Interactive Bike Lane Planning using Sharing Bikes' Trajectories

Tianfu He, Jie Bao, Sijie Ruan, Ruiyuan Li, Yanhua Li, Hui He, Yu Zheng

Abstract—Cycling as a green transportation mode has been promoted by many governments all over the world. As a result, constructing effective bike lanes has become a crucial task to promote the cycling life style, as well-planned bike lanes can reduce traffic congestions and safety risks. Unfortunately, existing trajectory mining approaches for bike lane planning do not consider one or more key realistic government constraints: 1) budget limitations, 2) construction convenience, and 3) bike lane utilization. In this paper, we propose a data-driven approach to develop bike lane construction plans based on the large-scale real world bike trajectory data collected from Mobike, a station-less bike sharing system. We enforce these constraints to formulate our problem and introduce a flexible objective function to tune the benefit between coverage of users and the length of their trajectories. We prove the NP-hardness of the problem and propose greedy-based heuristics to address it. To improve the efficiency of the bike lane planning system for the urban planner, we propose a novel trajectory indexing structure and deploy the system based on a parallel computing framework (Storm) to improve the system's efficiency. Finally, extensive experiments and case studies are provided to demonstrate the system efficiency and effectiveness.

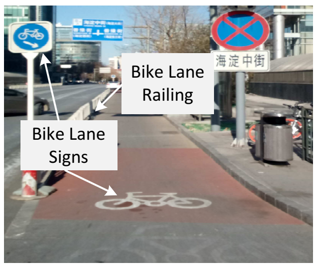
Index Terms—Data Mining, Distributed Computing, Urban Computing.

1 Introduction

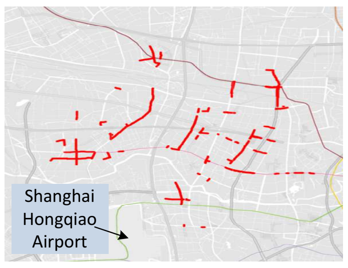
Cycling as a commonly used urban transit mode for daily commute has been promoted by multiple governments all over the world [1], [2] for several reasons: 1) it is an affordable and environment-friendly transportation mode for users; 2) it reduces road traffic congestions; and 3) it is a healthy lifestyle [3]. As a result, building effective bike lanes, demonstrated in Figure 1a, becomes a vital task for governments to promote the cycling lifestyle. Well planned and implemented bike lanes not only make cycling easier, but also reduce the safety risks for both cyclists and drivers of motor vehicles [4].

Traditional approaches to plan bike lanes in a city rely mainly on empirical experience and surveys [5], [6], [7]. With widespread availability of GPS embedded devices, more data-driven approaches on planning bike lanes have emerged, e.g., [8], [9], [10]. However, existing works [8], [9], [10] merely focus on summarizing commonalities of bike trajectory data while ignoring the realistic constraints and requirements faced by the government:

- **Budget Limitations.** There are costs to realizing a bike lane on a road segment [11], [12], which may include: 1) the space for creating bike lanes; and 2) the price of building bike lane railing, and painting signs (demonstrated in Figure 1(a)). Unfortunately, governments often have limited budgets.



(a) An Example of Bike Lane



(b) An Example of Inconvenient Plan

Fig. 1. Motivating Examples.

- **Construction Convenience.** To implement the bike lanes, construction teams need to be dispatched to construction zones, with the number of teams required also being a hard constraint. For the ease of management, the government would like to avoid spreading teams out to construction zones in far reaching locations (e.g., red lines in Figure 1(b) highlight the top-100 segments with the most bike trajectories), and prefer to have them clustered, as a limited number of connected components in the road network.
- **Bike Lane Utilization.** As a public service, from the government's point of view, the objective of building bike lanes is to increase the usability for more bikers and cover more possible routes.

To incorporate these real world constraints, in this paper, we propose a data-driven approach for planning the bike lanes based on the massive trajectories collected from Mobike [1]. Mobike is a fully station-less bike-sharing system currently deployed in many large cities in China, where it serves over 25 million daily bike requests [13]. Compared to

- *Tianfu He and Hui He are with Harbin Institute of Technology. Hui He is the corresponding author. E-mail: Tianfu.D.He@outlook.com and hehui@hit.edu.cn*
- *Jie Bao, Yu Zheng are with JD Intelligent Cities Research and JD Intelligent Cities Business Unit. E-mail: baojie@jd.com and msyuzheng@outlook.com*
- *Sijie Ruan and Ruiyuan Li are with Xidian University and JD Intelligent Cities Research. E-mail: {ruansijie, ruiyuan.li}@jd.com*
- *Yanhua Li is with Worcester Polytechnic Institute. Email: yli15@wpi.edu*

1. https://en.wikipedia.org/wiki/Mobike

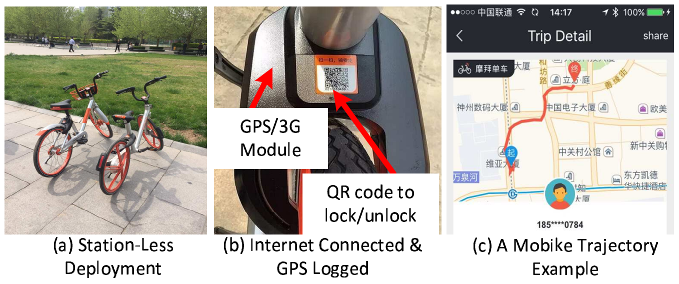

GPS/3G Module
QR code to lock/unlock
Trip Detail
share
(a) Station-Less Deployment
(b) Internet Connected & GPS Logged
(c) A Mobike Trajectory Example

Fig. 2. The Mobike Example.

the traditional station-based bike sharing systems, trajectories generated by Mobike have two distinctive advantages for the bike lane planning problem:

- **Realistic Travel Demands.** Unlike many existing station-based bike sharing systems, which require the users to pick up and drop off bikes from designated stations, Mobike offers a more flexible system, where the users can pick up and drop off their bikes at arbitrary locations (Figure 2(a)). Even if there's still gap between user's demand and supply position especially when picking up (E.g. a user starts inside an residential area but finds a bike at the gate), the trajectories still capture the major part of realistic travel demands.
- **Rich Travel Information.** A 3G communication component and a GPS module are embedded on the lock system in Mobike (demonstrated in Figure 2(b)), which enables the users to find bikes with their phones. It also keeps the track of the exact route traversed by the users (Figure 2(c)), while traditional station-based bike sharing systems can only provide the check-in/out information.

In this paper, we design, implement and deploy a data-driven bike lane planning system on Microsoft Azure, which not only leverages the massive bike trajectories generated by thousands of Mobike users, but also fulfills the constraints requested by the government. The proposed system contains two main components: 1) *Pre-Processing*, which pre-processes the trajectories from Mobike users and maps them on the road network; and 2) *Bike Lane Planning*, which takes the user's input (i.e., requirements from the government) and provides bike lane suggestions. Going beyond the early version of this work [14], we propose a novel trajectory index structure (i.e., *score index*), and deploy the bike path planning algorithm on the parallel computing platform, i.e., Storm, which significantly improves the efficiency and response time. Therefore, the urban planner can interact with the system more effectively. The main contributions are summarized as follows:

• We consider the bike lane planning problem with various construction constraints, and propose a flexible tuning parameter to characterize the trade-off between the number of covered users and the length of the continuously covered bike trips. The problem proves to be NP-hard.

• We propose a *greedy network expansion* algorithm, which provides an approximate solution to the bike lane planning problem. To achieve a better effectiveness, we also propose two different approaches to initialize the algorithm, which work well for low and high budget scenarios, respectively.

• To improve the system efficiency, we employ a novel *trajectory score index* to overcome the system efficiency bottleneck, i.e. the beneficial score computing step in *greedy network expansion*. Moreover, we design and implement the system on the parallel computing platform, i.e., Storm, to fit large scale trajectory data that cannot loaded into single machine and further improve the system response time.

• We evaluate the proposed algorithms extensively over one month Mobike trajectory data (i.e., from 9/1/2016 - 9/30/2016) from the City of Shanghai. We also provide an extensive data analysis and discover many useful insights. Moreover, on-field case studies are conducted to evaluate the effectiveness of our bike lane recommendations.

• An online system with the real dataset is deployed and available in public [15]. Finally, we collect the feedback from the government officials, from which our system received very positive reviews.

2 Overview

In this section, we model and define the bike lane planning problem, and outline our solution framework.

2.1 Problem Definition

Given a road network graph $G = (V, E)$ (where the vertex set V represents intersections and the edge set E represents all relevant road segments, our data-driven bike lane planning problem aims to discover a subset of edges $E' \subseteq E$, that follows three criteria: (i) construction budget constraint, (ii) connectivity constraint, (iii) maximum usage benefit.

Construction budget constraint. There is a monetary cost $e_i.c$ associated with each road segment e_i, to convert a road segment into a bike lane (e.g., building the railings and clearing the space). On the other hand, the government has an overall budget constraint B to building bike lanes, and the total cost of the construction cannot exceed the overall construction budget B, as highlighted in eq.(1) below.

$$\sum_{e_i \in E'} e_i.c \le B. \tag{1}$$

Connectivity constraint. As has been outlined in the introduction section, for the construction and management convenience, the government prefers to deploy bike lanes with up to k connected components (i.e., to be assigned to k construction teams). The following inequality eq.(2) reflects such a constraint:

$$C(E') \le k, \tag{2}$$

where $C(E')$ denotes the operator that counts the number of connected components from an edge set E'.

Maximum usage benefit. The goal here is to maximize overall usage of deployed bike lanes, which should 1) facilitate as many users as possible, and 2) cover more continuous road segments along their trip routes. Note that continuous road coverage in bike lane planning is crucial, as it increases the users' quality of experience (QoE). For example, a bike user travels on a path (i.e., $e_1 \rightarrow e_2 \rightarrow e_3$), shown as blue

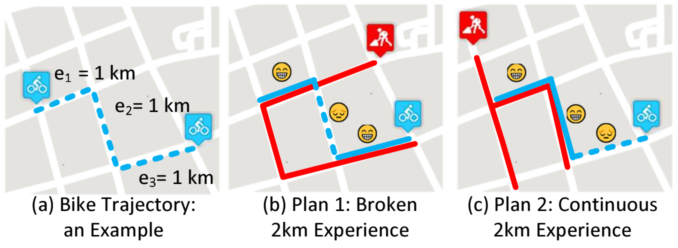

e1 = 1 km
e2= 1 km
e3= 1 km
(a) Bike Trajectory: an Example
(b) Plan 1: Broken 2km Experience
(c) Plan 2: Continuous 2km Experience

Fig. 3. Motivation of Trajectory Score Function.

dotted lines in Figure 3(a)). Though both of the bike lane plans(Figure 3(b) & (c)) covers the same length(2km) of his trip, Plan 2 is preferred by users as it provides continuous 2km's experience, while the trajectory coverage of *Plan 1* is broken into two disconnected segments. The QoE would get even worse when there were more disconnected segments.

Unfortunately, these two objectives (i.e., serving more users vs. covering longer and continuous trips) usually conflict with each other, as user trips usually have different destinations. Hence, we propose a flexible score function for decision makers to adjust their preference between the two objectives for a trajectory tr_i:

$$\mathcal{S}(s_j.l) = \alpha^{\frac{s_j.\ell}{min(e.\ell)}} \times \frac{s_j.\ell}{min(e.\ell)} \tag{3}$$

$$tr_i.g = \sum_{s_j \in S_i} \mathcal{S}(s_j.l), \alpha \geq 1. \tag{4}$$

where $tr_i.g$ is the beneficial score for trajectory tr_i, S_i is the set of continuous road segments that overlap with trajectory tr_i in the path plan E', s_j is one continuous road segments in set S_i. $\mathcal{S}$ is the function to calculate the score of each continuous road segment s_j, where $\frac{s_j.\ell}{min(e.\ell)}$ normalizes the length of the continuous road segment $s_j \in S_i$ (where $min(e.\ell)$ is the minimum length of the road segment in the network), with the guarantee that its value is no less than 1, and α is the tuning parameter to set the preference on the number of covered users versus the length of continuous coverage. The reason for designing a score function using the exponential function of the normalized length is that when $\alpha > 1$, the continuous segment gets a higher score. Otherwise, without the exponential function $\alpha^{\frac{s_j.\ell}{min(e.\ell)}}$, *Lane Plan 1* and *Lane Plan 2* will have the same score. A smaller α indicates that more preference is given to the amount of user coverage (e.g., α = 1 means that we do not care about the continuous length coverage, and two path plans in Figure 3 have the same beneficial score), while a larger α means that the longer continuous length coverage of the user trips is preferred.

Then, since the motivation is to improve the overall experience of all the users' trips, we calculate the overall beneficial score of a bike lane plan $E'.g$ by aggregating the scores of all the trajectories Tr that overlap with road segment set E':

$$E'.g = \sum_{tr_i \in Tr \& tr_i \cap E' \neq \emptyset} tr_i.g. \tag{5}$$

We formalize our bike lane planning problem as follows.

Problem definition. *Given a set of trajectories Tr, a road network $G = (V, E)$ with a cost value $e_i.c$ on each edge e_i, a tuning parameter α, a value k, and a total construction budget B, we want to find a set of edges $E' \subseteq E$, which maximizes the total beneficial score g, and fulfills two constraints: 1) the total budget is no more than B; and 2) the number of connected components in E' is less than k.* Formally, it is represented as an integer programming problem:

$$\text{max:} \quad E'.g, \qquad \text{s.t.:} \sum_{e_i \in E'} e_i.c \leq B, \qquad C(E') \leq k. \tag{6}$$

Such a problem of finding k budget constrained connected components with maximum beneficial score is NP-hard as proven in Lemma 1 below.

Lemma 1 (NP-Hardness). Finding k budget constrained connected components in a graph with the maximal beneficial score is NP-hard.

Proof: We reduce our problem from the 0-1 knapsack problem. Recall that in a 0-1 Knapsack problem, we are given a set of n items, each with a weight w_i and a value v_i, along with a maximum capacity W. Our goal is to find an $S \subseteq [n]$ to maximize $\sum_{i \in S} v_i$ subject to the constraint $\sum_{i \in S} w_i \leq W$.

Given a 0-1 Knapsack problem, we construct a k-budget constrained connected component problem instance as follows. First, an item corresponds to an edge (road segment). The weight w_i (value v_i) of an item corresponds to the construction cost (beneficial score) of the edge it associates with. Next, we set $k = |E|$ so that the component constraint becomes trivial; we set $\alpha = 1$ so that we do not care about the continuous length coverage.

One can see that E' is feasible in the k-budget constrained connected component problem if and only if the "items" in E' is feasible in the knapsack problem. Consequently, the optimal value for the knapsack problem coincides with that of the k-budget constrained connected component problem, which completes our proof. □

Given it is an NP-hard problem, we develop a greedy-algorithm based heuristic to tackle the issue.

2.2 System Framework

Figure 4 gives an overview of our system, which consists of two main components:

Pre-Processing. This component takes the bike trajectories and the road network and performs three main tasks: 1) *Trajectory Data Parsing*, which removes the outlier GPS

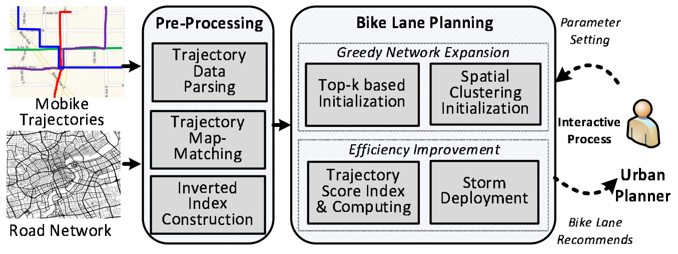


Fig. 4. An Overview of System.

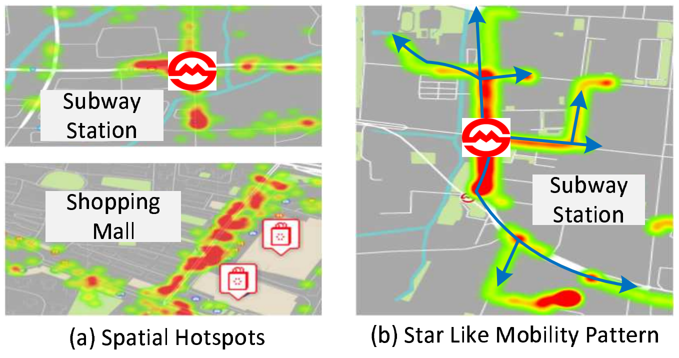


Fig. 5. Spatial Insights of Mobike Data.

points; 2) *Trajectory Map-Matching*, which projects the bike trajectories onto the corresponding road segment; and 3) *Inverted Index Construction*, which builds an index to speed up the lookup process of retrieving trajectories based on road segment IDs (detailed in Section 3).

Bike Lane Planning. This component takes the user's parameters, e.g., the total budget, number of connected components, and the α value, and outputs the bike lane recommendation results. If the user is satisfied by the results, parameters can be tuned to get a new set of recommendations. This component contains two main modules: 1) *Greedy Network Expansion*, where we propose two different approaches to initialize the network expansion (detailed in Section 4); and 2) *Efficiency Improvement*, where we propose a novel trajectory score index to speed up the planning algorithm and deploy the system on the parallel computing platform, i.e., Storm (detailed in Section 5).

3 Pre-Processing

Pre-processing takes the road network and the trajectories as input, and performs the following three tasks:

Trajectory Parsing. This step cleans the raw trajectories from Mobike by filtering the noisy GPS points using a heuristic-based outlier detection method [16].

Trajectory Map-Matching. In this step, the system maps each GPS point onto the corresponding road segment. We employ a revised version of an interactive-voting based map matching algorithm [17], where the speed constraint of the road segments is not used.

Inverted Index Construction. In this step, the system builds the *inverted index* for each road segment, recording the trajectory IDs passing it. In this way, we can speed up the road segment-based trajectory look-up. The index construction process is done in parallel on Microsoft Azure [18].

4 Greedy Network Expansion

4.1 Greedy Network Expansion Framework

Main Idea. The intuition of the greedy network expansion algorithm is to expand a set of k starting road segments in the network. This is inspired by the two key insights discovered in the dataset, namely *spatial hot spots* and *star-like mobility patterns*: *Spatial hot spots.* Figure 5(a) shows the two hot spots with the highest number of trip starting locations, where the upper side reflects a subway station, and the lower side illustrates a popular shopping mall. The intuition behind the observation is straightforward: although the mall is very popular, it is not close to any subway stations, which makes cycling the best option; similarly for the terminal station, the fastest & most economic option to get home from there is cycling.

Star-like mobility patterns. We further investigate travel directions around spatial hot spots, and we discover that the bike trips go to different destinations from the same starting location, just like multiple edges with one shared end, namely, a star-like mobility pattern, as demonstrated by the arrows in Figure 5(b). Taking these observations into considerations, our greedy-based bike lane planning starts from the hotspots and expands greedily to generate the star patterns. The algorithm extends the incremental network expansion algorithm in road network, e.g., [19], [20]. The algorithm has three phases:

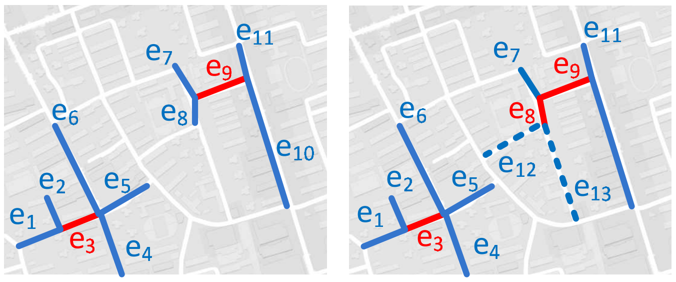
Algorithm 1 Greedy Network Expansion Algorithm

```
    Input: Road Network $G = (V, E)$, Inverted index $I$, Trajectory Dataset $Tr$, Total budget $B$, tuning parameter $\alpha$, and a value $k$.
    Output: Result road segment set $E'$.
    //Stage 1: Initialization
1:  Road Segment Set $E' \leftarrow k$ starting road segments
2:  Candidate set $\mathcal{C} \leftarrow$ adjacent road segments of $E'$
3:  Remaining Budget $B \leftarrow B - \sum_{e_i \in E'} e_i.c$
    //Stage 2: Network Expansion
4:  while Budget $B > 0$ do
5:      $MaxGain \leftarrow 0$; $e_{next} \leftarrow \emptyset$
6:      for $e_i \in$ Candidate set $\mathcal{C}$ do
7:          if $e_i.c < B$ then
8:              Retrieve trajectories $Tr'$ from $I$ based on $E' \cup e_i$
9:              Get beneficial score difference per cost $\Delta g = \frac{g'-g}{e_i.c}$
10:             if $MaxGain < \Delta g$ then
11:                 $MaxGain = \Delta g$; $e_{next} \leftarrow e_i$
12:     $E' \leftarrow E' \cup e_{next}$; $B \leftarrow B - e_{next}.c$
13:     $\mathcal{C} \leftarrow \mathcal{C} \cup$ none-selected adjacent edges of $e_{next}$
    //Stage 3: Termination
14: return $E'$
```

- **Stage 1: Initialization.** The algorithm starts by selecting k starting road segments. In this way, we can guarantee that the final road segment recommendation produced by the algorithm fulfills the connectivity constraint, i.e., does not generate more than k connected components.
- **Stage 2: Network Expansion.** In this stage, the algorithm runs iteratively. In each iteration, the algorithm selects the best road segment (i.e., with the highest the beneficial score gain per cost) to the result set E' and adds its none-selected adjacent segments to the candidate set.
- **Stage 3: Termination.** The algorithm terminates when budget limit B is met, and then returns the resulting road segment set E' as the recommended bike lane plan.

Edge ID	e_1	e_2	e_4	e_5	e_6	e_7	e_8	e_{10}	e_{11}
Δ Gain	1	4	5	7	8	4	5	9	5
Cost	2	2	2	2	5	2	1	5	2

(a) A Candidate Set Example



(b) An Initialization Example (c) An Network Expansion Iteration

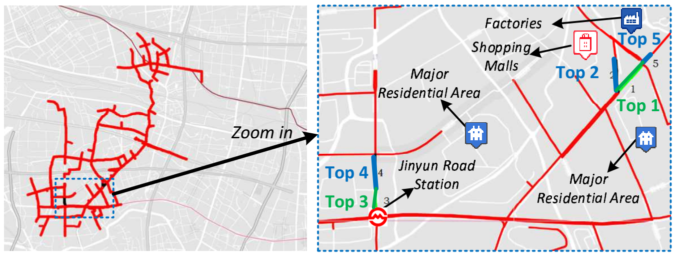
Fig. 6. Greedy Network Expansion Example.

Algorithm Design. Algorithm 1 gives the pseudo-code of our greedy network expansion algorithm. In the *initialization stage*, the algorithm first selects k starting road segments into the resulting set E', puts all adjacent road segments of the start segments into candidate set $\mathcal{C}$, and updates the budget value by subtracting the total cost of the starting road segments (Line 1-3).

In each iteration of the *network expansion stage* (Line 5-13), the algorithm checks each road segment e_i in the candidate set $\mathcal{C}$. If the cost of the road segment is smaller than the remaining budget, the algorithm retrieves all the trajectories Tr that has been covered by the road segment e_i and the result road segment set E' (Line 8). After all the covered trajectories are retrieved, we calculate an updated beneficial score g' based on Equation 5. Then, we calculate the corresponding beneficial score gain per cost (Line 9). During the process, we keep track of the road segment e_{next}, which has the maximum beneficial score gain per cost in the iteration. e_{next} is inserted in to the resulting road segment set E', the remaining budget is updated by subtracting the cost of the selected road segment $e_{next}.c$. When there're ties on Δg between candidates, we greedily select the one with the maximum total score that meets the remaining budget constraint. Road segment e_{next} is removed from candidate set $\mathcal{C}$, and all of its none-selected adjacent segments of e_{next} are inserted in the candidate set $\mathcal{C}$ for further iterations (Line 10- 13).

Finally, when all the budget is used up, the algorithm terminates, and the road segment set E' is returned as the recommended plan. Note that although there are many works (E.g. [11], [12]) that discuss about budget estimation of lane planning, our work simply chooses the road length as budget. Nevertheless, in our solution, the budget can be defined as any scheme, such as price, length and etc.

Example. Figure 6 gives an example of the greedy network expansion algorithm. In the *initialization stage*, two starting road segments are selected (marked in red), and all of their adjacent segments are inserted in the candidate set (marked in blue). During the *network expansion stage*, in the iteration, we calculate the beneficial score gain difference for each segment in the candidate set (illustrated in Figure 6(a)), based on Equation 5. After that, we divide the beneficial score difference by the cost of each segment and select the highest one to expand the network, which is e_8 in our example. Then, the adjacent segments of e_8 are added as the new candidates (i.e., e_{12} and e_{13} in Figure 6(c)). The algorithm terminates when the budget is used up.



(a) Result of Top-k Initialization (b) Top-k Start Segments

Fig. 7. Top-k Initialization Example.

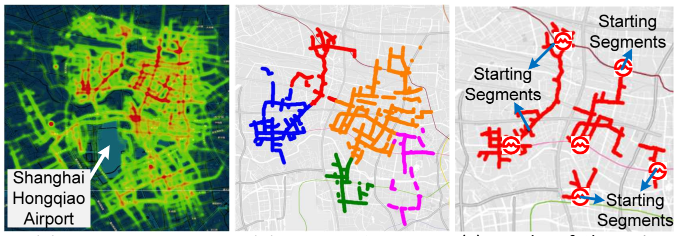
Analysis. As demonstrated in the example, it is clear that the performance of the final results E' is highly determined by the selection of the starting road segments. As a consequence, finding an effective method to perform initialization becomes a vital task in our greedy network expansion algorithm.

4.2 Top-k based Initialization

Main Idea. The most straightforward method is *Top-k Initialization*, which essentially selects the highest ranked k segments based on the beneficial score per cost (i.e., $\frac{e_i.g}{e_i.c}$), as the starting segments for network expansion. The intuition behind this approach is that these road segments usually represent the spatial hot spots, which should always be included in the final result.

Example. Figure 7(a) gives an example result of greedy network expansion with top-k based initialization, with $k = 5$. The recommended bike lanes are marked in red in the figure, which form one large set of connected components. The reason the result contains only one connected component, rather than five (i.e., k value) is that the top-5 highest ranked segments are connected with each other. Figure 7(b) is the detailed view of the boxed area in Figure 7(a), where the selected five starting road segments are marked in green and blue, which form two groups (i.e., {Top 1, Top 2, Top 5} and {Top 3, Top 4}). The first group contains the road segments between a major residential area and nearby shopping malls/factories, while the second group contains the road segments near the terminal station for subway Line 13. The reason why the top ranked segments are usually close to each other, is that a large amount of trajectories may share a lot of road segments, as they traverse from or to the same location (e.g., a subway station or a shopping mall).

Analysis. The top-k based initialization approach guarantees that the algorithm will never miss any segment with the highest beneficial score per cost. However, as most of the top-k ranked segments are very close to each other, it can only expand with a much lower number of connected components in the network, which limits the search space in the candidate set and may miss some important areas, especially when the budget B is large.



(a) Trajectory Data Distribution (b) Results of Spatial Clustering (c) Results of Clustering based Expansion

Fig. 8. Spatial Clustering based Initialization.

4.3 Spatial Clustering-based Initialization

In order to include more spatially diversified starting locations in the initialization stage and be more effective when the budget is larger, we take advantage of spatial clustering techniques to select the starting road segments.

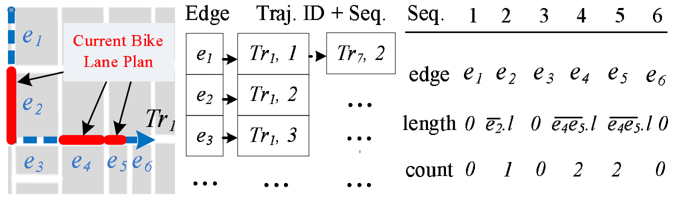
Main Idea. The intuition behind the *spatial clustering-based initialization* is from the observation of the trajectory heat map (i.e., Figure 8(a)), which visually has some rough clusters over the space. In this way, we can avoid the drawbacks of the top-k based initialization, which has the starting segments connected to each other and limits the search space. This method has two main steps:

- **Candidate Selection.** In this step, we select a subset of road segments with high ranks (e.g., top 1% ranked segments in our implementation based on the score per cost), as the candidates for clustering. The reason for selecting a subset of road segments for clustering is to remove the road segments that will never be in the final result and reduce computational cost.
- **Spatial Clustering.** In this step, the candidate road segments are clustered based on an agglomeration hierarchical clustering method, e.g., [21]. After that, the highest ranked road segment in each cluster is selected as a starting segment.

The hierarchical-based clustering method is employed in our system, as it does not need to tune the clustering parameters (e.g., in DBSCAN [22]) and it always generates stable results (unlike it is in K Means [23]). Thus, it is more intuitive for government users that are not familiar with clustering technical details.

Example. Figure 8(b) and (c) gives an example of the execution results of spatial clustering-based initialization, where $k = 5$. In the first step, we compute the clusters generated by our algorithm, i.e., Figure 8(b). After that, the highest ranked road segments are selected as the starting segments, which are the black segments in Figure 8(c). It is interesting to note that four of the starting segments are at subway stations. The recommended paths actually cover the neighborhood of six subway stations, as illustrated in the figure.

Analysis. Compared to the results generated by the top-k initialization method, spatial clustering based initialization clearly has better diversity and coverage. The main reason is that after the spatial clustering step, the starting segments are no longer connected with each other. As we will show in our experiments, with more budgets, the spatial clustering-based initialization method is more effective.



(a) Trajectory Tr_1 (b) Traj. Inverted Index (c) Traj. Score Index of Tr_1

Fig. 9. Index Structures in Efficient Traj. Score Computing.

5 Efficiency Improvement

In each network expansion iteration, the score computation (i.e., the beneficial score difference per cost for each road segment) is dependent to the current resulting set E' when $\alpha > 1$, therefore, the processing of a large number of trajectories is unavoidable. Moreover, it may take a large number of expansion iterations, when the budget is large. As a result, the response time of *Greedy Network Expansion* algorithm can be prohibitively long, e.g., several minutes for a typical bike lane planning request over one-month's trajectory data, which incurs the inconvenience for the urban planners to interact with the system (e.g., setting different k and budget values).

To improve the system efficiency and clear the burdens for urban planners, we develop two techniques: 1) *Efficient Trajectory Score Computing*, which significantly improves the efficiency on the score computing step by introducing a novel *trajectory score index*; and 2) *Parallel System Deployment*, which distributes the computational overhead to multiple computing nodes to reduce the system response time.

5.1 Efficient Trajectory Score Computing

A naive approach to execute the score computing step contains the following three steps: 1) retrieving all the trajectories based on the candidate road segment; 2) scanning each retrieved trajectory and extracting all the connected road segments; and 3) calculating the score for each connected road segment based on Equation 3 and adding it to the total score.

Clearly, the naive approach for score computing has two drawbacks: 1) the entire trajectory data needs to be fully scanned to identify the connected road segments; and 2) the computed score for each connected road segments is not reused, even if the connected road segment is not changed with the newly added candidate segment. As a result, the first drawback incurs significant redundant I/O access, and the second drawback introduces much redundant computing. To this end, we propose a novel trajectory score index and an efficient score computing algorithm to overcome these drawbacks and improve the system efficiency.

5.1.1 *Data Structure*

We first introduce two data structures used in the algorithm:

Inverted Trajectory Index. The index is a hash map, where the key is the edge ID and the value contains a list of trajectory information (including trajectory ID and the sequential ID of the edge in the trajectory).

Figure 9(b) gives an example of the trajectory inverted index, where each edge is linked to a list of trajectory information, including: 1) Tr_i, which is the trajectory ID that passes the corresponding edge, and 2) *sequential number*, which is the position of the edge in Tr_i.

Trajectory Score Index. Each trajectory Tr_i maintains its own trajectory score index TS_i during the process. The trajectory score index is a hash map, where the key is the sequential ID of the edge passed by the trajectory, and the value contains three attributes: 1) *edge ID*; 2) *length*, which records the length of the continuous road segments with the current bike lane plan that involves this edge; and 3) *count*, which is the number of edges of the continuous road segments with the current bike lane plan that involves this edge. Note that, since the *length* and *count* are related to the current plan, the score index is updated as the plan expands.

Figure 9(c) gives an example of the trajectory score index TS_1 of Tr_1 with the current bike lanes as shown in Figure 9(a). The *length* and *count* for the edges, $TS_1.seq[1]$, $TS_1.seq[3]$, and $TS_1.seq[4]$, are 0, as they are not covered by the current bike lane plan (shown in red). On the other hand, the *length* and *count* of $TS_1.seq[2]$ are $\overline{e_2}.l$ and 1, respectively, as e_2 is covered solely by the current bike lane plan. Similarly, the *length* and *count* of $TS_1.seq[4]$ and $TS_1.seq[5]$ are $\overline{e_4e_5}.l$ and 2, as e_4 and e_5 are covered by a continuous bike lane plan.

The modified *inverted trajectory index* is used to avoid the redundant full scan of the trajectory. With the trajectory sequential number in the index, we can locate the newly added candidate road segments in the trajectory. The *trajectory score index* is used to speed up the score computation, by keep track of the partial results of the score computing, i.e., the total length of the continuous segments in each iteration of the network expansion. The length of the continuous segment is stored rather than the actual score in the index, as the new score cannot be generated directly by the previous scores directly, due to Equation 3.

5.1.2 Efficient Trajectory Score Computing

Main Idea. The intuition of the efficient score computation algorithm focuses on computing only the incremental score, i.e. Δg. In other words, for each candidate road segment, we use the *invert trajectory index* to identify the trajectories that are covered by the road segments and their corresponding positions. After that, by using the *trajectory score index*, we identify the length differences caused by the candidate segment and returns the incremental score. There are three possible cases, when the beneficial score of a trajectory is affected by the candidate road segments:

- **Case 1: None Connection.** In this case, the candidate segment does not extend any of the existing continuous road segments from its previous or next sequential neighbor in the trajectory, as shown in Figure 10(a). As a result, the incremental score is calculated directly based on the length of the edge $\overline{e_4}$ using Equation 4.
- **Case 2: Single Connection.** In this case, the candidate segment extends a set of existing continuous road segments from either its previous or next sequential neighbor of the trajectory. Figure 10(b) shows an example, where the candidate segment extends a continuous road segment plan based on its previous sequential neighbor. In this example, the incremental score is calculated by subtracting the score of $\overline{e_2}$ from $\overline{e_2e_3}$.
- **Case 3: Double Connection.** In this case, the candidate segment essentially connects two sets of continuous road segments, as shown in Figure 10(c). In this example, the incremental score is calculated by subtracting the score of both $\overline{e_2}$ and $\overline{e_4}$, from $\overline{e_2e_3e_4}$.

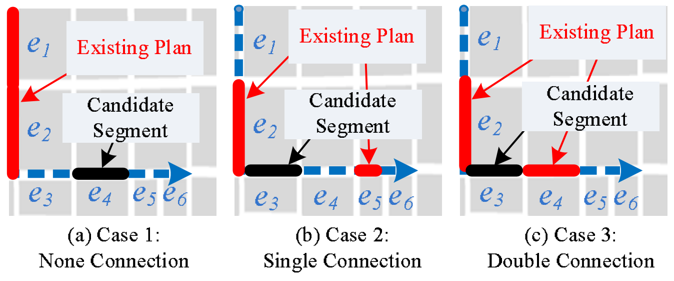


Fig. 10. Three Cases of Candidate Segment Addition.

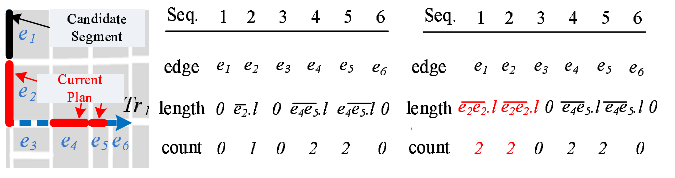
Algorithm 2 Efficient Trajectory Score Computing

```
Input: Trajectory Dataset $Tr$, Candidate Segment $e_c$, Inverted index $I$, Trajectory score index $TS$.
Output: The incremental score $\Delta g$.
1:  $\Delta g \leftarrow 0$
2:  Retrieve the list $L$ with {trajectory & sequence} from $I[e_c]$
3:  for each {trajectory $i$, sequence $j$} $\in L$ do
4:      switch Check $TS_i.seq[j-1]$ & $TS_i.seq[j+1]$ do
5:          case None Connection
6:              $\Delta g{+}{=}\mathcal{S}(e_c.l)$
7:          case Single Connection
8:              if $S_i.seq[j-1].l != 0$ then
9:                  $\Delta g{+}{=}\mathcal{S}(e_c.l + TS_i.seq[j-1].l) - \mathcal{S}(TS_i.seq[j-1].l)$
10:             else
11:                 $\Delta g{+}{=}\mathcal{S}(e_c.l + TS_i.seq[j+1].l) - \mathcal{S}(TS_i.seq[j+1].l)$
12:         case Double Connection
13:             $g' = \mathcal{S}(TS_i.seq[j+1].l) + \mathcal{S}(TS_i.seq[j+1].l)$
14:             $\Delta g{+}{=}\mathcal{S}(e_c.l + TS_i.seq[j-1].l + TS_i.seq[j+1].l) - g'$
15: return $\Delta g$
```

As a result, we are able to reduce the computational complexity of the score computing from $O(N)$ (i.e., fully scan of a trajectory) to $O(1)$ (i.e., just looking at the statistics of the neighbors). Also, we do not need to calculate the part of the existing continuous road segments if they are not affected by the candidate segment.

When a candidate segment is selected by the algorithm, i.e., getting the most beneficial score gain per cost in the *greedy network expansion* algorithm. The *trajectory score index* is updated accordingly: 1) we use *count* values of the neighbor edge to identify the update range; and 2) for all the edges in the update range, all their *counts* and *lengths* are updated for the further expansion.

Algorithm. Algorithm 2 provides the pseudo code of the proposed efficient score computing method. The algorithm first initializes the beneficial score difference Δg as 0, and then retrieves the list of trajectory ID and the corresponding sequential number pairs L from the *inverted trajectory index* I(Line 1- 2).

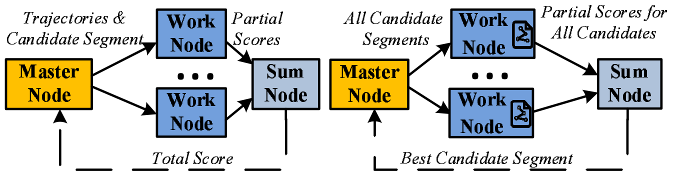


Seq.	1	2	3	4	5	6
edge	e_1	e_2	e_3	e_4	e_5	e_6
length	0	$\overline{e_2}.l$	0	$\overline{e_4e_5}.l$	$\overline{e_4e_5}.l$	0
count	0	1	0	2	2	0

Seq.	1	2	3	4	5	6
edge	e_1	e_2	e_3	e_4	e_5	e_6
length	$\overline{e_2e_2}.l$	$\overline{e_2e_2}.l$	0	$\overline{e_4e_5}.l$	$\overline{e_4e_5}.l$	0
count	2	2	0	2	2	0

(a) Trajectory Tr_1 (b) Initial Traj. Score Index of Tr_1 (c) Updated Traj. Score Index of Tr_1

Fig. 11. Example of Efficient Trajectory Score Computing & Update.



(a) A Naïve Parallel Computing Design (b) Proposed Parallel Computing Design

Fig. 12. Parallel System Design.

For each trajectory ID and sequence number pair $\{i, j\}$ in the list L, we calculate the incremental score and added it to Δg. We get the *trajectory score index* TS_i based on the trajectory ID i, and check the neighbors of the sequential ID j, i.e., $TS_i.seq[j-1]$ & $TS_i.seq[j+1]$, to see if any of them is covered by the existing bike lane plan (Line 4- 14). There are three cases: 1) *none connection*, where we directly add the score introduced by the candidate segment e_c based on Equation 3 to Δg; 2) *single connection*, depending on the candidate edge extends the existing bike plan from its previous edge ($TS_i.seq[j-1]$) or the next edge ($TS_i.seq[j-1]$), we first add the overall score ($\mathcal{S}(TS_i.seq[j \pm 1].l + e_c.l)$) and then subtract the score of the previous continuous segment($\mathcal{S}(TS_i.seq[j \pm 1].l)$; and 3) *double connection*, where we first calculate the old beneficial score g', as $\mathcal{S}(TS_i.seq[j-1].l) + \mathcal{S}(TS_i.seq[j+1].l)$. After that, we subtract the old beneficial score from the new total score, as $\mathcal{S}(e_c.l + TS_i.seq[j-1].l + TS_i.seq[j+1].l) - g'$.

Finally, when all the the trajectory ID and sequential number pairs are processed, the final incremental beneficial score Δg is returned (Line 15).

Example. Figure 11 gives an example of calculating the incremental score Δg and updating the corresponding *trajectory score index*. In Figure 11(a), the blue arrow indicates the trajectory Tr_1, the existing bike lanes are marked in red, and the candidate segment is marked in black. Figure 11(b) demonstrates the initial *trajectory score index* of trajectory Tr_1, where the *length* and *count* fields are filled based on the existing bike lanes.

When candidate segment e_1 is selected to calculate the incremental score, the algorithm looks at the its sequential neighbors and identifies that its next segment is covered by the existing bike lane. Then, the incremental score is calculated using the equation in *single connection* case. The incremental score is sent back to the greedy expansion algorithm to select the best candidate for the further iteration.

If a candidate segment is selected as the best one in the greedy expansion algorithm, e.g., in this case e_1, the *trajectory score index* is updated. Essentially, we identify the update range based on the *count* value of the neighbor edges, where in this example, we only need to update one neighbor edge e_2, as its *count* was one. And for the cases when *count* $>= 2$, since the algorithm only accesses the two ends of a continuous segment (Algorithm 2, Line 4), we only need to update *length* and *count* of the two ends without loss of correctness. In this way, the score index updating is also $O(1)$.

5.2 Parallel System Deployment

With the efficient score computing method, the *greedy network expansion* algorithm still needs to compute a huge number of scores to get the result, as: 1) the number of trajectories in a candidate segment is huge, as the algorithm is initialized in the urban centers; and 2) the number of expansion iterations is also enormous, especially when we have a large budget. In other words, to generate a bike lane plan, a large number of trajectory score computations needs to be processed. Fortunately, the score computation within the same expansion iteration is independent to each other. As a result, the intuition here is to distribute the computational and I/O overhead of the score computations to multiple computing nodes in a parallel computing platform to reduce the response time.

We adopt Apache Storm to implement the distributed *greedy network expansion* algorithm. The main reason to choose Storm over Spark is that after each expansion iteration, a lot of information in the *trajectory score index* needs to be updated. To realize this process in Spark, we need to create a new RDD (Resilient Distributed Dataset) in every expansion iteration, which leads to a large I/O overhead.

5.2.1 System Design

A naive parallel system design is shown as Figure 12, which directly computes the scores for each candidate segments in parallel (i.e., Line 7- 11 in Algorithm 1), with the following steps: 1) the master node checks one candidate segment by sending the affected trajectories to different worker nodes; 2) each worker node computes a partial beneficial score based on the sub-set of trajectories and passes it to the sum node; and 3) the sum node gets all the partial scores and calculates the total score and sends it back to the master node, until all the candidates are examined or the budget is used up.

However, there are two major drawbacks on the naive parallel system design, which significantly undermine the system efficiency:

1) **Huge Data Transfer.** In the naive design, trajectory data is transferred along with the candidate segment to worker nodes. Although high speed networks are employed in the parallel computing cluster, the overhead of data transferring is significant, when the trajectory data set is massive.
2) **Massive Cycling Iterations.** In each iteration, only the total score of one candidate segment is calculated and returned. As a result, the algorithm needs to perform massive cycling iterations(i.e., the number of network expansions times the number of candidate segments at each iteration) to get the final result. The response time can be prohibitively slow, as each iteration introduces the waiting time on 1) getting partial scores from all worker nodes; and 2) network delays between each two steps.

To this end, we propose an advanced parallel computing design, as demonstrated in Figure 12(b). Two improvements are employed: 1) instead of passing the trajectory data with the candidates, the trajectory data is pre-loaded and built into the *trajectory score index* in worker nodes. Moreover, to achieve a balanced workload, the trajectory data is divided temporally, e.g., on the daily basis; and 2) in each iteration, we pass all the candidate segments together to the worker nodes, rather than one by one. In this way, we significantly reduce the number of iterations to be the number of network expansions.

5.2.2 Storm Implementation

However, Storm is not designed to process the iterative tasks, as it is a parallel streaming system and does not support any circles in its topology. To this end, we modify the original Storm architecture and include a new message queue to store the complete confirmation of the score computations from different worker nodes (or bolts) and pass the overall score back to the master node (or Spout). Moreover, to avoid the massive data transfer during the process, we preload the partitioned trajectory data in worker nodes, before the user request.

In this sub-section, we first give an overview of the newly designed Storm framework. After that, we describe the two phases in the implementation: 1) *System Initialization*, and 2) *Service Providing*. Finally, we present some lessons learnt during the implementation.

System Overview. Figure 13 gives the Storm topology of our system, which consists of following four parts:

1) **ResultMQ.** It is a message queue service (i.e., Pivotal RabbitMQ) illustrated at the top of the figure. *ResultMQ* gets the completion confirmation from the *Report Bolt* and pass the best segment candidate in each iteration to *Command Spout*. In this way, we are able to implement the iterative greedy network expansion algorithm in Storm.
2) **Command Spout.** As demonstrated as the yellow box in the figure, *Command Spout* distributes the computational workloads to the different *Work Bolts*. Moreover, it also monitors the best candidate reported from the *report node* in each iteration.
3) **Work Bolt.** As illustrated as the blue boxes in the figure, *Worker Bolts* are used to load the partitioned trajectory dataset during the *system initialization*, and performs the distributed trajectory score computation during the *service providing*.
4) **Report Bolt.** It is shown as the light blue box in the figure. *Report Bolt* is used to summarize the score computation results of candidates from different *Worker Bolts* in each iteration and inserts the best candidate to the *ResultMQ*.

System Initialization. To achieve the load balancing among different *work bolts* during the trajectory score computing tasks, we partitioned the trajectory data based on their temporal information (with dates in our implementation), and organize them as different HDFS files. During the initialization, *command spout* sends out the file names randomly (i.e., *ShuffleGrouping*) to *worker bolts*, asking them to load the partitioned trajectory data from the corresponding files and build indexes (i.e., *inverted trajectory index* and *trajectory score index*) for the partial trajectories.

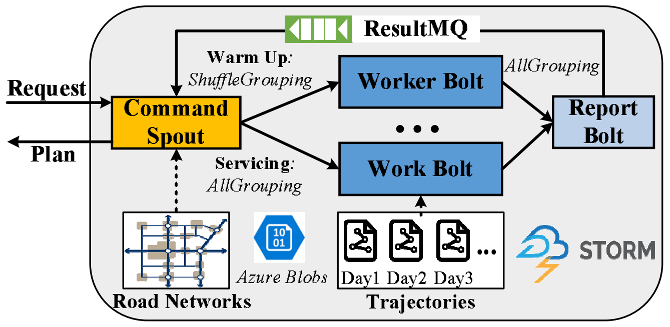


Fig. 13. Storm Topology.

Service Providing. When a bike lane planning request comes, the *CommandSpout* first selects the start segments based on spatial clustering or top-k methods. After that, the greedy expansion iterations starts: 1) all the candidate segments are broadcast (i.e., *AllGrouping*) to the *worker bolts*; 2) in each *work bolt*, the incremental beneficial scores for all the candidate segments are calculated based on the partial trajectory data and sent to the *report bolt*; 3) *report bolt* summarizes the scores and inserts the best candidate to the *ResultMQ*; and 4) *CommandSpout* updates the remaining budget. If the budget is used up, *CommandSpout* returns the set of best segments as the plan to the user. Otherwise, the iteration continues.

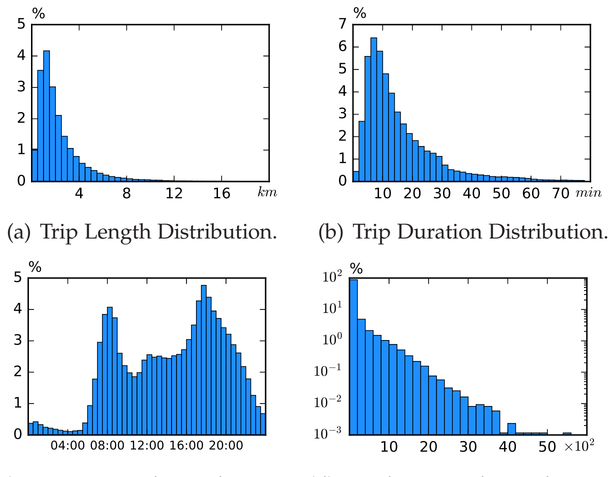
Lessons Learnt. In the actual implementation, our system still suffers from significant delays between different nodes in the Storm topology (e.g., between the *CommandSpout* and *Work Bolts*. We look into the communication techniques used in Storm and noticed that internal message queues like Netty or LMAX Disruptor are used. Moreover, to reduce the the I/O cost in the streaming event scenarios, these message queues are implemented with buffers. When a tuple is transferred, it is firstly stored in the buffer, rather than emitting it directly to the next bolt. The buffer is flushed out periodically, or the buffer is full.

However, in our design, in many cases, *spout* only sends out very few number of tuples, e.g., *CommandSpout* only sends a list of candidate segments to the *Work Bolts*. As a result, in default, the communication among nodes only happens after the flushing time out. The problem is amplified by the number of iterations executed during the network expansion algorithm.

As a result, we disable the buffer mechanism by setting the size of the buffers to 1, i.e. the tuple is sent out as soon as it is generated. This optimization speeds up the average tuple transferring time by an order of magnitude from $106ms$ to about $2ms$.

6 Experiments

In this section, we conduct extensive experiments to evaluate the effectiveness of our system. We first describe the dataset used in the paper. Then, we provide a detailed analysis on the mobility statistics of the Mobike trajectories.


(a) Trip Length Distribution. (b) Trip Duration Distribution.

(c) Trip Temporal Distribution. (d) Road Traversal Distribution.

Fig. 14. Mobike Trip Characteristics.

After that, we provide experiment results with different parameters. Based on the recommendation results, we present a set of real case studies to demonstrate the effectiveness of our system. We also present our online deployed system at the end of this section.

6.1 Datasets

Road Network. We use the road network of Shanghai, China from Bing Map, which contains $333,766$ intersections and $440,922$ road segments.

Mobike Trajectories. Each Mobike trajectory contains a bike ID, a user ID, a temporal range of the trajectory, a pair of start/end locations, and a sequence of intermediate GPS points.

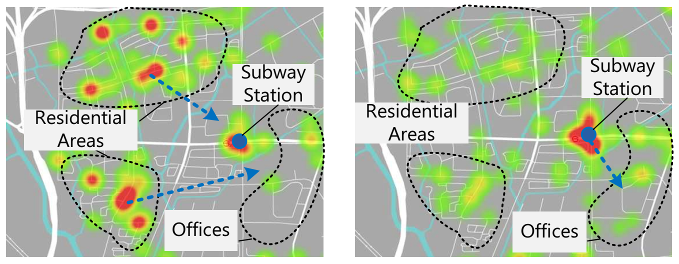
The Mobike dataset is collected in one month (i.e., 09/01/ 2016 - 09/30/2016) from the city of Shanghai. (Figure 8 gives an overview of the spatial distribution of GPS locations). The dataset contains 13,063 unique users, 3,971 bikes, and 230,303 trajectories (with a total of 18,039,283 unique GPS points).

6.2 Mobility Statistics of Mobike Data

Trip Length Distribution. Figure 14(a) summarizes the trip lengths distribution of the Mobike users. From the figure, it is clear that the majority of the trajectories are relatively short, i.e., more than 70% of the trips are shorter than 2 km, as people primarily take bikes for shorter trips. The observation is consistent with the assumption that shared bike service is the solution for the "last mile problem" in public transportation systems [24].

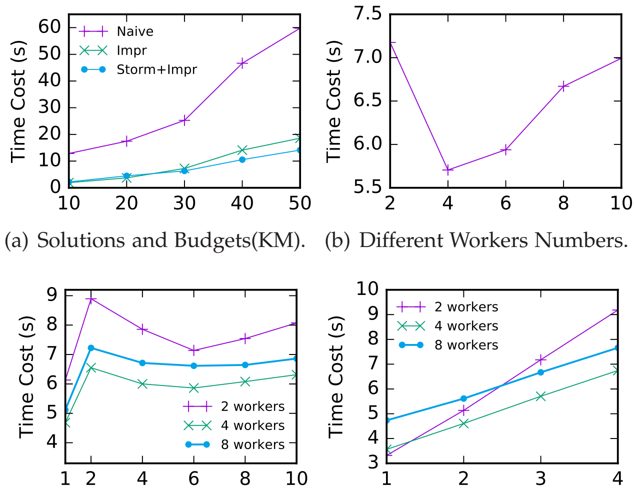
Trip Duration Distribution. Figure 14(b) gives the trajectory duration distribution, where the majority of the trips are within 30 mins. This is because: 1) most of the trips are less than 2 km, which should be completed within 15 mins, and 2) the pricing plan of Mobike charges a user one RMB per 30 mins (we also notice a sudden drop around the 30 min mark).

Trip Temporal Distribution. Figure 14(c) illustrates the distribution of the trip start time. It is obvious that there are two usage peaks, i.e., the morning/evening rush hours. It is interesting to see there is still a small amount of usage late at night, i.e., 10:00PM - 3:00AM, which is generated by the overtime workers.



(a) 6:00AM - 8:00AM (b) 8:00AM-10:00AM

Fig. 15. Temporal Imbalance Example of Mobike Trips.


(a) Solutions and Budgets(KM). (b) Different Workers Numbers.

(c) Different k Values. (d) Different Data Sizes (week).

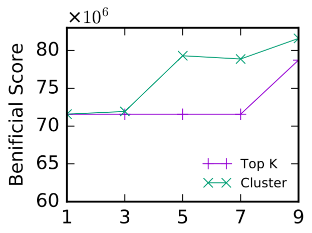
Fig. 16. Efficiency Study.

Road Traversal Distribution. Figure 14(d) depicts the road segment distribution with respect to the number of traversed trajectories (in semi-log scale). It is obvious that most road segments are covered by less than 100 trajectories, which echoes that bikers have destinations all over the urban area. On the other hand, there are over 2,000 road segments, with more than 1,000 trajectories, which validate the necessity of planning effective bike lanes.

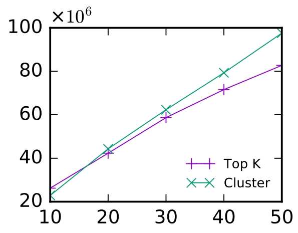
Temporal Imbalance. Figure 15 gives the Mobike trajectory starting locations at different time periods, which exhibits significant temporal imbalance. For example, in the early morning, i.e., Figure 15(a), more trajectories start at the residential areas. However, around 08:00 a.m. to 10:00 a.m, more trips start at the subway station (as Figure 15(b)). After we analyze their final destinations, it is clear that in the early morning, people who live nearby ride bikes to the subway stations for work. Then, after one hour, more people arrive at the subway station and ride to nearby malls and offices.

6.3 Efficiency Evaluation

To evaluate the efficiency of our system (Figure 16), we conduct extensive experiments to study the impact of 4 factors to the execution time, including: the comparison of different solution framworks, the number of Storm workers, the number of components, and the trajectory data size. Unless specified explicitly, the default settings are: $k = 6$, budget $B = 30KM$ (we use the length of the segment as the



(a) Different k Values.



(b) Different Total Budgets.

Fig. 17. Effectiveness Evaluation.

cost $e_i.c$, as the cost and the length are highly correlated), 3 weeks of trajectory data, 8 workers and $\alpha = 1.0$.

Different Solution Frameworks. Figure 16(a) compares the efficiency of different solution frameworks under different total budgets. The frameworks under comparison are brute-force score computing(Naive), the improved score computing algorithm on a single machine (Impr), and the improved algorithm on Storm cluster (Impr+Storm). The budget varies from 10 KM to 50 KM. From the figure, we make the following observations: 1) the time cost grows with the budget. The reason is that, when the budget is large, the expansion can introduce many expansion iterations; 2) the proposed efficient score computing algorithm cuts down the time cost significantly; 3) Storm based system beats the single machine based one when the budget becomes large. This is because, Storm distributes the candidates to its workers, so that the score can be computed in parallel.

Different Storm Worker Numbers. Figure 16(b) presents the execution time under different worker numbers from 2 to 10, where we get 2 observations: 1) as the worker number increases from 2 to 4, the execution time drops down significantly, due to the increase of parallelism; 2) while the worker number increases from 4 to 10, however, the system slows down, since the internal communication time rises. It implies the necessity to make tradeoff between internal communication time and parallelism.

Different k Values. We also investigate the influence of the component number to the execution time. From Figure 16(c), we can observe that the time cost varies with k, however, without an explicit correlation. This is because that, different k values would introduce different numbers of candidates during each expansion iteration, and it's possible that larger k introduces instead less candidates to compute.

Different Trajectory Data Size. Figure 16(d) gives the correlation of time cost to the trajectory data size. The figure indicates that the time cost grows almost in linear with the data size, just as is analyzed in Section 5.1. In addition, we observe that when the data size is small, i.e. one week of trajectories, 2-workers is the most efficient setting, while as the data size increases from 2 to 4 weeks, the system with 4 workers outperforms others, which achieves a good tradeoff between parallelism and communication time cost.

6.4 Effectiveness Studies

In this subsection, we study the effects of different parameters in our system. Note that we didn't compare the effectiveness with other works([8], [9], [10]) because of the differences in constraints and optimization target. Unless

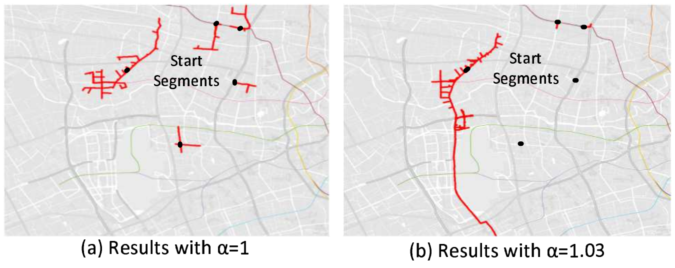


Fig. 18. Effects of α Values.

mentioned explicitly, the default parameters used in the experiments are: $k = 5$, total construction budget $B = 30KM$ and $\alpha = 1$.

Different k Values. Figure 17(a) gives the total beneficial scores $E'.g$ of choosing different numbers of components (i.e., k values). As a result, we have the following insight: 1) in most cases, the spatial clustering-based initialization method gets a higher score; 2) the scores for Top-k method stays the same for $k < 7$, as all the top-7 segments are connected; 3) when k value is small, two methods are similar. This is because in these cases the starting segments of clustering results are the same as the top-k.

Different Total Budgets. Figure 17(b) illustrates the total scores with different total budgets, from 10 KM to 50 KM. From the figure, we make the following observations: 1) the spatial clustering-based initialization method performs better when the budget is larger. 2) when the budget is small, top-k method is better than spatial clustering based method. This is because, when the budget is small, the best strategy may be expanding the segment with the most number of trajectories (essentially the intuition of top-k method). However, when the budget is large, the segments with high scores per cost around the top-1 or top-2 ranked segments can be fully covered (as most bike trajectories is less than 2 KM). At this time, a more effective way should include the segments around other spatial hot spots, rather than still expanding around that top-1 or top-2 ranked segments.

Different α Values. Figure 18 provides the results with different α settings, with the spatial clustering based method, where the red lines are recommended paths and the black dots are their start segments. It is interesting that, when α is large, most of the network expansions happened in one connected component. Moreover, with a higher α, the result of the expansion goes further on some major roads. The reason behind these two phenomena is that, when α is large, higher beneficial scores are given for covering more portion of the bike trajectories.

6.5 Case Study

To better understand the effectiveness of our bike lane recommendations, we conduct a field case study. We choose to visit the area near Jinyun Road subway station, as this area appears in all of our recommendations, regardless of the parameters.

Figure 19(c) gives an overview of the overall POI distribution of the area: 1) Jinyun Road is the terminal station of subway line 13, 2) there is a very large shopping mall (Shanghai Jiangqiao Wanda Plaza) next to the subway

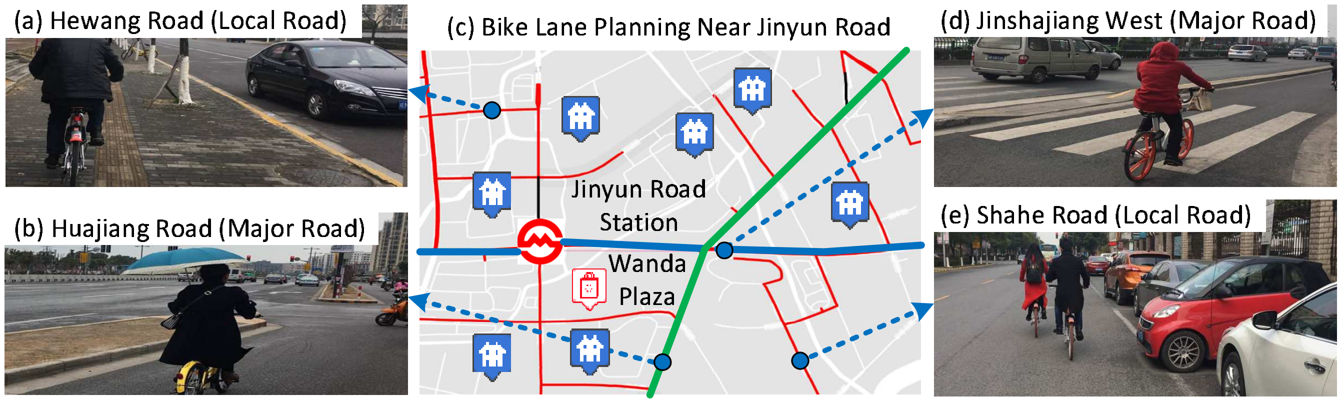

(a) Hewang Road (Local Road)
(b) Huajiang Road (Major Road)
(c) Bike Lane Planning Near Jinyun Road
Jinyun Road Station
Wanda Plaza
(d) Jinshajiang West (Major Road)
(e) Shahe Road (Local Road)

Fig. 19. A Real Case Study Near Jinyun Road Station.

station; and 3) around the subway station, there are many populated residential areas within a 2 km radius, marked as the blue icons on the figure. As a result, cycling is the most convenient way for the residents in this area to go to the subway station or the shopping mall, which explains this area having the highest bike usage density in our dataset.

When we arrive at the Jinyun Road station, we discover that the government has built a few designated bike lanes. Based on our observation, the government plans these bike lanes with a simple strategy: building designated bike lanes for all major roads, and painting bike lanes for the most of the local roads.

For example, the major roads in the figure have designated bike lanes, which are the Jinshajiang West Road (i.e., highlighted in blue) and Huajiang Road (i.e., highlighted in green), as shown in our photos: (Figure 19(b) for Huajiang Road and Figure 19(d) for Jinshajiang West Road). These observations demonstrate the effectiveness of our system, as all of these major roads are included in our bike lane recommendation results.

On the other hand, there are no designated bike lane on local roads, e.g., Hewang Road (Figure 19(a)) and Shahe Road (Figure 19(e)). However, we observe that there is also extensive bike usage on these roads, as they are the paths to highly populated residential areas. Although there are painted bike lanes on the road, the cycling conditions are pretty bad. In Figure 19(a), the bike users have to ride on the sidewalk, as the original bike lane is taken by a parked car. As a consequence, it not only makes the cycling experience much worse, but also is potentially dangerous for people walking or running on the sidewalk. In the other example, i.e., Figure 19(e), at Shahe Road, the bike users are forced to ride on the main lane of the road, as all the space of the biking path is taken by cars, which may lead to traffic accidents.

As a result, based on our analysis, we conclude that the government's current strategy, i.e., building bike lanes only on major roads, is insufficient. With the real bike trajectories and data-driven analysis, we propose that the cycling conditions in these local road segments in our recommendation should be improved. For example, the government should build designated bike lanes, replace off-street parking spaces with (underground) parking garages, and enforce better management of illegal parking.

6.6 System Deployment

Our bike lane planning system is publicly available online [15], where the website user interface is implemented using bootstrap, C#, Asp.NET and Bing Map V8 API, and the system is deployed on Microsoft Azure. The system allows users to interact with it using different parameters, and get bike lane construction recommendations in a short time.

Experts' Review. We presented our system to the government officials from Xuhui District, Shanghai, and collected their feedback. Overall, they highly appreciated our data-driven bike path planning approach and found the system is very useful to help their planning. One of the officials commended: " The idea of using the real sharing bike trajectories for planning the bike lanes is very reasonable. The data mining results from the system will serve as a very solid foundation for our urban planners to build more effective bike lanes in Shanghai".

7 Related Work

Data-Driven Urban Planning. With the availability of massive amounts mobility data from users, vehicles and public transportation systems, urban computing techniques have become more and more popular in many urban planning tasks, as the massive mobility data reflects real travel demands in the physical world [25]. For example, [26] mines patterns in taxi trajectories to suggest road constructions and public transportation projects. [27] infers different function zones in a city based on traffic patterns and POI distribution. [28], [29] identify potential traffic patterns and anomalies in the city based on multiple mobility datasets. In this paper, we focus on providing a data-driven approach to find a more effective and economic way for bike lane planning.

Trajectory Data Mining. The bike lane planning problem is related to the trajectory data mining [**?**], [30], [31], [32], [33], [34], [35], [36], [37]. Many systems have been proposed to discover frequently used routes based on massive trajectory data, e.g., [30], [31], [32], [33], [37], [38]. There are also some projects on clustering/summarizing trajectories on the road network [39], [40], which help urban planners to know the popular routes and improve public transportation system. The closest projects on bike trajectory mining are [8], [9],

[10], which focus on summarizing the trajectory commonality and find out the K-Primary Corridors for bike lanes. However, all of these works can not be directly used for bike lane planning, as they fail to consider the realistic budget and connectivity constraints.

Traditional Bike Lane Planning. Traditional bike lane planning in a city is mainly studied in the transportation domain, and relies heavily on the empirical experience, e.g., [7], [41]. To evaluate the necessity of building bike lanes, [6], [42] provide some high level suggestions based on public surveys and the statistics, such as the road network and POI distributions. There have been some attempts [5] to systematically discover factors for actual bike route choices based on survey data. Recently, there have also been some works on traffic predication and route suggestion based on the station-based bike-sharing systems, e.g., [43], [44].

8 Conclusion

In this paper, we propose a data driven approach to plan bike lanes based on the bike trajectories collected from Mobike (a major station-less bike sharing system) in Shanghai. Our system can address the bike lanes planning problem in a more realistic way, considering the constraints and requirements from urban planners' perspective: 1) budget, 2) construction convenience, and 3) utilization. We also propose a flexible beneficial score function to adjust preferences between the number of covered users and the length of covered trips. The formulated problem is proven to be NP-hard, thus we propose a *greedy network expansion* algorithm with *spatial clustering*. In addition, to improve the efficiency and scalability of the system, we propose a novel trajectory index structure and deploy the system on the cloud.

We perform extensive experiments on a large-scale Mobike dataset and demonstrate the efficiency and effectiveness of our system. We observe that the proposed trajectory score index with Storm significantly improves the efficiency and scalability of the system. We also conduct an on-field case study based on our path recommendation results, and present many important insights to improve cycling convenience in a given area. A demonstration system is deployed for public use, and the expert feedback from the government officials from Xuhui District, Shanghai, confirms the effectiveness and usability of our system.

Acknowledgments

Yanhua Li was supported in part by NSF grants CNS-1657350 and CMMI-1831140, and a research grant from DiDi Chuxing Inc. Hui He was supported by NSFC (Grant No. 61672186, No. 61872110), and the National Key R&D Program of China, NO.2017YFB803304. Yu Zheng was supported by NSFC (Grant No. 61672399, No. U1609217).

References

[1] "Transport minister encourages people to get on their bike for cycle to work day," https://www.gov.uk/government/news/transport-minister-encourages-people-to-get-on-their-bike-for-cycle-to-work-day, 2015.

[2] X. Zhu, "Bike sharing schemes promote green transport," http://www.telegraph.co.uk/news/world/china-watch/technology/sharing-bikes-to-promote-green-transport/, 2016.

[3] D. Rojas-Rueda, A. De Nazelle, O. Teixidó, and M. Nieuwenhuijsen, "Replacing car trips by increasing bike and public transport in the greater barcelona metropolitan area: a health impact assessment study," *Environment international*, vol. 49, pp. 100–109, 2012.

[4] J. Pucker, "Cycling safety on bikeways vs. roads." *Transportation Quarterly*, vol. 55, no. 4, pp. 9–11, 2001.

[5] T. Hyodo, N. Suzuki, and K. Takahashi, "Modeling of bicycle route and destination choice behavior for bicycle road network plan," *JTRB*, no. 1705, pp. 70–76, 2000.

[6] G. Rybarczyk and C. Wu, "Bicycle facility planning using gis and multi-criteria decision analysis," *Applied Geography*, 2010.

[7] G. French, J. Steer, and N. Richardson, "Handbook for cycle-friendly design," https://goo.gl/m3DwoY, 2014.

[8] M. R. Evans, D. Oliver, S. Shekhar, and F. Harvey, "Summarizing trajectories into k-primary corridors: a summary of results," in *SIGSPATIAL GIS*. ACM, 2012, pp. 454–457.

[9] ——, "Fast and exact network trajectory similarity computation: a case-study on bicycle corridor planning," in *UrbComp*. ACM, 2013, p. 9.

[10] Z. Jiang, M. Evans, D. Oliver, and S. Shekhar, "Identifying k primary corridors from urban bicycle gps trajectories on a road network," *Information Systems*, vol. 57, pp. 142–159, 2016.

[11] J. Odeck, "Cost overruns in road construction—what are their sizes and determinants?" *Transport policy*, 2004.

[12] W. Chung, J. Stückelberger, K. Aruga, and T. W. Cundy, "Forest road network design using a trade-off analysis between skidding and road construction costs," *Canadian journal of forest research*, vol. 38, no. 3, pp. 439–448, 2008.

[13] J. Russell, "Meet Mobike, a billion-dollar bike-sharing startup from China," https://techcrunch.com/2017/07/12/chinese-bike-sharing-startup-mobike/, 2017.

[14] J. Bao, T. He, S. Ruan, Y. Li, and Y. Zheng, "Planning bike lanes based on sharing-bikes' trajectories," in *SIGKDD*. ACM, 2017, pp. 1377–1386.

[15] "Urbanbike system," http://urbanbike.chinacloudsites.cn/, 2017.

[16] Y. Zheng, "Trajectory data mining: an overview," *TIST*, vol. 6, no. 3, p. 29, 2015.

[17] J. Yuan, Y. Zheng, C. Zhang, X. Xie, and G.-Z. Sun, "An interactive-voting based map matching algorithm," in *MDM*. IEEE Computer Society, 2010, pp. 43–52.

[18] J. Bao, R. Li, X. Yi, and Y. Zheng, "Managing massive trajectories on the cloud," in *SIGSPATIAL GIS*. ACM, 2016, p. 41.

[19] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," in *VLDB*. VLDB Endowment, 2003, pp. 802–813.

[20] J. Bao, C.-Y. Chow, M. F. Mokbel, and W.-S. Ku, "Efficient evaluation of k-range nearest neighbor queries in road networks," in *MDM*. IEEE, 2010, pp. 115–124.

[21] J. H. Ward Jr, "Hierarchical grouping to optimize an objective function," *Journal of the American statistical association*, vol. 58, no. 301, pp. 236–244, 1963.

[22] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *KDD*, vol. 96, no. 34, 1996, pp. 226–231.

[23] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[24] P. DeMaio, "Bike-sharing: History, impacts, models of provision, and future," *JPT*, vol. 12, no. 4, p. 3, 2009.

[25] Y. Zheng, L. Capra, O. Wolfson, and H. Yang, "Urban computing: concepts, methodologies, and applications," *TIST*, vol. 5, no. 3, p. 38, 2014.

[26] Y. Zheng, Y. Liu, J. Yuan, and X. Xie, "Urban computing with taxicabs," in *Proceedings of the 13th international conference on Ubiquitous computing*. ACM, 2011, pp. 89–98.

[27] J. Yuan, Y. Zheng, and X. Xie, "Discovering regions of different functions in a city using human mobility and pois," in *SIGKDD*. ACM, 2012, pp. 186–194.

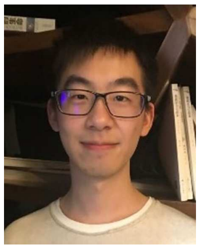
[28] S. Chawla, Y. Zheng, and J. Hu, "Inferring the root cause in road traffic anomalies," in *ICDM*. IEEE, 2012, pp. 141–150.

[29] L. Hong, Y. Zheng, D. Yung, J. Shang, and L. Zou, "Detecting urban black holes based on human mobility data," in *SIGSPATIAL GIS*. ACM, 2015, p. 35.

[30] X. Li, J. Han, J.-G. Lee, and H. Gonzalez, "Traffic density-based discovery of hot routes in road networks," in *International Symposium on Spatial and Temporal Databases*. Springer, 2007, pp. 441–459.
[31] Z. Chen, H. T. Shen, and X. Zhou, "Discovering popular routes from trajectories," in *ICDE*. IEEE, 2011, pp. 900–911.
[32] W. Luo, H. Tan, L. Chen, and L. M. Ni, "Finding time period-based most frequent path in big trajectory data," in *SIGMOD*. ACM, 2013, pp. 713–724.
[33] D. Oliver, S. Shekhar, X. Zhou, E. Eftelioglu, M. R. Evans, Q. Zhuang, J. M. Kang, R. Laubscher, and C. Farah, "Significant route discovery: A summary of results," in *International Conference on Geographic Information Science*. Springer, 2014, pp. 284–300.
[34] 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*, vol. 23, no. 1, pp. 1–10, 2017.
[35] 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 GIS*. ACM, 2016, p. 51.
[36] G. Wu, Y. Ding, Y. Li, J. Bao, Y. Zheng, and J. Luo, "Mining spatio-temporal reachable regions over massive trajectory data," in *ICDE*, 2017, pp. 1–12.
[37] A. M. Hendawi, J. Bao, M. F. Mokbel, and M. Ali, "Predictive tree: An efficient index for predictive queries on road networks," in *ICDE*. IEEE, 2015, pp. 1215–1226.
[38] T. He, J. Bao, R. Li, S. Ruan, Y. Li, C. Tian, and Y. Zheng, "Detecting vehicle illegal parking events using sharing bikes' trajectories," *SIGKDD. ACM*, 2018.
[39] A. M. Hendawi, J. Bao, and M. F. Mokbel, "iroad: a framework for scalable predictive query processing on road networks," *Proceedings of the VLDB Endowment*, vol. 6, no. 12, pp. 1262–1265, 2013.
[40] A. Kharrat, I. S. Popa, K. Zeitouni, and S. Faiz, "Clustering algorithm for network constraint trajectories," in *Headway in Spatial Data Handling*. Springer, 2008, pp. 631–647.
[41] B. Han, L. Liu, and E. Omiecinski, "Neat: Road network aware trajectory clustering," in *ICDCS*. IEEE, 2012, pp. 142–151.
[42] J. Dill and K. Voros, "Factors affecting bicycling demand: initial survey findings from the portland, oregon, region," *JTRB*, no. 2031, pp. 9–17, 2007.
[43] K. M. Parker, J. Rice, J. Gustat, J. Ruley, A. Spriggs, and C. Johnson, "Effect of bike lane infrastructure improvements on ridership in one new orleans neighborhood," *Annals of behavioral medicine*, vol. 45, no. 1, pp. 101–107, 2013.
[44] Y. Li, Y. Zheng, H. Zhang, and L. Chen, "Traffic prediction in a bike-sharing system," in *SIGSPATIAL GIS*. ACM, 2015, p. 33.
[45] J. Liu, L. Sun, W. Chen, and H. Xiong, "Rebalancing bike sharing systems: A multi-source data smart optimization," in *SIGKDD*. ACM, 2016, pp. 1005–1014.

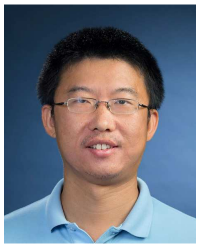
Tianfu He Tianfu He is a Ph.D. student in School of Computer Science, Harbin Institute of Technology. Before that he recieved the B.E degeree from Harbin Institute of Technology in 2016. His current research interest involves urban computing, spatio-temporal data management and data mining, especially trajectory data mining.

Jie Bao Jie Bao got his Ph.D degree in Computer Science from University of Minnesota at Twin Cities in 2014. He worked as a researcher in Urban Computing Group at MSR Asia from 2014 to 2017. He currently leads the Data Platform Division in Urban Computing Business Unit, JD Finance. His research interests include: Spatio-temporal Data Management/Mining, Urban Computing, and Location-based Services.



Sijie Ruan Sijie Ruan is a Ph.D. student in the School of Computer Science and Technology, Xidian University. He received his B.E. degree from Xidian University in 2017. His research interests include urban computing, spatio-temporal data mining, and distributed systems. He was an intern in MSR Asia from 2016 to 2017. He is now a research intern in JD Intelligent Cities Research, under the supervision of Prof. Yu Zheng and Dr. Jie Bao.

Ruiyuan Li Ruiyuan Li is a Ph.D. student at the School of Computer Science and Technology, Xidian University, China. He received his B.E. degree and M.S. degree from Wuhan University, Hubei, China in 2013 and 2016, respectively. His research focuses on Urban Computing, Spatio-temporal Data Management on the Cloud, and Distributed Computing. He is now an intern student in Urban Computing Lab, JD Group, China, under the supervision of Prof. Yu Zheng and Dr. Jie Bao.



Yanhua Li Yanhua Li received two Ph.D. degrees in electrical engineering from Beijing University of Posts and Telecommunications in 2009 and in computer science from University of Minnesota at Twin Cities in 2013, respectively. He has worked as a researcher in HUAWEI Noah's Ark LAB at Hong Kong from Aug 2013 to Dec 2014, and has interned in Bell Labs in New Jersey, Microsoft Research Asia, and HUAWEI research labs of America from 2011 to 2013. He is currently an Assistant Professor in the Department of Computer Science at Worcester Polytechnic Institute in Worcester, MA. His research interests are big data analytics and urban computing in many contexts, including urban network data analytics and management, urban planning and optimization.

Hui He Hui He is currently an Associate Professor in School of Computer Science, Harbin Institute of Technology. She is a member of the IEEE, ACM and CCF. Her reserach interests inlcudes information technology, big data processing and analysis and mobile network computing. She has accomplished many projects such as National High Technology Research and National Science Foundation Projects.

Yu Zheng Yu Zheng is a Vice President and Chief Data Scientist at JD Digits, passionate about using big data and AI technology to tackle urban challenges. His research interests include big data analytics, spatio-temporal data mining, machine learning, and artificial intelligence. He also leads the JD Urban Computing Business Unit as the president and serves as the director of the JD Intelligent City Research. Before Joining JD, he was a senior research manager at Microsoft Research. Zheng is also a Chair Professor at Shanghai Jiao Tong University, an Adjunct Professor at Hong Kong University of Science and Technology.