IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 1, 2022, accepted April 5, 2022, date of publication April 13, 2022, date of current version April 28, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3167033

$\mathcal{E}$-Ride: An Adaptive Event-Driven Windowed Matching Framework in Ridesharing

HAN WU, YU CHEN, LIPING WANG, AND GUOJIE MA
School of Software Engineering, East China Normal University, Shanghai 200062, China
Corresponding author: Liping Wang (lipingwang@sei.ecnu.edu.cn)

ABSTRACT Ridesharing services aim at reducing the users' travel cost and optimizing the drivers' routes to satisfy passengers' expected maximum matching times in practice request dispatching. Existing works can be roughly classified into two types, i.e., online-based and batch-based methods, in which the former mainly focus on responding quickly to the requests and the latter focuses on enumerating request combinations meticulously to improve the service quality. However, online-based methods perform poorly in terms of service quality due to the neglect of the sharing relationship between requests, while batch-based methods fail on efficiency. None of these works can smoothly balance the service quality and matching time cost since the matching window is not sufficiently explored or even neglected. To cope with this problem, we propose a novel framework $\mathcal{E}$-Ride, which comprehensively leverages the matching time window based on the event model. Specifically, an adaptive windowed matching algorithm is proposed to adaptively consider personalized matching time and provide a matching solution with higher service rates at lower latencies. Besides, we maintain the request groups through a mixed graph and further integrate the subsequent arrival requests to optimize the matching results, which can scale to or satisfy online use demands. The extensive experimental results demonstrate the efficiency and effectiveness of our proposed method.

INDEX TERMS Ridesharing, windowed matching, graph maintenance, KL-UCB.

I. INTRODUCTION

Ridesharing services have made a significant contribution to modern transportation, facilitating the relief of traffic pressure and reduction of exhaust pollution [1], [2]. In urban transportation, ridesharing service provides a travel mode between public transit and the cab service, balancing the charging cost and travel convenience. So, it has become an essential travel option in their daily lives. In the ridesharing services, passengers are willing to accept a limited detour to share available seats with other passengers in exchange for a discount. Therefore, the existing ridesharing service providers, such as Didi [3] and Uber [4], continuously strive to improve service quality for their users by minimizing total travel cost [5]–[8] or maximizing the platform service rates [9], [10].

In reality, passengers usually hold specific requirements for the *response time* of a ridesharing system and are eager for a high-quality matching to save the travel cost. In general, requests willing to wait longer tend to be better matched. Therefore, a sound ridesharing system requires a trade-off between the response time and the service quality. Existing widely-applied request dispatching methods can be roughly classified into two types, i.e., online-based mode and batch-based mode.

In online-based methods [5], [6], [11], [12], the requests are assigned to the candidate vehicles with an updated route sequentially in chronological order. The online-based methods have widely adopted *insertion* [13] operator for the route planning. The *insertion* method plans a locally optimal route for the vehicle in linear time by inserting the source and destination of the request into the vehicle's route, without reordering waypoints among the original route (as shown in Figure 1). While in batch-based methods [14]–[17], the requests are usually grouped over a fixed time window and then executed once with the specific matching algorithm between the request group and candidate vehicles (i.e., bipartite graph matching, linear programming [14]). The state-of-the-art online-based methods [13], [18] are efficient in response time, which benefits from its linear time complexity. Although batch-based algorithms tend to have higher service quality by meticulously enumerating feasible request groups, it takes more time for computation. Moreover, in the batch-based algorithm, the request that arrives closer to the

The associate editor coordinating the review of this manuscript and approving it for publication was Rashid Mehmood.

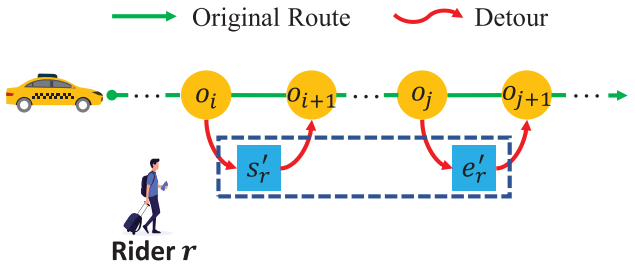


FIGURE 1. An illustration example of insertion operator.

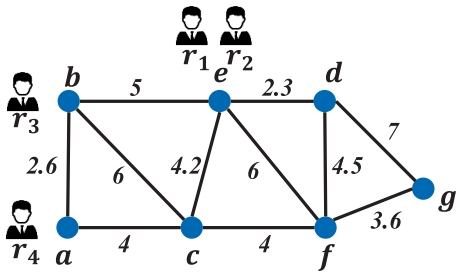


FIGURE 2. A motivation example.

batch trigger time will be dispatched immediately, but such just-arriving requests are willing to wait for more time to get a better matching result.

If it is feasible to plan a route that serves a set of requests simultaneously, then we call these requests are *shareable*. In this paper, we implement a fine-grained realization of request matching based on the concept of matching windows for better service quality. We illustrate our motivation with the following example.

Example 1: In the road network consisting of nodes $a \sim g$ shown in Figure 2, there are four online arrivals requests $r_1 \sim r_4$. The details of the requests are shown in Table 1. Suppose that the vehicles have enough capacity to accommodate three requests simultaneously, while it takes one unit of time to travel one unit of distance on the road network. If the request dispatching framework adopts the batch mode and the processing interval $T = 3$, since requests r_1, r_2 and r_3 are shared to save 11 units of total travel distance (let $cost(R)$ denote the shortest travel distance to serve all requests $r \in R$, then the saved travel distance is denoted as $\sum_{r \in R} cost(\{r\}) - cost(R)$), so the dispatching framework will dispatch r_1, r_2 and r_3 to the same vehicle.

But the request r_3 has just been released on the platform, and in fact, r_3 would like to wait for some more time to get a better match rather than leaving immediately. When r_4 is released to the platform, we can get a better assignment scheme with groups of $\{r_1, r_2\}$ and $\{r_3, r_4\}$, which saves 16 units of total travel distance in sum.

Specifically, in this paper, we construct an edge for any two shareable requests based on the online arrival requests as nodes to obtain a *dynamic shareability graph*, which helps prune infeasible request groups to improve the efficiency of request group enumeration. Furthermore, we maintain the request group for each request within a mixed graph structure named as *State Graph*. As the arrival and leaving of requests, the state graph is continuously adjusted and optimized until the node of request reaches the maximum time it is willing to wait, or the current matching is challenging to be improved anymore. Additionally, to balance response time and service quality, we propose an adaptive windowed matching algorithm that considers requests' personalized tolerable matching time to solve the ridesharing problem with an efficient and effective matching strategy.

TABLE 1. Requests release detail.

request	source	destination	release time	deadline
r_1	e	f	0	11
r_2	e	g	0	15
r_3	b	f	3	16
r_4	a	f	4	17
...			...	
r_i	s_i	e_i	d_i	t_i

We summarize the contributions of this paper as follows.

- We study a novel dynamic windowed ridesharing problem that takes the requests' personalized matching time into consideration.
- We devise an event-driven windowed matching algorithm, EGWM, which maintains and improves the request groups within the matching windows and provides a request matching solution with higher service rates at lower latencies.
- We propose an adaptive windowed matching algorithm based on the KL-UCB [19] policy, which adaptively balances the service quality and matching time through a user-defined parameter ϵ to meet different application scenarios.
- We conduct extensive experiments over two real-world datasets to demonstrate that our method achieves a better service quality with a shorter running time than the existing methods.

II. PROBLEM DEFINITION

In this section, we introduce the dynamic windowed ridesharing problem (DWRP) to be solved in this paper with its related concepts. The road network is the foundation of route planning in the ridesharing problem, and in this paper, we use a directed weighted graph to represent the real-life road network. Specifically, the nodes represent intersections, and the edges with their weights $cost(u, v)$ show the average travel time between two intersections u and v.

A. DEFINITIONS

Definition 2 (Request): Let $r_i = \langle s_i, e_i, n_i, t_i, d_i, w_i \rangle$ denote an online request r_i released at time t_i and required to be dispatched within time w. It contains n_i passengers that depart at s_i and expect to arrive at e_i before the deadline d_i.

TABLE 2. Symbols and descriptions.

Symbol	Description
R	a set of m time-constrained rider requests
r_i	rider request r_i of rider i
S_j	the planned route for vehicle v_j
T_i	the matching window of rider r_i
G	a candidate request group with size $\|G\| \leq c$
$N_G^+(v)$	the out-neighbors of node v in graph G
$N_G^-(v)$	the in-neighbors of node v in graph G

In practice, riders' patience is limited, so the ridesharing platform must complete each request's matching within the maximum tolerable waiting time w_i. So we define the *Matching Window* of the request as follows.

Definition 3 (Matching Window): Let $T_i = [t_i, t_i + w_i)$ denote a matching window of request r_i which starts from the release time t_i to the maximum waiting time $t_i + w_i$.

We denote the request r_b as *available* for r_a if and only if $T_a \cap T_b \neq \emptyset$ (i.e., $|t_a - t_b| \leq \min(w_a, w_b)$). In ridesharing, each vehicle v_j can be assigned with multiple requests R_j. Thus, while assigning requests to vehicles, the platform has to plan a *driving path* for them that can serve multiple assigned requests $r \in R_j$ simultaneously. We define the driving path of each vehicle as the route below.

Definition 4 (Route): Given a vehicle v_j with its assigned set R_j of m requests, let $S_j = \langle o_1, \ldots, o_{2m} \rangle$ denote the route for v_j where o_x is the pickup location s_i or drop-off location e_i of request $r_i \in R_j$.

The route for each vehicle consists of a sequence of sources and destinations for the assigned requests $r \in R_j$. And we mark a route as *feasible* if and only if it satisfies the following constraints:

- **Sequential constraint.** The source o_x and destination o_y of assigned request $r \in R_j$ in the feasible route S must satisfy $x < y$.
- **Capacity constraint.** For any location $o_x \in S$, the total number of passengers on the vehicle should not exceed the maximum capacity c_j of the vehicle v_j.
- **Deadline constraint.** For any location $o_x \in S$, $\sum_{k=1}^{x} cost(o_{k-1}, o_k) \leq ddl(o_k)$, where $ddl(o_k)$ shown in equation 1.

$$ddl(o_k) = \begin{cases} e_i - cost(s_i, e_i), & \text{if } o_x \text{ is source} \\ e_i, & \text{otherwise} \end{cases} \quad (1)$$

With the definitions above, we define the *Dynamic Windowed Ridesharing Problem* as follows.

Definition 5 (Dynamic Windowed Ridesharing Problem): Given a set R of n online requests with personalized matching window T_i for each $r_i \in R$, and a vehicle set W with maximum capacity constraint, the Dynamic Windowed Ridesharing Problem (DWRP) requires planning a feasible route for each vehicle $w \in W$ to serve $r \in R$, which minimizes a specific utility function.

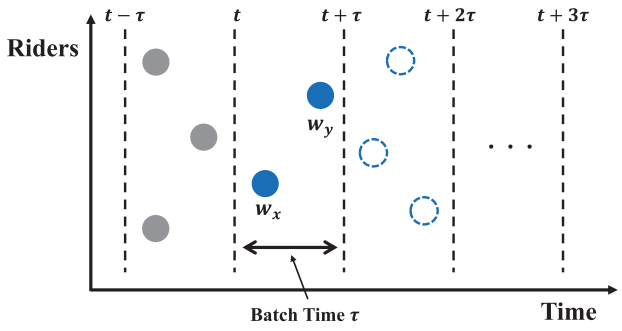


FIGURE 3. Matching window preprocessing in brute-force algorithm.

In this paper, we adopt the following unified cost UC defined in [18] as the optimization utility function. With specific parameters α and penalty p_r, the unified cost UC can be used to represent a variety of different optimization objectives, such as minimizing the total distance traveled, maximizing the number of service orders, and maximizing total revenue, etc.

$$UC(R, w) = \alpha \sum_{w_i \in W} \mu(S_{w_i}) + \beta \sum_{r_i \in R^-} cost(s_i, e_i) \quad (2)$$

$$\mu(S_w) = \sum_{o_x \in S_{w_i}} cost(o_{x-1}, o_x) \quad (3)$$

Table 2 summarizes the notations mainly used in this paper.

Hardness of DWRP: The dynamic ridesharing problem has been shown to be an NP-Hard problem [6], [11], [18], [20], and Tong *et al.* proved that there is no polynomial-time algorithm for the constant competitive ratio of the ridesharing problem in [18].

B. BRUTE-FORCE SOLUTION

Given a group of requests R, we call they are *shareable* if and only if there exists a feasible route S for serving $r \in R$ simultaneously. The existing batch-based methods [14]–[17] for the Dynamic Ridesharing Problem are based on a two-phase framework: (1) the enumeration of shareable request groups among the request in each batch; (2) the matching between request groups and vehicles to minimize the utility function. However, since each request $r_i \in R$ needs to be matched within the matching window T_i in the DWRP, existing batch-based methods [14]–[17] cannot be directly applied. Thus, in the Brute-Force algorithm of DWRP, we preprocess the matching window as shown in Figure 3 to fit the batch-based algorithm. That is, in each batch-based algorithm's trigger timestamp t, we prioritize and process those advent requests $r_x \in R$ whose matching window deadline w_x expires earlier than the next batch time $t + \tau$.

The detailed algorithm of Brute-Force is shown in Algorithm 1. We first retrieve the advent requests R^-, which expire before the next trigger time $t + \tau$ (line 3). After that, we try to

Algorithm 1: Brute-Force Solution

Input: A set R of n requests, a set W of m vehicles and a batch period τ
Output: The planned routes set $\mathbb{S}$ for vehicle $w \in W$

1 $t \leftarrow$ current timestamp
2 **while** R *is not empty* **do**
3 $R^- \leftarrow \{r_i | r_i \in R \wedge t_i + w_i < t + \tau\}$
4 **foreach** $w_j \in W$ **do**
5 $G \leftarrow$ initialize a set for shareable groups
6 **for** $k \in [1..c]$ **do**
7 $G' \leftarrow$ enumerate shareable request groups g among R^- where $|g| = k$
8 $G \leftarrow G \cup G'$
9 $g^* \leftarrow \min_{g \in G} UC(g, w)$
10 $S_j \leftarrow$ planning route for serving $r \in g^*$
11 $R \leftarrow R \setminus R^-$; $t \leftarrow t + \tau$
12 **return** $\mathbb{S} = \{S_j | w_j \in W\}$

identify the request group with the minimum unified cost for each vehicle w_j (line 4-10). Specifically, we first enumerate request groups of different sizes (line 5-8). To satisfy the maximum capacity constraint of the vehicle w_j, we only enumerate the request groups whose size does not exceed the vehicle capacity constraint c_j. Then, we select the request group g^* with the minimum unified cost from all the candidate request groups G and plan the optimal driving route S_j for the vehicle w_j to serve the requests $r \in g^*$ (line 9-10). Note that we plan the route for each vehicle by enumerating the sequence of the sources and destinations of assigned requests in a different order.

Complexity Analysis: For each vehicle w_j, we enumerate up to $O(n^c)$ candidate request groups. Then, to search for an optimal route that serves all requests $r \in g^*$ simultaneously, we need to enumerate up to $O((2c)!)$ different sequences of routes. And we have to examine each candidate route in $O(c)$. Thus, the time complexity of the Brute-Force algorithm is $O(m \times n^c \times (2c)! \times c)$.

III. GRAPH-BASED WINDOWED MATCHING

In Section II-B, we present the Brute-Force algorithm by prioritizing the advent requests based on the existing batch-based framework. However, it has not tailored design to the features of the matching window for the DWRP problem, which may result in insufficient utilization of the personalized allowed matching time of requests. Meanwhile, it's inefficient in the enumeration of candidate request groups, which costs up to $O(n^c)$ times. Therefore, in this section, we propose a well-tailored $\mathcal{E}$-Ride framework based on the event model for the DWRP. Additionally, we also proposed the concept of the shareability graph. We pruned the infeasible groups by their shareable relationships in the shareability graph, facilitating a more efficient enumeration of candidate request groups.

A. DYNAMIC SHAREABILITY GRAPH

In batch-based methods, the enumeration of request groups is a fundamental operation. To facilitate the enumeration of request groups, we first define the sharability graph to visually represent the shareable relationships between requests.

Definition 6 (Shareability Graph [21], [22]): Given a set of requests R, $SG = \langle R, E \rangle$ denotes the shareability graph of R, where $e = (r_i, r_j) \in E$ reflects that request r_i and r_j are shareable.

With the shareability graph, we have the following Theorem 7 for transforming shareable request groups into the k-clique structure in the shareability graph to continuously optimize the request groups in the platform within the online scenario. The clique [23]–[25] indicates a subset of interconnected nodes in the graph.

Theorem 7: Given a shareable request group Q of size k, the corresponding nodes of these k requests form a k-clique in the shareability graph.

Proof: Assume we have a set, R, of k requests, and there exists a feasible route S consisting of their sources and destinations. Then, for any two requests $r_a, r_b \in R$, let S' be a subsequence of S through removing the sources and destinations of all other requests except for r_a and r_b. Removing the locations from S will reduce the detours, which will maintain the validation of S. Thus, S' must still be a feasible route. According to the definition of the shareability graph, there must be an edge connecting r_a and r_b in the corresponding shareability graph. In conclusion, there must be an edge for any two requests $r_a, r_b \in R$ in the corresponding shareability graph, which means the nodes of k requests form a k-clique. □

According to Theorem 7, candidate shareable request groups will only exist in the k-cliques in the shareability network.

Dynamic Update of the Shareability Graph: While in the online scenario, we have to maintain the shareability graph dynamically as requests arrive or leave the platform.

- On the arrival of a request r_a, we first have to filter out all candidate shareable requests for r_a. Since the shareable requests often share similar sources to meet the deadline constraint, we can quickly filter out the candidate shareable requests R_a by maintaining a grid index. Next, we try to construct a feasible route for each candidate request $r_b \in R_a$ by insertion [13], and we add an edge $e = (r_a, r_b)$ to the SG if such a route exists.
- On the leaving of a request r_a, we simply update the sharability graph SG by removing the corresponding node r_a with all related edges $\{e = (r_a, r_b) | r_b \in N_{SG}(r_a)\}$, where $N_{SG}(v)$ represents the neighbors of node v in the SG.

B. AN OVERVIEW OF THE $\mathcal{E}$-RIDE FRAMEWORK

We first briefly introduce the essential parts of our $\mathcal{E}$-Ride framework, as illustrated in Figure 4.

- **Event Handler.** The $\mathcal{E}$-Ride framework mainly works on the event model, which will make corresponding

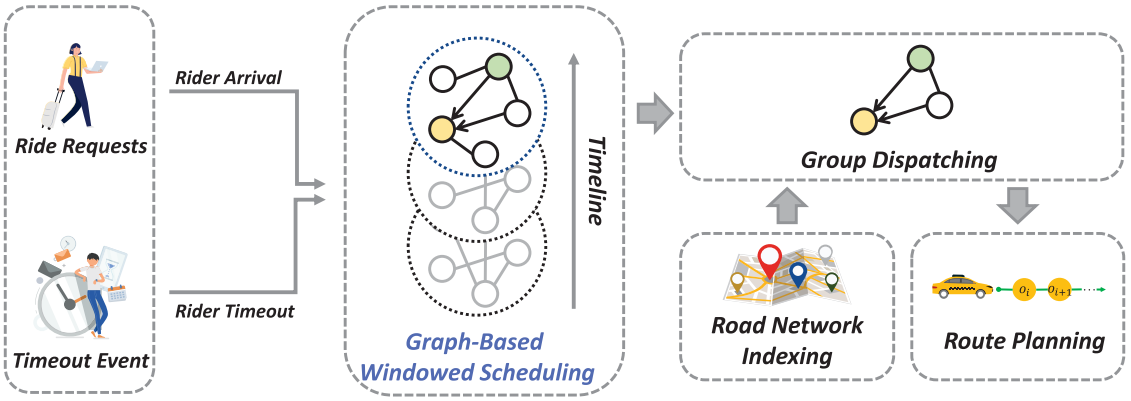


FIGURE 4. An overview of the $\mathcal{E}$-ride framework.

actions according to the type of the arriving events. The dynamic windowed ridesharing problem is an online problem where requests arrive at the platform dynamically. Both the arrival and leave of a request will affect the matching result of previously released requests in the platform, which makes it necessary for the framework to handle both cases, and we call these the arrival and leave of a request as *events*. As the arrival event arises, the framework will update the graph structure maintained for the requests within its matching window for optimizing the requests' groups.

- **Graph-Based Matching.** Graphs are powerful tools for maintaining relationships between nodes. Thus we store the groups among requests that are still within the matching window in the framework via a mixed graph (i.e., State Graph in Section III-C). As a new request r_i arrives, we update the existing request groups stored in the graph only if r_i can provide optimization (lower unified cost) for some existing groups.
- **Indexing Structure.** Since vehicles are always moving on the platform, the index structure adopted by the framework needs to satisfy the requirement of fast updates. A simple but effective way is to divide the full map into $m \times n$ grid cells. As the vehicle's location is updated, we can calculate the new grid based on the latitude and longitude of the vehicle in constant time to complete the update. Moreover, based on the grid index, we can achieve the approximate range query in constant time, speeding up the search for candidate vehicles in the assignment phase.

The brief processing flow of $\mathcal{E}$-Ride is shown in the Algorithm 2. As a new request $r_{\sigma(i)}$ arrives, we first include the node $r_{\sigma(i)}$ into the shareability graph SG. Then, we try to find out all candidate nodes R in the shareability graph SG that can share a trip with $r_{\sigma(i)}$, and update the shareability graph by adding undirected edges between $r_{\sigma(i)}$ and $r_j \in R$ (line 1-5). After that, we triggered the arrival handler (Algorithm 3) to seek for the optimal request group of node $r_{\sigma(i)}$ by enumerating candidate cliques in SG with Theorem 7 and update it to the state graph MG. And for each unit time t, we check if requests in SG exceed the matching window T_r (line 7-9). If the request needs dispatch immediately, we will perform the request group dispatching by triggering the leave handler (Algorithm 4).

Algorithm 2: Event-Driven Windowed Matching

Input: The request arrival sequence σ with a matching window T and a vehicle set W.
Output: An updated route set $\mathbb{S}$ for vehicles W.

```
1 SG ← initialize a empty shareability graph
2 foreach arrival request r_σ(i) according to σ do
3     foreach request r ∈ SG do
4         if r_σ(i) is shareable with r_j then
5             SG ← SG ∪ (r_σ(i), r_j)
6     trigger the Arrival Handler for r_σ(i) via Algo. 3
7     foreach request r ∈ SG do
8         if current time t exceeds T_r then
9             trigger the Leave Handler for r via Algo. 4
```

C. GRAPH-BASED WINDOWED MATCHING

In the dynamic windowed ridesharing problem, requests can be temporarily stored in the platform within the matching window in exchange for better request groups. Therefore, to maintain the temporary request groups in the platform, we extend the shareability graph SG to the following state (mixed) graph by adding a directed edge set A. For the convenience of explanation, in the following, we take the notation commonly used in the graph theory: $N^+(r_i)$ denotes the set of the out-neighbors of the node r_i, $N^-(r_i)$ for the in-neighbors, and $N(r_i)$ for all undirected neighbors.

Definition 8 (State Graph): Let the mixed graph $MG = \langle R, E, A \rangle$ denotes the state graph on the shareability graph

$SG = \langle R, E \rangle$, where for each of the node $r_i \in R$ with its in-neighbors $N^-_{MG}(r_i)$ indicates the request group stored in the state graph.

For example, with the state graph shown in the left of Figure 5, the request node r_2 with its in-neighbors $N^-_{MG}(r_2) = \{r_3, r_4\}$ indicates the reqeust group Q_2 stored in the state graph MG. The node r_1's in-neighbors $N^-_{MG}(r_1) = \emptyset$ shows that request r_1 forms a self-contained group Q_1.

Because of the mutual sharing relationship between requests, we assign each node in the state graph to be *seller* or *buyer* to avoid redundant group enumeration. The node as a *buyer* implies that the node has selected a previously arrived node in the shareability graph to form a group. Conversely, a node as a *seller* means that it was ''bought'' (selected) by several nodes labeled with *buyer* to share their trips. Therefore, for any request r in the sharability graph, two types of request groups need to be analyzed: the group of r as buyer and seller. We present the ''bought'' relationship by the directional edges A in the state graph MG. And because of the capacity constraint c and the uniqueness of the request, the number of in-neighbors and out-neighbors of request r_i in the state graph MG should satisfy the following two constraints.

$$|N^-_{MG}(r_i)| < c \tag{4}$$

$$|N^+_{MG}(r_i)| \leq 1 \tag{5}$$

Therefore, a request group consists of only one seller node and less than c buyer nodes in the state graph. We note a request group with seller node r_i as $Q_i = \{r_i \cup N^-(r_i)\}$. And we define the group confliction as Definition 9 once the given request group pairs share a same *seller* node.

Definition 9 (Group Confliction): Given a pair of groups $Q_a = \{r_a \cup N^-(r_a)\}$ and $Q_b = \{r_b \cup N^-(r_b)\}$, we say that Q_a and Q_b are conflict if and only if $r_a = r_b$. Moreover, we denote the nodes $Q_a \setminus Q_b$ and $Q_b \setminus Q_a$ as *conflict nodes* for Q_a and Q_b , respectively.

Example 10: Consider the state graph in the middle of Figure 5, the seller node r_2 together with the buyer nodes r_i and r_4 form the request group Q^*_2, and the seller node r_2 also form the request group Q_2 with the buyer nodes r_3, r_4. Therefore, we say Q_2 and Q^*_2 are conflict groups. And r_3 is the conflict node for group Q^*_2.

The $\mathcal{E}$-Ride framework always maintains and improves the request groups in the state graph so that the request groups within it have a better unified cost after several rounds of iterations. Since a better request group Q implies that the requests $r \in Q$ are well shared, the induced sub-groups in the state graph also perform better in the unified cost. Therefore, we adopt the following two strategies in the group enumeration for a newly arrived request r_i:

1) Add r_i to an existing request group Q;
2) Take the request $r \in Q$ from the existing group Q to form a new request group Q' with r_i.

After updating the state graph, the critical decision for the request r_i is to continue waiting for a better request group or to be scheduled with the current one after events are triggered. Since we cannot predict whether a better request group will emerge or not in the online scenario, a simple but feasible solution is to make a decision randomly. Therefore, for each request r_i, it will decide with a probability of $1/2$ independently to stay and wait or to leave immediately.

With the above state graph and update strategies, we propose a novel matching algorithm, *Event-driven Graph-based Windowed Matching (EGWM)*. Once a new request r_i is released on the platform, EGWM first enumerates and searches for the optimal request group Q_i of r_i as a buyer and updates them to the state graph MG. While updating the requests $r \in Q_i$ may cause some conflict request groups to be unavailable. Therefore, we will recursively update such affected (conflict) nodes. Since subsequent arrival requests may improve the current request group Q_i (r_i as seller), it is not reasonable to dispatch Q_i immediately compared to the online-based methods. Thus, we will ''toss a coin'' to decide that the request group Q_i stays in the MG or dispatch. But the request r_i will stay in the SG and MG for at most w_i time because of the constraint of the matching window. And the request r_i will trigger a timeout event requiring us to dispatch Q_i after w_i time elapses.

The details of the arrival handler are shown in the Algorithm 3. We first enumerate the request groups for r_i by iterating over its neighbors $r_p \in N(r_i)$. With the Theorem 7, we only need to enumerate the k-cliques that contain r_i. (line 3-10). We adopt different enumeration strategies depending on the label of the neighbor r_p. In case the neighbor r_p as seller, which means that it already belongs to a request group Q_p, we try to enumerate among the nodes $r \in Q_p$ and insert the feasible request groups to the priority queue $\mathcal{Q}$ sorted by the unified cost (line 4-5). Note that we only reserve the request groups for which the unified cost satisfies $UC(Q) < UC(Q_p)$. Once the request r_p is labeled as the buyer in the state graph, we are not required to enumerate all request groups since the groups containing other requests in Q_p have already been listed in line 5. So that we only need to take r_p from Q_p for a new request group $Q_i = \{r_i, r_p\}$, and check if the unified cost satisfied $UC(Q_i) + UC(Q_p \setminus Q_i) < UC(Q_p)$ (line 6-8). Otherwise, if r_p is not labeled in MG, the 2-clique $Q_i = \{r_i, r_p\}$ can be directly considered as an available candidate request group for improving the pairing rate (line 9-10). After that, we take the clique Q^*_i with the minimum unified cost from the priority queue $\mathcal{Q}$ as the optimal request group of r_i and update it to the state graph MG (line 12). If the request group Q^*_i enumerated by the seller r_p of the original group Q_p (i.e., Q^*_i is the conflict group with Q_p), the conflict nodes $\overline{Q^*_i} = Q_p \setminus Q^*_i$ need to be updated due to the unavailability of r_p (line 16). Therefore, we clear the original labels from MG and rehandle the conflict nodes $r \in \overline{Q^*_i}$ by Algorithm 3 recursively (line 17-19). Since it is hard to determine whether a better request group will emerge shortly, we decide whether to assign the current request groups directly by tossing a coin, and we dispatch the request r_i by Algorithm 4 to complete

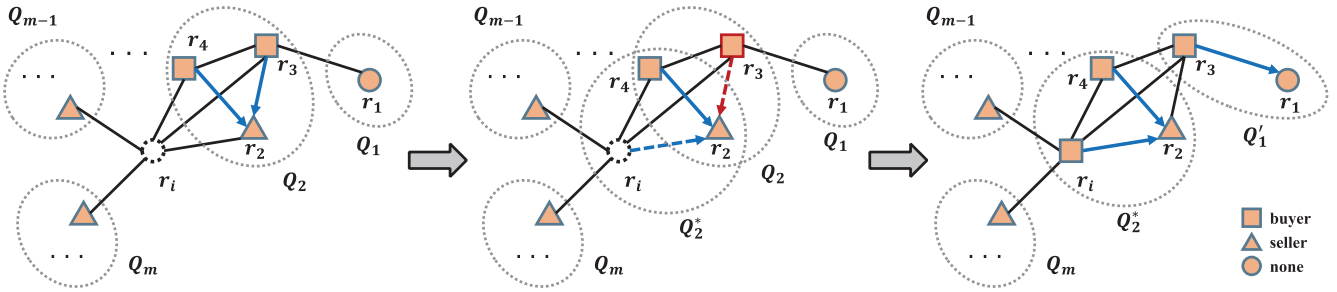


FIGURE 5. An illustration example of the state graph updating by EGWM algorithm.

Algorithm 3: Arrival Event Handler

Input: The arrived request r_i, state graph $MG = \langle R, E, A\rangle$ and vehicle set W.
Output: An updated set of workers and updated state graph MG.

1 $L_{MG}(r_i) \leftarrow$ initialize the label of r_i to *none*
2 $\mathcal{Q} \leftarrow$ initialize a priority queue ordered by unified cost (UC)
3 **foreach** $r_p \in N(r_i)$ **do**
4 **if** $L_{MG}(r_p)$ *is seller* **then**
5 $\mathcal{Q}[r_p] \leftarrow \{Q | Q \in CliqueEnum(r_i \cup Q_p) \wedge (r_i, r_p \in Q)\}$
6 **else if** $L_{MG}(r_p)$ *is buyer* **then**
7 **if** $UC(\{r_i, r_p\}) + UC(Q_p \setminus \{r_p\}) < UC(Q_p)$ **then**
8 $\mathcal{Q}[r_p] \leftarrow \mathcal{Q}[r_p] \cup \{r_i, r_p\}$
9 **else** ▷ *None Label*
10 $\mathcal{Q}[r_p] \leftarrow \mathcal{Q}[r_p] \cup \{r_i, r_p\}$
11 **if** $|\mathcal{Q}| > 0$ **then**
12 $\{r_p, Q^*\} \leftarrow$ retrieve group with maximum UC in $\mathcal{Q}$
13 $L_{MG}(r_i) \leftarrow$ *buyer*
14 $A \leftarrow A \cup (r_i, r_p)$
15 **if** r_p *is seller* **then**
16 $\overline{Q^*} \leftarrow Q_p \setminus Q^*$
17 clear labels of $r \in \overline{Q^*}$ and related edges from MG
18 **foreach** $r \in \overline{Q^*}$ **do**
19 Rehandling r via Algorithm 3 recursively
20 **else**
21 $L_{MG}(r_p) \leftarrow$ *seller*
22 **if** *toss a coin results in leaving directly* **then**
23 dispatching request group of r_i via Algorithm 4

Algorithm 4: Leaving Event Handler

Input: The leaving request r_i, the state graph MG and the vehicle set W.
Output: An updated set of workers and the updated state graph MG.

1 **if** $L_{MG}(r_i)$ *is seller* **then**
2 $Q_i^* \leftarrow N^-(r_i) \cup \{r_i\}$
3 **else if** $L_{MG}(r_i)$ *is buyer* **then**
4 $Q_i^* \leftarrow \bigcup_{r \in N^+(r_i)} N^-(r) \cup N^+(r_i)$
5 **else**
6 $Q_i^* \leftarrow \{r_i\}$
7 retrieve the nearest vehicle $w \in W$ to server Q_i^*
8 remove $r \in Q_i^*$ and related edges from SG and MG

the assignment. Specifically, we retrieve the request group Q_i^* in the state graph MG (line 1-6) and assign it to the nearest available vehicle (line 7). Finally, we remove the nodes $r \in Q_i^*$ and related edges associated with the assigned requests from the SG and MG.

Example 11: Let's consider the requests in Example 2. Assume that there exists a newly arrived request r_i who triggers an arrival event, and the updated shareability graph is shown in left of Figure 5. We first iterate over all neighbors of r_i, $N(r_i)$, and try to enumerate the optimal request group. Since r_2 is currently a seller, we enumerate all cliques containing r_i and r_2, i.e., $\{r_i, r_2\}, \{r_i, r_2, r_3\}, \{r_i, r_2, r_4\}, \{r_i, r_2, r_3, r_4\}$, and we add them to the priority queue $\mathcal{Q}$ with their corresponding unified cost. Then we visit the neighbor r_4, and because the label of r_4 is buyer, we check the sum of the unified cost of the group $Q_2 \setminus \{r_4\}$ after the loss of r_4 from the original group Q_2 with the newly generated group $\{r_i, r_4\}$. We add the new request group to $\mathcal{Q}$ because $UC(\{r_i, r_4\}) + UC(Q_2 \setminus \{r_4\}) < UC(Q_2)$. The operations on the remaining neighbors are omitted here. After that, we pick the request group $Q_2^* = \{r_i, r_2, r_4\}$ with the minimum unified cost from the priority queue $\mathcal{Q}$ and update the label of r_i to buyer with its associated edges to MG. Since group Q_2^* is enumerated based on the neighbor r_2 labelled as seller, so the conflict node $r_3 \in Q_2 \setminus Q_2^*$ should be updated by Algorithm 3 for a new group. Finally, we derive the state graph as shown in right of Figure 5.

IV. ADAPTIVE DISPATCHING WITH KL-UCB POLICY

In Section III-C, the EGWM algorithm refines the processing framework for the matching window feature in the DWRP problem. However, the stochastic dispatch policy has not taken the response time metric into account, which significantly impacts users' experiences. Therefore, the EGWM can not balance response time and service quality for different practical scenarios. Moreover, random dispatching is not a wise choice for different requests. Thus, in this section, we propose an online learned policy method based on the Multi-Armed Bandit (MAB) model to make dispatching decisions dynamically.

A. ONLINE LEARNED DISPATCHING POLICY

The Multi-Armed Bandits (MAB) [26] problem is a classical machine learning problem, demonstrating the dilemma of exploration and exploitation. In the MAB problem, the agent selects the action without prior knowledge of the reward for t rounds and maximizes the cumulative expected reward (minimize expected regret). Through multiple rounds of interaction with the environment, the agent will gather observations of the distribution of rewards for each action. Therefore, the agent would like to select the best performing action from historical observations to obtain a relatively high reward (Exploitation) and try some actions that have not been observed enough to obtain potentially high payoffs (Exploration). Note that over-exploitation may cause agents to miss the optimal action, while over-exploration will cause agents to pay too much learning cost. How to balance exploitation and exploration is the critical issue to be considered by the solution algorithms of MAB.

To balance the exploration and exploitation, the Upper Confidence Bound (UCB) method is a typical strategy, which maintains a upper confidence bound of the reward for each action a_i by the empirical mean reward $\hat{Q}_t(a_i)$ of past observations and the confidence radius $\hat{U}_t(a_i)$. And the confidence radius $\hat{U}_t(a_i)$ is a function of $N_t(a)$, which will decrease as the number of observations $N_t(a)$ of a_i increases. Thus, in the UCB methods, we always greedily choose the action a_t^* with the maximum upper confidence bound in round t as shown in Equation 6.

$$a_t^* = \arg\max_{a_i} \left\{ \hat{Q}_t(a_i) + \hat{U}_t(a_i) \right\} \tag{6}$$

In [19], A Garivier *et al.* proposed *Kullback-Leibler UCB* (KL-UCB) algorithm based on Bernoulli Kullback-Leibler divergence, which satisfies a uniformly better expected reward. The formal description of *KL-UCB*(a_i) for each arm a_i is shown in Equation 7, which can be efficiently calculated by Newton iteration.

$$\max \left\{ q \in \Theta : N_t(a_i)\, d\left(\frac{S_t(a_i)}{N_t(a_i)}, q\right) \leq \log(t) + c\log(\log(t)) \right\}, \tag{7}$$

where $N_t(a_i)$ and $S_t(a_i)$ denote the number of times action a_i gets selected and the total reward of a_i in t rounds; c is a parameter for the regret bound; $d(p, q)$ is the Bernoulli Kullback-Leibler divergence as shown in Equation 8.

$$d(p, q) = KL(p, q) = p \log \frac{p}{q} + (1-p) \log \frac{1-p}{1-q} \tag{8}$$

B. ADAPTIVE WINDOWED MATCHING

In the DWRP problem, the balance between response time and service quality can be regarded as whether to leave the platform early or not. The action of early leaving can reduce the request's response time while staying in the platform as long as possible may improve the request group and thus improve the service quality to reduce the cost. The KL-UCB policy can be conveniently embedded in such a process. We can consider staying and leaving as two candidate actions and design the reward function to maximize the desired objective by learning a better strategy through continuous online iterations.

For KL-UCB to be able to balance the response time and the service quality, we define the following reward function $Reward(Q, w) =$

$$\epsilon \cdot \left(1 - \frac{\mu(S_w)}{\sum_{r \in Q} cost(r)}\right) + (1-\epsilon) \cdot \left(1 - \frac{\sum_{r_i \in Q}(t - t_i)}{\sum_{r_i \in Q} |T_i|}\right)$$

where the left part of $Reward(Q, w)$ evaluates the ratio of travel time saved by the request group; the right part analyzes the time saved by the earlier response with respect to the maximum allowed matching time; the parameter ϵ balances these two components so that they can be applied to scenarios with different emphases.

Based on the KL-UCB Policy with the reward function above, we have the following adaptive matching algorithm shown in Algorithm 5. As new request r_i arrives, we first enumerate the request groups for r_i and update the state graph MG by following the same methods as in Algorithm 3, lines 1-21 (line 1). Then we retrieve the recommended action a_i^* with the largest KL-UCB from the policy instance $\mathcal{P}$ of the platform (line 2). Once the recommendation a_i^* of $\mathcal{P}$ is to dispatch r_i immediately (line 3-8), we will select the service vehicle w_i^* by the reward function and feed the reward to the policy instance $\mathcal{P}$ (line 5-6). Finally, we plan the route S_j for the selected vehicle w_j^* and remove the request r_i with related edges from the state graph MG (line 7-8).

V. EXPERIMENTS

A. EXPERIMENTAL SETUP

1) DATASETS

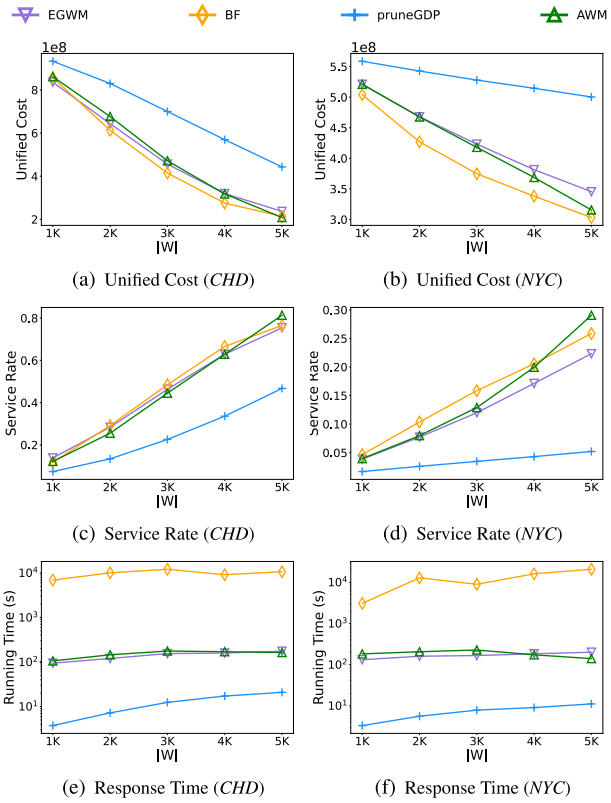
The request datasets of Chengdu (noted as *CHD*) and New York City (noted as *NYC*) were used to demonstrate the effectiveness and efficiency of our proposed methods in this paper. The road networks of both cities are downloaded from Geofabrik [27] and segmented by Osmconverter [28] with city boundaries on OpenStreetMap [29] for *CHD* [30] and *NYC* [31], respectively. In addition, we also carefully trimmed the road networks according to the distribution boundaries of request sources and destinations so that there are fewer

Algorithm 5: Adaptive Windowed Matching

Input: The newly arrival request r_i, the State Graph MG, the instance of KL-UCB Policy $\mathcal{P}$ and a vehicle set W.

Output: An updated route set $\mathbb{S}$ for vehicles W.

Update state graph MG via line 1-21 of Algorithm 3
$a^* \leftarrow \arg\max_{a_i \in \{\text{stay,leave}\}} KL\text{-}UCB(\mathcal{P}, a_i)$
if $a^* = leave$ **then**
  $Q_i^* \leftarrow$ retrieve the request group of r_i from MG
  $w_j^* \leftarrow \arg\max_{w_j \in W} Reward(Q_i^*, w)$
  $\mathcal{P}$.updateState$\left(Reward(Q_i^*, w_j^*)\right)$
  $S_j \leftarrow$ planning route for serving $r \in Q_i^*$
  remove $r \in Q_i^*$ and related edges from MG
return $\mathbb{S} = \{S_j | w_j \in W\}$



(a) Unified Cost (*CHD*) (b) Unified Cost (*NYC*)
(c) Service Rate (*CHD*) (d) Service Rate (*NYC*)
(e) Response Time (*CHD*) (f) Response Time (*NYC*)

FIGURE 6. Performance of varying $|R|$.

irrelevant regions as possible. The weight associated with each edge on the road network is the average travel time of the road segment. Since there is no field about the number of passengers in *CHD* dataset, we generate the field based on the distribution in *NYC* as [18]. And we set the deadline of request r_i as $d_i = t_i + \gamma \cdot cost(r_i)$, which is a commonly used configuration in many existing works [6], [11], [20].

In the experiments for analyzing the effects of different parameters, we used data from *CHD* on October 31, 2016, and *NYC* on April 09, 2016, for a full day of testing. These datasets are available on Didi GAIA [32] and NYC Official Website [33], respectively. The detailed experiment-related parameters are shown in Table 3 (default parameters are in bold).

TABLE 3. Experimental settings.

Parameters	Values
the number, n, of requests	10K, 50K, 100K, 150K, 200K, **250K**
the number, m, of vehicles	1K, 2K, 3K, 4K, **5K**
the capacity of vehicles c	2, 3, **4**, 5, 6
the deadline parameter γ	1.2, 1.3, **1.5**, 1.8, 2.0
the penalty coefficient p_r (β)	**10**

2) IMPLEMENTATION

We simulated the ridesharing and the driver's moving based on the released time of the requests. We pre-map the sources and destinations of the requests to the nearest nodes on the road network through the VP-Tree [34]. The initial location of the worker is set to the earliest occurrence of GPS track points in the dataset. The pruning strategy based on Euclidean spaces is approximated by multiplying the average travel time by the maximum speed.

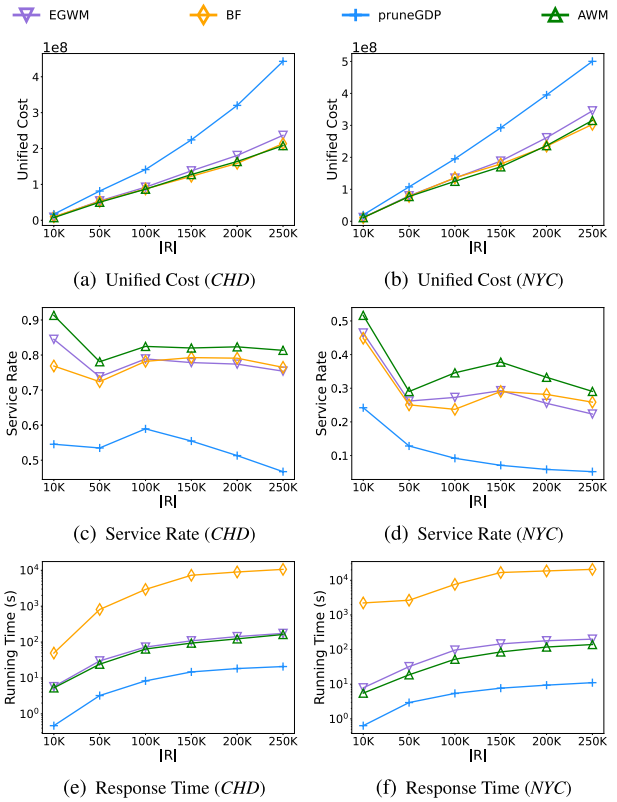
3) ENVIRONMENTS

All algorithms are implemented with C++ and compiled with -O3 optimization. The algorithms run on a single server equipped with Xeon(R) Silver 4210R CPU @ 2.40GHz and 128GB RAM. Besides, all algorithms are implemented in a single thread.

B. APPROACHES AND MEASUREMENTS

We compare the following four algorithms in the experimental study.

- ***pruneGDP [18].*** It inserts the request into the vehicle's current schedule sequentially and selects the vehicle with the least increased distance for service.
- ***BF.*** The Brute-Force method shown in Algorithm 1. It is in batch mode and enumerates all request groups for each vehicle's candidate requests in random order. We performed preprocessing as shown in Section II-B to accommodate the matching window.
- ***EGWM.*** The event-driven based algorithm proposed in Section III maintains the request group by graphs. It continuously iterates to optimize the request group within the matching window allowed by request. Also, it utilizes a randomized approach to decide whether a request is to be dispatched before the maximum matching window or not.
- ***AWM.*** It is a variant of the EGWM algorithm proposed in Section IV, which dynamically learns to decide whether a request is dispatching before the maximum matching window based on the KL-UCB policy. Additionally, the reward function's adaptation parameter ϵ has been fine-tuned for better service quality by default.



(a) Unified Cost (*CHD*) (b) Unified Cost (*NYC*)
(c) Service Rate (*CHD*) (d) Service Rate (*NYC*)
(e) Response Time (*CHD*) (f) Response Time (*NYC*)

FIGURE 7. Performance of VARYING *c*.

We report all algorithms' unified cost, service rate, and overall running time. Specifically, the unified cost adopts the evaluation of total revenue in [18], and the varying penalty coefficient p_r is equivalent to the balance between income per unit time and fare per unit distance. The service rate evaluates the number of requests the platform accepts with a limited number of vehicles. The overall running time demonstrates the efficiency of the algorithms for processing the same number of requests. We early terminated those not completed experiments within 12 hours.

C. EXPERIMENTAL RESULTS

1) EFFECT OF THE NUMBER OF VEHICLES

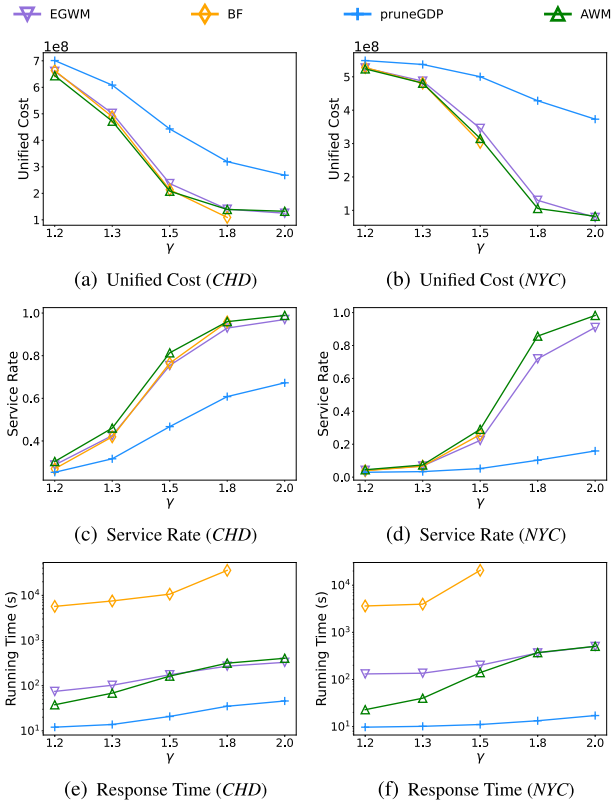
Figure 6 shows the results of varying the number of vehicles from 1K to 5K. As the number of vehicles increases, so does the service quality of the evaluated methods. The BF algorithm leads other methods for the uniform cost, which mainly benefit from its brute force enumeration strategy. The EGWM and AWM have a very close performance with the BF algorithm. However, in terms of the overall running time, because of the high time complexity of the brute force computation in the BF algorithm, it takes nearly up to three hours and six hours to run on the two test datasets, respectively. In contrast, the performance of the EGWM and AWM methods proposed in this paper is 61.03 times and 148.70 times faster compared with the BF algorithm on the CHD and NYC datasets (as shown in Figure 6(e) and 6(f)). It mainly results from the fact that the clique enumeration strategy proposed in Section III-A avoids unnecessary enumeration of request combinations, and we store the better request group through the state graph and keep optimizing them, which provides a near online performance. Benefiting from the linear time complexity of the online algorithm *GDP*, it is leading in terms of overall running time. However, it performs not do well in service quality (service rate and unified cost) because it does not analyze request groups among requests. Furthermore, the superiority of EGWM and AWM gradually appears when the number of vehicles is large enough. As shown in Figure 6(c) and 6(d), when the number of vehicles is 5K, the service rate of EGWM and AWM achieves 4.85% and 3.2% improvement compared to the BF algorithm (about 12125 and 8000 requests, respectively).

2) EFFECT OF THE NUMBER OF REQUESTS

Figure 7 presents the results of varying the number of requests from 10K to 250K. Because the number of accepted and rejected requests increased significantly, the unified costs of all experiment algorithms are growing. For service rate shown in Figure 7(c) and 7(d), the AWM algorithm performs the best, achieving improvements ranging from 3.57% $\sim$ 34.61% and 2.84% $\sim$ 30.66% over other methods on the two datasets CHD and NYC, respectively. For the running time, the insertion-based methods are still faster. In Figure 7(e), EGWM and AWM are 9.26$\times$ $\sim$ 78.3$\times$ faster than BF on CHD. And on the NYC dataset, the AWM algorithm performs even 401.52$\times$ faster than the BF algorithm as shown in Figure 7(f).

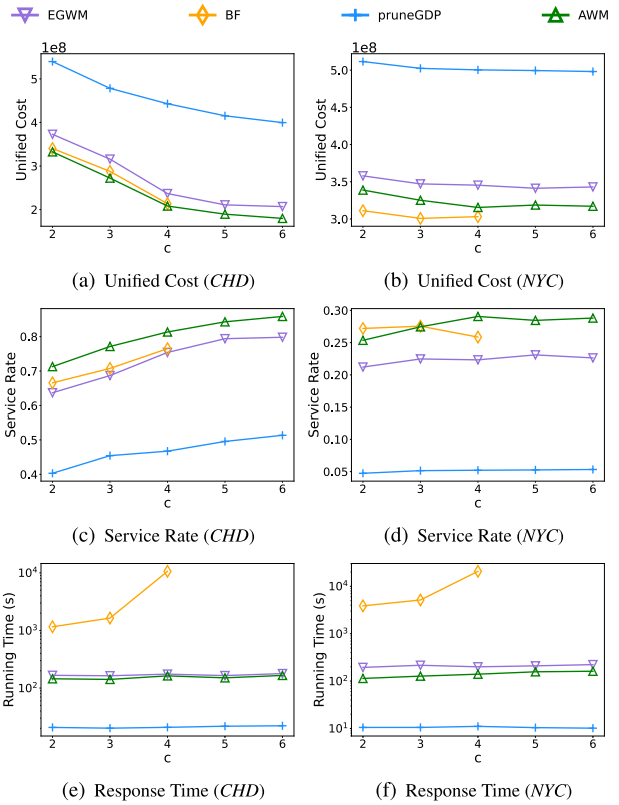
3) EFFECT OF DEADLINE

Figure 8 presents the results of the varying deadline of requests by changing the deadline parameter γ from 1.2 to 2.0. The results for service rate are similar among the compared algorithms when we strictly set the deadline of requests, i.e., $\gamma = 1.2$. The reason is that the number of candidate vehicles for each request reduces significantly with a minor deadline, making it challenging to achieve noticeable performance improvements by applying request group analyzing strategies. However, the AWM algorithm achieves similar service quality using only 0.6% of the running time of the BF. The superiority of AWM is more explicit compare to the EGWM, where the AWM is 1.98$\times$ and 5.95$\times$ faster than EGWM on two datasets when $\gamma = 1.2$. With the increase of deadline, the superiority of group-based algorithms (i.e., BF, EGWM and AWM) gradually realizes. The service rate of these methods achieves more than 90% when the deadline is 1.8$\times$. Note that BF performs inefficiently on two datasets with $\gamma \geq 1.8$, which is primarily due to the increase of request groups with the relaxation of the deadline. Besides, BF enumerates all the combinations of requests and schedules almost for each vehicle. However, in EGWM and AWM, the request groups are stored in the state graph and improve gradually. Moreover, the request group only updated when the following arrived requests improved the utility. Thus, the



(a) Unified Cost (*CHD*) (b) Unified Cost (*NYC*)
(c) Service Rate (*CHD*) (d) Service Rate (*NYC*)
(e) Response Time (*CHD*) (f) Response Time (*NYC*)

FIGURE 8. Performance of Varying γ.



(a) Unified Cost (*CHD*) (b) Unified Cost (*NYC*)
(c) Service Rate (*CHD*) (d) Service Rate (*NYC*)
(e) Response Time (*CHD*) (f) Response Time (*NYC*)

FIGURE 9. Performance of Varying c.

time cost for the request group enumeration is significantly reduced in EGWM and AWM, benefiting from our "event-driven" execution strategy. The results of BF in *CHD* with $\gamma = 2.0$ and *NYC* with $\gamma \geq 1.8$ are not presented because the running time exceeds the limit of our experiment settings (i.e., 12 hours). In addition, AWM performs the best in terms of unified cost and service rate as shown in Figure 8(a) to 8(d), which improves up to 31.6% and 82.97% compared with other algorithms on two datasets.

4) EFFECT OF VEHICLE'S CAPACITY CONSTRAINT

Figure 9 illustrates the results of varying the vehicle's capacity from 2 to 6. In terms of unified cost, BF and AWM have similar results on the CHD dataset, but BF leads on the NYC dataset by enumerating the best request group. However, since the number of request groups increases dramatically with vehicle capacity for the BF method (e.g., when $c = 6$, the BF algorithm needs to enumerate C_n^6 different request groups), the BF algorithm cannot finish within the given time limit when $c \geq 5$. The unified cost of AWM is slightly better than that of EGWM on two datasets, which mainly benefited from the decision of the KL-UCB policy. As for service rate, AWM is still the best among all tested algorithms (except $c = 2$ on NYC). In terms of running time, AWM is the fastest among group-based methods (BF, EGWM and AWM), which is 8.03× $\sim$ 147.85× faster than BF.

5) DISCUSSION

The linear insertion method benefits from its linear time complexity, which allows the request to be scheduled to the vehicle within a short time after arrival. However, such an approach suffers from poor performance on service rate and unified cost due to the low level of sharing between requests. On the other hand, the windowed matching model requires requests to stay on the platform for a relatively short time window, which is a trade-off between matching time and service quality. The experimental results also show that although the windowed matching model requires waiting and a longer matching time than the online-based model, such waiting is acceptable in real-life scenarios.

6) SUMMARY OF THE EXPERIMENTAL STUDY

- The group-based methods (i.e., BF, EGWM, AWM) have superior performance on service quality (i.e., higher service rates and lower unified costs) compared to the online-based methods (i.e., pruneGDP). For example, the AWM achieves a service rate improvement up to 35% compared to the other tested algorithm.
- The EGWM and AWM algorithms can lead and show excellent performance in most cases. For example, AWM runs up to 401.52 times faster than BF in Figure 7(f). In other words, AWM can process NYC requests in 3 minutes, but BF takes up to 5.8 hours

- The KL-UCB strategy has a noticeable effect on improving the service quality. For instance, AWM has improved the service rate of the random policy over EGWM by up to 8.44% and 6.73% on the two datasets, respectively.

VI. RELATED WORK

With the development of GPS equipment, mobile Internet, and sharing economy, ridesharing service has gradually become an indispensable choice for people to travel. The ridesharing problem can be reduced to a variant of the Dial-a-Ride (DARP) problem [35], [36], which dedicates to planning service routes for vehicles to serve n requests with specified origins, destinations, and practical constraints. Most of the existing works [37], [38] mainly focused on the static case, where all requests were already known in advance. However, with the increasing demand for ridesharing, the concept of the dynamic ridesharing problem is more relevant to practical applications. For the dynamic ridesharing problem, the existing solutions are mainly in online mode [5], [6], [13], [18] or batch mode [14]–[16], [39].

In online mode, *insertion* [40] is the state-of-the-art operation of the existing works [41], [42] in route planning, which inserts the pickup and drop-off locations of the request into the vehicle's current schedule without reordering. The insertion-based algorithms are more efficient in practice on real-life datasets [13]. Furthermore, Tong *et al.* [18] proposed an improved insertion method based on dynamic programming, which checks the constraints in constant time and dispatches requests in linear time. On the other hand, Huang *et al.* [6] proposed the structure of the kinetic tree, which is used to trace all feasible routes for each vehicle to reduce the total driving distance. Whenever the schedule changes with the arrival of a new rider, Kinetic Tree always provides the optimal schedule for the vehicle. However, existing online-based methods perform poorly in service quality due to the lack of detailed enumeration and analysis of shareable request groups.

The batch-based algorithms usually partition the request into groups with a route and then assign groups to their appropriate vehicles. In [14], Alonso-Mora *et al.* propose RTV-Graph to model the relationship and constraints among requests, trips, and vehicles, where trips are the groups composed of shareable requests. With the RTV-Graph, they minimize the utility function by linear programming to get the allocation result between vehicles and trips. But the time cost for enumerating trips in the process of building RTV-Graph grows exponentially. In [15], Zeng *et al.* proposed an index called additive tree for pruning the infeasible groups during the group enumeration, and greedily choosing the most profitable request group to the server. Nevertheless, existing batch-based methods dispatch requests through a fixed interval without taking personalized matching windows for requests into account. Therefore, in this paper, we propose an event-driven ridesharing framework, $\mathcal{E}$-Ride, which improves the service quality of the existing algorithms and provides efficiency close to the online methods.

VII. CONCLUSION

In this paper, we study the dynamic ridesharing problem with a personalized matching window. Concretely, we first proposed an event-driven framework, $\mathcal{E}$-Ride, which maintains the request groups through a state graph extended by the shareability graph. Then, we propose an efficient request group enumeration strategy based on the k-clique in the shareability graph, which helps improve the request groups by the arrival of the subsequent requests efficiently. Furthermore, to satisfy different application scenarios, we designed the reward function for the KL-UCB policy to learn the dispatching strategy dynamically to balance the matching efficiency and service quality. In the experimental study, the extensive experiment results demonstrated that our method achieves a better service rate, less unified cost, and shorter running time than the state-of-the-art methods.

REFERENCES

[1] G. Laporte, F. Meunier, and R. Wolfler Calvo, "Shared mobility systems: An updated survey," *Ann. Oper. Res.*, vol. 271, no. 1, pp. 105–126, Dec. 2018, doi: 10.1007/s10479-018-3076-8.

[2] S. C. Ho, W. Szeto, Y.-H. Kuo, J. M. Y. Leung, M. Petering, and T. W. Tou, "A survey of dial-a-ride problems: Literature review and recent developments," *Transp. Res. B, Methodol.*, vol. 111, pp. 395–421, May 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0191261517304484

[3] *Didi*. Accessed: Mar. 1, 2022. [Online]. Available: https://www.didiglobal.com/

[4] *Uber*. Accessed: Mar. 1, 2022. [Online]. Available: https://www.uber.com/

[5] 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)*, C. S. Jensen, C. M. Jermaine, and X. Zhou, Eds. Brisbane, QLD, Australia, Apr. 2013, pp. 410–421, doi: 10.1109/ICDE.2013.6544843.

[6] Y. Huang, R. Jin, F. Bastani, and X. S. Wang, "Large scale real-time ridesharing with service guarantee on road networks," *Proc. VLDB Endowment*, vol. 7, pp. 2017–2028, Oct. 2013. [Online]. Available: http://www.vldb.org/pvldb/vol7/p2017-huang.pdf

[7] 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. Geographic Inf. Syst.*, S. Ravada, M. E. Ali, S. D. Newsam, M. Renz, and G. Trajcevski, Eds. Burlingame, CA, USA, Oct. 2016, pp. 3:1–3:10, doi: 10.1145/2996913.2996974.

[8] M. Asghari and C. Shahabi, "An on-line truthful and individually rational pricing mechanism for ride-sharing," in *Proc. 25th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, E. G. Hoel, S. D. Newsam, S. Ravada, R. Tamassia, and G. Trajcevski, Eds. Redondo Beach, CA, USA, Nov. 2017, pp. 7:1–7:10, doi: 10.1145/3139958.3139991.

[9] B. Cici, A. Markopoulou, and N. Laoutaris, "Designing an on-line ride-sharing system," in *Proc. 23rd SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, J. Bao, C. Sengstock, M. E. Ali, Y. Huang, M. Gertz, M. Renz, and J. Sankaranarayanan, Eds. Bellevue, WA, USA, Nov. 2015, pp. 60:1–60:4, doi: 10.1145/2820783.2820850.

[10] S. Yeung, E. Miller, and S. Madria, "A flexible real-time ridesharing system considering current road conditions," in *Proc. 17th IEEE Int. Conf. Mobile Data Manage. (MDM)*, Porto, Portugal, Jun. 2016, pp. 186–191, doi: 10.1109/MDM.2016.37.

[11] P. Cheng, H. Xin, and L. Chen, "Utility-aware ridesharing on road networks," in *Proc. ACM Int. Conf. Manage. Data (SIGMOD)*, S. Salihoglu, W. Zhou, R. Chirkova, J. Yang, and D. Suciu, Eds. Chicago, IL, USA, May 2017, pp. 1197–1210, doi: 10.1145/3035918.3064008.

[12] J.-F. Cordeau and G. Laporte, "A Tabu search heuristic for the static multi-vehicle dial-a-ride problem," *Transp. Res. B, Methodol.*, vol. 37, no. 6, pp. 579–594, 2003.

[13] Y. Xu, Y. Tong, Y. Shi, Q. Tao, K. Xu, and W. Li, "An efficient insertion operator in dynamic ridesharing services," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Macao, China, Apr. 2019, pp. 1022–1033.

[14] J. Alonsomora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," *Proc. Nat. Acad. Sci. USA*, vol. 114, no. 3, pp. 462–467, 2017, doi: 10.1073/pnas.1611675114.

[15] Y. Zeng, Y. Tong, Y. Song, and L. Chen, "The simpler the better: An indexing approach for shared-route planning queries," *Proc. VLDB Endowment*, vol. 13, no. 13, pp. 3517–3530, Sep. 2020. [Online]. Available: http://www.vldb.org/pvldb/vol13/p3517-zeng.pdf

[16] L. Zheng, L. Chen, and J. Ye, "Order dispatch in price-aware ridesharing," *Proc. VLDB Endowment*, vol. 11, no. 8, pp. 853–865, 2018. [Online]. Available: http://www.vldb.org/pvldb/vol11/p853-zheng.pdf

[17] X. Bei and S. Zhang, "Algorithms for trip-vehicle assignment in ride-sharing," in *Proc. 32nd AAAI Conf. Artif. Intell. (AAAI) 30th Innov. Appl. Artif. Intell. (IAAI) 8th AAAI Symp. Educ. Adv. Artif. Intell. (EAAI)*, S. A. McIlraith and K. Q. Weinberger, Eds. New Orleans, LA, USA, Feb. 2018, pp. 3–9. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16583

[18] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu, "A unified approach to route planning for shared mobility," *Proc. VLDB Endowment*, vol. 11, no. 11, pp. 1633–1646, 2018. [Online]. Available: http://www.vldb.org/pvldb/vol11/p1633-tong.pdf

[19] A. Garivier and O. Cappé, "The KL-UCB algorithm for bounded stochastic bandits and beyond," in *Proc. 24th Annu. Conf. Learn. Theory*, Budapest, Hungary, vol. 19, 2011, pp. 359–376. [Online]. Available: http://proceedings.mlr.press/v19/garivier11a/garivier11a.pdf

[20] J. Wang, P. Cheng, L. Zheng, C. Feng, L. Chen, X. Lin, and Z. Wang, "Demand-aware route planning for shared mobility services," *Proc. VLDB Endowment*, vol. 13, no. 7, pp. 979–991, Mar. 2020.

[21] C. Wang, Y. Song, Y. Wei, G. Fan, H. Jin, and F. Zhang, "Towards minimum fleet for ridesharing-aware Mobility-on-Demand systems," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Vancouver, BC, Canada, May 2021, pp. 1–10, doi: 10.1109/INFOCOM42981.2021.9488862.

[22] H. Zhang and J. Zhao, "Mobility sharing as a preference matching problem," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 7, pp. 2584–2592, Jul. 2019, doi: 10.1109/TITS.2018.2868366.

[23] C. Zhang, Y. Zhang, W. Zhang, L. Qin, and J. Yang, "Efficient maximal spatial clique enumeration," in *Proc. 35th IEEE Int. Conf. on Data Eng. (ICDE)*, Macao, China, Apr. 2019, pp. 878–889, doi: 10.1109/ICDE.2019.00083.

[24] M. Danisch, O. Balalau, and M. Sozio, "Listing k-cliques in sparse real-world graphs," in *Proc. World Wide Web Conf. World Wide Web (WWW)*, P. Champin, F. Gandon, M. Lalmas, and P. G. Ipeirotis, Eds. Lyon, France, 2018, pp. 589–598, doi: 10.1145/3178876.3186125.

[25] J. Cheng, Y. Ke, A. W. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks by H*-graph," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, A. K. Elmagarmid and D. Agrawal, Eds. Indianapolis, Indiana, IND, USA, Jun. 2010, pp. 447–458, doi: 10.1145/1807167.1807217.

[26] M. N. Katehakis and A. F. Veinott, "The multi-armed bandit problem: Decomposition and computation," *Math. Oper. Res.*, vol. 12, no. 2, pp. 262–268, May 1987, doi: 10.1287/moor.12.2.262.

[27] *Geofabrik Download Server*. Accessed: Mar. 1, 2022. [Online]. Available: https://download.geofabrik.de/

[28] *Osmconvert*. Accessed: Mar. 1, 2022. [Online]. Available: https://wiki.openstreetmap.org/wiki/Osmconvert

[29] *OpenStreetMap*. Accessed: Mar. 1, 2022. [Online]. Available: https://www.openstreetmap.org/

[30] *Relation: Chengdu (2110264)*. Accessed: Mar. 1, 2022. [Online]. Available: https://www.openstreetmap.org/relation/2110264

[31] *Relation: New york (61320)*. Accessed: Mar. 1, 2022. [Online]. Available: https://www.openstreetmap.org/relation/61320

[32] *DiDi GAIA*. Accessed: Mar. 1, 2022. [Online]. Available: https://outreach.didichuxing.com/research/opendata/

[33] *TLC Trip Record Data*. Accessed: Mar. 1, 2022. [Online]. Available: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[34] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proc. 4th Annu. ACM/SIGACT-SIAM Symp. Discrete Algorithms*, V. Ramachandran, Ed. Austin, TX, USA, Jan. 1993, pp. 311–321. [Online]. Available: http://dl.acm.org/citation.cfm?id=313559.313789

[35] N. H. Wilson, R. W. Weissberg, and J. Hauser, "Advanced dial-a-ride algorithms research project," Dept. Civil Eng., Cambridge, MA, USA. Tech. Rep. R76-20, 1976.

[36] J.-F. Cordeau and G. Laporte, "The dial-a-ride problem (DARP): Variants, modeling issues and algorithms," *Quart. J. Belg., Fr. Italian Oper. Res. Societies*, vol. 1, no. 2, pp. 89–101, Jun. 2003.

[37] K. I. Wong and M. G. H. Bell, "Solution of the Dial-a-Ride problem with multi-dimensional capacity constraints," *Int. Trans. Oper. Res.*, vol. 13, no. 3, pp. 195–208, May 2006.

[38] J.-F. Cordeau, "A branch-and-cut algorithm for the dial-a-ride problem," *Oper. Res.*, vol. 54, no. 3, pp. 573–586, 2006.

[39] L. Zheng, P. Cheng, and L. Chen, "Auction-based order dispatch and pricing in ridesharing," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Apr. 2019, pp. 1034–1045, doi: 10.1109/ICDE.2019.00096.

[40] J.-J. Jaw, A. R. Odoni, H. N. Psaraftis, and N. H. M. Wilson, "A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time Windows," *Transp. Res. B, Methodol.*, vol. 20, no. 3, pp. 243–257, Jun. 1986.

[41] I. Ioachim, J. Desrosiers, Y. Dumas, M. M. Solomon, and D. Villeneuve, "A request clustering algorithm for door-to-door handicapped transportation," *Transp. Sci.*, vol. 29, no. 1, pp. 63–78, Feb. 1995.

[42] L. Häme, "An adaptive insertion algorithm for the single-vehicle dial-a-ride problem with narrow time Windows," *Eur. J. Oper. Res.*, vol. 209, no. 1, pp. 11–22, Feb. 2011.

HAN WU received the bachelor's degree from East China Normal University, China, in 2019, where he is currently pursuing the master's degree. His research interests include food delivery and spatial crowdsourcing.

YU CHEN received the bachelor's degree from East China Normal University, China, in 2019, where he is currently pursuing the master's degree. His research interests include spatial data management, ridesharing, spatial crowdsourcing, and dense subgraph.

LIPING WANG received the Ph.D. degree in computer application and technology from East China Normal University, in 2013. She is currently an Associate Professor with the Software Engineering Institute, East China Normal University. Her research interests include temporal-spatial data management and pattern recognition.

GUOJIE MA received the Ph.D. degree from the University of Technology Sydney, Australia. She is currently a Postdoctoral Research Fellow with East China Normal University. Her research interests include big data analysis for finance, fintech, and knowledge graph.