Utility-Aware Ridesharing on Road Networks

Peng Cheng
Hong Kong University of Science and Technology
Hong Kong, China
pchengaa@cse.ust.hk

Hao Xin
Hong Kong University of Science and Technology
Hong Kong, China
hxinaa@cse.ust.hk

Lei Chen
Hong Kong University of Science and Technology
Hong Kong, China
leichen@cse.ust.hk

ABSTRACT

Ridesharing enables drivers to share any empty seats in their vehicles with riders to improve the efficiency of transportation for the benefit of both drivers and riders. Different from existing studies in ridesharing that focus on minimizing the travel costs of vehicles, we consider that the satisfaction of riders (the utility values) is more important nowadays. Thus, we formulate the problem of utility-aware ridesharing on road networks (URR) with the goal of providing the optimal rider schedules for vehicles to maximize the overall utility, subject to spatial-temporal and capacity constraints. To assign a new rider to a given vehicle, we propose an efficient algorithm with a minimum increase in travel cost without reordering the existing schedule of the vehicle. We prove that the URR problem is NP-hard by reducing it from the 0-1 Knapsack problem and it is unlikely to be approximated within any constant factor in polynomial time through a reduction from the DENS k-SUBGRAPH problem. Therefore, we propose three efficient approximate algorithms, including a bilateral arrangement algorithm, an efficient greedy algorithm and a grouping-based scheduling algorithm, to assign riders to suitable vehicles with a high overall utility. Through extensive experiments, we demonstrate the efficiency and effectiveness of our URR approaches on both real and synthetic data sets.

Keywords

Ridesharing; Task Scheduling

1. INTRODUCTION

Recently, companies (e.g., Uber [7] and Lyft [3]) have been allowing their users to enjoy a ridesharing service through sharing their locations and arranging rides with other passengers within minutes with dual goals to alleviate the public traffic congestion and to monetarily benefit drivers and riders. In a report [4], apps (e.g., Uber and Lyft) saw a surge in riders willing to undertake ridesharing with strangers in 2016. Moreover, several companies [4, 8] launched unlimited ride packages – \$69 for a week and \$255 for a month, to allow customers to freely use ridesharing services for a registered period of time. Consequently, riders' levels of satisfaction are mainly determined by their preferences of vehicles

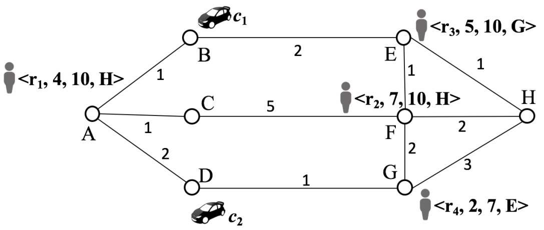


Figure 1: An Ridesharing Example.

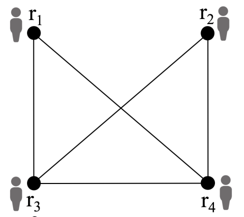


Figure 2: Social Connections of Riders.

Rider	Vehicle	Utility Value
r_1	c_1	0.2
r_1	c_2	0.4
r_2	c_1	0.6
r_2	c_2	0.3
r_3	c_1	0.2
r_3	c_2	0.8
r_4	c_1	0.2
r_4	c_2	1.0

Table 1: Matrix of Utility Values.

(e.g., the brand or model), the drivers (e.g., the gender or habits), the other riders (e.g., interests) and the lengths of trips (e.g., detours). For example, a female rider may prefer a vehicle with a female driver for safety in the late evening. A sports-lover may be willing to taking a vehicle with other riders who share similar tastes. In addition, riders usually prefer trips with less detours (e.g., the actual length of a ridesharing trip is close to that of the shortest path). Subsequently, a clear trend for such companies is to promote their businesses through maximizing the satisfaction levels of the customers.

Following this trend, in this paper, we consider a practical problem in ridesharing, namely *utility-aware ridesharing on road networks* (URR), which assigns riders to available vehicles to maximize their overall satisfaction (evaluated with *rider utility*) subject to the constraints of vehicles' capacities and riders' deadlines. In the sequel, we illustrate the URR problem with a running example.

Example 1. ***Utility-Aware Ridesharing on Road Networks Example.*** *In this example of the utility-aware ridesharing problem, assume that four riders, $r_1 \sim r_4$, and two vehicles, c_1 and c_2 are on a road network, as shown in Figure 1. Specifically, there are 8 nodes in the road network indicating 8 locations, and the numbers on the edges represent the travel costs. The capacities of two vehicles are both 2. Each vehicle is currently located at a node (e.g., vehicle c_1 locates at node B). For each rider, he/she is located at his/her current location and the 4-tuple near to him/her follows the pattern $\langle$rider id, deadline of pickup, deadline of delivery, destination$\rangle$. For example, rider r_1 is located at A and wants to go to H. He/she desires to be picked up before 4 and delivered to H before 10. Table 1 lists the preference value (vehicle-related utility) of each rider towards different vehicles. In addition, as Figure 2 shows, we know the social connections between the four riders based on their so-*

SIGMOD'17, May 14-19, 2017, Chicago, IL, USA
© 2017 ACM. ISBN 978-1-4503-4197-4/17/05. . . \$15.00
DOI: http://dx.doi.org/10.1145/3035918.3064008

cial media accounts (e.g., Facebook) or their ridesharing history records (e.g., taking the same car).

The riders send their requests to the ridesharing system server. Then, subject to the constraint of riders' deadlines and the vehicles' capacities, the server assigns riders to vehicles to maximize the overall utility of riders, which means the riders' preferences are best satisfied. As introduced in Section 2.4, one rider's utility consists of three components: vehicle-related utility, rider-related utility and trajectory-related utility. Here, we assume that the rider's utility is the average of the three components.

With spatial-temporal and capacity constraints (e.g., deadline of pickup or delivery of riders and the capacity of vehicles), one possible solution is to assign the schedule $\{r_1^+, r_3^+, r_1^-, r_3^-\}$ *to vehicle* c_1 *and* $\{r_4^+, r_2^+, r_4^-, r_2^-\}$ *to vehicle* c_2*. Here a schedule is an event sequence for a vehicle to perform. For example, the schedule* $\{r_1^+, r_3^+, r_1^-, r_3^-\}$ *means pick up* r_1 *at node* A *then pick up* r_3 *at node* E*. Next, deliver rider* r_1 *to node* H*, then deliver rider* r_3 *to node* G*. Calculated with Equation 1, the utility value of rider* r_1 *taking vehicle* c_1 *is* $0.4208 (= \frac{1}{3}(0.2 + 0.25 \times 0.25 + 1))$ *and this possible solution leads to an overall utility value of* 1.6894*. However, the optimal solution is to assign schedule* $\{r_1^+, r_2^+, r_1^-, r_2^-\}$ *and* $\{r_4^+, r_4^-, r_3^+, r_3^-\}$ *to* c_1 *and* c_2*, respectively. As a result, the optimal overall utility value is* 2.5628*.*

As shown in the above example, in this paper, we study the URR problem, which assigns the riders to the most suitable vehicles to maximize the overall utility of riders, subject to the constraints of vehicle capacities (e.g., each vehicle takes a limited number of riders) and the deadlines of riders (e.g., a rider should be picked up before the specified pickup deadline). Existing studies [19, 20, 25] on ridesharing mainly focus on real-time matching and scheduling riders to vehicles to minimize the overall travel costs of vehicles subject to the spatial-temporal, monetary or capacity constraints. However, no studies on ridesharing have focused on the satisfaction of the riders, thus the existing solutions cannot be directly applied to solve the URR problem.

In this paper, we first prove that our URR problem is NP-hard by reducing it from the *knapsack problem* [34] and unlikely to be approximated within any constant factor in polynomial time through a reduction from the DENS k-SUBGRAPH problem [17]. As a result, the URR problem is not tractable and it is very challenging to achieve the optimal solution. Therefore, we propose effective heuristic approaches, including *bilateral arrangement*, *efficient greedy* and *grouping-based scheduling*, to tackle the URR problem by efficiently computing schedules with high overall utilities satisfying the constraint of riders' deadlines and vehicles' capacities.

Specifically, we make the following contributions:

- We propose a new problem, called the *utility-aware ridesharing on road networks* (URR), in Section 2, subject to the constraints of vehicles' capacities and riders' deadlines, and we prove that the URR problem is NP-hard in Section 2.6.
- We propose an efficient algorithm to allocate one rider to a given vehicle without reordering its existing schedule such that the travel cost increase is minimized in Section 3.
- We propose three novel heuristic algorithms, namely bilateral arrangement, efficient greedy and grouping-based scheduling approaches, to tackle URR in Sections 4, 5, and 6, respectively.
- We have conducted extensive experiments on real and synthetic data sets, to show the efficiency and effectiveness of our URR approaches in Section 7.

In addition, the remaining sections of the paper are arranged as follows. We review and compare previous studies on ridesharing in Section 8 and conclude the work in Section 9.

2. PROBLEM DEFINITION

In this section, we present the formal definition of the utility-aware ridesharing, in which we assign riders to the most suitable vehicles under the constraints of the capacities of vehicles and valid time windows of riders.

We use a graph $G = \langle V, E \rangle$ to represent a road network that consists of a vertex set V and an edge set E. Each edge, $(u, v) \in E$ $(u, v \in V)$, is associated with a weight $cost(u, v)$ indicating the travel cost from vertex u to v through edge (u, v). Here the travel cost can be defined as the travel time or the travel distance. When the speeds of vehicles are known, they can be converted from one to another. Here we do not differentiate between them and use travel cost consistently in the rest of this paper.

2.1 Time-Constrained Rider

Definition 1. (Time-Constrained Riders) Let $R = \{r_1, r_2, ..., r_m\}$ be a set of m riders. Each rider r_i submits his/her request q_i to the server, which is associated with a source location s_i, destination location e_i, a pickup deadline rt_i^- and a drop off deadline rt_i^+.

In practice, a rider r_i submits a ride request q_i to notify the server that he/she wants to be picked up at location s_i and to be dropped off at location e_i. To guarantee the service, the rider can also specify a pickup deadline rt_i^-, which indicates the latest time he/she prefers to be picked up, and a drop off deadline rt_i^+, which indicates the deadline that he/she wants to be sent to the destination e_i. Here, the two timestamps of a request q_i should have a relationship in $rt_i^- < rt_i^+$ to enable the server to arrange a suitable vehicle and that the vehicle can complete the trip.

2.2 Dynamically Moving vehicles

Definition 2. Let $C = \{c_1, c_2, ..., c_n\}$ be a set of n moving vehicles that prefer to provide ridesharing services. Each vehicle c_j is currently at location $l(c_j)$ and is associated with a capacity a_j.

From the definition of vehicles, at any time the number of the riders in any vehicle c_j should not exceed its capacity a_j (i.e., not including the driver). Moreover, for each rider r_i assigned to vehicle c_j, we indicate the utility of rider r_i as $\mu(r_i, c_j)$.

2.3 Valid Scheduling

Definition 3. Let a schedule $S_j = \{x_1, x_2, ..., x_w\}$ be a sequence of events for vehicle c_j, where each event x_k is to pickup or deliver some rider r_i at location $l(x_k)$ $(= s_i$ or $e_i)$.

A schedule S_j is valid iff for any rider r_i assigned to vehicle c_j, the pickup event is before its dropoff event in S_j; vehicle c_j can satisfy the deadlines of the assigned riders; and the number of riders in vehicle c_j does not exceed its capacity a_j at any time.

Here we assume that vehicles always take the shortest path from one location to another. A given schedule S_j of vehicle c_j consists of a list TR_j of consecutive trajectories tr_k, where tr_k is the kth trajectory of vehicle c_j from location $l(x_k)$ to location $l(x_{k+1})$. We note the travel cost of trajectory tr_k as $cost(tr_k)$.

2.4 Utility Value

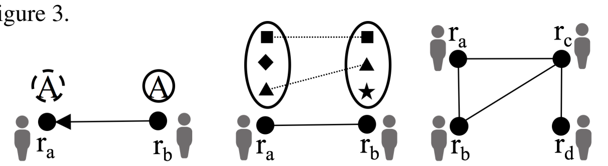
Here we first discuss the utility value and formally define it. Nowadays, being involved in ridesharing is not just about saving money but rather for a better user experience, as unlimited rides packages enable them to pay for the ridesharing service weekly or monthly with constant fees no matter what the summation of travel distances are. The user experiences of the riders in ridesharing will be affected by the brand and model of the cars, the service level of the drivers, and the interaction with other riders. Then, the utility value of a rider r_i to a vehicle c_j comprises of three components: *vehicle-related utility* $\mu_v(r_i, c_j)$, *rider-related utility* $\mu_r(r_i, c_j)$, and *trajectory-related utility* $\mu_t(r_i, c_j)$. Thus, we have:

$$\mu(r_i, c_j) = \alpha\cdot\mu_v(r_i, c_j)+\beta\cdot\mu_r(r_i, c_j)+(1-\alpha-\beta)\cdot\mu_t(r_i, c_j), \quad (1)$$

where $\alpha, \beta \in [0, 1]$ are balancing parameters and $\alpha + \beta \leq 1$. These two balancing parameters represent the preferences of riders towards the three utility components and are configured by the riders or the ridesharing system.

Note that other factors, such as the travel distances of the empty vehicles, the sceneries along the trips and so on, may also affect the utility of riders. To keep this paper clear and focused on the main factors (i.e., the three components in Equation 1), we do not include other factors, which, however, can be easily embedded in this framework (i.e., adding more balancing parameters and utility components in Equation 1) if necessary.

Discussion on social matching. Matching the riders with drivers and other riders with respect to their social factors is crucial to improve the user experience, when the monetary benefits become relatively insignificant (e.g., unlimited weekly/monthly ride packages). In general, there are three types of social matchings: satisfaction matching, similarity matching and structure matching, as shown in Figure 3.



(a) Satisfaction Matching (b) Similarity Matching (c) Structure Matching

Figure 3: Types of Social Matching.

Satisfaction Matching. As Figure 3(a) shows, the satisfaction matching happens between two riders r_a and r_b, where rider r_a desires some particular property A (e.g., speaking Chinese) and rider r_b can provide the property A. In other words, this type of matching is a supply-demand relationship. In this paper, the vehicle-related utility represents the suitableness of the satisfaction matching between riders and drivers.

Similarity Matching. As Figure 3(b) shows, similarity matching means connecting two riders, r_a and r_b, based on the similarity of their profiles (e.g., students), interests (e.g., NBA fans) or social circles (e.g., common friends). The more similar two people are, the better the match is. Our rider-related utility reflects the evaluation of similarities between riders. Thus, we should arrange a rider r_i to a vehicle c_j which would result in a higher rider-related utility.

Structure Matching. The social connections between four riders are shown in Figure 3(c). Then, the structure matching will match the riders based strictly on their social structures. For example, we may first construct coalitions [30] of riders (e.g., a coalition of riders r_a, r_b and r_c), and then assign the riders in the same coalition to the same vehicle. However, the size of the coalition may break the vehicle's capacity constraint. In addition, the riders in the same coalition may be physically located far from each other, then it is not practical to try to assign them to the same vehicle. Thus, in this paper, we do not consider the structure matching of riders.

Vehicle-related utility. For a given pair of rider r_i and vehicle c_j, the vehicle-related utility, $\mu_v(r_i, c_j) \in [0, 1]$, reflects the preference of rider r_i towards vehicle c_j. For example, an old rider may prefer to take a vehicle with a good suspension system to make the ride smooth. The vehicle-related utilities can be estimated with the categorically stated preferences of riders towards vehicles and drivers: i.e., riders can stipulate their preferences of vehicle brands and drivers (e.g., experienced or high-rated), then the ridesharing system can evaluate their preferences towards different vehicles.

Rider-related utility. Riders are more likely to enjoy a trip with other riders who share a lot of interests. Riders with similar tastes can easily communicate with each other and thus have a more interesting and enjoyable journey. The rider-related utility of rider r_i will change when other riders in vehicle c_j change. Specifically, we define rider-related utility in the following form,

$$\mu_r(r_i, c_j) = \sum_{tr_k \in TR_j^i} \frac{cost(tr_k)}{cost(TR_j^i)}\Big(\sum_{r_i' \in R_j^k - \{r_i\}} \frac{s(r_i, r_i')}{|R_j^k - \{r_i\}|}\Big), \quad (2)$$

where TR_j^i is a set of trajectories of the vehicle c_j with rider r_i in it, R_j^k indicates the riders in vehicle c_j during trajectory tr_k, and $s(r_i, r_i')$ is the social similarity of rider r_i and r_i'. Intuitively, for a rider r_i, sharing with other high-similarity riders for a larger portion of TR_j^i will lead to a higher rider-related utility, $\mu_2(r_i, c_j)$.

As nowadays social networks are very popular, we can easily access the friendship of riders when they register or connect ridesharing services with their social media accounts (e.g., Facebook). If any of the riders do not use social media accounts to register for ridesharing services, we can measure their similarities based on their ridesharing history or historical location records (e.g., common trips or popular locations). Here we use Jaccard similarity [23] to measure the similarities between riders. Specifically, for the two riders r_i and r_i', their social similarity is calculated as below:

$$s(r_i, r_i') = \frac{|\Gamma(r_i) \cap \Gamma(r_i')|}{|\Gamma(r_i) \cup \Gamma(r_i')|}, \quad (3)$$

where $\Gamma(r_i)$ denotes the set of friends of r_i and $|\Gamma(r_i)|$ is the number of elements of $\Gamma(r_i)$. Jaccard similarity is widely used in existing studies [10, 31, 11] in a range of fields, including social networks, data mining and statistics. Note that other methods can also be used to estimate the similarities between riders.

Trajectory-related Utility. Trajectory-related utility is to measure the satisfaction level of the rider r_i towards the route that vehicle c_j takes to deliver him/her to his/her destination. In general, the trajectory-related utility $\mu_t(r_i, c_j)$ decreases when the extra travel cost is incurred (e.g., caused by detours) to deliver rider r_i to his/her destination. We first define the *travel cost ratio* σ_{ij} of assigning rider r_i to vehicle c_j as below:

$$\sigma_{ij} = \frac{\sum_{tr_k \in TR_j^i} cost(tr_k)}{cost(s_i, e_i)} \quad (4)$$

where TR_j^i is a set of trajectories of the vehicle c_j with rider r_i in it. As $\sum_{tr_k \in TR_j^i} cost(tr_k) \geq cost(s_i, e_i)$, σ_{ij} is not less than 1.

In this work, we define the trajectory-related utility $\mu_t(r_i, c_j)$ as a decreasing function based on the logistic function [9] below:

$$\mu_t(r_i, c_j) = 2 - \frac{2}{1 + e^{-(\sigma_{ij}-1)}} = \frac{2}{1 + e^{(\sigma_{ij}-1)}}, \quad (5)$$

where e is the natural logarithm base (i.e., e = 2.71828...). Since σ_{ij} is not less than 1, the trajectory-related utility $\mu_t(r_i, c_j)$ will be in the range of (0, 1]. For example, when the travel cost ratio $\sigma_{ij} = 1$, the trajectory-related utility $\mu_t(r_i, c_j)$ is 1. The logistic function is widely used in a range of fields, including artificial neural networks, linguistics, and statistics. Here, we use the logistic function to model the scenario that the less the vehicle detours, the more the rider is satisfied. Note that, other decreasing functions can also be used to calculate $\mu_t(r_i, c_j)$.

2.5 Utility-Aware Ridesharing on Road Networks Problem

We formally define our URR problem below.

Definition 4. (Utility-Aware Ridesharing on Road Networks, URR) Given a set R of m riders and a set C of n vehicles on a road network G, the problem of *utility-aware ridesharing on Road Networks* (URR) is to arrange the riders to vehicles, such that:

Table 2: Symbols and Descriptions.

Symbol	**Description**
R	a set of m time-constrained riders
r_i	a rider r_i sending ride request q_i
s_i	the source location of ride request q_i
e_i	the destination location of ride request q_i
rt_i^-	the pickup deadline of ride request q_i
rt_i^+	the dropping off deadline of ride request q_i
C	a set of n dynamically moving vehicles
c_j	a vehicle c_j
a_j	the capacity of vehicle c_j
S_j	the schedule of pickup/dropoff events of vehicle c_j
TR_j	a list of consecutive trajectories of vehicle c_j
$cost(u, v)$	the travel cost between location u to location v
$\mu(r_i, c_j)$	the utility value of assigning rider r_i to vehicle c_j

1. at any time the number of riders in any vehicle c_j should not exceed its capacity a_j; and
2. the departure and arrival deadlines of each arranged rider r_i should be valid for the assigned vehicle c_j; and
3. the overall utility, $\sum_{r_i \in R'} \mu(r_i, c_{r_i})$, of the arranged riders R' is maximized, where c_{r_i} is the vehicle that the rider r_i is arranged to.

2.6 Hardness of Utility-Aware Ridesharing on Road Networks Problem

In this section, we first prove that our URR problem is NP-hard, through reducing a well-known NP-hard problem, *0-1 knapsack problem* [34], to the URR problem.

Theorem 2.1. *(Hardness of the URR Problem) The problem of the Utility-Aware Ridesharing on Road Networks (URR) is NP-hard.*

Proof. Please refer to Appendix B. □

Next, we discuss that URR is hard to approximate within any constant factor. Specifically, we show this result by reducing it from the DENSE k-SUBGRAPH problem [17], which is inspired by the work on social event organization [24]. By assuming the hardness of a conjecture, namely Unique Games with Small Set Expansion Conjecture [22], the authors [28] show that DENSE k-SUBGRAPH is hard to approximate within any constant factor. Due to space limitations, for the details and history of the conjecture, please refer to [28].

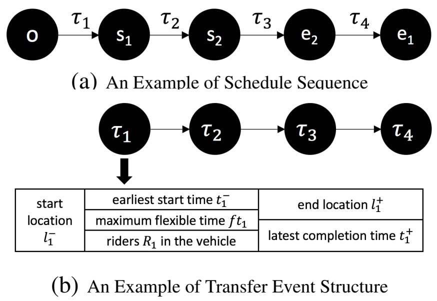
Theorem 2.2. *Assuming the Unique Games with Small Set Expansion Conjecture, it is NP-hard to approximate the URR problem within any constant factor in polynomial time.*

Proof. Please refer to Appendix C. □

The URR problem needs to assign riders to vehicles to achieve the maximum overall utility of riders while satisfying the constraints of riders and vehicles (e.g., capacity of vehicle). Unfortunately, URR is NP-hard and unlikely to be approximated within any constant factor in polynomial time as shown in Theorems 2.1 and 2.2. Thus, due to the NP-hardness of URR, in subsequent sections, we first propose an efficient algorithm to insert one rider for a given vehicle such that the incremental travel cost is minimized. Then we propose three heuristic algorithms, namely bilateral arrangement, efficient greedy, and grouping-based scheduling approaches to efficiently achieve assignments and schedules for our URR problem.

3. ARRANGE A SINGLE RIDER

Usually, a ridesharing system needs to respond to each rider quickly. One straight problem is to arrange a single rider r_i to a given vehicle c_j. From the perspective of drivers, they want to serve all the riders with the least travel cost. Unfortunately, this problem is known as the *single-server dial-a-ride problem* [15], which is NP-hard by reducing it from the Travelling Salesman Problem (TSP) [34]. According to the experimental results of the existing work on real-time ridesharing problems, it is not in practice necessary to reorder the points of a schedule before the insertion of a new rider with the goal of minimizing the travel distance [25]. Therefore, we assume the existing schedule for each vehicle will not be reordered. In this section, we propose one efficient approach to arrange a new rider to a given vehicle with the minimum incremental travel cost (i.e., an incremental update operation).



o
τ_1
s_1
τ_2
s_2
τ_3
e_2
τ_4
e_1
(a) An Example of Schedule Sequence
τ_1
τ_2
τ_3
τ_4
start location l_1^-
earliest start time t_1^-
maximum flexible time ft_1
riders R_1 in the vehicle
end location l_1^+
latest completion time t_1^+
(b) An Example of Transfer Event Structure

Figure 4: Illustration of Transfer Event Structure.

3.1 Transfer Event Structure

Before we represent the approach to arrange a single rider r_i for a given vehicle c_j, we introduce a data structure to store the status of the vehicle c_j.

In Figure 4(a), we present a valid schedule for vehicle c_j, where each node indicates a location and each edge indicates a transfer event. For example, from location o to location s_1, the edge τ_1 presents the transfer event from location o to s_1. Then we can generate a corresponding transfer event list as shown in Figure 4(b). We connect the transfer event nodes following the sequence in Figure 4(a). Specifically, for each transfer event node τ_u, we store the following information: a) start location l_u^- and end location l_u^+; b) the earliest start time t_u^- and the latest completion time t_u^+; c) riders R_u in the vehicle c_j on event τ_u; d) the maximum flexible time ft_u on the trajectory between l_u^- and l_u^+. We show the structure of transfer event τ_1 at the bottom of Figure 4(b).

The earliest start time t_u^-. The earliest start time t_u^- of the transfer event τ_u indicates the time when the vehicle can reach location l_u^- if no time has been wasted or delayed in any prior transfer event $\tau_v (v < u)$. Specifically, when vehicle c_j is currently located at location o at timestamp $\bar{t}$ and the current ongoing transfer event is τ_v, for a travelling event τ_u $(v < u)$, the earliest start time t_u^- is calculated as follows:

$$t_u^- = \bar{t} + cost(o, l_v^+) + \sum_{i=v+1}^{u} cost(l_i^-, l_i^+), \tag{6}$$

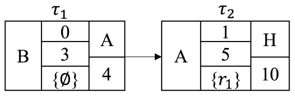
where l_i^- and l_i^+ are the start and the end location of the transfer event τ_i, respectively, and $cost(l_i^-, l_i^+)$ is the travel cost from location l_i^- to location l_i^+. Note that after inserting or deleting one transfer event τ_v from the transfer sequence, we only need to update the earliest start time field of the subsequence nodes τ_u $(v < u)$.

The latest completion time t_u^+. The latest completion time t_u^+ of the transfer event τ_u represents the latest time for the vehicle to arrive at the end location l_u^+ of τ_u such that it is still possible to complete the next event if no time will be wasted or delayed in the next transfer event. Specifically, for a vehicle c_j with a list of n transfer events, the latest completion time of the transfer event τ_u is calculated as follows:

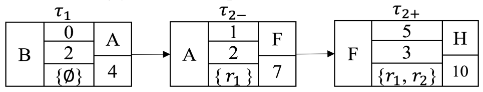
$$t_u^+ = \begin{cases} \min(t_{u+1}^+ - cost(l_{u+1}^-, l_{u+1}^+), dl(l_n^+)), & u \neq n \\ dl(l_n^+), & u = n \end{cases} \tag{7}$$

where l_{u+1}^- and l_{u+1}^+ are the start location and the end location of the transfer event τ_{u+1}, respectively. $dl(l_n^+)$ is the deadline to reach location l_n^+ (i.e., the required pickup or the dropoff deadline of the corresponding rider).

The flexible time ft_u. The flexible time ft_u for a transfer event τ_u is the time that vehicle c_j can stop or detour while moving from



(a) Transfer Sequence of Vehicle c_1



(b) Transfer Sequence of Vehicle c_1 after Inserting Location F

Figure 5: An Example of Inserting a Pickup Location.

location l_u^- to l_u^+, which allows the vehicle c_j to spend more time from location l_u^- to l_u^+ to pick up or deliver one additional rider, such that it is still possible to complete all subsequent events. We notice that for each transfer event τ_u in a transfer sequence of n transfer events, its flexible time can be calculated as follows:

$$ft_u = \begin{cases} \min(t_u^+ - t_u^- - cost(l_u^-, l_u^+), ft_{u+1}), & u \neq n \\ t_u^+ - t_u^- - cost(l_u^-, l_u^+), & u = n \end{cases} \quad (8)$$

where l_u^- and l_u^+ are the start location and the end location of the transfer event τ_u, respectively, and t_u^- and t_u^+ are the earliest start time and the latest completion time of event τ_u, respectively.

Example 2. *(Pickup Location Insertion) For the vehicles and riders in the road network shown in Figure 1, assume we currently arrange vehicle c_1 to pick up rider r_1 at A and deliver him/her to destination H. We then have a transfer sequence T_1 as shown in Figure 5(a), where each node indicates a transfer event. For example, node τ_1 represents that the vehicle c_1 is currently located at B and needs to reach location A before timestamp 4. As the travel cost from B to A is 1, the flexible time of τ_1 is 3 (=4-0-1). To insert a pickup location F in T_1, it is only valid to insert F to τ_2. As for τ_1, its flexible time of 3 is not enough for vehicle c_1 to pick up r_2 at F first, then arrive at location A before the deadline time* 4.

After we insert the pickup location F into τ_2, we get two transfer events τ_2^- (from A to F) and τ_2^+ (from F to H). To maintain the fields (e.g., flexible time ft) of the transfer events, we update the earliest start times from τ_2^+ to the tail of T_1, then from the tail of T_1 backwards to τ_1 update the latest completion times and the flexible times of transfer events with Equations 7 and 8, respectively.

3.2 Single Rider Insertion

In this subsection, we propose an algorithm, namely ArrangeSingleRider, to insert a new rider r_i into an existing transfer sequence T_j of vehicle c_j such that the incremental travel cost is minimized.

Insert one location. To arrange a new rider r_i into an existing transfer sequence T_j of vehicle c_j, we need to first filter out the valid transfer events for the source location s_i and destination location e_i to be inserted. We present Lemma 3.1 here to guide the selection of valid insertion positions.

Lemma 3.1. *(Valid Insertion) For a transfer event τ_u, the insertion of a location x_i ($x_i = s_i$ or e_i) of a rider r_i is valid if and only if the following conditions are all satisfied:*
a) $t_u^- \leq dl(x_i)$;
b) $cost(l_u^-, x_i) \leq dl(x_i) - \bar{t}$;
c) $cost(l_u^-, x_i) + cost(x_i, l_u^+) - cost(l_u^-, l_u^+) \leq ft_u$;
d) $|R_u| + 1 \leq a_j$ (for $x_i = s_i$),
where $dl(x_i)$ is the deadline for reaching location x_i (i.e., the pickup or dropoff deadline of the corresponding rider), $\bar{t}$ is the current timestamp, $|R_u|$ is the number of riders in the vehicle during transfer event τ_u, and a_j is the capacity of vehicle c_j.

Condition a) tells that the deadline for reaching location x_i should be later than the earliest start time of event τ_k. Condition b) is required as the remaining time to reach location x_i is enough for vehicle c_j to move from start location l_u^- to x_i. Condition c) dictates the flexible time ft_u is larger than the incremental travel cost

Algorithm 1: ArrangeSingleRider

```
    Input: An existing transfer sequence $T_j$ of vehicle $c_j$ and a rider $r_i$
    Output: A best transfer sequence $\hat{T}_j$
 1  $\hat{\Delta} \leftarrow$ INFINITY
 2  $\hat{T}_j \leftarrow \emptyset$
 3  $T_s \leftarrow$ ValidEvents($T_j$, $s_i$)
 4  $T_e \leftarrow$ ValidEvents($T_j$, $e_i$)
 5  sort $T_s$ and $T_e$ based on incremental travel costs $\Delta_{ij}$ in
      ascending order
 6  for transfer event $\tau_u$ in $T_s$ do
 7      if $\Delta_{iu} \geq \hat{\Delta}$ then
 8          break
 9      $T_j' \leftarrow$ insert $s_i$ in transfer event $\tau_u$ of $T_j$
10      updateEventFields($T_j'$, $\tau_u$, $s_i$)
11      for transfer event $\tau_v$ in $T_e$ do
12          if $v < u$ then
13              continue
14          if $\Delta_{iu} + \Delta_{iv} \geq \hat{\Delta}$ then
15              break
16          if $\tau_v$ is still valid to insert $e_i$ then
17              $\hat{T}_j \leftarrow$ insert $e_i$ in transfer event $\tau_v$ of $T_j'$
18              $\hat{\Delta} \leftarrow \Delta_{iu} + \Delta_{iv}$
19  return $\hat{T}_j$
```

$\Delta_{iu} (= cost(l_u^-, x_i) + cost(x_i, l_u^+) - cost(l_u^-, l_u^+))$ after inserting location x_i. Finally, condition d) means the count of riders in the vehicle c_j should not exceed the capacity a_j of c_j. Only when all the conditions in Lemma 3.1 are satisfied, the transfer event τ_u is valid to insert the location x_i.

Although the conditions in Lemma 3.1 are easy to check, we still need to traverse all the transfer events of the existing transfer sequence T_j of vehicle c_j to choose the valid events for insertion, which is not efficient. We notice that for any event τ_v, the earliest start time t_u^- of any subsequent transfer event τ_u ($v < u$) is larger than t_u^-. Thus, we have the following Lemma 3.2 to stop searching for valid events for insertion earlier.

Lemma 3.2. *(Earliest Start Time Pruning) For an insert location x_i and an existing transfer sequence T_j of vehicle c_j, if the earliest start time t_u^- of transfer event τ_u is later than the deadline of reaching location x_i, then the subsequent transfer events τ_v ($v > u$) are invalid to insert x_i.*

Lemma 3.2 asserts that when searching for valid transfer events to insert in location x_i from the beginning of a given sequence, it is safe to stop when we find the earliest start time t_u^- of the current checking event τ_u is later than the deadline $dl(x_i)$ of reaching location x_i.

The pseudo code of the algorithm ArrangeSingleRider is illustrated in Algorithm 1. We first initialize the current minimum increment travel cost $\hat{\Delta}$ as infinity such that any possible insertion can lead to a smaller travel cost increment (line 1). As no best transfer sequence has been found, we set the best transfer sequence $\hat{T}_j$ as empty (line 2). Then, we pick out the valid insertion events T_s and T_e for source location s_i and destination location e_i of the new rider r_i respectively using Lemmas 3.1 and 3.2 (lines 3-4), and sort the events in T_s and T_e based on their incremental travel cost Δ_{ij} in ascending order separately (line 5). Then we check every possible insertion events pair (one event from T_s and the other one from T_e) and report the pair with the minimum incremental travel cost,

$\Delta_{iu} + \Delta_{iv}$ (lines 6-18). Specifically, we first pick out one transfer event τ_u with the least incremental travel cost Δ_{iu} (line 6). If Δ_{iu} is already larger than the current minimum incremental travel cost $\hat{\Delta}$, we can safely stop checking other event pairs (lines 7-8). We insert location s_i into the transfer event τ_u of T_j and assign the resulting sequence to T'_j (line 9). Then we update the fields of events in T'_j (line 10). Here we update the earliest start times (with Equation 6) and riders from event τ_u forwards to the end of T'_j, and then update the flexible times (with Equation 8) from the tail of T'_j backwards to the head of it. Similarly, we traverse the valid events τ_v to insert location e_i in ascending order of the incremental travel costs (line 11). If the event τ_v is prior to the event τ_u, which means vehicle c_j needs to deliver r_i before picking up him/her, then, we skip this event τ_v (lines 12-13). What is more, if the incremental travel cost $\Delta_{iu} + \Delta_{iv}$ is larger than the currently found minimum incremental travel cost $\hat{\Delta}$, we can stop this traverse on T_e (lines 14-15). Moreover, if we find a new valid pair with a smaller incremental travel cost than $\hat{\Delta}$, then we update the best sequence $\hat{T}_j$ and $\hat{\Delta}$ (lines 16-18). Finally, we return the found bets sequence $\hat{T}_j$.

The Time Complexity. For a transfer sequence T_j with n transfer events, the time complexity of Algorithm 1 is $O(n^2)$. It needs $O(1)$ to check whether a location can be inserted into a transfer event with Lemma 3.1, and $O(n)$ to select valid events (lines 3-4). Sorting the events in T_s and T_e needs $O(n \log n)$ (lines 5). The iterations on T_s at most runs n times (lines 6-18). Insert s_i into event τ_u needs $O(1)$ (line 9). To update the fields of events in T'_j needs $O(n)$ (line 10). Also, iterations from line 11 to line 18 run at most n times, and it each time needs $O(1)$. Thus, the whole time complexity of the algorithm is $O(n^2)$. Note that the algorithm runs fast in average situations, as many events have been pruned in the selection of the valid events with Lemma 3.1 and 3.2.

Discussion on the Optimality. In this paper, we assume that the existing transfer sequences will not be reordered, as a result, our approach ArrangeSingleRider can achieve the exact solution for arranging a single rider to a given vehicle with the minimum increase on its travel cost. When the reorder operation is allowed, one existing method [20] can report the optimal transfer sequence with the minimum incremental travel cost. It utilizes a kinetic tree structure to store all the valid schedule sequences of each vehicle, then inserts the single rider to the best stored schedule sequence. However, to arrange multiple riders to multiple vehicles with the minimum travel cost (known as the Dial-a-Ride Problem), our approach and the kinetic tree approach [20] all just achieve local optimal results as the deletion and the switch operations on the arranged riders are not allowed (i.e., no arranged riders will be deleted or switch to other vehicles). Existing solutions [13] have the high computational complexities and only can solve small-size instances (e.g., less than 10 vehicles and 50 riders), thus cannot handle the real-world instances (e.g., thousands of vehicles and riders).

4. THE BILATERAL ARRANGEMENT APPROACH

As discussed in Section 2.6, the URR problem is NP-hard and unlikely to be approximated within any constant factor in polynomial time. An alternative choice is to design efficient heuristic algorithms to achieve high quality results within affordable time. Thus, we propose the bilateral arrangement algorithm in this section.

From the perspective of the riders, they want to have a nice ridesharing experience, when they enjoy a ridesharing service. On the other side, for each vehicle driver with a set of riders to pick up at sources and deliver to destinations, the driver wants to have a smart schedule sequence, such that the total travel cost is minimized. In fact their objectives (to maximize the utilities and to minimize travel distances) are consistent. To serve a rider with less travel cost increase, a vehicle may have more flexible time. Then, it could be flexible enough to provide service to more other high-utility riders. Inspired by this observation, we propose a bilateral arrangement algorithm, which assigns each rider to a suitable vehicle. Each time we assign a rider r_i to a vehicle c_j, the assignment is suitable iff no other vehicle c_k exists such that $\mu(S'_j) - \mu(S_j) \leq \mu(S'_k) - \mu(S_k)$ and $cost(S'_j) - cost(S_j) \geq cost(S'_k) - cost(S_k)$, where S'_j is the schedule of vehicle c_j after arranging rider r_i to it, and $\mu(S_j)$ and $cost(S_j)$ are the total utility and total travel cost of vehicle c_j arranged with the schedule S_j, respectively.

Algorithm 2: BilateralArrangement

```
   Input: A set $C$ of $n$ avaliable vehicles and their $n$ incumbent
          schedules $S$, and a set $R$ of $m$ rider
   Output: A set of updated scheduling sequences $S$
1  foreach $r_i \in R$ do
2      retrieve a list $C_i$ of vehicles that are valid to $r_i$
3  while $R \neq \emptyset$ do
4      randomly pick one rider $r_i$
5      remove $r_i$ from $R$
6      while $C_i \neq \emptyset$ do
7          pick one vehicle $c_j$ with the highest utility increase for $r_i$
8          remove $c_j$ from $C_i$
9          if rider $r_i$ can be arranged in $c_j$ then
10             arrange $r_i$ to $c_j$
11             break
12         else if $r_i$ can replace rider $r'_i$ of $c_j$ then
13             replace $r'_i$ with $r_i$
14             put $r'_i$ back to $R$
15             break
16 return $\mathbb{S}$
```

Specifically, for each rider r_i, we maintain a list C_i of valid vehicles. Then, for each rider, we first try to assign it to the vehicle c_j with the highest utility. If we can directly insert r_i to the schedule of c_j, we simply insert r_i into the schedule of c_j. Otherwise, if there is no flexible time for c_j to serve r_i, we try to replace one rider r'_i already assigned to c_j such that the total travel cost of c_j can be reduced and the overall utility can be improved.

The pseudo code of the algorithm BilateralArrangement is shown in Algorithm 2. We first, for each rider r_i, we retrieve a list C_i of vehicles that are valid to serve rider r_i. Here a valid vehicle c_j means that the rider r_i can find a valid position of the schedule S_i according the Lemma 3.1 (line 2). The loop from line 4 to line 14 process at least one rider in each iteration. In each iteration, we randomly select one rider r_i that has not been arranged to any vehicle (lines 5-6). Then, we greedily pick the vehicle c_j that has the highest utility $\mu(r_i, c_j)$ (line 8). To avoid a rider r_i being replaced and reinserted back to a particular vehicle c_j, when we test whether r_i can be arranged to c_j, we remove c_j from the preference vehicle list C_i of rider r_i (line 9). Next, we arrange the riders from the perspective of vehicles (lines 10-16). If rider r_i can be arranged to c_j, we simply insert r_i into the schedule sequence of c_j (lines 10-12). Otherwise, we try to replace another rider r'_i with r_i to reduce the total travel cost of c_j and improve the overall utility (lines 13-16).

5. THE EFFICIENT GREEDY APPROACH

Although the bilateral arrangement algorithm in Section 4 can achieve high-utility results efficiently, it needs to adjust the settled arrangements when better choices appear, which leads to its running time is hard to analyze.

A popular alternative is the greedy-based approximate method. In this section, we introduce an efficient greedy algorithm, which greedily selects a rider-and-vehicle pair with maximum utility efficiency. Here, the utility efficiency f_{ij} of assigning rider r_i to vehicle c_j, whose incumbent scheduling is S_j, is defined as follows:

$$f_{ij} = \frac{\mu(S'_j) - \mu(S_j)}{cost(S'_j) - cost(S_j)} \tag{9}$$

where S'_j is the schedule after arranging rider r_i to vehicle c_j, and $\mu(S_j)$ and $cost(S_j)$ are the total utility and the total travel cost of vehicle c_j arranged with the schedule S_j, respectively. The intuition is that some rider-and-vehicle pair may have a high incremental utility and a large travel cost increase, which may exhaust the potential of the vehicle to serving other riders. Then, a rider-and-vehicle pair with higher utility efficiency can have a higher incremental utility with a smaller travel cost increase.

Specifically, to arrange one more rider r_i to a particular vehicle c_j with an incumbent scheduling S_j, we utilize Algorithm 1 to achieve a non-reordered local optimal sequence S'_j.

The pseudo code of the algorithm EfficientGreedy is illustrated in Algorithm 3. We first initialize the set $\mathbb{I}$ of valid rider-and-vehicle pairs as empty (line 1), then for each rider r_i, we retrieve the valid vehicles c_j and put a pair $\langle r_i, c_j\rangle$ to $\mathbb{I}$ (lines 2-4). Here, the valid vehicles are filtered out with the condition a) and b) of Lemma 3.1, which can be sped up with a spatial index [29]. Then we calculate the utility efficiency f_{ij} of each pair $\langle r_i, c_j\rangle$ (lines 5-6). Next, we greedily select the pair $\langle r_i, c_j\rangle$ with the current highest efficiency and arrange the rider r_i to vehicle c_j with Algorithm 1 (lines 9-10). As the insertion may change some efficiencies of pairs $\langle ., c_j\rangle$ related to vehicle c_j, we update their efficiencies (line 11). Then, we remove some invalid pairs (line 12). Finally, the updated scheduling sequences for vehicles are returned (line 13).

Algorithm 3: EfficientGreedy

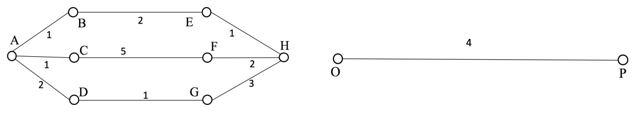
```
   Input: A set $R$ of $m$ riders, a set, $C$, of $n$ avaliable vehicles
          and their $n$ incumbent schedules $S$
   Output: A set of updated schedules $S$
1  $\mathbb{I} \leftarrow \emptyset$
2  for rider $r_i \in R$ do
3      retrieve the possible vehicles $C_i$ for $r_i$
4      $\forall c_j \in C_i$, push one pair $\langle r_i, c_j\rangle$ to $\mathbb{I}$
5  for rider-and-vehicle pair $\langle r_i, c_j\rangle \in \mathbb{I}$ do
6      calculate the utility increasing efficiency $f_{ij}$ with Equation 9
7  sort the pairs of each rider based on their utility efficiency in
     descending order, separately
8  while $\mathbb{I} \neq \emptyset$ do
9      select one pair $\langle r_i, c_j\rangle$ with the highest efficiency
10     insert rider $r_i$ to vehicle $c_j$ and arrange the scheduling $S_j$
         with Algorithm 1
11     update the efficiencies of pairs $\langle ., c_j\rangle$ in $\mathbb{I}$
12     remove pairs $\langle r_i, .\rangle$ and invalid pairs from $\mathbb{I}$
13 return $S$
```

The Time Complexity. For a set of n vehicles C and their n incumbent scheduling sequences to support a set of m rider R, the time complexity of Algorithm 3 is $O(\max(mx^2, mn\log(n)))$, where x is the average length of the vehicle schedule. As there are at most mn possible rider-and-vehicle pairs, to retrieve the possible rider-and-vehicle pairs $\mathbb{I}$ needs $O(mn)$ (lines 2-4). Also, to calculated the utility efficiency of all the pairs in $\mathbb{I}$ requires $O(mn)$ (lines 5-6). It requires $O(n\log(n))$ to sort each group of (at most n) rider-and-vehicle pairs of a rider. Then, it requires $O(mn\log(n))$ to sort all the m groups of pairs separately (line 7). As the pairs of each rider have been sorted, to select one pair with the highest efficiency needs $O(n)$ (line 9). Assume that the length of each schedule sequence is k, to insert a new rider with Algorithm 1 needs $O(x^2)$. In each iteration from lines 8 to line 12, we need to update the efficiencies of at most m pairs. To maintain the order of one updated pair in its group, we need $O(\log(n))$. Then, the updating operation needs $m\log(n)$ (line 11). In addition, in each iteration we need to remove at most n invalid pairs from $\mathbb{I}$, which needs $O(n)$. As in each iteration, we will assign at least one rider to some vehicle, there will be at most m iterations, then lines 8 - 12 need $O(\max(mx^2, mn\log(n)))$. Thus, the whole time complexity of Algorithm 3 is $O(max(mx^2, mn\log(n)))$.

6. THE GROUPING-BASED SCHEDULING APPROACH

Although the bilateral assignment algorithm and efficient greedy algorithm can assign riders to suitable vehicles to maximize the total utility value, their time complexities are large as they process riders one by one. Then one straightforward idea to optimize the running speeds is to group the riders and solve each group of riders separately. According to a report [33] of the statistics of New York City taxis in 2014, more than 50% of trips are less than 3 miles. We may find some key vertices and attach the other vertices to them. For a key vertex u_k and its attached vertices, they form an area A_k. For example, as Figure 6 shows, the original road network is illustrated in Figure 6(a). Then, we pick out two key vertices A and H. For the key vertex A, we attach a set $\{B, C, D\}$ to it. They form an area O, whose center is A. Similarly, we get an area P consisting of $\{E, F, G, H\}$, whose center is H. Moreover, the distance between two areas is the length of the shortest part between A and H.

In this section, we first propose a k-path covers based areas construction algorithm. Next, based on the travel distances, we classify the ride requests into two classes: short-trips and long-trips. Then, we process different kinds of trips.



(a) A Road Network Example (b) The Corresponding Road Skeleton

Figure 6: Example of 3-Path Cover.

6.1 Area Construction Algorithm

Preprocessing. As the lengths of the edges in road networks are not necessarily even, there may be some edges of tens of miles. To construct areas with similar radii, we modify the original road networks with a simple preprocessing, which breaks the long edges evenly into shorter edges through inserting pseudo nodes. Specifically, for a given upper bound of edge length d_{max}, we check whether or not to insert pseudo nodes into each edge (u, v). As we do not differentiate between the length and travel cost in this work, d_{max} can also be used as the maximum travel cost. Based on the length of (u, v), we can calculate the number of pseudo nodes to insert with the equation below:

$$n_e = \lfloor \frac{cost(u, v)}{d_{max}} \rfloor. \tag{10}$$

Then, we uniformly insert n_e pseudo nodes into the edge (u, v) such that the travel cost between each two successive node of $n_e + 2$ nodes (including u and v) is $\frac{cost(u,v)}{n_e}$.

Algorithm 4: AreaConstruction

```
   Input: A graph $G = \langle V, E \rangle$ and a positive integer $k$
   Output: A set $A$ of constructed areas
1  $A \leftarrow \emptyset$
2  Retrieve a set $V'$ of $k$-shortest-path cover vertices of $G$
3  foreach $u_j \in V'$ do
4      create a area $a_j \leftarrow \{u_j\}$
5      insert $a_j$ to $A$
6  foreach $v_i \in V - V'$ do
7      find the closest vertex $u_j \in V'$ of vertex $v_i$
8      add $v_i$ to the corresponding area $a_j$
9  return $A$
```

Area Construction. To construct the areas, we first need to select the key vertices, which can represent the skeleton of the origin road network. Here we use the state-of-art algorithms [18] of the k-path covers problems to retrieve a subset $V' \subset V$ of key vertices such that any path consisting of k vertices has at least one vertex $u \in V'$. As we only consider the shortest paths, here we use the algorithm to solve the minimum k-shortest-path cover (k-SPC) problem to select the key vertices. Specifically, for a given graph $G = \langle V, E \rangle$ and a positive integer k, k-SPC is to select a minimum subset of vertices $V' \subset V$ such that, for every shortest path $\pi = v_1, ..., v_k$ in G, $\pi \cap V' \neq \emptyset$. For example, in Figure 6, the set $\{A, H\}$ is a solution of a 2-SPC problem shown in Figure 6(a).

The pseudo code of algorithm AreaConstruction to construct the areas of a given road network is shown in Algorithm 4, which returns a set of constructed areas A of a given graph G and a positive integer k. At the beginning, we initialize A to an empty set as no areas exists (line 1). Then, we retrieve a subset V' through an existing algorithm to solve the k-SPC problem [18] (line 2). Next, we create an area a_j for each key vertex $v_j \in V'$ and insert a_j to A (lines 3-5). For the other vertices, we attach each one to a corresponding area of a closest key vertex (lines 6-8).

Short-/Long-Trips Grouping. According to the travel distances, the trips can be classified into two categories: short-trips and long-trips. One trip is a short-trip, if its travel distance is smaller than the upper bound of the radii of the constructed areas (i.e., $d_{max} \cdot k$); otherwise, it is a long-trip. We group the short-trips starting in the same area as a group and the long-trips as a group. Next, we first arrange the long-trips as they may have huge impacts on the schedules of vehicles. Then, for the groups of short-trips, we process them following the order from groups with more trips to groups with less trips. We next propose the grouping-based scheduling algorithm.

6.2 Arrange Groups of Trips

Based on the area construction and classification of trips, we can now arrange trips with different priorities. The pseudo code of the grouping-based scheduling algorithm is shown in Algorithm 5. We first initialize $\eta + 1$ groups of trips as empty sets (line 1). Then, for the short-trips, we group the trips starting in the same area into the same group, while the long-trips are classified into group g_0 (lines 2-6). Next, we sort the trip groups based on the number of trips in descending order (line 7). We solve the trip groups with our bilateral arrangement algorithm or efficiency greedy algorithm. As the long-trips have a larger impact on the schedules of vehicles, we first arrange the trips in group g_0(line 8). Then, for the short-trips, we process them group-by-group based on the number of trips in each short-trip group. We give higher priorities to groups with more trips. Finally, we return the updated scheduling sequences.

Algorithm 5: GroupArranging

```
    Input: A graph $G = \langle V, E \rangle$ and the set $A$ of its $\eta$ constructed
           areas, a set of $n$ avaliable vehicles $C$ and their $n$
           incumbent schedules $S$, and a set $R$ of $m$ riders
    Output: A set of updated schedules $S$
1   $\forall g_x \leftarrow \emptyset, x = 0, 1, ..., \eta$
2   foreach $r_i \in Q$ do
3       if $r_i$ is a short-trip then
4           put $r_i$ to the group $g_x$ where $s_i$ is in the area $a_x$
5       else
6           put $r_i$ to group $g_0$
7   sort trip groups $g_x$ based on their numbers of trips in
      descending order
8   solve the trips in group $g_0$
9   while $\exists g_x$ is unsolved do
10      select the group $g_x$ with maximum number of trips among
          unsolved groups
11      solve the trips in group $g_x$
12  return $S$
```

Fast Valid Vehicles Filtering. To arrange the riders in one group g_x, we need to filter out the valid vehicles for the riders. A direct way is to filter out a set of the valid vehicles of each rider, then merge all the valid vehicle sets, which may cost at least the same computational cost compared to non-grouping methods. However, with the help of the constructed areas, we can quickly filter out the valid vehicles. Specifically, for a given group g_x and a vehicle c_j, we find out the latest pick up deadline rt^-_{max} among all the riders classified into g_x, then use the location of the key vertex u_x of the area of the group g_x to apply the following condition to filter out valid vehicles:

$$cost(u_x, l(c_j)) - d_{max} \cdot k < rt^-_{max} - \bar{t}$$

where d_{max} is the upper bound of the edge length and $\bar{t}$ it the current timestamp. The condition implies the vehicle that can arrive at the original location of any rider in group g_x is valid for the group. With this condition, we can quickly filter out the valid vehicles and thus improve the solving speed of group g_x.

The Time Complexity. As the AreaConstruction procedure is in fact a preprocessing for the road network, it does not affect the arranging process. Here, we mainly analyze the time complexity of Algorithm 5, namely GroupArranging. The time complexity to classify one rider is $O(\log \eta)$ with the help of the index data structures (e.g., B+ Tree). Thus, to classify the riders, the time complexity is $O(m \log \eta)$ (lines 2-6). To sort the trip groups, the time complexity is $O(\eta \log \eta)$ (line 7). Assume the trips and vehicles are evenly distributed, then, for each trip group, there are $\frac{m}{\eta}$ trips and $\frac{n}{\eta}$ valid vehicles. In addition, we assume Algorithm 3, namely Efficient$m$Greedy, is used to solve each trip group and its time complexity is $O(mn \log n)$ to arrange m riders to n vehicles. Then, the time complexity to solve η trip groups is $O(\eta(\frac{m}{\eta} \cdot \frac{n}{\eta}) \log(\frac{n}{\eta}) = O(\frac{mn}{\eta} \log \frac{n}{\eta})$. Then, the total time complexity of Algorithm 5 is $O(\max(\eta \log \eta, \frac{mn}{\eta} \log \frac{n}{\eta}))$.

6.3 Cost-Model-Based Estimation of the Best Number of Groups

With the grouping-based scheduling (GBS) algorithm, we can improve the processing speed of the URR algorithms. However, the parameter k may affect the running speed of the GBS algorithm in two parts of the calculation: 1) constructing the areas and classifying the trips; 2) resolving each groups of trips. When k is large, the prior part needs low time cost but the later part needs high time cost. In this subsection, we analyze the running cost of the GBS algorithm and propose a cost model, then derive a best k value such that the running time of GBS algorithm is minimized.

Specifically, the cost of the GBS algorithm includes 2 parts: the cost, F_a, of the area construction algorithm (in Algorithm 4) and the cost, F_g, of the group scheduling algorithm (in Algorithm 5).

The cost, F_a, of areas construction. From algorithm AreaConstruction (in Algorithm 4), we first need to retrieve a set of k-shortest-path cover vertices of the original graph G with s vertices. As discussed in [18], their QuickPruning algorithm first adds all the nodes to the coverage set G', then tries to remove each node if they are not necessary in set G' to maintain the k-SPC properties that are discussed in Section 6.1. For each node v, to determine whether to remove it or not requires a time cost of $O(N_k(v)logN_k(v) + N_k(v)^2)$, where $N_k(v)$ denotes the number of nodes in G that lie on a k-path originating in v. As this time complexity varies for each node, it is hard to analyze the overall time complexity of the QuickPruning algorithm [18]. According to the experimental study in [18], the QuickPruning algorithm runs faster than the *adaptive-sampling* algorithm in [32], whose total cost is $O(s\bar{\sigma}_{k-1}\log\bar{\sigma}_{k-1})$, where $\bar{\sigma}_{k-1}$ is the average number of $(k-1)$-hop neighbors of the vertices in G. As analyzed in [32], when $\bar{\sigma}_{k-1}$ is far smaller than s, $O(s\bar{\sigma}_{k-1}\log\bar{\sigma}_{k-1})$ grows linearly with n. Thus, we simply note the cost of retrieving a set of k-shortest-path cover vertices as $O(C_k s)$, where C_k is a constant estimated for the given road network G and s is the number of vertices of G. Unfortunately, the QuickPruning algorithm does not have a bound for the size of the coverage set G' [18], thus we note the size of set G' as η for a given parameter k for our further analysis. Note that, the size of set G' is equal to the number of constructed areas. To attach the other vertices in $G - G'$ to their corresponding areas, with the help of spatial index structures (e.g., B+ Tree), it costs $O(s\log\eta)$. From the discussion above, we can obtain the cost F_a below:

$$F_a = s(C_k + \log\eta),$$

where C_k is a constant estimated for the given road network G while calculating the k-shortest path cover vertices, s is the number of vertices of G, and η is the size of the coverage set G'.

The cost, F_g, of group scheduling. In Algorithm 5, to classify the riders, lines 2-6 need $O(2m\log\eta)$ cost. To sort the η groups based on their trip counts, it needs $O(\eta\log\eta)$ cost. Then, to use our EfficientGreedy algorithm to process η groups of trips in lines 9-11, it needs $O(\frac{mn}{\eta}\log\frac{n}{\eta}))$ cost (discussed in Subsection). As the k is usually small (e.g., 16) in practice for a city scale road network, and the larger k is, the smaller η is, we have $\eta >> 1$, thus we ignore the cost to process the group g_o in line 8. Then, we have the cost of group scheduling as below:

$$F_g = 2m\log\eta + \eta\log\eta + \frac{mn}{\eta}\log\frac{n}{\eta}$$

The total cost of the GBS algorithm. The total cost, $Cost_{gbs}$, of the GBS algorithm can be given by summing up the two parts of cost, F_a and F_g. Thus, we have

$$Cost_{gbs} = s(C_k + \log\eta) + 2m\log\eta + \eta\log\eta + \frac{mn}{\eta}\log\frac{n}{\eta}$$

We take the derivation of $Cost_{gbs}$ over l, and let it be 0. In particular, we have:

$$\frac{\partial Cost_{gbs}}{\partial\eta} = \frac{s+2m}{\eta} + \log\eta + 1 - \frac{mn}{\eta^2}(\log\frac{n}{\eta} + 1) = 0$$

We notice that when $\eta = 1$, $\frac{\partial Cost_{gbs}}{\partial\eta}$ is much smaller than 0 but increases when η grows. In addition, η can only be an integer and determined by k. When k is large, η is small. Thus, we can do a binary search for the most suitable k value in range $[1, s]$ such that $\frac{\partial Cost_{gbs}}{\partial\eta}$ is larger than 0, which induces a minimum running cost on the GBS algorithm.

7. EXPERIMENTAL STUDY

7.1 Experimental Methodology

7.1.1 Data Set

We use both real and synthetic data to test our proposed URR approaches. Specifically, for real data, we use the taxi trip data sets in NYC [5] and Chicago [6] and the USA road networks data set [1] and Gowalla Check-in data set [2].

Road Networks. In the USA road networks data set, it includes the latitude and longitude of each node. For each edge, its travel time from one node to another node is given. Then, we have a graph of the road networks in the NYC area (with longitude from $-74.5°$ to $-73.5°$ and latitude from $40.3°$ to $41.3°$), which includes 264,346 nodes and 733,846 edges, and the Chicago area (with longitude from $-87.94°$ to $-87.53°$ and latitude from $41.64°$ to $42.02°$), which includes 57,181 nodes and 175,416 edges.

Taxi Trips. The NYC's taxi trip data set is provided by the NYC's Taxi and Limousine Commission, which contains 14,776,615 taxi trip records from Feb. 2013 in the NYC area. The Chicago taxi trip data set is provided by the City of Chicago on the Chicago Data Portal, which contains over 100 million taxi rides in Chicago dating back to 2013. Each taxi trip record contains the pickup location and timestamp, the drop-off location and other information. Figure 7 shows the distribution of time costs of two taxi trip data sets. In the two data sets, more than half of the taxi trips require less than 1,000 seconds to finish, which can be treated as short trips.

Geo-Social Networks. Gowalla is a location-based social networking website where users share their locations by checking-in. Gowalla Check-in data set was collected by E. Cho et al [12], which includes a friendship network consisting of 196,591 nodes (users) and 950,327 undirected edges (friendship connections), and 6,442,890 check-ins of these users over the period of Feb. 2009 - Oct. 2010. We filter out the subset of the data set in the NYC area, which includes 159,257 check-in records.

7.1.2 Experiment Configuration

For the experiments on the real data set, we use the pickup location and timestamp of one record of NYC's taxi trip data set to set up the pickup location and timestamp of one rider. To configure the initial location of vehicles at a given timestamp t_j, we select a set T of taxi trip records ending in a time frame f_j of range of $[t_j - \delta_j, t_j]$, where δ_j is the length of frame f_j in time units, then for each record in T we set up the location of one vehicle as the drop-off location of the trip record.

For the experiments on the synthetic data set, we do not randomly generate the riders and the available vehicles on the road networks, but simulate them through mining the taxi trip data set with the simulation method in [25]. According to the observations in [25], in a time frame f_j (a period of time), the arrival of riders on each road node u_i approximately follow a Poisson distribution. Then, for time frame f_j, we denote the number of riders that originate from u_i as nr_i^j, and estimate the parameterλ_i^j of the Poisson distribution for road edge u_i in time frame f_j with Eq. 11 below:

$$\lambda_i^j = nr_i^j/\delta_j, \tag{11}$$

where δ_j is the length of frame f_j in time units.

For the transition probability p_{ik}^j from nr_i to nr_k during the time frame f_j, we can estimate with Eq. 12 below:

$$p_{ik}^j = nr_{ik}^j/c_i^j, \tag{12}$$

Table 3: Experimental Settings.

Parameters	**Values**
the number, m, of riders	1K, **3K**, 5K, 8K, 10K
the number, n, of vehicles	100, **200**, 300, 400, 500
the pickup deadline range $[rt^-_{min}, rt^-_{max}]$	**[1, 10]**, [10, 30], [30, 60]
the capacity of vehicles a_j	2, **3**, 4, 5
the balancing parameters (α, β)	(0, 0), (1, 0), (0, 1), **(0.33, 0.33)**
the flexible factor ε	1.2, **1.5**, 1.7, 2
the length δ_j of time frame f_j	30 mins

where nr^j_{ik} is the number of riders that start from edge u_i and end at edge u_k in time frame f_j.

Then, for a given time frame f_j, we first generate the ride requests originating on each road edge u_i, and generate the destinations of rider according to the transition probability p^j_{ik}.

To simulate the initial locations of vehicles, we assume the appearance of vehicles on each road node u_i also approximately follows a Poisson distribution. Different from the simulation of the arrival of riders, we use the drop-off locations and timestamps to mine the parameter λ^j_i of the Poisson distribution of the appearance of vehicle on the node u_i in time frame f_j. For all initialized vehicles, we assume there are no riders in them at the beginning.

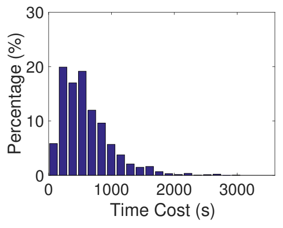
In addition, for both real and synthetic data sets, we simulate the drop-off deadlines through adding time differences calculated by multiplying the shortest travel cost $cost'$ with a flexible factor ε. Here, we assume the drivers in the taxi trip data set are experienced and can deliver their passengers to the destinations with the minimum travel cost. For the pickup deadlines, we generate them following Uniform distribution in the given time range $[rt^-_{min}, rt^-_{max}]$. Then, for a trip tr^j_{ik} from node u_i to node u_k in time frame f_j, we use the average travel cost of all the trips from node u_i to node u_k in the same time frame f_j.

Next, we map the riders and drivers to the geo-social networks. Specifically, we use the pickup locations of riders and initial locations of drivers as their current locations respectively. Then, for a rider/driver, we search the closest check-in record in Gowalla data set in the current time frame, and use the social relationships of the corresponding Gowalla user to initialize the rider/driver.

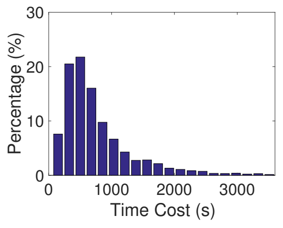
7.1.3 URR Approaches and Measurements

We conduct experiments to evaluate the effectiveness and efficiency of our three approaches, which includes *bilateral arrangement* (BA), *efficient greedy* (EG) and *grouping-based scheduling* (GBS), in terms of the total utility value and the running time. Specifically, for GBS, we can have two different combinations: GBS+BA and GBS+EG, which use BA and EG as the base methods to arrange the trips in each constructed area, respectively. In addition, we compare our approaches with one baseline method, namely *cost-first greedy* (CF), which greedily selects rider-and-vehicle pairs with the lowest incremental travel cost in each iteration. As proved in Section 2.6, the URR problem is NP-hard and unlikely to be approximated within any constant factor in polynomial time, and thus it is infeasible to calculate the optimal result as the ground truth. Alternatively, we have conducted a set of experiments on a small-size URR instance with 3 vehicles and 8 riders. Then we enumerate the optimal result and compare it with our URR approaches' results.

Table 3 introduces our experiment settings, where the default values of parameters are in bold font. In each set of experiments, we vary one parameter, while setting other parameters to their default values. For each experiment, we report the running time and the assignment score of our tested approaches. All our experiments were run on an Intel Xeon X5675 CPU @3.07 GHZ with 32 GB RAM in Python.



(a) NYC Taxi Trips



(b) Chicago Taxi Trips

Figure 7: Distribution of Time Costs of Taxi Trips.

7.2 Experimental Results

7.2.1 Experiments on Real Data

In this section, we show the effects of the range $[rt^-_{min}, rt^-_{max}]$ of the pickup deadline and the capacity of vehicles a_j.

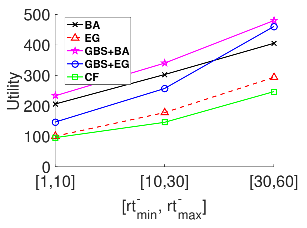
Effect of the Range, $[rt^-_{min}, rt^-_{max}]$**, of Pickup Deadlines.** Figure 8 illustrates the experimental results on different ranges, $[rt^-_{min}, rt^-_{max}]$, of pickup deadlines of riders from [1,10] to [30, 60] (in minutes), where other parameters are set to their default values. In Figure 8(a), the overall utilities of all tested approaches increase, when the value range of pickup deadlines gets larger. When the pickup deadlines of riders increase, more vehicles become valid to arrive at the pickup locations of riders. Then, for the riders, they have more potential vehicles to select, which allows the test approaches to arrange riders to more suitable vehicles with higher probabilities. As a result, the overall utilities of the tested approaches increase. Among the tested approaches, GBS+BA and BA can achieve similar utility value, which are higher than the utilities of other approaches. The utilities of GBS+EG are much higher than that of EG. The reason will be discussed in Section 7.2.2. As a baseline algorithm, CF is less effective than our URR approaches. As shown in Figure 8(b), the baseline algorithm, CF, runs the fastest, as it only needs to select the rider-and-vehicle pair with the lowest travel cost. BA needs to devote a lot of effort on adjusting the assignments of riders to vehicles to search for better rider-and-vehicle pairs with higher utilities and lower travel costs, which causes BA runs slowest among all the tested approaches. EG needs to calculate the utility efficiencies for each possible rider-and-vehicle pair, which is a little more complex than CF, thus EG needs similar time costs with CF. Particularly, our GBS approach can reduce the time complexity through efficiently grouping the trips into different areas, especially when we can choose a suitable parameter k with our cost-model-based estimation. As a result, GBS+BA and GBS+EG are faster than BA and EG, respectively.

Effect of the Capacity of Vehicles a_j**.** Figure 9 shows the experimental results with different vehicle capacities a_j from 2 to 5 over real data, where other parameters are set to their default values. In Figure 9(a), the utilities of tested approaches increase slightly, when the vehicle capacity a_j increases. The reason is that a higher capacity can allow each vehicle serve more riders, thus improve the total served riders and the overall utilities. GBS+BA can achieve the highest entire utility values. The utility values of EG are lower than BA and GBS+EG, but still much higher than that of BA and CF. In Figure 9(b), the vehicle capacities a_j also have almost no effect on the running time of the four approaches. BA is the slowest and CF is the fastest. GBS+EG is a little bit faster than EG.

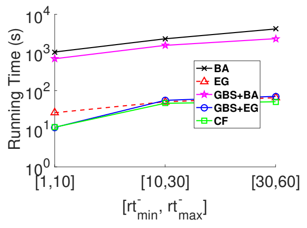
The experimental results of varying the range $[rt^-_{min}, rt^-_{max}]$ of the pickup deadline and the capacity of vehicles a_j on Chicago data set are similar to that on NYC data set. For the details, please refer to Appendix D.

7.2.2 Experiments on Synthetic Data

In this section, we first show the effectiveness of our URR approaches by comparing their results with the enumerated optimal

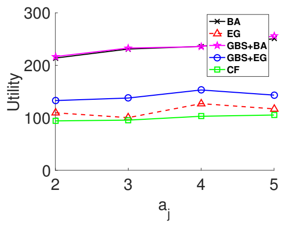


(a) Utility Value

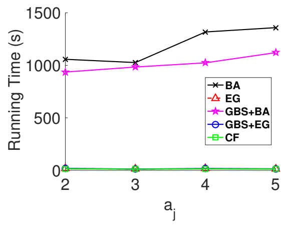


(b) Running Time

Figure 8: Effect of the Range of Pickup Deadlines $[rt^{-}_{min}, rt^{-}_{max}]$ (NYC).



(a) Utility Value



(b) Running Time

Figure 9: Effect of the Capacity of Vehicles a_j (NYC).

result (OPT) on a small scale URR instance (with 3 vehicles and 8 riders). Then, we test the effectiveness and scalability of our URR approaches compared with the baseline method, CF, by varying the balancing parameters (α, β), the flexible factor ϵ, the number, m, of riders and the number, n, of vehicles on the synthetic data sets.

Results of a small scale URR instance.

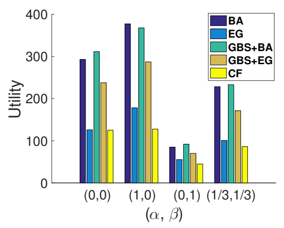
To show the effectiveness of our URR approaches, we compare the results achieved by our URR approaches with the optimal result (calculated through enumeration) on a small scale URR instance with 3 vehicles and 8 riders. As shown in Table 4, the optimal utility is 2.047802. Our BA can achieve a very close utility value, 1.741861. EG is better than CF in achieving high-utility results.

Table 4: Results of the Small Scale URR Instance.

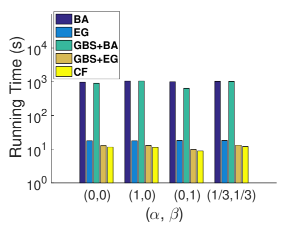
Approaches	Utility	Running Time
BA	1.741861	0.002197
EG	0.806841	0.002411
GBS+BA/EG	-	-
CF	0.635385	0.001287
OPT	2.047802	7218.234246

Particularly, BA runs 10^6 times faster than naively enumeration. EG is better than CF, but less effective than BA. As the URR instance is too small, GBS-related approaches cannot divide the trips into different areas, thus we did not test them on this URR instance.

Effect of the balancing parameters (α, β). Figure 10 shows the effect of the balancing parameters, (α, β), in Equation 1, by changing α and β with four sets of values in Table 3, where other parameters are set to their default values. The utilities of tested approaches are shown in Figure 10(a), the utilities achieved by GBS-related approaches (GBS+BA and GBS+EG) are usually higher than their base methods. For example, GBS+EG is always better than EG w.r.t. the utilities. The reason is when we divide the riders into different groups and solve them one-by-one, we can avoid the distraction from other riders in different groups, which may lead the flexible times of the vehicles tight. With more flexible time, we can arrange riders to more suitable vehicles. What is more, as BA can adjust the rider-and-vehicle pairs to search for better arrangement, it can repair the bad rider-and-vehicle pairs of the subsequent scheduled riders, which has a similar effect of GBS. Thus, GBS+BA cannot significantly improve the performance of BA w.r.t. the utilities and GBS+BA sometimes achieves a lower utility than BA (i.e., when $\alpha = 1$ and $\beta = 0$). Particularly, the utility values are relatively low when $\alpha = 0$ and $\beta = 1$ (i.e., only the rider-related utility is counted), as the social similarities of riders are usually very low (e.g., 0.1) calculated with Equation 3. Moreover, we notice that the utility of EG and CF are almost same when $\alpha = 0$ and $\beta = 0$ (i.e., only the trajectory-related utility is counted). The reason is, for each rider, his/her the trajectory-related utility is high when the travel cost is low, which leads EG and CF greedily selecting similar rider-and-vehicle pairs in each iteration. As shown in Figure 10(b), CF runs fastest as its simple mechanism. GBS+BA and GBS+EG are faster than BA and EG, respectively, but EG is still much faster than GBS+BA. In addition, we notice the balancing parameters have very little effect on the speed of tested approaches, as the balancing parameters just influence the priorities of rider-and-vehicle pairs to select.

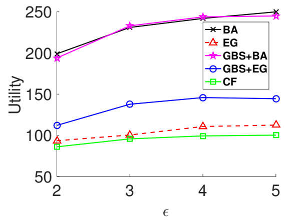


(a) Utility Value

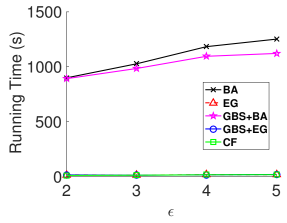


(b) Running Time

Figure 10: Effect of the Balancing Parameters (α, β) (Synthetic).



(a) Utility Value



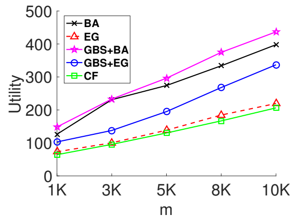
(b) Running Time

Figure 11: Effect of the flexible factor ϵ (Synthetic).

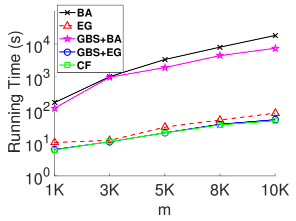
Effect of the flexible factor ϵ. Figure 11 presents the effect of the flexible factor, ϵ, by varying ϵ from 1.2 to 2.0, while other parameters are set to their default values. As shown in Figure 11(a), GBS+BA and BA can achieve similar utilities, which are higher than those of the other tested approaches. Due to the same reason in the above discussion of the effect of balancing parameter, GBS+EG can achieve higher utilities compared with EG. Most importantly, when the flexible factor ϵ increases, the utilities of all the tested approaches increase at the same. The reason is that larger flexible factor means riders can accept longer detours/delays, which enables vehicles to serve more riders.

In Figure 11(b), the running time of all the tested approaches increase when ϵ increases. The reason is that when riders allow more detours/delays (i.e., the flexible factor is larger), more other riders will be valid for each vehicle, which causes the tested approaches spending more time to handle more valid rider-and-vehicle pairs. GBS+BA is faster than BA. EG is slightly slower than GBS+EG, but much faster than GBS+BA. CF is still the fastest.

Effect of the Number of Riders, m. Figure 12 illustrates the effect of the number, m, of riders, by varying m from 1K to 10K, where other parameters are set to their default values. As shown in Figure 12(a), when the number of riders increases, the entire utility values of the results achieved by all the approaches will increase. The reason is that when more riders are available, they will be served and satisfied by the vehicles until the vehicles cannot handle the ride requests. As a result, the utilities of EG and GBS increase fast when m increases from 1K to 3K, however, they increase slowly when m gets larger than 3K. The reason is when the riders become

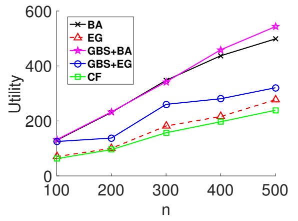


(a) Utility Value

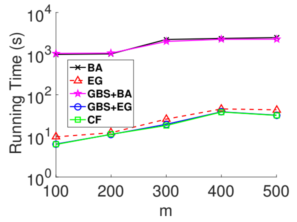


(b) Running Time

Figure 12: Effect of the Number of Riders m (Synthetic).



(a) Utility Value



(b) Running Time

Figure 13: Effect of the Number of Vehicles n (Synthetic).

saturated, all the capabilities of vehicles are used and no more utility increases can be achieved. In Figure 12(b), when m increases, the running times also increase. This is because, we need to deal more rider-and-vehicle pairs for a large m. Again, EG is slower than GBS+EG but faster than BA and GBS+BA. In addition, the baseline CF is the fastest as its simple mechanism.

Effect of the Number of Vehicles, n. Figure 12 shows the experimental results with different numbers of vehicles, n, from 100 to 500 over synthetic data, where other parameters are set to their default values. Similar to previous results about the effect of m, in Figure 12(a), our URR approaches can obtain good results with high entire utility values, compared with the baseline approach, UF. Moreover, when the number, n, of workers increases, the utilities achieved by all tested approaches also increase. The reason is that when the number of n increases, the riders have more valid vehicles to be assigned to, which may alleviate the competition of vehicles among riders and lead to higher overall utilities.

In Figure 13(b), the running time of the tested approaches increases, with the increasing number of vehicles. The reason is that when more vehicles are available, we need to handle more valid rider-and-vehicle pairs, which enlarges the problem space. Similarly, BA runs slowest and GBS is faster than EG. As a baseline, due to its simpleness, CF runs fastest.

In summary, over both real data and synthetic data sets, BA can achieve the highest overall utilities, but it runs the slowest among the tested approaches. GBS can increase the running speeds compared with the corresponding base method (i.e., BA). More important, when we use EG to solve the subproblems of GBS, GBS+EG can always achieve much higher utilities than EG.

8. RELATED WORK

Recent years, with the popularity of GPS-equipped smart devices and social networks, people can join ridesharing services conveniently. On the other hand, the emergence of online-to-offline trip markets (e.g., Uber [7] and Lyft [3]) have enabled their users to directly contact drivers to reserve personalized traveling services. As a more efficient alternative, Uber-like companies have started to provide sharing-traveling services, which allow customers to share vehicles with other customers, then reduce the travel cost (as it is distributed to all the customers in the same trip). Though ridesharing and real-time taxi-sharing have been studied in several previous works [21, 16, 19, 20], they mainly focus on the efficiency of vehicles, which is to provide ridesharing services to customers to guarantee required constraints at minimum travel costs. However, saving travel cost is not a crucial point any more, as more and more companies [4, 8] have launched unlimited ride packages to provide unlimited ridesharing services for a period of registered time. Thus, the customers involved in ridesharing activities are not only for saving money but also for having better user experiences, which is measured with utility values in this work. Specifically, we take into consideration the vehicle-related utility, the riders-related utility and the trajectory-related utility.

The utility-aware ridesharing problem can be viewed as a variant of the dial-a-ride problem (DARP), which designs vehicle routes and schedules for n riders who specify pick-up and drop-off requests between origins and destinations [14]. Existing works on DARP have primarily focused on the static DARP, where all customer ride requests are known in prior [21, 35, 13]. The general DARP is NP-hard and intractable, only small instances can be solved exactly [13]. Several works on DARP focus on single-vehicle problem instances, where only one vehicle is considered to schedule riders and plan the travel route [27, 26]. Some works group the ride requests with bounded distance errors, then use heuristic algorithms to dispatch a set of riders to each vehicle to provide a high running efficiency such that the ridesharing services can be used in on-line situations with large scale existing ride requests [19]. For online ridesharing/taxi-sharing, existing works [25, 20] focus on fast responses to the arrival riders one by one without guaranteeing the global optimality. In [25], they proposed one framework to handle the online taxi-sharing problem, where riders and taxis keep arriving and leaving. Their framework responds to each coming rider by scheduling it to the most suitable vehicle such that the time window and monetary constraints are satisfied. According to their experimental study, they pointed out that schedule reordering is not necessary for large scale ridesharing services, as the time cost will increase a lot but the effectiveness measurements change very little. In [20], the researchers propose a kinetic tree structure to trace the valid schedule plans for each vehicle. When a new rider arrives, they provide the best route among all the vehicles with the least travel cost increase. However, our URR problem takes three components of utility into consideration and targets on maximizing the overall utility of riders such that the user experience can be improved, thus no previous solutions can be used directly. To solve the URR problem, we propose two approximate algorithms, then propose a grouping-based algorithm to improve the running speed.

9. CONCLUSION

In this paper, we propose the problem of utility-aware ridesharing on road networks (URR), which assigns time-constrained riders to capacity-constrained and dynamically moving vehicles, such that the entire utility value, which includes the vehicle-related utility, the riders-related utility, and the trajectory-related utility, is maximized. We prove that the URR problem is NP-hard, and thus we propose three approximate approaches (i.e., a bilateral arrangement algorithm, an efficient greedy algorithm and a grouping-based scheduling algorithm), which can efficiently retrieve URR answers. Extensive experiments have shown the efficiency and effectiveness of our URR approaches on both real and synthetic data sets.

10. ACKNOWLEDGMENT

This work is supported in part by Hong Kong RGC project 16202215, NSFC Grant No. 61328202, NSFC Guang Dong Grant No. U1301253, National Grand Fundamental Research 973 Program of China under Grant 2014CB340303, HKUST-SSTSP FP305, Microsoft Research Asia Collaborative Grant, Google Faculty Award 2013 and ITS170.

APPENDIX

A. REFERENCES

[1] [online] 9th dimacs implementation challenge - shortest paths. http://www.dis.uniroma1.it/challenge9/.

[2] [online] gowalla check-in dataset. https://snap.stanford.edu/data/loc-gowalla.html.

[3] [online] lyft. https://www.lytf.com.

[4] [online new york post news] apps see surge in riders willing to get comfy with strangers. http://nypost.com/2016/08/13/apps-see-surge-in-riders-willing-to-get-comfy-with-strangers.

[5] [online] taxi trip data. http://chriswhong.com/open-data/foil_nyc_taxi/.

[6] [online] taxi trips in city of chicago data portal. https://data.cityofchicago.org/Transportation/Taxi-Trips/wrvz-psew.

[7] [online] uber. https://www.uber.com.

[8] [online] via. https://www.via.com.

[9] A. Agresti and M. Kateri. *Categorical data analysis*. Springer, 2011.

[10] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *ACM WWW*, 2007.

[11] T. Y. Berger-Wolf and J. Saia. A framework for analysis of dynamic social networks. In *ACM SIGKDD*, 2006.

[12] E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *ACM SIGKDD*, 2011.

[13] J.-F. Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 2006.

[14] 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*, 2003.

[15] W. E. de Paepe, J. K. Lenstra, J. Sgall, R. A. Sitters, and L. Stougie. Computer-aided complexity classification of dial-a-ride problems. *IJOC*, 2004.

[16] P. M. d'Orey, R. Fernandes, and M. Ferreira. Empirical evaluation of a dynamic and distributed taxi-sharing system. In *ITSC*. IEEE, 2012.

[17] U. Feige, D. Peleg, and G. Kortsarz. The dense k-subgraph problem. *Algorithmica*, 2001.

[18] S. Funke, A. Nusser, and S. Storandt. On k-path covers and their applications. *PVLDB*, 2014.

[19] G. Gidofalvi, T. B. Pedersen, T. Risch, and E. Zeitler. Highly scalable trip grouping for large-scale collective transportation systems. In *EDBT*. ACM, 2008.

[20] Y. Huang, F. Bastani, R. Jin, and X. S. Wang. Large scale real-time ridesharing with service guarantee on road networks. *PVLDB*, 2014.

[21] E. Kamar and E. Horvitz. Collaboration and shared plans in the open world: Studies of ridesharing. In *IJCAI*, 2009.

[22] S. Khot. On the power of unique 2-prover 1-round games. In *ACM STOC*, 2002.

[23] E. A. Leicht, P. Holme, and M. E. J. Newman. Vertex similarity in networks. *Phys. Rev. E*, Feb 2006.

[24] K. Li, W. Lu, S. Bhagat, L. V. Lakshmanan, and C. Yu. On social event organization. In *ACM SIGKDD*, 2014.

[25] S. Ma, Y. Zheng, and O. Wolfson. Real-time city-scale taxi ridesharing. *TKDE*, 2015.

[26] H. N. Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 1980.

[27] H. N. Psaraftis. An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. *Transportation science*, 1983.

[28] P. Raghavendra and D. Steurer. Graph expansion and the unique games conjecture. In *ACM STOC*, 2010.

[29] M. Rice and V. J. Tsotras. Graph indexing of road networks for shortest path queries with label restrictions. *PVLDB*, 2010.

[30] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial intelligence*, 1998.

[31] C. Tantipathananandh, T. Berger-Wolf, and D. Kempe. A framework for community identification in dynamic social networks. In *ACM SIGKDD*, 2007.

[32] Y. Tao, C. Sheng, and J. Pei. On k-skip shortest paths. In *ACM SIGMOD*, 2011.

[33] N. Taxi, L. Commission, et al. Taxicab factbook. *New York Taxi and Limousine Com http://www.nyc.gov/html/tlc/downloads/pdf/2014_taxicab_fact_book.pdf*, 2014.

[34] V. V. Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[35] K. Wong and M. G. Bell. Solution of the dial-a-ride problem with multi-dimensional capacity constraints. *ITOR*, 2006.

B. PROOF OF THEOREM 2.1 (HARDNESS OF THE URR PROBLEM)

Proof. We prove the theorem by a reduction from the 0-1 Knapsack problem [34]. A 0-1 knapsack problem can be described as follows: given a set of m items numbered from 1 up to m, each with a weight w_i and a value v_i, along with a knapsack that has a maximum weight capacity W, the problem is to maximize the sum of the values of the items in the knapsack so that the sum of the weights is less than or equal to the knapsack's capacity W.

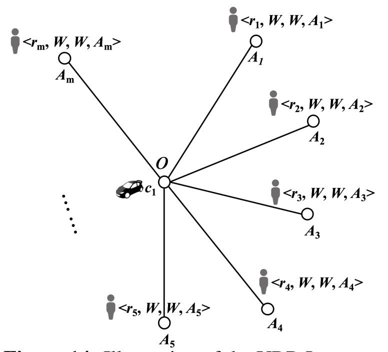


Figure 14: Illustration of the URR Instance.

For a given knapsack problem, we can transform it to an instance of URR shown in Figure 14 as follows: we give only one vehicle c_j with $k(\geq 1)$ capacity located at a node o with no riders. In addition, we give a set of m ride requests at different locations, and the pickup deadlines and delivery deadlines of requests are set to W. Moreover, the destinations of the requests are the same as the riders' current locations respectively. Then, we set the traveling time from node o to a node A_i as $w_i/2$ and the utility value of ride request q_i as v_i. Then, for this URR instance, we want to arrange a schedule for the given vehicle such that the summation utility value is maximized.

As the destinations of the requests are the same as the current locations of the riders, each time the vehicle moves to some node A_i, then it can satisfy the rider immediately. In addition, as the traveling time from node o to any other node A_i is $w_i/2$ and no routes between the locations of requests, to serve a request q_i and continue to the other ones, the vehicle needs to go to node A_i then go back to node o, which leads to a traveling time cost of w_i and increases the summation of utility value with v_i. As the deadlines of all the requests are set to W, the vehicle can finish ride requests before W. Thus, to maximize the summation utility value satisfying the deadlines is same as to maximize the total value of the items in the knapsack problem under the constraint of the capacity of the given knapsack. Given this mapping it is easy to show that the knapsack problem instance can be solved if and only if the transformed URR problem can be solved.

This way, we can reduce the knapsack problem to the URR problem. Since the knapsack problem is known to be NP-hard [34], URR is also NP-hard, which completes our proof. □

C. PROOF OF THEOREM 2.2

Proof. We prove the theorem by a reduction from the DENSE k-SUBGRAPH problem [17]. The DENSE k-SUBGRAPH problem can be described as follows: given a graph $G = (V, E)$ (on n vertices) and a parameter k, the problem is to find a subgraph $G' = (V', E')$ of G induced on k vertices, such that the density of $G' = \frac{2|E'|}{|V'|} (= \frac{2|E'|}{k})$ is maximized.

For a given DENSE k-SUBGRAPH problem instance $\mathcal{I}$ defined by a graph $G = (V, E)$ (on n vertices) and a positive integer k, we can create an URR instance $\mathcal{J}$ as follows: For any node v_i in G, we create a rider r_i in R. Then for any edge $(v_i, v_j) \in E$, we set $s(r_i, r_j) = 1$ (i.e., the similarity of the corresponding rider pair is 1); otherwise, $s(r_i, r_k) = 0$ if $(v_i, v_k) \notin E$. Let $C = \{c_x\}$ and $a_x = k$ (i.e., there is only one vehicle c_x and its capacity is k). In addition, we set $\beta = 1$, then the utility $\mu(r_i, c_x) = \mu_r(r_i, c_x)$ (i.e., we only consider the rider-related utilities). The road network has only two nodes o_1 and o_2, and $l(c_x) = s_i = o_1, e_i = o_2, \forall r_i \in R$ (i.e., the vehicle c_x and all the riders are located at o_1 and the destinations of all the riders are o_2). What is more, the delivery deadlines of riders are set as $rt_i^+ = cost(o_1, o_2), \forall r_i \in R$, which means it is just sufficient for vehicle c_j to move from o_1 to o_2 once.

As a result, to solve the URR instance $\mathcal{J}$ is to select a subset R' of k riders for vehicle c_x to deliver from o_1 to o_2 such that the overall utility is maximized, where the overall utility is

$$\begin{aligned} & \sum_{r_i \in R'} \sum_{r'_i \in R' - \{r_i\}} \frac{s(r_i, r'_i)}{|R' - \{r_i\}|} \\ = & \sum_{r_i \in R'} \sum_{r'_i \in R' - \{r_i\}} \frac{s(r_i, r'_i)}{k-1} \\ = & \frac{2|E'|}{k-1} \qquad (13) \end{aligned}$$

where E' is the edge set of the subgraph induced on the k corresponding vetrices of the selected subset R' of riders. Equation 13 holds, as $s(r_i, r'_i)$ equals to 1 only when the corresponding edge exists in G (also in G'). Thus, as k is a given constant, to maximize the overall utility of the solution for $\mathcal{J}$ is same to maximize the density of the selected subgraph G' for $\mathcal{I}$.

Suppose there is a PTIME algorithm $\mathcal{A}$ approximating URR within a factor of $c \in (0, 1)$, then there must be a PTIME algorithm $\mathcal{B}$ that can approximate DENSE k-SUBGRAPH within the same factor through converting $\mathcal{I}$ to $\mathcal{J}$ as above, running $\mathcal{A}$ on $\mathcal{J}$, and outputting the vertices corresponding the the selected riders R' by $\mathcal{A}$. This cannot be true unless the Unique Games with Small Set Expansion Conjecture does not hold. □

D. EXPERIMENTAL RESULTS ON CHICAGO DATA SET

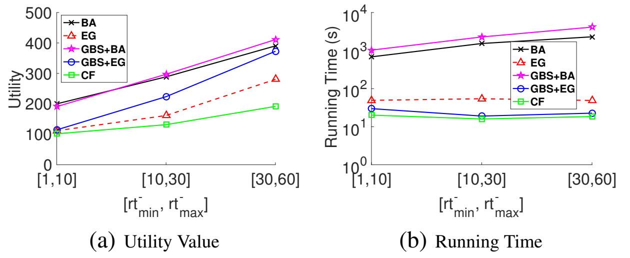


Figure 15: Effect of the Range of Pickup Deadlines $[rt^-_{min}, rt^-_{max}]$ (Chicago).

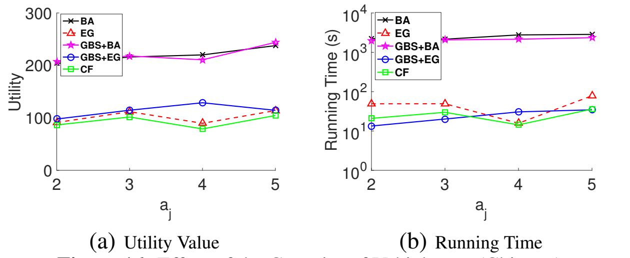


Figure 16: Effect of the Capacity of Vehicles a_j (Chicago).

The results of the experiments on Chicago data set are similar to the results of the experiments on NYC data set. Specifically, in Figure 15, when the range of the pickup deadline of riders gets larger, the utilities of all the tested approaches increase at the same time. GBS+BA can achieve the close utility values compared with BA, and they are better than other approaches w.r.t. the utility of results. CF reports the fastest results with the lowest utilities. GBS-related approaches (GBS+BA and GBS+EG) run faster than their base methods (e.g., BA and EG). As for the effect of the capacity, a_j, of vehicles, the utilities achieved by the tested approaches increase slightly, when the vehicle capacity a_j increases. The reason is that a higher capacity can allow each vehicle to serve more riders, thus improve the total served riders and the overall utilities. GBS+BA and GBS+EG usually achieve higher utilities than their base methods, BA and EG, respectively. GBS+EG is still worse than BA w.r.t. the achieved utilities.

The observation of the experimental results on Chicago data set is similar to that on NYC data set. Our BA algorithm is good at achieving high utilities, but runs very slow. GBS can improve the running speed of its base method and usually achieve higher utilities than its base method.