Mining the Most Influential k-Location Set From Massive Trajectories

Yuhong Li, Jie Bao, *Member, IEEE*, Yanhua Li, *Member, IEEE*, Yingcai Wu, *Member, IEEE*, Zhiguo Gong, *Senior Member, IEEE*, and Yu Zheng, *Senior Member, IEEE*

Abstract—Mining the most influential location set finds k locations, traversed by the maximum number of unique trajectories, in a given spatial region. These influential locations are valuable for resource allocation applications, such as selecting charging stations for electric automobiles and suggesting locations for placing billboards. This problem is NP-hard and usually calls for an interactive mining processes involving a user's input, e.g., changing the spatial region and k, or removing some locations (from the results in the previous round) that are not eligible for an application according to the domain knowledge. Efficiency is the major concern in conducting this human-in-the-loop mining. To this end, we propose a complete mining framework, which includes an optimal method for the light setting (i.e., small region and k) and an approximate method for the heavy setting (i.e., large region and k). The optimal method leverages *vertex grouping* and *best-first pruning* techniques to expedite the mining process. The approximate method can provide the performance guarantee by utilizing the greedy heuristic, and it is comprised of *efficient updating strategy*, *index partition* and *workload-based optimization* techniques. We evaluate the efficiency and effectiveness of our methods based on two taxi datasets from China, and one check-in dataset from New York.

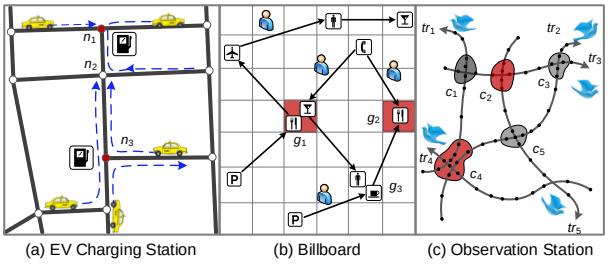
Index Terms—Location Selection, Most Influential k-Location Set, Maximum Coverage Problem, Trajectory Data Mining.

1 Introduction

Advances in location acquisition technology have resulted in massive trajectories, representing the mobility of a diversity of moving objects, such as human, vehicles, and animals. As a consequence, many techniques have been proposed for processing and mining trajectory data with a broad range of applications over the last decade, ranging from trajectory pattern mining [1], [2], trajectory classification and clustering [3], [4], trajectory outlier detection [5], to location-based services [6], [7], [8] etc. Different from previous works, we focus on identifying a set of appropriate locations which are traversed by the maximum number of unique trajectories in a given spatial region.

Applications. Mining the most influential k-location set is vital to many resource allocation applications.

The first application is selecting charging stations for electric vehicles according to their GPS trajectories. As shown in Fig. 1(a), the candidate locations are the road intersections. Among them, intersections n_1 and n_3 form the most influential 2-location set as they cover 5 unique trajectories. Intersections n_2 and n_3 are not the most influential set, as they only cover 4 unique trajectories.



(a) EV Charging Station (b) Billboard (c) Observation Station

Fig. 1: Application Scenarios.

Though they individually have the most number of trajectories (i.e., 3 for each) traversing them.

The second application is selecting locations for placing billboards based on users' check-in histories or trajecotires [9]. as shown in Fig. 1(b), a location can be defined as a uniform grid covering a few points of interests (POIs). Grids g_1 and g_2 form a most influential 2-location set, traversed by 4 unique trajectories, i.e., visited by 4 users. Grids g_1 and g_3 cannot construct the most influential 2-location set, as they only cover 3 users.

The third application is to place observation stations for migratory birds, where a location can be a cluster of birds' stay points detected from their moving trajectories. As shown in Fig. 1(c), clusters c_2 and c_4 form the most influential 2-location set, they totally cover all birds' trajectories.

Challenges. There are three major challenges in mining the most influential k-location set from massive trajectories: i) this problem can be mapped to the MAX-k-COVER problem, which is NP-hard and computational intensive; ii) different users may be interested in mining k locations in different spatial regions. For instance, as shown in Fig. 2(a), two local business owners may want to place different number of advertisements in different areas. However,

- *Y.H. Li and Z.G. Gong are with the Department of Computer and Information Science, University of Macau, Macau, China. This work was done when Yuhong Li was a visiting student supervised by Jie Bao and Yu Zheng at Microsoft Research Asia. E-mail: {yb27407,fstzgg}@umac.mo*
- *J. Bao and Y. Zheng are with Microsoft Research, Beijing, China. Y. Zheng is also affiliated with School of Computer Science and Technology, Xidian University, and Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. E-mail: {jiebao, yuzheng}@microsoft.com*
- *Y.H. Li is with the Computer Science Department, Worcester Polytechnic Institute, Worcester, MA, USA. E-mail: yli15@wpi.edu*
- *Y.C. Wu is with the State Key Lab of CAD & CG, Zhejiang University, Zhejiang, China. E-mail: ycwu@cad.zju.edu.cn*

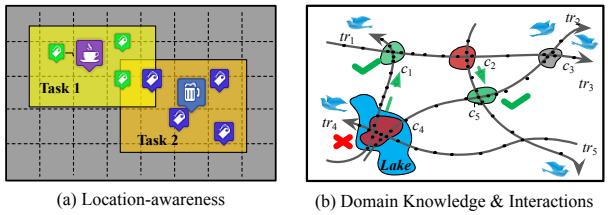
it is not possible to pre-compute one location set serving all requests with different mining parameters; and iii) users, i.e., domain experts, may need to interact with our system several times based on their domain knowledge. For example, as depicted in Fig. 2(b), c_4 is located in a lake where we cannot find land to place an observation station. Thus, c_4 should be removed from the returned set and $\{c_1, c_5\}$ becomes the most influential 2-location set.

Although the MAX-k-COVER problem has been studied [10], [11], [12], [13], [14], existing methods are off-line approaches that find a one-time result. Different from these works, our problem setting allows a user 1) to specify a spatial region and k, and 2) to refine returned results interactively and iteratively. In order to attract users to pursue interactions in the mining process, it is crucial to improve the system's efficiency.

Contributions. To address the aforementioned challenges, a complete mining framework is proposed to find the most influential k-location set efficiently. Our system consists of two main modules: i) *pre-processing module*, which creates the spatial networks from different types of trajectory data and builds a set of indices to speed up the mining process; and ii) *location set mining module*, which finds a k-location set by taking spatial region, k value, and choices made during the user's interaction as the input. Our location mining module not only provides the optimal solution for small spatial region and k, but also provide an approximate solution for large spatial region and k with performance guarantee. The contributions of this paper can be summarized as follows:

- We introduce a novel trajectory data mining task, i.e., mining the most influential k-locations set from massive trajectories, with many potential applications.
- We propose an efficient algorithm to provide the optimal result, when k and spatial area are small. The algorithm groups the nearby locations together and performs the best-first pruning to avoid checking some unpromising candidates, i.e., k-location set.
- We propose an efficient algorithm to find the location set with the greedy heuristic, in case of large k and spatial area. The efficiency is enabled by precomputing several data indices which can speed up the updating phase in the greedy heuristic.
- Experimental evaluations on a taxi dataset from Tianjin show that our proposed system is an order of magnitude faster than the baseline solution. We also provide two case studies to demonstrate the applicability of our proposed system. Moreover, we have already deployed a system to select the appropriate k-location set for placing billboards [9], [15].

Outline. The rest of the paper is organized as follows. Sect. 2 provides the preliminary and overview of our proposed system. Sect. 3 presents the pre-processing module of our system. The optimal solution for the light setting is presented in Sect. 4, and we describe the approximate solution for the heavy setting in Sect. 5. Experimental evaluation are conducted in Sect. 6, followed by the related works in Sect. 7. Finally, we conclude the work.



(a) Location-awareness (b) Domain Knowledge & Interactions

Fig. 2: Summary of Challenges.

2 Overview

In this section, we formally define the problem and present the framework of our proposed system.

2.1 Preliminary

We first introduce some basic definitions that are widely used in this paper.

Definition 1 (Trajectory)**.** *A trajectory tr is a sequence of spatial points that a moving object follows through space as a function of time. Each point consists of an object ID, latitude, longitude, and a time stamp.*

Definition 2 (Location)**.** *A location is a spatial point or region, which can be defined in three forms: 1) an intersection in a road network, e.g., n_1 as shown in Fig. 1(a); 2) a grid cell, e.g., g_1 as depicted in Fig. 1(b); or 3) a stay point or a cluster of points from trajectories, e.g. c_2 as illustrated in Fig. 1(c).*

Definition 3 (Spatial network)**.** *A spatial network can be denoted as a directed graph $G = (V, E)$, where the vertex set V represents the locations* [1] *and the directed edge set E represents the set of edges where each has two terminal vertexes (locations).*

Definition 4 (Trajectory coverage)**.** *A location v_i covers a trajectory tr_j, if and only if the trajectory tr_j passes the location v_i. Given a location on a spatial network (e.g., an intersection v_i on a road network), its coverage set $Tr(v_i)$ represents the set of trajectories passing through the location v_i. Similarity, we use $Tr(V)$ to denote the set of trajectories passing through the location set V, and it can be formally calculated as $Tr(V) = \cup_{v_i \in V} Tr(v_i)$.*

2.2 Problem Definition.

The problem proposed in this work, i.e., mining the most influential k-location set in a given spatial region, is formally defined as:

Definition 5 (The Most Influential k-Location Set)**.** *Given a user-specified spatial region R, a k value and a trajectory set Tr, we denote the spatial network in R as $G_s = (V_s, E_s)$. The most influential k-location set in R finds k* [2] *locations in V_s, such that the total number of unique trajectories being covered by these k locations is maximized.*

To be precise, we use the following integer linear programming (ILP) formulation to captures the problem exactly. We use $v_i.s$ and $tr_j.s$ to indicate the solution of the problem. For each location $v_i \in V_s$, $v_i.s = 1$ if v_i is selected in the result set, and $v_i.s = 0$ otherwise; for each trajectory $tr_j \in Tr$, $tr_j.s = 1$ if tr_j is covered by these selected locations, and $tr_j.s = 0$, otherwise.

1. Vertex and location are interchangeable in this work.
2. Here $k \leq |V_s|$.

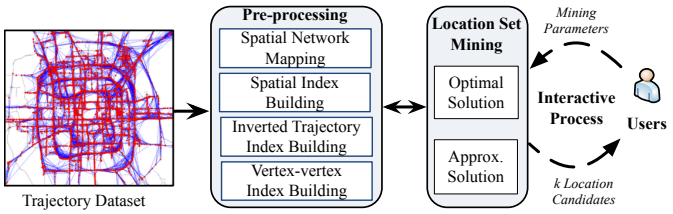


Fig. 3: System Overview.

$$\max: \sum_{tr_j \in Tr} tr_j.s, s.t.: \sum_{v_i \in V_s} v_i.s \leq k, \sum_{tr_j \in Tr(v_i)} v_i.s \geq tr_j.s \quad (1)$$

The objective of Eq. 1 is to maximize the total number of unique trajectories being covered by the selected k locations. The first constraint guarantees that the total number of selected locations is no more than k; the second constraint ensures that if a trajectory tr_j is covered, then at least one location v_i that $tr_j \in Tr(v_i)$ should be selected in the result set. This problem is equal to the MAX-k-COVER problem and is NP-hard as been proven in [10], [16], [17].

Moreover, in some application scenarios, the system need to support interactions with the help of domain experts to find the qualified k-location set. Specifically, at the initial step, the system returns k locations that maximize the number of covered trajectories based on the current parameters, i.e., spatial region and k. Then, the expert involves and marks $0 \leq \ell \leq k$ disqualified locations from these k (selected) locations, based on his domain knowledge. In the following steps, the system needs to remove these ℓ marked locations and re-selects k locations, covering the maximum number of unique trajectories. This process iterates, until the expert accepts all the returned k locations [3].

Therefore, each interaction needs to be done in a timely fashion, so that the expert can proceed to further mark disqualified locations. This motivates our system to achieve two objectives: 1) maximizing the trajectory coverage; and 2) minimizing the response time.

2.3 System Overview

Fig. 3 gives the overview of our proposed system. It contains two main modules:

Pre-processing Module. As shown in the left portion of Fig. 3, *pre-processing module* takes the trajectory dataset as the input, and it performs the following four procedures:

Step 1- *Spatial Network Mapping*, which maps the raw trajectory onto the corresponding spatial network (e.g., the road network as shown in Fig. 1(a)). The output of this step is the trajectory-vertex index.

Step 2- *Spatial Indexing*, which indexes the vertices (locations) based on their spatial coordinates, i.e., latitude and longitude. The goal of this step is to boost the spatial range search.

Step 3- *Inverted Trajectory Indexing*, which aggregates the trajectory IDs over each vertex in the spatial network and generates the *vertex-trajectory index*.

3. As a remark, pre-estimating the quality of all locations for different applications maybe infeasible.

Step 4- *Vertex-vertex Indexing*, which calculates the number of shared trajectories between two vertices.

Location Set Mining Module. As presented at the right part of Fig. 3, *Location Set Mining Module* takes user's mining parameters, i.e., a spatial region R, a value k and a set of marked vertices as the input, and returns k locations as the result. The process goes multiple iterations until the user satisfies the final returned result. In this paper, we propose an efficient *optimal solution* to process each iteration with relatively smaller R and k (detailed in Sect. 4); and an efficient *approximate solution*, which utilizes the greedy heuristic to choose the candidate locations for larger R and k (detailed in Sect. 5).

3 Pre-Processing

In this section, we present the four procedures (cf. Sect. 2.3) of the pre-processing module in detail.

Spatial Network Mapping This step contains two tasks: 1) spatial network construction, the system first identifies the locations based on different scenarios, e.g., the intersections, spatial cells, or the stay points, then constructs the spatial network; 2) trajectory map-mapping, the system needs to map the raw trajectories onto the corresponding spatial network, e.g., using a map matching algorithm as proposed in [18]. The output of the procedure is a *trajectory-vertex index* which is denoted as $\mathcal{I}_{tv}$. In this index, each entry records a set of locations that a trajectory traversed, i.e., the entry of tr_i is $\{tr_i | v_x, v_y, ..., v_z\}$.

Spatial Index Building The spatial index is used to speed up the spatial selection process. In this step, we take the constructed spatial network $G = (V, E)$ as the input and use R$^+$-tree [19] to index the spatial vertices (locations). The output of this procedure is a hierarchical tree structure $\mathcal{I}_{spatial}$.

Inverted Trajectory Index Building In this step, the system builds the *vertex-trajectory index*, which is denoted as $\mathcal{I}_{vt}$. In this index, it stores the covered trajectory IDs for each location, i.e., the entry of v_i is $\{v_i | tr_x, tr_y, ..., tr_z\}$. The construction of the vertex-trajectory index is quite simple, i.e., it scans each entry (trajectory) in *trajectory-vertex index* and adds the current trajectory ID to each scanning vertex.

Vertex-vertex Index Building The *vertex-vertex index* records the number of shared trajectories between two vertices, i.e., the entry of v_i is $\{v_i | (v_x, c_{ix}), (v_y, c_{iy}), ..., (v_z, c_{iz})\}$, where c_{ix} indicates the number of shared trajectories between vertices v_i and v_x. To construct the *vertex-vertex index*, we take the *trajectory-vertex index* as the input. For each trajectory, it adds one to every pair of vertices in the *vertex-vertex index*. The utilization and optimization of the vertex-vertex index are presented in Sect. 5.

4 Optimal Location Set Mining

For small spatial region R and k, it is possible to derive an optimal solution to find the most influential k-location set from massive trajectories. In this section, we first introduce a naive algorithm (cf. Section 4.1), that finds the optimal solution by enumerating and examining all possible k-location sets in a given spatial region. Then, we develop an efficient optimal solution by applying vertices grouping and best-first pruning techniques (cf. Section 4.2).

4.1 Naive Optimal Algorithm

The naive optimal algorithm is straightforward with three steps: 1) extracting all the vertices, denoted as V_s, within the spatial region R by using the spatial index $\mathcal{I}_{spatial}$; 2) generating all the possible k-location sets from the extracted vertex set V_s; and 3) calculating the trajectory coverage for all combinations (i.e., k-location sets) and returning the combination with the maximum coverage value as the result.

Obviously, the naive optimal algorithm is computing infeasible for large $|V_s|$ (i.e., spatial region R) and k. Specifically, the algorithm needs to check $C^k_{|V_s|} = \frac{|V_s|!}{k!(|V_s|-k)!}$ k-location sets, i.e., the number of possible combinations increases exponentially with both $|V_s|$ and k. Moreover, calculating the trajectory coverage for a k-location set is not an efficient process, especially when the size of trajectory dataset is huge. A classical implementation for counting the coverage of a k-location set is sorting the trajectories in each vertex (location) according to their trajectory IDs[4], then the covered (unique) trajectories of a k-location set can be calculated by a linear scan of their sorted trajectory lists.

4.2 Group Pruning Optimal Algorithm

The naive optimal algorithm needs to exhaustively scan all the possible k-location sets to find the optimal result. To avoid this drawback and improve the efficiency, the *group pruning optimal (GPO)* algorithm is proposed.

Main idea. The intuition of GPO algorithm is to prune the unpromising k-location sets by batch with two techniques:

• *Vertices grouping.* The vertices in the spatial region R can be clustered into g groups. Instead of checking all the k-location sets directly, we first estimate all the k-group sets. The number of k-group sets should be much more less than the k-location sets. If the coverage upper bound of a k-group set is already smaller than the current best result, all the k-location sets belong to this k-group set can be pruned safely.

• *Best-first pruning.* To improve the pruning ability, we can apply the best-first strategy to prioritize the order of execution. By processing the k-group set with the highest coverage upper bound firstly, a better coverage bound can be expected to prune the remaining unpromising k-group sets more effectively.

Algorithm 1 Group Pruning Optimal (GPO) Algorithm

Input: Vertex-trajectory index $\mathcal{I}_{vt}$, spatial index $\mathcal{I}_{spatial}$, spatial range R, and k value.
Output: The optimal k-location set V_{opt}.
1: $V_s := RangeSearch(\mathcal{I}_{spatial}, R)$
2: Divide V_s into a set of groups G
3: Generate all the k-group sets GS
4: Estimate the coverage upper bound for each k-group set
5: **for** $gs \in GS$ in descending order of coverage upper bound **do**
6: **if** $UB(gs) > |Tr(V_{opt})|$ **then**
7: **for** $V_i \in gs$ **do**
8: **if** $|Tr(V_i)| > |Tr(V_{opt})|$ **then**
9: $V_{opt} := V_i$
10: **else**
11: break;
12: Return V_{opt}

4. The sorting of trajectory IDs for all vertices can be done in the pre-processing module.

Algorithm. Algorithm 1 gives the pseudocode of the group pruning optimal (GPO) algorithm, with the following steps:

Step 1. Spatial Selection. The algorithm selects the vertices in the mining region R, using the spatial index $\mathcal{I}_{spatial}$ (i.e., Line 1).

Step 2. Online Vertices Grouping. In this step, we divide all the candidate vertices V_s into a set of groups $g \in G$ (Line 2). To achieve a tight upper bound, our grouping is based on the observation that vertices, which are close to each other, usually share more common trajectories. In our implementation, we apply a R^+-tree to group the vertices, where the number of groups is controlled by maximum size of the leaf node. As a remark, other grouping techniques (e.g., clustering [20], KD-tree [21], and Hilbert curve[22]) are also applicable. After this phase, the trajectory coverage $Tr(g)$ for each group $g \in G$ is calculated.

Step 3. k-group Sets Generation. In this step, the algorithm: 1) generates all possible k-group sets that may produce the k-location set (Line 3); and 2) estimates the coverage upper bound of each k-group set, by counting the number of unique trajectory IDs (Line 4). Formally, the coverage upper bound of a k-group set gs is defined as:

$$UB(gs) = |\cup_{g \in gs} Tr(g)| \tag{2}$$

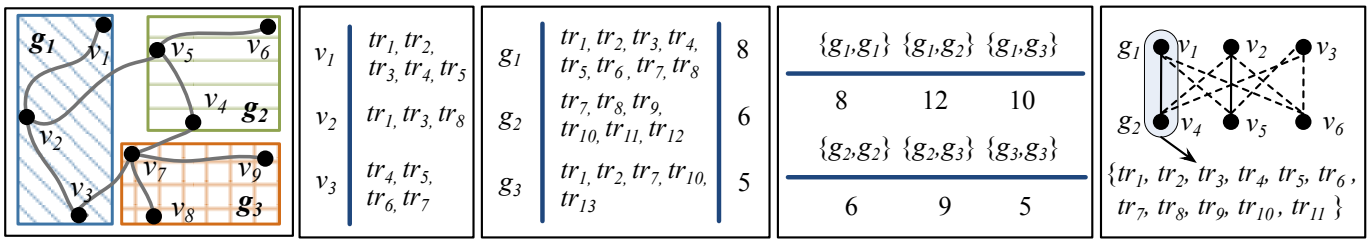
Step 4. Best-first Pruning. In this step, we first sort all the k-group sets based on their coverage upper bound. Then we check the k-group sets in descending order of their coverage upper bound (Line 5). For each qualified k-group set, i.e., the k-group set whose coverage upper bound is larger than the best-so-far (Line 6), we check all the k-location sets within this k-group set. The temporal result is updated if one of these k-location set is better than the current best-so-far V_{opt} (Line 8-9). This kind of process continues until the coverage upper bound of a k-group set is less than the current best-so-far (V_{opt}). Finally, the algorithm terminates and returns V_{opt} as the result (Line 12).

Example. Figure 4 gives a example of choosing 2-location set within a given region. As the first step, the algorithm identify the vertices within the given spatial region R, and 9 candidate vertices are selected from the spatial network (i.e., v_1 to v_9) as shown in Figure 4(a).

After that, the algorithm divides the vertices into groups. In our example, they are divided into 3 groups (i.e., g_1 to g_3), which are bounded by different rectangles as shown in Figure 4(a). Figure 4(b) shows the trajectory coverage for all vertices in the group g_1. Accordingly, we can estimate the number of covered trajectories for this group. As shown in Figure 4(c), the number of covered trajectories for g_1, g_2 and g_3 are 8, 6 and 5 respectively.

Then, the *k-group Sets Generation* step is executed, as demonstrated in Figure 4(d). In this step, we first generate all the possible 2-location sets. Note that it is possible that both two vertices comes from the same group, thus we have the 2-group set like $\{g_1, g_1\}$. We can estimate the coverage upper bound for all the 2-group sets based on the Equation 2.

Finally, the algorithm runs the *best-first pruning*, as shown in Figure 4(f). It first check the 2-group set with the highest coverage upper bound (i.e., $\{g_1, g_2\}$ in the example). For this 2-group set, we need to exhaustively check all 2-location sets in it. In this example, we find the 2-location set $\{v_1, v_4\}$ possess the current best with a total of 11 covered trajectories. As the maximum coverage upper bound of the remaining 2-group sets is 10, the algorithm then stops, and reports $\{v_1, v_4\}$ as the final result.



(a) Spatial Partitions in Region R (b) Vertex Coverage in Group g_1 (c) Group Coverage (d) Coverage Upper Bound of 2-group Sets (f) Best-first Pruning

Fig. 4: Illustration of Group Pruning Optimal (GPO) Algorithm.

Algorithm 2 Framework of Greedy Heuristic Algorithm

Input: Vertex-trajectory index $\mathcal{I}_{vt}$, spatial index $\mathcal{I}_{spatial}$, vertex coverage table vct, spatial range R, and k value.
Output: k-location set V_{gdy}
1: $V_s := RangeSearch(\mathcal{I}_{spatial}, R)$
2: $V_{gdy} := \emptyset$
3: **for** i = 1 to k **do**
4: $v_{cur} := \arg\max_{v_i \in V_s \setminus V_{gdy}} vct[v_i]$
5: $V_{gdy} \leftarrow V_{gdy} \bigcup v_{cur}$
6: Update the coverage values of vct
7: Return V_{gdy}

Comparing to the naive optimal algorithm, which needs to check $C_9^2 = \frac{9\times8}{2} = 36$ k-location sets, the group pruning optimal (GPO) algorithm only needs to check 6 k-group sets and 9 k-location sets. In this example, we can expect at least a 100% efficiency improvement.

5 Approximate Location Set Mining

In the case of large k value and spatial region, the optimal solution may take a long time even by utilizing the GPO algorithm. An efficient approximate solution becomes more promising, especially when we need multiple rounds of interactions from the field experts[5]. In the literature, the greedy heuristic has been proved [23] as the best polynomial time solution with the guarantee of $1 - \frac{1}{e}$ approximation ratio.

In this section, we first present the framework of greedy heuristic algorithm which contains a selection phase and an updating phase. The updating phase dominants the performance of the greedy heuristic algorithm (cf. Sect. 5.1). To reduce its processing time, we introduce the *basic updating algorithm* by using the *trajectory-vertex index*. However, even by the trajectory-vertex index, it still takes more than 10 seconds for a trajectory dataset with million trajectories (cf. Sect. 6). Finally, the *partitioned index batch updating (PIBU)* algorithm is proposed by utilizing the *vertex-vertex index* to select locations from massive trajectories efficiently.

5.1 Framework of the Greedy Heuristic

In the greedy heuristic algorithm, we maintain a *vertex coverage table*, i.e., vct. Each entry of vct is identified by the vertex id v_i, and is associated with a coverage value. The coverage value of vertex v_i is denoted as $vct[v_i]$, and it records the number of newly covered trajectories if we put v_i to the current result set.

5. Whether the optimal or approximate solution should be used depends on their response time and application scenarios.

Algorithm 3 Basic Updating (Basic) Algorithm

Input: vertex-trajectory index $\mathcal{I}_{vt}$, trajectory-vertex index $\mathcal{I}_{tv}$, candidate vertices V_s, selected vertex v_{cur}, vertex coverage table vct.
Output: Updated *vertex coverage table*
1: $Tr_{new} \leftarrow$ newly covered trajectories by v_{cur}
2: **for** each $tr \in Tr_{new}$ **do**
3: **for** each $v \in \mathcal{I}_{tv}[tr]$ **do**
4: **if** $v \in V_s \setminus V_{gdy}$ **then**
5: Update coverage value of v in vct.

The greedy heuristic algorithm is very simple, as shown in Algorithm 2. Similar to Algorithm 1, it first selects a set of candidate vertices V_s in the spatial region R (i.e., Line 1). After that, a k-iterative process is executed with the following two phases:

- *Selection Phase.* In this phase, the algorithm selects the vertex with maximum coverage value in vct (i.e., Line 4) and put it in the result set (i.e., Line 5).
- *Updating Phase.* In this phase, the algorithm updates the coverage values in vct for all the unselected vertices by removing the newly covered trajectories from their coverage (i.e., Line 6).

In Algorithm 2, the spatial range search and selection operations can be processed very efficiently. However, the updating step is hideous and time consuming, especially in case of massive trajectories, e.g., millions of entries, as the updating process needs to remove all the newly covered trajectories from the *vertex coverage table*. As a result, the key to expedite the mining processing lays on improving the efficiency of the updating phase.

5.2 Basic Updating Algorithm

After each selection operation, a set of trajectories are newly covered by the selected vertex. Thus, the coverage values of the remaining vertices in vct need to be updated, i.e., removing the newly covered trajectories by the selected vertex. Specifically, the basic updating (Basic) algorithm scans the newly covered trajectories and find the vertices that need to be updated by using the *trajectory-vertex index* $\mathcal{I}_{tv}$.

Algorithm 3 shows the basic updating algorithm. After the current vertex with maximum coverage v_{cur} is selected, the algorithm gets the *newly* covered trajectories Tr_{new}, when adding the v_{cur} to V_{gdy} (Line 1). Specifically, Tr_{new} is calculated as $Tr_{new} = \mathcal{I}_{vt}[v_{cur}] \setminus Tr(V_{gdy})$, where $\mathcal{I}_{vt}[v_{cur}]$ and Tr_{gdy} are the covered trajectories of v_{cur} and V_{gdy} respectivially. Finally,

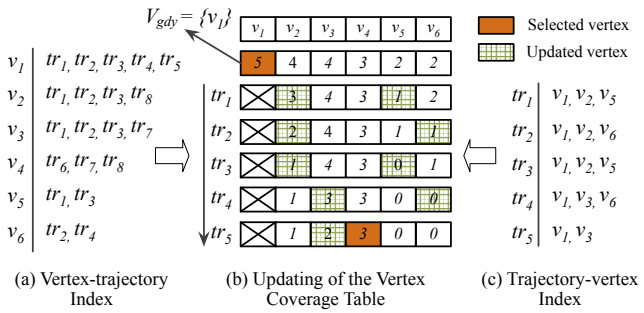


Fig. 5: Illustration of Basic Updating (Basic) Algorithm.

the algorithm goes through the *trajectory-vertex index* for each newly added trajectory in Tr_{new} to update the values (i.e., minus one) in vertex coverage table vct (Line 2-5).

Example. Figure 5 gives a running example of applying the basic updating algorithm to extract the most influential 2-location set using greedy heuristic algorithm. In this example, there are 6 vertices within the spatial region, i.e., $\{v_1, v_2, v_3, v_4, v_5, v_6\}$. The corresponding *vertex-trajectory index* is shown in Figure 5(a), and the *trajectory-vertex index* is shown in Figure 5(c). The updating details of the *vertex coverage table* are demonstrated in Figure 5(b), where each row indicates the updated coverage values after removing a trajectory in Tr_{new}. Initially, the coverage value of each vertex is their original covered trajectories, i.e., $\{5, 4, 4, 3, 2, 2\}$ for $\{v_1, v_2, v_3, v_4, v_5, v_6\}$ respectively.

At the first iteration, v_1 is selected and added to the result set V_{gdy}, and all trajectories covered by v_1, e.g., $Tr_{new} = \{tr_1, tr_2, tr_3, tr_4, tr_5\}$, should be removed from the trajectory sets of other locations. Then, the algorithm utilizes the *trajectory-vertex index* of each trajectory in Tr_{new} to update the coverage values of the remaining vertices, i.e., v_2 to v_6. To be precise, as shown in the first row of Figure 5(b), the algorithm notices that the trajectory tr_1 passes the vertices v_1, v_2 and v_5 from the *trajectory-vertex index*. Thus, the coverage values of v_2 and v_5 should be updated, i.e., decreasing by 1, as tr_1 has been covered. The updating process continues until it checks all the newly covered trajectories tr_1 to tr_5. After the updating phase, the coverage values of the remaining vertices, i.e., $\{v_2, v_3, v_4, v_5, v_6\}$ are $\{1, 2, 3, 0, 0\}$.

Based on the updated vertex coverage table, the greedy heuristic algorithm continues to select the second vertex. In this case, it will select v_4, as v_4 covers the most number of trajectories, i.e., 3. Finally, the algorithm stops, as it has enough candidates.

Performance Analysis. The cost of the selection phase is relatively small, i.e., a linear scan of the coverage values of the remaining vertices, with the time complexity of $O(|V_s|)$. The dominant cost of the algorithm lays in the updating phase as it not only needs to scan the trajectory list, but also the *trajectory-vertex index* one by one. The time complexity of updating phase by using Algorithm 3 is $O(Tr(V_{gdy})) \times \gamma)$, where $Tr(V_{gdy})$ is the total number of trajectories covered by the selected results, and γ is the average length of each trajectory. Obviously, in the case of large-scale trajectory dataset, the basic updating algorithm can be prohibitively inefficient.

5.3 Partition Index Batch Updating

The performance bottleneck of the basic updating algorithm is on the traversal of trajectory-vertex index, which scans every covered trajectory and updates coverage values of the vertices in *vertex coverage table* one by one. Actually, the objective of updating phase is to deduct the number of common trajectories in Tr_{new} from the remaining vertices. In the best scenario, if we can know the exact value changes in the vertex coverage table, and we can just subtract the number without scanning the trajectory-vertex index.

Based on this observation, we propose a *partitioned index batch updating* algorithm. The proposed algorithm takes advantage of the information stored in *vertex-vertex index* and tries to avoid as much scanning operations on the *trajectory-vertex index* as possible. To achieve this, it contains three optimization techniques: 1) *efficient updating strategy*; 2) *index partition*; and 3) *workload-based optimization*.

5.3.1 Efficient Updating Strategy

The main challenge in using the *vertex-vertex index* is that this index records the number of all shared trajectories for each vertex pair. However, if we subtract the number of all shared trajectories from the vertices in current iteration, some trajectories which are covered by the selected vertices in previsous iteration will be removed twice. Therefore, the vertex-vertex index can not be used for the updating directly.

Main idea. We have the observation that, after a selection phase (e.g., v_{cur} is added into V_{gdy}), there can only be two possible cases, when we want to update coverage values by the newly covered trajectories Tr_{new} introduced by v_{cur}:

- *Case 1.* $|Tr_{new}| \leq \frac{1}{2} \cdot |\mathcal{I}_{vt}[v_{cur}]|$: In this case, when v_{cur} is selected, more than half of its trajectories have been covered by the previous selections. In this case, the vertex-vertex index is less useful. We directly apply the basic updating algorithm which scans the newly covered trajectories to update the coverage table.
- *Case 2.* $|Tr_{new}| > \frac{1}{2} \cdot |\mathcal{I}_{vt}[v_{cur}]|$: In this case, when the vertex v_{cur} is selected, less than half of its trajectories have been covered by the previous selections. In this case, we first subtract the number of shared trajectories from the vertex-vertex index for each remaining vertex. As the previous operation subtracted the coverage values of the previously covered trajectories, we need to scan the previously covered trajectories (i.e., the smaller portion) in v_{cur} and add back the coverage values to the vertex coverage table.

Algorithm. Algorithm 4 gives the pseudocode of the efficient updating strategy. When picking a vertex v_{cur}, the algorithm first extracts two sets of trajectories: 1) Tr_{pre}, which is the v_{cur}'s trajectories that has been covered by V_{gdy} (i.e., Line 1); and 2) Tr_{new}, which is the newly covered trajectory by v_{cur} (i.e., Line 2).

Then, the algorithm makes a *smart* decision, which always chooses the smaller portion of trajectories to perform updation. In the case that when Tr_{pre} is relatively larger (i.e., case 1), the algorithm simply calls the basic updating algorithm and utilizes the trajectory-vertex index to update the coverage values (Line 3-4); Otherwise, if the Tr_{pre} is the smaller (i.e., case 2), the algorithm first subtracts the values in the vertex-vertex index from the corresponding entries in the vertex coverage table. After that,

Algorithm 4 Efficient Updating Strategy

Input: Vertex-trajectory index $\mathcal{I}_{vt}$, trajectory-vertex index $\mathcal{I}_{tv}$, vertex-vertex index $\mathcal{I}_{vv}$, result vertices V_{gdy}, selecting vertex v_{cur}, vertex coverage table vct
Output: Updated *vertex coverage table*

1: $Tr_{pre} \leftarrow$ common trajectories between v_{cur} and V_{gdy}
2: $Tr_{new} \leftarrow v_{cur}$'s trajectories minus Tr_{pre}
3: **if** $|Tr_{new}| \leq |Tr_{pre}|$ **then** ▷ Case 1
4: Perform basic updating algorithm
5: **else** ▷ Case 2
6: Update coverage values using vertex-vertex index
7: **for** each $tr \in Tr_{pre}$ **do**
8: **for** each $v \in \mathcal{I}_{tv}[tr]$ **do**
9: **if** $v \in V_s \setminus V_{gdy}$ **then**
10: Add one to the coverage value of v.

the algorithm scans each trajectory in Tr_{pre} and add back the values to the vertex coverage table (Line 5-10).

Example. The updating process of case 1 is straightforward, while case 2 is more complicated. We use Figure 6 to demonstrate the updating process for case 2.

The initial status is presented in Figure 6(a), where the mining process has been executed for a certain number of iterations and chooses v_1 as the next vertex to the result set. The newly covered trajectories introduced by v_1 are marked in grey. The algorithm identifies that $|Tr_{new}| > \frac{1}{2} \cdot |\mathcal{I}_{vt}[v_1]|$(i.e., 4 > 2.5), thus it directly subtracts the values in the vertex-vertex index from the vertex coverage table, as demonstrated in Figure 6(b). Then, because the original vertex-vertex index is calculated including tr_1 (which is already covered in the previous iterations), we essentially have subtracted tr_1 from the vertex coverage table twice. As a result, the algorithm scans the trajectory-vertex index and adds back the value in the vertex coverage table, as depicted in Figure 6(c).

By using the basic updating algorithm, it needs to scan four trajectories, i.e., tr_2, tr_3, tr_4, tr_5, in the update phase. However, as shown in this example, with the efficient update strategy, it only needs to scan one trajectory, i.e., tr_1, in the updating phase.

5.3.2 Index Partition

Applying the efficient updating strategy improves the updating efficiency comparing to the basic updating algorithm. However, even by using the efficient updating strategy, we may still need to scan half of the trajectory list (to perform the updating) in the worst case. Figure 7(a) gives an example, the vertex v_j (e.g., a busy intersection in the downtown area) is covered with millions of trajectories and the number of its Tr_{pre} and Tr_{new} is equal. In this case, millions of trajectoris is still expected in the updating phase. To this end, we propose *index partition* technique to

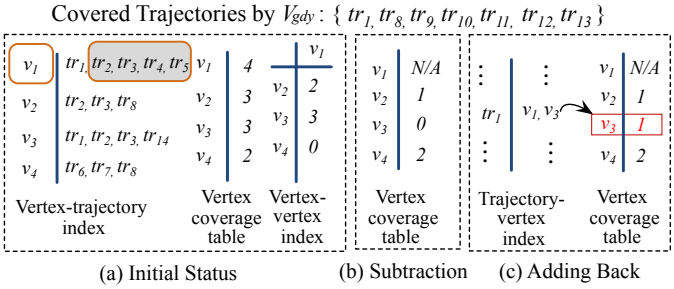


Fig. 6: Illustration of Efficient Updating Strategy.

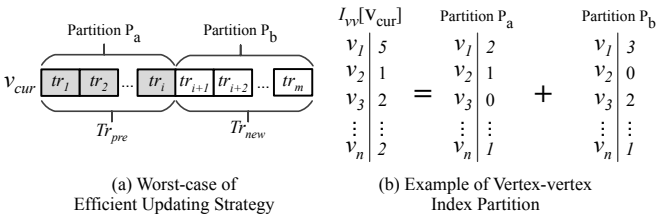


Fig. 7: Motivation of Index Partition.

improve the efficiency by further reducing the number of scanned trajectories.

Data structure. Partitioned vertex-vertex index is used, where each entry, i.e., v_{cur}, in vertex-vertex index is divided into multiple groups. Each group contains a disjoint trajectory subset of v_{cur}, and records the common trajectory numbers shared between v_{cur} and other vertices within the group. Figure 7(b) gives an example of partitioned vertex-vertex index, where the trajectories of an entry (e.g., v_{cur}) are divided into two groups. From the partitioned vertex-vertex index, we can identify that the common trajectories between v_{cur} and v_1 in trajectory partition P_b is 3.

Main idea. The main intuition of the apply partition on the *vertex-vertex index* originates from that, for the Figure 7(a)'s example, if we have a partition of vertex-vertex index containing just the newly covered trajectories available, we can directly subtract that number in the table without scanning any trajectories. Moreover, we have the observation that even if we cannot have such optimal partition scenario, we can still reduce the total number of trajectories scans in the updating phase significantly, as long as the vertex-vertex index are divided into multiple partitions.

The idea of using index partition technique is that, after dividing the vertex-vertex index into multiple partitions, the efficient updating strategy can be applied on each of the partition. In this way, we can further reduce the number of scan operations and improve the efficiency. The performance improvement is guaranteed by Lemma 1.

Lemma 1. *The number of trajectory scans with the index partition is always smaller than only with the efficient updating strategy.*

Proof. Suppose the trajectories of the vertex v_{cur} are divided into $\rho \geq 1$ partitions. With only the efficient updating strategy, the algorithm will always select the smaller size part between Tr_{pre} and Tr_{new} for the updating, thus the total number of trajectories scanned is equal to $\min\{\sum_{i=1}^{\rho} |P_i.Tr_{new}|, \sum_{i=1}^{\rho} |P_i.Tr_{pre}|\}$, where $P_i.Tr_{pre}$ and $P_i.Tr_{new}$ are the previously covered and newly added trajectories of partition P_i respectively. By combining the index partition method and efficient updating strategy, the algorithm will always choose the smaller part in each partition to perform the updating, the number of scanned trajectories is $\sum_{i=1}^{\rho} \min\{|P_i.Tr_{new}|, |P_i.Tr_{pre}|\}$. It can easily derive that $\sum_{i=1}^{\rho} \min\{|P_i.Tr_{new}|, |P_i.Tr_{pre}|\} \leq \min\{\sum_{i=1}^{\rho} |P_i.Tr_{new}|, \sum_{i=1}^{\rho} |P_i.Tr_{pre}|\}$. As a result, Lemma 1 holds. □

Algorithm. Algorithm 5 gives the pseudocode of Partition Index Batch Updating (BIPU) algorithm. For each trajectory partition P_i of vertex v_{cur}, it first identifies the $P_i.Tr_{pre}$ and $P_i.Tr_{new}$ of each trajectory partition (Line 2). After that, the algorithm performs the efficient updating strategy by using the corresponding partition of partitioned vertex-vertex index ($\mathcal{I}_{vv}[v_{cur}].P_i$) to update the corresponding coverage value, i.e., it always chooses

Algorithm 5 Partition Index Batch Updating (PIBU) Algorithm

Input: Vertex-trajectory index $\mathcal{I}_{vt}$, trajectory-vertex index $\mathcal{I}_{tv}$, partitioned vertex-vertex index $\mathcal{I}_{vv}$, result vertices V_{gdy}, selecting vertex v_{cur}, vertex coverage table vct
Output: Updated *vertex coverage table*
1: **for** each $P_i \in \mathcal{I}_{vv}[v_{cur}]$ **do**
2: Identify $P_i.Tr_{pre}$ and $P_i.Tr_{new}$
3: Perform efficient updating strategy using $\mathcal{I}_{vv}[v_{cur}].P_i$

to scan the smaller part of trajectories to perform the updating (Line 3).

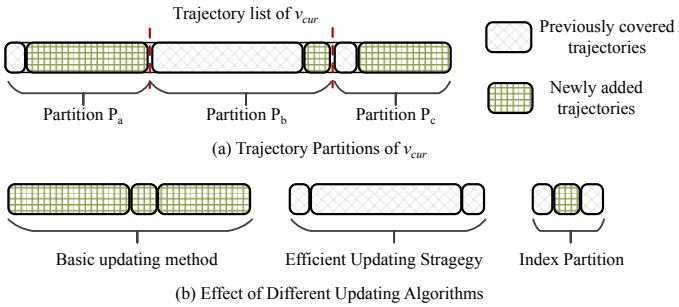


Fig. 8: Updating Example with Three Techniques.

Example. Figure 8 demonstrates an overall comparison between three different update techniques. Figure 8(a) gives the initial status, the vertex v_{cur} is selected in current iteration, a part of its trajectories are covered by the previous iterations (i.e., marked in grey shade), and another part of trajectories are newly added trajectories (i.e., marked in green shade). There are three imperfect partitions, i.e., P_a, P_b and P_c, depicted by the brackets. Figure 8(b) demonstrates the updating cost for three different updating algorithms: 1) the basic updating algorithm, which needs to scan all the newly added trajectories (i.e., the green part) in the update phase; 2) efficient updating strategy makes a better choice, which always scans the smaller part between the previously covered trajectories and newly added trajectories; 3) index partition takes advantage of the partitions in the vertex-vertex index and performs the efficient updating strategy on each partition (i.e., scanning the smaller part in each partition.) As a result, the index partition technique can significantly reduce the number of trajectory scans.

5.3.3 Workload-based Optimization

Lemma 1 proves the superiority of index partition in the updating phase. However, the index partition also introduces a significant storage overhead. The storage overhead is $O(\rho \cdot |V^2|)$, where ρ is the average number of partitions on each entry. It would be memory infeasbile in the case of large road network, e.g., the road network of Tianjin has about 100K vertices, which means 20G entries in partitioned vertex-vertex index when $\rho = 2$. Another issue in the index partition is that how to effectively making the partition of trajectories. To this end, we further propose *workload-based optimization*, which includes two techniques: 1) *workload-based partition*, and 2) *workload-based selective indexing*.

Workload-based Partition. The main intuition to partition the vertex-vertex index is that we want to partition the trajectories associated with each entry (i.e., vertex v) into different groups, where trajectories in each group share more common vertices. A straightforward method is trajectories clustering. However, this method is infeasible as it requires $|Tr|^2$ trajectory similarity computations. After we exam a set of mining results, we find an observation that, in many cases, a set of vertices are often selected in the same order. And a mining workload will naturally divide the trajectories in a vertex into multiple partitions. If we can keep track of this partitions during the process, in the best case scenario, we may not need to scan any trajectory in the updating phase.

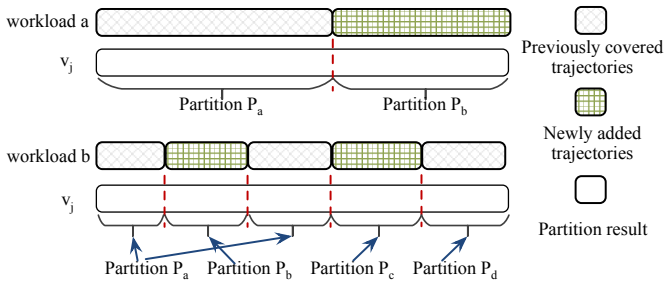


Fig. 9: Example of Workload-based Index Partition

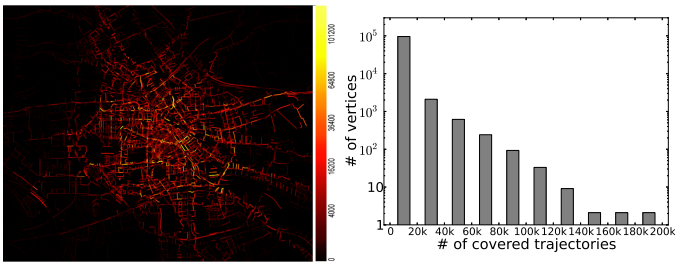
Figure 9 illustrates the workload-based partition method. Given a vertex v_j which are selected by mining workload a in some iteration, the trajectories covered by vertex v_j are naturally divided into two parts, e.g., P_a and P_b, as its Tr_{new} and Tr_{pre}. For another mining workload, i.e., b, which also selected v_j, the original two partitions of v_j will be further divided into four partitions, i.e., P_a, P_b, P_c and P_d as shown in this figure. To avoid very small trajectory partition, we use a parameter ξ to control the partition process. The workload-based partition stops for a trajectory parition when its size is less than ξ.

By using the workload-based partition, we can divide the trajectories into several partitions without calculating their pair-wise similarity.

Workload-based Selective Indexing. As discussed before, it is infeasible to store all the *partitioned vertex-vertex index* in the memory. Thus, we need to select a subset of vertices to index. There are two basic intuitions to select the vertices for indexing: 1) the number of trajectories passed the vertex is large, otherwise, we can directly apply basic updating algorithm efficiently; and 2) it is wasteful to index the vertices which would never hit by the mining tasks. The reason is that some vertices even with large number of trajectory coverage are never selected in the mining tasks, because these vertices shares many common trajectories with other vertices that are selected earlier (e.g., several consecutive vertices on a major road). To this end, we also apply the workload-based method to select the vertices to index. The main idea is that after running a set of mining tasks, we get all the vertices in the result sets and only index the vertices with highest hit ratio. In this way, the indexed vertices are with large trajectory coverage and can provide high hit ratio for the coming mining tasks.

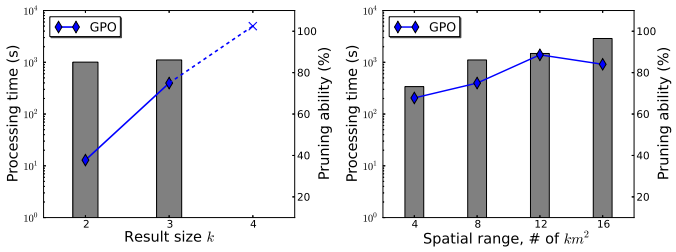
As a remark, for the newly selected vertex which is not in the *partitioned vertex-vertex index*, we can easily use the basic updating algorithm to update the coverage values. Our system can easily adopt typical *caching* policies (e.g., *least recently used*, *first-in-first-out* and *random replacement* [24]) to dynamically replace the vertices in the index to adapt to the new mining workloads.

6 Experiments



(a) Trajectory distribution (b) Vertex coverage distribution

Fig. 10: Trajectory Distributions in Tianjin.



(a) Varying result size, k (b) Varying spatial range

Fig. 11: Evaluations of GPO.

In this section, we first provide a set of efficiency experiments based on large-scale taxi trajectories collected from Tianjin (Section 6.1). After that, we provide two case studies: 1) charging station placement based on taxi trajectories in Beijing, and 2) advertisement placement based on check-in data collected from Foursquare in NYC, to demonstrate the effectiveness of our proposed system (Section 6.2).

6.1 Efficiency Study

6.1.1 Dataset & Settings

Road networks. We extract the road network of Tianjin, which contains 99,007 vertices and 133,726 road segments. The road network covers an area of 123×187 km^2 with a total length of 32,487 km [25].

Taxi trajectories. The GPS trajectory dataset is generated by 3,501 taxicabs from Tianjin in 61 days [25]. It contains 4,509,519 trajectories (segmented based on the passenger on/off events) and the total number of GPS points reaches 753,059,212. The average sampling rate is 24.05 seconds per point. After map-matching, the total length of trajectories is 46,028,698 where each trajectory passes 10.2 vertices on average.

Figure 10(a) shows the spatial distribution of trajectories using a heat map. The lighter the denser, and most of the trajectories are crowded within the downtown area. As shown in Figure 10(b), most of the vertices, e.g., about $95,904$ vertices, are only traversed by trajectories with the size of 0 to $20k$, and the maximum number of trajectories covered by a vertex is about $200k$.

TABLE 1: Parameter Settings

Type	Parameter	Range
Optimal	Result size, k	2, **3**
	Area of queries, $\|R\|$ (km^2)	4, **8**, 12, 16
Greedy	Result size, k	10, **20**, 30, 40
	Area of queries, $\|R\|$ (km^2)	**Rand**, 20, 40, 60, 80
	Trajectories datasets, d (# of days)	15, 31, 46, **61**

Table 1 summarizes the ranges of investigated parameters in our efficiency studies with their default values in bold. The performance are evaluated on a machine running Ubuntu 12.04 with Intel Core 6-Cores (12-Threads) i7-3930K 3.2GHz and 16GBytes of main memory. In each experiment, we vary a single parameter, while setting the others to their default values.

6.1.2 Optimal Location Set Mining

We have proposed the group pruning optimal (GPO) algorithm for exact results with small size in section 4. We demonstrate the experimental results of GPO in this subsection.

Different k Values. Figure 11(a) shows the processing time (in grey bars) and pruning ability (in blue line) of GPO with different k values. The processing time grows exponentially with the increase of k, and we have to terminate the execution of GPO when $k = 4$, as the processing time exceeds 5000 seconds, despite of the fact that our GPO have pruned at least 85% candidates in all settings.

Different Sizes of Spatial Region. Figure 11(b) shows the processing time of GPO versus the size of the spatial region. The processing time increases with the size of spatial region, e.g., from 4 km^2 to 12 km^2; but decreases slightly at 16 km^2 as the GPO can prune 96.3% combinations in this setting. It takes 727.1 seconds to complete a round of mining task on average when $k = 3$.

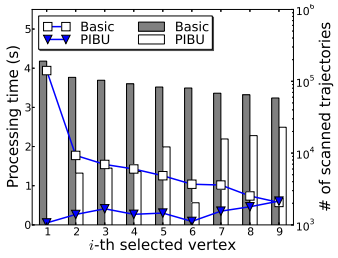
In summary, the GPO algorithm provides an impressive pruning ability on finding the exact location set. However, the GPO algorithm only works for small mining parameters (i.e., small spatial region and k values), therefore, is difficult to apply it to support the interactive mining process for large mining parameters.

6.1.3 Approximate Location Set Mining

In this subsection, we demonstrate the experimental results for our efficient approximate solutions. Under the default settings, the construction time of $\mathcal{I}_{vv}$ is 3231.87 seconds, the average number of partitions of each vertex in the index is 2.322. We set Δ to 1000, and the memory usages of $\mathcal{I}_{spatial}$, $\mathcal{I}_{vt}$, $\mathcal{I}_{tv}$ and $\mathcal{I}_{vv}$ are 2.99 MB, 829 MB, 846 MB and 1,959 MB. We compare two algorithms:(1) Basic Updating Algorithm (i.e., Basic) (Section 5.2) and our Partition Index Batch Updating (i.e., PIBU) Algorithm (Section 5.3).

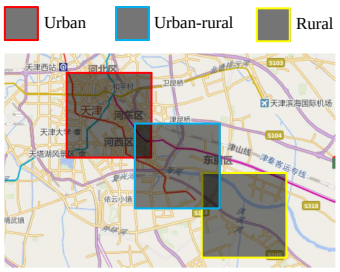
Performance Overview. Before evaluating the effect of various mining parameters, we give the performance overview for our proposed methods. We select a spatial region with $3,406$ candidate locations, and aim at mining 10-locations set. The selected result set covers a total of $911,244$ trajectories. In terms of the performance, the Basic approach takes 14.22 seconds, while our proposed method PIBU is 5.02 times faster with only 2.83 seconds. Figure 12(b) shows the number of scanned trajectories (in bars) and the processing time (in lines) for vertex selection iteration. As a remark, the 10-th selected vertex does not need to update. In the Basic algorithm, the number of scanned trajectories for a vertex equals to its newly added trajectories Tr_{new}. Totally,

(a) Selected vertices, k = 10

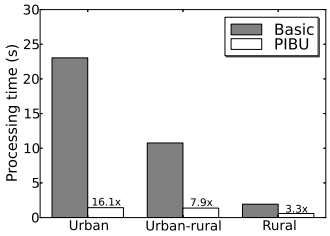


(b) Updating cost for each vertex

Fig. 12: Performance overview

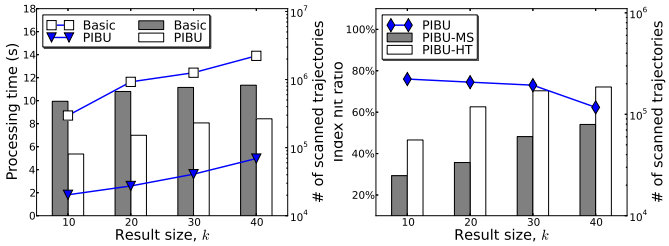


(a) Spatial regions



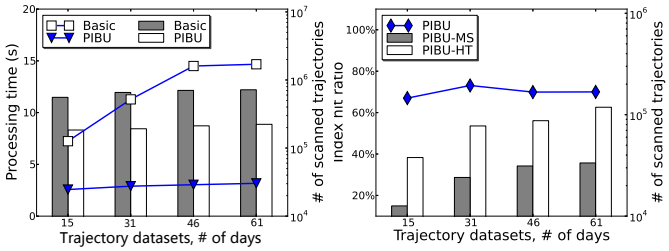
(b) Varying spatial regions

Fig. 14: Effect of Different Spatial Regions



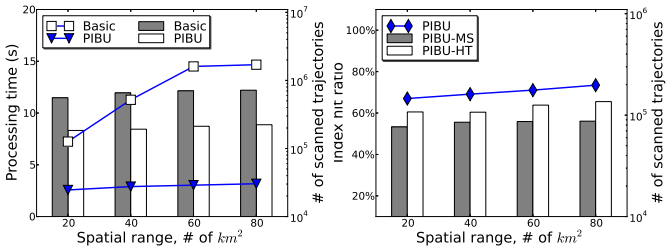
(a) Varying result size, k

(b) PIBU, varying result size, k



(c) Varying dataset size, d

(d) PIBU, varying dataset size, d



(e) Varying spatial range, $|R|$

(f) PIBU, varying spatial range, $|R|$

Fig. 13: Scalability Evaluations of Greedy Solutions

the Basic algorithm scans $905,623$ trajectories. PIBU can significantly reduce the number of scanned trajectories. Totally, PIBU scans only $87,330$ trajectories, which is ten times less than the Basic approach.

Scalability. In this part, we evaluate three scalability parameters of the proposed solutions, i.e., result size k, size of trajectories datasets d, and area of mining spatial region $|R|$.

Figure 13(a) shows the processing time (by lines) and the number of scanned trajectories (by bars) in updating phase, with different k values. The processing time and number of scanned trajectories increase linearly with k for both algorithms. Our efficient updating algorithm (i.e., PIBU) is 3.9 times faster than Basic algorithm on average.

Figure 13(c) shows the processing time (by lines) and the number scanned trajectories (by bars) by varying the size of trajectories datasets, d. Not surprisingly, the processing time and number of scanned trajectories increase with d for both approaches. However, our PIBU achieves more performance gain comparing to the Basic updating method, with the larger trajectory data size, e.g., from 3.2 times (d = 15 days) to 4.5 times (d = 61 days). This chart confirms that our PIBU method is able to handle the queries over large scale trajectory datasets.

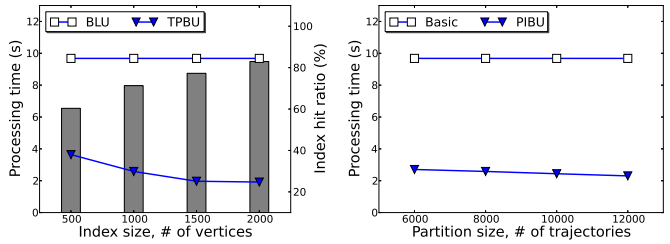
Figure 13(e) shows the processing time (by lines) and the number scanned trajectories (by bars) versus the area of mining region, $|R|$. The processing time increases for both approach, as more vertices in the selected region trend to introduce more covered trajectories. Our proposed method PIBU achieves at least 3.8 times performance gain comparing to the Basic method with all settings.

Figure 13(b), 13(d) and 13(f) illustrate the index hit ratio of PIBU by lines, the number of scanned trajectories for vertices with index (PIBU-HT) or without index (PIBU-MS) by bars. The number of scanned trajectories for vertex with index is larger than the vertex without index in all settings as the vertices in the index typically cover more trajectories than the non-indexed vertices. In a summary, the index hit ratio reduces with the increasing of k; increases slightly with the increasing of $|R|$; there is no significantly difference with the size of trajectory datasets, d.

Further Exploration by Varying Regions. We further explore the effect of different spatial regions. There are three different types of spatial region as shown in Figure 14(a), i.e., *urban*, *urban-rural*, *rural*. Given three different regions with equal area, it is very time consuming to find the most influential k-location set in the *urban* region by Basic method as the vertices in this region typically cover massive trajectories. Nevertheless, our proposed method PIBU works very efficiently as it can significantly reduce the number of scanned trajectories in the updating phase, e.g., 16.1 times faster than Basic as shown in Figure 14(b). The performance gap between PIBU and Basic reduces in the *rural* region as the $\mathcal{I}_{vv}$ in PIBU trends to index less vertices in the rural region. As a result, our PIBU approach is much more efficient in the urban area, where we expect to have more mining tasks in the real world.

Tuning of System Parameters. We also test the robustness of PIBU by varying two system parameters, i.e., number of indexed vertices χ and trajectory partition size ξ.

We first study the effect of χ, which can be determined by the size of available memory in the system. Figure 15(a) shows the effect of χ. The memory consumption increase with χ, e.g., it consumes 972 MB memory for indexing $\mathcal{I}_{vv}$ when $\chi = 500$, and 3891 MB when $\chi = 2,000$. The processing time (by lines) decreases from 3.6 seconds to 1.9 seconds as the index hit ratio



(a) Effect of indexed size, χ (b) Effect of partition size, ξ

Fig. 15: Tuning of System Parameters

increases with χ (by bars). According to our experiments, we pick $\chi = 1,000$ as the default setting as it gives a good trade-off between the time efficiency and memory consumption.

Finally, we study the effect of trajectory partition size ξ. As shown in Figure 15(b), PIBU is not very sensitive to ξ, and we set $\xi = 8,000$ as our default setting.

6.2 Case Studies

In this subsection, we provide two case studies, i.e., billboards placement and charging station placement, to demonstrate the applicability of our proposed system.

6.2.1 Billboards Placement

Task. In this case study, a business owner would like to put three billboards in New York City (NYC) to promote their products. Assuming that each billboard can influence a spatial region with size of 500×500 m^2, and we want to maximize the influences of the billboards (i.e., the number of covered unique users) as much as possible.

Dataset. We use a location-based social network dataset as a sample of human movements in the city. The dataset is collected from Foursquare [26], which contains 221,128 tips generated by 49,062 users in NYC. We divide NYC into equal size (i.e., 500×500 m^2) grids as demonstrated in Figure 16.

Results. The mining results are presented in Figure 16 as below:

• Figure 16(a) illustrates a heat map of the original users' check-in distributions in NYC, where the lighter area indicates more users' check-ins. The three selected girds are the areas with the most number of users' check-ins, which are: 1) Manhattan Mall, 2) Broadway Shopping Area A, and 3) Union Square Stations. However, only counting the number of users' check-ins may not maximize the total number of covered unique users, as some users may check-in multiple times in the same grid.

• Figure 16(b) illustrates a heat map of unique user's check-in distributions in NYC, where the lighter color indicates more number of unique users visited the area. This approach eliminates the scenario of one user's multiple check-ins in a same grid. By using this elimination, the three selected grids with maximum unique user's check-in are: 1) Broadway Shopping Area A, 2) Broadway Shopping Area B and 3) East Village. However, it still suffers from the drawback of overlapped users in the selected three grids.

• Figure 16(c) shows the result of our solution, where we apply our technique for approximate location set mining. The three selected grids (i.e., the most influential 3-location set) are: 1) Broadway Shopping Area A, 2) Union Square Station and 3) Chinatown. Our selection captures more unique users (6,320) than the other two approaches (i.e., 5,625 and 5,543). Our approach not only covers a more diverse grids on the map, but also with a very diverse semantics, which includes a shopping area, a transportation center and a dining area.

As a result, our solution for selecting the most influential 3-location set in NYC is more effective (influence more unique users in the city) than all the other two approaches.

6.2.2 Charging Station Placement

Task. In this case study, the government wants to deploy three electric vehicle (EV) charging stations in Wangjing Area (a district in Beijing) to promote the green-energy. As the charing station is a public service, we need to cover as many users' travels as possible. Moreover, the placement of EV charging stations also need consider the following three domain constraints: 1) the selected location needs to have space for parking; 2) the nearby area needs a diverse array of POI categories; and 3) each two selected locations should not be very close to each other.

Dataset. We use a GPS trajectories of 33,619 taxicabs in Beijing as a sample of users' vehicle movements. We perform a map-matching algorithm to map the trajectories on the road network of Beijing, which contains 186,266 vertices and 249,080 segments. The target Wangjing area is demonstrated as the shaded polygons in Figure 17.

Results. Figure 17 demonstrates the results using our most influential k-location set technique with multiple iterations from the field experts.

• Figure 17(a) gives the selection results in the first iteration, where three intersections are selected on the map (marked as red, orange and green). The three selected locations covers a total of 11,558 trajectories in the area. However, when we exam closely on each locations, we find that: 1) Node 2 and Node 3 do not have enough places for parking, as demonstrated in the street map view; and 2) the nearby POI distribution of Node 2 and Node 3 does not satisfy the diversity requirement (i.e., without any medical services), illustrated in the POI distribution view[6]. As a result, we only keep Node 1 in the result, and perform a new selection iteration.

• Figure 17(b) gives the selection results for the next two iterations. In the second iteration, we find a new set of three locations, where we keep Node 1 and Node 2 in the result and remove Node 3, as it does not have enough space for parking. On the third iteration, we find a new Node 3. However, it still does not statisfy our requirement, as it is very close to Node 1 in our result, as they are intersections of a very popular bi-direction entry point of Wangjing area. Thus, we need to remove Node 3 and perform our algorithm continuously.

• Figure 17(c) gives the final result in the Wangjing area which is demonstrated on the map (i.e., locations which are marked as red, blue and yellow). All the selected locations fulfill our requirement, where each of them has enough space for parking (demonstrated in the street view) and statisfies the POI diversity (shown in the bar chart). Most importantly to note here, the 3-location set covers a total of 10,993 trajectories, which is very close to the total number covered trajectories (i.e., 11,558) in the first iteration.

6. The POI distributions is calculated by aggregating the POIs within 1 km range of the target location.

(a) Check-in Distribution (b) Unique Check-in Distribution (c) Most Influentitial k-location Set

Fig. 16: Placing Billboards in NYC

7 Related Works

Trajectory Query Processing. The increasing pervasiveness of location-acquisition techniques has enabled collection of massive trajectory data with a board range of applications [27][2][28][29][30]. [2] studies the problem of discovering the gathering patterns from trajectories. [28] proposes to estimate the travel-time of a path in real-time in a city based on the GPS trajectories of vehicles. [29] studies a query which finds the most frequent path of user-specified source and destination from historical trajectories. However, their objectives are different from ours, we aim to find the locations covers more trajectories within a spatial region.

Location Selection. The location selection problem has been extensively studied by researchers in both operations research and database communities [31][32][33][34][35][36][37][38][39][40]. The classical location selection problem takes two spatial datasets C = $\{c_1,c_2,...,c_n\}$ (i.e., clients) and F = $\{f_1,f_2,...,f_m\}$ (i.e., candidate locations for facilities) as the input, and returns k (k > 0) locations in F that optimizes a predefined metric for the clients. Based on the objective function, these works can be divided into two categories, i.e., the MinSum model [36], [37], [38] and the MinMax model [35]. The MinSum (MinMax) model aims at locating k facilities such that the average (max) cost to reach all clients can be minimized. More specifically, [33] studies an efficient solution to locate one optimal location in road network. [34] tackles the problem of optimal retail store placement in the context of location-based social networks. However, none of them focus on selecting the locations that covers the maximum number of trajectories within a spatial region.

Maximum Coverage. Maximum coverage problem has great utility for several real world applications [11][12][13][14]. The widely used greedy implementation (cf. Section 5.1) does not behave well when the data is disk resident [11][13]. [13] proposes an efficient disk-based algorithm which can find a solution that is provably close to that of greedy. [12] introduces the online set-cover problem which focuses on minimizing the number of total selected items to cover every requirement coming online. Besides, [14] proposes to maintain k blogs[7] to cover the list of interesting topics for a given user. Given a Netflix user, [41] aims to find k other users which can cover the like or un-like movies of a given user. And it is the most similar work to ours. However, their solution aims at approximating the greedy solution by using *coverage oracle*. In a summary, all the works aforementioned consider different problem as ours, thus their solutions can not be applied directly.

7. The topics of a blog can change after the updating.

8 Conclusion

This work presents a comprehensive study on mining the most influential k-location set from massive trajectory dataset. It has many potential applications in resource allocation applications. Our study covers both the exact and approximate solutions. The exact solution works for small k and spatial region R, while the efficient approximate algorithm is studied to address the large k and spatial region R. Extensive experiments on real datasets show that our proposed solutions is up to an order of magnitude faster than the baseline solutions. We also demonstrate the applicability of our proposed solution by performing two case studies: 1) Billboard placement in NYC and 2) EV charging station placement in Beijing. Finaly, we have already deployed our system and it is available via [15].

9 Acknowledgments

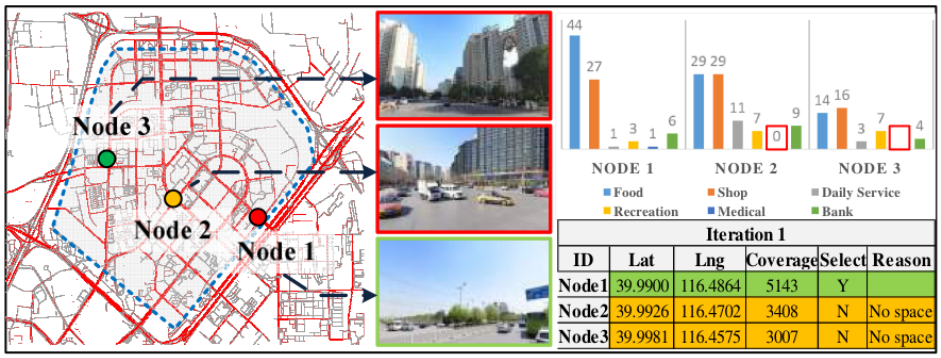
This work was kindly supported by the NSF of China (Grant No. 61672399, No. U1609217 and No. 61502416), the National 973 Program of China (No. 2015CB352400 and No. 2015CB352503), MYRG2015-00070-FST, MYRG2017-00212-FST from UMAC Research Committee, FDCT/116/2013/A3, FDCT/007/2016/AFJ from Macau FDCT, NSF CRII grant CNS-1657350, and a research grant from Pitney Bowes, Inc.

References

[1] L. A. Tang, Y. Zheng, J. Yuan, J. Han, A. Leung, C. Hung, and W. Peng, "On discovery of traveling companions from streaming trajectories," in *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, 2012, pp. 186–197.

[2] K. Zheng, Y. Zheng, N. J. Yuan, and S. Shang, "On discovery of gathering patterns from trajectories," in *ICDE*, 2013, pp. 242–253.

[3] A. Kharrat, I. S. Popa, K. Zeitouni, and S. Faïz, "Clustering algorithm for network constraint trajectories," in *Headway in Spatial Data Handling, 13th International Symposium on Spatial Data Handling, Montpellier, France, 23-25 July 2008*, 2008, pp. 631–647.



Iteration 1

ID	Lat	Lng	Coverage	Select	Reason
Node1	39.9900	116.4864	5143	Y	
Node2	39.9926	116.4702	3408	N	No space
Node3	39.9981	116.4575	3007	N	No space

(a) Charging Station Suggestions in the 1^{st} iteration

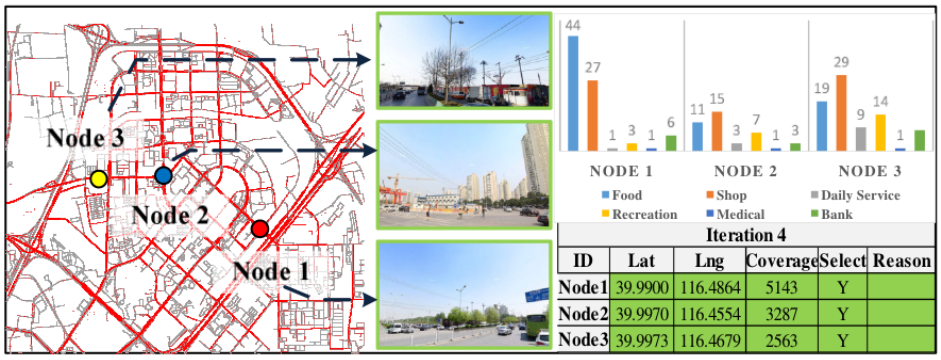
Iteration 2

ID	Lat	Lng	Coverage	Select	Reason
Node1	39.9900	116.4864	5143	Y	
Node2	39.9970	116.4554	3287	Y	
Node3	39.9927	116.4704	2768	N	No space

Iteration 3

ID	Lat	Lng	Coverage	Select	Reason
Node1	39.9900	116.4864	5143	Y	
Node2	39.9970	116.4554	3287	Y	
Node3	39.9903	116.4862	2689	N	Close to 1

(b) Iterative Process in the 2^{nd} and 3^{rd} iterations



Iteration 4

ID	Lat	Lng	Coverage	Select	Reason
Node1	39.9900	116.4864	5143	Y	
Node2	39.9970	116.4554	3287	Y	
Node3	39.9973	116.4679	2563	Y	

(c) Charging Station Suggestions in the 4^{th} iteration

Fig. 17: Placing Charging Stations in Wangjing Area, Beijing.

[4] T. Sohn, A. Varshavsky, A. LaMarca, M. Y. Chen, T. Choudhury, I. E. Smith, S. Consolvo, J. Hightower, W. G. Griswold, and E. de Lara, "Mobility detection using everyday GSM traces," in *UbiComp 2006: Ubiquitous Computing, 8th International Conference, UbiComp 2006, Orange County, CA, USA, September 17-21, 2006*, 2006, pp. 212–224.

[5] J. Lee, J. Han, and X. Li, "Trajectory outlier detection: A partition-and-detect framework," in *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, 2008, pp. 140–149.

[6] Y. Zheng, X. Xie, and W. Ma, "Geolife: A collaborative social networking service among user, location and trajectory," *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 32–39, 2010.

[7] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "T-drive: Enhancing driving directions with taxi drivers' intelligence," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 1, pp. 220–232, 2013. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2011.200

[8] J. Bao, Y. Zheng, D. Wilkie, and M. F. Mokbel, "A survey on recommendations in location-based social networks," *ACM Transaction on Intelligent Systems and Technology*, 2013.

[9] D. Liu, D. Weng, Y. Li, J. Bao, Y. Zheng, H. Qu, and Y. Wu, "SmartAdp: Visual analytics of large-scale taxi trajectories for selecting billboard locations," *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE VAST 2016*, vol. 20, no. 12, 2014.

[10] F. Chierichetti, R. Kumar, and A. Tomkins, "Max-cover in map-reduce," in *WWW*. ACM, 2010, pp. 231–240.

[11] D. S. Hochba, "Approximation algorithms for np-hard problems," *SIGACT News*, vol. 28, no. 2, pp. 40–52, 1997.

[12] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. Naor, "The online set cover problem," in *STOC*, 2003, pp. 100–105.

[13] G. Cormode, H. J. Karloff, and A. Wirth, "Set cover algorithms for very large datasets," in *CIKM*, 2010, pp. 479–488.

[14] B. Saha and L. Getoor, "On maximum coverage in the streaming model & application to multi-topic blog-watch," in *SDM*, 2009, pp. 697–708.

[15] "Smartadp," smartadp.chinacloudapp.cn.

[16] D. S. Hochbaum, "Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems," in *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1996, pp. 94–143.

[17] Y. Li, J. Bao, Y. Li, Y. Wu, Z. Gong, and Y. Zheng, "Mining the most influential k-location set from massive trajectories," in *SIGSPATIAL*. ACM, 2016.

[18] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang, "Map-matching for low-sampling-rate gps trajectories," in *SIGSPATIAL*. ACM, 2009, pp. 352–361.

[19] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The r+-tree: A dynamic index for multi-dimensional objects," 1987.

[20] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, 1999.

[21] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[22] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz, "Analysis of the clustering properties of the hilbert space-filling curve," *TKDE*, vol. 13, no. 1, pp. 124–141, 2001.

[23] U. Feige, "A threshold of ln n for approximating set cover," *Journal of the ACM (JACM)*, vol. 45, no. 4, pp. 634–652, 1998.

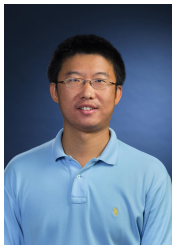
[24] J. E. Smith and J. R. Goodman, "Instruction cache replacement policies and organizations," *Trans. Computers*, vol. 34, no. 3, pp. 234–241, 1985.

[25] Y. Li, Y. Zheng, S. Ji, W. Wang, L. H. U, and Z. Gong, "Location se-

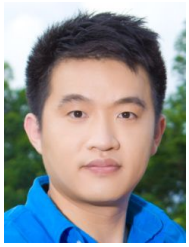
lection for ambulance stations: a data-driven approach," in *SIGSPATIAL*. ACM, 2015, pp. 85:1–85:4.
[26] J. Bao, Y. Zheng, and M. F. Mokbel, "Location-based and preference-aware recommendation using sparse geo-social networking data," in *SIGSPATIAL*. ACM, 2012, pp. 199–208.
[27] Y. Zheng, "Trajectory data mining: an overview," *TIST*, vol. 6, no. 3, p. 29, 2015.
[28] Y. Wang, Y. Zheng, and Y. Xue, "Travel time estimation of a path using sparse trajectories," in *SIGKDD*, 2014, pp. 25–34.
[29] W. Luo, H. Tan, L. Chen, and L. M. Ni, "Finding time period-based most frequent path in big trajectory data," in *SIGMOD*, 2013, pp. 713–724.
[30] Y. Zheng, Y. Liu, J. Yuan, and X. Xie, "Urban computing with taxicabs," in *UbiComp*, 2011, pp. 89–98.
[31] Y. Li, J. Luo, C.-Y. Chow, K.-L. Chan, Y. Ding, and F. Zhang, "Growing the charging station network for electric vehicles with trajectory data analytics," in *ICDE*, 2015.
[32] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit, "Local search heuristics for k-median and facility location problems," *SIAM J. Comput.*, vol. 33, no. 3, pp. 544–562, 2004.
[33] X. Xiao, B. Yao, and F. Li, "Optimal location queries in road network databases," in *ICDE*, 2011, pp. 804–815.
[34] D. Karamshuk, A. Noulas, S. Scellato, V. Nicosia, and C. Mascolo, "Geo-spotting: mining online location-based services for optimal retail store placement," in *SIGKDD*, 2013, pp. 793–801.
[35] Z. Chen, Y. Liu, R. C. Wong, J. Xiong, G. Mai, and C. Long, "Efficient algorithms for optimal location queries in road networks," in *SIGMOD*, 2014, pp. 123–134.
[36] K. Deng, S. W. Sadiq, X. Zhou, H. Xu, G. P. C. Fung, and Y. Lu, "On group nearest group query processing," *TKDE*, vol. 24, no. 2, pp. 295–308, 2012.
[37] S. Li and O. Svensson, "Approximating k-median via pseudo-approximation," in *STOC*, 2013, pp. 901–910.
[38] J. Qi, R. Zhang, L. Kulik, D. Lin, and Y. Xue, "The min-dist location selection query," in *ICDE*, 2012, pp. 366–377.
[39] D. Yan, Z. Zhao, and W. Ng, "Efficient processing of optimal meeting point queries in euclidean space and road networks," *Knowl. Inf. Syst.*, vol. 42, no. 2, pp. 319–351, 2015.
[40] Y. Zheng, L. Capra, O. Wolfson, and H. Yang, "Urban computing: Concepts, methodologies, and applications," *ACM TIST*, vol. 5, no. 3, pp. 38:1–38:55, 2014.
[41] I. Antonellis, A. D. Sarma, and S. Dughmi, "Dynamic covering for recommendation systems," in *CIKM*, 2012, pp. 26–34.

Yuhong Li is currently a senior algorithm engineer in Security Department at Alibaba Group. He got his Ph.D. degree from the Department of Computer and Information Science, University of Macau, under the supervision of Prof. Leong Hou U and Prof. Zhiguo Gong. His research focuses on spatio-temporal database, data mining for social security and high performance parallel computing.

Jie Bao is currently an associate researcher in Urban Computing Group at MSR Asia. He currently works on large scale urban data management on the cloud (i.e., Microsoft Azure). His main research interests include: Spatio-temporal Data Management/Mining, Database Systems, and Distributed Computing Platforms (e.g., Hadoop, Storm and Spark). He got his Ph.D. degree from the Department of Computer Science & Engineering at University of Minnesota at Twin Cities under the advisory of Prof. Mohamed Mokbel in 2014. Before joining MSR, he was a research intern in IBM T.J Watson Lab in 2013, and Microsoft Research, Asia in 2011. He has published over 30 research papers (e.g., SIGKDD, ICDE, VLDB, SIGMOD and SIGSPATIAL) in refereed journals and conferences.



Yanhua Li is currently an Assistant Professor in the Computer Science Department, Worcester Polytechnic Institute (WPI). He received the Ph.D. degree in electrical engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2009, and the Ph.D. degree in computer science from the University of Minnesota at Twin Cities, Minneapolis, MN, USA, in 2013. From 2011 to 2013, he worked as an Intern with Bell Labs, NJ, Microsoft Research Asia, and Huawei Research Labs of America. From August 2013 to December 2014, he worked as a Researcher with Huawei Noah's Ark Lab, Hong Kong. He is currently an Assistant Professor with the Department of Computer Science, Worcester Polytechnic Institute, Worcester, MA, USA. His research interests include network performances and data analytics in many contexts, such as communication protocols design, and urban network data analytics and management. He served on Technical Program Committees of INFOCOM 2015-2016, ICDCS 2014-2015, and SIGSPATIAL GIS 2015. He is the Cochair of SIMPLEX 2015.



Yingcai Wu is a tenure-track assistant professor at the State Key Lab of CAD & CG, Zhejiang University, Hangzhou, China. He has been selected by China's 1000-Talents Program for young scholars in 2016. His research interests lie broadly in visual analytics and visualization. He received his Ph.D. degree in Computer Science from The Hong Kong University of Science and Technology (HKUST), Hong Kong in 2009 and obtained his B.Eng. degree in Computer Science and Technology from South China University of Technology, Guangzhou, China in 2004. Prior to his current position, Yingcai Wu was a researcher in Microsoft Research from May 2012 to January 2015. He was a postdoctoral researcher at the Visualization research group in HKUST from January to May 2010, and at the Visualization and interface Design Innovation (VIDi) research group in the University of California, Davis from June 2010 to March 2012.

Zhiguo Gong received his PhD degree from the Department of Computer Science, Institute of Mathematics, Chinese Academy of Science. He is currently an Associate Professor in the Department of Computer and Information Science, University of Macau, Macau, China. His research interests include databases, web information retrieval, and web mining.

Yu Zheng is a senior research manager in Urban Computing Group at Microsoft Research, passionate about using big data to tackle urban challenges. His research interests include big data analytics, spatio-temporal data mining, machine learning, and artificial intelligence. Zheng currently serves as the Editor-in-Chief of ACM Transactions on Intelligent Systems and Technology and a member of Editorial Advisory Board of IEEE Spectrum. He is also an Editorial Board Member of GeoInformatica and IEEE Transactions on Big Data, and the founding Secretary of SIGKDD China Chapter. He has served as chair on over 10 prestigious international conferences, e.g. as the program co-chair of ICDE 2014 (Industrial Track) and CIKM 2017 (Industrial Track), and as senior PCs or PCs on KDD, IJCAI, AAAI, UbiComp, SDM, ICDM, and ACM SIGSPATIAL. Zheng is an ACM Distinguished Scientist, a Distinguished Member and Distinguished Speaker of China Computer Federation (CCF).