

Received November 29, 2018, accepted December 17, 2018, date of publication January 9, 2019, date of current version January 29, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2891570

# PPVF: A Novel Framework for Supporting Path Planning Over Carpooling

**BIN WANG<sup>1</sup>**, **RUI ZHU<sup>2</sup>**, **SITING ZHANG<sup>1</sup>**, **ZHENG ZHAO<sup>1</sup>**,  
**XIAOCHUN YANG<sup>1</sup>**, (Member, IEEE), AND **GUOREN WANG<sup>3</sup>**, (Member, IEEE)

<sup>1</sup>Collage of Computer Science, Northeastern University, Shenyang 110004, China

<sup>2</sup>Collage of Computer Science, Shenyang Aerospace University, Shenyang 110136, China

<sup>3</sup>Collage of Computer Science, Beijing Institute of Technology, Beijing 100081, China

Corresponding author: Rui Zhu (neuruzhu@gmail.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61572122, Grant U1736104, Grant 61702344, and Grant 61532021, in part by the Fundamental Research Funds for the Central Universities under Grant N171602003 and Grant N161606002, and in part by the Liaoning BaiQianWan Talents Program.

**ABSTRACT** With the rapid development of mobile Internet and sharing economy, carpooling over real-time taxi-calling service has attracted more and more attention. Many popular taxi-calling service platforms, e.g., didi and Uber, have developed carpooling service to the passengers. Their goal is maximizing overall profit while meeting passengers' convenience and economic benefits. Many researchers have deeply studied this problem. They usually focus on minimizing the total travel distance of drivers or making passengers' cost as fair as possible. However, these efforts ignore the fact that a "high quality" planning path crowded with potential passengers is more valuable than a "low quality" path for both workers and passengers. An effective high quality path planning algorithm is more desired. In this paper, we propose an efficient framework, named PPVF (short for path prediction and verification-based framework) for path planning over the road network. We first select a group of high quality paths from the historical transaction record set and manage them by proposing a novel index named PCR-Tree. Then we use them for supporting the path planning. Furthermore, we propose a *searching and verification*-based algorithm for further improving the path planning quality. Theoretical analysis and extensive experimental results demonstrate the effectiveness of the proposed algorithms.

**INDEX TERMS** Path planning, path selection, PCR-tree, ELM.

## I. INTRODUCTION

With the rapid development of mobile Internet and sharing economy, O2O (short for Online To, Offline) service platforms has gained increasing popularity [1]. Many commercial applications have emerged in daily life. As a representative one, carpooling over real-time taxi-calling service has attracted more and more attention. Many popular taxi-calling service platforms, e.g. didi, Uber, have developed carpooling services.

Different from traditional service pattern which restricts a taxi to serve for a single passenger at one moment, carpooling allows a taxi to provide service for multi-passengers [2], [3]. In other words, a group of passengers can share a taxi simultaneously. Once a passenger submits a request consisting of the pick-up and drop-off points to the platform, the platform determines a driver (even en-route) to pick up the passenger according to their locations. From then on, the platform *plans*

paths for drivers and *quotes* to passengers according to the trip distance. Within carpooling platforms, passengers could reduce trip cost by sharing a taxi while enjoying fast and convenient transportation. Other than traditional path planning problem that usually use the shortest path as the planning path, path planning under carpooling allows distance consumption with certain thresholds (i.e., detour) in exchange for users' economic benefits (i.e., a lower price).

Intuitively, the price under *carpooling model* is mainly determined by two factors. (i) the distance between the starting/destination locations of passengers; (ii) the number of passengers sharing the taxi. Usually, the more the passengers share the taxi together, the less the fees they pay. However, if platform focuses too much on passengers' economic benefits, it may allow a taxi to be shared by too many passengers. Besides, it may lead a big difference between the shortest path and planning path. All the aforementioned issues may reduce

the service quality. Therefore, it is very important to plan a reasonable path to satisfy both passengers' convenience and economic benefits.

The rich application inspires lots of research works on carpooling problem. To the best of our knowledge, the state-of-the-art efforts on ride-sharing can be categorized into two groups, namely *distance-based* approaches and *price-based* approaches. The key of distance-based approaches is minimizing the total travel distance of drivers [4], [5], [5], [6]. However, minimizing drivers' total travel distance is not always equivalent to minimizing overall shorter trips for passengers [2]. In other words, when a taxi is assigned with a new request, the minimum increase in total travel distance is not necessarily the most cost-effective option. In this case, parts of passengers' convenience and economic benefits may not be satisfied. Price-based approaches solve this problem via allowing both passengers and taxis set their monetary expectations for participating in ride-sharing based on their predefined profiles [7], [8], [8]. However, these efforts still ignore the fact that a "high quality" path crowded with potential passengers is more valuable than a "low quality" path for both taxis and passengers. An effectively "high quality" path planning algorithm is desired.

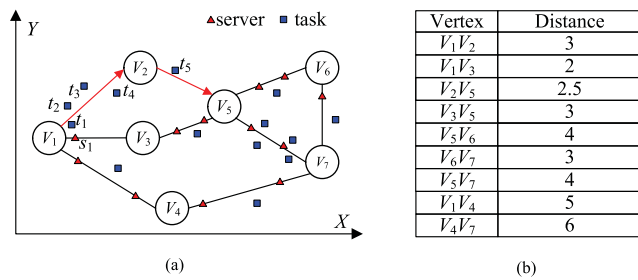


FIGURE 1. The Paths Diversity.

*Example 1 (Path Planning Under Carpooling):* Take an example in Fig 1, passenger  $u$  in taxi  $w$  wants to go from  $v_1$  to  $v_5$ , two paths  $p_1$  and  $p_2$  can be used as the candidate planning paths. Here,  $p_1$  is the shortest path between  $v_1$  and  $v_5$ , and  $p_2$  is another path between  $v_1$  and  $v_5$ . Compared with  $p_1$ ,  $p_2$  is more preferable. The reason behind is, the distance between  $v_1$  and  $v_5$  over  $p_2$  is only a little longer than that of under  $p_1$ , but there are many other passengers around  $p_1$ ,  $u$  has more chance to share the taxi with others, and hence the traveling fee could be effectively reduced. Therefore, we regard the path  $p_1$  as a high quality path, and regard  $p_2$  as a relatively low quality path.

Based on the above observation, in this paper, we propose an efficient framework, named PPVF(short for path prediction and verification based framework) for path planning. For simplicity, we explain our solution under road network environment. Given road network  $G$ , the key of PPVF is pre-computing "high quality" paths between every two vertices in  $G$  according to the historical transaction record set. In this way, when a passenger submits a request to the platform, the platform provides the passenger with a reasonable

path according to the pre-computing results. However, PPVF meets the following three challenges:

**A. CHALLENGES**

First of all, it is difficult to evaluate which kinds of paths could be regarded as "high quality". Secondly, it is impractical to maintain "high quality" paths for every two vertices, which causes high maintenance cost. Whats more, the distribution of taxis/passengers always changes over time, which leads to that the pre-computing results may not be suitable as the planning path. Therefore, the second challenge is effectively verifying the pre-computing results and dynamically adjusting the planning result if needed.

**B. CONTRIBUTIONS**

In summary, the contribution of this paper are as follows.

- A Novel Query Definition: We define a novel query called  $(\epsilon, \varphi)$ -CBPP (short for constraint based path planning) query over road network. We use two thresholds  $\epsilon$  and  $\varphi$  as the constraint condition, where  $\varphi$  is the passengers amount threshold. As for  $\epsilon$ , it is a threshold using for bounding distance difference between shortest path and planning path. Our goal is providing passengers with high quality path in the premise of thresholds constraint condition.
- High Quality Path Maintenance: We propose a group of strategies for high quality path maintenance. First of all, a novel algorithm *distance-constraint based weight-diversity* is designed for high quality path selection. Furthermore, we develop a compression algorithm to reduce the selected paths scale. Finally, we propose a novel index named PCR-Tree(short for Path-Compression-R)-Tree to maintain these paths. As the basic, we self-adaptively partition these paths into a few sub-paths according to their location information. In this way, PCR-Tree could use a group of MBRs with relatively small area to bound these sub-paths, and enhance the powerful filtering ability accordingly.
- Query Algorithm: We design the algorithm HQPS to answer  $(\epsilon, \varphi)$ -CBPP query. Its main idea is searching on PCR-Tree for finding candidate paths. Based on the searching results, we propose an ELM-based method for verifying whether the taxi/passenger distribution over candidate paths are similar with that of the ones in the historical record set, and select the planning path based on the verification results.

The rest of this paper is organized as follows. Section 2 reviews related work and presents the definition of  $(\epsilon, \varphi)$ -CBPP. Section 3 gives an overview of the framework. Section 4 and Section 5 discuss high quality maintenance and query processing algorithm. We report our experimental results in Section 6. Section 7 concludes this paper.

**II. PRELIMINARIES AND DEFINITIONS**

In this section, we first review the algorithms about the problem of spatio-temporal crowdsourcing, ride-sharing and

**TABLE 1.** The summary of notations.

Notation	definition
$t$	task
$w$	worker
$IR(x)$	the impact region of the point $x$
$CWF(e)$	the cumulative weight function of an edge $e$
$KP(i, j)$	the KPath set corresponding to the vertexes $v_i$ and $v_j$
$SW(w)$	the weight of the worker $w$

path planning. Thereafter, we introduce a novel machine learning technique called ELM. Lastly, we define the  $(\varepsilon, \varphi)$ -CBPP query over road network. Table 1 summarizes the mathematical notations used in the paper.

## A. RELATED WORK

### 1) SPATIO-TEMPORAL CROWDSOURCING

Recently, spatio-temporal crowdsourcing has attracted more and more attention. The main participants in spatio-temporal crowds mainly include crowdsourcing task requesters and crowdsourcing participants, who connect through the time-space crowdsourcing platform. The platform preprocesses the requested task and participant information first, and then feedback the relevant information to the requester and participants.

The task could be divided into static offline tasks and dynamic online tasks. The main difference between them is that the tasks and participants in the dynamic online tasks are unknown. Kazemi *et al.* [9] puts forward by the spatial-temporal crowds allocation queries in static offline scenarios, using bipartite graph to model the query. Participants of the task are regarded as the left and right point sets in the bipartite graph that do not intersect. Accordingly, the query problem is reduced to the maximum bipartite graph matching problem. However, the real-time is demand of more spatial-temporal crowds is very high, which belongs to online dynamic tasks.

Tong *et al.* [10] proposes a matching model for online matching. The model allows tasks and participants to dynamically appear in any position and any order. Whenever a new task appears, the platform assigns a participant for it immediately or “suspends” it until a subsequent participant comes or the mission deadline. Some online algorithms are used to solve such problems, such as simple greedy algorithms. Its issue is the performance is affected by the arrival order of tasks and participants. Tong *et al.* [11] further gives an efficient online algorithm for analyzing the worst case and the average case. Subsequently, they have shown that the dynamic task allocation model of greedy algorithm has good performance effect through a large number of real and simulation experiments. Furthermore, they proposed a two-step framework, which integrates both offline prediction and online task assignment, aims at addressing the problem that a worker may move around while waiting for a task. A prediction-based online task assignment algorithm is designed, which achieve a 0.47 competitive ratio.

### 2) RIDE-SHARING

The majority of ride-sharing problem fall into two categories: static and dynamic ride sharing. In static condition, the driving route is pre-computed since all drivers, passengers are known and will not change dynamically over time. Furuhata *et al.* [18] made a survey of various types of static ride-sharing considering different settings and objectives. Santi *et al.* [19] quantified the potential of ride-sharing based on a graph-based approach. Furthermore Cici *et al.* [20] evaluated the potential of carpooling using four cities mobile dataset. In addition, ride-sharing problem was proven to be NP-hard.

There has been a surge of research in real-time ride-sharing [22]–[25] due to the emergence of many ride-sharing mobile applications (e.g., Uber and Lyft). In real-time ride-sharing, high computation cost are not allowed since the riders and drivers statuses changes over time. With the objective to reduce the total travel distance of drivers, Ma *et al.* used spatial-temporal index to retrieve the candidate drivers and proposed a ride-sharing dispatch system named T-share. On the other hand, Huang developed a kinect tree scheduling algorithm which can dynamically assign trip request to drivers with minimum travel distance. Ota *et al.* introduced a data-driven simulation framework that was capable of analyze ride-sharing. A ride-sharing system to maximize the number of matched request was proposed by Santos *et al.* The majority of these studies aimed to minimize the total travel distance of drivers, however, this did not mean shorter travel distance for the riders. To maximize the total profit of the platform, Asghari *et al.* [2] proposed an auction-based framework called APART. In APART framework, drivers and passengers are considered as bidders and goods respectively. Once a new request is generated, each nearby driver submits a bid to the server. The bid is computed by the pricing model which satisfying both riders and passengers self-defined profiles. The driver who gives the highest bid will get the request.

### 3) PATH PLANNING OVER ROAD NETWORK

Generally, the shortest path algorithm can be divided into two basic categories: Index-based and non-index-based.

Among those non-index-based algorithm, the Dijkstra’s Algorithm [12] is a classic algorithm, which achieves a time complexity of  $O(n^2)$ . It spreads outward from the source until reaching to the end. To address the problem that the Dijkstra’s Algorithm can not deal with negative weighted edges, Bellman, Ford, and Moore developed the Bellman-Ford algorithm [13] which can solve the shortest path on negative weighted graph through repeated iterations. Whats more, it can detect negative circles. However, the Bellman-Ford algorithm is much slower than the Dijkstra’s Algorithm. Both the Dijkstra’s Algorithm and the Bellman - Ford algorithm solve single-source shortest path problems(sssp). Floyd algorithm [14] attempts to solve all pairs shortest-paths (APSP). It is actually a dynamic programming algorithm. The shortest path matrix between each two points of a graph is obtained

by the weight matrix of the graph. Its time complexity is  $O(n^3)$ .

For the index-based algorithms, Fakcharoenphol and Rao [15] propose a method that generates a non-planar graph through vertex subsets then calculates the shortest path tree. The space and time complexity of the preprocessing stage is  $O(n)$ . Klein et al. [16] use Jordan curve to preprocess the graph and use the Dijkstra's Algorithm and the Bellman - Ford algorithm to calculate the shortest path. Wulff-Nilsen [17] use the wiener index to find the path query answers which can only deal with unweight graph.

### B. EXTREME LEARNING MACHINE

In this subsection, we present a brief overview of extreme learning machine (ELM) [26], developed by Huang et al, which is used for identifying unreliable documents in our paper. ELM is based on a generalized single-hidden-layer feed forward network (SLFN). Also, it has many variants [28]–[30]. Compared with neural networks, its hidden-layer nodes are randomly chosen instead of iteratively tuned. In this way, it provides good generalization performance at thousands of times faster than traditional popular learning algorithms (e.g., SVM [31], neural networks). Also, it gets better performance due to its *universal approximation capability* and *classification capability* [26].

$$\beta_{k+1} = \beta_k + \mathbf{P}_{k+1} \mathbf{H}_{k+1}^T (\mathbf{T}_{k+1} - \mathbf{H}_{k+1} \beta_k) \quad (1)$$

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \mathbf{P}_k \mathbf{H}_{k+1}^T (\mathbf{I} + \mathbf{H}_{k+1} \mathbf{H}_{k+1}^T)^{-1} \mathbf{H}_{k+1} \mathbf{P}_k \quad (2)$$

As an improved version, the online sequential extreme [26] learning machine (abbreviated as OS-ELM) is proposed. Compared with the basic ELM, OS-ELM could learn data one-by-one or chunk-by-chunk with fixed or varied size. Thus, it is suitable for processing streaming data.

$$\mathbf{H}_{k+1} = \begin{bmatrix} \mathbf{G}(\mathbf{a}_1, b_1, \mathbf{x}_{\sum N_j+1}) & \dots & \mathbf{G}(\mathbf{a}_N, b_N, \mathbf{x}_{\sum N_j+1}) \\ \vdots & \dots & \vdots \\ \mathbf{G}(\mathbf{a}_1, b_1, \mathbf{x}_{\sum N_j+1}) & \dots & \mathbf{G}(\mathbf{a}_1, b_1, \mathbf{x}_{\sum N_j+1}) \end{bmatrix}_{N_{k+1} \times l} \quad (3)$$

Specially, given a set of samples  $(x_i, t_i)$ , where  $x_i = [x_{i1}, x_{i2}, \dots, x_{in}]^T \in \mathcal{R}^n$  and  $t_i = [t_{i1}, t_{i2}, \dots, t_{in}]^T \in \mathcal{R}^m$ , OS-ELM first selects the activation function, the hidden node number and so on. Then, OS-ELM is employed in a two-phase method including: (i) **initialization** (ii) **sequential learning**.

$$\mathbf{T}_{k+1} = \left[ \mathbf{T}_{(\sum_{j=0}^k N_j)+1} \dots \mathbf{T}_{(\sum_{j=0}^k N_j)} \right]_{N_{k+1} \times m}^T \quad (4)$$

In the first phase, OS-ELM uses a small set of samples for training. The second phase employs the learning in a chunk-by-chunk way. In the  $k$ -th chunk of new training data, OS-ELM firstly computes the partial hidden layer and the output matrix  $\mathbf{H}_{k+1}$ . Lastly, OS-ELM computes the output weight matrix  $\beta_{k+1}$ , where  $\beta_{k+1}$ ,  $\mathbf{H}_{k+1}$ , and  $\mathbf{T}_{k+1}$  are computed according to Equation 5 to Equation 4.

### C. PROBLEM DEFINITION

*Definition 2 (Task):* A task  $t$ , denoted by the tuple  $\langle w, u, s, d, p \rangle$ , is created when a passenger  $u$  submits a requirement to the platform with a starting location  $s$  and a destination location  $d$  over road network. Once a task is created, the platform should assign a worker  $w$  for  $t$ , and plan a path  $p$  so as to transport  $u$  from  $s$  to  $d$ .

A moment thought could reveal that it is suitable for using the shortest path as the planning path. However, in the *carpooling mode*, it allows a taxi to execute multiple tasks simultaneously. In other words, a taxi could transport more than one passenger at the same time. For simplicity, in the following, we call taxi as worker. In order to guarantee the service quality, we propose the concept of  $(\varphi, \varepsilon)$ -worker. A  $(\varphi, \varepsilon)$ -worker  $w$  contains three states, that are, *empty*, *non-full*, and *full*. In the following, we will formally introduce the concept of worker,  $(\varphi, \varepsilon)$ -worker, and three states of a  $(\varphi, \varepsilon)$ -worker.

*Definition 3 (Worker):* A worker  $w$ , denoted by the tuple  $\langle s, \mathcal{T}, \mathcal{P} \rangle$ , is serving for passengers, which transports passengers from their starting locations to their destination locations. Here,  $s$  is the location of  $w$ , and  $\mathcal{T}$  is the task sets,  $\mathcal{P}$  is the path sets. In addition,  $\mathcal{P}$  should contain all the starting locations(also destination locations) of the tasks in  $\mathcal{T}$ .

*Definition 4 (( $\varphi, \varepsilon$ )-Worker):* Given a set of tasks  $\mathcal{T} \{t_1, t_2, \dots, t_m\}$  taken by the worker  $w$ , we call  $w$  as a  $(\varphi, \varepsilon)$ -Worker if (i)  $w$  could take at most  $\varphi$  tasks simultaneously; (ii) for each task  $t_i$ ,  $|\mathbf{sp}(t_i)|$  should be less than  $\varepsilon|t_i.p|$ . Here,  $\mathbf{sp}(t_i)$  refers to the shortest path between  $t_i.s$  and  $t_i.d$ ,  $|\mathbf{sp}(t_i)|$  refers to distance under  $\mathbf{sp}(t_i)$ .

- **Empty Worker:** An empty worker executes none task currently. Given an empty worker  $w$  and a new task  $t$ , if the distance between them is less than a threshold  $r$  over 2D space, the platform assigns  $t$  to  $w$  with the probability  $p_1$ . Here, the probability computation will be discussed in Section V-A.
- **Non-Full Worker:** A non-full worker  $w$  executes  $|w|$  tasks currently, i.e.,  $0 < |w| < \varphi$ . When a new task  $t$  is submitted, if  $w$  and  $t$  are satisfying the following constraint conditions, the platform assigns  $t$  to  $w$  with the probability  $p_2$ . Here, the constraint conditions are: (i) the distance between  $t.s$  and  $w.s$  is less than  $r$  in the 2D space; (ii)  $w$  should still be a  $(\varphi, \varepsilon)$ -worker once  $t$  is assigned to  $w$ .
- **Full Worker:** A full worker  $w$  executes  $\varepsilon$  tasks currently. The platform could not assign new task for it until it turns to a non-full worker.

*Example 5 (Worker States):* Take an example in Fig 2. The worker  $w$  is empty at the moment  $T_0$ . Since the distance between  $w.s$  and  $t_{1.s}$  is less than the threshold  $r$  over 2D space, the platform assigns  $t_1$  to  $w$ . We can find the corresponding shortest path,  $p_1$ , via accessing the road network  $G$  according to  $t_1.d$ (i.e., vertex  $v_5$ ). Among all the other paths, for the reason that  $p_2$  satisfies the distance constraint condition, and there exist many tasks around  $p_2$ , we select  $p_2$  as the planning path. After  $w$  passes the vertexes  $v_2, t_2$  and  $t_3$

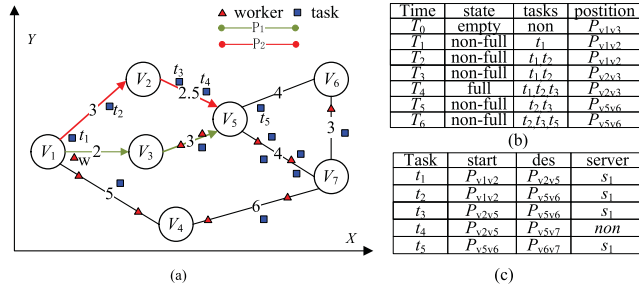


FIGURE 2. Worker States Statements.

are assigned to  $w$ . At that moment, the tasks set of  $w$  turns to  $\{t_1, t_2, t_3\}$ . In addition, since  $\varphi$ ,  $w$  becomes a *full worker*, and cannot take other tasks. After  $w$  achieves the task  $t_1$ ,  $w$  turns to *non-full*, and has the ability to take new task again.

In order to evaluate the cost of tasks and benefit workers, we introduce the parameter *worker weight*. For simplicity, we call a path with the length  $m$  as a unit path. Given a worker  $w$  with the unit paths set  $\mathcal{U}(w)$ , its weight, denoted by  $\mathbf{SW}(w)$ , equals to  $\frac{\sum \mathbf{SW}(w, up_i)}{|\mathcal{U}(w)|}$ . Here,  $up_i$  refers to the  $i$ -th unit path contained in  $\mathcal{U}(w)$ , and  $\mathbf{SW}(w, up_i)$  refers to the maximal number of tasks  $w$  executes during the moment  $w$  passes  $up_i$ .

$$\text{cost}(t) = \frac{|\text{sp}(t)|\mu \sum e^{(1-\rho\mathbf{SW}(w, up_i))}}{|t.p|} \quad (5)$$

$$\text{gain}(w) = \sum_{i=0}^{i=|\mathcal{T}(w)|} \frac{\mu|\text{sp}(t_i)|}{|t_i.p|} C e^{(1-\rho\mathbf{SW}(w, up_i))} t_i.p \cap up_j \neq \emptyset \quad (6)$$

Equation 5 and Equation 6 depict the cost of a task and the revenue of a worker respectively. We can conclude that the higher the worker weight is, the lower the task costs, and the more the worker gains per unit path. Obviously, it is significant for workers to execute, as much as possible, tasks at the same time. However, if a region  $R$  contains many workers but few tasks, the workers in  $R$  almost have no chance to take many tasks(e.g., the path  $p_2$  in Fig 2). Therefore, the goal of this paper is to plan “high quality” paths so as to benefit workers and passengers.

In this paper, we introduce the *cumulative weight function* for evaluating the quality of a path. Let  $e$  be an edge belong to the road network  $G$ . For each point  $x_i$  in  $e$ , its weight, i.e.,  $\mathbf{W}(x)$  is computed by  $\frac{\alpha}{\beta}$ . Here,  $\alpha$  refers to the number of workers contained in the impact region of  $x$ , denoted by  $\text{IR}(x)$ ,  $\beta$  refers to the number of tasks contained in  $\text{IR}(x)$ , where  $\text{IR}(x)$  is a circle with the center  $x$  and the radius  $r$ . Based on the points weight, the cumulative weight function of  $e$  could be described by the function  $\text{CWF}(x)$  ( $x \in e$ ), and the cumulative weight function of a path could be described by  $\text{CWF}(x)$  ( $x \in p$ ), i.e., equals to the expected worker weight of  $w$  during the moment  $w$  passes  $p$ . We now formally introduce the  $(\varepsilon, \varphi)$ -CBPP problem.

**Definition 6:** ( $(\varepsilon, \varphi)$ -CBPP problem). Given a  $(\varepsilon, \varphi)$ -worker  $w$ , the passenger set  $P$  and the worker set  $W$

over road network  $G$ , the  $(\varepsilon, \varphi)$ -CBPP problem is to find a path  $p \in G$  according to its cumulative weight function of edges in  $G$  so as to maximize the expected worker weight in the premise of thresholds constraints.

### THE PROBLEM ANALYSIS

It is difficult to answer  $(\varepsilon, \varphi)$ -CBPP problem both efficiently and effectively. For the efficiency problem, when a task is created, the platform should first access the road network  $G$  for enumerating all the paths that are satisfying the distance constraint. Since it is a NP-hard problem, we are not able to find the solution in polynomial time. In addition, the platform should use integral to compute the expected worker weight over each path, this part of cost is still high. Therefore, we can conclude that it is hard to answer  $(\varepsilon, \varphi)$ -CBPP problem in real-time. For the effectiveness problem, it is difficult to find a suitable cumulative weight function for each edge in the road network.

### III. THE PPVBF FRAMEWORK OVERVIEW

In this section, we present our framework PPVF(short for path prediction and verification based framework) to answer  $(\varepsilon, \varphi)$ -CBPP problem. Given the road network  $G$ , PPVF first constructs summary information for each edge in  $G$  according to the historical transaction records so as to approach the characters of their cumulative weight function. Based on the summarization results, we design the algorithm WDPS to select a group of “high quality” paths for every two vertex. Such paths are maintained in the path set  $\mathcal{KP}$  for providing high quality paths to workers when they are going to carry out a task.

To maintain the elements in  $\mathcal{KP}$  both effectively and efficiently, in this paper, we propose a novel index named PCR-Tree(short for Path-Compression-R)-Tree. As the basic, we propose a pre-processing algorithm named PCPS(short for Path Cover-Partition- Substitution). Our goal is to reduce the  $\mathcal{KP}$  scale and provide PCR-Tree with powerful filtering ability. The first phase of  $\mathcal{KP}$  scale reduction is called as *path cover*. Since some paths are covered by other paths, we are going to deleting such paths. The second phase of  $\mathcal{KP}$  scale reduction is called as *path substitution*. Since there exist multitudes of common subpaths among elements in  $\mathcal{KP}$ , we could use a few “tags” to substitute the original paths by effectively labeling these common subpaths. In such a way, the  $\mathcal{KP}$  scale could be further reduced. For the *path partition*, we partition the elements in  $\mathcal{KP}$  into a few sub-paths, use a group of MBRs with relatively small area to bound these paths. In this way, we could provide these paths with tight boundary, and enhance the filtering ability of PCR-Tree.

Based on the pre-processing results, we construct PCR-Tree. It is a R-tree based index, an MBR  $R$  bounding all these sub-paths is computed according to the root of the PCR-Tree and have an association with it. The construction of PCR-Tree is done by recursively partitioning each region into smaller subregions, which guarantees that the amount of children in each subregion is roughly the same and less than

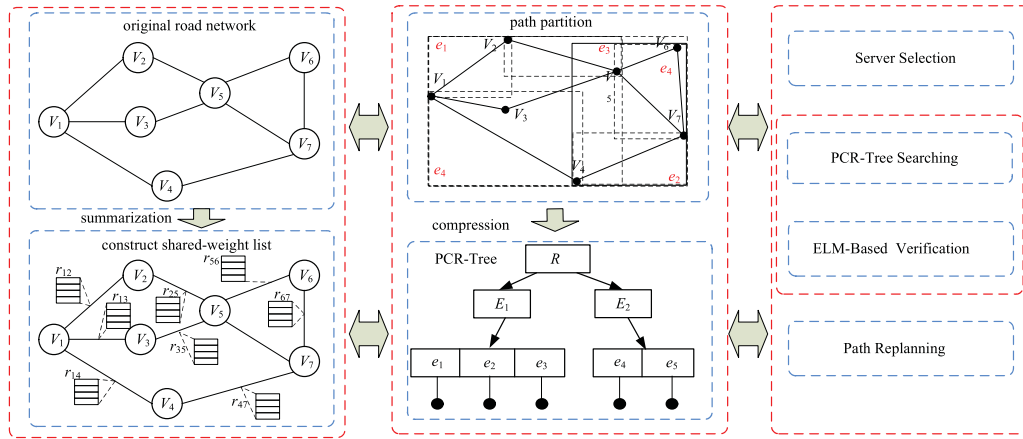


FIGURE 3. The Framework Overview.

the page size (eg., 4KB). Finally, we construct an inverted list for every leaf node  $e_f$  to record which sub-paths are contained in  $e_f$ . A group of bit-compression techniques to express the information maintained in PCR-Tree are proposed to further reduce the space cost. Accordingly, the overall space cost of PCR-Tree could be reduced a lot.

Once a task  $t$  is submitted, the platform should assign a worker  $w$  to  $t$  and plan a suitable path. To achieve this goal, the platform first submits a range query to the system, searches the workers contained in the query range. If there exist empty workers contained in the query region, we randomly select an empty worker for the task  $t$ . In addition, we search on the PCR-Tree for selecting the candidate paths. Finally, we utilize an ELM-based algorithm for verification. If there not exist empty worker contained in the query region, we invoke the algorithm PRP for finding a suitable worker, and incrementally re-planning the path.

#### IV. HIGH QUALITY PATH MAINTENANCE

In this section, we first discuss the summary information construction, and then discuss the high quality paths selection and pre-condition. Last of this section, we discuss the index PCR-Tree.

##### A. ROAD NETWORK SUMMARIZATION

As discussed in Section 3, it is difficult to construct a suitable cumulative weight function for evaluating worker weight. In this section, we address this challenge by computing the summary information of tasks/workers distribution via accessing the historical transaction record set. The summarization algorithm contains two phases which are *initialization* and *merge*.

Let  $HT$  be the historical transaction record set. In the initialization phase, we build a vector  $gv$  for each edge  $e \in G$ . Its role is to describe the tasks/workers distribution over  $e$ . To be more specific, given an edge  $e$  contained in  $G$ , we first partition  $e$  into  $\lceil \frac{|e|}{m} \rceil$  sub-edges. For each sub-edge  $e_i$ , we search on  $HT$  for finding the tasks/workers contained in its

impact region. Here, the impact region of  $e$  is  $\cup IR(x_i)(x_i \in e)$ . Assuming the tasks amount is  $\alpha_i$  and workers amount is  $\beta_i$ , we set  $gv[i]$  to  $\frac{\alpha_i}{\beta_i}$ . After accessing all the sub-edges,  $gv$  is initialized to  $\langle \frac{\alpha_1}{\beta_1}, \frac{\alpha_2}{\beta_2}, \dots, \frac{\alpha_{\lceil \frac{|e|}{m} \rceil}}{\beta_{\lceil \frac{|e|}{m} \rceil}} \rangle$ .

After initializing, the algorithm enters into the merge phase, where we merge the adjacent elements in  $gv$  together if their difference is smaller than a threshold  $\tau$ . Compared with the initialization result, we could use as small space cost as possible to describe the edge weight distribution function. Specifically, given the  $i$ -th element  $gv[i]$  in  $gv$ , if  $|gv[i] - gv[i - 1]|$  is smaller than a threshold  $\tau$ , we use the tuple  $\langle \frac{gv[i] + gv[i - 1]}{2}, cnt \rangle$  to substitute the original one. Here,  $cnt$  refers to the number of the elements which are merged together. For each edges, the above operations are repeating executed to construct vectors.

#### DISCUSSION

The distribution of workers/tasks is not only relate to their location but also time. For example, the amount of workers/tasks around an edge is usually large at eight o' clock in the morning, but small at eleven o' clock in the evening. Thus, in real applications, we should use a two-dimension vector to describe the relationship among workers/tasks distribution, namely location and time. The technique discussed in this section could be applied to construct the two-dimension vector. In the following, we will use an example to explain the details.

*Example 7 (Road Network Summarization):* Take an example in Fig. 4. Given the historical transaction record set  $R$ , we partition the records in  $R$  into 12 parts. Here, the records in  $R_1$  are generated from the time period 0:00pm to 2:00pm in each day, the records in  $R_i$  are generated from the time period  $i + 2 : 00pm$  to  $i + 4 : 00pm$  in each day. After partitioning, we compute the number of records contained in each subset, and merge the subsets together if their size difference is small. For each merging result, we compute the summary information for each edge in  $G$  according to the

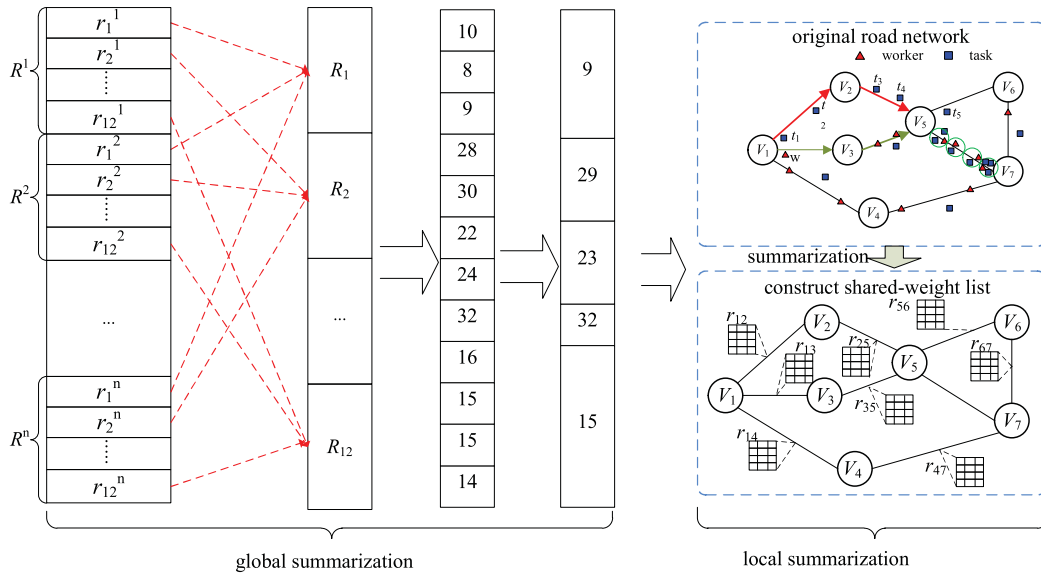


FIGURE 4. The Records Summarization.

merging results, where we compute how many tasks/workers are contained in the impact region of each edge respectively. Given an edge  $e_{57}$ (from  $V_5$  to  $V_7$ ), we partition it into 4 sub-edges  $\{e_{57}^1, e_{57}^2, e_{57}^3, e_{57}^4\}$ . For the sub-edge  $e_{57}^1$ , there exist 2 tasks and 1 worker in its impact region, we set  $gv[0]$  to 2. After initializing,  $gv$  is set to  $\{2, 1, 1, 3\}$ . Since  $\tau = 1$ , we could merge  $gv[0]$ ,  $gv[1]$  and  $gv[2]$  together. After merging, the vector turns to  $\{(1.3, 3), (3, 1)\}$ . Last of all, we could use a two-dimensional vector to express the summary information of an edge.

**B. HIGH QUALITY PATH SELECTION ALGORITHM**

Once the summary information of the road network  $G$  is constructed, we select a group of high quality paths for every two vertexes. In the following, we introduce the concept of **KPath** and discuss the high quality path selection algorithm.

*Definition 8 (KPath):* Given two vertexes  $v_i, v_j \in G$ , the path  $p_u(i, j)$  in the **KPath** set  $KP(i, j)$  should satisfy the following two conditions: (i) the inequality  $\frac{|p_u(i, j)|}{|\text{opt}(i, j)|} \leq \varepsilon$  should be satisfied; (ii) for all the paths satisfying condition (i), the elements in  $KP(i, j)$  are the  $k$  paths with highest scores. Here, the score of a path is computed based on Equation 7.

$$F(p) = EW(p) \times Div(p) \tag{7}$$

Here,  $EW(p)$  is the expected worker weight when  $w$  passes the path  $p$ , which is computed by Equation 8.  $\psi(l)$ , i.e.,  $0 < \psi(l) < 1$ , describes the distribution of task distance.  $Div(p)$  evaluates the diversity of  $p$  under  $KP(i, j)$ . As depicted in Fig 5,  $p_1, p_2, p_3$  are three paths with highest expected work weight, i.e.,  $EX(p_1) > EX(p_2) > EX(p_3)$ . Intuitively, we usually recommend  $p_1$  as the planning path since it has the highest expected worker weight. However, since  $p_1$  is computed according to the historical transaction record set, if the distribution of tasks and workers around  $p_1$  is significant

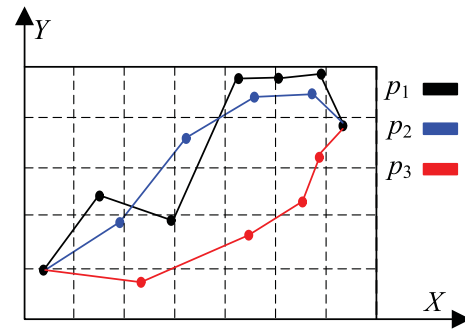


FIGURE 5. The Paths Diversity.

different from the current condition, the quality of recommendation result may be low. Hence we should select other paths from  $KP(i, j)$ . Among them,  $p_2$  is nearer to  $p_1$  than  $p_3$ . Although  $p_2$  has relatively higher expected worker weight, the tasks and workers distribution over  $p_2$  may be similar with that of  $p_1$ . In this case, it still may be a low quality recommendation result. In this case,  $p_3$  is more suitable to the set  $KP(i, j)$  than  $p_2$ .

$$EW(p)$$

$$= \sum_{i=1}^{\lceil \frac{|p|}{m} \rceil} \min(\sum EW(up_j) \int_{m^{(j-1)}}^{mj} \psi(l) + W(up_i), \varphi) \tag{8}$$

$$Div(p)$$

$$= \begin{cases} x = 1 & \forall p_i, \sum \text{euclidean}(c(up_i^u), c(up_j^{\lceil \frac{mu}{n} \rceil})) \geq \eta \\ & p_i \in \text{kPath} \\ y = 0 & \exists p_i, \sum \text{euclidean}(c(up_i^u), c(up_j^{\lceil \frac{mu}{n} \rceil})) < \eta \\ & p_i \in \text{kPath} \end{cases} \tag{9}$$

Before explaining the **kPath** construction, we first explain the path distance diversity evaluation. In this section, we use the distance among paths to evaluate the diversity of paths. Given two paths  $p_i, p_j$  with the unit path set  $\mathbf{UP}_i$  and  $\mathbf{UP}_j$ , if  $|\mathbf{UP}_i| = m$  and  $|\mathbf{UP}_j| = n (m > n)$ , the distance between  $p_i$  and  $p_j$  is  $\sum \text{euclidean}(c(up_i^u), c(up_j^{\lceil \frac{mu}{n} \rceil}))$ . Here,  $c(up_i^u)$  refers to center of the  $u$ -th unit path contained in the path  $p_i$ .  $\text{euclidean}(c(up_i^u), c(up_j^{\lceil \frac{mu}{n} \rceil}))$  refers to the Euclidean distance between  $c(up_i^u)$  and  $c(up_j^{\lceil \frac{mu}{n} \rceil})$ . Accordingly, the diversity of a path is computed based on Equation 9. The intuition behind is given two paths with the same starting location/destination location, if  $\mathbf{EW}(p_1) > \mathbf{EW}(p_2)$  and their distance is small, it is unnecessary to maintain  $p_2$ .

The **kPath** construction is depicted in Algorithm 1. We first compute the shortest path between two given vertexes and then enumerate all the paths that are satisfying the distance constraint condition. Subsequently, we sort them according to their expected worker weight. Based on the sorting results, we select the  $k$ -th highest scores as the final result. Since the process is simple, we skip the details for the limitation of space.

---

#### Algorithm 1 The kPath Construction Algorithm

---

**Input:** Road Network  $G$ , vertex  $v_i, v_j$   
**Output:** The kPath  $P$

- 1 Path  $p \leftarrow \text{shortestPath}(v_i, v_j)$ ;
- 2 Path Set  $PS \leftarrow \text{constraintPath}(v_i, v_j, \varepsilon)$ ;
- 3 **comSharedWeight**( $PS$ ), **sort**( $PS$ ) ;
- 4  $PS[0].div \leftarrow 1, P \leftarrow P \cup PS[0]$ ;
- 5 **while**  $i$  from 2 to  $|PS|$  **do**
- 6      $F(PS[i]) \leftarrow \text{comScore}(PS[i])$ ;
- 7     **if**  $|P| = k \vee \text{min}(P) < F(PS[i])$  **then**
- 8          $P \leftarrow P \cup PS[i]$ ;
- 9          $P \leftarrow P - \text{min}(P)$ ;
- 10    **if**  $|P| < k$  **then**
- 11          $P \leftarrow P \cup PS[i]$ ;
- 12 **return**;

---

### C. HIGH QUALITY PATH PRECONDITIONING ALGORITHM

Directly using **kPath** between every two vertexes to answer  $(\varepsilon, \varphi)$ -CBPP problem is effective but impractical for the high space cost. In addition, paths in  $\mathcal{KP}$  are usually expressed by irregular fold lines, it is difficult to effectively provide them with tight boundary when indexing them. In this section, our solution is inspired by the following three observations. From observation 4.1 and 4.3, we can conclude that if we could delete the redundant paths(or sub-paths) in  $\mathcal{KP}$ , the  $\mathcal{KP}$  scale could be effectively reduced. From observation 4.2, if paths are properly partitioned into sub-paths, we could provide elements in  $\mathcal{KP}$  with tight boundary. Based on the above observations, we propose the algorithm **PCPS**, which contains three steps: *covered path removing*, *path partition*

and *path substitution*. In the following, we first discuss the *path cover*.

*Observation 9:* Given two paths  $p_i$  and  $p_j$  in the path set  $\mathcal{KP}$ , if  $p_i$  is a sub-path of  $p_j$ ,  $\mathcal{KP}$  is equivalent to  $\mathcal{KP} - p_i$ .

*Observation 10:* Let  $p_i$  be a path in  $\mathcal{KP}$ ,  $p_i^1$  and  $p_i^2$  are two sub-paths of  $p_i$ , i.e.,  $p_i^1 \cup p_i^2 = p_i$  and  $p_i^1 \cap p_i^2 = \emptyset$ , if we construct three **MBRs**  $\text{MBR}(p_i)$ ,  $\text{MBR}(p_i^1)$  and  $\text{MBR}(p_i^2)$  for bounding them,  $|\text{MBR}(p_i)|$  must be larger than  $|\text{MBR}(p_i^1)| + |\text{MBR}(p_i^2)|$ .

*Observation 11:* Given two paths  $p_i$  and  $p_j$  in the path set  $\mathcal{KP}$ , if  $p_i \cap p_j \neq \emptyset$ ,  $|p_i| + |p_j| < |p_i \cap p_j|$ .

#### 1) THE COVERED PATH REMOVING

From observation 4.1, we can conclude that if a path  $p_i$  is a sub-path of  $p_j$  in  $\mathcal{KP}$ , it is unnecessary to maintain two paths respectively. In other words, if we can find a path set that could cover all the elements in  $\mathcal{KP}$ , we could delete them, and the overall space cost could be reduced a lot accordingly.

In order to achieve this goal, we construct the set  $V$ , where the element  $V_i \in V$  contains all the vertexes in  $p_i$ . Next, we are going to find another set  $V'$ . It should satisfy that, for each  $V_i$ , we could find a set  $V'_u \in V'$  containing  $V_i$ . In the following, we apply the greedy strategy to construct  $V'$ . To be more specifically, we sort all the elements in  $V$  according to their size. After sorting, we input the element with the largest size into  $V'$ . From then on, we access the other ones in  $V$ . For each element  $V_i$ , if it is contained by another element in  $V'$ , we ignore it. Otherwise, we insert it into  $V'$ . When all the ones in  $V$  are accessed, we use the ones in  $V'$  as the compression result.

---

#### Algorithm 2 The MPC Algorithm

---

**Input:** Path Set  $\mathcal{KP}$   
**Output:** The Compression Result  $\mathcal{KP}$

- 1 Set  $V \leftarrow \text{constructVSet}(\mathcal{KP})$ ;
- 2 **sort**( $V$ );
- 3 Set  $V' \leftarrow V[0]$ ;
- 4 **while**  $|V| \neq 0$  **do**
- 5     **if**  $V[0]$  is a subset of the elements in  $V'$  **then**
- 6          $V' \leftarrow V' \cup V[0]$ ;
- 7          $V \leftarrow V - V[0]$ ;
- 8  $\mathcal{KP} \leftarrow \text{update}(V)$ ;
- 9 **Return**;

---

#### 2) THE PATH SELF-ADAPTIVELY PARTITION

As will be reviewed Section 4.4, we use a group of minimal bounding rectangles (short for **MBRs**) as the boundaries of these high quality paths in  $\mathcal{KP}$ . In order to provide these paths with tight boundary, we partition them into a group of sub-paths. Accordingly, the corresponding **MBRs** area sum could be reduced a lot. In this section, we employ the greedy strategy for optimizing the partition. Our goal is to make the area sum of these **MBRs** as small as possible in the premise



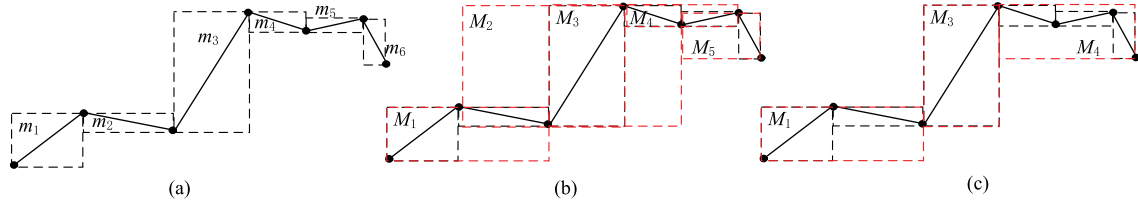


FIGURE 6. The Records Summarization.

that the partition amount of a path is less than a threshold  $\varepsilon|p|$ . Here,  $|p|$  is the vertexes amount contained in  $p$ , and  $0 < \varepsilon < 1$ .

**Algorithm 3** The Path Partition Algorithm

```

Input: Path  $p$ , Partition max size  $\varepsilon|p|$ 
Output: Partition Result  $\mathcal{P}$ 
1 MBR Set  $M_r, M_v$ , Num Set  $\Delta$ ;
2  $M_r[1] \leftarrow \mathbf{initMBR}(v_1, v_2)$ ;
3 for  $i$  from 2 to  $|e| - 1$  do
4    $M_r[i] \leftarrow \mathbf{initMBR}(v_i, v_{i+1})$ ;
5    $M_v[i - 1] \leftarrow \mathbf{mergeMBR}(M_r[i - 1], M_r[i])$ ;
6    $\Delta[i - 1] \leftarrow |M_r[i]| - |M_v[i + 1]| - |M_v[i]|$ ;
7 Min-Heap  $H \leftarrow \mathbf{initMinHeap}(\Delta)$ ;
8 while  $|H| \leq \varepsilon|e|$  do
9   Node  $\Delta[t] \leftarrow \mathbf{getRoot}(H)$ ;
10   $M_v[u] \leftarrow \mathbf{search}(\Delta[t])$ ;
11   $M_v[u - 1] \leftarrow \mathbf{mergeMBR}(M_v[u - 1], M_v[u])$ ;
12   $M_v[u + 1] \leftarrow \mathbf{mergeMBR}(M_v[u], M_v[u + 1])$ ;
13  delete( $M_u$ ), adjust( $H$ );
14  $\mathcal{P} \leftarrow \mathbf{partition}(p, H)$ ;
15 return  $r$ ;

```

As depicted in Algorithm 4, given a set of vertexes contained in the path  $p$ , we construct a set of MBRs, denoted by  $M_v$ , for every two adjacent vertexes. Based on  $M_v$ , we then construct another group of MBRs, denoted by  $M_r$ , for every two adjacent MBRs contained in  $M_v$ . The reason we construct  $M_v$  is to evaluate the area difference relationship among MBRs in  $M_v$  and  $M_r$ . Intuitively, if the area difference is small, the corresponding vertexes could be partitioned into the same sub-paths. Take an example in Fig.6(a), since  $|M_v[4]| - |M_r[4]| - |M_r[5]|$  is small, the vertexes  $v_4, v_5$  should be partitioned into the same sub-path. In this case, the corresponding MBR could also tightly bound the edges  $e_4$  and  $e_5$ . By contrast,  $|M_v[2]| - |M_r[2]| - |M_r[3]|$  is relatively large,  $v_2$  and  $v_3$  should belong to different sub-paths.

$$\Delta_i = |M_r[i]| - |M_v[i + 1]| - |M_v[i]| \quad (10)$$

After initializing the set  $M_v$  and  $M_r$ , we construct another set named  $\Delta$ . Here,  $\Delta_i \in \Delta$  is computed according to Equation 10, and  $M_r[i]$  is the  $i$ -th MBR contained in  $M_r$ . The elements in  $\Delta$  are sorted in ascending order, and maintained by a *min-heap*  $H$ . The reason we introduce  $H$  is to evaluate

which vertexes could be partitioned into the same sub-paths according to the root of  $H$ . To be more specifically, assuming  $R(H)$  is the root of  $H$ , its corresponding MBRs in  $M_r$  is  $M_r[i]$ , we first update  $M_r[i - 1]$  to  $M_r[i - 1] \cup M_r[i]$ , and  $M_r[i + 1]$  to  $M_r[i] \cup M_r[i + 1]$ . We then delete  $M_r[i]$ . Accordingly, we update  $H$ . From then on, we repeat the above operations to maintain  $H$ . When the size of  $H$  reduces to  $\varepsilon|p|$ , the algorithm is terminated.

Back to the example in Fig6. There are 6 vertexes contained in the path  $p$  and  $\varepsilon$  is 0.5. We should partition  $p$  into 3 sub-paths. We first construct the MBR set  $M_r, M_v$ , and initialize the min-heap  $H$  accordingly. From then on, we repeat line7 to line13 in Algorithm 4 to maintain  $H$ . When the size of  $H$  decreases to 3, the algorithm is terminated. At that moment, the path  $p$  is partitioned into  $\{v_1, v_2\}$ ,  $\{v_3\}$ , and  $\{v_4, v_5, \dots, v_6\}$ .

3) THE PATH SUBSTITUTION

From observation 4.3, if there exist common sub-paths among different paths in  $\mathcal{KP}$ , we could construct a group of “tags” for substituting the original paths. Take an example in Fig 7.  $p_1$  is constructed by  $\{v_1, v_3, v_5, v_6\}$ , and  $p_3$  is constructed by  $\{v_1, v_3, v_5, v_7\}$ .  $p_1 \cap p_3$  is  $\{v_1, v_3, v_5\}$ . If we use a tag  $V_1$  to express  $\{v_1, v_3, v_5\}$ ,  $p_1$  and  $p_3$  could be substituted by  $\{V_1, v_6\}$  and  $\{V_1, v_7\}$ .

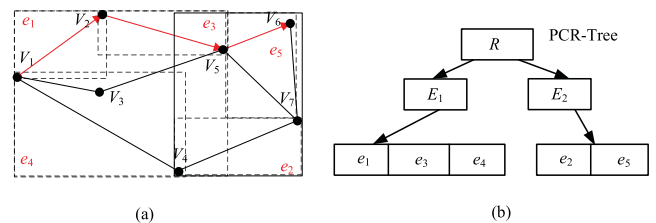


FIGURE 7. Searching On PCR-Tree( $k = 1$ ).

Based on this observation, our goal is to find and label a group of common sub-paths from  $\mathcal{KP}$ , then use the labelling results for substituting the original paths. In this paper, we propose a hash-based approach to achieve this goal. To be more specific, for each path  $p$  in  $\mathcal{KP}$ , we first enumerate all the sub-paths contained in  $p$ . We then initialize the hash table  $HT$ , where the enumerating result is mapped to a hash table  $HT$ . Assuming  $\mathbf{enum}(KP)$  contains all the enumerating results of all the elements in  $\mathcal{KP}$ , we set the size of the hash table  $HT$  as  $2 \times |\mathbf{enum}(KP)|$ . Furthermore, each element in

$HT$  is expressed by the tuple  $\langle p, cnt \rangle$ . Here,  $p$  refers to a sub-path,  $cnt$  counts how many paths in  $\mathcal{KP}$  have the common sub-path  $p$ .

Let  $sp$  be a sub-path of  $p$ ,  $HT$  be the hash table, and  $HF$  be the hash function. If  $HF(sp) = i$ , we access  $HF[i]$ . If  $HF[i].p$  equals to  $sp$ , we add  $HF[i].cnt$  to  $HF[i].cnt + 1$ . If  $HF[i]$  is null, and we set  $HF[i].p$  to  $sp$ , set  $HF[i].cnt$  to 1. In particular, if  $HF[i].p \neq sp$ , we adopt *linear detection method* for maintaining  $sp$ . After initializing, we are going to construct the common sub-paths set, and use these sub-paths for substituting the original paths. We first sort the elements in  $\mathcal{KP}$  according to their length in descending order. After sorting, we access the longest path  $p_{lst}$  in  $\mathcal{KP}$  for constructing the set  $\mathbf{cover}(p_{lst})$ . It contains two types of elements. The first type of elements are the vertexes contained in  $p_{lst}$ . The second type of elements are sub-paths in  $p_{lst}$  that are “shared” by the other paths. Intuitively, given a sub-path  $sp \in p_{lst}$ , if  $HF[HF(sp)].cnt > 1$ ,  $sp$  must be shared with other paths. Next, we access  $\mathbf{cover}(p_{lst})$ , and construct another set  $\mathbf{min}(p_{lst})$ , where the elements in  $\mathbf{min}(p_{lst})$  should satisfy the following three condition. Here,  $\mathbf{vertex}(p_{lst})$  refers to the vertex set of  $p_{lst}$ .

- for each two selected elements  $S_i$  and  $S_j$  in  $\mathbf{cover}(p_{lst})$ ,  $S_i \cap S_j = \emptyset$ ;
- $\mathbf{vertex}(\mathbf{min}(p_{lst})) = \mathbf{vertex}(p_{lst})$ .
- the size of  $\mathbf{min}(p_{lst})$  should be as small as possible.

In this paper, we use the key of greedy for constructing  $\mathbf{min}(p_{lst})$ . Since the algorithm is similar with one discussed in *path cover*, for the limitation of space, we skip the details. For the sub-path  $sp'$  that are not contained in  $\mathbf{min}(p_{lst})$ , we access the corresponding hash table  $HF[HF(sp')]$ , and set  $HF[HF(sp')].cnt$  to  $HF[HF(sp')].cnt - 1$ . We repeat the above operations to construct  $\mathbf{min}(p)$  for other path  $p$ . After accessing all the paths contained in  $\mathcal{KP}$ , we check hash table  $HT$  for finding which sub-paths are selected as the common sub-paths. For these ones, we associate them with an  $ID$ . Accordingly, the paths in  $\mathcal{KP}$  could be substituted by a group of sub-path ID information.

#### D. THE PCR-TREE

In this section, we propose a novel index named PCR-Tree for indexing high quality paths. It is a R-Tree based index. Given a node  $e$  in the PCR-Tree  $T$ , it contains two types of information, which are *location* and *sub-paths*. In this section, we use bit vectors to store these two types of information so as to reduce the space cost. For the location information maintenance, given a MBR  $R$  that bounds all the vertexes, we use a “virtual grid”  $VG(2^m \times 2^m)$  to partition  $R$ . In this way, we could use “cell ID” to approach the location of a node. Since the ID information of a cell could be expressed by a bits vector, we could use a  $m$ -bits vector to express the left-bottom (right-upper) coordinates of a MBR. Compared with using true coordinates to express the location of a MBR, the bit vector helps us reduce the space cost a lot.

We then study the path information maintenance. In this section, we use inverted-list to maintain sub-paths contained

in a node. Given a node  $e$ , the  $i$ -th sub-paths, i.e., denoted by  $e.l[i]$ , could be expressed by the tuple  $\langle gId, sId \rangle$ . Here,  $gId$  refers to the  $Id$  of a path  $p$ , and  $sId$  is the sequence number of  $p$ . In this section, we still use a  $\lceil \log u \rceil + \lceil \log v \rceil$ -bits vector to express a tuple. Here,  $u$  is the total number of paths in, and  $n$  is the maximal partition amount among all these paths. Assuming we select  $2^{20}$  high quality paths from the historical record set, and the partition amount is bounded by 10, we could use a 23-bit vector to express a tuple, and this part of space cost could be reduced a lot.

## V. INCREMENTAL PATH PLANNING

Recalling Section 3, when a task  $t$  is submitted, the platform should select a worker, and plan the path for  $t$ . In the following, we first discuss the worker selection.

### A. WORKER SELECTION ALGORITHM

In this section, we propose the algorithm DCWS (short for distance constraint based worker selection) for worker selection. The algorithm contains two steps, which are **searching** and **verification**. In the first step, we submit a range query  $r \times r$  with the center  $t.s$  to the platform, and then search the workers contained in the query range. From then on, algorithm enters into the second step.

---

#### Algorithm 4 The Worker Selection Algorithm

---

**Input:** Task  $t$ , Road NetWork  $G$   
**Output:** Worker  $S$

- 1 Worker  $S \leftarrow \mathbf{RangeQuery}(t.s, G)$ ;
- 2 **if**  $S = \emptyset$  **then**
- 3      $\perp$  return  $\emptyset$ ;
- 4 **sort**( $W$ );
- 5 **if**  $\mathbf{existEmptySet}(S \neq \emptyset)$  **then**
- 6     Worker  $S_0 \leftarrow \mathbf{getEmptySet}(W)$ ;
- 7      $s \leftarrow \mathbf{random}(S_0)$ ;
- 8     **CPS**( $w$ );
- 9      $\perp$  return  $w$ ;
- 10 **for**  $i$  from 1 to  $|S|$  **do**
- 11     **if**  $\mathbf{IncVer}(S[i]) = \mathbf{true}$  **then**
- 12          $\perp$  **IPP**( $S[i]$ ) return  $S[i]$ ;
- 13 return  $\emptyset$ ;

---

The second step is choosing the suitable worker. Recalling Section 3, the workers can be divided into three types, which are *full*, *empty* and *non-full*. In order to guarantee the “worker load balance”, we prior consider the empty workers in the query result set  $S$ . Let  $S_0$  be the empty workers contained in the query region. We randomly select a worker for the task  $t$ , and then invoke the CPS algorithm for planning the path (See Section 5.2). If there is no empty worker in the query result, we invoke the algorithm PRP for finding the suitable worker and incrementally re-plan the path.

## B. THE CPS ALGORITHM

Once the platform assigns worker  $w$  to the task  $t$ , it should search PCR-Tree to find the corresponding **kPath** according to the starting location(destination location) of the task  $t$ . Since **kPath** contains a group of paths, we design an ELM-based algorithm for further verification. In the following, we first discuss the PCR-Tree searching algorithm.

### 1) SEARCHING ON PCR-TREE

The searching algorithm is to find the corresponding set **kPath** according to the starting location(destination location) of the task  $t$ . In the following, we use observation 5.1 for pruning subtrees that do not contain qualifying paths, and observation 5.2 for finding leaf nodes that contain the final **kPath**.

*Observation 12:* For a task  $t$  with starting location  $t.s$  and destination location  $t.d$ , the subtree of an intermediate entry  $e$  can be pruned if  $t.s$  (or  $t.d$ ) does not intersect with **MBR**( $e$ ).

*Observation 13:* For a task  $t$  with starting position  $t.s$  and destination  $t.d$ , assuming  $t.s$  is contained in  $e_s$ ,  $t.d$  is contained in  $e_d$ , if  $e_s.tp \cap e_d.tp \neq \emptyset$ , the final  $e$ -path may be contained in  $e_s.tp \cap e_d.tp$ .

Based on the above observations, we formally discuss the **kPath** searching algorithm. Given the task  $t(v, u, s, d, p)$ , the search starts from the root, and eliminates its entries according to Observation 5.1. For each remaining entry, we retrieve its child node, and perform the above process recursively until a leaf node is reached. For the leaf nodes that are not able to be pruned by Observation 5.1, we use Observation 5.2 for further reducing the candidate leaf nodes scale. After the necessary nodes in the PCR-tree have been visited, we start the refinement step for processing scan. In this phase, the intersection results, i.e., a set of sub-paths are first grouped by their associated paths. For each path, the scanning is performed to check the detailed information. Last of all, we use the final result  $k$  as the candidate paths set. Example 5.1 shows the details of searching on PCR-Tree. For simplicity, we set  $k$  to 1.

*Example 14 (Searching On PCR-Tree):* Take an example in Fig 7. The PCR-Tree  $T$  maintains 8 paths. Given the task  $t(v, u, s, d, p)$ , we first search on  $T$  for finding the leaf nodes that contain  $t.s$  and  $t.d$ . Because  $t.s$  is contained in  $e_2$  and  $t.d$  is contained in  $e_4$ , we further access these two nodes for finding the final path. Since the paths corresponding to  $e_2$  is  $\{p_2, p_4, p_5\}$ , and the the paths corresponding to  $e_4$  is  $\{p_1, p_3, p_4\}$ , the final result is  $p_4$ .

### 2) CANDIDATE PATH VERIFICATION

Since the size of a **kPath** set is larger than 1, we should further access them for finding the final result. Usually, we select the path with the highest score as the planning path. However, in many cases, the distribution of tasks/workers in the current road network is different from the one in the historical transaction record set. In this case, we should compute the excepted worker weight of the other path, and

select the one with the highest weight as the final result. From Equation 8, we can conclude that it is costly to compute the excepted worker weight of a path. In this section, we propose a novel algorithm named EWP(ELM-based Worker Weight Prediction) to solve this problem.

ELM is an efficient machine learning based algorithm. Given the **kPath** set  $kp(i, j)$ , ELM is used for judging whether the current distribution of tasks(or workers) contained in  $IR(p_{max})$  is similar with the ones in the historical record set. Here,  $p_{max}$  refers to a path in  $kp(i, j)$  with highest score, and  $IR(p_{max})$  refers to the impact region of  $p_{max}$ . Intuitively, if the answer is yes,  $p_{max}$  can be used as the query result. Otherwise, we should further check the other candidates. In this case, the algorithm first constructs a feature vector according to the tasks/servers distribution around candidate paths, and then input the feature vector into an ELM-based classifier for verification. In the following, we first discuss the tasks/workers distribution summarization. Based on the summarization result, we explain the feature vector construction.

#### a: TASKS/WORKERS DISTRIBUTION SUMMARIZATION

Given a path  $p$ , we use the summary information of the tasks/workers distribution over  $p$  to evaluate the worker weight. At the beginning, we discuss the workers and tasks maintenance. Given the worker set  $W$ , we use a grid file  $\mathcal{G}$  with the resolution  $r \times r$  for indexing the location of these workers. In addition, we maintain the number of workers in each cell  $c$ , denoted by  $|c.s|$ . When a worker moves from the cell  $c_1$  to the cell  $c_2$ , we add  $|c_2.s|$  to  $|c_2.s| + 1$ , and minus  $|c_1.s|$  to  $|c_1.s| - 1$ . We use the same method to maintain the passengers in  $\mathcal{G}$ . For the limitation of space, we skip the details.

Let  $UP(p)$  be the unit paths contained in the path  $p$ . Similar with the algorithm discussed in Section 5.1, the summary information is also described by a vector  $sv(p)$  with the size  $|UP(p)|$ . In this paper, for each unit path  $up_i \in p$ , we submit a range query with the region  $IR(up_i)$  to the system for finding cells contained in (or intersected with)  $IR(up_i)$ . If a cell  $c$  is contained in  $IR(up_i)$ , we add  $up_i.w$  to  $up_i.w + |c.s|$ . If a cell  $c$  is intersected with  $IR(up_i)$ , we approach the number of workers contained in  $c'$  based on  $\frac{|c'.s|Area(c, q_i)}{r \times r}$ . We use the similar algorithm to compute the tasks contained in  $IR(up_i)$ . Accordingly, the  $i$ -th element in the vector is computed by Equation 11. We repeat the above operations to finish constructing  $sv(p)$ .

$$sv[i] = \sum_{u=0}^{u=|v|} \frac{|c_i.p|Area(c, q_i)}{r \times r} / \sum_{u=0}^{u=|v|} \frac{|c_i.s|Area(c, q_i)}{r \times r} \quad (11)$$

#### b: FEATURE VECTOR DISCUSSION

Let  $sv(p)$  be the summary information vector of a path  $p$ , we first study the relationship between worker weight and the distribution of the elements in  $sv(p)$ . And then, we extract a few characters from  $sv(p)$  for constructing the feature vector. To be more specifically, we use the tuple  $\langle sum, wsum, seg \rangle$  as the feature vector. The first element named  $V(sum)$  is

computed based on  $\sum sv(p)[i]$ . Intuitively, the higher the  $V(sum)$ , the more the tasks a worker may take. It is significantly to use  $V(sum)$  as an element of the feature vector.

$V(wsum)$  is computed by Equation 12. Compared with  $V(sum)$ , we use the function  $f(i)$  to describe the weight of elements in  $sv(p)$ . Here,  $f(i)$  is decreasing with the parameter  $i$ . Given two paths  $p_1$  and  $p_2$  with the summary information vectors  $\langle 3, 0.5, 1, 0.5 \rangle$  and  $\langle 0.5, 0.5, 1, 3 \rangle$ , although  $V_1(sum)$  equals to  $V_2(sum)$ , the workers under  $p_1$  may take more tasks than that of  $p_2$ . The reason is, the earlier the worker  $w$  takes tasks, the earlier the  $w$  finishes these tasks, and it has more chances for  $w$  to execute other tasks. Therefore,  $V(wsum)$  could more effectively evaluate the expected worker weight of a path.

$$V(wsum) = \sum f(i)sv[i] \quad 0 < c < 1 \quad (12)$$

$V(seg)$  is computed by Equation 13. Compared with  $V(wsum)$ , it could more effectively evaluate the number of tasks a worker may execute in many cases. Given two paths  $p_3$  and  $p_4$  with the vectors  $\langle 3, 0.1, 3, 0.1 \rangle$  and  $\langle 3, 3, 0.1, 0.1 \rangle$ , although  $V_1(sum) = V_2(sum)$  and  $V_1(wsum) < V_2(wsum)$ , the workers  $w$  over  $p_3$  may take more tasks than that of  $p_4$ . The reason is the worker  $w$  may turn to full when it passes the first unit path of  $p_3$  and  $p_4$ . In this case, although  $sv(p_4)[2] = 3$ , the worker  $w$  may have no ability to take other tasks after  $w$  passes the second unit path of  $p_4$ . By contrast, although  $sv(p_3)[2] = 0$ , it has no impact to the benefit of  $w$ . When the worker  $w$  arrives in the third unit path of  $p_3$ , some tasks of  $w$  may be finished, it could take new tasks with high probability since  $sv(p_3)[3] = 3$ .

$$V(seg) = \sum_{i=0}^{i=d} \max(\varphi, sv(p)[seg * i + 1], \dots, sv(p)[seg * (i + 1)]) \quad d = \frac{|sv(p)|}{seg} \quad (13)$$

### c: THE PATH SELECTION

In this section, we divide paths into five types, which are **High**, **rHigh**, **Median**, **rLow**, and **Low**(See Equation 14). Here,  $EX(p)$  refers to the expected work weight of a path. Given two vertexes  $v_i$  and  $v_j$  and their corresponding **kPath** set  $kp(i, j)$ , the role of path selection is if  $C(p_{max})$  under historical record set is no smaller than that of under current road network, we use  $p_{max}$  as the planning path. Otherwise, we should access the other elements in  $KP(i, j)$  for further evaluation. Here,  $p_{max}$  is the path with highest score in  $KP(i, j)$ .

In this section, we propose an **ELM**-based algorithm for evaluation. Since the classification speed of **ELM** is much faster than that of the other machine learning algorithm, it is suitable for classification in the high overload environment. To be more specifically, we access the path in the candidate set one by one, and end the algorithm when we find a suitable path. We first construct the feature vector for  $p_{max}$ , and then input it into the **ELM**-based classifier. After classifying, if the

path selection rule is satisfied, we use  $p_{max}$  as the final result. Otherwise, we go on accessing other paths. In particularly, if no path satisfies the path selection rule, we still use  $p_{max}$  as the planning result.

$$C(p) = \begin{cases} \text{high} & 0.8c \leq EX(p) \leq c \\ \text{r-high} & 0.6c \leq EX(p) \leq 0.8c \\ \text{median} & 0.4c \leq EX(p) \leq 0.6c \\ \text{r-low} & 0.2c \leq EX(p) \leq 0.4c \\ \text{low} & 0 \leq EX(p) \leq 0.2c \end{cases} \quad (14)$$

### d: THE CACHE-BASED PATH SELECTION

In many real applications, some regions could be regarded as “hot region”, where these regions are usually regarded as the starting/destination position. In this paper, we construct a hash-based cache  $HC$  for such regions and their corresponding high quality paths. Intuitively, if a user  $u$  submits a requirement to the platform with the starting/destination position  $\langle u.s, u.d \rangle$ , and  $\langle u.s, u.d \rangle$  is maintained by  $HC$ , we could use the corresponding path for answer the query. In this case, we could avoid searching on PCR-Tree and the classification.

To be more specifically, we construct a hash table  $H$  for these hot regions. The elements in each bucket  $b \in H$  are maintained by an inverted-list. In this section, we use the tuple  $t \langle ED(s, d), p, cnt \rangle$  to express the starting/destination position  $\langle s, d \rangle$  and their corresponding path  $p$ . Here,  $\langle ED(s, d) \rangle$  refers to the encoding result of the tuple  $\langle s, d \rangle$ .  $cnt$  records how many times  $t$  is accessed.

when a user submit a requirement to the platform with the starting/destination position  $\langle u.s, u.d \rangle$ , we compute the hash value according to  $F(E(u.s), E(u.d))$ . Here,  $F$  refers to the hash function. Let  $F(E(u.s), E(u.d))$  be  $v$ . If  $H(v)$  is not empty, we check whether  $\langle E(u.s), E(u.d) \rangle$  is contained in  $H$  according to  $E(u.s, u.d)$ . If so, we use the corresponding path as the final result. Otherwise, we access PCR-Tree for searching. In addition, we construct the tuple  $\langle ED(u.s, u.d), p, 1 \rangle$ , and insert it into  $H$ .

A natural question is if an element in  $H$  is maintained for a long time, the distribution of workers/tasks may turn to difference. In addition, if too many elements are contained in  $H$ , the accessing cost may be high. In this paper, we periodically update  $H$ . When this operation is invoked, we access all the elements in  $H$ . For each of them  $t$ , if  $t.cnt \leq \delta$ , we delete it from  $H$ . Otherwise, we invoked the algorithm **CPS** for update the path.

### 3) PATH RE-PLANNING ALGORITHM

If we cannot find empty workers in the query region, we should select a non-full worker to execute the task  $t$ . For each non-full worker  $w$  contained in the query region, we should judge whether  $w$  is still a  $(\varepsilon, \varphi)$ -worker once  $t$  is assigned to  $w$ . If so, we assign  $w$  to  $t$ , and then re-plan the path for  $w$ .

**Constraint 15 (Distance Constraint):** Given the task  $t_i$  taken by the worker  $w$ , if its corresponding pre-planning path is  $p_i$ ,  $p_i$  should satisfy  $\frac{|sp_i| - |SP(p_i)|}{|sp_i|} \leq \varepsilon$ .

**Algorithm 5** The Incremental Path Re-Planning Algorithm

---

**Input:** Road Network  $\mathcal{G}$ , Worker  $w$ , Task  $t$   
**Output:** Worker  $w$

- 1 Vertex  $v \leftarrow (\text{findNearest})(s.p, t.s)$ ; **if**  $\angle t_s v v' > 90^\circ$  **then**
- 2     Path  $p_1 \leftarrow \text{CPR}(t_s, v)$ ;
- 3     Path  $p_2 \leftarrow \text{CPR}(v, v')$ ;
- 4      $s.P \leftarrow s.P \cup p_1 \cup p_2 - p_{vv'}$ ;
- 5 **if**  $\angle t_s v v' < 90^\circ$  **then**
- 6     Path  $p_1 \leftarrow \text{CPR}(t_s, v)$ ;
- 7     Path  $p_2 \leftarrow \text{CPR}(v, v')$ ;
- 8      $s.P \leftarrow s.P \cup p_1 \cup p_2 - p_{vv'}$ ;
- 9 **DisConstraint**();
- 10 **return** ;

---

*Constraint 16 (Benefit Constraint):* Given the task set  $\mathcal{T}(w)$  of the worker  $w$  and a new task  $t'$ , once  $t'$  is assigned to  $w$ , the total cost of task set  $\mathcal{T}'(w)$  (i.e.,  $\text{cost}(\mathcal{T}'(w))$ ) should be more than  $\text{cost}(\mathcal{T}(w))$ .

To be more specifically, given the starting/destination points set  $\text{SU}(w)$  of the worker  $w$  and the task  $t$ , we first search on  $\text{SU}(w)$  for finding the point  $p_i$  in  $\text{SU}(w)$  that is nearest to  $t_s$ . If  $\angle t_s p_i p_{i+1} > 90^\circ$ , we invoke the CPS algorithm to plan the path that passes the vertexes  $p_{i-1}$ ,  $t_s$  and  $p_i$ . Otherwise, we invoke the CPS algorithm to re-plan the path. Next, we repeat the above operations to plan path for  $t_d$ . After re-planning, we update the path set according to the re-planning results, and check whether the new path of  $w$  could satisfy Constraint 1. If so, we assign  $w$  to  $t$ . Otherwise, we rollback the re-planned path to the original version.

**VI. EXPERIMENTAL EVALUATION**

In this section, we conduct extensive experiments to demonstrate the efficiency of PPVF. The experiments are based on three real datasets. In the following, we first explain the settings of our experiments, and then report our findings.

**A. EXPERIMENTAL SETTING**

## 1) DATA SET

In total, three real datasets are used in our experiments, including BeiJing, California and Texas. BeiJing refers to the road network of Beijing in China. In order expressed the distribution of passengers/workers over  $G$ , we first partition the whole space via a grid file with the resolution  $\frac{\text{len}}{r} \times \frac{\text{width}}{r}$ , where  $\text{len}$  and  $\text{width}$  refers to the length/width of Beijing. After partitioning, we randomly generate a group of passengers/workers for each cell, i.e., (5000 workers and 10000 tasks for defaults) In order to simulate the changing of passengers/workers distribution, we further randomly generate a few passengers/workers for parts of cells. In addition, we delete a few passengers/workers for other cells per period. we use the similar method to construct tasks/workers sets

under California and Texas respectively. The details of these three dataset could be seen from Table 2.

**TABLE 2.** Dataset description.

Name	Nodes	Edges	Description
BEIJING	171,504	433,391	Road network of Beijing
CALIFORNIA	1,965,206	2,766,607	Road network of California
TEXAS	1,379,917	1,921,660	Road network of Texas

## 2) COMPARISONS, METRICS AND PARAMETERS SETTING

we compared the results of our framework named PPVF with two other approaches: TREE and APART. When implementing, for one thing, we randomly select a few passengers for tasks construction. Once it is assigned to a driver, we delete it from the current cell. In addition, we randomly construct a group of new tasks for some cells in the grid file.

In our experiments, we measure the following metrics by varying different parameters of the system, which are *service rate* and *response time*. Here, *service rate* refers to the percentage of requests that were completed. *response time* refers to the time a request is matched by a driver. We consider three parameters in our study. The parameter settings are listed in Table 3 with the default values bolded. All the algorithms are implemented with C++, and all the experiments are conducted on an CPU i7 with 32GB memory, running Microsoft Windows 7.

**TABLE 3.** Parameter settings.

Parameter	value
Total Passengers	1000, 2000, 5000, 10000, 20000
Max Passengers ( $\varphi$ )	2, 3, 4, 5, 6
Max Allowed Detour ( $\varepsilon$ )	0.2, 0.4, 0.6, 0.8, 1

**B. CLASSIFICATION EVALUATION**

## 1) CLASSIFICATION EVALUATION

In this subsection, we are going to evaluate the effect of the classifiers based on the ELM, DNN, and SVM under different data sets, i.e., called as ELM-classifier, DNN-classifier, and SVM-classifier respectively. First of all, we firstly evaluate the training time against different training set. We find that the training time of ELM-classifier is much smaller than that of other two classifiers.

For the accuracy of these three classifiers, we count the “False Positive” (and “False Negative”) times when we run the partition algorithm on the three data set, and use the accuracy ratio as the final result. We find that the accuracy of ELM-classifier is roughly as the same as that of SVM-classifier and DNN-classifier. Most important of all, we find that ELM-classifier is also advantageous in terms of classification efficiency. The reason behind is it spent less time than the other two classifiers.

In summary, ELM-classifier performs better than both SVM-classifier and DNN-classifier due to its better training

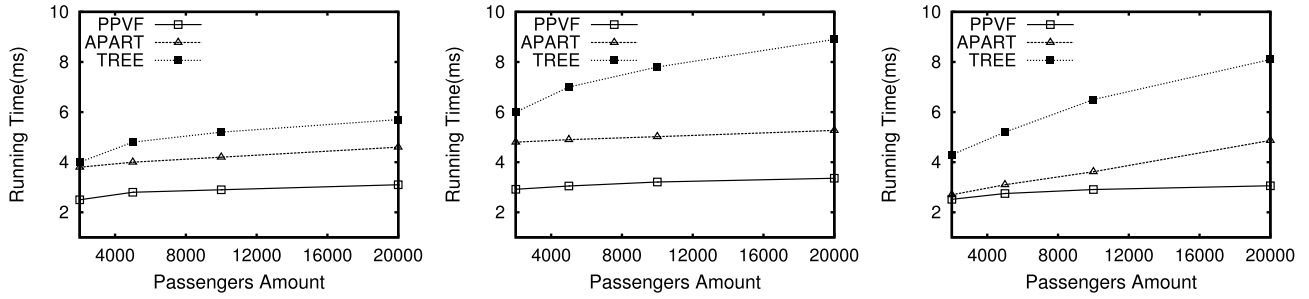


FIGURE 8. Running time comparison under different passengers amount.

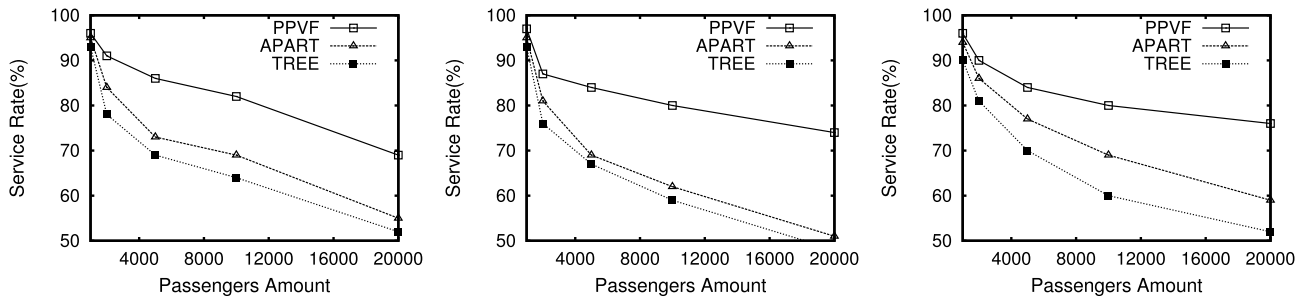


FIGURE 9. Service ratio comparison under different passengers amount.

TABLE 4. Training time(s).

DataSet	ELM	SVM	DNN
BEIJING	4	17.7	19.6
CALIFORNIA	5.9	21.2	24.8
TEXAS	7	30.4	39.2

TABLE 5. Classification time(s).

DataSet	ELM	SVM	DNN
BEIJING	17	366	303
CALIFORNIA	23	412	379
TEXAS	21	389	393

efficiency and classification efficiency as well. We want to highlight that, under real-time taxi-calling application, the classification efficiency is vital important since the algorithm should satisfy the real-time requirement of users, and ELM-classifier is suitable for classifying under such environment.

## 2) COMPRESSION EVALUATION

Next, we are going to evaluate the effect of the compression under these three datasets. As is depicted in table 6, we find that the larger the scale of road network, the higher the compression ratio. When we use 10% of the vertexes in each data set, the compression ratio is only 21%. When we use the whole dataset for evaluation, the compression ratio is 10%. The reason is, for one thing, we remove all the covered paths. For another, the path substitution helps us further reduce the path scale.

TABLE 6. Compression evaluation.

DataSet	10%	30%	50%	70%	100% ( $\times  D $ )
BEIJING	21.8	11.9	9.3	9.9	10.1
CALIFORNIA	23.6	14.3	12.3	11.9	10.4
TEXAS	22.9	16.9	12.3	11.7	9.8

## C. ALGORITHM PERFORMANCE

In this section, we compare our proposed framework PPVF with that of TREE and APART. We first evaluate the impact of passengers amount to the algorithm performance. We set the parameter  $\varphi$  to 4 and  $\varepsilon$  to 0.2. As is depicted in Fig 8, PPVF performs best of all. The reason behind is, for one thing, we use a hash-based cache to maintain a group of “hot requirements”. In this way, we could avoid searching on PCR-Tree and classification. More important, it is insensitive to the scale of both workers/passengers amount. For another, the algorithm CPS is very efficient, which helps us quickly find the “high quality” paths.

For the service ratio, as is depicted in Fig 9, PPVF always select high quality paths for the workers. Intuitively, the higher quality of paths, the more tasks are taken, and the higher the service ratio. With the increasing of the passengers amounts, the service rate under PPVF decreases much slowly than that of TREE and APART. Therefore, we can conclude that PPVF could provide workers with as high quality as service.

We then evaluate the impact of parameter  $\varphi$  to the algorithm performance. We set the parameter passengers amount to 10000. In addition, we set  $\varepsilon$  to 4. As is depicted in Fig 11,

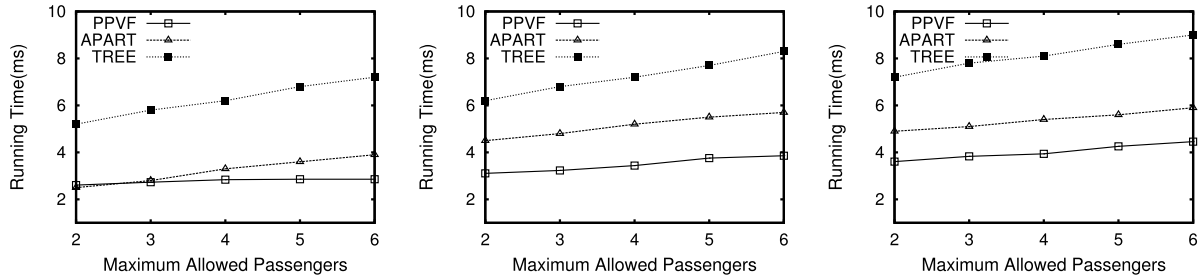


FIGURE 10. Running time comparison under different  $\varphi$ .

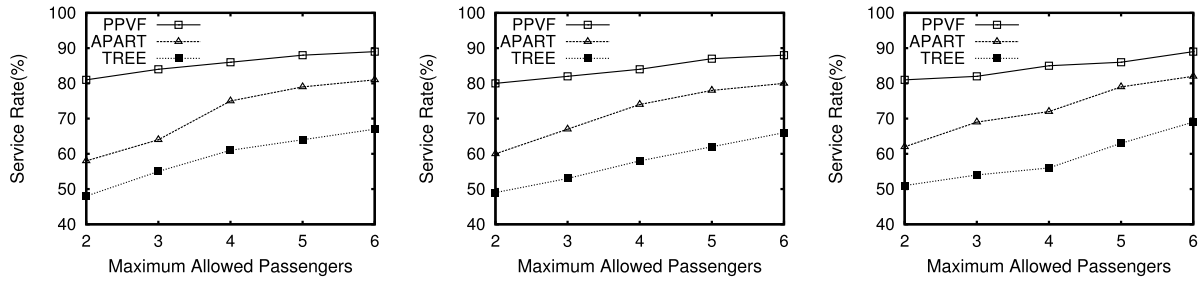


FIGURE 11. Service ratio comparison under different  $\varphi$ .

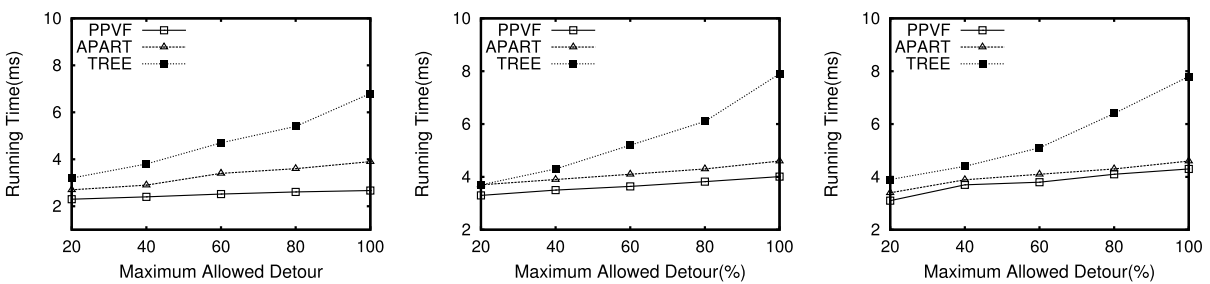


FIGURE 12. Running time comparison under different  $\epsilon$ .

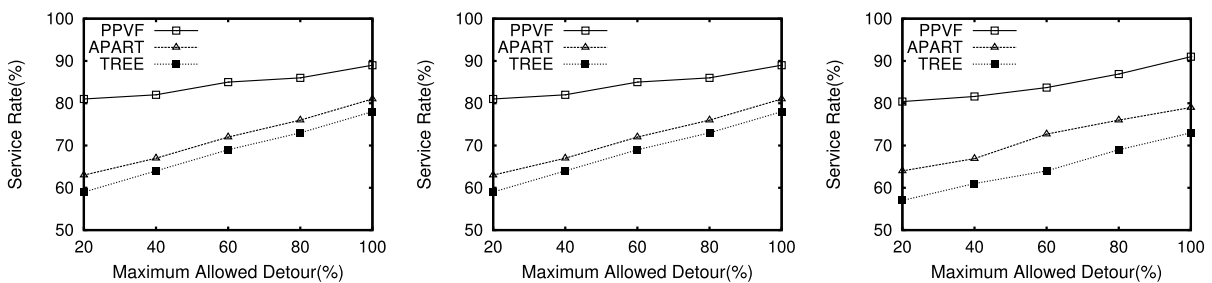


FIGURE 13. Service ratio comparison under different  $\epsilon$ .

PPVF performs best of all. With the increasing of the parameter  $\varphi$ , we could find that the service ratio under PPVF increases much faster than that of TREE and APART before  $\varphi$  achieves to 4. When  $\varphi$  achieves to 6, the service rate under PPVF is similar with that of under 4. The reason is when  $\varphi$  is large, most of tasks could be taken.

For the response time, from Fig 10, we find that PPVF performs slightly better than the other two ones (only 66% of the previous algorithms). The reason is PCR-Tree could

provide us with a powerful filtering ability for one thing. For another, the classifying speed of ELM is much faster than the others. Most important of all, we could use our proposed hash-based cache to find high quality paths under  $O(1)$  computational time.

Last of all, we evaluate the impact of parameter  $\epsilon$  to the algorithm performance. We set the parameter passengers amount to 10000. In addition, we set  $\varphi$  to 4. As is depicted in Fig 12 and Fig 13, PPVF still performs best of all. With the

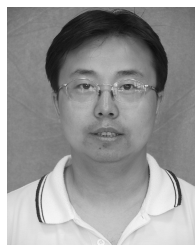
increasing of the parameter  $\varepsilon$ , the service ratio under **APART** and **TREE** increases faster than that of **PPVF**. The reason is the higher the  $\varepsilon$ , the more the tasks could be taken by workers. By contrast, **PPVF** could provide workers with high quality paths, it has the ability to process, as much as possible, tasks in real-time. Therefore, **PPVF** is stable which help workers to obtain high revenue in the premise of low  $\varepsilon$ .

## VII. CONCLUSION

In this paper, we propose a novel framework named **PPVF** for supporting path planning over carpooling. Different from existing works, we focus on planning high quality paths for workers to meet both passengers/workers' convenience and economic benefits. The algorithm **HQPS** for selecting high quality paths among vertexes is proposed firstly, and then we propose a novel index named **PCR-Tree** to maintain these paths. Furthermore, we propose a *prediction-and-verification*-based algorithm for path planning. We conduct extensive experiments to evaluate the performance of **PPVF**. The results demonstrate the superior performance of **PPVF**.

## REFERENCES

- [1] Y. Tong et al., "Flexible online task assignment in real-time spatial data," *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1334–1345, 2017.
- [2] M. Asghari, D. Deng, C. Shahabi, U. Demiryurek, and Y. Li, "Price-aware real-time ride-sharing at scale: An auction-based approach," in *Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst. (GIS)*, Burlingame, CA, USA, Oct./Nov. 2016, pp. 3:1–3:10.
- [3] N. Ta, G. Li, T. Zhao, J. Feng, H. Ma, and Z. Gong, "An efficient ride-sharing framework for maximizing shared route," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 2, pp. 551–565, Feb. 2017.
- [4] M. Ota, H. Vo, C. Silva, and J. Freire, "A scalable approach for data-driven taxi ride-sharing simulation," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Oct. 2015, pp. 888–897.
- [5] B. Cici, A. Markopoulou, and N. Laoutaris, "Designing an on-line ride-sharing system," in *Proc. 23rd SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst. (GIS)*, 2015, pp. 60:1–60:4.
- [6] D. Pelzer, J. Xiao, D. Zehe, M. Lees, A. Knoll, and H. Aydt, "A partition-based match making algorithm for dynamic ridesharing," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 5, pp. 2587–2598, May 2015.
- [7] E. Kamar and E. Horvitz, "Collaboration and shared plans in the open world: Studies of ridesharing," in *Proc. 21st Int. Joint Conf. Artif. Intell. (IJCAI)*, 2009, pp. 187–194.
- [8] A. Kleiner, B. Nebel, and V. A. Ziparo, "A mechanism for dynamic ride sharing based on parallel auctions," in *Proc. IJCAI*, 2011, pp. 266–272.
- [9] L. Kazemi and C. Shahabi, "Geocrowd: Enabling query answering with spatial crowdsourcing," in *Proc. SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, 2012, pp. 189–198, doi: [10.1145/2424321.2424346](https://doi.org/10.1145/2424321.2424346).
- [10] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen, "Online mobile micro-task allocation in spatial crowdsourcing," in *Proc. 32nd Int. Conf. Data Eng.*, May 2016, pp. 49–60, doi: [10.1109/ICDE.2016.7498228](https://doi.org/10.1109/ICDE.2016.7498228).
- [11] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu, "Online minimum matching in real-time spatial data: Experiments and analysis," *Proc. VLDB Endowment*, vol. 9, no. 12, pp. 1053–1064, 2016, doi: [10.14778/2994509.2994523](https://doi.org/10.14778/2994509.2994523).
- [12] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, 1959.
- [13] R. Bellman, "On a routing problem," *Quart. Appl. Math.*, vol. 16, no. 1, pp. 87–90, 1958.
- [14] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, pp. 344–348, 1962.
- [15] J. Fakcharoenphol and S. Rao, "Planar graphs, negative weight edges, shortest paths, and near linear time," *J. Comput. Syst. Sci.*, vol. 72, no. 5, pp. 868–889, 2006.
- [16] P. N. Klein, S. Mozes, and O. Weimann, "Shortest paths in directed planar graphs with negative lengths: A linear-space  $O(n \log^2 n)$ -time algorithm," *ACM Trans. Algorithms*, vol. 6, pp. 1–10, Jan. 2010.
- [17] C. Wulff-Nilsen, "Constant time distance queries in planar unweighted graphs with subquadratic preprocessing time," *Comput. Geometry*, vol. 46, no. 7, pp. 831–838, 2013.
- [18] M. Furuhashi, M. Dessouky, F. Ordóñez, M.-E. Brunet, X. Wang, and S. Koenig, "Ridesharing: The state-of-the-art and future directions," *Transp. Res. B. Methodol.*, vol. 57, pp. 28–46, Nov. 2013.
- [19] P. Santi, G. Resta, M. Szell, S. Sobolevsky, S. H. Strogatz, and C. Ratti, "Quantifying the benefits of vehicle pooling with shareability networks," *Proc. Nat. Acad. Sci. USA*, vol. 111, no. 37, pp. 13290–13294, 2014.
- [20] B. Cici, A. Markopoulou, E. Frias-Martinez, and N. Laoutaris, "Assessing the potential of ride-sharing using mobile and social data: A tale of four cities," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2014, pp. 201–211.
- [21] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Manag. Sci.*, vol. 6, no. 1, pp. 80–91, Oct. 1959.
- [22] Y. Li, D. Deng, U. Demiryurek, C. Shahabi, and S. Ravada, "Towards fast and accurate solutions to vehicle routing in a large-scale and dynamic environment," in *Proc. Int. Symp. Spatial Temporal Databases*. Springer, 2015, pp. 119–136.
- [23] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *Proc. IEEE 29th Int. Conf. Data Eng. (ICDE)*, Apr. 2013, pp. 410–421.
- [24] Y. Huang, F. Bastani, R. Jin, and X. S. Wang, "Large scale real-time ridesharing with service guarantee on road networks," *Proc. VLDB Endowment*, vol. 7, no. 14, pp. 2017–2028, 2014.
- [25] D. O. Santos and E. C. Xavier, "Dynamic taxi and ridesharing: A framework and heuristics for the optimization problem," in *Proc. 23rd Int. Joint Conf. Artif. Intell. (IJCAI)*, Beijing, China, Aug. 2013, pp. 2885–2891.
- [26] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: A new learning scheme of feedforward neural networks," in *Proc. Int. Symp. Neural Netw.*, vol. 2, Jul. 2004, pp. 985–990.
- [27] G.-B. Huang, X. Ding, and H. Zhou, "Optimization method based extreme learning machine for classification," *Neurocomputing*, vol. 74, pp. 155–163, Dec. 2010.
- [28] C. Deng, B. Wang, W. Lin, G.-B. Huang, and B. Zhao, "Effective visual tracking by pairwise metric learning," *Neurocomputing*, vol. 261, pp. 266–275, Oct. 2017.
- [29] A. Lendasse, Q. He, Y. Miche, and G.-B. Huang, "Advances in extreme learning machines (ELM2013)," *Neurocomputing*, vol. 149, pp. 158–159, Feb. 2017.
- [30] S. Wang, C. Deng, W. Lin, G.-B. Huang, and B. Zhao, "NMF-based image quality assessment using extreme learning machine," *IEEE Trans. Cybern.*, vol. 47, no. 1, pp. 232–243, Jan. 2017.
- [31] G. Caruana, M. Li, and M. Qi, "A MapReduce based parallel SVM for large scale spam filtering," in *Proc. Fuzzy Syst. Knowl. Discovery*, 2011, pp. 2659–2662.



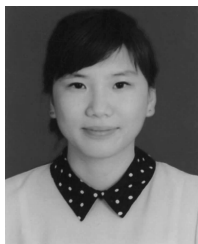
**BIN WANG** received the Ph.D. degree in computer science from Northeastern University, in 2008.

He is currently an Associate Professor with the Computer System Institute, Northeastern University. His research interests include design and analysis of algorithms, queries processing over streaming data, and distributed systems.

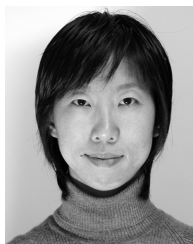


**RUI ZHU** received the M.Sc. degree in computer science from the Department of Computer Science, Northeastern University, China, in 2008, and the Ph.D. degree in computer science from Northeastern University, in 2017. He is currently a Lecturer with Shenyang Aerospace University. His research interests include design and analysis of algorithms, databases, data quality, and distributed systems.





**SITING ZHANG** received the bachelor's degree from Northeastern University, in 2017, where she is currently pursuing the master's degree with the Computer System Institute. Her research interests include trajectory data processing and mining.



**XIAOCHUN YANG** received the Ph.D. degree in computer science from Northeastern University, China, in 2001, where she is currently a Professor with the Department of Computer Science. Her research interests include data quality and data privacy. She is a member of the ACM, the IEEE Computer Society, and a Senior Member of the CCF.



**ZHENG ZHAO** received the bachelor's degree in computer science from Northeastern University, in 2017, where he is currently pursuing the master's degree. His research interest includes image understanding.



**GUOREN WANG** received the Ph.D. degree from the Department of Computer Science, Northeastern University, China, in 1996. He is currently a Professor with the School of Computer Science, Beijing Institute of Technology. His research interests include XML data management, and query processing and optimization.

...