldots<j_n\le m$ such that $a_1=\beta_{j1}$, $a_2=\beta_2,\ldots, a_n=\beta_{jn}$. If trajectory sequence $\alpha$ is contained in trajectory sequence $\beta$, then we call $\beta$ contains $\alpha$.

*Definition 5 (Trajectory pattern)*
The support count of a trajectory sequence $S$ in a trajectory sequence database $D$ is defined as the number of trajectory sequences that contain $S$, and the support of a trajectory sequence $S$ is defined as the percentage of trajectory sequences that contain $S$. The support of $S$ in $D$ is denoted by $sup_D(S)$. Given a minimal support $\xi$, a trajectory sequence $S$ is called a trajectory pattern on $D$ if $sup_D(S)\ge\xi$. A trajectory pattern with length $k$ is called a $k$ trajectory pattern and defined as $L_k$. The taxi trajectory pattern mining problem can be then defined as the following. Given an input trajectory sequence database $D$ and a minimal support $\xi$, find all trajectory patterns with support equal to or bigger than $\xi$ in $D$.

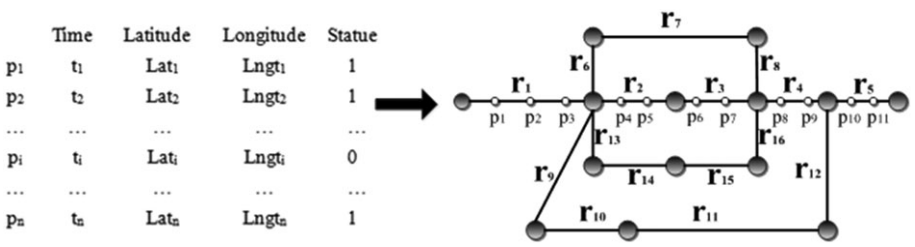


Figure 2. (a) GPS log, a trajectory sequence.



DOI: 10.1002/spe

*Definition 7 (Trajectory cluster)*
A taxi trajectory cluster is a set of taxi trajectory sequences supporting at least one previously discovered trajectory pattern. The taxi trajectory clustering problem can be then defined as the following. Given an input trajectory sequence database $D$, partition $D$ into trajectory clusters, such that trajectories in a trajectory cluster are similar to one another, yet dissimilar to trajectories in other trajectory clusters.

## 4. IMPLEMENTATION OF TRAJECTORY PATTERN MINING

The trajectory pattern mining algorithm presented in our paper is derived from a well-known GSP algorithm [17]. In this section, we first describe the detail of the GSP algorithm improvement to handle taxi trajectory. Then, we describe our DTPM algorithm based on SPARK.

### *4.1. GSP algorithm improvement*

The basic structure of the GSP algorithm for finding trajectory pattern is as follows. The algorithm makes multiple database scans. In the first pass, all single road segments are counted to find $L_1$. From $L_1$, a set of candidate *2* trajectory sequences ($C_2$) are formed, and another scan is made to identify their support to find $L_2$. The $L_2$ are used to generate the candidate *3* trajectory sequence ($C_3$), and this process is repeated until no more trajectory patterns are found.

If a trajectory sequence database contains large amount of sequences, many candidate trajectory sequences may be generated. In particular, if the minimal support is defined low, many candidate *2* trajectory sequences are generated. The improved GSP algorithm modifies the candidate trajectory pattern generation procedure according to property 1. The modified candidate trajectory pattern generation procedure is as follows: A one trajectory sequence $s_1$ and a one trajectory sequence $s_2$ can be joined only when $s_1$ and $s_2$ are adjacent road segments. So, the number of generated candidate *2* trajectory sequences is reduced; the time of trajectory sequence database scan will be reduced, thus improving the performance of GSP algorithm. The modified candidate trajectory pattern generation procedure is as follows:

**Algorithm 1:** ModifiedGenCandidate($L_k$)

**INPUT**: $L_k$: the $k$ trajectory pattern
**Output**: $C_{k+1}$: candidate $k+1$ trajectory pattern
1:**for** each trajectory $s_1 \in L_k$
2: **for** each trajectory $s_2 \in L_k$
3: **if** ($k$==1)
4: **if**($s_1$ and $s_2$ are adjacent road segment)
5: c = $s_1 \infty s_2$; /?/Join Phase
6: add c to $C_{k+1}$;
7: **end if**
8: **else**
9: **if**(($s_1[2] = s_2[1]) \wedge (s_1[3] = s_2[2]) \wedge \ldots \wedge (s_1[k] = s_2[k-1])$)
10: c = $s_1 \infty s_2$ ;/?/Join Phase
11: **if**(c contains non-trajectory pattern)
12: delete c;
13: **else**
14: add c to $C_{k+1}$;
15: **end if**
16: return $C_{k+1}$



DOI: 10.1002/spe

*4.2. DTPM algorithm.* DTPM algorithm includes two phases. Figure 3 demonstrates the running process of DTPM algorithm.

*Phase I*

As shown in the left part of Figure 3, firstly, the trajectory sequence database from HDFS is loaded into the SPARK RDD to make the advantage of the cluster memory. Then, two flatMap() functions are applied to read all trajectory sequences and get all road segment items. After two flatMap() functions are completed, a map() function is applied to convert each road segment item into <item, 1> key/value pairs. Finally, a reduceBykey() function is invoked to calculate the support count of the road segment items and find the road segment item with the minimal support $\xi$, that is $L_1$.

*Phase II*

In this phase, we iteratively use $k$ trajectory pattern ($L_k$) to generate $k+1$ trajectory pattern ($L_{k+1}$). As illustrated in the right part of Figure 3, at $k$th iteration, firstly, we read $L_k$ from RDD to generate candidate $k+1$ trajectory pattern ($C_{k+1}$) via modified candidate trajectory pattern generation procedure. Then, a map() function is applied to yield the $<C_{k+1}, 1>$ key/value pairs. Finally, a reduceBykey() function is invoked to calculate the support count of $C_{k+1}$ and obtain $L_{k+1}$.

The pseudo of DTPM algorithm is as follows:

**Algorithm 2(a):** Phase I of DTPM algorithm

**Input**: $D$: the trajectory sequence database, $\xi$:the minimal support
**Output**: <key,value>: key is the $L_1$, value is the support count of $L_1$
1:**Foreach** trajectory sequence $S$ in trajectory sequence database $D$
2: flatMap(line offset, $S$);
3: **Foreach** road segment item $r$ in $S$
4: yield $< r$,1>;
5:reduceBykey($r$,count)
6: sum = 0;
7: **while** (item $r$ in partition)
8: sum + =count;
9: **if**(sum $\geq \xi$)
10: yield $< r$,sum>;

**Algorithm 2(b):** Phase II of DTPM algorithm

**Input**: $L_k$, $\xi$: the minimal support
**Output**:$<L_{k+1}$,the support count of $L_{k+1}>$
1:$C_{k+1}$ = ModifiedgenCandidate($L_k$)
1:**Foreach** trajectory sequence $S$ in trajectory sequence database $D$
2: flatMap(line offset, $S$);
3: **Foreach** candidate $c$ in $C_{k+1}$
4: yield $< c$,1>;
5:reduceBykey($c$,count)
6: sum = 0;
7: **while** (candidate $c$ in partition)
8: sum + =count;
9: **if**(sum $\geq \xi$)
10: yield $< c$, sum>



DOI: 10.1002/spe

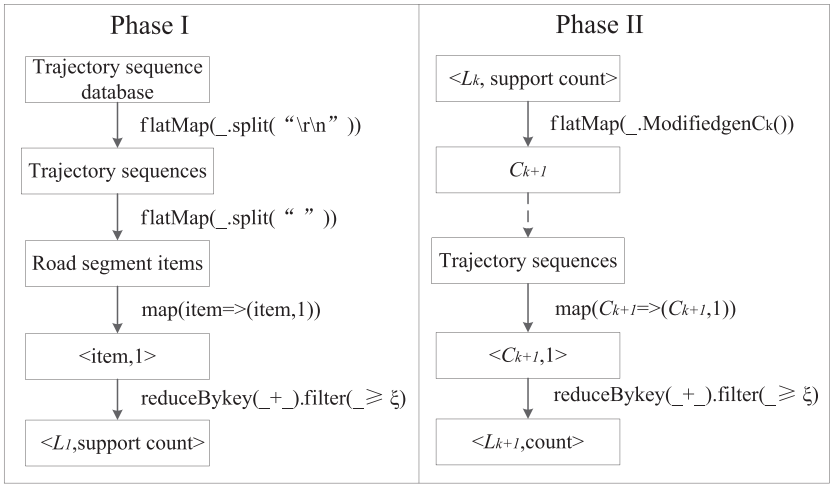


Figure 3. Running process of DTPM algorithm.

## 5. IMPLEMENTATION OF TRAJECTORY CLUSTERING

In this section, we first propose a new trajectory similarity measurement method. Then, we describe our proposed DTC algorithm based on SPARK.

### *5.1. Trajectory similarity measurement*

The similarity computation is the base of the clustering algorithm. We use previously discovered trajectory pattern as feature describing taxi trajectory. Assume that we have obtained four trajectory patterns: $tp_1 = <r_1>$, $tp_2 = <r_2, r_3>$, $tp_3 = <r_9, r_{10}, r_{11}>$ and $tp_4 = <r_4, r_5>$, we can encode two different-length trajectory sequence $s_1 = <r_1, r_2, r_3, r_4, r_5>$ and trajectory sequence $s_2 = <r_1, r_6, r_7, r_8, r_4, r_5>$ into two $n$-dimensional vector according to trajectory sequence bitmap as shown in Table I, where $n$ is the number of trajectory patterns. Because $s_1$ contains $tp_1$, the first digit of $n$-dimensional vector is 1. Because $s_1$ does not contain $tp_3$, the third digit is 0. By this analogy, each trajectory sequence of different length can be represented by an $n$-dimensional vector.

Not like previous works, we judge the degree of trajectory similarity by considering the trajectory sequence containing similar trajectory patterns. We apply Jaccard coefficient to define the trajectory similarity according to Equation (1).

$$Sim\_traj(s_1, s_2) = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|} \tag{1}$$

where $|s_1 \cap s_2|$ is the number of intersection of the trajectory patterns that $s_1$ and $s_2$ contain, and $|s_1 \cup s_2|$ is the number of the union set of the trajectory patterns that $s_1$ and $s_2$ contain.

Because trajectory clustering algorithm suffers from computing the similarity of trajectory sequence, we transform the similarity computation of trajectory sequence to the similarity computation of $n$-dimensional vector. In order to calculate the similarity between $s_1$ and $s_2$ represented by $n$-dimensional vector, we take advantage of AND and OR operators to process similarity

Table I. Trajectory sequence bitmap.

| | $tp_1$ | $tp_2$ | $tp_3$ | $tp_4$ |
|---|---|---|---|---|
| $s_1$ | 1 | 1 | 0 | 1 |
| $s_2$ | 1 | 0 | 0 | 1 |



DOI: 10.1002/spe

calculation. For example, we have obtained $s_1 = 1101$ and $s_2 = 1001$. Then, we conclude $s_1 \wedge s_2 = 1001$, that the quantity of '1' is 2 means that the number of intersection of the trajectory patterns that $s_1$ and $s_2$ contain is 2. We conclude $s_1 \vee s_2 = 1101$, that the quantity of 1 is 3 means the number of union set of the trajectory patterns that $s_1$ and $s_2$ contain is 3. So, the Sim_traj($s_1$, $s_2$) = 2/3. In this way, this similarity calculation with $n$-dimensional vector can further be transformed into AND and OR operators, thus improving the efficiency of the clustering algorithm.

In our paper, we employ $k$-means algorithm scheme for efficient trajectory clustering. We also need to define the method of calculating the center of a trajectory cluster. The center $X_i$ of a trajectory cluster $C_i$ is also an $n$-dimensional vector. So, we need to determine the $j$th digit $x_{ij}$ of $X_i$. If the number of $j$th trajectory pattern $S$ that $C_i$ contains is bigger than or equal to $sup_D(S)$, then we call the cluster $C_i$ supports the $j$th trajectory pattern. The value of $x_{ij}$ indicates whether the cluster $C_i$ supports the $j$th trajectory pattern. If the cluster $C_i$ supports the $j$th trajectory pattern, the value of $x_{ij}$ is 1; otherwise, the value of $x_{ij}$ is 0.

### 5.2. DTC algorithm

Based on the proposed similarity measurement, this paper employs $k$-means algorithm scheme for efficient trajectory clustering. In this regard, this paper addresses DTC algorithm based on SPARK capable of handling a large amount of taxi trajectory. Figure 4 demonstrates the running process of DTC algorithm.

Firstly, the trajectory sequence database represented by $n$-dimensional vectors from HDFS is loaded into the SPARK RDD to make the advantage of the cluster memory. Then, we randomly choose $k$ trajectory sequences as the initial centers of the clusters. Finally, we iteratively update the centers of clusters until the centers do not change. As illustrated in Figure 4, at each iteration, firstly, we compute the distance between the trajectory sequence $S$ and the centers of the clusters and then assign $S$ to the closest cluster $C_i$ based on the distance. Then, a map() function is applied to yield the $<S, C_i>$ key/value pairs. Finally, a reduceBykey() function is invoked to merge all the trajectory sequences assigned to the same cluster and then compute the new centers for next iteration.

## 6. EVALUATION

### 6.1. Experiment data and experiment environment

We use the road network data of Beijing, which contains 148,110 road nodes and 96,307 road segments. The total length of the road segments is 21,895 km, and the spatial area covered by the road network is 2507 $km^2$. We use the GPS trace data generated by over 33,000 taxis on 1 November 2012 and 2 November 2012. The data in these 2 days is hereinafter referred to as dataset 1 and dataset 2. The total distance of taxi trajectories is more than 18 million km, and the number of points reaches eight million. The experimental platform is a computer cluster. Each machine has one

**Algorithm 3:** DTC algorithm

**Input**: $D$: the trajectory sequence database,
**Output**: <key,value>: key is the index of the center, value is the center of the cluster
1:**Foreach** trajectory sequence $S$ represented by $n$-dimensional vector in $D$
2: **Foreach** the center $C_i$ of the cluster
3: ComputeDist ($S$,$C_i$);
4: yield < *index*,*S*>;//key is the index of the closest center of $S$, value is $S$
5:reduceBykey(*index*,*S*)
6: Foreach the cluster
7: $C_i'$ = ComputeNewCenter( ); /?/compute the new center for each cluster
8: yield < index, $C_i'$ >;



DOI: 10.1002/spe

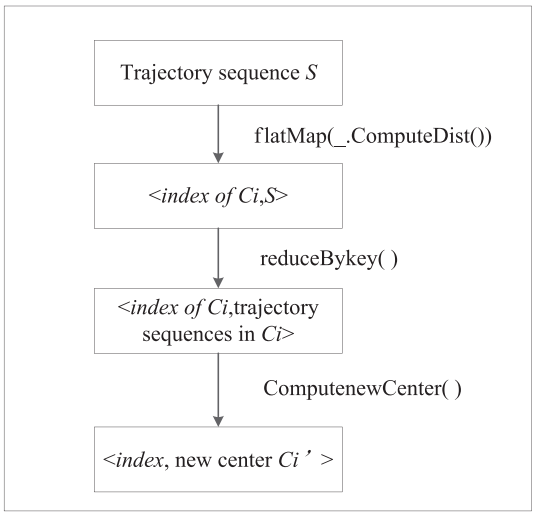


Figure 4. Running process of DTC algorithm.

single-core processor and 4 GB memory; software configuration is on CENTOS 6.6, HADOOP 2.4 and SPARK 1.2.

### 6.2. *Performance analysis*

*Experiment 1*

In this experiment, we analyze the running time of DTPM algorithm and plute algorithm [16] for different minimal support in a cluster (one Master and five DataNodes). The experimental results are shown in Figure 5a,b. Then, we use trajectory patterns generated by DTPM algorithm

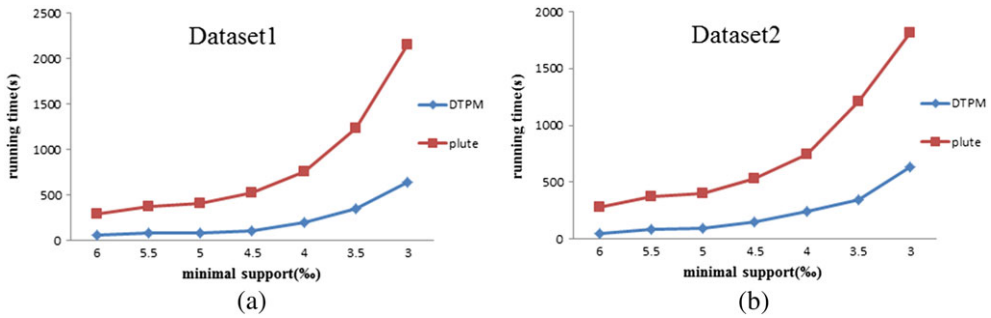


Figure 5. (a) Dataset 1. (b) Dataset 2. Running time of DTPM and plute with different minimal support. [Colour figure can be viewed at wileyonlinelibrary.com]

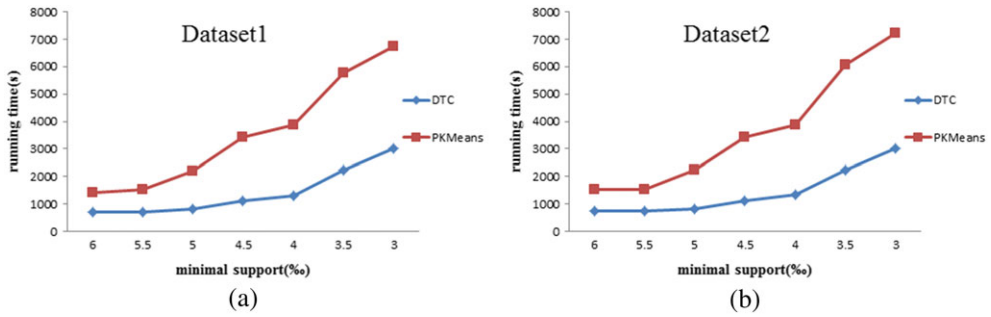


Figure 6. (a) Dataset 1. (b) Dataset 2. Running time of DTC and PKMeans with different minimal support. [Colour figure can be viewed at wileyonlinelibrary.com]



DOI: 10.1002/spe

with different minimal support as test datasets to analyze the running time of DTC algorithm and PKMeans [18] algorithm (set $k$ to 100). The experimental results are shown in Figure 6a,b.

*Result analysis*

As we can see in Figure 5, the running time of DTPM and plute increases along with the decrease of minimal support because the smaller the minimal support is, the more trajectory patterns there will be. As we can see in Figure 6, the running time of DTC and PKMeans also increase along with the decrease of minimal support because the more generated trajectory patterns, the higher dimensional the trajectory will be. But, DTPM and DTC adopt in-memory computation and load trajectory sequence database into RDD that reduce the overall time spent in I/O and network communication. On the other hand, DTPM algorithm modifies the candidate trajectory pattern generation procedure to avoid to generating redundant trajectory patterns, thus significantly improving the overall performance. DTC algorithm transforms the trajectory similarity calculation into AND and OR operators to improve the overall performance. So, the performance of DTPM and DTC is more stable along with the decrease of minimal support.

*Experiment 2*

In this experiment, we test the speedup of DTPM and DTC with different number of DataNodes. The minimal support is set to 0.5%. *Speedup* means the rate of performance enhanced when adding extra nodes to the cluster. The experimental results are shown in Figure 7a, b.

*Result analysis*

It can be observed that DTPM and DTC have good speedup performance, because the amount of trajectory sequences being handled by each node reduces with the number of DataNodes increasing. But, the speedup will not be fully proportional to the number of DataNodes, because the running time is affected by data transmission between machines.

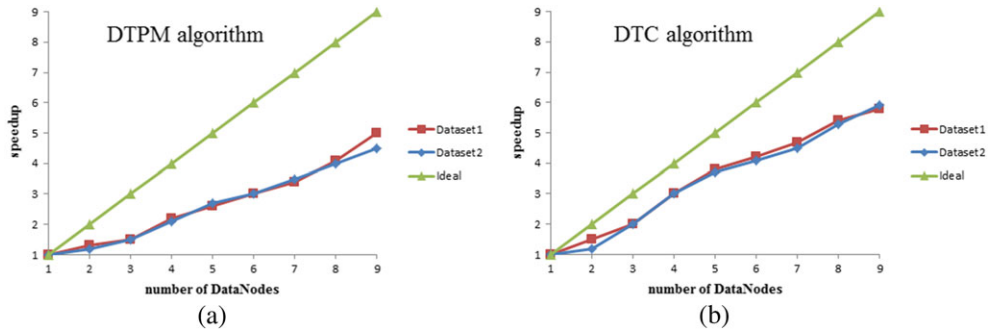


Figure 7. (a) DTPM algorithm. (b) DTC algorithm. Speedup comparison on cluster size. [Colour figure can be viewed at wileyonlinelibrary.com]

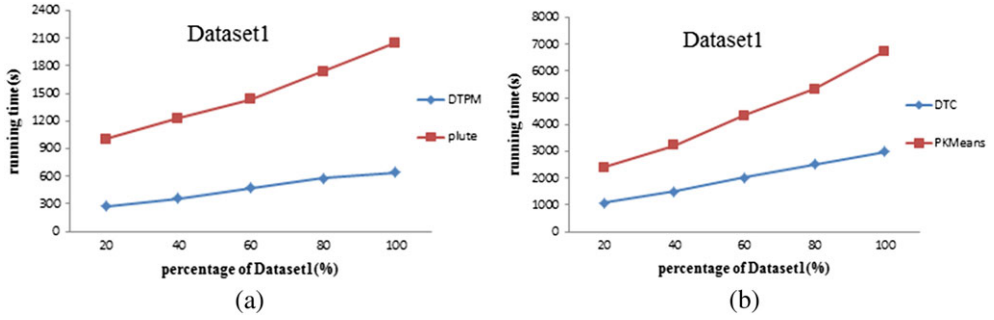


Figure 8. (a) Running time of DTPM and plute. (b) Running time of DTC and PKMeans. Running time of algorithms with different number of sequences. [Colour figure can be viewed at wileyonlinelibrary.com]



DOI: 10.1002/spe

*Experiment 3*

In this experiment, we use five datasets each containing 20 to 100% of dataset 1 as test datasets with minimal support 0.3%. We analyze the running time of DTPM and plute for different size of test datasets in a cluster (one Master and five DataNodes). The experimental results are shown in Figure 8a. Then, we use trajectory patterns generated by DTPM algorithm with different sizes of test datasets to analyze the running time of DTC algorithm and PKMeans algorithm. The experimental results are shown in Figure 8b.

*Result analysis*

As we can see, the running time of plute and PKMeans will increase proportionally to the size of the dataset. But, for DTPM and DTC, as shown in Figure 8, their running time increases slightly along the size of the dataset because of in-memory computation strategy. In addition, modifying the candidate trajectory pattern generation procedure and transforming the trajectory similarity calculation into AND and OR operators also enhance the overall performance of DTPM algorithm and DTC algorithm respectively.

## 7. CONCLUSION

In this paper, we propose a cloud-based taxi trace mining framework for smart city. It mainly involves efforts from three aspects: trace data preprocessing, SPARK-based trajectory pattern mining and SPARK-based trajectory clustering. To preprocess raw GPS trace data, we transform GPS position coordinate into trajectories expressed on a road segment level so that we can avoid the sharp boundary problem and guarantee the applicability of discovered trajectory patterns and trajectory clusters. To overcome the high I/O overhead and communication overhead, the proposed DTPM algorithm and DTC algorithm adopt in-memory computation and load the trajectory sequence database into the SPARK RDD to make the advantage of the cluster memory. To further improve the mining efficiency, two specific improved strategies are applied, that is, modifying the candidate trajectory pattern generation procedure and transforming the trajectory similarity calculation into AND and OR operators. The experimental results indicated the feasibility and the effectiveness of the proposed DTPM algorithm and DTC algorithm. Our future research direction includes (1) achieving the minimum dataset storage in the cloud [35]; (2) visualizing the mining results on Baidu Map; (3) implementing the distributed algorithms using other distributed platforms such as the G-Hadoop platform [36]; (4) parallel modelling & simulation of crowd based on the mined trajectory pattern [37, 38]; (5) exploring more trajectory sequence feature extraction methods for trajectory clustering [39, 40]; and (6) exploring dictionary learning algorithms for taxi spatial data [41, 42].

ACKNOWLEDGEMENTS

This work is partly supported by the grants of National Natural Science Foundation of China (61572374, U1135005, 61303025, 61363030, 61440054).

REFERENCES

1. Castro PS, Zhang D, Chen C, et al. From taxi GPS traces to social and community dynamics: a survey. *ACM Computing Surveys* 2013; **46**(2): 17.
2. Qi G, Li X, Li S, et al. Measuring social functions of city regions from large-scale taxi behaviors. In *Proceedings of the 2011 International Conference on Pervasive Computing and Communications Workshops*, 2011; 384–388.
3. Li X, Pan G, Wu Z, et al. Prediction of urban human mobility using large-scale taxi traces and its applications. *Frontiers of Computer Science* 2012; **6**(1): 111–121.
4. Song C, Qu Z, Blumm N, et al. Limits of predictability in human mobility. *Science* 2010; **327**(5968): 1018–1021.
5. Castro PS, Zhang D, Li S. Urban traffic modelling and prediction using large scale taxi GPS traces. *Pervasive Computing* 2012: 57–72.
6. Zheng Y, Liu Y, Yuan J, et al. Urban computing with taxicabs. In *Proceedings of the 13th International Conference on Ubiquitous computing*, 2011; 89–98.



DOI: 10.1002/spe

7. Ceikute V, Jensen CS. Vehicle routing with user-generated trajectory data. In *Proceeding of the16th IEEE International Conference on Mobile Data Management*, 2015; **1**: 14–23.
8. Yuan J, Zheng Y, Zhang C, et al. T-drive: driving directions based on taxi trajectories. In *Proceeding of the 18th SIGSPATIAL International conference on advances in geographic information systems*, 2010; 99–108.
9. Han B, Liu L, Omiecinski E. Road-network aware trajectory clustering: integrating locality, flow, and density. *IEEE Transactions on Mobile Computing* 2015; **14**(2): 416–429.
10. Mohan P, Padmanabhan V N, Ramjee R. Nericell: rich monitoring of road and traffic conditions using mobile smartphones, In: *Proceeding of the 6th ACM conference on Embedded network sensor systems*, 2008; 323–336.
11. Jara AJ, Genoud D, Bocchi Y. Big data for smart cities with KNIME a real experience in the SmartSantander testbed. *Software: Practice and Experience* 2015; **45**(8): 1145–1160.
12. Morzy M. Mining frequent trajectories of moving objects for location prediction. *Machine Learning and Data Mining in Pattern Recognition.* 2007; 667–680.
13. Yang J, Hu M. Trajpattern: mining sequential patterns from imprecise trajectories of mobile object. *Advances in Database Technology-Edbt* 2006; 664–681.
14. Giannotti F, Nanni M, Pinelli F, et al. Trajectory pattern mining, In *Proceeding of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2007; 330–339.
15. Zhang C, Han J, Shou L, et al. Splitter: mining fine-grained sequential patterns in semantic trajectories. In *Proceeding of the VLDB Endowment*, 2014; **7**(9): 769–780.
16. Qiao S, Li T, Peng J, et al. Parallel sequential pattern mining of massive trajectory data. *International Journal of Computational Intelligence Systems* 2010; **3**(3): 343–356.
17. Srikant R, Agrawal R. *Mining sequential patterns: generalizations and performance improvements*. Berlin Heidelberg: Springer, 1996: 1–17.
18. Zhao W, Ma H, He Q. Parallel k-means clustering based on mapreduce. *Cloud computing* 2009: 674–679.
19. Anchalia P P, Koundinya A K, Srinath N K. MapReduce design of k-means clustering algorithm. In *International Conference on Information Science and Applications*, 2013; 1–5.
20. Mohebi A, Aghabozorgi S, Ying Wah T, et al. Iterative big data clustering algorithms: a review. *Software: Practice and Experience* 2016; **46**(1): 107–129.
21. Baidu Geocoding. 2016. Available from: http://lbsyun.baidu.com/
22. Chen Z, Shen H T, Zhou X. Discovering popular routes from trajectories, In *Proceeding of the 27th International Conference on Data Engineering*, 2011; 900–911.
23. Wei LY, Zheng Y, Peng W C. Constructing popular routes from uncertain trajectories, In *Proceeding of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012; 195–203.
24. Luo W, Tan H, Chen L, et al. Finding time period-based most frequent path in big trajectory data, In *Proceeding of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013) 713–724.
25. Jiang R, Zhao J, Dong T, et al. A density-based approach for mining movement patterns from semantic trajectories, In *Proceeding of the IEEE Region 10 Conference on TENCON* 2015; 1–6.
26. Qiao S, Han N, Zhu W et al. TraPlan: an effective three-in-one trajectory-prediction model in transportation networks. *IEEE Transactions on Intelligent Transportation Systems* 2015; **16**(3): 1188–1198.
27. Nanni M, Pedreschi D. Time-focused clustering of trajectories of moving objects. *Journal of Intelligent Information Systems* 2006; **27**(3): 267–289.
28. Lee J G, Han J, Whang K Y. Trajectory clustering: a partition-and-group framework, In *Proceeding of the 2007 International Conference on Management of data*, 2007; 593–604.
29. Won J I, Kim S W, Baek J H, et al. Trajectory clustering in road network environment, In *Proceeding of the 2009 IEEE Symposium on Computational Intelligence and Data Mining*, 2009; 299–305.
30. Kharrat A, Popa IS, Zeitouni K et al. Clustering algorithm for network constraint trajectories. In *Headway in Spatial Data Handling*, 2008; 631–647.
31. Roh GP, Hwang S. Nncluster: an efficient clustering algorithm for road network trajectories. In *Database systems for advanced applications*, 2010; 47–61.
32. Li Z, Lee JG, Li X et al. Incremental clustering for trajectories. In *Database Systems for Advanced Applications*, 2010: 32–46.
33. Lee J G, Han J, Li X, et al. TraClass: trajectory classification using hierarchical region-based and trajectory-based clustering, In *Proceeding of the VLDB Endowment*, 2008; **1**(1): 1081–1094.
34. Han B, Liu L, Omiecinski E. Neat: road network aware trajectory clustering, In *Proceeding of the IEEE 32nd International Conference on Distributed Computing Systems*, 2012; 142–151.
35. Yuan D, Yang Y, Liu X et al. A highly practical approach toward achieving minimum data sets storage cost in the cloud. *IEEE Transactions on Parallel and Distributed Systems* 2013; **24**(6): 1234–1244.
36. Wang L, Tao J, Ranjan R et al. G-Hadoop: MapReduce across distributed data centers for data-intensive computing. *Future Generation Computer Systems* 2013; **29**(3): 739–750.
37. Chen D, Wang L, Wu X et al. Hybrid modelling and simulation of huge crowd over a hierarchical grid architecture. *Future Generation Computer Systems* 2013; **29**(5): 1309–1317.
38. Chen D, Wang L, Zomaya A, Dou M, Chen J, Deng Z, Hariri S. Parallel Simulation of Complex Evacuation Scenarios with Adaptive Agent Models. *IEEE Transactions on Parallel and Distributed Systems* 2015; **26**(3): 847–857.
39. Chen D, Li X, Wang L, Khan S, Wang J, Zeng K, Cai C. Fast and Scalable Multi-way Analysis of Massive Neural Data. *IEEE Transactions on Computer* 2015; **64**(3): 707–719.




DOI: 10.1002/spe

40. Wang L, Song W, Liu P. Link the remote sensing big data to the image features via wavelet transformation. *Cluster Computing* 2016; **19**(2): 793–810.
41. Wang L, Lu K, Liu P, Ranjan R, Chen L. IK-SVD: Dictionary Learning for Spatial Big Data via Incremental Atom Update. *Computing in Science and Engineering* 2014; **16**(4): 41–52.
42. Wang L, Geng H, Liu P, Lu K, Kolodziej J, Ranjan R, Zomaya A. Particle Swarm Optimization based dictionary learning for remote sensing big data. *Knowledge-Based Systems* 2015; **79**: 43–50.



DOI: 10.1002/spe