Chen W, Zhao L, Xu JJ *et al.* Trip oriented search on activity trajectory. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 30(4): 745–761 July 2015. DOI 10.1007/s11390-015-1558-6

Trip Oriented Search on Activity Trajectory

Wei Chen [1] (陈 伟), Lei Zhao [1,2,*] (赵 雷), *Member, CCF, ACM*
Jia-Jie Xu [1,2] (许佳捷), *Member, CCF, ACM*, Guan-Feng Liu [1,2] (刘冠锋), *Member, CCF, ACM*
Kai Zheng [1] (郑 凯), *Member, CCF, ACM, IEEE*, and Xiaofang Zhou [1,3] (周晓方), *Member, CCF, ACM, IEEE*

[1] *School of Computer Science and Technology, Soochow University, Suzhou 215006, China*

[2] *Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing 210023, China*

[3] *School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane QLD 4072, Australia*

E-mail: wchzhg@gmail.com; {zhaol, xujj, gfliu}@suda.edu.cn; {kevinz, zxf}@itee.uq.edu.au

Received February 1, 2015; revised May 13, 2015.

Abstract Driven by the flourish of location-based services, trajectory search has received significant attentions in recent years. Different from existing studies that focus on searching trajectories with spatio-temporal information and text descriptions, we study a novel problem of searching trajectories with spatial distance, activities, and rating scores. Given a query q with a threshold of distance, a set of activities, a start point S and a destination E, trip oriented search on activity trajectory (TOSAT) returns k trajectories that can cover the activities with the highest rating scores within the threshold of distance. In addition, we extend the query with an order, i.e., order-sensitive trip oriented search on activity trajectory (OTOSAT), which takes both the order of activities in a query q and the order of trajectories into consideration. It is very challenging to answer TOSAT and OTOSAT efficiently due to the structural complexity of trajectory data with rating information. In order to tackle the problem efficiently, we develop a hybrid index AC-tree to organize trajectories. Moreover, the optimized variant RAC$^+$-tree and novel algorithms are introduced with the goal of achieving higher performance. Extensive experiments based on real trajectory datasets demonstrate that the proposed index structures and algorithms are capable of achieving high efficiency and scalability.

Keywords trajectory search, rating score, activity trajectory

1 Introduction

With proliferation of mobile devices and rapid development of wireless sensor technology, a large amount of trajectory data are generated at an unprecedent scale. In the last two decades, trajectory similarity search has been extensively studied to achieve effective utilization of trajectory data, and a lot of indexing structures[1-2] and knowledge discovery methods[3-5] have been proposed. Above techniques have been widely used in developing effective location-based recommendation systems (LBRS), such as trip planning, online navigation and supply chain management systems.

In most of the existing work[6-9], a trajectory is modeled as a sequence of spatio-temporal locations, and the queries are conducted in spatial domain. However, in recent years, a huge volume of activity trajectories have been generated from the social media websites, such as a sequence of geo-tagged records in Foursquare, Facebook and Bikely. Each location in an activity trajectory contains both spatial and activity information (e.g., sport, dining and entertaining). This new type of trajectory records the semantically meaningful snapshots of our life, and thus provides us unprecedented potential to make advisable recommendations that can fully match users' intention in both spatial and activity

Regular Paper
Special Section on Data Management and Data Mining
This work was supported by the National Natural Science Foundation of China under Grant Nos. 61073061, 61303019, 61003044, 61232006, 61472263, 61402312, and 61402313, the Doctoral Fund of Ministry of Education of China under Grant No. 20133201120012, and Jiangsu Provincial Department of Education under Grant No. 12KJB520017.

*Corresponding Author

domains. To fulfill this purpose, some efforts[10-11] have been made in recent years to support efficient activity trajectory search, e.g., to return k activity trajectories that can cover the intended activities with the minimum distance. However, given an activity, they only judge if it can be carried out at the location (i.e., a place of interest) of a trajectory or not, without considering the actual user experience based on historical records. Obviously, this is not reasonable in most cases, because users do expect to fulfill their intended activities with the best experience. Having observed the limitation of previous work, we investigate a novel problem of rating-based activity trajectory search, to maximize users' experience on their intended activities.

Consider the example demonstrated in Fig.1, Table 1 and Table 2, where τ_1, τ_2 and τ_3 are historical activity trajectories. Each activity trajectory (e.g., τ_2) passes several locations (e.g., $p_{2,1}$, $p_{2,2}$ and $p_{2,3}$), and each location has one or more capable activities (e.g., c for $p_{2,2}$) to perform, and a corresponding rating score (e.g., 9) marked by previous visitors. A user specifies a query q with a start point S, a destination E, a set of intended activities $\{e, c, a\}$ and a threshold of distance 25 km. Clearly, τ_3 is not a choice since the goal activities cannot be fulfilled in it. Existing activity trajectory search method[10] returns τ_1 as the result, since the user can fulfill (e, c, a) in the trip $(S, p_{1,2}, p_{1,3}, p_{1,4}, E)$ with the minimum distance 14 km. However, we can easily observe that τ_2 is a more satisfactory choice than τ_1, because it has much higher rating score (27.1 vs 8.8) and the similar distance (15.5 km vs 13.5 km) within the threshold 25 km, in the trip $(S, p_{2,2}, p_{2,3}, p_{2,4}, E)$. Obviously, most users prefer τ_2 to τ_1 in real applications.

Compared with conventional activity trajectory search, the problem proposed in this study is more challenging for two main reasons. Firstly, it is hard to design an index structure that can organize spatial distance, activities and rating scores in a seamless fashion. Secondly, it is difficult to find a balance between different factors (i.e., distance, activity and rating score). If we focus on minimizing the distance while searching activity trajectories, as [10] does, the rating score could be low. Similarly, if the activity trajectory traversal is made for those with higher rating score first, the travel distance could be larger than the threshold. That means, it tends to incur much larger search space and thus causes severe efficiency issue if we just focus on a certain factor. As far as we know, existing methods are not applicable in this problem, since their indexing mechanism and trajectory search algorithms cannot fully support this work.

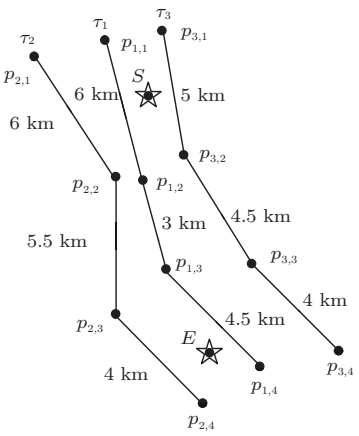

Fig.1. Example of trajectory matching.

Table 1. Rating Information

Trajectory	POI	Activity	Rating
τ_1	$p_{1,1}$	e, d	$e.r$: 2.5, $d.r$: 3.9
	$p_{1,2}$	c, d	$c.r$: 2.2, $d.r$: 2.7
	$p_{1,3}$	a	$a.r$: 3.0
	$p_{1,4}$	b, e	$b.r$: 2.6, $e.r$: 3.6
τ_2	$p_{2,1}$	d	$d.r$: 7.5
	$p_{2,2}$	c	$c.r$: 9.0
	$p_{2,3}$	a, b	$a.r$: 8.8, $b.r$: 9.5
	$p_{2,4}$	d, e	$d.r$: 9.1, $e.r$: 9.3
τ_3	$p_{3,1}$	a	$a.r$: 6.7
	$p_{3,2}$	c, d	$c.r$: 5.9, $d.r$: 8.8
	$p_{3,3}$	a	$a.r$: 9.9
	$p_{3,4}$	c	$c.r$: 3.1

Note: POI: point of interest.

To answer the query efficiently, we propose the following methods. Firstly, we devise a new hybrid index called AC-tree to organize trajectory segments and activities. By retrieving the AC-tree, we prune the search space efficiently and obtain a small subset of trajectories as candidates. Secondly, to achieve effective pruning effect in trajectory search, a novel method is pro-

Table 2. Euclidean Distance (km)

	$p_{1,1}$	$p_{1,2}$	$p_{1,3}$	$p_{1,4}$	$p_{2,1}$	$p_{2,2}$	$p_{2,3}$	$p_{2,4}$	$p_{3,1}$	$p_{3,2}$	$p_{3,3}$	$p_{3,4}$
S	3	4.0	7	13	5	4	9.0	13	3	3	8	14.0
E	12	6.5	3	2	13	7	3.5	2	12	7	4	4.5

posed to compute the maximum rating for each candidate trajectory. Thirdly, an optimized index RAC$^+$-tree is developed to achieve higher query performance. Different from the AC-tree, the rating information is considered in the RAC$^+$-tree. We can further prune the search space with the RAC$^+$-tree, and the cardinality of the new candidate set is much more smaller than that obtained by retrieving the AC-tree. In addition, a collaborative algorithm is proposed to make computing faster. Lastly, we extend the query to be order-sensitive by taking into account the order of query activities, as the users may want to perform their intended activities in an order.

To sum up, the main contributions of this work are as follows:

• We formulate a novel problem of trajectory search with spatial information, activities and rating scores. We also extend the query in an order-sensitive scenario by taking the order of both query activities and trajectories into consideration.

• We develop a hybrid index structure called AC-tree to organize trajectory segments and activity information. To achieve higher query performance, we design an optimized index structure RAC$^+$-tree, by traversing which a larger number of non-relevant trajectories are pruned.

• In order to answer the query efficiently, we propose novel algorithms to compute the maximum rating within the threshold of distance.

• We conduct extensive experiments based on real trajectory datasets to study the performance of the proposed index structures and algorithms. The experimental results demonstrate the efficiency and scalability of the proposed approaches.

The rest of this paper is organized as follows. In Section 2, we briefly view existing work related to activity trajectory search. Section 3 presents the problem statement and necessary notations in this work. We introduce the index tree in Section 4 and algorithms in Section 5, which is followed by the optimization of the proposed index structures and algorithms in Section 6. The update of hybrid index is presented in Section 7. In Section 8, we report the experimental results. This paper is concluded in Section 9.

2 Related Work

In this section, we review existing work on trajectory search and present some representative contributions on spatial keyword search, and the studies that consider the fusion of them are also discussed.

In the past decade, trajectory search has received significant attentions. Existing work[6-9,12] focused on searching trajectories in spatial-temporal domain. Vlachos *et al.*[6] investigated the problem of discovering similar trajectories that are modeled as a sequence of consecutive locations in a multidimensional Euclidean space. Chen *et al.*[7] introduced the edit distance on real sequence (EDR) for moving object trajectories. Chen *et al.*[8] tackled the problem of searching trajectories by locations in spatial domain, where multiple locations are used as the query, and similarity functions are developed to score how well a trajectory connects the query points. A transfer network was established to discover popular routes from historical trajectories in [9]. Given a location to a destination, the network is used to compute the transfer probability for the transfer nodes. Lastly, the most popular route that has the highest probability is returned as the result.

Meanwhile, with the prevalence of spatial web objects on the Internet, a lot of work appears in spatial keyword search. Existing work[10,13-18] addressed the problem of spatial keywords search by utilizing spatial and textual information. Zhou *et al.*[13] proposed a hybrid index that integrates inverted files and R*-trees to handle both textual and location aware queries. Cao *et al.*[16] investigated a problem of retrieving a group of spatial web objects, where the query keywords are covered by the groups' keywords, and the objects are nearest to the query locations. A distance owner-driven approach was proposed in [18], where an existing cost measurement called the maximum sum cost is used to guarantee the algorithms are near-to-optimal solutions.

In order to address spatial keywords search efficiently, some hybrid index structures were proposed[15,19-22]. The indexing strategy KR*-tree proposed in [15] was constructed in a way similar to how an R*-tree was constructed with minimal overhead in handling the keywords. The queries are answered in a two-step filtering process, space followed by text, with the KR*-tree. De Felipe *et al.*[20] introduced a new index called IR^2-tree, which combines the R-tree with superimposed text descriptions. Furthermore, the variant index MIR^2-tree is designed to overcome some drawbacks of the IR^2-tree. Cong *et al.*[22] proposed an IR-tree, which is an integration of the R-tree and inverted files. Each node of the tree records the location and the text information of all objects in the sub-tree. By traversing the IR-tree, the search space can be pruned efficiently with spatial information and textual descriptions. The IR-tree was applied in [16] to organize the objects. Each

leaf node of the IR-tree contains the bounding rectangle and an identifier, and each non-leaf node is associated with the minimum bounding rectangle and the text description of the child nodes.

[10] is the most similar work to our query, where both of spatial distance and keywords information are considered while searching trajectories. Multiple locations with a set of keywords are used as the query to search trajectories that have the minimum match distance with respect to the query. A novel grid index GAT, which consists of four components (i.e., HICL, ITL, TAS, APL), is presented to organize trajectory segments and activities, and novel algorithms are proposed to tackle the problem with high efficiency. The query is also extended to be order-sensitive and take both the order of query activities and trajectories into account. Despite the great contributions made by [10], it does not take the rating scores into account during processing. Hence, both the index structure and the algorithms of [10] are not suitable for our problem.

In spite of the significant contributions made by the aforementioned work, none of them take rating scores into consideration, which is an important new feature of trajectories. Meanwhile, the hybrid indexes and the corresponding algorithms proposed in their work cannot be adapted directly to our problem. As a consequence, we propose novel indexes and algorithms in this study.

3 Problem Definition

In this section, we present all the definitions used throughout the paper and formulate the problem. Before that, the notations used in this paper are summarized in Table 3.

Table 3. Definitions of Notations

Notation	Definition
α	Activity
$\alpha.r$	Rating of activity α
τ	Activity trajectory
θ	Upper bound of POIs in a grid
$\mathcal{D}$	Set of τ
$\widehat{d}$	Threshold of distance
$p.\varphi$	Set of activities of POI p
ω	Trajectory matching
ω^o	Order-sensitive trajectory matching
$r(\omega)$	Rating of a trajectory matching ω
$d(\omega)$	Distance of a trajectory matching ω

Note: POI: point of interest.

We use α to represent a type of activity (e.g., shopping, sport and entertaining) that the tourist can take at a location. The rating score of the activity α is defined as $\alpha.r$.

Definition 1 (Activity Trajectory). *Let $p = (x, y, \varphi, r)$ be a POI (point of interest) where x is the longitude, y is the latitude, φ denotes the set of activities that can be performed in the location (x, y), and r is a set of rating information of φ. An activity trajectory consists of a sequence of POIs, denoted as $\tau = (p_1, p_2, \ldots, p_n)$.*

For the sake of simplicity, we use trajectory to denote activity trajectory in the sequel, if no ambiguity can be caused.

Definition 2 (Sub-Trajectory). *Given a trajectory $\tau = (p_1, p_2, \ldots, p_n)$, a sub-trajectory of τ is $\tau_s^e = (p_s, p_{s+1}, \ldots, p_e)$, where $1 \leqslant s \leqslant e \leqslant n$, denoted as $\tau_s^e \subseteq \tau$.*

Definition 3 (Trajectory Matching). *Let $q = (S, E, \varphi, \widehat{d})$ be a query, where S is the start point of a trip, E denotes the destination, φ represents a set of activities needed to be performed, and the threshold of distance is $\widehat{d}$. Given a trajectory τ, a tuple $\omega = (\tau, q)$ is defined as a trajectory matching if there exists a sub-trajectory $\tau_s^e \subseteq \tau$, such that $q.\varphi \subseteq \bigcup_{p_i \in \tau_s^e} p_i.\varphi$.*

Definition 4 (Distance of Trajectory Matching). *Given a trajectory matching $\omega = (\tau, q)$, the distance between τ and q is defined as:*

$$d(\tau, q) = dis(S, p_1) + \sum_{i=1}^{n-1} dis(p_i, p_{i+1}) + dis(E, p_n),$$

where $\tau = (p_1, p_2, \ldots, p_n)$ and $dis()$ is to get the Euclidean distance between two locations.

Definition 5 (Rating of Trajectory Matching). *Given a trajectory matching $\omega = (\tau, q)$, suppose τ_1, τ_2, ..., τ_m are part of $\tau's$ sub-trajectories, and each of them is a matching to the query q, denoted as $\omega_i = (\tau_i, q)$, then the rating of the trajectory matching is defined as:*

$$r(\omega, \widehat{d}) = \max_{d(\omega_i) \leqslant \widehat{d}} \left(\sum_{\alpha \in q.\varphi} \max_{p_j \in \tau_i} (p_j.\alpha.r) \right). \quad (1)$$

Consider the running example in Fig.1: τ_3 is not a match to the query q, $r((\tau_1, q), \widehat{d}) = \max\{r(((p_{1,2}, p_{1,3}, p_{1,4}), q), 25), \quad r(((p_{1,1}, p_{1,2}, p_{1,3}), q), 25)\} = 8.8$ and the distance of the trip is $d((p_{1,2}, p_{1,3}, p_{1,4}), q) = 13.5$ km. In trajectory τ_2, $r((\tau_2, q), \widehat{d}) = r(((p_{2,2}, p_{2,3}, p_{2,4}), q), 25) = 27.1$, and the distance of this trip is 15.5 km.

Trip Oriented Search on Activity Trajectory (*TOSAT*). Given a set of trajectories $\mathcal{D}$ and a query

q, let $\mathcal{C}(\mathcal{C} \subseteq \mathcal{D})$ be a candidate set, which means for any trajectory $\tau \in \mathcal{C}$, it follows that (τ, q) is a trajectory matching. Given a positive integer k, the TOSAT returns a set $\mathcal{R}(\mathcal{R} \subseteq \mathcal{C}, |\mathcal{R}| = k)$, such that $\forall \tau \in \mathcal{R}$ and $\forall \tau' \in \mathcal{C} - \mathcal{R}$ it follows that $r((\tau, q), \widehat{d}) \geqslant r((\tau', q), \widehat{d})$.

4 Hybrid Index: AC-Tree

It is challenging to search trajectories by utilizing both spatial distance and activities, especially when the cardinality of the given trajectory dataset is large. As a result, we devise a hybrid index called AC-tree to prune the search space.

As depicted in Fig.2(a), τ_0, τ_1 and τ_2 are historical trajectories, and the details of them are presented in Table 4. Due to the uneven distribution of POIs in practice, the search regions are partitioned into different cells based on the number of POIs in each grid cell. If the number of POIs attached to a cell exceeds a threshold θ, the cell should be partitioned into four components. θ is set to 2 in this example. For the sake of convenience, the cell_id of the i-th sub-cell of current cell is defined as $4 \times current\_cell\_id + i$. As seen from Fig.2(a), the number of POIs in cell 1 is 3 (i.e., p_0, p_1 and p_4), and then the cell is divided into four partitions. Repeating the action of dividing, we can create an AC-tree. In particular, each tree node corresponds to a grid cell. The details of AC-tree are illustrated as follows.

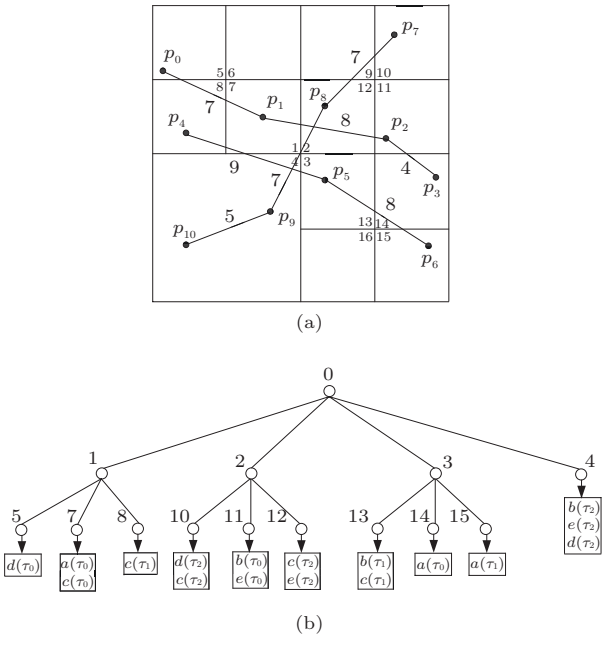

Fig.2. Details of AC-tree. (a) Search regions. (b) AC-tree.

Table 4. Trajectory Information

Trajectory	POI	Activity	Rating
τ_0	p_0	d	$d.r : 9$
	p_1	a, c	$a.r : 7,\ c.r : 8.5$
	p_2	b, e	$b.r : 2,\ e.r : 8.5$
	p_3	a	$a.r : 8$
τ_1	p_4	c	$c.r : 9$
	p_5	b, c	$b.r : 4,\ c.r : 9$
	p_6	a	$a.r : 3$
τ_2	p_7	d, c	$d.r : 7,\ c.r : 6$
	p_8	c, e	$c.r : 7,\ e.r : 6.5$
	p_9	b	$b.r : 5$
	p_{10}	e, d	$e.r : 3,\ d.r : 8$

Non-Leaf Nodes. Each non-leaf node of the AC-tree contains the coordinates of four vertices of the cell and the pointers to their child nodes. As seen from Fig.2, the tree node 3 corresponds to the grid cell 3. If one cell is divided and the number of POIs of its i-th child is 0, then the pointer to the i-th child is null. As a result, tree node 3 contains three child nodes, since there is no POI in the grid cell 16.

Leaf Nodes. Each leaf node of the AC-tree is used as an inverted file to store the trajectory information and activities. From Fig.2(b), the activities of POIs and the trajectories which they belong to are stored in the corresponding leaf node. In Fig.2(a), the trajectory τ_2 contains activities (a, b, c) in the grid cell 4. As a result, leaf node 4 should store this information.

Next, we develop an approach to construct the AC-tree. As presented in Algorithm 1, we propose a queue $\mathcal{L}$ to store the tree nodes that need to be visited. For each process, if the number of POIs in a tree node exceeds the threshold θ, it needs to be split (line 5). Otherwise, we create a leaf node by adding the activity and trajectory information into the tree node (line 9).

Algorithm 1. Construct the AC-Tree
Input: a trajectory dataset $\mathcal{D}$
Output: an AC-tree
1: Create the root node *Rnode* based on the grid partition;
2: $\mathcal{L}$.Enqueue(*Rnode*);
3: **while** $\mathcal{L}$ is not empty **do**
4: *treenode* $\leftarrow$ $\mathcal{L}$.Dequeue;
5: **if** *treenode* should be split **then**
6: Create the child nodes for *treenode*;
7: $\mathcal{L}$.Enqueue(the created new nodes);
8: **else**
9: Create the inverted file for *treenode*;
10: **end if**
11: **end while**
12: **return** AC-tree;

5 Algorithms of TOSAT and OTOSAT

To the best of our knowledge, there is the first work on trajectory search considering activities, spatial distance and rating information simultaneously. Given a query q and a trajectory dataset $\mathcal{D}$, the method of TOSAT consists of two steps: firstly, traversing the AC-tree to get a candidate set $\mathcal{C}$; secondly, computing the rating of trajectory matching for each candidate trajectory in $\mathcal{C}$. A baseline algorithm is proposed to prune the search space and get a candidate set, and the details of the algorithm are presented as follows.

5.1 Traversing AC-Tree

In this subsection, a new approach is developed to prune search space and get a candidate set $\mathcal{C}$. We commence this part by traversing the AC-tree beginning at the root node with the breadth-first strategy. The details are illustrated in Algorithm 2.

Algorithm 2. Traversing Index Tree
Input: query q, AC-tree *tree*
Output: candidate set $\mathcal{C}$

```
1:  Inserting the root node of tree into l
2:  M[*, *] ← 0
3:  while l ≠ φ do
4:      g ← the first entry of l
5:      if mdist(g, q) ⩽ d̂ then
6:          if g is a non-leaf node then
7:              Insert all child nodes of g into l;
8:          else
9:              Update matrix M;
10:         end if
11:     end if
12:     Remove the first entry of l;
13: end while
14: for j ← 1 to |D| do
15:     if all p_{i,j} = 1 (i ∈ [1, |q.φ|]) then
16:         Put trajectory τ_j into C;
17:     end if
18: end for
19: return C;
```

Given a grid cell g and a query q, the minimum distance between them is defined as:

$$mdist(g,q) = \min_{p \in g}\{dis(S,p) + dis(E,p)\}, \qquad (2)$$

where p is a POI contained by the cell g and $dis(S,p) + dis(E,p)$ is the distance between the queries q and p, denoted as $dis(p,q)$.

Lemma 1. *Given a grid cell g and a query q, if $mdist(g,q) > \widehat{d}$, the cell g is pruned.*

Proof. According to (2), $mdist(g,q)$ represents the minimum distance between q and g, and for each POI p contained in g, it holds that $dis(p,q) \geqslant mdist(g,q)$. Hence, if $mdist(g,q) > \widehat{d}$, we have $dis(p,q) > \widehat{d}$. As a consequence, if the minimum distance between q and g is larger than $\widehat{d}$, there is no need to consider all POIs contained by the grid cell g, and we can prune it directly. □

Assuming the query activities are $\{\alpha_1, \alpha_2, \ldots, \alpha_m\}$, we define a matrix $\mathcal{M}$, each element $p_{i,j}$ of which denotes whether the trajectory j contains the query activity α_i.

$$\mathcal{M} = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \vdots & \vdots & & \vdots \\ p_{m,1} & p_{m,2} & & p_{m,n} \end{pmatrix},$$

where

$$p_{i,j} = \begin{cases} 1, & \text{if trajectory } j \text{ contains } \alpha_i, \\ 0, & \text{otherwise,} \end{cases}$$

and $n = |\mathcal{D}|$.

The retrieval of candidate trajectories consists of two steps, which are illustrated in Algorithm 2.

Step 1. The algorithm maintains an FIFO queue l to store the nodes that should be visited. At the beginning, inserting the root node of AC-tree into l and the matrix $\mathcal{M}$ is initialized to zero. For each loop, if the minimum distance between a node g and a query q is smaller than $\widehat{d}$, we insert the child nodes of g into list l, or update the matrix $\mathcal{M}$ based on the list attached to g, if it is a leaf node (line 9). In addition, if the distance $dis(q,g)$ is larger than $\widehat{d}$, we can prune the region based on Lemma 1.

Step 2. After traversing the AC-tree, we can obtain an updated matrix $\mathcal{M}$. By processing it, we can get a candidate trajectory dataset $\mathcal{C}$ (lines 14∼18). For each trajectory τ_j in $\mathcal{D}$, if all $p_{i,j} = 1$, it will be a candidate since all activities of $q.\varphi$ are likely to be fulfilled in τ_j within the threshold of distance.

5.2 Computing the Rating of Trajectory Matching

For each trajectory τ in $\mathcal{C}$, a straightforward way to compute $r(\omega, \widehat{d})$ is to find out all possible sub-trajectories that match query q and compute $r(\omega_i, \widehat{d})$, and then the algorithm returns the maximum $r(\omega_i, \widehat{d})$

as the result. However, the computation complexity of this method is too high. In the sequel, a more efficient algorithm is proposed to resolve the problem.

Given a query q, a threshold of distance $\widehat{d}$ and a candidate trajectory $\tau = (p_1, p_2, \ldots, p_n)$, the details of computing $r(\omega, \widehat{d})$ are illustrated in Algorithm 3 which has two steps.

Algorithm 3. Computing the Rating of Trajectory Matching
Input: query q, trajectory τ
Output: $r(\omega, \widehat{d})$
1: Create a link list l;
2: $r(\omega, \widehat{d}) \leftarrow 0$, $s \leftarrow 1$, $e \leftarrow 1$;
3: **while** $e \leqslant l.length$ **do**
4: **if** τ_s^e is a match to q and $d(\tau_s^e, q) \leqslant \widehat{d}$ **then**
5: Compute $r((\tau_s^e, q), \widehat{d})$ based on (1);
6: **if** $r((\tau_s^e, q), \widehat{d}) > r(\omega, \widehat{d})$ **then**
7: $r(\omega, \widehat{d}) \leftarrow r((\tau_s^e, q), \widehat{d})$;
8: **end if**
9: **else**
10: $s \leftarrow s + 1$;
11: Continue;
12: **end if**
13: $e \leftarrow e + 1$;
14: **end while**
15: **return** $r(\omega, \widehat{d})$;

Step 1. We commence the algorithm by creating a list l to store the POIs that need to be visited (line 1). For each POI p_i in trajectory τ, p_i is inserted into l if $p_i.\varphi \wedge q.\varphi \neq \phi$ and $dis(q, p_i) \leqslant \widehat{d}$. Compared with using the whole trajectory τ, the "dead" POIs that contain no query activity or beyond the threshold of distance $\widehat{d}$ are pruned in this approach. As a consequence, the computation cost is reduced since less POIs are taken into account. The method is efficient especially when the number of POIs in each candidate trajectory is large.

Step 2. In this subsection, s and e are used to denote the start point and the end point of a sub-trajectory respectively. For each loop, if all query activities can be fulfilled in current sub-trajectory within the threshold of distance, we compute $r((\tau_s^e, q), \widehat{d})$ based on (1) and update $r(\omega, \widehat{d})$ if necessary, according to the distance between τ_s^e and q (lines 4∼8).

5.3 Computing the Rating in Order-Sensitive Situation

In many real applications, users may want to perform their activities with an order. For instance, a user plans to go shopping after having a haircut, i.e., the order of activities is *haircut* → *shopping*. Obviously, TOSAT is not applicable in this scenario. Consequently, we extend the query to be order-sensitive, and develop novel algorithms to address the new problem.

Definition 6 (Order-Sensitive Trajectory Matching). *Given a trajectory matching $\omega = (\tau, q)$ and a threshold of distance $\widehat{d}$, $\omega = (\tau, q)$ is called an order-sensitive trajectory matching, denoted as $\omega^o = (\tau, q)^o$, on condition that there exists $\omega_i = (\tau_i, q)$ and for any pair of query activities $(\alpha_i, \alpha_j,\ i < j)$ of $q.\varphi$, there exist p_m and p_n $(m \leqslant n)$ of τ_i, such that $\alpha_i \in p_m.\varphi$ and $\alpha_j \in p_n.\varphi$. The rating of $\omega^o = (\tau, q)^o$ is defined as:*

$$r(\omega^o, \widehat{d}) = \max_{d(\omega_i^o) \leqslant \widehat{d}} \Big(\sum_{\alpha \in q.\varphi} \max_{p_j \in \tau_i} (p_j.\alpha.r) \Big),$$

where $d(\omega_i^o) = d(\omega_i)$.

Order-Sensitive Trip Oriented Activity Trajectory Search (*OTOSAT*). Given a trajectory dataset $\mathcal{D}$, a query q and a positive integer k, OTOSAT returns a set $\mathcal{R}$ $(R \subseteq \mathcal{D}, |\mathcal{R}| = k)$ such that $\forall \tau_i \in \mathcal{R}$ and $\forall \tau_j \in \mathcal{D} - \mathcal{R}$ it holds that $r(\omega_i^o, \widehat{d}) \geqslant r(\omega_j^o, \widehat{d})$.

Consider the example in Fig.1 again, assuming the orders of τ_1 and τ_2 are $p_{1,1} \rightarrow p_{1,2} \rightarrow p_{1,3} \rightarrow p_{1,4}$ and $p_{2,1} \rightarrow p_{2,2} \rightarrow p_{2,3} \rightarrow p_{2,4}$. If the order of query activities is $e \rightarrow c \rightarrow a$, the sub-trajectory $(p_{1,1}, p_{1,2}, p_{1,3})$ is the only alternative. This is because the order of performing (e, c, a) in the trips $(S, p_{1,2}, p_{1,3}, p_{1,4}, E)$ and $(S, p_{2,2}, p_{2,3}, p_{2,4}, E)$ does not keep the order of τ_1 and τ_2.

The index structure AC-tree is also constructed for OTOSAT to organize trajectory segments and activity information. Same with TOSAT, OTOSAT is composed of two components: 1) traversing the hybrid index AC-tree to get a candidate set $\mathcal{C}$; 2) computing the rating of trajectory matching for each candidate in $\mathcal{C}$, and the k trajectories with the maximum rating are returned as the results. Algorithm 2 is still adopted in this part to prune search space with spatial and activity information. However, computing the rating of trajectory matching in order-sensitive case is more challenging, as it needs to make the order of performing activities in a trajectory consistent with the query, and try to maximize the rating within the threshold of distance. Given a candidate trajectory τ, a naive method of OTOSAT is to enumerate all possible sub-trajectory matches and find the maximum $r(\omega_i^o, \widehat{d})$. Clearly, this is not efficient. A novel approach is illustrated in the sequel.

Given a trajectory $\tau = (p_1, p_2, \ldots, p_n)$ and a query q, let $q.\varphi = (\alpha_1, \alpha_2, \ldots, \alpha_m)$. We define an $m \times n$ matrix $\mathcal{R}$ such that its element $\mathcal{R}[i, j] (1 \leqslant i \leqslant m,\ 1 \leqslant$

$j \leqslant n$) denotes the maximum rating between the sub-query $q_1^i.\varphi = (\alpha_1, \alpha_2, \ldots, \alpha_i)$ and the sub-trajectory $\tau_1^j = (p_1, p_2, \ldots, p_j)$. The element of $\mathcal{R}$ is given as follows:

$$\mathcal{R}[i,j] = \max_{1 \leqslant k \leqslant j} \{\mathcal{R}[i-1,k] + mr(\alpha_i, \tau_k^j)\}, \quad (3)$$

where $mr(\alpha_i, \tau_k^j)$ is the maximum rating of α_i in sub-trajectory τ_k^j.

As presented in (3), $\mathcal{R}[i,j]$ is derived by maximizing the rating between the sub-trajectory τ_1^k and the sub-query $q_1^{i-1}.\varphi$ plus the maximum rating of query activity α_i from p_k to p_j. Note that, the element $\mathcal{R}[m,n]$ holds the maximum rating between the query q and the given candidate. It is guaranteed to be an order-sensitive query based on (3).

Computing the rating of trajectory matching in the order-sensitive case is illustrated in Algorithm 4, which consists of two components. Firstly, we propose a link list l to store the POIs that need to be computed, and the construct of l is the same with that in Algorithm 3. Secondly, the maximum rating within $\widehat{d}$ is computed. For each process, if the query activities can be performed in current sub-trajectory with an order, and the distance of the trajectory matching is less than $\widehat{d}$, the algorithm updates matrix $\mathcal{R}(*,*)$ according to (3) (lines 4∼6). In addition, the returned result $r(\omega^o, \widehat{d})$ is updated if necessary (lines 7∼8).

Algorithm 4. Computing the Rating in Order-Sensitive Case

Input: query q, trajectory τ

Output: $r(\omega^o, \widehat{d})$

```
1:  Create a link list l;
2:  R(*,*) ← 0, s ← 1, e ← 1, r(ω^o, d̂) ← 0;
3:  while e ⩽ l.length do
4:      if d(ω_s^e, q) ⩽ d̂ and τ_s^e is a order-sensitive match to q then
5:          R(*,*) ← 0;
6:          Update each R[i,j] based on (3)
7:          if R[|q.φ|,|τ|] > r(ω^o, d̂) then
8:              r(ω^o, d̂) ← R[|q.φ|,|τ|];
9:          end if
10:     else
11:         s ← s + 1;
12:         Continue;
13:     end if
14:     e ← e + 1;
15: end while
16: return r(ω^o, d̂);
```

6 Optimization

As discussed in Section 5, we prune the search space efficiently with AC-tree by utilizing the spatial distance between a tree node and a query. Even though this method is capable of achieving high query performance, we can further improve the efficiency with rating information. The optimized approaches are developed in the sequel.

6.1 Enhanced Index: RAC$^+$-Tree

Note that some search regions are pruned with the AC-tree based on Theorem 1. However, we need to traverse all regions within $\widehat{d}$ even though the region contains no query activity, and compute the rating of trajectory matching for each trajectory in $\mathcal{C}$. It is costly, especially when the cardinality of $\mathcal{C}$ is large. Having observed the drawbacks of AC-tree, we design an enhanced hybrid index RAC$^+$-tree to organize trajectory data, which is a variant of the AC-tree.

As depicted in Fig.3, the leaf nodes of RAC$^+$-tree contain not only activity information but also rating information, and the minimum distance to conduct two activities. The entry of a list attached to a leaf node is a tuple in the form of $\alpha(\tau_i : \alpha.r)$ and (α, β, dis), where τ_i is the trajectory that contains α, $\alpha.r$ represents the maximum rating of α in current leaf node in trajectory τ_i, and dis is the minimum distance to conduct α and β in current grid cell. As shown in Fig.2(a) and Table 4, cell 4 contains activities (b, e, d), the maximum ratings of which are $(5, 3, 8)$, and the distance between p_9 and p_{10} is 5, and thus we insert this information into the leaf node 4.

For each non-leaf node of RAC$^+$-tree, it contains all activities that can be fulfilled in its child nodes, and the minimum distance to conduct two activities is updated based on its child nodes. All lists of the RAC$^+$-tree are presented in Table 5. Shown in Fig.2(a), the activities that can be performed in the grid cell 3 are (a, b, c), the minimum distance to conduct b and c obtained from the leaf nodes 13, 14 and 15 is 0, and then this information is inserted into the node 3. Note that, the tree node 4 holds the entry $(b, e, 5)$ and list 2 contains $(b, e, 0)$; thus the minimum distance to conduct b and e in the root node is 0, and list 0 holds this information. Besides, the definition of each element of matrix $\mathcal{M}$ is changed:

$$p_{i,j} = \begin{cases} \alpha_i.r, & \text{if trajectory } j \text{ contains } \alpha_i, \\ 0, & \text{otherwise,} \end{cases}$$

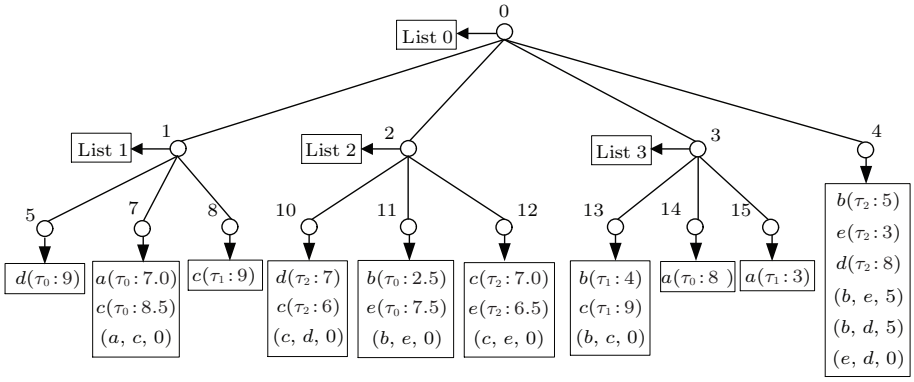

Fig.3. RAC$^+$-tree.

where $\alpha_i.r$ is the maximum rating of the query activity α_i in trajectory j within $\widehat{d}$.

Table 5. Lists of Non-Leaf Nodes

List	Information
List 0	(a,c,d,e), $(a,c,0)$, $(b,c,0)$, $(c,d,0)$
	$(b,d,5)$, $(b,e,0)$, $(e,d,0)$, $(c,e,0)$
List 1	(a,c,d), $(a,c,0)$
List 2	(b,c,d,e), $(c,d,0)$, $(b,e,0)$, $(c,e,0)$
List 3	(a,b,c), $(b,c,0)$

Compared with using AC-tree, the query is more efficient with the RAC$^+$-tree. On one hand, we can further prune the search space with activity information, and the tree nodes that contain no query activity are pruned. On the other hand, the search regions are pruned with spatial distance. Given a tree node p and a grid cell g, the tree nodes rooted at p are pruned if $dis(p,g) > \widehat{d}$ according to Lemma 1. Furthermore, we can use the minimum distance between different activities to prune the search space. Given a query q with $q.\varphi = (\alpha,\beta,\gamma)$, and a tree node p with the entries $(\alpha,\beta,dis1)(\beta,\gamma,dis2)$ in its list, p is pruned if $dis(q,p) + dis1 + dis2 > \widehat{d}$. The pruning is reasonable, since the minimum distance to conduct (α,β,γ) in p is $dis(q,p)+dis1+dis2$, and there is no entry $(\alpha,\gamma,dis3)$ in the list of p, which means that the minimum distance to conduct α and β in p is larger than $dis1$ and $dis2$. For each process, the element $p_{i,j}$ is updated based on the new inverted files attached to the leaf nodes. After traversing the RAC$^+$-tree, we get a new matrix $\mathcal{M}$, the process of which is presented in Algorithm 5.

As seen in Algorithm 5, we can obtain a candidate set $\mathcal{C}$ by computing the matrix $\mathcal{M}$ (lines 1∼5). Then we can get a degressive $\mathcal{C}$ by sorting all candidates according to $\sum_{i=1}^{|q.\varphi|} p_{i,j}$ (lines 6∼7).

Algorithm 5. Compute the Matrix $\mathcal{M}$
Input: a matrix $\mathcal{M}$
Output: candidate set $\mathcal{C}$
1: **for** $j \leftarrow 1$ *to* $|\mathcal{D}|$ **do**
2: **if** all $p_{i,j} > 0$ $(i \in [1, |q.\varphi|])$ **then**
3: Put trajectory τ_j into $\mathcal{C}$;
4: **end if**
5: **end for**
6: Compute $\sum_{i=1}^{|q.\varphi|} p_{i,j}$ for each trajectory in $\mathcal{C}$;
7: Sort all trajectories in $\mathcal{C}$ by descending based on $\sum_{i=1}^{|q.\varphi|} p_{i,j}$;
8: **return** $\mathcal{C}$

Lemma 2. *For each candidate trajectory τ_j in $\mathcal{C}$, it holds that $r((\tau_j,q),\widehat{d}) \leqslant \sum_{i=1}^{|q.\varphi|} p_{i,j}$.*

Proof. According to the new definition of $p_{i,j}$, it denotes the maximum rating of query activity α_i in trajectory τ_j within $\widehat{d}$, and thus the upper bound of the rating of query activities in candidate τ_j is $\sum_{i=1}^{|q.\varphi|} p_{i,j}$, which is larger than the real rating $r((\tau_j,q),\widehat{d})$. □

Lemma 3. *Given a candidate set $\mathcal{C} = \{\tau_1,\tau_2,\ldots,\tau_n\}$, if the current k-th maximum result $res[k]$ is larger than $\sum_{i=1}^{|q.\varphi|} p_{i,j}$, for any candidate τ_t in the remained set $\{\tau_{j+1},\ldots,\tau_n\}$, it holds that $res[k] \geqslant r((\tau_t,q),\widehat{d})$.*

Proof. For each trajectory τ_t in the set $\{\tau_{j+1},\ldots,\tau_n\}$, it holds that $\sum_{i=1}^{|q.\varphi|} p_{i,j} \geqslant \sum_{i=1}^{|q.\varphi|} p_{i,t}$, since all trajectories have been sorted by descending in Algorithm 5. As a result, if $res[k] \geqslant \sum_{i=1}^{|q.\varphi|} p_{i,j}$, it holds that $res[k] \geqslant \sum_{i=1}^{|q.\varphi|} p_{i,t} (j+1 \leqslant t \leqslant n)$, then we have $res[k] \geqslant r((\tau_t,q),\widehat{d})$ based on Lemma 2. □

In the new method, less candidate trajectories are considered. For each computation, the algorithm is terminated early, if $res[k] \geqslant \sum_{i=1}^{|q.\varphi|} p_{i,j}$ for the candidate trajectory τ_j, since there is no need to compute the rating for each candidate in the set $\{\tau_j,\tau_{j+1},\ldots,\tau_n\}$. The results are guaranteed to be reasonable based on Lemma 3.

6.2 Collaborative Algorithm for Computing Rating

In this subsection, we introduce a collaborative algorithm to compute the maximum rating within $\widehat{d}$. Different from Algorithms 3 and 4 that take all nodes of l into consideration, a different list $l' = \{(p_1, p_{11}), (p_2, p_{22}), \ldots, (p_n, p_{nn})\}$ is created, where p_i is the POI such that $p_i.\varphi \wedge q.\varphi \neq \phi$ and $dis(p_i, q) \leqslant \widehat{d}$, and p_{ii} is a tuple in the form of $(\alpha_{1r}, \ldots, \alpha_{mr})$, where α_{jr} denotes the maximum rating of the query activity α_j from p_i to the last node of l' and $q.\varphi = (\alpha_1, \ldots, \alpha_m)$.

Lemma 4. *Given a candidate trajectory τ and its corresponding list l', for each entry (p_i, p_{ii}) in l', it has the following properties:*

1) *if* $r((\tau, q), \widehat{d}) \geqslant p_{ii}.(\sum_{k=1}^{m} \alpha_{kr})$, $r((\tau, q), \widehat{d}) \geqslant p_{jj}.(\sum_{k=1}^{m} \alpha_{kr})(j > i)$;

2) *if the current k-th maximum result $res[k]$ is larger than $p_{ii}.(\sum_{k=1}^{m} \alpha_{kr})$, $res[k] \geqslant p_{jj}.(\sum_{k=1}^{m} \alpha_{kr})(j > i)$.*

Proof. Since $p_{ii}.(\sum_{k=1}^{m} \alpha_{kr})$ is the sum of the maximum rating of query activities from p_i to the last node of l', for any remained node $p_j (j > i)$ in list l', it holds that $p_{ii}.(\sum_{k=1}^{m} \alpha_{kr}) \geqslant p_{jj}.(\sum_{k=1}^{m} \alpha_{kr})$. As a consequence, if $r((\tau, q), \widehat{d}) \geqslant p_{ii}.(\sum_{k=1}^{m} \alpha_{kr})$, then $r((\tau, q), \widehat{d}) \geqslant p_{jj}.(\sum_{k=1}^{m} \alpha_{kr})$, and if $res[k] \geqslant p_{ii}.(\sum_{k=1}^{m} \alpha_{kr})(j > i)$, then we have $res[k] \geqslant p_{jj}.(\sum_{k=1}^{m} \alpha_{kr})$. □

For each process in the collaborative algorithm, a vector $\boldsymbol{vec}$ is used to prune the search space, where $vec[i]$ denotes the maximum rating of the query activity α_i in current sub-trajectory τ_s^e. Obviously, $\sum_{i=1}^{m} \max(vec[i], p_{ee}.\alpha_{ir})$ denotes the sum of the maximum rating of the query activities from p_s to the last node of l'. Consequently, if $r((\tau, q), \widehat{d}) \geqslant \sum_{i=1}^{m} \max(vec[i], p_{ee}.\alpha_{ir})$ or $res[k] \geqslant \sum_{i=1}^{m} \max(vec[i], p_{ee}.\alpha_{ir})$, the algorithm is terminated early. In addition, for each sub-trajectory τ_s^e, if there exists $p_{ss}.\alpha_{i,r} = 0$, which means that we cannot fulfill the query activity α_i in the remained nodes, the algorithm is terminated.

7 Update of Hybrid Index

In real scenario, we are faced with an inevitable problem. The volume of new trajectories generated by users becomes increasingly larger as time goes on, and it has great significance to take these trajectories into account, since they represent the popular preferences of users. Meanwhile, some trajectories become outdated, which means that there is no need to maintain them in the index structure. In order to make more reasonable recommendation, we update the proposed hybrid index structures by adding new trajectories and removing outdated trajectories in the sequel. Particularly, we just present the update of RAC$^+$-tree, as it covers the update of AC-tree.

Seen from Fig.4, we add a new trajectory τ_3, the details of which are presented in Table 6. Algorithm 6 is proposed to update the RAC$^+$-tree, while adding new trajectories. Note that we just process the tree node p that contains new POIs, and the process is threefold. 1) If p is a non-leaf node and there exists new POIs that fall into a certain child node of p, and the node contains no POI before, we create the child node for p (lines 5∼7). 2) If p is a leaf node, and the number of POIs in p is larger than θ after inserting new trajectories, we split p and create child nodes for it (lines 9∼10). 3) If p is a leaf node and there is no need to split p, we update the inverted file attached to p (line 12). In addition, the ancestors of p should be updated if necessary.

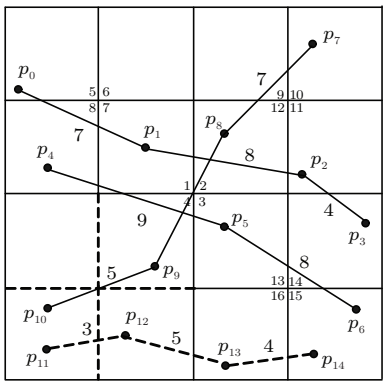

Fig.4. Adding a new trajectory.

Table 6. Inserted Trajectory

Trajectory	POI	Activity	Rating
τ_3	p_{11}	e, d	$e.r : 9,\ d.r : 7$
	p_{12}	a	$a.r : 6$
	p_{13}	b, d	$b.r : 3,\ d.r : 7$
	p_{14}	a	$a.r : 8$

As depicted in Fig.4 and Fig.5, the grid cell 4 should be split, as the number of POIs in which is larger than the threshold θ, whose value is 2. Then the tree nodes 18, 19 and 20 are created. In addition, the leaf node 16 is generated as there exists a new POI p_{13} which falls into the grid cell 16, and the inverted file attached to

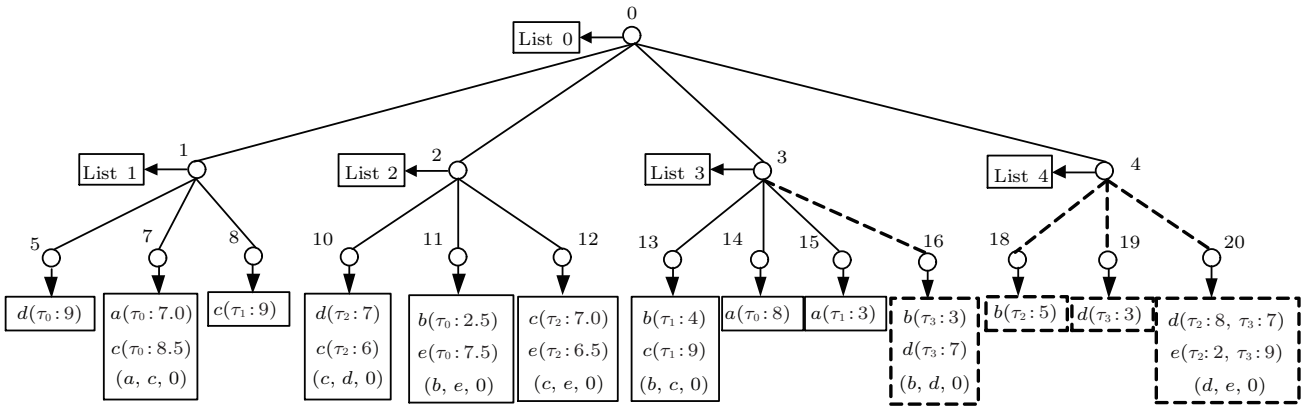

Fig.5. Updated RAC$^+$-tree after adding.

the leaf node 15 is updated based on the information contained by p_{14}. After creating the leaf node 16, the tree node 3 should be updated due to the appearance of the new activities d. Consequently, it needs to update the ancestors of present node if necessary (line 14).

Algorithm 6. Update RAC $^+$-Tree (Add)
Input: an RAC$^+$-tree
1: $\mathcal{L}$.Enqueue(*root node*);
2: **while** $\mathcal{L}$ is not empty **do**
3: $p \leftarrow \mathcal{L}$.Dequeue;
4: **if** p contains new POIs **then**
5: **if** p is a non-leaf node **then**
6: Create new child nodes for p if necessary;
7: $\mathcal{L}$.Enqueue(child nodes of p);
8: **else**
9: **if** p should be split **then**
10: Create child nodes for p;
11: **else**
12: Update the inverted file of p;
13: **end if**
14: Update the ancestors of p;
15: **end if**
16: **end if**
17: **end while**

Next, we will discuss the update of RAC$^+$-tree, while removing the outdated trajectories. As shown in Fig.6, we remove the historical trajectory τ_0. Algorithm 7 is developed to tackle the problem. For each process, the update of the tree node p that contains outdated POIs is two-fold. 1) If p is a leaf-node, we just update the inverted file of p. 2) If p is a non-leaf node and the number of POIs in p is less than θ after removing, we merge the child nodes of p.

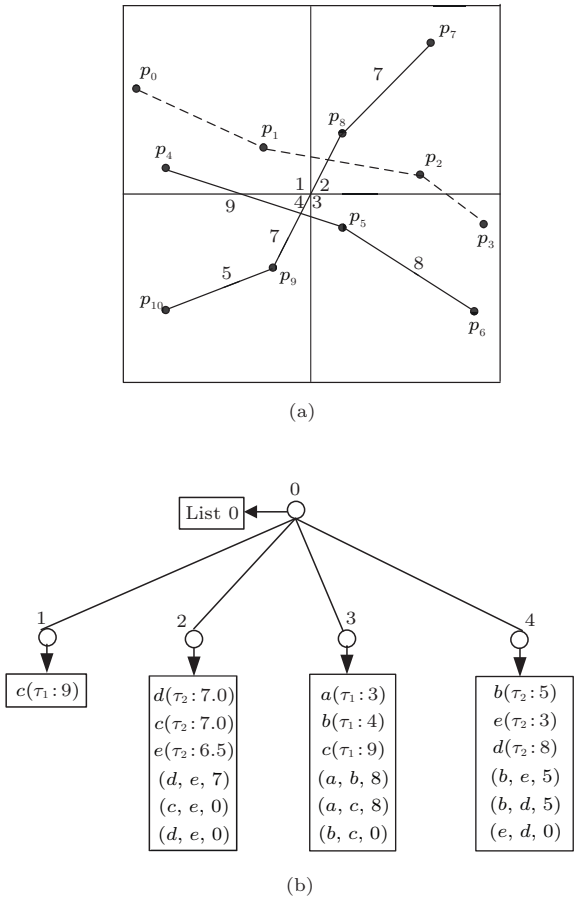

Fig.6. Example of removing a trajectory. (a) Search regions. (b) Updated RAC$^+$-tree.

Seen from Fig.6(a), the child nodes of grid cells 1, 2 and 3 are merged after removing τ_0, since the number of POIs contained by each of them is within the threshold θ. In addition, the inverted files of the tree nodes that contain removed POIs are also updated.

Algorithm 7. Update RAC $^+$-Tree (Remove)
Input: an RAC$^+$-tree
1: $\mathcal{L}$.Enqueue(*root node*);
2: **while** $\mathcal{L}$ is not empty **do**
3: $p \leftarrow \mathcal{L}$. Dequeul;
4: **if** p contains outdated POIs **then**
5: **if** p is a leaf node **then**
6: Update the inverted file of p;
7: **else**
8: **if** p should be merged **then**
9: Merge the child nodes of p;
10: **else**
11: $\mathcal{L}$.Enqueue(child nodes of p);
12: **end if**
13: Update the ancestors of p;
14: **end if**
15: **end if**
16: **end while**

8 Experimental Study

In this section, we conduct extensive experiments on real trajectory datasets to demonstrate the performance of proposed indexes and algorithms. The dataset contains 41 376 trajectories in Beijing with 2 562 547 locations. We crawl the data from the Internet and the rating scores of activities are from historical users. During the update of the hybrid index, we use five different trajectory datasets, the cardinalities of which are 10k (673 517 locations), 15k (1 036 766 locations), 20k (1 353 854 locations), 25k (1 645 754 locations) and 30k (2 021 163 locations). All algorithms are implemented on a Core i5-3470 3.20 GHz machine with 8 GB memory and a Windows platform. The settings of experiment are presented in Table 7.

Table 7. Settings of Experiment

Parameter	Range	Default Value
Results k	5, 10, 15, 20, 25	10
$\widehat{d}$	5 km∼25 km	15 km
$\lvert q.\varphi\rvert$	3, 4, 5, 6, 7	5
θ	400∼1 200	800

We study the query time and the number of visited trajectories of different algorithms: 1) using hybrid index AC-tree to organize trajectory data and unoptimized algorithm to compute the rating of trajectory matching, denoted as B-TOSAT and B-OTOSAT for TOSAT and OTOSAT respectively; 2) RAC$^+$-tree and unoptimized algorithm based method, denoted as R-TOSAT and R-OTOSAT respectively; 3) we use C-TOSAT and C-OTOSAT to denote organizing trajectory data with RAC$^+$-tree and using the collaborative algorithm to compute the rating respectively.

Effectiveness of k. First of all, we investigate the effect of k by comparing the time cost and the number of visited trajectories of different algorithms. As shown in Fig.7, all methods except B-TOSAT and B-OTOSAT need more query time while varying k from 5 to 25. This is because the current k-th maximum result tends to be smaller when k increases, which means that more candidate trajectories should be taken into account. Hence, the query time of them monotonously increases with the increase of k. Compared with B-TOSAT and B-OTOSAT, R-TOSAT and R-OTOSAT need to consider less trajectories, since a large number of trajectories are pruned by traversing the RAC$^+$-tree. Even though R-TOSAT and C-TOSAT have the same number of visited trajectories for different k, C-TOSAT performs better than R-TOSAT since less points are considered in C-TOSAT with the collaborative algorithm. Similarly, C-OTOSAT performs better than R-OTOSAT.

Effectiveness of $|\mathcal{D}|$. Then we study the scalability of the three approaches. In Fig.8, the cardinality of $\mathcal{D}$ varies from 20k to 60k. With no surprise, it needs more time in query with the increasing of the number of historical trajectories. Besides, from Figs.8(b) and 8(d), we know that using RAC$^+$-tree to organize trajectory data outperforms using AC-tree, as the number of visited trajectories of the approaches with RAC$^+$-tree does not change obviously with the increasing of $|\mathcal{D}|$. The optimization of computing the ranking of trajectory matching is not sensitive in disorder case. However, this optimization is sensitive in order-sensitive case.

Effectiveness of $\widehat{d}$. Another important concern of query is the threshold of distance. Seen from Figs.9(b) and 9(d), the number of visited trajectories becomes larger with the increase of $\widehat{d}$. This is due to that a trajectory is more likely to be a candidate with a greater $\widehat{d}$. Consequently, all approaches need more time to answer the query with the increase of $\widehat{d}$.

Effectiveness of θ. Next we study the query performance while varying θ from 400 to 1 200 in Fig.10. Although a smaller θ means a finer-grained index tree and we should spend more time to traverse the hybrid index, we can prune more trajectories with a smaller θ. As a result, the query time and the number of visited trajectories become larger with the increase of θ, even though the change is not obvious in Fig.10.

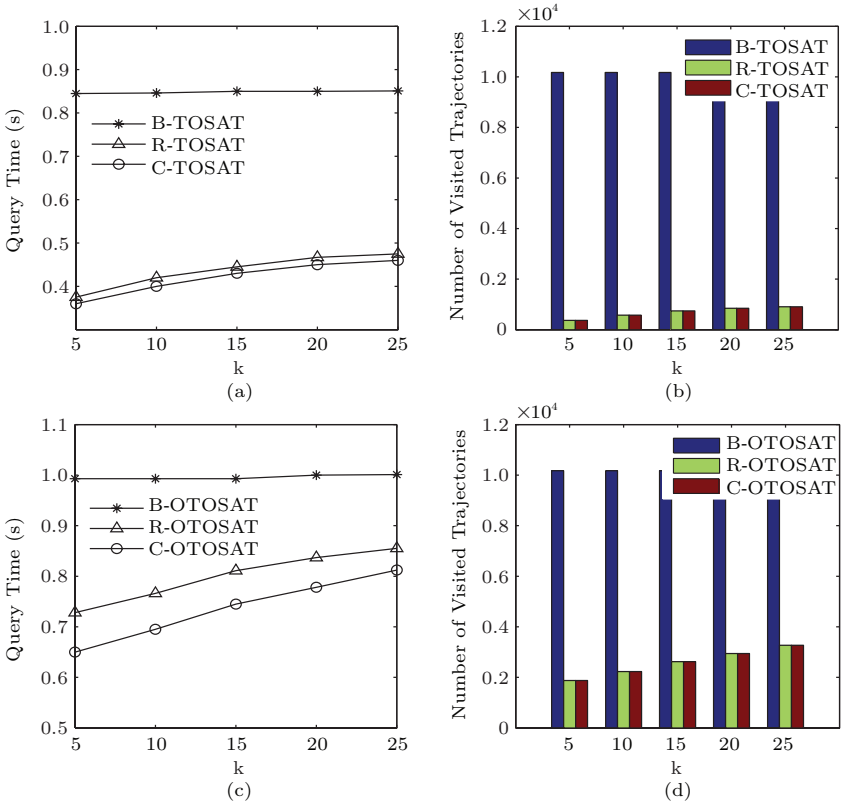

Fig.7. Effectiveness of k. (a) Query time of different TOSATs. (b) Number of visited trajectories of different TOSATs. (c) Query time of different OTOSATs. (d) Number of visited trajectories of different OTOSATs.

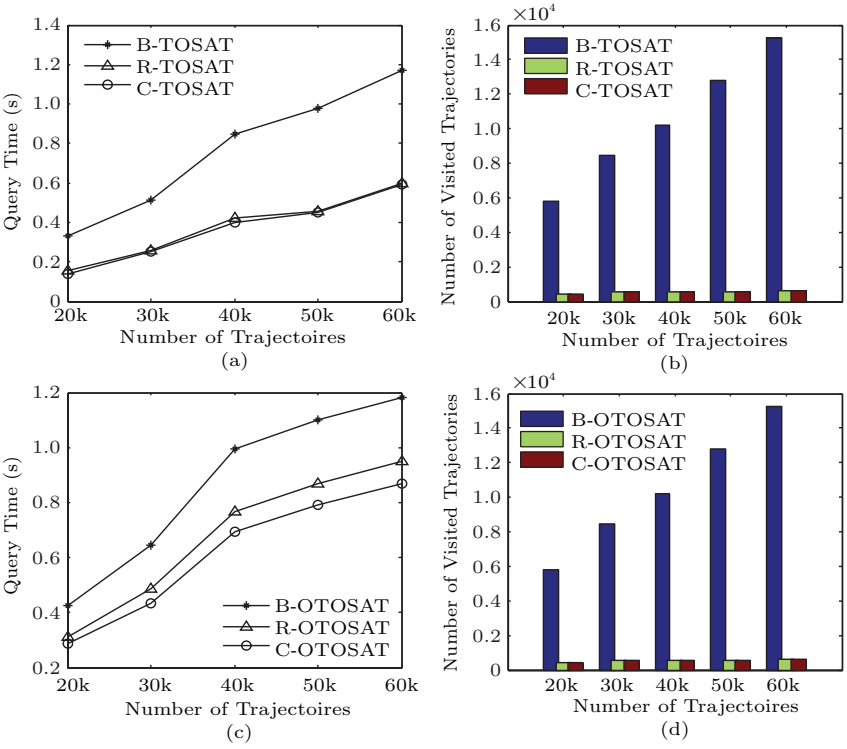

Fig.8. Effectiveness of $|\mathcal{D}|$. (a) Query time of different TOSATs. (b) Number of visited trajectories of different TOSATs. (c) Query time of different OTOSATs. (d) Number of visited trajectories of different OTOSATs.

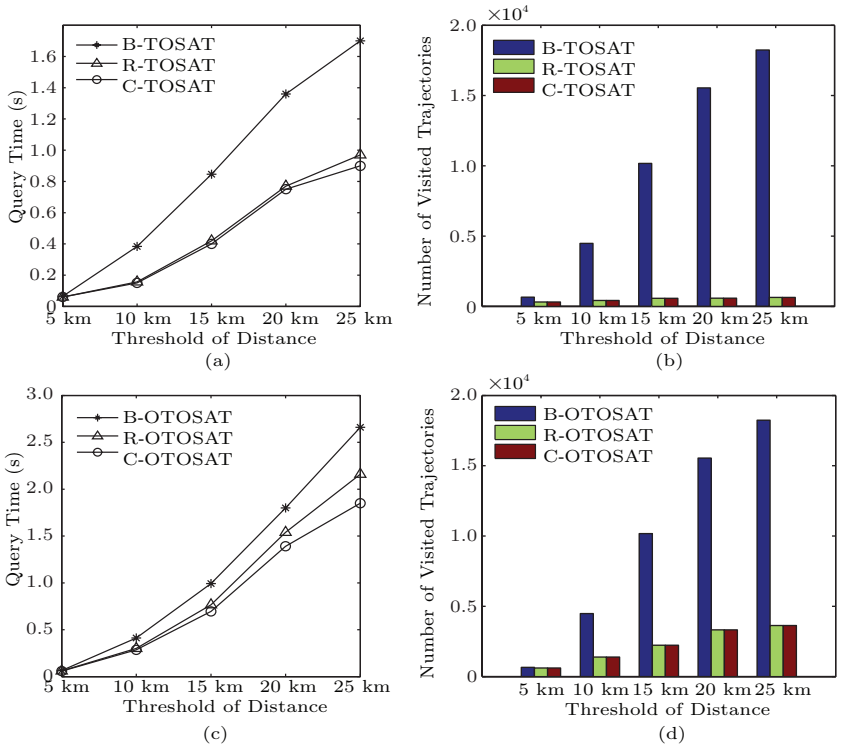

Fig.9. Effectiveness of $\widehat{d}$. (a) Query time of different TOSATs. (b) Number of visited trajectories of different TOSATs. (c) Query time of different OTOSATs. (d) Number of visited trajectories of different OTOSATs.

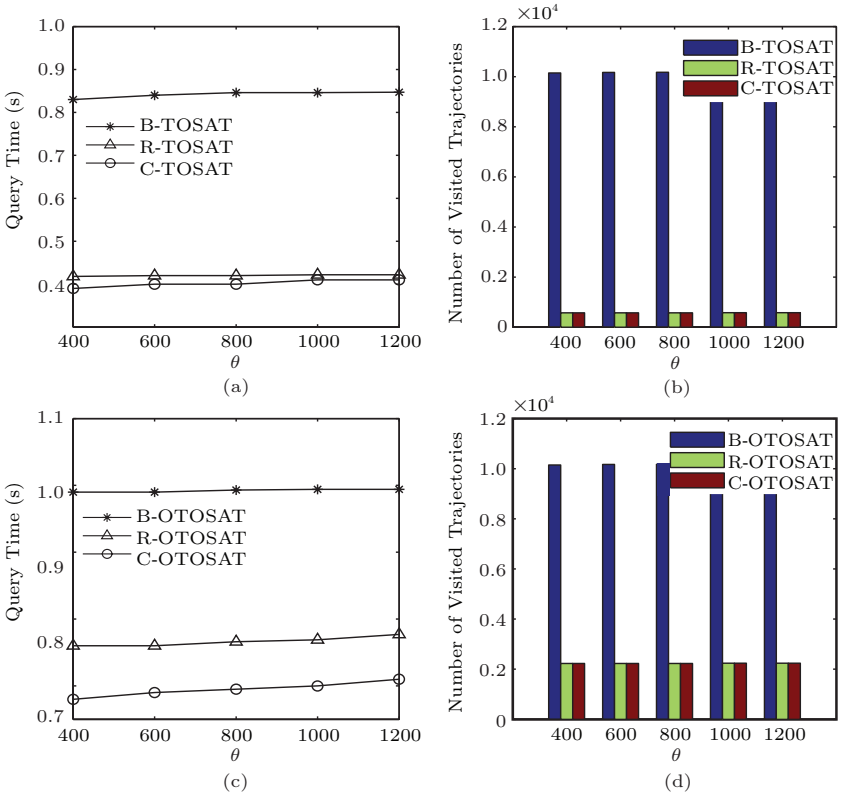

Fig.10. Effectiveness of θ. (a) Query time of different TOSATs. (b) Number of visited trajectories of different TOSATs. (c) Query time of different OTOSATs. (d) Number of visited trajectories of different OTOSATs.

Effectiveness of $|q.\varphi|$. Finally, we investigate the effect of the number of query activities. As depicted in Figs.11(b) and 11(d), the number of visited trajectories of B-TOSAT and B-OTOSAT becomes smaller with the increase of the query size, since a trajectory is less likely to cover all query activities with a greater $q.\varphi$. However, they need more time to compute the maximum rating for each candidate trajectory. Consequently, the time costs of them become larger with the increase of $|q.\varphi|$. For other methods, they should compute the rating for more trajectories, which leads to the increase of query time.

In addition, we present the time cost in updating the proposed hybrid indexes in Fig.12. Compare with

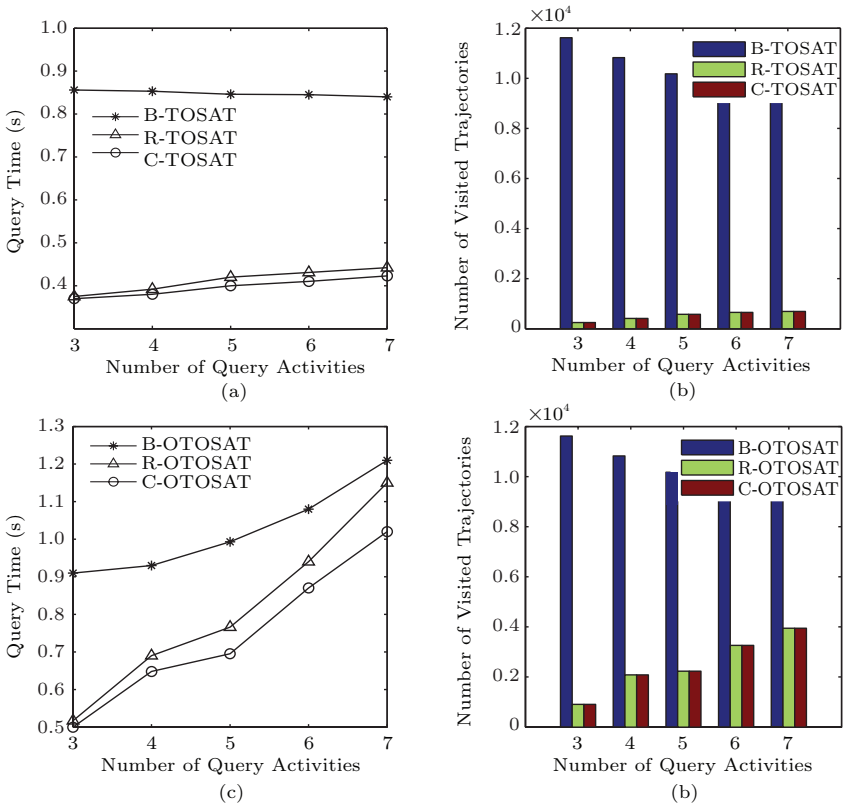

Fig.11. Effectiveness of $|q.\varphi|$. (a) Query time of different TOSATs. (b) Number of visited trajectories of different TOSATs. (c) Query time of different OTOSATs. (d) Number of visited trajectories of different OTOSATs.

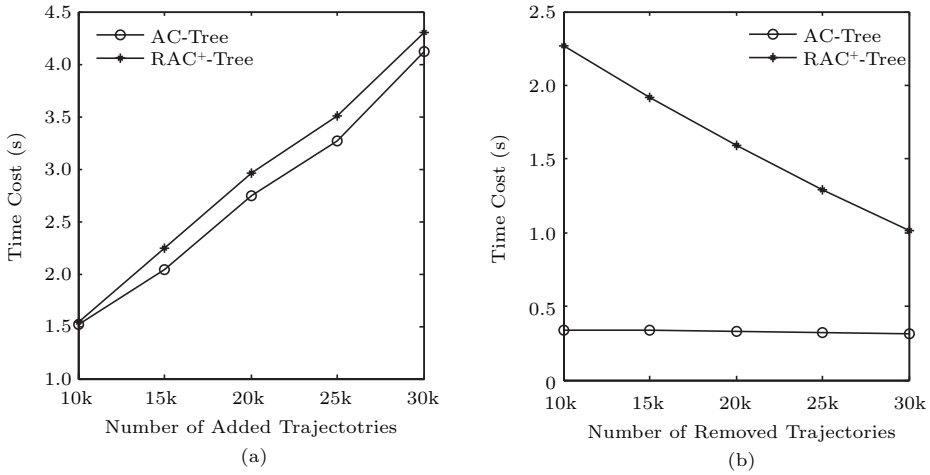

Fig.12. Performance of updating index.

the time cost in updating AC-tree while adding new trajectories, it needs more time to update RAC$^+$-tree, since the extra rating information and the spatial distance should be taken into account while updating the RAC$^+$-tree. Besides, as seen from Fig.12(b), it needs less time to update the index structure with the increase of the number of removed trajectories, as a tree node is more likely to be merged, and it needs to deal with less tree nodes, with a greater number of removed trajectories.

9 Conclusions

This paper investigated a novel problem of searching trajectories with activities, spatial distance and rating information. We formulated two types of queries, TOSAT and OTOSAT, depending on whether the order of query activities is taken into account. In order to tackle the problem efficiently, we proposed an AC-tree to organize trajectory data and developed novel algorithms to compute the rating of trajectory matching. We also optimized the AC-tree by developing an enhanced index RAC$^+$-tree to prune the search space with rating information and spatial distance. Besides, a collaborative algorithm was developed with the goal of improving the query performance. Experimental results based on real trajectory datasets show that the optimization of index structure and algorithms are capable of achieving high efficiency and scalability.

In real applications, time cost of a trip is an important concern for many users. Thus, we can take this factor into account in the future work. Obviously, the algorithms can recommend more satisfactory results to users by considering the new factor.

References

[1] Li Z, Ding B, Han J, Kays R. Swarm: Mining relaxed temporal moving object clusters. *Proceedings of the VLDB Endowment*, 2010, 3(1/2): 723-734.

[2] Zheng K, Zheng Y, Yuan N, Shang S, Zhou X. Online discovery of gathering patterns over trajectories. *IEEE Trans. Knowledge and Data Engineering*, 2014, 26(8): 1974-1988.

[3] Huang M, Hu P, Xia L. A grid based trajectory indexing method for moving objects on fixed network. In *Proc. the 18th Int. Conf. Geoinformatics*, June 2010.

[4] Popa L S, Zeitouni K, Oria V *et al.* Indexing in-network trajectory flows. *The VLDB Journal*, 2011, 20(5): 643-669.

[5] Chu S, Yeh C, Huang C. A cloud-based trajectory index scheme. In *Proc. the 12th ICEBE*, October 2009, pp.602-607.

[6] Vlachos M, Kollios G, Gunopulos D. Discovering similar multidimensional trajectories. In *Proc. the 18th ICDE*, Feb. 26-Mar. 1, 2002, pp.673-684.

[7] Chen L, Özsu M T, Oria V. Robust and fast similarity search for moving object trajectories. In *Proc. the 24th SIGMOD*, June 2005, pp.491-502.

[8] Chen Z, Shen H, Zhou X, Zheng Y, Xie X. Searching trajectories by locations: An efficiency study. In *Proc. the 29th SIGMOD*, June 2010, pp.255-266.

[9] Chen Z, Shen H, Zhou X. Discovering popular routes from trajectories. In *Proc. the 27th ICDE*, April 2011, pp.900-911.

[10] Zheng K, Shang S, Yuan N J, Yang Y. Towards efficient search for activity trajectories. In *Proc. the 29th ICDE*, April 2013, pp.230-241.

[11] Zhang C, Han J, Shou L, Lu J, La Porta T. Splitter: Mining fine-grained sequential patterns in semantic trajectories. *Proceedings of the VLDB Endowment*, 2014, 7(9): 769-780.

[12] Ying J, Lee W, Weng T, Tseng V. Semantic trajectory mining for location prediction. In *Proc. the 19th SIGSPATIAL*, November 2011, pp.34-43.

[13] Zhou Y, Xie X, Wang C, Gong Y, Ma W. Hybrid index structures for location-based web search. In *Proc. the 14th International Conference on Information and Knowledge Management*, October 31-November 1, 2005, pp.155-162.

[14] Chen Y, Suel T, Markowet A. Efficient query processing in geographic web search engines. In *Proc. the 25th SIGMOD*, June 2006, pp.277-288.

[15] Hariharan R, Hore B, Li C, Mehrotra S. Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In *Proc. the 19th SSBDM*, July 2007, Article No. 16.

[16] Cao X, Cong G, Jensen C, Ooi B. Collective spatial keyword querying. In *Proc. the 30th SIGMOD*, June 2011, pp.373-384.

[17] Shang S, Ding R, Yuan B, Xie K, Zheng K, Kalnis P. User oriented trajectory search for trip recommendation. In *Proc. the 15th EDBT*, March 2012, pp.156-167.

[18] Long C, Wong R, Wang K, Fu A. Collective spatial keyword queries: A distance owner-driven approach. In *Proc. the 32nd SIGMOD*, June 2013, pp.689-700.

[19] Wang C, Xie X, Wang L, Lu Y, Ma W. Web resource geographic location classification and detection. In *Proc. the 14th International Conference on World Wide Web*, May 2005, pp.1138-1139.

[20] De Felipe I, Hristidis V, Rishe N. Keyword search on spatial databases. In *Proc. the 24th ICDE*, April 2008, pp.656-665.

[21] Zhang D, Chee Y, Mondal A, Tung A, Kitsuregawa M. Keyword search in spatial databases: Towards searching by document. In *Proc. the 25th ICDE*, March 29-April 2, 2009, pp.688-699.

[22] Cong G, Jensen C S, Wu D. Efficient retrieval of the top-k most relevant spatial web objects. *Proceedings of the VLDB Endowment*, 2009, 2(1): 337-348.

Wei Chen is currently a Ph.D. candidate in the School of Computer Science and Technology, Soochow University, Suzhou. His research interests include data mining and spatial-temporal database.

Lei Zhao received his Ph.D. degree in computer science in 2006 from Soochow University, Suzhou. He has been a faculty member of the School of Computer Science and Technology of Soochow University since 1998. He is now a professor of the Department of Software Engineering. His research interests include distributed data processing, data mining, parallel and distributed computing, etc.

Jia-Jie Xu is an associate professor of Soochow University, Suzhou, and a member of the Research Center on Advanced Data Analytics, Soochow University. He got his M.S. and Ph.D. degrees from Swinburne University of Technology and University of Queensland in 2006 and 2011 respectively, and then worked in the Institute of Software, Chinese Academy of Sciences, Beijing, as an assistant professor before joining Soochow University. His research interests mainly include spatio-temporal database systems, big data analytics, mobile computing and workflow systems.

Guan-Feng Liu is an associate professor of the Research Center on Advanced Data Analytics (ADA) in Soochow University, Suzhou. He received his Ph.D. degree in computer science from Macquarie University, Australia, in 2013. His research interests include social networks and graph mining.

Kai Zheng is currently a professor in the School of Computer Science and Technology at Soochow University. He received his Ph.D. degree in computer science from The University of Queensland in 2012. His research focus is to find effective and efficient solutions for managing, integrating and analyzing big data for business, scientific and personal applications. He has been working in the area of social-media analysis, spatial-temporal databases, uncertain databases, data mining and bioinformatics. He has published over 50 papers in the most prestigious journals and conferences such as SIGMOD, ICDE, EDBT, The VLDB Journal, ACM Transactions and IEEE Transactions. He was the program committee chair of the International Workshop on Human Mobility Computing (HuMoComp), and the International Workshop on Big Data Management and Service (BDMS) in 2013. He is a PC member of SIGMOD 2015, 2016, CIKM 2014, 2015, DASFAA 2013, 2015. He is on the reviewer board of several prestigious journals such as IEEE TKDE, ACM TODS, VLDB Journal, KAIS and Geoinformatica. He was the recipient of Australian Discovery Early Career Research Award in 2013.

Xiaofang Zhou is a professor of computer science at the University of Queensland. He received his B.S. and M.S. degrees in computer science from Nanjing University, Nanjing, and Ph.D. degree in computer science from the University of Queensland, Australia. His research focus is to find effective and efficient solutions for managing, integrating and analyzing very large amount of complex data for business, scientific and personal applications. He has been working in the area of spatial and multimedia databases, data quality management, in-memory databases, high performance query processing, Web information systems, data mining and bioinformatics.