E-Ride: An Adaptive Event Driven Windowed Matching Framework in Ridesharing

YU CHEN, HAN WU, LI-PING WANG, GUOJIE MA
East China Normal University, Shanghai, China
Corresponding author: Liping Wang (e-mail: lipingwang@sei.ecnu.edu.cn).

ABSTRACT Ridesharing services aim to reduce the users’ travel cost and optimize 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. The former mainly focuses on responding quickly to the requests, and the latter focuses on meticulously enumerating request combinations to improve service quality. However, online-based methods perform poorly in 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 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 the public transit and the cab service, balancing the charging cost and travel convenience. So that, 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 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 achieve a fine-grained implementation of request matching based on the concept of a matching window to achieve 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 $\text{cost}(R)$ denote the shortest travel distance to serve all requests $r \in R$, then the saved travel distance is denoted as $\sum_{r \in R} \text{cost}([r]) - \text{cost}(R)$), 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 network, 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.

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 $\epsilon$ 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 shorter running time than the existing methods.

II. PROBLEM DEFINITION

In this section, we introduce and analyze the dynamic windowed ridesharing problem DWRP studied in this paper. We used a directed weighted graph to represent the road network, where each node in the graph represents an intersection, and each edge indicates the road between intersections. Besides, each edge in the graph is associated with a weight $\text{cost}(u, v)$, which indicates the cost to travel from $u$ to $v$. In this paper, $\text{cost}(u, v)$ shows the average travel time.

A. DEFINITIONS

Definition 1 (Request). Let $r_i = (s_i, e_i, n_i, t_i, d_i, w_i)$ denote an online request $r_i$ released at time $t_i$, which contains $n_i$ passengers departing from $s_i$ and requires to arrive at $e_i$.
before the deadline \( d_i \). \( w_i \) is the maximum tolerable waiting time for system to match.

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 rider as follows.

**Definition 2** (Matching Window). Let \( T_i = [t_i, t_i + w_i] \) denote a matching window of rider \( r_i \), which starts from the release time \( t_i \) to the maximum waiting time \( t_i + w_i \).

We denote the request \( r_i \) as available for \( r_n \) if and only if \( T_n \cap T_i = \emptyset \) (i.e., \( |T_n - T_i| \leq \min(w_n, w_i) \)). For each vehicle \( v_j \), it may be assigned with several numbers of mutually available requests \( R_j \) simultaneously. Therefore we also need to plan a route \( S_j \) for each vehicle \( v_j \), which consists of a sequence of pickup and drop-off locations for the requests \( r \in R_j \). We define the route for each vehicle as follows.

**Definition 3** (Route). Given a set of \( m \) requests \( R \), let \( S = (o_1, \ldots, o_{2m}) \) denote the route where \( o_x \) is the source location \( s_i \) or destination \( e_i \) of request \( r_i \in R \).

We mark a route as feasible if and only if it satisfies the following three constraints:

- **Sequential constraint.** The pickup location \( s_i \) of request \( r_i \in R_i \) should be located before the drop-off location \( e_i \) in the feasible route.

- **Capacity constraint.** At any location \( o_x \in S \), the total number of riders on the vehicle should not exceed the capacity of the vehicle.

- **Deadline constraint.** For any location \( o_x \in S \), \( \sum_{k=1}^{x} \text{cost}(o_{k-1}, o_k) \leq ddl(o_k) \), where \( ddl(o_k) \) satisfied following equation \( (1) \) for different location type (source or destination).

\[
\text{ddl}(o_k) = \begin{cases} 
  e_i - \text{cost}(s_i, e_i), & \text{if } o_x \text{ is source } \space e_i, \\
  e_i, & \text{otherwise} 
\end{cases} 
\]

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

**Definition 4** (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.

In this paper, we refer to the following unified cost \( UC \) defined in [18] as the optimization utility function.

\[
UC(R, w) = \alpha \sum_{w_i \in W} \mu(S_{w_i}) + \beta \sum_{r_i \in R} \text{cost}(s_i, e_i) \tag{2}
\]

\[
\mu(S_{w_i}) = \sum_{o_x \in S_{w_i}} \text{cost}(o_{x-1}, o_x) \tag{3}
\]

**Hardness of DWRP.** Dynamic ridesharing problem has been proved NP-hard in many existing works [9], [11], [18], [20], thus intractable. Moreover, Tong et al. proved that there is no polynomial-time algorithm with a constant competitive ratio for dynamic ridesharing problem [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 [13]–[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.

Firstly, we retrieve the advent requests \( R^- \), which expire before the next trigger time \( t + \tau \) (line 3). Then we tried to select request groups for each vehicle \( w \in W \) (line 4-10). Specifically, we first enumerate request groups of size up to the vehicle capacity constraint \( c \) (line 5-8). After that, we select the group \( g^* \in G \) with the minimum unified cost and plan a route \( S_j \) for \( w_j \) to serve \( r \in g^* \) (line 9-10).

Complexity Analysis. For each vehicle \( w \), we need to enumerate to up to \( \sum_i C_n^k \) request groups. Since the capacity constraint \( c \ll n \) in practice, \( \sum_i C_n^k \) can also be noted as \( O(n^k) \). Then, to identify whether the riders in each group are shareable, we need to examine up to \( A_{2c}^2 \) candidate routes \( (a_1, \ldots, a_2) \), and check whether each route satisfies the deadline constraint in linear time. Therefore, the time complexity of the Brute-Force algorithm is \( O(m \times n^k \times (2c)! \times 2c) \).

### 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 on the enumeration of candidate request groups, which costs up to \( O(n^k) \) times. Therefore, in this section, we propose a well-tailored \( \varepsilon \)-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

Shareable request group enumeration is a fundamental operator in batch-based methods. Therefore, to optimize the efficiency of the shareable request group enumeration, we first define the following shareability graph for visualizing the shareable relationships between requests intuitively.

**Definition 5** (Shareability Graph). Given a set of requests \( R, SG = (R, E) \) denotes the shareability graph of \( R \), where \( e = (r_i, r_j) \in E \) reflects that request \( r_i \) and \( r_j \) are shareable.

Here, clique \([21]–[23]\) is an extensively studied subgraph structure. And \( k \)-clique is a subset of \( k \) nodes in the graph that satisfies any two distinct nodes in the \( k \)-clique are adjacent in graph theory. With the shareability graph, we have the following Theorem III.1 for enumerating those request groups that form a \( k \)-clique in the shareability graph rather than an arbitrary enumeration, which helps to reduce the search space by pruning infeasible groups.

**Theorem III.1.** Given a feasible route \( S \) for \( k \) requests, the corresponding nodes of these \( k \) requests form a \( k \)-clique in the shareability graph.

**Proof.** We will prove it by a contradiction. Suppose a feasible route \( S \) of \( k \) requests whose corresponding nodes did not form a \( k \)-clique in the shareability network. Thus, at least two nodes \( r_a \) and \( r_b \) that are not connected. We derive the subroute \( S' \) from \( S \) by removing location \( o_e \) except the source and destination of \( r_a \) and \( r_b \). Since removing existing waypoints reduces the detour, the subroute \( S' \) is also a feasible route. According to the definition of the shareability graph, there must exist an edge between \( r_a \) and \( r_b \), which contradicts our assumption. In summary, these \( k \) requests form a \( k \)-clique in the shareability graph.

With the Theorem III.1, we have that a shareable request group of size \( k \) in the shareability graph must constitute a \( k \)-clique.

**Dynamic update of the shareability graphs.** 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 constraint of deadline, 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 shareability 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 \( \varepsilon \)-RIDE FRAMEWORK

We first briefly introduce the essential parts of our \( \varepsilon \)-Ride framework, as illustrated in Figure 4.

- **Event Handler.** The \( \varepsilon \)-Ride framework mainly works on the event model, which will make corresponding 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 if only if \( r_i \) can provide optimization (lower unified cost) for some existing groups.

- **Indexing Structure.** Since vehicles are always moving in 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 of candidate vehicles in the assignment phase.

The brief processing flow of \( E \)-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 3 and update it to the state graph \( MG \). And for each unit time \( t \), we check if requests in \( SG \) exceed the matching window \( T_e \) (line 7-9). If the request needs dispatch immediately, we will perform the request group dispatching by triggering the leave handler (Algorithm 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,

**Algorithm 2: Event-driven Windowed Matching**

**Input:** The request arrival sequence \( \sigma \) with a matching window \( T \) and a vehicle set \( W \).

**Output:** An updated route set \( S \) for vehicles \( W \).

1. \( SG \leftarrow \) initialize a empty shareability graph
2. **foreach** arrival request \( r_{\sigma(i)} \) according to \( \sigma \) do
3.  
4.  
5.  
6.  
7. **foreach** request \( r \in SG \) do
8.  
9.  

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 in-neighbors, and \( N(r_i) \) for all undirected neighbors.

**Definition 6** (State Graph). Let the mixed graph \( MG = (R, E, A) \) denotes the state graph on the shareability graph \( SG = (R, E) \), where for each of the node \( r_i \in R \) with its in-neighbors \( N^{-}_M(r_i) \) indicates the request group stored in the state graph.

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

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

---

**FIGURE 4:** An Overview of the \( E \)-Ride Framework
node in the sharability 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 \( r \) 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 7 once the given request pairs share a same seller node.

**Definition 7 (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 2.** Consider the state graph in the middle of Figure 5 the seller node \( r_2 \) together with the buyer nodes \( r_1 \) and \( r_4 \) form the request group \( Q^*_1 \), and the seller node \( r_2 \) also form the request group \( Q_1 \) with the buyer nodes \( r_3 \) and \( r_4 \). Therefore, we say \( Q_1 \) and \( Q^*_1 \) are conflict groups. And \( r_3 \) is the conflict node for group \( Q^*_1 \).

The \( E \)-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 \( G \) implies that the requests \( r \in G \) 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 arrived request \( r_i \) to an existing request group \( G \);
2. Take some requests from the existing group to generate a new request group with the arrived request.

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. That is, 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 request \( r \in Q_i \) 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 \( Q_i \) with \( r_i \) 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 III.1, we only need to enumerate the \( k \)-cliques that contain \( r_i \). (line 3-10). We work differently depending on the current 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 \( Q \) sorted by the unified cost (line 4-5). Note that we only take the clique \( Q \), which satisfied \( U(Q) > U(Q_p) \). Suppose neighbor \( r_p \) is the buyer in the state graph. In that case, we try to take \( r_p \) from

---

**Algorithm 3: Arrival Event Handler**

**Input:** The arrived request \( r_i \), state graph \( MG = \{R, E, A\} \) 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. \( Q \leftarrow \) initialize a priority queue ordered by unified cost
3. foreach \( r_p \in N(r_i) \) do
   4. if \( L_{MG}(r_p) \) is seller then
      5. \( Q[r_p] \leftarrow \{Q|Q \in \text{CliqueEnum}(r_i \cup Q_p) \land (r_i, r_p) \in Q\} \)
   6. else if \( L_{MG}(r_p) \) is buyer then
      7. if \( U_1(r_i, r_p) + U_2(Q_p \setminus \{r_p\}) > U(Q_p) \) then
         8. \( Q[r_i] \leftarrow Q[r_i] \cup \{r_i, r_p\} \)
   9. else \( \Rightarrow \) None Label
      10. \( Q[r_i] \leftarrow Q[r_i] \cup \{r_i, r_p\} \)
11. if \( |Q| > 0 \) then
   12. \( \{r_p, Q^*_p\} \leftarrow \) retrieve group with maximum \( UC \) in \( Q \)
   13. \( L_{MG}(r_i) \leftarrow \) buyer
   14. \( A \leftarrow A \cup \{r_i, r_p\} \)
   15. if \( r_p \) is seller then
      16. \( Q^*_p \leftarrow Q_p \setminus Q^*_p \)
      17. clear labels of \( r \in Q^*_p \) and related edges from \( MG \)
      18. foreach \( r \in Q^*_p \) do
         19. \( \Rightarrow \) 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

---

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
the group \( Q_p \) it currently belongs, and forms a new group \( Q_i = \{ r_i, r_p \} \), because the enumeration among the requests in \( Q_p \) has already been checked in the enumeration of the seller in \( Q_p \) (line 6-8). The newly generated group \( Q_i \) to be considered as a valid candidate request group only when the unified cost satisfied \( U(Q_i) + U(Q_p \setminus Q_i) > U(Q_p) \) (line 7-8). And if \( r_p \) is not yet labeled in \( MG \), then this 2-clique can be directly considered a valid candidate request group (line 9-10). After that, we take the clique \( Q_i^* \) with the minimum unified cost from the priority queue \( Q \) 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 \( Q_i^* \leftarrow Q_p \setminus Q_i^* \) need to be updated due to the unavailability of \( r_p \) (line 16). Therefore, we clear the original labels from \( MG \) and reinitialize the conflict nodes by Algorithm \( \text{Algorithm 3} \) 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 \( \text{Algorithm 4} \) to complete 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 3. 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 3. We first iterate over all neighbors of \( r_i, N(r_i) \), and try to enumerate the optimal request group. Since \( a_2 \) is currently a seller, we enumerate all cliques containing \( a_2 \) and \( a_3 \), i.e., \( \{ a_2, a_3 \}, \{ a_2, a_3, a_1 \}, \{ a_2, a_3, a_1, a_4 \} \), and we add them to the priority queue \( Q \) with their corresponding unified cost. Then we visit the neighbor \( a_3 \), and because the label of \( a_3 \) is buyer, we check the sum of the unified cost of the group \( Q_1 \setminus \{ a_1 \} \) after the loss of \( a_1 \) from the original group \( Q_1 \) with the newly generated group \( \{ a_2, a_3 \} \). We add the new request group to \( Q \) because \( U(\{ a_2, a_3 \}) + U(Q_1 \setminus \{ a_1 \}) > U(Q_1) \). The operations on the remaining neighbors are omitted here. After that, we pick the request group \( Q_i^* = \{ r_i, r_2, r_3 \} \) with the minimum unified cost from the priority queue \( Q \) and update the label of \( r_i \) to buyer with its associated edges to \( MG \). Since group \( Q_i^* \) is enumerated based on the neighbor \( r_2 \) labelled as seller, so the conflict node \( r_3 \in Q_1 \setminus Q_i^* \) should be updated by Algorithm \( \text{Algorithm 3} \) for a new group. Finally, we derive the state graph as shown in right of Figure 3.

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 has a significant impact on 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) 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 \( N_t(a_i) \), which will decrease as the number of observations \( N_t(a_i) \) of \( a_i \) increases. Thus, in the UCB methods, we always greedily choose the action \( a_i^* \) with the maximum upper confidence bound in round \( t \) as shown in Equation (6).

\[
\begin{align*}
    a_i^* = \arg \max_{a_i} \left\{ \hat{Q}_t(a_i) + \hat{U}_t(a_i) \right\}
\end{align*}
\]
where above, we have the following adaptive matching algorithm maximum allowed matching time; the parameter travel time saved by the request group; the right part analyzes KL divergence, which satisfies a uniformly better expected re-

The service quality, we define the following reward function online iterations.

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 \( P \) of the platform (line 2). Once the recommendation \( a_i^* \) of \( P \) 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 \( P \) (line 5-6). Finally, we plan the route \( S_j \) for the selected vehicle \( w_i^* \) and remove the request \( r_i \) with related edges from the state graph \( MG \) (line 7-8).

V. EXPERIMENTS

A. EXPERIMENTAL SETUP

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 \( ^{25} \) and segmented by Osmconverter \( ^{26} \) with city boundaries on OpenStreetMap \( ^{27} \) for CHD \( ^{28} \) and NYC \( ^{29} \), respectively. In addition, we also carefully trimmed the road networks according to the distribution boundaries of rider sources and destinations so that there are fewer irrelevant regions as possible. The weight associated

\[
d(p, q) = KL(p, q) = p \log \frac{p}{q} + (1 - p) \log \frac{1 - p}{1 - q}
\]
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 riders 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 \( d_i = t_i + \gamma \cdot \text{cost}(r_i) \), which is a commonly used configuration in many existing works \[11\], \[20\], \[30\].

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 for testing. These datasets are available on Didi GAIA \[31\] and NYC Official Website \[32\], respectively. The detailed experiment-related parameters are shown in Table 3 (default parameters are in bold).

**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 \[33\]. 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.

**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** \[15\]. 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.

### 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 deadline parameter \( \gamma \) | 1.2, 1.3, 1.5, 1.8, 2.0 |
| the penalty coefficient \( p_v(\beta) \) | 2.5, 10, 20, 30 |

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
window based on the KL-UCB policy. Additionally, the reward function’s adaptation parameter $\epsilon$ has been fine-tuned for better service quality by default.

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 $p$, 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 unit time and fare per unit distance. 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

Effect of the number of vehicles. Figure 8 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 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).

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% ~ 34.61% and 2.84% ~ 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× ~ 78.3× faster than BF on CHD. And on the NYC dataset, the AWM algorithm performs even 401.52× faster than the BF algorithm as shown in Figure 7(f).

**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., γ = 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 compared to the EGWM, where the AWM is 1.98× and 5.95× faster than EGWM on two datasets when γ = 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×. Note that BF performs inefficiently on two datasets with γ ≥ 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 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 γ = 2.0 and NYC with γ ≥ 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.

**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 \( C_{10}^6 \) different request groups), the BF algorithm cannot finish within the given time limit when c ≥ 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× ~ 147.85× faster than BF.

**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.

**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 overall 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**

The ridesharing problem can be reduced to a variant of the Dial-a-Ride (DARP) problem [34], [35], aiming to plan the vehicle routes and trip schedules for n riders who specified source and destination with practical constraints. The existing works on ridesharing services are categorized as static and dynamic, depending on whether all requests are known in advance. Most of the existing works [36], [37] on DARP are in static environment. For the dynamic ridesharing problem, the existing solutions are mainly in online mode [5], [6], [13], [18] or batch mode [14]–[16], [38].

In online mode, insertion [39] is the state-of-the-art operation of the existing works [40], [41] in route planning, which inserts the pickup and drop-off locations of a new request into the vehicle’s schedule without reordering. Tong et al. [18] proposed an insertion method based on dynamic programming, which checks the constraints in constant time and dispatches requests in linear time. Huang et al. proposed the structure of kinetic tree in [4] to trace all feasible routes for each vehicle to reduce the total drive distance. The kinetic tree always provides the optimal vehicle schedule whenever the schedule changes (i.e., a new rider arrives).

Batch-based algorithms partition the requests into groups and assign groups to their appropriate vehicles. Alonso-Mora et al. [14] proposed RTV-Graph to model the relationship and constraints among requests, trips, and vehicles, where trips are the groups composed of shareable requests. The RTV-Graph minimizes the utility function by linear programming to allocate between vehicles and trips. The time cost for enumerating trips in building RTV-Graph grows exponentially.
Zeng et al. [15] proposed an index called additive tree for pruning the infeasible groups during the group enumeration and greedily chose the most profitable request group for each vehicle.

The structure of the shareability graph is intuitive, and thus some similar structures have been used in some existing works, which are designed in different ways for the vehicle-request matching. Wang et al. [42] formulate a tree cover problem to serve urban demands with as few vehicles as possible. Alonso-Mora et al. [14] optimally assign vehicles to shareable groups of customers through linear programming. Zhang et al. [43] formulate the passenger matching as a monopartite matching problem and solve it by Irving-Tan algorithm.

VII. CONCLUSION

In this paper, we study the dynamic ridesharing problem with a personalized matching window. Concretely, we first proposed an event-driven framework, $\varepsilon$-Ride, which maintains the request groups through a state graph extended by the shareability network. Then, we propose an efficient request group enumeration strategy based on the k-clique in the shareability graph, which helps to 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 experimental 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. W. Calvo, “Shared mobility systems: an updated survey,” Ann. Oper. Res., vol. 271, no. 1, pp. 105–126, 2018.

[2] S. C. Ho, W. Szeto, Y.-H. Kuo, J. M. Leung, M. Petering, and T. W. Tou, “A survey of dial-a-ride problems: Literature review and recent developments,” Transportation Research Part B: Methodological, vol. 111, pp. 395–421, 2018.

[3] “[online] Didi Chuxing.” [https://www.didiglobal.com/]. 2018.

[4] “[online] uuberPOOL.” [https://www.uuber.com/]. 2018.

[5] S. Ma, Y. Zheng, and O. Wolfson, “T-share: A large-scale dynamic taxi ridesharing service,” in 29th IEEE International Conference on Data Engineering, ICDE 2013, pp. 410–421, 2013.

[6] Y. Huang, F. Bastani, R. Jin, and X. S. Wang, “Large scale real-time ridesharing with service guarantee on road networks,” PVLDB, vol. 7, no. 14, pp. 2017–2028, 2014.

[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 Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS 2016, pp. 3:1–3:10, 2016.

[8] M. Asghari and C. Shahabi, “An on-line truthful and individually rational pricing mechanism for ride-sharing,” in Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS 2017, Redondo Beach, CA, USA, November 7-10, 2017, pp. 7:1–7:10, 2017.

[9] B. Cici, A. Markopoulou, and N. Laoutaris, “Designing an on-line ride-sharing system,” in Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 60:1–60:4, 2015.

[10] S. Yeung, E. Miller, and S. Madria, “A flexible real-time ridesharing system considering current road conditions,” in IEEE 17th International Conference on Mobile Data Management, MDM, pp. 186–191, 2016.

[11] P. Cheng, H. Xin, and L. Chen, “Utility-aware ridesharing on road networks,” in Proceedings of the 2017 ACM International Conference on Management of Data, pp. 1197–1210, ACM, 2017.

[12] J.-F. Cordeau and G. Laporte, “A tabu search heuristic for the static multi-vehicle dial-a-ride problem,” Transportation Research Part B: Methodological, 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 35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019, pp. 1022–1033, IEEE, 2019.

[14] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, “On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment,” Proc. Natl. Acad. Sci. USA, vol. 114, no. 3, pp. 462–467, 2017.

[15] Y. Zeng, Y. Tong, Y. Song, and L. Chen, “The simpler the better: An indexing approach for shared-route planning queries,” Proc. VLDB Endow., vol. 13, no. 13, pp. 3517–3530, 2020.

[16] L. Zheng, L. Chen, and J. Ye, “Order dispatch in price-aware ridesharing,” Proceedings of the VLDB Endowment, vol. 11, no. 8, pp. 853–865, 2018.

[17] X. Bei and S. Zhang, “Algorithms for trip-vehicle assignment in ridesharing,” in Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18) (S. A. McIlraith and K. Q. Weinberger, eds.), pp. 3–9, AAAI Press, 2018.

[18] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu, “A unified approach to route planning for shared mobility,” PVLDB, vol. 11, no. 11, pp. 1633–1646, 2018.

[19] A. Garivier and O. Cappé. “The kl-ucb algorithm for bounded stochastic bandits and beyond,” Mathematics, 2011.

[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 Endow., vol. 13, no. 7, pp. 979–991, 2020.

[21] C. Zhang, Y. Zhang, W. Zhang, L. Qin, and J. Yang. “Efficient maximal spatial clique enumeration,” in 35th IEEE International Conference on Data Engineering, ICDE 2019, pp. 878–889, IEEE, 2019.

[22] M. Danisch, O. Balalau, and M. Sozio. “Listing k-cliques in sparse real-world graphs,” in Proceedings of the 2018 World Wide Web Conference on Wide World Web (P. Champin, F. Gandon, M. Lalmis, and P. G. Ipeirotis, eds.), pp. 589–598, ACM, 2018.

[23] J. Cheng, Y. Ke, A. W. Fu, J. X. Yu, and L. Zhu. “Finding maximal cliques in massive networks by $h^*$-graphs,” in Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010 (A. K. Elmagarmid and D. Agrawal, eds.), pp. 447–458, ACM, 2010.

[24] M. N. Katehakis and A. F. Veinott. “The multi-armed bandit problem: Decomposition and computation,” Math. Oper. Res., vol. 12, pp. 262–268, 1987.

[25] “[online] Geofabrik.” [https://download.geofabrik.de/]. 2022.

[26] “[online] Osmconverter.” [https://wiki.openstreetmap.org/wiki/Osmconvert]. 2022.

[27] “[online] OpenStreetMap.” [https://www.openstreetmap.org/]. 2022.

[28] “[online] Relation: Chengdu.” [https://www.openstreetmap.org/relation/2110264]. 2022.

[29] “[online] Relation: New York City.” [https://www.openstreetmap.org/relation/61320]. 2022.

[30] “[online] Relation: TLC Trip Record Data.” [https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page]. 2022.

[31] “[online] Data source: Didi Chuxing GAIA Initiative.” [https://outreach.didichuxing.com/research/opensdata]. 2022.

[32] “[online] Data source: TLC Trip Record Data.” [https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page]. 2022.

[33] P. N. Yianilos. “Data structures and algorithms for nearest neighbor search in general metric spaces,” in Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1993.
[34] N. H. Wilson, R. Weissberg, B. Higonnet, and J. Hauser, “Advanced dial-a-ride algorithms,” tech. rep., 1975.

[35] J.-F. Cordeau and G. Laporte, “The dial-a-ride problem (darp): Variants, modeling issues and algorithms,” Quarterly Journal of the Belgian, French and Italian Operations Research Societies, vol. 1, no. 2, pp. 89–101, 2003.

[36] K.-I. Wong and M. G. Bell, “Solution of the dial-a-ride problem with multi-dimensional capacity constraints,” International Transactions in Operational Research, vol. 13, no. 3, pp. 195–208, 2006.

[37] J.-F. Cordeau, “A branch-and-cut algorithm for the dial-a-ride problem,” Operations Research, vol. 54, no. 3, pp. 573–586, 2006.

[38] L. Zheng, P. Cheng, and L. Chen, “Auction-based order dispatch and pricing in ridesharing,” in 2019 IEEE 35th International Conference on Data Engineering (ICDE), pp. 1034–1045, IEEE, 2019.

[39] J.-J. Jaw, A. R. Odoni, H. N. Psaraftis, and N. H. Wilson, “A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows,” Transportation Research Part B: Methodological, vol. 20, no. 3, pp. 243–257, 1986.

[40] I. Ioachim, J. Desrosiers, Y. Dumas, M. M. Solomon, and D. Villeneuve, “A request clustering algorithm for door-to-door handicapped transportation,” Transportation science, vol. 29, no. 1, pp. 63–78, 1995.

[41] L. Häme, “An adaptive insertion algorithm for the single-vehicle dial-a-ride problem with narrow time windows,” European Journal of Operational Research, vol. 209, no. 1, pp. 11–22, 2011.

[42] C. Wang, Y. Song, Y. Wei, G. Fan, H. Jin, and F. Zhang, “Towards minimum fleet for ridesharing-aware mobility-on-demand systems,” in 40th IEEE Conference on Computer Communications, INFOCOM 2021, Vancouver, BC, Canada, May 10-13, 2021, pp. 1–10, IEEE, 2021.

[43] H. Zhang and J. Zhao, “Mobility sharing as a preference matching problem,” IEEE Trans. Intell. Transp. Syst., vol. 20, no. 7, pp. 2584–2592, 2019.

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 in the Software Engineering Institute, East China Normal University. Her research interests include temporal-spatial data management and pattern recognition.

Yu Chen received the bachelor’s degree from East China Normal University, China, in 2019. He is currently pursuing a master’s degree at the East China Normal University, China. His research interests include spatial data management, ridesharing, spatial crowdsourcing, dense subgraph.

Han Wu received the bachelor’s degree from East China Normal University, China, in 2019. He is currently pursuing a master’s degree at the East China Normal University, China. His research interests include food delivery, spatial crowdsourcing.

GuoJie Ma is currently a Postdoc research fellow with East China Normal University. She received the Ph.D. degree from the University of Technology, Sydney, Australia. Her research interests include big data analysis for finance, fintech and knowledge graph.