Dynamic Trip-Vehicle Dispatch with Scheduled and On-Demand Requests

Taoan Huang
Tsinghua University
hta15@mails.tsinghua.edu.cn

Bohui Fang
Shanghai Jiaotong University
fangbohui@sjtu.edu.cn

Hoon Oh
Carnegie Mellon University
hooh@andrew.cmu.edu

Xiaohui Bei
Nanyang Technological University
xhbei@ntu.edu.sg

Fei Fang
Carnegie Mellon University
feifang@cmu.edu

ABSTRACT

In recent years, transportation service providers that dispatch drivers and vehicles to riders start to support both on-demand ride requests posted in real time and rides scheduled in advance. The presence of both types of ride requests leads to new challenges to the service providers in deciding which requests to accept and how to dispatch the vehicles in a dynamic and optimal way, which, to the best of our knowledge, have not been addressed by existing works. To fill the gap, we provide the following contributions: (i) a novel two-stage decision-making model where in the first stage, the system decides whether to accept requests scheduled in advance in an online fashion, and in the second stage, dispatches vehicles to on-demand ride requests in real time given the accepted scheduled requests as constraints; (ii) a novel sequential dynamic programming-based algorithm for the second stage given a set of accepted scheduled requests and an estimated distribution of on-demand ride requests; (iii) a randomized best-fit algorithm for the first stage which builds upon the algorithm for the second stage; (iv) in addition, we demonstrate the effectiveness of the algorithms through extensive experiments on a real-world dataset of an online ride-hailing platform.

KEYWORDS

Coordination and Control Models for Multiagent Systems; Socio-Technical Systems

1 INTRODUCTION

The growth in location-tracking technology, the popularity of smartphones, and the reduced cost in mobile network communications have led to a revolution in mobility and the prevalent use of on-demand transportation systems, with a tremendous positive societal impact on personal mobility, pollution, and congestion. One key research problem faced by transportation service providers is how to dispatch drivers or vehicles to pick up riders. There is a rising interest in this problem from the aspect of multi-agent control and coordination [2, 16, 28], and it is closely related to other research problems in multi-agent systems, including dispatching agents for collection and delivery of goods [39], traffic control [8, 34], metro lines scheduling, and last-mile transportation[15].

Recently, a growing number of transportation service providers start to support both on-demand ride requests posted in real time and rides scheduled in advance, providing riders with more flexible and reliable service. For example, ride-hailing platforms such as Uber, Lyft, and Didichuxing match drivers and their private cars and riders in real time upon riders' request and also allow the riders to schedule rides in advance. Similarly, companies such as Curb, Shenzhou Zhuanche and ComfortDelGro transform traditional taxi-hailing and chauffeured car service to satisfy scheduled rides as well as on-demand requests. Shuttle service and other services which traditionally rely on advance booking also start to accept on-demand requests in real time [4].

The presence of both scheduled requests and on-demand requests lead to new challenges to the service providers. Accepting scheduled requests becomes a two-sided sword: on the one hand, such requests reduce the demand uncertainty and give the service provider more time to prepare and optimize these trips; On the other hand, certain scheduled requests may lead to waste in time on the way to serve them or prevent the assigned driver from serving more valuable on-demand requests, hurting the overall revenue of the driver and the service provider. So it is important for the service providers to design a system that decides whether to accept the scheduled requests and how to dispatch the drivers and vehicles to ride requests optimally.

From an algorithmic perspective, the scheduled requests act as additional constraints when the system dispatches drivers to real-time on-demand requests, hence bringing extra complexity to the trip-vehicle dispatch problem. Some existing systems simply treat all scheduled requests as regular on-demand requests when their pick-up time is due, and directly apply an existing dispatch algorithm for on-demand requests [2, 26, 29, 36, 40, 42]. Such practice, however, overlooks a fundamental difference between scheduled and on-demand requests: any scheduled request, once accepted by the platform, becomes a commitment that the platform *must* fulfill. Many scheduled requests are for important purposes, such as going to the airport to catch a flight or attending an important meeting. Failing to serve these rides could hurt the credibility of the service provider and therefore its long-term sustainability. This presents to the design of the dispatch with new challenges, which, to the best of our knowledge, have not been addressed by existing works.

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 2019, Montreal, Canada

https://doi.org/doi

In this paper, we study the trip-vehicle dispatch with both scheduled and real-time requests. First, we propose a novel two-stage decision-making model for this problem. In the first stage (Stage 1), the system is presented with a sequence of scheduled requests, which are received before any of the on-demand requests. The system needs to select which requests to accept and decide how to dispatch vehicles to the accepted requests in an online fashion. In the second stage (Stage 2), the system will receive on-demand requests on top of these scheduled requests in real-time. The system needs to dispatch drivers to the on-demand ride requests in real-time while ensuring the scheduled requests are satisfied, or suggest relocations of empty vehicles. We assume the platform knows the spatio-temporal distribution of the on-demand requests, which can be estimated from historical data. However, due to the irregularity of scheduled requests, we do not make any prior assumption on the scheduled requests.

Our second contribution is a novel sequential dynamic programming-based algorithm for trip-vehicle dispatch in Stage 2 given a set of accepted scheduled requests. Based on the known distribution, the algorithm computes a score function for each vehicle, measuring the expected total value starting from a specific time-location pair to the next scheduled request assigned to it. We prove the optimality of the algorithm for single-driver setting, and it provides an efficient policy for the multiple-driver case.

Further, we design a randomized best-fit algorithm for the online selection and dispatch problem in the first stage. The algorithm builds upon the score function used in Stage 2, which efficiently takes into account the on-demand requests distribution. We also present theoretical bounds on the competitive ratio of online selection algorithms when there are no on-demand requests in the second stage. We demonstrate the effectiveness of the algorithms through extensive experiments on a real-world dataset of an online ride-hailing platform.

2 RELATED WORK

In the context of multi-agent systems, dispatching multiple agents for transportation tasks through a centralized platform has been a focus in the literature. While we focus on ride-hailing related problems, including goods delivery [39], traffic control [8, 34], metro lines scheduling, ride-sharing [16] and last-mile transportation [1, 15, 38], have been explored in several recent work.

The problem of trip-vehicle dispatch, which assigns vehicles to serve ride requests through a centralized platform, has been studied extensively, focusing either on real-time on-demand requests or scheduled requests. With only real-time on-demand requests, dispatch algorithms use different approaches, such as greedy match [26], collaborative dispatch [29, 36, 42], planning and learning framework [40], and receding horizon control approach [30]. A recent work [10] focuses on the dynamic vehicle routing problem but ignores the distribution of future demands and plans myopically based on the currently received requests. Methods with demand prediction have also been investigated [31, 37, 41]. Alonso-Mora et al. [2] propose a model for high-capacity dispatch which is further extended to consider demand distribution [3]. Coordinating dispatching and pricing has been studied in recent work [6, 13, 21, 28]. With only scheduled requests, well-known examples are the Dial-a-Ride Problem (DARP) [18, 33] and several variants of it [7, 14, 17, 19, 20, 25, 35]. However, none of these work considers the presence of both scheduled and on-demand requests, which is an important trend in modern on-demand transportation systems and is the focus of this paper.

In our model, the Stage 2 problem is similar to trip-vehicle dispatch with on-demand requests only, except that the accepted scheduled requests brought in hard constraints. However, simple extensions of existing algorithms, such as the algorithm proposed by Xu et al. [40] and a sampling-based approach [27], to our setting does not lead to a good performance as shown in our experiments.

Our Stage 1 problem is closely related to the problem of online packing/covering [11] and online bipartite graph matching [24]. Example problems in this domain include the online Steiner problem [5, 9, 23] and the online flow control packing problem [12, 22]. The online packing model could be applied to our Stage 1 problem, however, once geographical constraints are added into the model, none of those results or techniques could be directly applied to our setting.

3 MODEL

We consider a discrete-time, discrete-location model with single-capacity vehicles and impatient riders, and use drivers and vehicles interchangeably in the rest of the paper. We discuss relaxation of these assumptions in Section 7.

We split the duration of one day into T discrete time steps. The set of time steps is $[T] = \{1, \dots, T\}$. We also divide the whole map into a set of N different locations or regions $[N] = \{1, \dots, N\}$. We denote $\delta(u, v)$ to be the shortest time to travel from $u \in [N]$ to $v \in [N]$. $\mathcal{D}$ denotes the set of vehicles. Each vehicle $c \in \mathcal{D}$ is associated with time-location pairs $(\tilde{t}_c, \tilde{o}_c)$ and $(\tilde{t}'_c, \tilde{o}'_c)$, where $\tilde{t}_c$ represents the earliest time that c can be dispatched and $\tilde{o}_c$ represents its initial location. Similarly, $(\tilde{t}'_c, \tilde{o}'_c)$ is defined to be the time-location pair representing when and where c ends its service.

We employ a two-stage model for processing the scheduled requests and on-demand requests. In Stage 1, the system receives a sequence of scheduled requests. A scheduled request is described by a tuple $r = (o_r, d_r, t_r, v_r)$, representing a requested ride from the origin o_r to the destination d_r that needs to start at exactly time t_r, and v_r represents the *value* the platform will receive for serving this request. We assume v_r is given to the system when the request is made, e.g., provided by the rider or an external pricing scheme. When a scheduled request arrives, the platform should either accept and assign it to a specific driver or reject it. The decision needs to be made immediately before the next request arrives. We assume that the scheduled requests for a specific day will all arrive before the day starts. Let $\tilde{R}_c$ denote the set of accepted scheduled requests assigned to $c \in \mathcal{D}$ by the end of Stage 1 and define $\tilde{R} = \cup \tilde{R}_c$. W.l.o.g., we assume the vehicle always ends its service by serving a scheduled request in $\tilde{R}_c$. If not, a virtual scheduled request with request time and origin corresponding to $(\tilde{t}'_c, \tilde{o}'_c)$ could be added to $\tilde{R}_c$.

In Stage 2, the system starts to take real-time on-demand requests. An on-demand request is described by a tuple $r = (o_r, d_r, t_r, v_r)$ with the same meaning as scheduled requests. We define the type of an on-demand request r to be (o_r, d_r, t_r) and let $\mathcal{W}$ be the set of all possible types. We assume on-demand requests with type $w = (o, d, t)$ have the same value V_w (or $V_{o,d,t}$) and V_w $\forall w \in \mathcal{W}$ is known to the system in advance. This is a reasonable assumption if, for example, the variation in value for a trip of type w is small in history and can be estimated from historical data. Note that these requests are received in real-time, i.e., request r will appear exactly at time t_r. Upon receiving a set of on-demand requests R_t at time t, the platform needs to decide immediately for each request (1) either to dispatch it to an available vehicle c currently at location o_r , in which case vehicle c will start to serve the request and become available again at location d_r at time $t_r + \delta(o_r, d_r)$, (2) or to reject this request. During the processing of these on-demand requests, the platform also needs to ensure that *all* scheduled requests that it previously accepted must be served at their respective scheduled times and by their respective drivers. In Stage 2, we also allow for relocating a vehicle to location d when it is dispatched for no request. In this case, the vehicle will operate in the way as if it were taking a virtual request with destination d and value 0.

The goal of the system is to maximize the total value of all accepted requests, including both scheduled and on-demand requests. We do not assume any prior knowledge of the distribution of the scheduled requests, but we assume the distribution of real-time on-demand requests is known or can be estimated from historical data. Let random variable X_w denote the number of requests of type $w \in \mathcal{W}$ that the system receives in Stage 2. The on-demand request distribution is described as $\Pr[X_w \geq i]$, for all $w \in \mathcal{W}$ and $i \in \mathbb{N}$.

4 SOLUTION APPROACH FOR STAGE 2

The system needs to deploy two algorithms for trip-vehicle dispatch, one for each stage. The decisions made in Stage 1 will serve as constraints in Stage 2, and the design of Stage 1 algorithm should be adaptive to the dispatch rule in Stage 2. For example, if the system always dispatches drivers to long-distance trips in Stage 2, then it is better to accept only short-distance trips in Stage 1. Therefore, we should first analyze how to solve the problem in Stage 2, analogous to solving the inner layer of a two-layered optimization problem. In this section, we present our solution approach for Stage 2, based on which we study the problem for Stage 1 in the next section.

The problem in Stage 2 can be viewed as a Markov Decision Process (MDP), where the state encodes the time, the joint location and availabilities of all vehicles and information about the received on-demand requests at the time. The possible actions are the dispatch of available vehicles to requests or locations. The state transition function is determined by the action and demand distribution. The immediate reward is the total value of served requests. A dispatch algorithm in Stage 2 implicitly represents a policy for the MDP. We aim to design an algorithm that induces an optimal or near-optimal policy. When there is only one vehicle in the system, we design an algorithm DPDA (Dynamic Programming based Dispatch Algorithm), and prove that it induces the optimal policy for the MDP. The algorithm can be extended to multiple-driver setting to find the optimal policy, but it requires exponential memory and runtime since it is necessary to include the time-location pairs of all vehicles in the states used in dynamic programming. Therefore, we provide an alternative algorithm DPDA-SU (DPDA with Sequential Update) that extends DPDA by sequentially dispatching available vehicles and updating a virtual demand distribution.

Single-Vehicle Case. First, we provide details of the MDP model for the single-vehicle case, introduce the DPDA algorithm, and prove its optimality. When the system has only one vehicle c, the system is faced with an MDP defined by $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{V})$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P}$ is the state transition probability matrix and $\mathcal{V}$ is the reward function. The set of accepted scheduled requests $\tilde{R}_c$ should be served reliably, and we encode this constraint in the definition of $\mathcal{S}$ and $\mathcal{A}$.

A state $s \in \mathcal{S}$ is defined by $(t, l, R_{t,l})$ where $t \leq \min\{T, \tilde{t}_c'\}$ is the time, l is the location of the vehicle that is waiting to be dispatched, and $R_{t,l}$ is the set of currently received on-demand requests that can potentially be served by the vehicle while ensuring a reliable service for the scheduled requests. Define $D(t, l|r) := \{d : d \in [N], \delta(l, d) + \delta(d, o_r) \leq t_r - t\}$ as the set of locations the vehicle could leave for from l so that he will be able to reach (o_r, t_r) after arriving at the location. Then given R_t and $\tilde{R}_c$, we have $R_{t,l} = \{r' : r' \in R_t, o_{r'} = l, d_{r'} \in D(t, l|\tilde{r}_c^t), t_{r'} = t\}$ where $\tilde{r}_c^t \in R_c$ is the earliest scheduled request for c with a pick-up time at or after time t

Let $\mathcal{A}(s)$ be the set of available actions at state $s = (t, l, R_{t,l})$. If the pickup time of the request $\tilde{r}_c^t$ is t, then the only available action at s is to assign the vehicle to $\tilde{r}_c^t$. Otherwise, $\mathcal{A}(s)$ consists of two types of actions: assigning c to a request $r' \in R_{t,l}$, or relocating c to a location $l' \in D(t, l|\tilde{r}_c^t)$. The state transition probability $\mathcal{P}_{ss'}^a = \Pr[S_{\tau+1} = s'|S_\tau = s, A_\tau = a]$ is non-zero only when the vehicle becomes available again at (t', l') of s' after taking action a at s and $\mathcal{P}_{ss'}^a = \prod_{w \in \mathcal{W}} \Pr[X_w = X_{s',w}]$, where $X_{s',w}$ is the number of type-w requests in $R_{t',l'}$ at state s'. The immediate reward $\mathcal{V}_s^a$ is v_r if a corresponds to dispatching c to a scheduled or on-demand request r, and $\mathcal{V}_s^a = 0$ otherwise. The state value function and action value function under a policy π are denoted by $v_\pi(s)$ and $q_\pi(s, a)$ respectively. The total reward of the MDP is regarded as the sum of $\mathcal{V}_s^a$, without applying a discount factor.

To solve this MDP, we introduce the DPDA algorithm, which implicitly induces a policy for the MDP. As shown in Algorithm 2, DPDA suggests a way to make the online decision for the vehicle given its current state $s = (t, l, \mathcal{R}_{t,l})$ and $\tilde{r}_c^t$, with the aid of a score value score$(t, l|r)$. The score$(t, l|r)$ is defined for a vehicle at (t, l) with r ($t_r \geq t$) being the next request it is required to serve. The score value is defined by a dynamic programming based calculation as shown in Algorithm 1.

Lemma 4.1. *If $\tilde{R}_c$ contains only one (real or virtual) request r and by definition c ends its service upon completing r, then* score$(\cdot)$ *computed by Algorithm 1 satisfies*

$$\text{score}(t, l|r) = \mathbb{E}_{\mathcal{R}_{t,l}}\left[v_{\pi^*}(t, l, \mathcal{R}_{t,l})\right]$$

where π^* is the optimal policy and it follows

$$q_{\pi^*}(s, a) = \mathcal{V}_s^a + \text{score}(t_a, l_a|r),$$

where (t_a, l_a) is the time-location pair that action $a \in \mathcal{A}(s)$ will lead to when the vehicle becomes available again.

Proof. First we build up a transition graph $G' = (\mathcal{S}, E')$ among the set of possible states in the corresponding MDP, where $E' = \{(s,s')|\exists a \in \mathcal{A} \text{ s.t. } \mathcal{P}^a_{ss'} > 0\}$. Clearly, G' is a directed acyclic graph (DAG). Thus from Bellman Expectation Equation

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s)\left(\mathcal{V}^a_s + \sum_{s'\in\mathcal{S}} \mathcal{P}^a_{ss'} v_\pi(s')\right),$$

we can see that there exists an optimal deterministic policy, $\pi(a^*|s) = 1$ where

$$\begin{aligned} a^* &= \arg\max_{a\in\mathcal{A}(s)} \mathcal{V}^a_s + \sum_{s'\in\mathcal{S}} \mathcal{P}^a_{ss'} v_\pi(s') \\ &= \arg\max_{a\in\mathcal{A}(s)} \mathcal{V}^a_s + \mathbb{E}_{\mathcal{R}_{t_a,l_a}}\left[v_\pi(t_a, l_a, \mathcal{R}_{t_a,l_a})\right], \end{aligned} \tag{1}$$

and the state value together with the optimal policy can be determined following the topological ordering of states in G'.

Next we show that $\text{score}(t,l|r) = \mathbb{E}_{\mathcal{R}_{t,l}}\left[v_\pi(t,l,\mathcal{R}_{t,l})\right]$ by induction on t. It holds for $t = t_r$ where $\text{score}(t_r, l_r|r) = 0$ as line 1 of Algorithm 1 shows. Assume it holds for all (t,l) with $t > t'$. Then we are to show it is correct for $t = t'$. By the induction hypothesis and Equation (1), the platform will always choose an available action a with the highest $\mathcal{V}^a_s + \text{score}(t_a, l_a|r)$. In line 5-7 and as visualized in Figure 1, the platform will look in the order of $a_1, \ldots, a_j$ and pick the first location a_i that is a destination of a request $r \in \mathcal{R}_{t',l}$. Otherwise, the driver will be guided to drive idly to d^*. Let $P_i = \Pr[X_{l,a_i,t'} \geq 1 | X_{l,a_k,t'} = 0,\ \forall k < i]$, thus we have

$$\begin{aligned} &\mathbb{E}_{\mathcal{R}_{t',l}}\left[v_\pi(t',l,\mathcal{R}_{t',l})\right] \\ = &\sum_{i=1}^{j} P_i \cdot (V_{l,a_i,t'} + \text{score}(t' + \delta(l,a_i), a_i|r)) \\ &+ \Pr[X_{l,a_k,t'} = 0,\ \forall k \leq j] \cdot \text{score}(t' + \delta(l,d^*), d^*|r) \\ = &\ \text{score}(t', l|r). \end{aligned}$$

The last equality follows from line 8-13, which concludes the proof. It follows $q_{\pi^*}(s,a) = \mathcal{V}^a_s + \text{score}(t_a, l_a|r)$ as a corollary. □

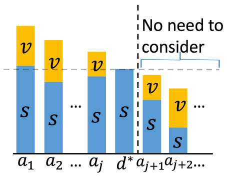


Figure 1: An Illustration of Algorithm 1. In the bar for a_i, s represents $\text{score}(t + \delta(l,a_i), a_i)$ and v represents $V_{l,a_i,t}$.

By lemma 4.1, the score value $\text{score}(t,l|r)$ is the weighted average state value of states with time t and location l and varying $R_{t,l}$. Therefore, it represents the expected value a vehicle at (t,l) could gain before it reaches r under the optimal policy. Thus the recursive algorithm shown in Algorithm 1 can be interpreted as follows. It first calculates $\text{score}(\cdot)$ for relevant future time-location pairs (line 1-4), then determines an ordered list of destination locations $a_1, \ldots, a_j, d^*$ worth considering (line 5-7), which encodes the system's preferences over requests. If there is a request to a_i but no request to a_k $\forall k < i$, the request to a_i will be served, leading to an immediate reward $V_{l,a_i,t}$ and an expected future gain of $\text{score}(t + \delta(l,a_i), a_i|r)$. The algorithm computes the score value $\text{score}(t,l|r)$ based on the probability that each of these events happens and the corresponding reward (line 9-12). The system will never consider certain requests since guiding the vehicle to drive idly towards d^* is more promising (line 13).

Thus back to the DPDA, we can see that it first calculates the score values $\text{score}(t_a, l_a|\tilde{r}^t_c)$ for all $a \in \mathcal{A}(s)$ (line 2-4) and chooses the action with the highest expected value the vehicle could gain before reaching $\tilde{r}_c$ (line 5). Next, we show that Algorithm 2 induces an optimal policy for the MDP.

Algorithm 1 Calculate $\text{score}(t,l|r)$

```
1:  if $l = o_r$ and $t = t_r$ then
2:      return 0
3:  for $d \in D(t,l|r)$ do
4:      Calculate $\text{score}(t + \delta(l,d), d|r)$
5:  Denote $\{a_i\}$ the sequence of $d \in S(l,t|r)$ in decreasing order
    of $V_{l,d,t} + \text{score}(t + \delta(l,d), d|r)$
6:  $d^* \leftarrow \arg\max_{d\in S_{l,t}} \text{score}(t + \delta(l,d), d|r)$
7:  $j \leftarrow$ the largest index of $\{a_i\}$ such that $V_{l,a_j,t} + \text{score}(t +$
    $\delta(l,a_j), a_j|r) > \text{score}(t + \delta(l,d^*), d^*|r)$
8:  $p \leftarrow 1$
9:  $F \leftarrow 0$
10: for $i = 1$ to $j$ do
11:     $F \leftarrow F + p \cdot \Pr[X_{l,a_i,t} \geq 1](V_{l,a_i,t} + \text{score}(t + \delta(l,a_i), a_i|r))$
12:     $p \leftarrow p \cdot (1 - \Pr[X_{l,a_i,t} \geq 1])$
13: $F \leftarrow F + p \cdot \text{score}(t + \delta(l,d^*,d^*)|r)$
14: $\text{score}(t,l|r) \leftarrow F$
```

Algorithm 2 $\text{DPDA}(s = (t,l,\mathcal{R}_{t,l})|\tilde{r}^t_c)$

```
1: Determine the action set $\mathcal{A}(s)$
2: for $a \in \mathcal{A}(s)$ do
3:     $(t_a, l_a,) \leftarrow$ the time-location pair action $a$ leads to
4:     Calculate $\text{score}(t_a, l_a|\tilde{r}^t_c)$
5: $a^* = \arg\max_{a\in\mathcal{A}(s)} \mathcal{V}^a_s + \text{score}(t_a, l_a|\tilde{r}^t_c)$
6: return $a^*$
```

Theorem 4.2. *Algorithm 2 induces an optimal policy.*

Proof-sketch We claim that the optimal policy π^* for state $s = (t_c, l_c, \mathcal{R}_{t_c,l_c})$ should only depend on $\tilde{r}^t_c$. Thus the overall MDP can be decomposed into several local MDPs with respect to each of the scheduled requests in $\tilde{R}_c$. Hence by Lemma 4.1, solving the overall MDP is equivalent to solving the local MDP corresponding to $\tilde{r}^t_c$ (line 5 in Algorithm 2), which concludes the proof. □

Multi-Vehicle Case. To compute the optimal solution for the multi-vehicle case at time step t, a dynamic programming that is similar to Algorithm 1 could still be applied. However, it suffers from the curse of dimensionality, which will lead to an exponential algorithm regarding time complexity and space complexity.To circumvent the difficulty, we provide a heuristic sequential algorithm as follows, as well the intuition behind. We start from the case with two vehicles c_1 and c_2. For the first vehicle c_1, we treat it as if it were the only vehicle in the system and we decide an action a^* for c_1 by running the DPDA. Let p_w be the probability a request of type $w \in \mathcal{W}$ being served by this vehicle given $X_w \geq 1$, and notice that p_w could be obtained during the computation of the score value. Afterward, we could obtain a new marginal distribution of on-demand requests. That is, given a^*, for any $i \in \mathbb{N}$ we have

$$\begin{aligned}\Pr[X'_w \geq i | a_{c_1} = a^*] = (1 - p_w) \cdot \Pr[X_w \geq i] \\ + p_w \cdot \Pr[X_w \geq i + 1]\end{aligned} \quad (2)$$

where random variable X'_w denotes the number of remaining requests of type w. Then for the second vehicle, we run the DPDA again as if it were the only vehicle and use the updated marginal distribution as the new distribution of on-demand requests.

For the case with more than 2 vehicles, we dispatch orders sequentially for each vehicle by simply repeating the procedure described above. That is, we sequentially run DPDA for each vehicle and update a virtual demand distribution. Note that after the second vehicle, the marginal distribution we maintained is not accurate anymore since we ignored their potential correlation. Nevertheless, it could serve as a reasonable estimation of the actual probability for our algorithm. In the description below, we use function $h(\cdot)$ to denote this estimated marginal distribution.

Following the intuition described above, we formally introduce the DPDA-SU in Algorithm 3. In line 1-3, the distribution is loaded in and the probabilities are stored in $h(\cdot)$'s. In line 4-7, we sequentially decide actions for each vehicle by the DPDA and then update the distribution through UpdateProbDist. UpdateProbDist$(h(\cdot), a, r)$ is derived based on Equation (2) and the calculation of score$(t, l|r)$. We defer the pseudocode to Appendix A[1].

Algorithm 3 DPDA-SU

```
1: Get the probabilities $\Pr[X_w \geq i]$ and value $V_w$ for all $w \in \mathcal{W}$
   from historical data
2: for $w \in \mathcal{W}, i \in \mathbb{N}$ do
3:     $h(X_w \geq i) \leftarrow \Pr[X_w \geq i]$
4: for $c \in \mathcal{D}$ do
5:     $r_c \leftarrow$ next scheduled request for $c$
6:     $a_c \leftarrow$DPDA$((t_c, l_c, \mathcal{R}_{t_c,l_c})|\tilde{r_c})$ with $h(\cdot)$ as $\Pr[\cdot]$
7:     $h(\cdot) \leftarrow$ UpdateProbDist$(h(\cdot), a_c, r_c)$
```

Note that when Algorithm DPDA-SU calls Algorithm 1 and 2 within, $h(\cdot)$ should be used as the probabilities $\Pr[\cdot]$.

5 SOLUTION APPROACH FOR STAGE 1

In this section, we design and analyze request selection algorithms for Stage 1. In this stage, the platform receives a sequence of scheduled requests and needs to decide their assignments in an online fashion. These requests are all received before any of the on-demand requests.

[1]An anonymous link to the Appendix: https://www.dropbox.com/s/8k3u9s5b0rw66tp/Appendix.pdf?dl=0

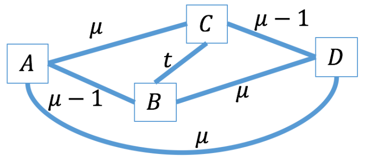


Figure 2: A Road Network Graph

We aim to design efficient online selection algorithms for Stage 1. To evaluate the performance of such online algorithms, we employ the notion of *competitive ratio*, which is a commonly used benchmark in online algorithm analysis. Given an input instance $\mathcal{I}$. We denote OPT$(\mathcal{I})$ and ALG$(\mathcal{I})$ as the optimal offline solution and the solution of an online algorithm on $\mathcal{I}$. We say the online algorithm is γ-competitive, if $\frac{\mathbb{E}[\text{OPT}(\mathcal{I})]}{\mathbb{E}[\text{ALG}(\mathcal{I})]} \leq \gamma$ holds for every problem instance $\mathcal{I}$.

As a start, we focus on Stage 1 problem alone without any interference from Stage 2. That is, we first assume that there is no on-demand request in Stage 2, and the goal of the selection algorithm is to select a set of feasible scheduled requests with maximum total value. We further assume that the value v_r of any scheduled request r is simply $\delta(o_r, d_r)$.

In this setting, we show a tight competitive ratio on any deterministic online algorithms. This ratio depends on a parameter μ, which is defined to be the ratio between the largest and smallest value of all possible requests.

Theorem 5.1. *If $v_r = \delta(o_r, d_r)$ and there is no on-demand request in Stage 2, any deterministic online algorithm for Stage 1 has a competitive ratio at least* $4\mu - 1$.

Proof. Consider a graph of 4 vertices A, B, C, D. Let the weights of edges $\delta(B,D) = \delta(C,A) = \delta(A,D) = \mu$, $\delta(A,B) = \delta(D,C) = \mu - 1$ and $\delta(B,C) = t$ $(1 \leq t \leq \mu)$, which represent the number of time steps required to travel along the edges as shown in Figure 2. Suppose that $T = 4\mu - 2 + t$ and there is only 1 vehicle. The vehicle starts work at A at time 1.

Consider an instance on this graph and time horizon $[T]$ where we have 6 requests $r_1 = (B, C, 2\mu, t)$, $r_2 = (A, D, 1, \mu)$, $r_3 = (D, C, \mu + 1, \mu - 1)$, $r_4 = (C, B, 2\mu, t)$, $r_5 = (B, A, 2\mu + t, \mu - 1)$, $r_6 = (A, D, 3\mu + t - 1, \mu)$.

The platform receives sufficiently many of r_1's at the beginning and the algorithm will take one of r_1's with a total revenue of t. This is because a deterministic algorithm should always accept the first feasible request, otherwise it would lead to a competitive ratio of ∞.

After that, $r_2, r_3, \ldots, r_6$ appear sequentially but the platform could accept none of them.

In this case, the offline optimal solution would have taken all the requests except r_1 to fill up the entire time horizon with total revenue of $4\mu - 2 + t$. Thus we have the ratio to be at least $\frac{4\mu-2+t}{t}|_{t=1} = 4\mu - 1$.

□

Next, we show that a simple first-fit algorithm that always dispatches requests to the first available vehicle if there exists one is $4\mu - 1$ competitive.

Algorithm FirstFit:

Fix an arbitrary order of the vehicles. For each incoming scheduled request, always assign it to the first vehicle in order that could serve this request without any conflicts. If no such vehicle exists, reject this request.

Theorem 5.2. *If $v_r = \delta(o_r, d_r)$ and there is no on-demand request in Stage 2, algorithm* FirstFit *for Stage 1 is $(4\mu-1)$-competitive.*

Proof. Let R_{ALG} be the set of requests the First-Fit algorithm accepts and R_{OPT} be the one of the offline optimal solutions. For any request $r = (o_r, d_r, t_r, v_r) \in R_{\text{ALG}}$, assume it is served by driver c. Let $t_{r,1} = t_r$ and $t_{r,2} = t_r + \delta(o_r, d_r)$. Consider a time interval $I = [t_{r,1} - (2\mu - 1), t_{r,2} + (2\mu - 1)]$.

We claim that the travel time interval of any request r' that is incompatible with request r and driver c, lies within I. This is because if the time interval of a request does not lie entirely in I, then the end time of this request should be no later than $t_{r,1} - \mu$ (or the start time of this request should be no earlier than $t_{r,2} + \mu$), thus it should be compatible with r.

Thus, the total value of the requests in R_{OPT} incompatible with r and driver c, is at most $4\mu - 2 + t_{r,2} - t_{r,1}$. Let $\text{OPT} = \sum_{r \in \mathcal{R}_{\text{OPT}}} v_r$ and $\text{ALG} = \sum_{r \in \mathcal{R}_{\text{ALG}}} v_r$, hence $\text{OPT} \le \sum_{r \in R_{\text{ALG}}} (4\mu - 2 + t_{r,2} - t_{r_1})$. As a result,

$$\begin{aligned}
\frac{\text{OPT}}{\text{ALG}} &= \frac{\text{OPT}}{\sum_{r \in R_{\text{ALG}}} t_{r,2} - t_{r,1}} \\
&\le \frac{\sum_{r \in R_{\text{ALG}}} 4\mu - 2 + t_{r,2} - t_{r,1}}{\sum_{r \in R_{\text{ALG}}} t_{r,2} - t_{r,1}} \\
&= \frac{|R_{\text{ALG}}|(4\mu - 2)}{\sum_{r \in R_{\text{ALG}}} t_{r,2} - t_{r,1}} + 1 \\
&\le \frac{|R_{\text{ALG}}|(4\mu - 2)}{|R_{\text{ALG}}|} + 1 = 4\mu - 1.
\end{aligned}$$

□

Next, we take into consideration the Stage 2 on-demand requests, which are assumed to follow the distribution $Pr[X_w \ge i]\, \forall w \in \mathcal{W}, i \in \mathbb{N}$. However, we do not know in advance which on-demand requests a car will serve before and after a scheduled request.

First, upon the arrival of a scheduled request r, for each vehicle c that can serve r, we estimate the expected value increment from this assignment with the help of the score value $\text{score}(t, l|r)$. More specifically, let r_0 and r_1 be the requests vehicle c serves before and after r (if r_0 or r_1 does not exist, we set a virtual request that corresponds to the start or end time-location pair of vehicle c). Then we set

$$E_0 = \text{score}(t_{r_0} + \delta(o_{r_0}, d_{r_0}), d_{r_0} \mid r_1)$$

to be the estimated value of vehicle c without taking request r, and

$$\begin{aligned}
E_1 = \quad & \text{score}(t_{r_0} + \delta(o_{r_0}, d_{r_0}), d_{r_0} \mid r) \\
& + v_r + \text{score}(t_r + \delta(o_r, d_r), d_r \mid r1)
\end{aligned}$$

to be the estimated value of vehicle c after taking request r. We then define the estimated value increment of request r for vehicle c as $\Delta_c(r) = E_1 - E_0$.

Such estimation suggests a greedy algorithm as follow.

Algorithm BestScore:

For each incoming scheduled request r:

- assign it to the vehicle c with which serving r could yield the highest value increment $\Delta_c(r)$;
- if no vehicle could serve r, reject this request.

Note that in this algorithm, we do not reject any requests as long as there are vehicles that can serve it. We could also consider the following two variants of BestScore: (1) accept a request only if the highest incremented value $\Delta_c(r)$ is positive; (2) if the highest incremented value $\Delta_c(r)$ is negative, accept request r and assign it to c with probability $e^{\Delta_c(r)}$. We denote these two variants as BestScore-R and BestScore-A, respectively. As we will see in the experiments in Section 6, these two variants both result in lower performance than the original algorithm.

Finally, in the last algorithm, we add an additional random priority component to the value increment. Inspired by the online bipartite graph matching algorithm proposed by [24], we assign each vehicle a weight $\text{rank}(k) = e^{\alpha k}$ that denotes its priority, where k is a random variable drawn from the uniform distribution $U[0, 1]$ independently for each vehicle and α is a constant. The new estimated value increment of vehicle c serving request r then becomes $\Delta'_c(r) = E_1 - E_0 + \beta * e^{\alpha k}$, where β is another scaling parameter.

Our final algorithm, RandomBestScore, choose the vehicle based on this newly randomized value increment.

Algorithm RandomBestScore:

For each coming scheduled request r:

- assign it to the vehicle c with which serving r could yield the highest randomized value increment $\Delta'_c(r)$;
- if no vehicle could serve r, reject this request.

One could also define the two variants, RandomBestScore-R and RandomBestScore-A, in a similar way to the two variants of BestScore.

6 EXPERIMENT

In this section, we demonstrate the effectiveness of the proposed algorithms in real-world applications through extensive experiments. First, we introduce the dataset and describe how we process and extract information from it. Then we introduce baseline algorithms for both stages and present the experimental results.

6.1 Data Description and Processing

We perform our empirical analysis based on a dataset provided by Didichuxing. The dataset consists of 2×10^5 valid requests. Each request r consists of the start time t_r, the duration of the trip, the origin o_r, the destination d_r, and its assigned vehicle ID. The value

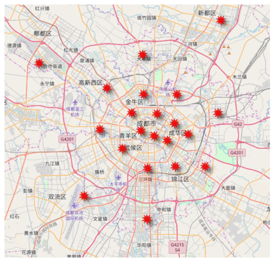


Figure 3: The Clustered Centers in a City in China Derived from Data.

v_r is not given from the dataset, and we set it to be proportional to the duration of the trip.

The locations in the dataset are represented by latitudes and longitudes. We transform them into discretized regions by running a k-means clustering algorithm on all the valid coordinates. We obtain 21 centers after 61 rounds of iteration as shown in Figure 3. Then the discretized label of each location in the dataset is represented by the label of its nearest center and $\delta(o,d)$ are calculated based on the coordinates of centers of regions o,d. The time horizon is discretized into 1 minute per time step. Finally, the distribution of on-demand requests from the data can be derived given the discretized time horizon and regions.

For each vehicle c, $(\tilde{t_c}, \tilde{o_c})$ are used as the earliest occurrence time and location of c given in the dataset. W.l.o.g., we set $(\tilde{t'_c}, \tilde{o'_c}) = (\tilde{t_c}, \tilde{o_c})$.

6.2 Experiment Setup

All experiments are done on an i7-6900K@3.20GHz CPU with 16G memory. We introduce the default global setup for all the experiments. The duration of a time step is set to 1 minute. Next, we sample on-demand requests from the historical distribution derived from the dataset, with an average number of 1804 generated requests in each iteration. For scheduled requests in Stage 1, we set their frequency to be 1/20 of that of the on-demand requests in Stage 2, and the types of 87 scheduled requests are drawn i.i.d. from $\mathcal{W}$ following the on-demand requests distribution. The value of a request of type w is set to be V_w. Finally, a set of 50 vehicles are drawn uniformly from the dataset. All experiments below follow this setup unless specified otherwise.

6.3 Baseline Algorithm

We compare our algorithms with several baseline algorithms for both Stage 1 and Stage 2. For Stage 1, we employ the First-Fit algorithm as the baseline.

For Stage 2, we employ two matching based algorithms, a learning and planning based algorithm (LPA), and a sampling-based mixed integer linear programming (S-MILP) algorithm.

The first matching based algorithm, named Greedy-KM, dispatch requests myopically considering only their values. The second algorithm, named Enhanced KM, is an extension of Greedy-KM with the score value. The details of both algorithms are deferred to Appendix C.

The LPA is adapted from Xu et al. [40] to handle the hard constraints brought in by the scheduled requests and we implement it with slight changes of the setting in [40]. The details of the LPA is deferred to Appendix D.

In [27], assignments between vehicles and riders at time step t are made by solving a MILP that takes into account several samples of requests at the time step $t+1$. The S-MILP is an extension of [27] by adding scheduled requests as constraints in the MILP. In our experiments, the number of samples is set to 10.

6.4 Experimental Results

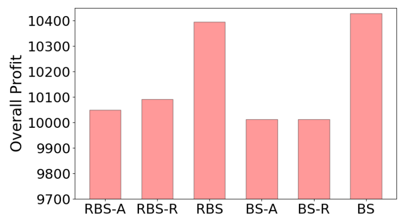
First, we combine DPDA-SU for Stage 2 with BestScore (BS), RandomBestScore (RBS) and their variants (BestScore-A, BestScore-R, RandomBestScore-A, RandomBestScore-R) for Stage 1, to evaluate the performance of all combinations of our proposed algorithms. For the multi-vehicle case, the results are shown in Figure 4a. One can see that each of RandomBestScore and BestScore outperforms its variants significantly. Thus, in the rest of the experiments, we employ RandomBestScore and BestScore as our Stage 1 algorithms. We also provide additional results for the single-vehicle case in Appendix E.

Next, we conduct experiments on pairwise combinations of Stage 1 and Stage 2 algorithms, as well as the LPA. The results are shown in Figure 4b. We can conclude that when one of FirstFit, BestScore, RandomBestScore is fixed for Stage 1, DPDA-SU always outperforms Greedy-KM, Enhanced-KM, and S-MILP. Though the LPA outperforms the other combinations without DPDA-SU, those with DPDA-SU are significantly better than the LPA.

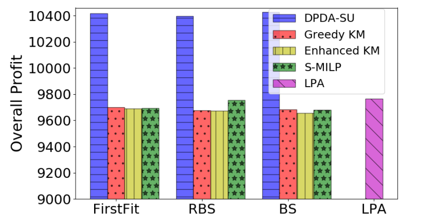
We further test our algorithms by varying the market parameters. We test with markets of different numbers of requests of 10^2, 10^3, 10^4, and different ratios κ between the numbers of scheduled and on-demand requests. We deploy FirstFit combined with Greedy-KM as one of the baselines. The reason we do not choose FirstFit with Enhanced-KM is that Greedy-KM outperforms Enhanced-KM in all combinations as shown in Figure 4b. We also choose the LPA and S-MILP as the other two baselines.

Figure 4d shows the increase of profit of our algorithms compared to the baselines. In the small market of 100 requests, the baselines perform better than our algorithms in some cases. However, the significance test shows the p-values are significantly larger than 0.1 in all cases in this market, which means no statistical conclusion can be drawn from these experiments. For larger markets of 10^3 and 10^4 requests, our algorithms are on average better than the baselines for every κ. We also conduct the significance test in each market and the p-values in all cases are less than 10^{-6}. Thus one can statistically conclude that our algorithm outperforms the baselines in large markets.

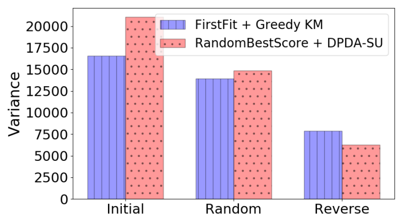
To verify the effectiveness of Stage 1 algorithms, we empirically compute the competitive ratios under the setting with only scheduled requests for each of FirstFit, BestScore, and RandomBestScore. We generate 50 instances, each with 87 scheduled requests. For each algorithm ALG, we compute $\frac{\mathbb{E}[\mathrm{OPT}(I)]}{\mathbb{E}[\mathrm{ALG}(I)]}$ for each instance $\mathcal{I}$ and the maximum is taken over all 50 instances as the empirical competitive ratio for ALG. For RandomBestScore which is a randomized algorithm, we run the algorithm on each instance 50 times and take the average output value as the estimate of $\mathbb{E}[\textsc{RandomBestScore}(I)]$.



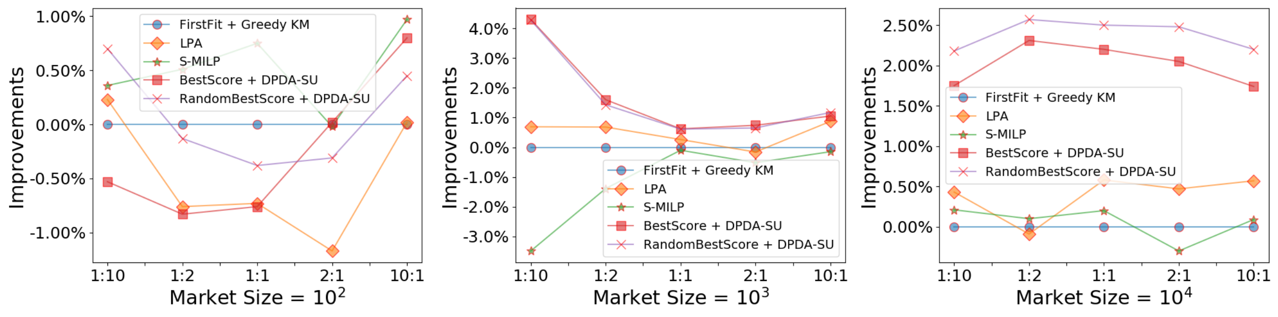
(a) Performances of Stage 1 Algorithms.



(b) Performance of Pairwise Combinations and the LPA.



(c) Variances when Using Different Sequence Choices.



(d) Improvement w.r.t. the Baseline in Different Market Sizes.

Figure 4: Experimental Results

The offline optimal value for each instance OPT($\mathcal{I}$) can be calculated by a flow-based approach, as described in Appendix B. The empirical ratios are summarized in Table 1, which shows that our algorithms have relatively low empirical competitive ratio compared to FirstFit. This suggests that BestScore and RandomBestScore are also good candidate algorithms for markets with only scheduled requests.

Algorithms in Stage 1	Competitive Ratio
BestScore	**1.38609**
RandomBestScore	1.39112
FirstFit	1.4454

Table 1: Stage 1 Competitive Ratios of Different Models.

In addition to the overall profit, we also test the variance of values gained by each vehicle with our algorithms. This relates to the fairness of a dispatching algorithm which is a practical concern in many ride-hailing platforms with self-interested drivers. Although reducing the variance is not the main focus of this paper, we investigate how the choice of sequence of vehicles impacts the variance. To this end, we consider the choice of vehicle sequences before running DPDA-SU that could lead to low variances without harming the total value. We test three variations. In the first one, denoted as Initial, we fix a vehicle order. In the second one, denoted as Reverse, we sort the vehicles in increasing order of the values they have already gained before running DPDA-SU. In the last variation, denoted as Random, we shuffle the vehicles randomly. When using these three variations in the same algorithm, the differences in the total value are within 0.60% from each other. The variance results are shown in Figure 4c. One can see that when applying Reverse, our algorithm RandomBestScore +DPDA-SU leads to a lower variance than FirstFit combined with Greedy-KM.

Finally, we evaluate the scalability of Stage 2 algorithms in terms of their space complexities and running times. The space complexities of different algorithms are summarized in Table 2. Here we denote $|\mathcal{D}|$ as the number of vehicles, M as the total requests, m as the maximum number of requests at one time step, N as the number of regions on the map, T as the total time steps, and θ as the sampling times for only S-MILP algorithm ($\theta = 10$ in experiments).

Because most of the baseline algorithms are heuristics or mixed integer linear programs, it is hard to analyze their theoretical time complexities. Instead, we evaluate the running times of these algorithms in a fixed time window with different numbers of vehicles. To test the most stressful situation, following the derived distribution, one time step with the most serious congestion is selected and amplified. On an average, 519 on-demand requests are generated. We assume no scheduled requests in Stage 1. For the market with 1000, 3000, 5000 idle vehicles, we test the running time respectively, and the results are summarized in Table 2. Though S-MILP has the shortest running time, when the number of vehicles increases, memory soon becomes a bottleneck for S-MILP. This is because the space required for S-MILP is quadratic in the number of vehicles. In our experiments, S-MILP runs out of memory when the number of vehicles reaches 5200 or higher. On the other hand, the space required for DPDA-SU is linear in the number of vehicles. As a result, our algorithm can handle much larger markets than S-MILP.

Vehicles	1000	3000	5000	Space Complexity
S-MILP	3.3	4.6	10.1	$O(\|\mathcal{D}\|m\theta \cdot \max(\|\mathcal{D}\|, m))$
DPDA-SU	10.4	31.6	56.2	$O(N^2T\|\mathcal{D}\|)$
Greedy-KM	1.7	32.0	133.7	$O(\|\mathcal{D}\|m)$
LPA	2.7	35.2	147.1	$O(\max(\|\mathcal{D}\|m, MT))$
Enhanced-KM	9.5	56	185.3	$O(N^2T\|\mathcal{D}\| + \|\mathcal{D}\|m))$

Table 2: Running Time (in seconds) with Different Number of Vehicles and Space Complexities of Different Algorithms.

7 CONCLUSION AND FUTURE WORK

In this paper, we investigated the problem of trip-vehicle dispatch with the presence of scheduled and on-demand request. We proposed a novel two-stage model and novel algorithms for both stages. Through extensive experiments, we demonstrated the effectiveness of the algorithms for real-world applications.

We consider a discrete-time discrete-location model with impatient requests, which is already a challenging problem when taking into account two types of request and the demand distribution. The model can be applied to or further extended for problems with relaxed assumptions. For example, our work can be applied to problems with patient requests, which can be treated as duplicated requests when there is only one driver. Our work can also be integrated with work on last-mile routing to handle the actual road network. We defer the further investigation to future work.

REFERENCES

[1] Lucas Agussurja, Shih-Fen Cheng, and Hoong Chuin Lau. 2018. A state aggregation approach for stochastic multi-period last-mile ride-sharing problems. *Transportation Science* (2018).

[2] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. 2017. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences* 114, 3 (2017), 462–467.

[3] Javier Alonso-Mora, Alex Wallar, and Daniela Rus. 2017. Predictive routing for autonomous mobility-on-demand systems with ride-sharing. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 3583–3590.

[4] Apple Inc. 2018. App Store Preview, Supershuttle. https://itunes.apple.com/us/app/supershuttle/id376771013?mt=8. (2018).

[5] Baruch Awerbuch, Yossi Azar, and Yair Bartal. 2004. On-line generalized Steiner problem. *Theoretical Computer Science* 324, 2-3 (2004), 313–324.

[6] Jiaru Bai, Kut C So, Christopher S Tang, Xiqun Chen, and Hai Wang. 2018. Coordinating supply and demand on an on-demand service platform with impatient customers. *Manufacturing & Service Operations Management* (2018).

[7] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. 2012. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research* 218, 1 (2012), 1–6.

[8] Ana LC Bazzan and Camelia Chira. 2015. A Hybrid Evolutionary and Multiagent Reinforcement Learning Approach to Accelerate the Computation of Traffic Assignment. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1723–1724.

[9] Piotr Berman and Chris Coulston. 1997. On-line algorithms for Steiner tree problems. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. ACM, 344–353.

[10] Dimitris Bertsimas, Patrick Jaillet, and Sébastien Martin. 2018. Online Vehicle Routing: The Edge of Optimization in Large-Scale Applications. (2018).

[11] Niv Buchbinder and Joseph Naor. 2005. Online primal-dual algorithms for covering and packing problems. In *European Symposium on Algorithms*. Springer, 689–701.

[12] Niv Buchbinder and Joseph Naor. 2009. Online primal-dual algorithms for covering and packing. *Mathematics of Operations Research* 34, 2 (2009), 270–286.

[13] Mengjing Chen, Weiran Shen, Pingzhong Tang, and Song Zuo. 2017. Optimal Vehicle Dispatching Schemes via Dynamic Pricing. *arXiv preprint arXiv:1707.01625* (2017).

[14] Zhi-Long Chen and Hang Xu. 2006. Dynamic column generation for dynamic vehicle routing with time windows. *Transportation Science* 40, 1 (2006), 74–88.

[15] Shih-Fen Cheng, Duc Thien Nguyen, and Hoong Chuin Lau. 2014. Mechanisms for arranging ride sharing and fare splitting for last-mile travel demands. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1505–1506.

[16] Brian J Coltin and Manuela Veloso. 2013. Towards ridesharing with passenger transfers. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1299–1300.

[17] Jean-François Cordeau. 2006. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research* 54, 3 (2006), 573–586.

[18] Jean-François Cordeau and Gilbert Laporte. 2007. The dial-a-ride problem: models and algorithms. *Annals of operations research* 153, 1 (2007), 29–46.

[19] Guy Desaulniers, Fausto Errico, Stefan Irnich, and Michael Schneider. 2016. Exact algorithms for electric vehicle-routing problems with time windows. *Operations Research* 64, 6 (2016), 1388–1405.

[20] Alain Faye and Dimitri Watel. 2016. Static Dial-a-Ride Problem with Money as an Incentive: Study of the Cost Constraint. (2016).

[21] Amos Fiat, Yishay Mansour, and Lior Shultz. 2018. Flow Equilibria via Online Surge Pricing. *arXiv preprint arXiv:1804.09672* (2018).

[22] Naveen Garg and Neal E Young. 2002. On-line end-to-end congestion control. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*. IEEE, 303–310.

[23] Makoto Imase and Bernard M Waxman. 1991. Dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics* 4, 3 (1991), 369–384.

[24] Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. 1990. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. ACM, 352–358.

[25] Taehyeong Kim. 2011. *Model and algorithm for solving real time dial-a-ride problem.* Ph.D. Dissertation.

[26] Der-Horng Lee, Hao Wang, Ruey Cheu, and Siew Teo. 2004. Taxi dispatch system based on current demands and real-time traffic conditions. *Transportation Research Record: Journal of the Transportation Research Board* 1882 (2004), 193–200.

[27] Meghna Lowalekar, Pradeep Varakantham, and Patrick Jaillet. 2018. Online spatio-temporal matching in stochastic and dynamic domains. *Artificial Intelligence* 261 (2018), 71–112.

[28] Hongyao Ma, Fei Fang, and David C Parkes. 2018. Spatio-Temporal Pricing for Ridesharing Platforms. *arXiv preprint arXiv:1801.04015* (2018).

[29] Shuo Ma, Yu Zheng, and Ouri Wolfson. 2013. T-share: A large-scale dynamic taxi ridesharing service. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE, 410–421.

[30] Fei Miao, Shuo Han, Shan Lin, John A Stankovic, Desheng Zhang, Sirajum Munir, Hua Huang, Tian He, and George J Pappas. 2016. Taxi dispatch with real-time sensing data in metropolitan areas: A receding horizon control approach. *IEEE Transactions on Automation Science and Engineering* 13, 2 (2016), 463–478.

[31] Luis Moreira-Matias, Joao Gama, Michel Ferreira, Joao Mendes-Moreira, and Luis Damas. 2013. Predicting taxi–passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems* 14, 3 (2013), 1393–1402.

[32] James Munkres. 1957. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics* 5, 1 (1957), 32–38.

[33] Ida Nedregård. 2015. *The Integrated Dial-a-Ride Problem-Balancing Costs and Convenience.* Master's thesis. NTNU.

[34] Deepika Pathania and Kamalakar Karlapalem. 2015. Social network driven traffic decongestion using near time forecasting. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1761–1762.

[35] Douglas O Santos and Eduardo C Xavier. 2015. Taxi and ride sharing: A dynamic dial-a-ride problem with money as an incentive. *Expert Systems with Applications* 42, 19 (2015), 6728–6737.

[36] Kiam Tian Seow, Nam Hai Dang, and Der-Horng Lee. 2010. A collaborative multiagent taxi-dispatch system. *IEEE Transactions on Automation Science and Engineering* 7, 3 (2010), 607–616.

[37] Yongxin Tong, Yuqiang Chen, Zimu Zhou, Lei Chen, Jie Wang, Qiang Yang, Jieping Ye, and Weifeng Lv. 2017. The simpler the better: a unified approach to predicting original taxi demands based on large-scale online platforms. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1653–1662.

[38] WJA van Heeswijk, Martijn RK Mes, and Johannes MJ Schutten. 2017. The delivery dispatching problem with time windows for urban consolidation centers. *Transportation science* (2017).

[39] Chao Wang, Somchaya Liemhetcharat, and Kian Hsiang Low. 2016. Multi-agent continuous transportation with online balanced partitioning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent*

Systems. International Foundation for Autonomous Agents and Multiagent Systems, 1303–1304.

[40] Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, and Jieping Ye. 2018. Large-Scale Order Dispatch in On-Demand Ride-Hailing Platforms: A Learning and Planning Approach. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 905–913.

[41] Lingyu Zhang, Tao Hu, Yue Min, Guobin Wu, Junying Zhang, Pengcheng Feng, Pinghua Gong, and Jieping Ye. 2017. A taxi order dispatch model based on combinatorial optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2151–2159.

[42] Rick Zhang and Marco Pavone. 2016. Control of robotic mobility-on-demand systems: a queueing-theoretical perspective. *The International Journal of Robotics Research* 35, 1-3 (2016), 186–203.

Appendices

A ALGORITHM FOR UPDATEPROBDIST

We show in Algorithm 5 the update of distribution. It calculates p_w's following the intuition as we introduced (line 1-3) and then the distribution is updated following Equation (2) (line 4-6). Algorithm 4 shows the calculation of p_w's, which follows a routine similar to calculating the score value in Algorithm 1

Algorithm 4 GetProbability$(l, t, x, h(\cdot)|r)$

```
1:  if $l = l_r \wedge t = t_r$ then return
2:  Retrieve the score values score$(d, t + \delta(l, d)|r)$ for $d \in S(l, t|r)$
3:  $d^* \leftarrow \arg\max_{d \in S(l,t|r)}$ score$(d, t + \delta(l, d)|r)$
4:  Denote $\{a_i\}$ the sequence of $d \in S(l, t|r)$ in decreasing order
    of $V_{l,d,t}$ + score$(d, t + \delta(l, d)|r)$
5:  $j \leftarrow$ the largest index of $\{a_i\}$ such that $V_{l,a_j,t}$ + score$(a_j, t +$
    $\delta(l, a_j)|r) >$ score$(d^*, t + \delta(l, d^*)|r)$
6:  $p \leftarrow 1$
7:  for $i = 1$ to $j$ do
8:      $w \leftarrow (l, a_i, t)$
9:      $p_w \leftarrow p_w + x \cdot p$
10:     GetProbability$(a_i, t + \delta(l, a_i), x \cdot p \cdot h(X_w \geq 1), h(\cdot)|r)$
11:     $p \leftarrow p \cdot (1 - h(X_w \geq 1))$
12: GetProbability$(d^*, t + \delta(l, d^*), x \cdot p, h(\cdot)|r)$
```

Algorithm 5 UpdateProbDist$(h(\cdot), a, r)$

```
1:  Initialize $p_w \leftarrow 0$ for all $w \in \mathcal{W}$
2:  $(l_a, t_a) \leftarrow$ the location-time pair action $a$ leads to
3:  GetProbability$(l_a, t_a, 1, h(\cdot)|r)$
4:  for $w \in \mathcal{W}$ do
5:      for $i = 1$ to $|\mathcal{D}|$ do
6:          $h(X_w \geq i) \leftarrow (1 - p_w) \cdot h(X_w \geq i) + p_w \cdot h(X_w \geq i + 1)$
7:  return $h(\cdot)$
```

B OFFLINE ALGORITHM FOR STAGE 1 WITHOUT ON-DEMAND REQUESTS

The offline optimal solution of Stage 1 without on-demand requests can be obtained by solving a maximum cost network flow (MCNF).

Given the set of available vehicles $\mathcal{D}$ and all the scheduled requests $\mathcal{R}$, we construct a network $G = (V, E)$. We construct two vertices v_i and v'_i for each vehicle i, one entry-vertex $v_{r,in}$ and one exit-vertex $v_{r,out}$ for each request r, 2 virtual vertices S and T as the global source and sink.

We construct four types of edges in E. First, we construct edges from S to v_i and from v'_i to T, each with flow 1 and cost 0. Secondly, we construct edges from $v_{r,in}$ to $v_{r,out}$, with flow 1 and cost v_r, which mean each request could be taken no more than once. Thirdly, we construct edges from v_i to $v_{r,in}$ and from $v_{r,out}$ to v'_i, each with flow 1 and cost 0, which mean the first and last request the vehicle i could possibly served. Lastly, we construct edges from $v_{r_i,out}$ to $v_{r_j,in}$ if the distance between region i and region j allows a vehicle to pick up request j after serving request i.

Then by applying any MCNF algorithm, we could obtain the optimal solution.

C THE GREEDY-KM AND ENHANCED-KM

Greedy-KM works as follows. Given the set of available vehicles $\mathcal{D}$ and the state $(t_c, l_c, \mathcal{R}_{t_c,l_c})$ of each vehicle, we construct a bipartite graph $G_B = (\mathcal{D}, \bigcup_{c \in \mathcal{D}} \mathcal{R}_{t_c,l_c}, E_B)$, where we have edges between $c \in \mathcal{D}$ and $r \in \mathcal{R}_{t_c,l_c}$ with weight v_r. Greedy-KM dispatches order by finding a weighted maximum matching on G_B. In implementation, we employ the Kuhn-Munkres (KM) algorithm [32] to solve it.

In Enhanced-KM, the bipartite graph is constructed in the same way as Greedy-KM, except that the edges between $c \in \mathcal{D}$ and $r \in \mathcal{R}_{t_c,l_c}$ have weight $v_r + \text{score}(t_c + \delta(o_r, d_r), d_r | \tilde{r_c})$, where $\tilde{r_c}$ is the next committed scheduled request of vehicle c.

D LEARNING & PLANNING ALGORITHM

The LPA is an adaptation of the work of Xu et al. [40] to the hard constraints brought in by the scheduled requests.

In their work, they regard consider the transportation as the MDP and construct a local-view MDP for each driver, with location-time pairs as the states. As for the state transition rules and rewards for each state, they are drawn from the historical data. Actions of drivers are to pick up on-demand requests nearby or to stay still. For an action that lasts for T' time steps with reward R, they apply a discount factor γ and the final reward is given by

$$R_\gamma = \sum_{t=0}^{T'-1} \gamma^t \frac{R}{T'}.$$

At every time step, they obtain the value function v' for all states and then dispatch orders via a matching approach. The calculation of the value function is shown as Algorithm 6. We are using the same notation in Algorithm 6 as Xu et al. [40] did, which do not have the same meaning as those in our main text.

We do the following to adapt their algorithms to our two-stage model. In Stage 1, we parse the scheduled requests and decide immediately for request r by the comparison of $R_\gamma(r) + V'(d_r, t_r + \delta(o_r, d_r))$ and $V'(o_r, t_r)$, meaning that a request would be accepted if it could lead to an increment in the expected value. In Stage 2, we will use the same reward function as the edge weights for all the possible state transitions. It is worth noting that we will forbid the driver to pick up an order whose ending time is too late for next scheduled request.

Moreover, Xu et al. [40] mentioned the updates in Algorithm 6 can be done iteratively with the planning. The time window in our algorithm, however, is only one day. Thus this iteration cannot help optimize the value function.

E SINGLE-VEHICLE CASE PERFORMANCE

In Figure 5, we demonstrate the single-vehicle performances among all BestScore, RandomBestScore,BestScore-A, BestScore-R, RandomBestScore-A, RandomBestScore-R, and label them as BS, RBS, BS-A, BS-R, RBS-A, RBS-R respectively in the figure.

Algorithm 6 LPA

```
1: Collect all the historical state transitions $\mathcal{T} = \{(s_i, a_i, r_i, s'_i)\}$ from data; each state is composed of a time-location pair: $s_i = (t_i, location_i)$; each action is composed of the initial state and transited state: $a_i = (s_i, s'_i)$;
2: Initialize $V'(s)$, $N'(s)$ as zeros for all possible states.
3: for $t = T - 1$ to 0 do
4:     Get the subset $D^{(t)}$ where $t_i = t$ in $s_i$.
5:     for each sample $(s_i, a_i, r_i, s'_i) \in D^{(t)}$ do
6:         $N'(s_i) \leftarrow N'(s_i) + 1$
7:         $V'(s_i) \leftarrow V'(s_i) + \frac{1}{N'(s_i)}[\gamma^{\Delta t(a_i)} V'(s'_i) + R_\gamma(a_i) - V'(s_i)]$
8: Return the value function $V'(s)$ for all states.
```

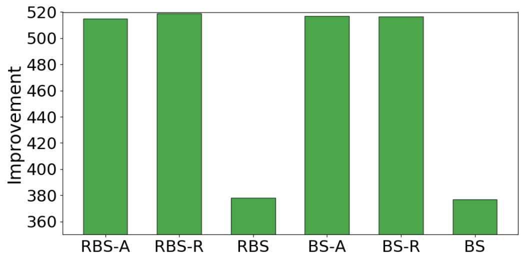


Figure 5: The Performance in Single-Vehicle Case.