Demand-Aware Route Planning for Shared Mobility Services

Jiachuan Wang [†], Peng Cheng [*], Libin Zheng [†], Chao Feng [‡], Lei Chen [†], Xuemin Lin [#,*], Zheng Wang [‡]
[†]*The Hong Kong University of Science and Technology, Hong Kong, China*
{jwangey, lzhengab, leichen}@cse.ust.hk
[*]*East China Normal University, Shanghai, China*
pcheng@sei.ecnu.edu.cn
[#]*The University of New South Wales, Australia*
lxue@cse.unsw.edu.au
[‡]*AI Labs, DiDi Chuxing, Beijing, China*
fengchaodavid@didichuxing.com, wangzhengzwang@didiglobal.com

ABSTRACT

The dramatic development of shared mobility in food delivery, ridesharing, and crowdsourced parcel delivery has drawn great concerns. Specifically, shared mobility refers to transferring or delivering more than one passenger/package together when their traveling routes have common sub-routes or can be *shared*. A core problem for shared mobility is to plan a route for each driver to fulfill the requests arriving dynamically with given objectives. Previous studies greedily and incrementally insert each newly coming request to the most suitable worker with a minimum travel cost increase, which only considers the current situation and thus not optimal. In this paper, we propose a demand-aware route planning (DARP) for shared mobility services. Based on prediction, DARP tends to make optimal route planning with more information about requests in the future. We prove that the DARP problem is NP-hard, and further show that there is no polynomial-time deterministic algorithm with a constant competitive ratio for the DARP problem unless P=NP. Hence, we devise an approximation algorithm to realize the insertion operation for our goal. With the insertion algorithm, we devise a prediction based solution for the DARP problem. Extensive experiment results on real datasets validate the effectiveness and efficiency of our technique.

PVLDB Reference Format:
Jiachuan Wang, Peng Cheng, Libin Zheng, Chao Feng, Lei Chen, Xuemin Lin, Zheng Wang. Demand-Aware Route Planning for Shared Mobility Services. *PVLDB*, 13(7): 979-991, 2020.
DOI: https://doi.org/10.14778/3384345.3384348

1. INTRODUCTION

Recently, shared mobility provides transportation services shared among users, such as food delivery, ride-sharing, and crowdsourced parcel delivery [36], which is considered as an efficient and sustainable way for transportation. With higher usage per worker, its service saves energy, reduces air pollution, and handles last-mile delivery [37].

To realize shared mobility on online platforms (e.g., DiDi Chuxing [1] and Meituan [3]), an essential issue is *route planning*. Given a set of workers and requests, route planning designs a route consisting of a sequence of pickup and drop off locations of assigned requests for each worker. In shared mobility, workers and requests arrive dynamically and are online arranged by platforms for different objectives. The objectives could be maxmizing the number of served requests [14, 18, 27, 35, 44], minimizing the total tour distance [7, 11, 20, 22, 23, 30, 32, 34, 35, 38], and maximizing the unified revenue [9, 10, 40]. To solve such a dynamical problem, an operation called *insertion* shows great effectiveness and efficiency [30, 23, 38, 32, 35, 44, 12, 14, 40]. *Insertion* aims to serve a newly coming request by inserting its origin and destination into a worker's route and greedily optimizing its objectives.

One key issue of the state-of-the-art route planning methods [10, 12, 40] is that *insertion* is a "greedy" operator, which means each insertion operation only optimizes the objective function upon the "current" arranged request leading to low performance during a long period(e.g., one day). In this paper, we will improve the effectiveness of route planning for shared mobility through taking the demand in near future (e.g., future 90 minutes) into consideration such that the objectives of the platforms can be improved for a relatively long period (e.g., one day). We first illustrate our motivation through the following example.

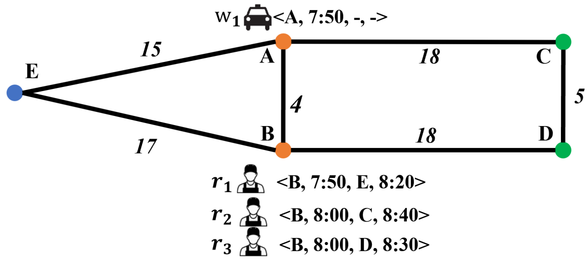


Figure 1: An Illustrative Example

Example 1. *In this ridesharing example, we assume that one vehicle w_1 and three riders $r_1 \sim r_3$ are on a road network as shown in Figure 1. Specifically, there are five nodes in the road network indicating five locations, and the numbers on the edges represent their travel cost. For the vehicle and riders, they are located at their current locations and the 4-tuple near to each one follows the pattern of $\langle$release location, release time, destination, deadline of delivery$\rangle$. For example, rider r_1 is located at A and releases his/her request at 7:50 indicating that he/she wants to be driven to location E before 8:20. At 7:50, the driver can serve r_1 to earn a profit routed as* **A→B→E**. *However, at 8:00 two other riders r_2 and r_3 want to reach node* **C** *and* **D** *from* **B** *before 8:40 and 8:30, respectively. If the vehicle w_1 rejects rider r_1, then riders r_2 and*

Proceedings of the VLDB Endowment, Vol. 13, No. 7
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3384345.3384348

*r_3 can be served with another route **A→B→D→C**. Existing solutions using greedy optimization objectives considering only current benefits (i.e., w_1 will only pick up r_1). The profit loss of these short-sighted methods leads to a puzzle that most online models face: how to consider the mobility of workers comprehensively in a more reliable model with effective assignment strategies, which realizes a spatial and temporal balance such that the global profit of the dynamic assignment is maximized?*

To address the issue illustrated in Example 1, we define a new problem, namely *Demand-Aware Route Planning* (DARP) problem. Specifically, DARP takes into account the potential profits from each assignment in the near future. The DARP problem has the flexibility to adjust the optimization goals subjected to specific requirements. We combine the mainstream optimization objectives with the demand-supply balance situations and propose a comprehensive objective function. As each insertion can reversely affect the demand-supply balance situation on the whole path, previous insertion algorithms [12, 14, 40, 30, 23, 44] only consider the effects of each scheduling on the travel cost at each insertion position, and thus can not be applied directly.

Based on previous studies on the hardness of route planning for shared mobility [9, 40], we prove that the DARP problem is NP-hard and there is no deterministic algorithm with constant competitive ratio for it. To solve the DARP problem, we first propose novel structures, demand number map and supply number map, to reflect and quantize the demand and supply in the particular spatiotemporal space based on the predicted distributions of upcoming workers and requests. In addition, we further present an indicator, *Demand-Supply Balance Score*, to guide the assignment/scheduling of requests. To handle every single request, we develop a demand-aware dynamic programing based insertion algorithm, namely DA-DP Insertion, to quickly arrange the origin and destination of the request considering the shift of supply on the whole route of the assigned worker due to detours. What is more, we design a demand-aware insertion based dual-phase framework (DAIF) to solve DARP, which first decide whether to serve a new request and then arrange the profitable request to the most suitable worker such that the incremental demand-aware cost is minimized and the overall serving rate is improved for a relatively long period.

Here we summarize our main contributions:

- We formulate a demand-aware route planning problem for shared mobility. We prove its NP-hardness and no deterministic algorithm can offer a constant competitive ratio for it in Section 3.
- Through detailed observations and analyses, we propose a new indicator, named Demand-Supply Balance Score, to quantize the potential profits of an assignment in Section 4.
- We devise a demand-aware dynamic programing based insertion algorithm to handle a new request taking demand-supply balance into account in Section 5. In addition, we develop a demand-aware insertion based dual-phase framework (DAIF) to improve the overall performance for a relatively long period in Section 6.
- We have conducted extensive experiments on synthetic and real data sets to show the efficiency and effectiveness of proposed algorithms in Section 7.

2. BACKGROUND AND RELATED WORK

Route planning for shared mobility (RPSM) is a variant of the *dial-a-ride* problem, which has been widely studied since 1975 [41, 42] and drawn great attention recently [6, 33]. Along with the popularity of the car-hailing platforms (e.g., Uber [5]), online RPSM problems without information of future workers requests in advance are studied in many recent papers [10, 14, 23, 30, 38, 44].

For online RPSM problems, previous solutions usually continuously add each up-coming request into the local optimal route instead of the global optimal. The objectives are mainly about: *(i)* minimizing the total travel distance [7, 11, 20, 22, 23, 30, 32, 34, 35, 38]; *(ii)* maximal-served-request [14, 18, 27, 35, 44]; *(iii)* maximizing the total revenue [9, 10, 40]. Some other variants include minimizing the finished time [8, 46], maximizing the score combined with social utilities from both workers and requests [12, 19], and minimizing the total delay time [34, 47].

Most solutions for dynamic RPSM problems are based on a core operation called *insertion* [12, 14, 23, 25, 31, 32, 34, 35, 38]. Using the basic enumeration to find the best insertion positions in the existing scheduling of a given vehicle will result in $O(n^3)$ time complexity, where n is the number of the locations in the scheduling. [12, 14, 23, 44]. Tong *et al.* improve the insertion operation to $O(n)$ time complexity by dynamically deriving the best position for the pick-up location given a certain drop-off position [40]. Ota *et al.* apply parallelism to speed up insertion [32]. Coslovich *et al.* use a two-phase algorithm to enable negligible online time cost by conducting most computation off-line [16]. Tong *et al.* adapte the insertion operation for the optimization objective from the requests perspective with $O(n)$ time complexity [43]. However, in the previous linear approaches [40, 43], only the arriving times of destinations affect their objective function. It cannot be applied to our problem as the demand-aware cost consists of an accumulation of demand-and-supply balance scores of all the paths. Thus, we propose a novel insertor for DARP with consideration of demand.

As an on-line problem, existing solutions for dynamic RPSM are largely greedy-based. Zheng *et al.* [30, 38] use a grid index to filter available workers to improve the speed of their algorithms. Huang *et al.* [23] devise a kinetic data structure to store all possible routes such that the new insertion can achieve the optimal schedule for each selected single vehicle (but still not the global optimal). Zheng *et al.* propose a batch-based solution to group similar riders into a package and apply bipartite matching to the packages of grouped riders and drivers [50]. However, these studies only consider the locally maximal profit at every single insertion. In this paper, we seeks for the global optimal through considering the effect of an insertion operation in the short future.

To improve the results for dynamic cases, many previous studies related to traffic propose various prediction models and strategies to use them. He *et al.* [21] improve participant recruitment in crowdsourcing using predictable mobility. Tong *et al.* devise a method to improve online task assignment using predicted information of up-coming requests [39]. Prediction models to off-line predict the demand of requests in given regions and periods are well-studied, including demand-supply prediction of traffic [13, 29], and spatial-temporal data prediction [17, 49]. Zhang et. al. [48] propose a Deep ST model, which combines historical and geographical traffic data to achieve accurate prediction results. To the best of our knowledge, the existing studies cannot be directly applied to the demand-aware based route planning problems for shareable mobility services, and thus we propose our demand-aware route planning solutions to solve it.

3. PROBLEM DEFINITION

3.1 Basic Notations

The road network is represented as a graph $G = \langle V, E \rangle$, where V and E indicate a set of vertices and a set of edges, respectively. Each edge, $(u, v) \in E$, is associated with a weight $dis(u, v)$ indicating the travel distance from u to v. A sequence of adjacent vertices can form a path. From source u to v, the path with the shortest total distance is called the shortest path. We also use $dis(u, v)$ to indicate the length of the shortest path from u to v in the remaining part of the paper.

Definition 1. (Workers) Let $W = \{w_1, w_2, , w_n\}$ be a set of n workers that can provide transportation services. Each worker w_i is located at the current location l_i and has a capacity a_i.

At any time, the number of the passengers in a taxi or the number of parcels for a worker w_i must not exceed its capacity a_i.

Definition 2. (Time-Constrained Requests) Let $R = \{r_1, r_2, , r_m\}$ be a set of m requests. Each request r_j has its source location s_j, destination location e_j, release time tr_j, deadline for the destination td_j, rejection penalty p_j, and a capacity a_j.

A request r_j can be served by worker w_i only if: (a) the remaining capacity of w_i is larger than a_j when he/she arrives at s_j; (b) w_i can drop r_j at e_j before td_j.

Note that the request with a tight deadline may not be served, which leads to the unavoidable rejections of the "urgent" requests on a platform, especially at rush hours. The monetary loss from rejecting r_j incurs a penalty p_j. Furthermore, we denote all the requests served by worker w_i as R_{w_i}. In addition, $\hat{R} = \cup_{w_i \in W} R_{w_i}$ and $\bar{R} = R/\hat{R}$ refers to the total served and unserved requests, respectively.

Definition 3. (Route) For a worker w_i, his/her schedule is a sequence of locations represented as $S_{w_i} = [l_i, l_{x_1}, l_{x_2} l_{x_k}, ...]$, where each l_{x_k} is an origin or a destination of a request $r_j \in R_{w_i}$.

A feasible scheduling route S_{w_i} must satisfy: (a) $\forall r_j \in R_{w_i}$, the arriving time of e_j is earlier than td_j; (b) $\forall r_j \in R_{w_i}$, its destination e_j is behind its origin s_j in S_{w_i}; (c) the summation of the capacities of the requests is less than the capacity a_i of worker w_i at any time. We denote the summation of the shortest travel distances to finish S_{w_i} as $D(S_{w_i}) = dis(l_i, l_{x_1}) + \sum_{k=1}^{|S_{w_i}|-2} dis(l_{x_k}, l_{x_{k+1}})$. In addition, let $S = \cup_{w_i \in W} S_{w_i}$ be the overall set of scheduled routes for all workers.

Definition 4 (Spatial Temporal Cell)**.** Let C_{xy} be a spatial temporal cell of area N_x and time span T_y. A worker w_i appears in C_{xy} indicates that w_i is in N_x during whole or part of time span T_y.

Definition 5. (Demand Number Map) Given a time span set $\mathcal{T} = \{T_1, T_2, \cdots, T_{|\mathcal{T}|}\}$ and each T with T^+ and T^- referring to the start and end time respectively, a set of transportation areas of interest $\mathcal{N} = \{N_1, N_2, ..., N_{|\mathcal{N}|}\}$, and a prediction model M, the demand number map is the set of predicted numbers of upcoming requests for each spatial temporal cell $C_{ab} = \langle N_a, T_b \rangle, \forall N_a \in \mathcal{N}, T_b \in \mathcal{T}$ using model M.

The demand number map can be represented by a mapping function as $DN\,(N_a, T_b) \to dn$, where $N_a \in \mathcal{N}, T_b \in \mathcal{T}$ and dn is the predicted number of upcoming requests in area N_a in time span T_b. One critical parameter for the demand number prediction model is the grid-size of areas. A prediction model with a too large gird-size will provide no suggestion to route planning with vanished local properties while a model with too small grid-size will lead to unreliable results. In this paper, we follow the existing work [39] to use a block sized 2000m × 2000m to grid the whole area of interest.

Based on the demand number map DN and the overall route S in the areas of interest, the Demand-Supply Balance Score (DSBScore), $DSB(S, DN)$, is proposed to measure how much the underlying profit will be achieved through suiting the supply and potential demand for each newly served request. We go through the detail of the Demand-Supply Balance score in Section 4.

3.2 Demand-Aware Route Planning Problem

Definition 6. (Demand-Aware Route Planning Problem) Given a transportation network G, a set of workers W, a set of dynamically arriving requests R, a demand number map DN, a distance cost coefficient α, and a set of balance coefficients β associated with time spans, the DARP Problem is to find the sets of routes $S = \{S_{w_1}, S_{w_2}, ..., S_{w_{|W|}}\}$ for all the workers to minimize Demand-Aware Cost:

$$DAC(W, R, DN) = \alpha \sum_{w_i \in W} D(S_{w_i}) - DSB(\beta, S, DN) + \sum_{r_j \in \bar{R}} p_j \quad (1)$$

where $\bar{R}$ is the set of rejected requests and p_j is the penalty to reject request r_j. The later a time span is, the weaker the prediction is, which is represented by the decayed β. In addition, the following constraints must be satisfied: (a) Feasibility constraint: each worker is assigned with a feasible route; (b) Non-undo constraint: if a request is assigned in a route, it cannot be canceled or assigned to another route; if it is rejected, it is unable to be revoked.

In the real-world application, our demand-aware cost can be regarded as an equivalent money cost of the platform. To be more specific, α is the money paid to workers for every second they spend to serve requests; β is the money loss from one potentially unservable request; and p_j is the lost money for rejecting a request. Note that rejection affects not only the monetary profit but also the dissatisfaction of the rejected requester, who may leave the platform forever. In real applications, companies usually want to serve as many requests as possible. Thus, the penalty of rejection is usually set to be a large factor.

3.3 Hardness Analysis

The basic route planning problem for shareable mobility services [30, 40] is reducible to the DARP problem by setting $\alpha = 1$ and $\beta = 0$. Since the basic route planning problem is NP-hard [30, 40], the DARP problem is also an NP-hard problem.

To analyze online problems, a commonly used metric is the Competitive Ratio (CR), which is defined as the ratio between the result achieved by the proposed online algorithm to the optimal offline result. The existing studies proved that there is no constant CR to maximize the total revenue for the basic route planning problem using neither deterministic nor randomized algorithms [9, 40]. As we discussed above, the DARP problem is a variant of the basic route planning problem for shareable mobility services. Thus, no randomized or deterministic algorithm guarantees a constant CP for the DARP problem.

We will first introduce the details of the grid-based demand number map and the demand-supply balance score. Then propose our methods to solve the DARP problem.

4. DEMAND-SUPPLY BALANCE SCORE

To realize the approximation of global optimal arrangement in the DARP problem, we introduce Demand-Supply Balance Score (DSBScore) for a better assignment. In this section, we first propose the spatiotemporal prediction model, namely Grid-Based Demand Number Map, then we devise a reliable method to generate our DSBScore based on the prediction model.

4.1 The Grid-Based Demand Number Map

The DSBScore is based on the mobility calculated from the route of current workers and the predicted demand for "future" requests. One critical issue is to generate the Demand Number Map with high accuracy.

In practice, the accurate location and timestamp of a particular upcoming rider are hard to predict due to the uncertain personal and environmental factors. In this paper, we predict the number of riders for a given region (i.e., a spatial range of area, such as square regions or hexagon regions) in a given period (i.e., next 15 minutes). We apply the state-of-the-art prediction model, DeepST [48], on the real-world taxi demand-supply data set to generate our demand number map offline. Specifically, DeepST uses previous order numbers from three different time scales: *closeness*, *period*, *trend*. Here, *closeness* means the previous N time slots; *period* indicates the same time in previous N days; *trend* refers to the same time in previous N weeks. Meanwhile, it also uses other features (e.g., weather information) to make a good prediction by using Convolutional Neural Network [28]. Note that, the DeepST model is trained offline, which is time-consuming (i.e., using several hours to train a DeepST model on one-month taxi order data set of NYC). However, when conducting online prediction, it can generate the demand numbers of all the regions for a particular time period within 10 microseconds on a normal GPU server, which can be ignored in real applications.

4.2 Supply Number Map

We first introduce some preliminary concepts.

Definition 7 (Arriving Time)**.** Given a worker w_i and his/her route S_{w_i} relabeled here for clearness to $S_{w_i} = [l_1, l_2, \cdots, l_{|S_{w_i}|}]$ at time t_0, we denote arriving time arr for each l_k as:

$$arr_i[l_k] = \begin{cases} t_0, & \text{if } k = 0 \\ arr_i[l_{k-1}] + dis(l_{k-1}, l_k), & \text{if } k > 0 \end{cases}$$

which indicates the time when worker w_i arrives each l_k.

Definition 8 (Supply Contribution)**.** Given a start location u, an end location v and a timestamp t_o, we assume that at time t_o an imaginary vehicle w_o moves from u to v going the shortest path. Let $C(u, v, t_o)$ be the set of spatial temporal cells that w_o visits during the move. In addition, let $CT_{xy}(u, v, t_o)$ be the duration time of w_o staying in area N_x in time span T_y when it moves from u to v at time t_o. Then, the supply contribution of w_o for spatial temporal cell C_{xy} is denoted as $SC_{xy}(u, v, t_o) = \gamma \cdot CT_{xy}(u, v, t_o)$, where the parameter γ refers to the ratio of the equivalent number of workers to the total time all the workers cost in a region during a certain time span.

In addition, an imaginary vehicle would stay at the destination at last. This also contributes for this area and those time spans. The supply contribution of w_o for spatial temporal cell C_{xy} is denoted as $SC_{xy}(l_n, -1, arr_o[l_n])$ where l_n is the destination of S_{w_o}.

Weight parameter γ is a ratio of the equivalent number of workers to the duration time of workers appearing in a spatial-temporal cell. For example, if the total duration time of vehicles in a spatial-temporal cell C_{xy} equals to $k * \gamma$, we assume k requests can be served by the passing vehicles. Note that in the setting of shared mobility, one worker can serve more than one request simultaneously. In the supply number map, one imaginary worker refers to the ability to serve one request.

Next, we define the supply number map as follows.

Definition 9 (Supply Number Map)**.** Given a set of time spans $\mathcal{T}$ and a set workers W, the supply number map is the set of estimated supply contribution numbers for each spatial temporal cell C_{xy} denoted as:

$$SN(N_x, T_y) = \sum_{w_i \in W} \sum_{k=1}^{|S_{w_i}|-1} SC_{xy}(l_k, l_{k+1}, arr_i[l_k]) \qquad (2)$$

With a supply number map, we can estimate the equivalent number of workers in a certain period. According to the dataset of New York City Taxi [4], only 0.05% requests cost more than 1.5 hour among all requests during Dec. 2013. In this paper, we keep updating the supply number map with a total time span of the upcoming 1.5 hours. To guide route planning, we divide 1.5 hours into 6 intervals each with 15 minutes because: (a) Too long intervals such as 30 minutes cannot provide useful guidance with the short waiting time of requests, such that several workers arrive a place with one request in the same interval but 20 minutes earlier than them leading to a canceled request and failed matching. (b) If the intervals are too short, the accuracy of Demand Number Map is poor. Besides, the computation complexity will greatly increase for further operations in Section 5.

4.3 Demand-Supply Balance Score

Given a specific time span $T = \{T^+, T^-\}$, the prediction model provides us with the corresponding $DN = \{dn_1, dn_2, \cdots, dn_{|\mathcal{N}|}\}$ as the number of requests in each region. As a classical discrete statistic case, we assume the number of requests in each area follows the Poisson distribution. We set the unit time as $T^+ - T^-$, thus the characteristic value $\lambda = dn$, which means the number of requests in one unit time. Then, the distribution is expressed as:

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}, k = 0, 1, \cdots$$

Next, given a certain area with supply number sn, we can define a local balance score LB as the expected number of matched worker-and-request pairs Y, which can be derived as:

$$\begin{aligned} LB(\lambda, sn) = E(Y) &= \sum_{k=0}^{\lfloor sn \rfloor - 1} k \frac{\lambda^k}{k!} e^{-\lambda} + \sum_{k=\lfloor sn \rfloor}^{\infty} \lfloor sn \rfloor \frac{\lambda^k}{k!} e^{-\lambda} \qquad (3) \\ &= \sum_{k=0}^{\lfloor sn \rfloor - 1} (k - \lfloor sn \rfloor) \frac{\lambda^k}{k!} e^{-\lambda} + \sum_{k=0}^{\infty} \lfloor sn \rfloor \frac{\lambda^k}{k!} e^{-\lambda} \\ &= \sum_{k=0}^{\lfloor sn \rfloor - 1} (k - \lfloor sn \rfloor) \frac{\lambda^k}{k!} e^{-\lambda} + \lfloor sn \rfloor \end{aligned}$$

In Equation 3, the first term refers to the situation that the number of requests k is smaller than that of workers, thus the expected matching pairs is k; the second term indicates that the number of requests k is larger than that of workers, thus the matching pairs will be rounded down to $\lfloor sn \rfloor$ as the number of served requests must be integer. If one more sn appears, we can derive the expected increased matching number as $\Delta_{LB}(\lambda, sn) = LB(\lambda, sn + 1) - LB(\lambda, sn)$.

Then, we formally define our Demand-Supply Balance Score for route planning using an incremental function below.

Definition 10 (Demand-Supply Balance Score)**.** For a set of ordered time spans $\mathcal{T}$ and a set of areas $\mathcal{N}$, we maintain the demand number map DN and supply number map SN. Then the Deman-Supply Balance Score (DSBScore) for an insertion operation is:

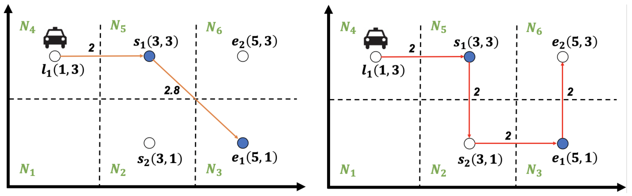
$$DSB = \sum_{x=1}^{|\mathcal{N}|} \sum_{y=1}^{|\mathcal{T}|} \beta_y \left({sn'}_{xy} - sn_{xy} \right) \cdot \Delta_{LB}(\lambda_{xy}, sn_{xy}) \qquad (4)$$

where $sn_{xy} = SN(N_x, T_y)$ as the original supply number of cell C_{xy} and sn' is the updated one after insertion, $\lambda_{xy} = DN(N_x, T_y)$, and β_x is the weight parameter for the xth time span. We assume that the earlier a time span is, the larger its β is, due to decay effect.

In summary, to derive DSBScore, we save all the prediction results of demand information in DN and mobility supply number based on all the scheduled routes in SN. DN will be updated according to the prediction model (e.g., every 15 minutes) and SN will be updated after every successful request insertion. The demand-supply score will be used to guide our assignment for each newly arrived request.

5. DEMAND-AWARE INSERTION

As proved in Section 3, there is no polynomial-time algorithm to achieve the optimal solution for DARP problem. Many existing studies show that *Insertion* is a practical approach to greedily deal with the shareable mobility problem. Comparing to the basic objective, our optimization goal considers a balance score such that



(a) Route and requests before insertion (b) One possible new route after insertion

Figure 2: Example For Insertion

the insertion based framework can achieve better results in a relatively long time period, however, the existing Insertion algorithms cannot be applied directly for our DARP problem. In this section, we first define the demand-aware insertion operation formally and introduce a basic insertion algorithm for our prediction-based insertion with $O(n^3)$ complexity. Then, we propose a dynamic-programming algorithm to reduce the time complexity to $O(n^2)$.

5.1 Insertion Operation

To extend Insertion [12, 23, 24, 25, 26, 30, 40] for the DARP problem, we follow the idea to search each route and locally optimally insert new vertex (or vertices) into a route. In our problem, there are two vertices (e.g. locations of source and destination) to be inserted for each request. We formally introduce the insertion operation [12, 23, 30, 40] as follows.

Definition 11 (Insertion)**.** Given a worker w_i with the current route S_{w_i} and a new request r_j, the insertion operation aims to find a new feasible route S'_{w_i} by inserting s_j and e_j into S_{w_i} with the minimum increased cost while maintaining the order of vertices in S_{w_i} unchanged in S'_{w_i}.

In DARP we need to maintain the supply number map. Any insertion will violate the balance of demand and supply since all of the arriving time of vertices behind the insertion place will be changed. We illustrate the challenge of updating the supply number map for DARP in the following example.

Example 2. *Let us consider an example of ridesharing in Figure 2. At timestamp 0, assume that a ride request r_1 is released with with s_1 and e_1 as its origin and destination, and its deadline of delivery is timestamp 11. Driver w_1 is assigned to serve r_1 with a scheduled route $S_{w_1} = [l_1, s_1, e_1]$. As shown in Figure 2(a), at timestamp 1, the current location of w_1 is $l_1 = (1,3)$; a new request r_2 is released with s_2 and e_2 as its origin and destination, and its deadline of delivery is timestamp 11. The travel cost is equal to the Euclidean distance between any pair of locations. Assume that the capacity of w_1 is enough to carry r_1 and r_2 at the same time. We set the length of time spans to be 3. Then, $\mathcal{T} = \{T_1 = \{3,6\}, \cdots T_6 = \{15,18\}\}$. We have six regions $N_1 \sim N_6$ with grid size of 2. The supply number map is shown in Table.1. The demand number map is shown in Table.2.*

Figure 2(b) shows a possible new route $S'_{w_1} = [l_1, s_1, s_2, e_1, e_2]$ after insert r_2 to S_{w_1}. It is feasible with arriving times $arr_1[e_1] = 1+2+2+2 = 7 \leq 11$ and similarly $arr_1[e_2] = 9 \leq 11$. In this example, $\alpha = 1$, $\beta = \left\{10, \frac{10}{e}, \cdots, , \frac{10}{e^5}\right\}$, and $\gamma = 0.1$.

After insertion, the supply will change for each spatial temporal cell C_{xy}. For example, before insertion of r_2, the total supply contribution of S_{w_1} to $C_{5,1}$ is $SC_{5,1}(l_1, s_1, 1) + SC_{5,1}(s_1, e_1, 3) + SC_{5,1}(s_1, -1, 5.8) = 0.14$. After the insertion, the total supply contribution of S'_{w_1} changes to $SC_{5,1}(l_1, s_1, 1) + SC_{5,1}(s_1, s_2, 3) + SC_{5,1}(s_2, e_1, 5) + SC_{5,1}(e_1, e_2, 7) + SC_{5,1}(e_2, -1, 9) = 0.1$. If we insert r_2, the value of $C_{5,1}$ in the supply number map will be updated to $0.5 - 0.14 + 0.1 = 0.46$.

Table 1: Supply Number Map

$\mathcal{N}$ $\mathcal{T}$	N_1	N_2	N_3	N_4	N_5	N_6
T_1	1.7	3.8	2.5	2.3	0.5	1.3
T_2	3.3	2.1	1.7	1.1	3.2	2.9
T_3	3.5	3.3	2.0	0.7	3.8	1.4
T_4	3.6	1.3	2.4	3.0	1.2	2.6
T_5	0.5	2.5	1.4	1.3	1.6	2.3
T_6	3.4	2.0	1.0	3.7	2.2	3.8

Table 2: Demand Number Map

$\mathcal{N}$ $\mathcal{T}$	N_1	N_2	N_3	N_4	N_5	N_6
T_1	2	3	5	2	4	3
T_2	3	2	4	2	3	2
T_3	3	3	4	2	2	2
T_4	3	1	4	3	2	2
T_5	4	2	4	3	1	3
T_6	3	2	3	4	2	2

5.2 Basic Insertion

We first introduce the general steps of the extended basic insertion algorithm [25, 26] to handle each new request in our DARP problem. Specifically, for a new request r_j, the basic insertion algorithm checks every possible position to insert the origin and destination locations and return the positions such that the incremental cost is minimized. Note that, as the Example 2 shows, different insertions may cause different updates on the supply number map. Thus, when we check every origin and destination insertion positions, we need to evaluate the effect of the updated schedule on the supply number map, and then calculate the consequent cost for each different insertion. We illustrate the basic insertion as follows:

Example 3 (Basic Insertion Example)**.** *Let us continue the setting in Example.2. We need to find the insertion positions in the currently scheduled route S_{w_1} with minimum increasing cost.*

If we insert the new request r_2 to $S_{w_1} = [l_1, s_1, e_1]$ to achieve the new scheduled route as $S'_{w_1} = [l_1, s_1, s_2, e_1, e_2]$ as Figure 2(b). The new cost S'_{w_1} can be calculated as $1 \cdot (2+2+2+2) - (10 \cdot 0.1 \cdot \Delta_{LB}(4, 0.5) + 10 \cdot 0.2 \cdot \Delta_{LB}(3, 3.8) + 10/e \cdot 0.2 \cdot \Delta_{LB}(4, 1.7) + 10/e \cdot 0.1 \cdot \Delta_{LB}(2, 2.9) + 10/e^2 \cdot 0.3 \cdot \Delta_{LB}(2, 1.4) + 10/e^3 \cdot 0.3 \cdot \Delta_{LB}(2, 2.6) + 10/e^4 \cdot 0.3 \cdot \Delta_{LB}(3, 2.3) + 10/e^5 \cdot 0.3 \cdot \Delta_{LB}(2, 3.8)) = 8 - (1.0+0.7+0.7+0.1+0.2+0+0+0) = 5.3$.

For every new route, we calculate their cost. Besides, we derive the cost of the original route to calculate increased costs. We list all the costs and the increased costs in Table 3. For simplification, we use notation (X, Y) to indicate the insertion positions of the origin and destination on the original route S_{w_1}. For example, $S'_{w_1} = [l_1, s_1, s_2, e_1, e_2]$ can be expressed as insertion position $(2,3)$. The cost of infeasible route (e.g. $(1,1)$) is set to ∞.

From Table 3, to achieve the lowest increased cost is to insert s_2 after position 2 (s_1) and e_2 after position 3 (e_1). Then, we return S'_{w_1} as the optimal local planning for inserting r_2 into S_{w_1}.

Table 3: Cost of New Route and Increased Cost

Insertion Position (x,y)	(1, 1)	(1, 2)	(1, 3)	(2, 2)	(2, 3)	(3, 3)
Cost	∞	6.2	6.8	6.3	5.3	6.1
Increased Cost	∞	5.6	6.2	5.7	4.7	5.5

Complexity Analysis. As the basic insertion algorithm needs to check every possible insertion position, which is $O(n^2)$. For every insertion position pair, we need to calculate the new cost of the new route based on the updated supply number map. However, each insertion location may update $O(n)$ spatial-temporal cells in the supply number map. Then, the overall time complexity of the basic insertion algorithm is $O(n^3)$.

5.3 Dynamic Programming Based Insertion

In this section, we propose a DP-based insertion algorithm, which reduces the time complexity of Basic Insertion from $O(n^3)$ to $O(n^2)$. The idea is to enumerate all possible pairs of inserting positions, but check whether a new route violates the constraints and calculate the increased cost Δ_{w_i} in $O(1)$ time rather than $O(n)$ time. We first introduce a way to check route feasibility because its definition is needed in the following section.

5.3.1 *Check Route Feasibility in $O(1)$ time*

Two conditions should be satisfied for a feasible route: *(i)* deadline constraint and *(ii)* capacity constraint in Definition 3.

For a route S_{w_i}, deadline of each location $ddl_i[l_k]$ can be derived as:

$$ddl_i[l_k] = \begin{cases} td_j - dis\left(s_j, e_j\right), & \text{if } l_k \text{ is } s_j \\ td_j, & \text{if } l_k \text{ is } e_j \end{cases}$$

We borrow the idea of [25] and denote that $slack_i[l_k]$ is the maximal time for w_i to delay (e.g. slack time) in order to arrive a destination l_k in time. $slack_i[l_k]$ can be derived as:

$$\begin{aligned} slack_i[l_k] &= \min_{k'>k}\left(ddl\left[k'\right] - arr_i\left[k'\right]\right) \\ &= \min\{slack_i[l_{k+1}], ddl_i[l_{k+1}] - \ arr_i[l_{k+1}]\} \end{aligned} \tag{5}$$

Note that the slack time of two location l_x and l_y always satisfies $slack_i[l_x] \leq slack_i[l_y]$ if $x < y$ [25].

Whenever a source or destination of r_j is inserted in a route S_{w_i} at l_w between l_x and l_y, it will cause a *detour* as $det(l_x, l_w, l_y) = dis(l_x, l_w) + dis(l_w, l_y) - dis(l_x, l_y)$.

In many previous studies, the objective function only consists of distance [16, 23, 40]. Thus, the calculation of increased cost Δ_{w_i} only occurs at inserted positions. We define the increased distance cost as:

$$\Delta^d_{x,y} = \begin{cases} dis\left(l_n, s_j\right) + dis\left(s_j, e_j\right), & \text{if } x = y = n \\ dis\left(l_x, s_j\right) + dis\left(s_j, e_j\right) & \\ \quad +dis\left(e_j, l_{x+1}\right) - dis\left(l_x, l_{x+1}\right), & \text{if } x = y < n \\ det\left(l_x, s_j, l_{x+1}\right) + dis\left(l_y, e_j\right), & \text{if } x < y = n \\ det\left(l_x, s_j, l_{x+1}\right) + det\left(l_y, e_j, l_{y+1}\right), & \text{otherwise} \end{cases}$$

However, in this paper, the demand-aware cost consists of the increased distance and the shift of balance score. We will discuss how to derive the shift of balance score in $O(1)$ in Section 5.3.2.

Lemma 5.1. *The deadline constraint will be satisfied if and only if* (a) $arr_i[l_x] + dis(l_x, s_j) \leq td_j - dis(s_j, e_j)$*;* (b) $det(l_x, s_j, l_{x+1}) \leq slack_i[l_{x+1}]$*;* (c) $arr_i[l_x] + dis(l_x, s_j) + dis(s_j, e_j) \leq td_j$ *when* $x = y$ *or* $arr_i[l_y] + det(l_x, s_j, l_{x+1}) + dis(l_y, e_j) \leq td_j$*when* $x < y$*; and* (d) $\Delta^d_{x,y} \leq slack_i[l_{y+1}]$.

Proof. With s_j of request r_j inserted at the xth position, condition *(a)* checks whether the deadline constraint of the new request r_j and condition *(b)* checks whether any deadline constraint of all the other requests is violated; with e_j of request r_j inserted at the yth position, condition *(c)* checks whether the deadline constraint of r_j is violated and condition *(d)* checks whether any deadline constraint of all the other requests is violated. □

Then we check the capacity constraint in $O(1)$ time. We define picked request $picked_i[l_k]$ which refers to the total capacity of the requests that are still on the route of w_i when the worker arrives at location l_k. Then, we have:

$$picked_i[l_k] = \begin{cases} picked_i[l_{k-1}] + a_j, & \text{if } l_k \text{ is } s_j \\ picked_i[l_{k-1}] - a_j, & \text{if } l_k \text{ is } e_j \end{cases} \tag{6}$$

In addition, if we insert the destination after position k, we define the smallest position to insert origin without violating the capacity constraint as $pso_i[k]$. Then, we have the lemma below to guarantee checking the capacity constraint in $O(1)$ complexity.

Lemma 5.2. *The capacity constraint of worker w will be satisfied if and only if $pso_i[y] \leq x$.*

Proof. Since s_j is inserted at the xth position, and e_j is inserted at the yth position, we need to guarantee whether there exists any $k \in \{x, y\}$ such that $picked_i[l_k]$ is greater than $a_i - a_j$. If there exists, at position k, $pso_i[x] \geq pso_i[k] = k > x$ and it will never change. Or else, $pso_i[y] = pso_i[x] \leq x$ will be satisfied. □

5.3.2 *Derive Increased Cost Δ_{w_i} in $O(1)$ time*

To realize a constant-time cost calculation in DARP, we propose a list to pre-derive all the needed cases such that any situation can look up the list to find the increased cost in constant time. Based on the granularity of time values (e.g. at least 30 seconds as the unit time for drop-off time td_j), we can dynamically derive all the situations according to finite cases of time delay after insertion.

Definition 12 (Supply-Shift)**.** Given two adjacent vertices l_k, l_{k+1} in a route sequence, the Supply-Shift (*SS*) is the change of supply contribution of sub-route (l_k, l_{k+1}) if delay Δ_t occurs before l_k.

SS is a double-key dictionary labeled by area N_x and time interval T_y. To derive it, we need to calculate *SC* for (l_k, l_{k+1}) with arriving time $arr_i[l_k]$ and a new *SC'* with arriving time $arr_i[l_k]+\Delta_t$. Then we have: $SS(l_k, l_{k+1}, arr_i[l_k], \Delta_t) = SC' - SC$. Here, *SC* and *SC'* are dictionaries, and the operator "$-$" refers to the difference of values of two dictionaries on each common key. If any key does not belong to one dictionary, the corresponding value of the dictionary is set as 0.

The slack time is limited and $\Delta_t \leq slack_i[l_k], \forall l_k \in S_{w_i}$. We denote the maximal time to delay as $slack^+$. Here, we divide the delayed time range $(0, slack^+]$ into a discretized space $[(0, \delta_t], (\delta_t, 2\delta_t], \cdots, ((m-1)\delta_t, m\delta_t]]$, where m is an integer constant and $\delta_t = \frac{slack^+}{m}$. Then, for a given Δ_t, if $(x-1)\delta_t < \Delta_t \leq x\delta_t$, we use the discretized delay $\Delta'_t = (x+1)\delta_t$ instead of Δ_t to calculate the corresponding supply shift.

Given a route S_{w_i}, we can calculate $SS(l_k, l_{k+1}, arr_i[l_k], \Delta'_t)$ for all pairs of adjacent vertices (l_k, l_{k+1}) by looking up the pre-derived supply-shift table quickly. This process costs $O(nm)$ time. We further introduce Total-Supply-Shift *TSS* as follows:

Definition 13 (Total-Supply-Shift)**.** Given a route S_{w_i}, the Total-Supply-Shift $TSS(k, \Delta_t)$ refers to the total shift of supply number after an insertion occurs at the kth position with detour Δ_t.

Assume that the length of the route $|S_{w_i}| = n$. With limited cases of time delay, we can derive $TSS[k, \Delta'_t]$ dynamically by starting with $TSS[n, \Delta'_t] = SC(l_n, -1, arr_i[l_n] + \Delta'_t) - SC(l_n, -1, arr_i[l_n])$ followed by: $TSS[k, \Delta'_t] = TSS[k+1, \Delta'_t] + SS(l_k, l_{k+1}, arr_i[l_k], \Delta'_t)$

Note that any $x < y$ satisfies $slack[x] \leq slack[y]$, which indicates that $TSS[k+1, \Delta'_t]$ always exits if Δ'_t delay is feasible for k^{th}. Now, given the insertion positions at x and y, we can derive all the additional supply shift generated from this insertion, named the Incremental Supply Shift (ISS) as follows:

$$\begin{aligned} &ISS_{x,y} \\ &= \begin{cases} SC(l_n, s_j, arr_i[l_n]) + SC(s_j, e_j, arr_i[s]) & \\ \quad + SC(e_j, -1, arr_i[e]) - SC(l_n, -1, arr_i[l_n]), & \text{if } x = y = n \\ SC(l_x, s_j, arr_i[l_x]) + SC(s_j, e_j, arr_i[s]) & \\ \quad + SC(e_j, l_{x+1}, arr_i[e]) & \\ \quad - SC(l_x, l_{x+1}, arr_i[l_x]) + TSS(x+1, \Delta^d_{x,y}), & \text{if } x = y < n \\ SC(l_x, s_j, arr_i[l_x]) + SC(s_j, l_{x+1}, arr_i[s]) & \\ \quad - SC(l_x, l_{x+1}, arr_i[l_x]) & \\ \quad + SC(l_n, e_j, arr_i[l_n] + det(l_x, s_j, l_{x+1})) & \\ \quad + SC(e_j, -1, arr_i[e]) - SC(l_n, -1, arr_i[l_n]) & \\ \quad + TSS(x+1, det(l_x, s_j, l_{x+1})) & \\ \quad - TSS(n, det(l_x, s_j, l_{x+1})), & \text{if } x < y = n \\ SC(l_x, s_j, arr_i[l_x]) + SC(s_j, l_{x+1}, arr_i[s]) & \\ \quad - SC(l_x, l_{x+1}, arr_i[l_x]) & \\ \quad + SC(l_y, e_j, arr_i[l_y] + det(l_x, s_j, l_{x+1})) & \\ \quad + SC(e_j, l_{y+1}, arr_i[e]) - SC(l_y, l_{y+1}, arr_i[l_y]) & \\ \quad + TSS(x+1, det(l_x, s_j, l_{x+1})) & \\ \quad - TSS(y, det(l_x, s_j, l_{x+1})) + TSS(y+1, \Delta^d_{x,y}), & \text{otherwise} \end{cases} \end{aligned} \tag{7}$$

where $arr_i[s]$ and $arr_i[e]$ denote the arriving times of s_j and e_j after insertion. All the detour times Δ_t in TSS are substituted with $\Delta'_t = k \cdot \delta_t$ where $(k-1) \cdot \delta_t < \Delta_t \leq k \cdot \delta_t$. There are four cases of calculating $ISS_{x,y}$: ***(i)*** If the new r_j is picked and served after finishing all the other requests, two new paths, $l_n \rightarrow s_j$ and $s_j \rightarrow e_j$, are added. Their supply contributions are $SC(l_n, s_j, arr_i[l_n])$ and $SC(s_j, e_j, arr_i[s])$, where

Algorithm 1: DA-DP Insertion.

```
   Input: a worker $w_i$ and its current route $S_{w_i}$, a request $r_j$,
          current time $t_{now}$, time intervals $\mathcal{T}$ and grid area $\mathcal{N}$
          with the demand and supply number map $DN$ and
          $SN$
   Output: an optimal route $S'_{w_i}$ after insertion with cost $\Delta_w$
1  $S'_{w_i} := S_{w_i}, \Delta_w := +\infty, ISS' := \emptyset$
2  get $arr_i[\cdot]$, $slack[\cdot]$, TSS according to definition 7 and
     Equations 5 and 13
3  foreach $x$ in $1$ to $|S_{w_i}|$ do
4      foreach $y$ in $x$ to $|S_{w_i}|$ do
5          $S^t_{w_i}$ := insert $l_s$ at x-th and $l_d$ at y-th in $S_{w_i}$
6          if $S^t_{w_i}$ is feasible then
7              get ISS according to Equation 7
8              $\Delta^t_w := \alpha\Delta^d_w$
9              foreach key$[N, T] \in ISS$ do
10                 $\Delta^t_w := \Delta^t_w - \beta_T \cdot ISS[N, T] \cdot$
                     $\Delta LB(DN(N, T), SN[N, T])$
11             if $\Delta^t_w < \Delta_w$ then
12                 $\Delta_w := \Delta^t_w$ $S'_{w_i} := S^t_{w_i}$ $ISS' := ISS$
13 return $S'_{w_i}$, $\Delta_{w_i}$, and $ISS'$
```

$arr_i[s] = arr_i[l_n] + dis(l_n, s_j)$. The original SC from staying at l_n after finishing the route is removed and the new one is added, that is, $SC(e_j, -1, arr_i[e]) - SC(l_n, -1, arr_i[l_n])$ where $arr_i[e] = arr_i[l_n] + \Delta^d_{x,y}$; ***(ii)*** If the new r_j is finished right after picked (e.g. $x = y$) but not the last one to serve, three parts of new contributions are added and the previous one from l_x to l_{x+1} is removed (i.e., $SC(l_x, s_j, arr_i[l_x]) + SC(s_j, e_j, arr_i[s]) + SC(e_j, l_{x+1}, arr_i[e]) - SC(l_x, l_{x+1}, arr_i[l_x])$, where $arr_i[s] = arr_i[l_x] + dis(l_x, s_j)$ and $arr_i[e] = arr_i[l_x] + dis(l_x, s_j) + dis(s_j, e_j)$). Every vertex after $(x+1)^{th}$ gets a detour $\Delta^d_{x,y}$ so supply number shift $TSS(x + 1, \Delta^d_{x,y})$ occurs; ***(iii)*** If the r_j is picked earlier but finished at last, two new paths, $l_x \rightarrow s_j$ and $s_j \rightarrow l_{x+1}$, from inserting s_j and one new path, $l_n \rightarrow e_j$ from finishing e_j are added. One old path from l_x to l_{x+1} is removed. Note that $arr_i[s] = arr_i[l_x] + dis(l_x, s_j)$ and $arr_i[e] = arr_i[l_n] + det(l_x, s_j, l_{x+1}) + dis(l_n, e_j)$. After deriving these supply contributions, all paths between $x{+}1$ to n are delayed by $det(l_x, s_j, l_{x+1},$ then we add $TSS(x+1, det(l_x, s_j, l_{x+1})) - TSS(n, det(l_x, s_j, l_{x+1}))$. The destination has been changed to e_j, thus we have $SC(e_j, -1, arr_i[e]) - SC(l_n, -1, arr_i[l_n])$ in addition; ***(iv)*** Otherwise, we add new contributions, $l_x \rightarrow s_j$, $s_j \rightarrow l_{x+1}$, $l_y \rightarrow e_j$ and $e_j \rightarrow l_{y+1}$, and distract former ones, $l_x \rightarrow l_{x+1}$ and $l_y \rightarrow l_{y+1}$ first. In this case, $arr_i[s] = arr_i[l_x] + dis(l_x, s_j)$ and $arr_i[e] = arr_i[l_y] + det(l_x, s_j, l_{x+1}) + dis(l_y, e_j)$. Then we add TSS from $x + 1$ to y with detour $det(l_x, s_j, l_{x+1})$ by $TSS(x + 1, det(l_x, s_j, l_{x+1})) - TSS(y, det(l_x, s_j, l_{x+1}))$. Finally, all paths after $y{+}1$ are delayed by $\Delta^d_{x,y}$ so we have $TSS(y + 1, \Delta^d_{x,y})$.

5.3.3 Demand-Aware Dynamic Programming based Insertion Algorithm

Algorithm 1 illustrate the Demand-Aware Dynamic Programming based Insertion (DA-DP Insertion) algorithm. Lines 1-2 initialize S'_{w_i} and Δ_{w_i} and derive needed parameters for Equation 7. Lines 3-5 check all possible insertion positions and generate new routes. Line 6 checks feasibility according to Section 5.3.1. Line 7 gets dictionary *ISS* and lines 8-12 adds up all the costs for insertion. Lines 11-12 save the result with the least increased cost.

Table 4: Values After Pre-calculation

Positions	$1(l_1)$	$2(s_1)$	$3(e_1)$
$arr_i(\cdot)$	1	3	5.8
$ddl_i(\cdot)$	∞	8.2	11
$slack_i(\cdot)$	∞	5.2	5.2

Table 5: Increased Distance Cost $\Delta^d_{x,y}$ in Example.4

Insertion Position (x,y)	(1, 1)	(1, 2)	(1, 3)	(2, 2)	(2, 3)	(3, 3)
$\Delta^d_{x,y}$	-1	4	4.8	4	3.2	4.8

Table 6: Values of Supply Shift SS

Δ'_t	$SS(l_2, l_3, 3, \Delta'_t)$
1	$\{(N_3, T_1) \rightarrow -0.08, (N_3, T_2) \rightarrow 0.08\}$
2	$\{ (N_5, T_1) \rightarrow -0.04, (N_5, T_2) \rightarrow 0.04,$ $(N_3, T_1) \rightarrow -0.14, (N_3, T_2) \rightarrow 0.14 \}$
3	$\{ (N_5, T_2) \rightarrow 0.14, (N_3, T_2) \rightarrow 0.14,$ $(N_5, T_1) \rightarrow -0.14, (N_3, T_1) \rightarrow -0.14 \}$
4	$\{ (N_5, T_2) \rightarrow 0.14, (N_3, T_2) \rightarrow 0.06, (N_3, T_3) \rightarrow 0.08,$ $(N_5, T_1) \rightarrow -0.14, (N_3, T_1) \rightarrow -0.14 \}$
5	$\{ (N_5, T_1) \rightarrow -0.14, (N_5, T_2) \rightarrow 0.10, (N_5, T_3) \rightarrow 0.04,$ $(N_3, T_3) \rightarrow 0.14, (N_3, T_1) \rightarrow -0.14 \}$
6	$\{ (N_5, T_3) \rightarrow 0.14, (N_3, T_3) \rightarrow 0.14,$ $(N_5, T_1) \rightarrow -0.14, (N_3, T_1) \rightarrow -0.14 \}$

Table 7: Values of Total Supply Shift TSS

k	Δ'_t	$TSS(k, \Delta'_t)$
3	1	$\{(N_3, T_1) \rightarrow -0.02, (N_3, T_2) \rightarrow -0.08\}$
	2	$\{(N_3, T_1) \rightarrow -0.02, (N_3, T_2) \rightarrow -0.18\}$
	3	$\{(N_3, T_1) \rightarrow -0.02, (N_3, T_2) \rightarrow -0.28\}$
	4	$\{(N_3, T_1) \rightarrow -0.02, (N_3, T_2) \rightarrow -0.30, (N_3, T_3) \rightarrow -0.08\}$
	5	$\{(N_3, T_1) \rightarrow -0.02, (N_3, T_2) \rightarrow -0.30, (N_3, T_3) \rightarrow -0.18\}$
	6	$\{(N_3, T_1) \rightarrow -0.02, (N_3, T_2) \rightarrow -0.30, (N_3, T_3) \rightarrow -0.28\}$
2	1	$\{(N_3, T_1) \rightarrow -0.1\}$
	2	$\{ (N_5, T_2) \rightarrow 0.04, (N_5, T_1) \rightarrow -0.04,$ $(N_3, T_1) \rightarrow -0.16, (N_3, T_2) \rightarrow -0.04 \}$
	3	$\{ (N_5, T_2) \rightarrow 0.14, (N_5, T_1) \rightarrow -0.14,$ $(N_3, T_1) \rightarrow -0.16, (N_3, T_2) \rightarrow -0.14 \}$
	4	$\{ (N_5, T_2) \rightarrow 0.14, (N_5, T_1) \rightarrow -0.14,$ $(N_3, T_1) \rightarrow -0.16, (N_3, T_2) \rightarrow -0.24 \}$
	5	$\{ (N_5, T_3) \rightarrow 0.04, (N_5, T_2) \rightarrow 0.1, (N_5, T_1) \rightarrow -0.14,$ $(N_3, T_1) \rightarrow -0.16, (N_3, T_2) \rightarrow -0.3, (N_3, T_3) \rightarrow -0.04 \}$
	6	$\{ (N_5, T_3) \rightarrow 0.14, (N_5, T_1) \rightarrow -0.14, (N_3, T_3) \rightarrow -0.14,$ $(N_3, T_2) \rightarrow -0.3, (N_3, T_1) \rightarrow -0.16 \}$

Example 4. *Let us continue the setting in Example.2. Our goal is to find the positions for insertion with minimum increasing demand-aware cost. The pre-calculated parameters are shown in Table 4 and incremental distances $\Delta^d_{x,y}$ are shown in Table 5. $\Delta^d_{x,y} = -1$ means the case violates the constraints. We choose $\delta_t = 1$ and show SS with position $1 < k \leq 3 - 1$ in Table.6. Note that $\lceil \frac{slack_1[l_2]}{1} \rceil = \lceil \frac{slack_1[l_3]}{1} \rceil = 6$, thus we have 6 possible ranges for Δ'_t. Then, TSS can be derived dynamically shown in Table.7.*

Then we go through all pairs of (x, y) for insertion. $(1, 1)$ violates the deadline constraints. $(1, 2)$ follows the case 4 ($1 < 2 < 3$). Thus, its $ISS_{1,2} = SC(l_1, s_2, 1) + SC(s_2, s_1, 3.8) - SC(l_1, s_2, 1) + SC(s_1, e_2, 5.8) + SC(e_2, e_1, 7.8) - SC(s_1, e_1, 3) + TSS(2, 3) - TSS(2, 3) + TSS(3, 4)$; $(1, 3)$ belongs to the case 3 ($1 < 3 = 3$). Its $ISS_{1,3} = SC(l_1, s_2, 1) + SC(s_2, s_1, 3.8) - SC(l_1, s_1, 1) + SC(e_1, e_2, 8.6) + SC(e_2, -1, 10.6) - SC(e_2, -1, 5.8) + TSS(2, 3) - TSS(3, 3)$;$(2, 2)$ belongs to the case 2 ($2 = 2 < 3$) with $ISS_{2,2} = SC(s_1, s_2, 3) + SC(s_2, e_2, 5) + SC(e_2, e_1, 7.8) - SC(s_1, e_1, 3) + TSS(3, 4)$;$(2, 3)$ follows the case 3 as well with a result $ISS_{2,3} = SC(s_1, s_2, 3) + SC(s_2, e_1, 5) - SC(s_1, e_1, 3) + SC(e_1, e_2, 7) + SC(e_2, -1, 9) - SC(e_1, -1, 5.8) + TSS(3, 2) - TSS(3, 2)$; $(3, 3)$ belongs to the case 1 ($3 = 3 = 3$) and results in $ISS_{3,3} = SC(e_1, s_2, 5.8) + SC(s_2, e_2, 7.8) + SC(e_2, -1, 10.6) - SC(e_1, -1, 5.8)$.

Finally, we derive all these ISSs and calculate the increased cost for each pair of insertion positions. The values of them are shown in Table.8. As the increased cost, 4.7, of insertion at $(2, 3)$ is the lowest, we return it as the optimal way to insert r_2 into S_{w_i}.

Table 8: Insertion Cost in Example.4

Insertion Index (x,y)	(1, 1)	(1, 2)	(1, 3)	(2, 2)	(2, 3)	(3, 3)
$\Delta^d_{x,y}$	∞	5.6	6.3	5.7	4.7	5.5

Note that, comparing to the basic insertion in Example 3, the increased costs of pair $(1, 2)$ *(6.3) is different (6.2 in previous one). This is because that we use* $TSS(2, 3)$ *and* $TSS(3, 3)$ *to approximate* $TSS(2, 2.8)$ *and* $TSS(3, 2.8)$.

Complexity Analysis. It cost $O(n)$ time to compute $arr_i[\cdot]$, $slack_i[\cdot]$ and $O(nm)$ time to calculate *TSS* at line 2. The number of possible insertion pairs is $O(n^2)$ in lines 6-10. It takes $O(1)$ time to check feasibility and $O(1)$ time to evaluate the increased cost. The other lines cost $O(1)$ time. Thus, the total time cost of Algorithm 1 is $O(n^2)$.

6. DEMAND-AWARE INSERTION BASED DUAL-PHASE FRAMEWORK

In this section, we introduce an efficient and effective framework, namely demand-aware insertion based dual-phase framework (DAIF) for the DARP problem. DAIF consists of two phases. The first one is the decision phase, which decides whether to serve a new request r_j or not. The second is the planning phase, which inserts r_j into the route of the selected worker.

6.1 Decision Phase

To minimize our objective function, we can reject a request when its penalty is smaller than the increased cost to serve it. A previous study proposes a lower bound of the minimum increased distance as the metric to make a fast decision about request rejection [40]. However, this is applicable only for the distance-based objective function. In this paper, we need to derive the new lower bound also considering the cost from the balance score. The decision phase can be completed in $O(n)$ time, where n is the number of locations in the scheduled route of the given worker.

6.1.1 Lower Bound of Minimum Increased Cost Δ

We use Δ to indicate the minimum cost to handle a new request r_j among all the workers. We denote the lower bound of Δ as Δ_-. Δ_- consists of the lower bound of increased distance cost Δ^d_- and increased balance cost Δ^b_-. For the lower bound of the increased distance cost Δ^d_-, we borrow the idea of using dynamic programming to derive Δ^d_- in $O(n)$ time [40]. For the lower bound of the increased balance cost Δ^b_-, we propose a novel method to calculate it also in $O(n)$ time.

Lower Bound of Distance Cost Δ^d_-. We briefly introduce the method to derive Δ^d_- [40]. The key point is that once two vertices are inserted nonadjacent, the two increased distances will not affect each other. The main idea is to enumerate the destinations *(y)* instead of both origin and destination locations *(x, y)* to find the minimum increased distance. We denote Δ^d_y as the minimum increased cost for a given destination position y. $\Delta^d_y = det(l_y, e_j, l_{y+1}) + \min_{x<y} det(l_x, s_j, l_{x+1})$. We define the second part as $Dio[y] = \min_{x<y} det(l_x, s_j, l_{x+1})$ to indicate the minimum detour for inserting s_j with e_j at the yth position of a given route sequence. It can be derived in a dynamic programming style:

$$Dio[y] = \begin{cases} \infty, \text{ if } picked[y-1] > a_i - a_j \\ Dio[y-1], \text{ if } det\left(l_{y-1}, s_j, l_y\right) > slack_i[l_{y-1}] \\ \min\left\{Dio[y-1], det\left(l_{y-1}, s_j, l_y\right)\right\}, \text{ otherwise} \end{cases} \tag{8}$$

Two methods can be further applied to achieve a lower bound Δ_-: *(i)* using Euclidean distance instead of the shortest path query, which usually has smaller value and less computation time; *(ii)* using the pre-calculated $arr_i[\cdot]$ to derive travel time.

By denoting the Euclidean distance of u and v as $eu(u, v)$, we can estimate the lower bound of detour from inserting v between u and w as $ld(u, v, w) = eu(u, v) + eu(v, w) - dis(u, w) \leq det(u, v, w)$.

In addition, after calculating $arr_i[\cdot]$, all the $dis[l_k, l_{k+1}]$ can be substituted by $arr_i[l_{k+1}] - arr_i[l_k]$. Hence:

$$ld\left(l_x, s_j, l_{x+1}\right) = eu\left(l_x, s_j\right) + eu\left(s_j, l_{x+1}\right) - \left(arr_i[l_{x+1}] - arr_i[l_I]\right)$$

$$ld\left(l_y, e_j, l_{y+1}\right) = eu\left(l_y, e_j\right) + eu\left(e_j, l_{y+1}\right) - \left(arr_i[l_{y+1}] - arr_i[l_J]\right)$$

Then we use above lower bound for Equation 8 to calculate Dio_E as Euclidean distance based lower bound of $Dio[]$.

$$Dio_E[y] = \begin{cases} \infty, \text{ if } picked[y-1] > a_i - a_j \\ Dio_E[y-1], \text{ if } ld\left(l_{y-1}, s_j, l_y\right) > slack_i[l_{y-1}] \\ \min\left\{Dio_E[y-1], ld\left(l_{y-1}, s_j, l_y\right)\right\}, \text{ otherwise} \end{cases} \tag{9}$$

Now we have the lower bound of minimum increased cost Δ^d_-:

$$\Delta^d_- = \min_{x \leq y, y=0:n} \begin{cases} eu\left(l_n, s_j\right) + \mathcal{L}, & \text{if } x = y = n \\ eu\left(l_x, s_j\right) + \mathcal{L} + eu\left(e_j, l_{x+1}\right) & \\ \qquad -\left(arr_i[l_{x+1}] + arr_i[l_I]\right), & \text{if } x = y < n \\ ld\left(l_y, e_j, l_{y+1}\right) + Dio_E[y] & \text{if } x < y \end{cases} \tag{10}$$

where $\mathcal{L} = dis(s_j, e_j)$.

Lower Bound of Balance Cost Δ^b_-. We first propose a lemma:

Lemma 6.1. *Given an supply number map* SN *and an demand number map* DN*, let us consider a particular spatial temporal cell* C_{xy} *of area* N_x *and time span* T_y*. If the demand exceeds the supply (i.e.,* $DN(N_x, T_y) > SN(N_x, T_y)$*), the greater* $DN[N_x, T_y]$ *is (or the lesser* $SN[N_x, T_y]$ *is), the more sensitive* DSB *is to the change of* $SN(N_x, T_y)$ *(i.e.,* DSB *will increase or decrease faster when* $SN(N_x, T_y)$ *increases or decreases).*

Proof. Here we denote $DN[N_x, T_y]$ as λ and $SN[N_x, T_y]$ as μ. Suppose that serving a new request with a worker updates the value of cell C_{xy} in the *SN* to μ^*. We demote $\Delta_\mu = \mu^* - \mu$ as the increment. The *DSB* can be treated as a function of $\Delta_m u$ according to its definition. We want to show that with a larger λ and a smaller μ, $\frac{\partial DSB}{\partial \Delta_\mu}$ is laerger. Recall the definition of *DSB*, for N_x and T_y we have:

$$\begin{aligned} \frac{\partial DSB}{\partial \Delta_\mu} =& \beta_y \cdot \Delta_{LB}\left(\lambda, \mu\right) \\ =& \beta_y \cdot \left(LB(\lambda, \mu+1) - LB(\lambda, \mu)\right) \\ =& \beta_y \cdot \left(1 - \sum_{k=0}^{\lfloor sn \rfloor} \frac{\lambda^k}{k!} e^{-\lambda}\right) \end{aligned}$$

The second part is going larger with higher μ. For a single component in the sum function $f(\lambda) = \frac{\lambda^k}{k!} e^{-\lambda}$, we take a derivative with respect to λ:

$$\frac{\partial f(\lambda)}{\partial \lambda} = \frac{\lambda^{k-1}}{(k-1)!} e^{-\lambda} - \frac{\lambda^k}{k!} e^{-\lambda} = \frac{\lambda^{k-1} \cdot (k - \lambda)}{k!} e^{-\lambda}$$

With the situation that demand exceeds supply, we can get $k \leq \lfloor \mu \rfloor < \lambda$ such that $\frac{\partial f(\lambda)}{\partial \lambda} < 0$. Thus, whenever we want a higher *DSB* with an increased μ, we need the second part as small as possible, that is, higher λ and lower μ. □

With Lemma 6.1, if the greatest/lowest value in $DN[\cdot, T]$ are λ_+/λ_- and the greatest/lowest value in $SN[\cdot, T]$ is μ_+/μ_- given certain time interval T, a reorder with dt long new paths in place of old paths may have at most:

$$DSB[T]_+ = \beta_T \cdot \gamma \cdot dt \cdot \left(\Delta_{LB}(\lambda_+, \mu_-) - \Delta_{LB}(\lambda_-, \mu_+)\right) \tag{11}$$

Recall the definition of the largest tolerable detour $slack_i[\cdot]$ for each vertex in a route. We check which time interval each vertex belongs to by its $arr_i[\cdot]$. For a set of vertices finished in same interval, the longest replaced paths can be $slack_i[l_n]$ where n is the last arrived vertices as proof in 5.3.1. Hence, the lower bound of balance cost Δ^b_- can be derived as: $\Delta^b_- = -\sum_{T \in \mathcal{T}} DSB[T]_+$

6.1.2 Demand-Aware Decision Algorithm

Algorithm 2 illustrates the decision phase. We enumerate all the workers to find their lower bounds Δ_-. Lines 7-18 find the lower bound by traversing all paths. In lines 5-11 and 14-17 we track the time span to update Δ^b_- according to Equation 12. lines 12 and 18 update Dio_E to derive Δ^b_-. We store the value of bounded cost for each worker which will be further used in the planning process.

Algorithm 2: Demand-Aware Decision Algorithm.

```
Input: $\alpha$, $\beta$, workers $W$,a request $r_j$, supply number map
       $SN$ and demand number map $DN$
Output: a set of lower bound $\tilde{\Delta_-}$ for each $w$
1  $\tilde{\Delta_-} := \emptyset$
2  foreach $w \in W$ do
3      $\Delta_-^b := 0, \Delta_-^d := \infty, Dio_E := \infty$
4      Initialize $ddl, arr, slack$
5      Denote $T$ as the interval contains $arr_i[l_1]$
6      foreach $y$ in $0$ to $|S_w|$ do
7          Denote $T'$ as the interval contains $arr_i[l_y]$
8          if $T' \neq T$ then
9              $dt := slack_i[l_{y-1}]$
10             Update $DSB[T]_+$ according to Equation 11
11             $\Delta_-^b := \Delta_-^b - DSB[T]_+$
12         Get $\Delta_y^d$ as the case $x = y$ in Equation 10
13         if $y > 0$ then
14             $\Delta_y^d := \min(ld(l_y, e_j, l_{y+1}) + Dio_E[y], \Delta_y^d)$
15             $\Delta_-^d := \min \Delta_y^d, \Delta_-$
16             if $arr_i[l_y] + dis(s_j, e_j) > td_j$ then
17                 Derive $DSB[T]_+$ with rest of $T$ using $dt$
                     $\Delta_-^b := \Delta_-^b - \sum_{T \in \text{rest } \mathcal{T}} DSB[T]_+$ break;
18             Update $Dio_E[y+1]$ according to Equation 9
19     $\tilde{\Delta_-}[w] := \alpha\Delta_-^d + \beta\Delta_-^b$
20 if $p_j < \min \tilde{\Delta_-}$ then
21     reject $r_j$
22 return $\tilde{\Delta_-}$
```

Complexity Analysis. Lines 4 and 6 take $O(|R|)$ time. Line 20 takes $O(|W|)$ time. The other lines are in the loop of line 2 which all take $O(1)$ and thus $O(|W|)$ in total. The only shortest path query is $\mathcal{L} = dis(s_j, e_j)$. If the shortest path query takes $O(q)$ time, the time complexity is $O(|W| + |R| + q)$.

6.2 Planning Phase

Given the set of lower bounds $\tilde{\Delta_-}$ for all workers, we apply an efficient branch-and-bound pruning strategy to save the computation time. It first prunes candidate workers and then greedily adds the new request into the route of the local optimal worker.

6.2.1 Pruning Candidate Workers

Many pruning strategies rely on the bounded deadlines and grid indices to realize a filter for candidate workers [9, 12, 23, 38]. Besides, the branch-and-bound pruning strategy is also applied in previous work using lower bounds (*LB*) of cost for all the workers [40, 15]. In this paper, we derive our *LB* in the decision phase. This strategy can greatly cut the time cost because given all the workers sorted according to Δ_- in $\tilde{\Delta_-}$, if w_x is ahead of w_y and Δ_{w_x} is smaller than Δ_- of w_y, we can safely ignore all workers after w_x.

6.2.2 Algorithm Sketch

Algorithm 3 illustrates the DAIF algorithm. In lines 1-2, we build a grid index and initialize $\bar{R}$ and *DAC*. For each new request, we first filter a set of candidate workers in line 4 and then start the decision phase in line 5. If at least one worker has the possibility to serve the request, we can insert it into a route in lines 7-17. Iterations in lines 8-10 are using the aforementioned pruning strategy. Then we use DA-DP insertion to calculate Δ_{w_i} and update the best worker w' if current cost is minimal. If a feasible worker w' is found at the end of an iteration, we update S_{w_i} and SN map accordingly. The corresponding cost is added to DAC. If it is rejected, the cost from penalty p_j will be added.

Algorithm 3: DAIF Framework.

```
Input: $\alpha$, $\beta$, workers $W$, requests $R$, supply number map
       $SN$ and demand number map $DN$
Output: a set of route $S$ and demand-aware cost $DAC$
1  Build grid index and initialize $\bar{R} := \emptyset$
2  $DAC := 0$
3  foreach new request $r_j \in R$ do
4      $Cand :=$ filter the candidate using grid index and
        deadline
5      Get $\tilde{\Delta_-}$ according to Algorithm 2
6      if $r_j$ is not rejected then
7          $w' := NIL, \Delta_{w'} := \infty, SSA' := \emptyset$
8          foreach $w_i : \Delta_-$ in sorted $\tilde{\Delta_-}$ do
9              if $\Delta_{w'} < \Delta_-$ then
10                 break
11             $\Delta_{w_i} :=$ result of DA-DP Insertion
12             if $\Delta_{w_i} < \Delta_{w'}$ then
13                 $w' := w_i, \Delta_{w'} := \Delta_{w_i}, SSA' := SSA$
14         if $w' \neq NIL$ then
15             Update $S_{w'}$ and $arr$ of $w_i$ accordingly
16             Update $SN$ according to $SSA$
17             $DAC := DAC + \Delta_{w'}$
18     else
19         Add $r_j$ to $\bar{R}$, $DAC := DAC + p_j$
20 return $S$ and $DAC$
```

Complexity Analysis. Line 3 has $O(|R|)$ iterations. Line 4 takes $O(|W|)$ and line 5 takes $O(|W| + |R|)$ time proved after Algorithm 2. The total time complexity of lines 4-5 is $O(|R|\,|W| + |R|^2)$. The sorting in line 8 takes $O(|W| \log |W|)$ time and lines 9-15 takes $O(|W| + |R|^2)$ in total with a square time DP-based insertion algorithm (e.g. workers with no served requests takes $O(|W|)$ time and one worker with all requests takes $O(|R|^2)$ time). The total time complexity of lines 8-17 is $O(|R|^3 + |R|\,|W| \log |W|)$. Line 19 takes $O(|R| + |W|)$. Thus, Algorithm 3 takes $O(|R|^3 + |R|\,|W| \log |W|)$ time.

7. EXPERIMENTAL STUDY

7.1 Experimental Methodology

Data set. We use both real and synthetic data to test our proposed DARP approaches. Specifically, for the real data, we use a public data set NYC [4] collected from two types of taxis (yellow and green) in 2013 in New York City, USA. We choose the data from November 1st to 30th to train the prediction model. Then, we utilize the model to predict demands on December 2nd and use the real requests to simulate the ridesharing requests in our experiments. For each taxi request in NYC, we initialize a corresponding ridesharing request with its pick-up and drop-off locations, and configure the release time as the timestamp of the taxi request. We assume that each request contains only one rider and thus set the default capacity of a request as 1. We utilize the road network of NYC from Geofabrik [2]. This data set has been widely used as a benchmark for ride-sharing studies [40].

To generate the synthetic data, we derive the distribution of requests of all the NYC requests in December and generate a synthetic dataset (SYN) following the existing synthetic method [30].

For the prediction model, DeepST, we set the default grid size as 2km×2km and the time interval as 15 minutes.

Table 9: Setting of Dataset and Model

Parameters	Settings
Number of vertices in NYC	61298
Number of edges in NYC	141372
Number of requests of NYC	427093
Number of requests of SYN	452116
Grid size	2km×2km
Time interval	15 minutes

We summarize the experimental settings for the dataset and the spatial temporal prediction model in Table 9.

Implementation. In general, we follow the settings of the existing studies [9, 23, 40]. While building the graph for the road network, we set the weights of edges as their time costs (divide the distance of Geofabric by the velocity of its road type). The vertices are indexed with an R-tree. Then, we map the origin and the destination in NYC of each request to the closest vertex in the road network. The initial location of each worker is randomly chosen from the vertexes of the road network . We summarize the major parameters in Table 10 and the default values are in bold font. We set grid size as 2km for both prediction and prune algorithm. The delivery deadline of each request is calculated by adding the release time of a request with the time cost of the shortest path times Deadline Coefficient e_r. For example, the deadline e_j for request r_j with release time tr_j is $tr_j + (1+e_r) \cdot dis(s_j, e_j)$. The capacity of workers a_i is varied from 2 to 10. We use unit time cost as the unit cost for *DSB* (i.e., $\alpha = 1$). As we mentioned before, our unified cost can be treated as the monetary loss of the platform. Thus, we need to define the unit of money. We set travel fees paid for one second of time cost as the smallest element, that is, $\alpha = 1$. The other parameters are converted based on it. The penalty p_j of request r_j is set as $p_o \cdot dis(s_j, e_j)$ by default. In real applications, the penalty can be treated mainly as the money lost from a request on rejection, which is usually proportional to the length of the trip. The penalty p_j is usually much larger than the incremental cost from the allowed detour, thus request r_j will always be served if it can be delivered before the deadline td_j. Different p_j will not affect the served rate, thus we do not need to compare it. In our experiments, the balance weight β and supply coefficient γ are derived based on their values in real applications.

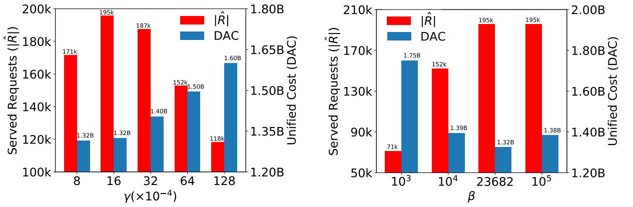
The experiments are conducted on a server with Intel(R) Xeon(R) E5 2.30GHz processors with hyper-threading enabled and 128GB memory. The simulation implementation is single-threaded, and the total running time (excluding the time to construct grid index and initialize LRU for shortest path and distance query) is limited to 14 hours for NYC. In reality, a real-time solution should stop before its time limit which is 24 hours here [23, 40]. All the algorithms are implemented in Java 11. We store the vertices and weighted edges of the road network (i.e., directed graph and distances used for grid). According to the setting of previous studies [23, 40], an LRU cache is maintained for shortest path queries.

Parameter Derivation. Equivalent supply number can be derived as Supply Coefficient γ multiplied by the sum of time (seconds) that all the workers stay in a particular area and time period. We derive $\gamma = 0.0016$ using the real ride-sharing data, which has 6576.87 workers at the rush hour to serve 465703.97 requests on average. A worker can serve a request if he/she stays in a region for $\frac{6576.87 \cdot 86400}{465703.97} = 1220$ seconds. In other times except for the rush hours, the platform just needs fewer cars. Besides, each ride-sharing vehicle can serve more than one request. Thus, we use a compensation factor $f = 2$ to derive that $\gamma = \frac{f}{1220} = 0.0016$.

When *DSBScore* decreases by 1, we would have a potential unserved request in the future. The average travel time for each trip is 789.40 seconds. Thus, potentially losing a request would cost $789.40 \cdot p_r = 23682$. We define this value as the optimal equivalent lose factor $p_r^* = 23682$, which is used to derive balance weight β in Table.10. The later a period is, the more imprecise its prediction result is. Thus, the weights decrease at the ratio of natural logarithm e per time span.

Table 10: Parameter Settings.

Parameters	Settings
Deadline Coefficient e_r	0.1, 0.2, **0.3**, 0.4, 0.5
Capacity a_i	2, **3**, 4, 7, 10
Distance Weight α	**1**
Balance Weight β	$\left[\boldsymbol{p_r^*, \frac{p_r^*}{e}, \frac{p_r^*}{e^2}, \cdots, \frac{p_r^*}{e^5}}\right]$
Supply Coefficient γ	**0.0016**
Penalty ratio p_o	**30**
Number of workers $\vert W\vert$	500, 1k, **3k**, 5k, 10k
Grid size g	1k×1k, **2k**×**2k**, 4k×4k



(a) Varying γ (b) Varying β

Figure 3: Performance of varying γ and β

We show the experimental results of varying γ and β in Figure 3. The left axis is the number of total served requests represented in the red histogram. The right axis is for the unified cost, displayed in blue. Note that the x-axis for β is the weight for its first time span. For example, value 1000 means that $\beta = \left[1000, \frac{1000}{e}, \frac{1000}{e^2}, \cdots, \frac{1000}{e^5}\right]$. With higher γ and β, the algorithm will focus more on serving more requests rather than reducing travel costs. We want to set the values of γ and β to serve as many requests with as small unified cost as possible. Through observing Figure 3(a), we confirm our setting $\gamma = 0.0016$. By Figure 3(b), we set $\beta = 23682$.

Compared Algorithms. We compare our DAIF framework (DAIF-B indicates using basic insertion algorithm and DAIF-DP denote using dynamic programming based insertion algorithm) with the state-of-the-art algorithms for route planning of shareable mobility services.

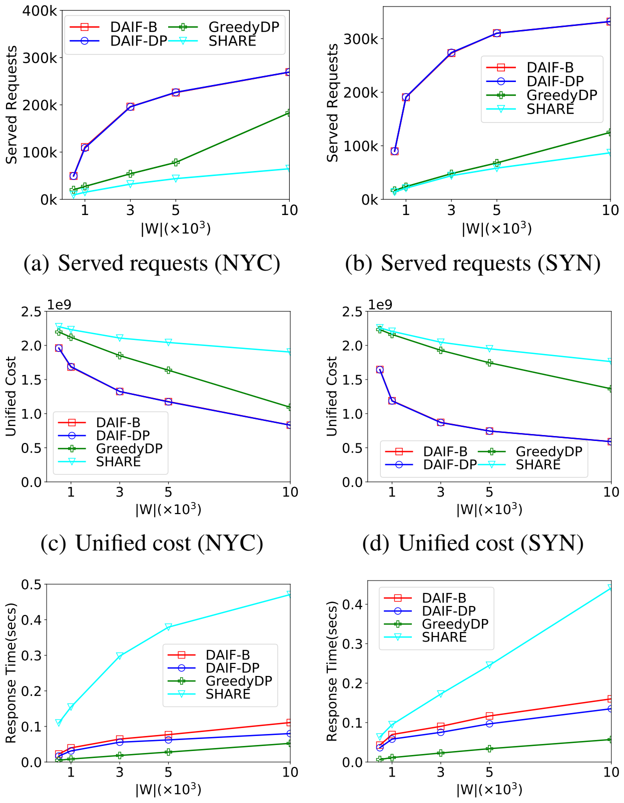
- **GreedyDP** [40]. It bases on the dynamic property of the ride-sharing problem to insert each coming request into a route of a worker with minimal increased distance.
- **SHARE** [45]. It uses historical information of nodes to choose a route with a higher possibility to pick passengers along the route. The algorithm is executed every 1 minute instead of a real-time assignment. It also requires k_1 minutes to wait for newly arrived requests and allows k_2 minutes of holding time before rejecting a request. To conduct a fair comparison, we set $k_1 = 0$ and $k_2 = 1$ to simulate an approximate real-time assignment.

Metrics. All the algorithms are evaluated in terms of total unified cost, served requests $\left|\hat{R}\right|$ and response time (average waiting time to arrange a request). The metrics are widely used in the existing large-scale online ride-sharing studies [23, 30, 40].

7.2 Experimental Results

In this section, we show the experimental results. We propose a prune strategy in Section 6 and implement it in all the following experiments. Compared with the DAIF framework without pruning, our solution prunes 37.8% (136151416 out of 360328505) candidates on default setting and prunes 51.5% (24123477 out of 46825174) candidates on non-prediction setting ($\beta = 0$).

Impact of Number of Workers $|W|$. Figure 4 presents the results with different numbers of workers. Overall, DAIF outperforms the other algorithms in terms of the number of served requests by 165.5% to 815.5% on SYN and 47.0% to 624.3% on NYC. With

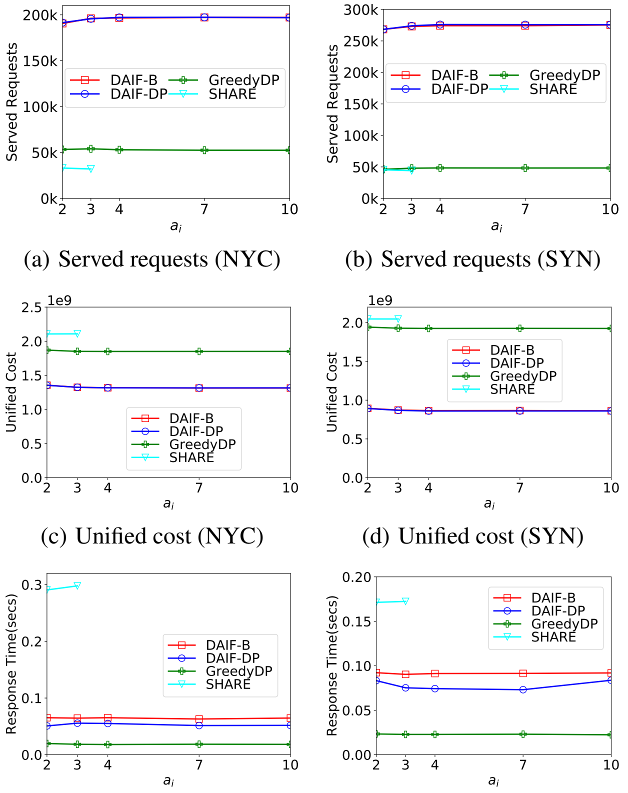


(a) Served requests (NYC) (b) Served requests (SYN)

(c) Unified cost (NYC) (d) Unified cost (SYN)

(e) Response time (NYC) (f) Response time (SYN)

Figure 4: Performance of varying workers $|W|$



(a) Served requests (NYC) (b) Served requests (SYN)

(c) Unified cost (NYC) (d) Unified cost (SYN)

(e) Response time (NYC) (f) Response time (SYN)

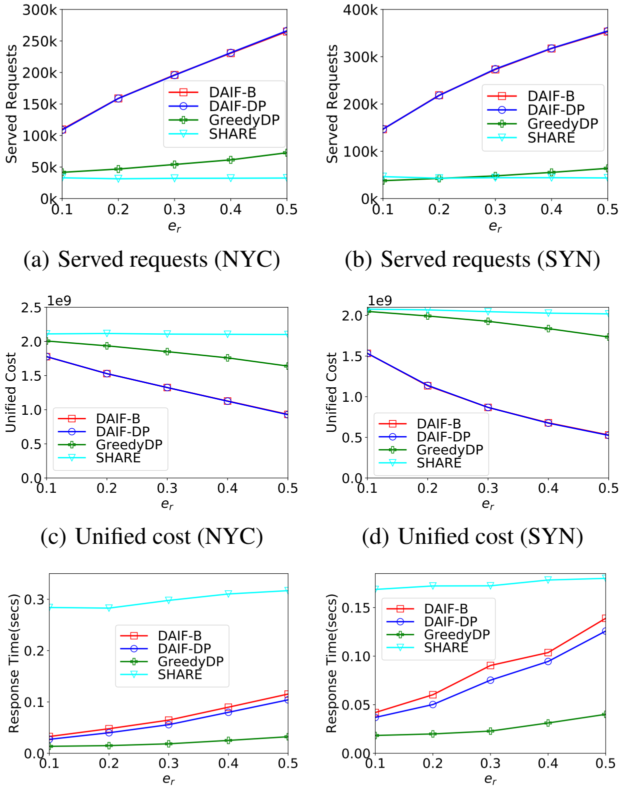
Figure 5: Performance of varying capacity a_i

more available workers, more requests are served, leading to a decrease in unified costs and an increase in the served rates of all the algorithms. Using a time slot ($\delta_t = 1$ minute in our experiments) for DAIF-DP, there is some little difference to the results of DAIF-B. The performances of DAIF-DP and DAIF-B are similar for unified cost and number of served requests. As we mentioned, we focus on the situation of rush hour, which has more requests and fewer riders. Most of the requests are not served, leading to a large value of unified cost and DAIF decreases it by 10.8% to 56.3% on NYC and 26.3% to 66.5% on SYN. GreedyDP runs the fastest. DAIF-B runs slower with an increasing number of workers than DAIF-DP. SHARE rejects many more requests compared with other methods. The reason is that SHARE uses a rough computation to first assign each request to a worker, which would be canceled if it violates the detour constraints. If this failed, the request would not be assigned in this round. SHARE works well when the waiting time of request is 15 min [45]. However, in online ridesharing, wait too long will break the deadline constraint of delivery. Thus, SHARE can serve fewer requests than other algorithms. As the synthetic data follows the distribution strictly, except the traditional method GreedyDP, all the other methods with this perfect "prediction" perform better than on NYC data set.

Impact of Capacity of Workers a_i. Figure 5 presents the effect of different capacities of workers. DAIF serves 259.3% to 509.7% more requests than the other algorithms on NYC and 470.3% to 521.1% more requests on SYN. Unified cost is decreased by 27.6% to 37.2% on NYC and 54.1% to 57.6% on SYN. At rush hour (workers are much fewer than requests), with a tight deadline coefficient, most of the workers can serve two or three requests. We can observe that increasing capacity does not lead to an increase but kind of fluctuation. However, when the capacity increases from 2 to 4, the number of served requests increases for DAIF and keeps still for GreedyDP. The reason is that routes are more suitable to serve more requests simultaneously for DAIF but GreedyDP only focuses on the current cost. For SHARE, capacity is limited to 3 because its algorithm only supports the cases with capacity 1, 2 and 3. When the capacity becomes larger than 3, most of the extra spaces are wasted and the number of served requests does not increase. Unified cost decreases when more requests are served. As for response time, GreedyDP still runs faster. DAIF-DP costs less time than DAIF-B, and the cost of DAIF-DP increases slower with larger capacity than that of DAIF-B. Prediction based methods (DAIF and SHARE) perform better on SYN than on NYC. For example, SHARE can only serves less than 50% requests compared with GreedyDP, but they perform similarly on SYN.

Impact of Deadline Coefficient e_r. Figure 6 shows the results of varying the deadline coefficient e_r. With a larger e_r, all the algorithms serve more requests with a lower unified cost. DAIF-B and DAIF-DP still perform similarly on served requests and unified cost. They serve 161.7% to 718.1% more requests than the others on NYC and 216.5% to 712.8% more on SYN. Compared with GreedyDP and SHARE, the unified cost of DAIF-B and DAIF-DP is decreased by 11.4% to 55.9% on NYC and 25.2% to 74.0% on SYN. With a larger deadline coefficient e_r, it is easier for requesters to share vehicles with other requesters. The serving rate of DAIF thus increases faster than that of GreedyDP with a powerful assignment strategy to achieve higher flexibility to carpool. The serving rate of SHARE increases slowly and the main barrier is the time for waiting is pretty short in our setting to perform an online service. Time cost increases for all the algorithms with a larger e_r. GreedyDP still runs the fastest and DAIF-DP is faster than DAIF-B. Similarly, the performances of prediction-based methods become better on SYN.

Impact of Grid Size g. Figure 7 shows the results of varying the grid size g. SHARE does not use a grid, thus we show results of SHARE with the default value of g for comparison. When g

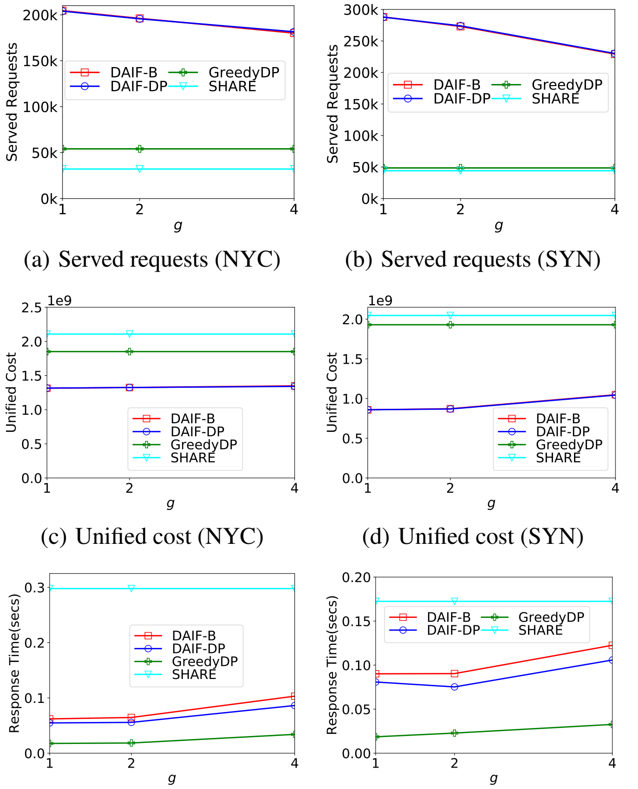


(a) Served requests (NYC) (b) Served requests (SYN)

(c) Unified cost (NYC) (d) Unified cost (SYN)

(e) Resp. time (NYC) (f) Unified cost (SYN)

Figure 6: Performance of varying deadline coefficient e_r



(a) Served requests (NYC) (b) Served requests (SYN)

(c) Unified cost (NYC) (d) Unified cost (SYN)

(e) Response time (NYC) (f) Response time (SYN)

Figure 7: Performance of varying grid size g

increases, the results fo the prediction model become vague and the travel cost in the same area increase, which leads to our demand-aware analyses less effective. Thus, the number of served requests of DAIF decreases and the unified cost of them increases when g increases. GreedyDP has no change in serving rate and unified cost when g changes. For the running time, a smaller g leads to a better grid prune and the time cost of all algorithms except SHARE becomes lower.

Impact of arranging idle workers. In our default setting, an idle worker would stay at the destination and provide a supply contribution to this area. What if the worker wanders around to look for new requests? In addition to the basic setting, we append random routes for idle workers. The result shows that the serving rate drops by 8.3% and the unified cost increases by 6.7% after a wander.

We modify the strategy by checking the supply shift after wander. After generating the random wander, we check whether the additional route improves the DSB. If so, we treat it as a good wander and add it; otherwise, we discard it and the worker just stays still. The result shows that the serving rate increases by 9.8% and the unified cost decrease by 6.7% with our filtered wander.

Summary of Results. We summarize the experimental results as follows:

- Our DAIF algorithms can serve 50% to 820% more requests than the state-of-art traditional ride-sharing algorithm [40] and the state-of-art prediction-based algorithm [45]. The unified cost of the results of DAIF algorithms is decreased by 11% to 74%.

- DAIF-B and DAIF-DP achieve nearly the same results on serving rate and unified cost. DAIF-DP is much faster than DAIF-B. Taking the prediction results into account, DAIF-DP is just slightly slower than GreedyDP and fast enough for the online ridesharing services. (i.e., serving a request in 0.2 second in large city-scale scenarios).

8. CONCLUSION

In this paper, we propose the DARP problem, which takes the balance of supply and predicted demand into account to reach a better assignment for potential profit instead of purely current profit. We prove that there is no polynomial-time algorithms with a constant competitive ratio to solve the DARP problem. We devise a basic insertion program with consideration of the balance score for DARP and develop a novel dynamic programming insertion program, which reduces the time complexity of insertion from cubic to quadric time. We then devise a grid-based solution leveraging the above DP-based insertion algorithm to address the DARP problem approximately. Extensive experiments on real datasets show that our proposed solutions outperform the state-of-the-art solutions in effectiveness greatly without sacrificing too much efficiency. Our paper provide a comprehensive theoretical reference for optimizing route planning with prediction information in shared mobility, and gives new direction for future research to design effective solutions with higher prediction accuracy and efficiency as guidance to large-scale shared mobility applications.

Acknowledgments

Jiachuan Wang, Libin Zheng and Lei Chen are partially supported by the Hong Kong RGC GRF Project 16209519, CRF project C603 0-18G, AOE project AoE/E-603/18, the National Science Foundation of China (NSFC) under Grant No. 61729201, Science and Technology Planning Project of Guangdong Province, China, No. 2015B010110006, Hong Kong ITC grants ITS/044/18FX and ITS/4 70/18FX, Didi-HKUST joint research lab Grant, Microsoft Research Asia Collaborative Research Grant, Wechat Research Grant and Webank Research Grant. Peng Cheng is supported by Shanghai Pujiang Program 19PJ1403300. Xuemin Lin is supported by NSFC61232006, 2018YFB1003504, ARC DP180103096 and DP2 00101338. Corresponding author: Peng Cheng.

9. REFERENCES

[1] [online] didi chuxing. `http://www.didichuxing.com/`.

[2] [online] geofabrik. `https://download.geofabrik.de/`.

[3] [online] meituan. `https://www.meituan.com/`.

[4] [online] tlc trip record data. `https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page/`.

[5] [online] uber. `https://www.uber.com/`.

[6] N. A. H. Agatz, A. L. Erera, M. W. P. Savelsbergh, and X. Wang. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, 223(2):295–303, 2012.

[7] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *PNAS*, 114(3):462–467, 2017.

[8] N. Ascheuer, S. O. Krumke, and J. Rambau. Online dial-a-ride problems: Minimizing the completion time. In *STACS*, volume 1770 of *Lecture Notes in Computer Science*, pages 639–650. Springer, 2000.

[9] M. Asghari, D. Deng, C. Shahabi, U. Demiryurek, and Y. Li. Price-aware real-time ride-sharing at scale: an auction-based approach. In *SIGSPATIAL*, pages 3:1–3:10. ACM, 2016.

[10] M. Asghari and C. Shahabi. An on-line truthful and individually rational pricing mechanism for ride-sharing. In *SIGSPATIAL*, pages 7:1–7:10. ACM, 2017.

[11] M. Charikar and B. Raghavachari. The finite capacity dial-a-ride problem. In *FOCS*, pages 458–467. IEEE Computer Society, 1998.

[12] P. Cheng, H. Xin, and L. Chen. Utility-aware ridesharing on road networks. In *SIGMOD Conference*, pages 1197–1210. ACM, 2017.

[13] J. Chu, K. Qian, X. Wang, L. Yao, F. Xiao, J. Li, X. Miao, and Z. Yang. Passenger demand prediction with cellular footprints. In *SECON*, pages 163–171. IEEE, 2018.

[14] B. Cici, A. Markopoulou, and N. Laoutaris. Designing an on-line ride-sharing system. In *SIGSPATIAL*, pages 60:1–60:4. ACM, 2015.

[15] A. Colorni and G. Righini. Modeling and optimizing dynamic dial-a-ride problems. *International transactions in operational research*, 8(2):155–166, 2001.

[16] L. Coslovich, R. Pesenti, and W. Ukovich. A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem. *European Journal of Operational Research*, 175(3):1605–1615, 2006.

[17] N. Cressie and C. K. Wikle. *Statistics for spatio-temporal data*. John Wiley & Sons, 2015.

[18] P. M. d'Orey, R. Fernandes, and M. Ferreira. Empirical evaluation of a dynamic and distributed taxi-sharing system. In *ITSC*, pages 140–146. IEEE, 2012.

[19] E. Feuerstein and L. Stougie. On-line single-server dial-a-ride problems. *Theor. Comput. Sci.*, 268(1):91–105, 2001.

[20] A. Gupta, M. T. Hajiaghayi, V. Nagarajan, and R. Ravi. Dial a ride from *k*-forest. *ACM Trans. Algorithms*, 6(2):41:1–41:21, 2010.

[21] Z. He, J. Cao, and X. Liu. High quality participant recruitment in vehicle-based crowdsourcing using predictable mobility. In *INFOCOM*, pages 2542–2550. IEEE, 2015.

[22] W. Herbawi and M. Weber. A genetic and insertion heuristic algorithm for solving the dynamic ridematching problem with time windows. In *GECCO*, pages 385–392. ACM, 2012.

[23] Y. Huang, F. Bastani, R. Jin, and X. S. Wang. Large scale real-time ridesharing with service guarantee on road networks. *PVLDB*, 7(14):2017–2028, 2014.

[24] H. Hung, R. Chapman, W. Hall, and E. Neigut. A heuristic algorithm for routing and scheduling dial-a-ride vehicles. In *ORSA/TIMS National Meeting*, 1982.

[25] J.-J. Jaw. *Solving large-scale dial-a-ride vehicle routing and scheduling problems*. PhD thesis, Massachusetts Institute of Technology, 1984.

[26] J.-J. Jaw, A. R. Odoni, H. N. Psaraftis, and N. H. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, 20(3):243–257, 1986.

[27] A. Kleiner, B. Nebel, and V. A. Ziparo. A mechanism for dynamic ride sharing based on parallel auctions. In *IJCAI*, pages 266–272. IJCAI/AAAI, 2011.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Commun. ACM*, volume 60, pages 84–90, 2017.

[29] Y. Li, Y. Zheng, H. Zhang, and L. Chen. Traffic prediction in a bike-sharing system. In *SIGSPATIAL*, pages 33:1–33:10. ACM, 2015.

[30] S. Ma, Y. Zheng, and O. Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *ICDE*, pages 410–421. IEEE, 2013.

[31] S. Ma, Y. Zheng, and O. Wolfson. Real-time city-scale taxi ridesharing. *IEEE Trans. Knowl. Data Eng.*, 27(7):1782–1795, 2015.

[32] M. Ota, H. T. Vo, C. T. Silva, and J. Freire. Stars: Simulating taxi ride sharing at scale. *IEEE Trans. Big Data*, 3(3):349–361, 2017.

[33] J.-F. Rougès and B. Montreuil. Crowdsourcing delivery: New interconnected business models to reinvent delivery. In *1st international physical internet conference*, pages 1–19, 2014.

[34] Z. B. Rubinstein, S. F. Smith, and L. Barbulescu. Incremental management of oversubscribed vehicle schedules in dynamic dial-a-ride problems. In *AAAI*. AAAI Press, 2012.

[35] D. O. Santos and E. C. Xavier. Dynamic taxi and ridesharing: A framework and heuristics for the optimization problem. In *IJCAI*, pages 2885–2891. IJCAI/AAAI, 2013.

[36] S. Shaheen, A. Cohen, and I. Zohdy. Shared mobility: current practices and guiding principles. Technical report, 2016.

[37] SUMC. [online] what is shared-use mobility? `https://goo.gl/3Jw6z7`, 2018.

[38] R. S. Thangaraj, K. Mukherjee, G. Raravi, A. Metrewar, N. Annamaneni, and K. Chattopadhyay. Xhare-a-ride: A search optimized dynamic ride sharing system with approximation guarantee. In *ICDE*, pages 1117–1128. IEEE, 2017.

[39] Y. Tong, L. Wang, Z. Zhou, B. Ding, L. Chen, J. Ye, and K. Xu. Flexible online task assignment in real-time spatial data. *PVLDB*, 10(11):1334–1345, 2017.

[40] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu. A unified approach to route planning for shared mobility. *PVLDB*, 11(11):1633–1646, 2018.

[41] N. H. Wilson, R. Weissberg, B. Higonnet, and J. Hauser. Advanced dial-a-ride algorithms. Technical report, 1975.

[42] N. H. M. Wilson, R. W. Weissberg, and J. Hauser. Advanced dial-a-ride algorithms research project. Technical report, 1976.

[43] Y. Xu, Y. Tong, Y. Shi, Q. Tao, K. Xu, and W. Li. An efficient insertion operator in dynamic ridesharing services. In *ICDE*, pages 1022–1033. IEEE, 2019.

[44] S. Yeung, E. Miller, and S. Madria. A flexible real-time ridesharing system considering current road conditions. In *MDM*, pages 186–191. IEEE Computer Society, 2016.

[45] C. F. Yuen, A. P. Singh, S. Goyal, S. Ranu, and A. Bagchi. Beyond shortest paths: Route recommendations for ride-sharing. In *WWW*, pages 2258–2269. ACM, 2019.

[46] Y. Zeng, Y. Tong, and L. Chen. Last-mile delivery made practical: An efficient route planning framework with theoretical guarantees. *PVLDB*, 13(3):320–333, 2019.

[47] Y. Zeng, Y. Tong, L. Chen, and Z. Zhou. Latency-oriented task completion via spatial crowdsourcing. In *ICDE*, pages 317–328. IEEE, 2018.

[48] J. Zhang, Y. Zheng, and D. Qi. Deep spatio-temporal residual networks for citywide crowd flows prediction. In *AAAI*, pages 1655–1661, 2017.

[49] J. Zhang, Y. Zheng, D. Qi, R. Li, and X. Yi. Dnn-based prediction model for spatio-temporal data. In *SIGSPATIAL*, pages 92:1–92:4. ACM, 2016.

[50] L. Zheng, L. Chen, and J. Ye. Order dispatch in price-aware ridesharing. *PVLDB*, 11(8):853–865, 2018.