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

Taoan Huang[1], **Bohui Fang**[2], **Xiaohui Bei**[3], **Fei Fang**[4]
[1]Tsinghua University, [2]Shanghai Jiaotong University,
[3]Nanyang Technological University, [4]Carnegie Mellon University

Abstract

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, leading to new challenges which, to the best of our knowledge, have not been addressed by existing works. To fill the gap, we design novel trip-vehicle dispatch algorithms to handle both types of requests while taking into account an estimated request distribution of on-demand requests. At the core of the algorithms is the newly proposed Constrained Spatio-Temporal value function (CST-function), which is polynomial-time computable and represents the expected value a vehicle could gain with the constraint that it needs to arrive at a specific location at a given time. Built upon CST-function, we design a randomized best-fit algorithm for scheduled requests and an online planning algorithm for on-demand requests given the scheduled requests as constraints. We evaluate the algorithms through extensive experiments on a real-world dataset of an online ride-hailing platform.

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. Recently, more and more 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 match drivers or vehicles (we will use drivers and vehicles interchangeably) and riders in real time upon riders' request and also allow the riders to schedule rides in advance. Companies like Curb, Shenzhou Zhuanche and ComfortDelGro and Supershuttle transform traditional taxi-hailing, chauffeured car service, and shuttle service to satisfy both types of requests [Apple Inc., 2018; ComfortDelGro Inc., 2018].

The presence of both scheduled requests and on-demand requests leads to new challenges in the task of trip-vehicle dispatch to service providers. Accepting scheduled requests is a double-edged sword: on 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 vehicle from serving more valuable on-demand requests. Some systems such as Uber simply treat 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. Such practice, however, overlooks a fundamental difference between scheduled and on-demand requests: the rider expect to be picked up on time for sure once their scheduled request is accepted, i.e., it is 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 its long-term sustainability. To the best of our knowledge, no existing work can deal with these essential challenges.

In this paper, we fill the gap and provide the first study on trip-vehicle dispatch with both scheduled and on-demand requests. We consider a two-stage decision-making process. In the first stage (Stage 1), the system is presented with a sequence of scheduled 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 needs to dispatch vehicles to the on-demand ride requests received in real-time or suggest relocations of empty vehicles, while ensuring the accepted scheduled requests in Stage 1 are satisfied. For expository purposes, we assume scheduled requests are received before all on-demand requests. However, our analysis and solution approach apply to a more general setting. While most work on trip-vehicle dispatch often ignores uncertainty in demand [Lee *et al.*, 2004; Bertsimas *et al.*, 2018; Alonso-Mora *et al.*, 2017a], recent work starts to emphasize the uncertainties and the value of data [Alonso-Mora *et al.*, 2017b; Lowalekar *et al.*, 2018; Moreira-Matias *et al.*, 2013; Tong *et al.*, 2017; Zhang *et al.*, 2017]. We also take a data-aware view and assume the platform knows the spatio-temporal distribution of the on-demand requests, which in practice can be estimated from historical data.

We propose new algorithms for both stages to handle both types of requests. In the design of these algorithms, we introduce a novel notion of Constrained Spatio-Temporal value function (CST-function). The CST-function is defined by construction with a polynomial-time algorithm we provide. We show that CST-function represents the expected value a vehicle could gain under the optimal dispatch policy with the constraint that it needs to arrive at a specific location at a given time in the future. Built upon CST-function, we design a randomized best-fit algorithm for Stage 1 to decide whether to accept requests scheduled in advance in an online fashion. We also present theoretical bounds on the competitive ratio of algorithms for Stage 1 when there are no on-demand requests to be considered in Stage 2. In addition, we build an online planning algorithm for Stage 2 to dispatch vehicles to on-demand ride requests in real time given the accepted scheduled requests as constraints. This online algorithm runs in polynomial time with guaranteed optimality for the single-vehicle case. When multiple vehicles exist, the algorithm sequentially updates CST-function to dispatch the vehicles one by one. We demonstrate the effectiveness of the algorithms through extensive experiments on a real-world dataset of an online ride-hailing platform.

2 RELATED WORK

Trip-vehicle dispatch has been studied extensively but existing work only considers real-time on-demand requests or scheduled requests. With only scheduled requests, the problem is known as the Dial-a-Ride Problem (DARP) [Cordeau and Laporte, 2007; Nedregård, 2015] and several variants of it have been studied [Cordeau, 2006; Kim, 2011; Santos and Xavier, 2015; Faye and Watel, 2016; Desaulniers *et al.*, 2016; Baldacci *et al.*, 2012; Chen and Xu, 2006]. With only on-demand requests, dispatch algorithms use different approaches, such as greedy match [Lee *et al.*, 2004; Bertsimas *et al.*, 2018], collaborative dispatch [Seow *et al.*, 2010; Zhang and Pavone, 2016; Ma *et al.*, 2013], planning and learning framework [Xu *et al.*, 2018], and receding horizon control approach [Miao *et al.*, 2016]. Our work is the first to consider both types of requests which lead to a two-stage decision-making process.

While our Stage 2 problem share similarities with the dispatch problem with on-demand requests only, the accepted scheduled requests in Stage 1 brought in hard constraints that cannot be handled easily. Simple extensions of existing algorithms [Xu *et al.*, 2018; Lowalekar *et al.*, 2018] 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 [Buchbinder and Naor, 2005] , online Steiner tree [Imase and Waxman, 1991; Awerbuch *et al.*, 2004], online bipartite graph matching [Karp *et al.*, 1990], and the online flow control packing [Garg and Young, 2002; Buchbinder and Naor, 2009]. Despite the similarities, none of those results or techniques could be directly applied to our setting due to the spatio-temporal constraints in our problem.

Other related work include efforts on last-mile transportation [Cheng *et al.*, 2014; Agussurja *et al.*, 2018; van Heeswijk *et al.*, 2017], coordinating dispatching and pricing [Chen *et al.*, 2017; Ma *et al.*, 2019; Bai *et al.*, 2018; Fiat *et al.*, 2018], with a different focus of study.

3 MODEL

We consider a discrete-time, discrete-location model with single-capacity vehicles and impatient riders. We discuss relaxation of the assumptions in Section 6.

Let $[T] = \{1, \ldots, T\}$ be the set of time steps, representing the discretized time horizon. Let $[N] = \{1, \ldots, N\}$ be the set of different locations or regions. We denote by $\delta(u, v), u, v \in [N]$ the shortest time to travel from u to v. $\mathcal{D}$ 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 $\tilde{t}_c \in [T]$ represents the earliest time that c can be dispatched and $\tilde{o}_c \in [N]$ 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 r is described by a tuple (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 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. We will discuss the relaxation of this assumption in Section 6. 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 described by a tuple (o_r, d_r, t_r, v_r) same 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 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 due to the irregularity of scheduled requests, but we assume the distribution of real-time on-demand requests is known or can be estimated from historical data. Let independent 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 APPROACHES

The system needs to deploy two algorithms for trip-vehicle dispatch, one for each stage. Critically, the scheduled requests accepted in Stage 1 will serve as constraints in Stage 2. Consider the simplest setting where there is only one vehicle. If the system has accepted a scheduled request r, then in Stage 2, when the system receives an on-demand request r' prior to the starting time of r, whether or not the system should dispatch the vehicle to serve r' depends on (i) whether the vehicle can still serve r after finishing the trip of r', (ii) how much it can gain if it serves r', (iii) how much it can gain if it does not serve r'. In hindsight, whether or not the system should accept the scheduled request r depends on the expected gain in Stage 2 with or without having r as a constraint.

Based on this intuition, in this section, we will first introduce a novel notion of *CST-function* which represents the expected gain of the only vehicle in the system in Stage 2 when it is committed to serve a scheduled request. We will then present the algorithm for Stage 2, followed by the algorithm for Stage 1, both built upon the CST-function.

4.1 CST-FUNCTION

The system's decision making problem in Stage 2 can be modeled as a Markov Decision Process (MDP) when there is only one vehicle in the system. Let c be the only vehicle in the system. 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 $t \leq \tilde{t}'_c$ 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. Given a request r, 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 $R_{t,l} = \{r' : r' \in R_t, o_{r'} = l, d_{r'} \in D(t, l|\tilde{r}^t_c), t_{r'} = t\}$ where $\tilde{r}^t_c \in R_c$ 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 state 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.

Given a vehicle at (t, l) with with r ($t_r \geq t$) being the next request it is required to serve, we define the CST-function $\mathsf{CST}(t, l|r)$ by an algorithm to compute it as shown in Algorithm 1. Instead of going through the algorithm, we will first show an important property of the CST-function.

Lemma 1. *If $\tilde{R}_c$ contains only one (real or virtual) request r, then $\mathsf{CST}(\cdot)$ computed by Algorithm 1 satisfies*

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

where π^* is the optimal policy and it follows

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

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.

The detailed proof is deferred to Appendix A. By lemma 1, $\mathsf{CST}(t, l|r)$ 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 $\mathsf{CST}(\cdot)$ 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) and d^* is the location with highest CST value if driving idly (line 6), 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 $\mathsf{CST}(t + \delta(l, a_i), a_i|r)$. The algorithm computes $\mathsf{CST}(t, l|r)$ 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).

Now we can claim that $\mathsf{CST}(t, l|r)$ represents the expected total value of trips a vehicle can serve between time t and the start time of r given (i) it is located at l and is available to serve an on-demand request at time t; (iii) it is committed to serve r in the near future; (iv) it is the only vehicle in the system. In fact, the CST-function is in concept similar to the state value function and Q-function (state-action value function) in sequential decision making [Howard, 1960], but it is specially designed for our problem with two key features. First, it considers the constraint due to Stage 1 in our problem. Second, CST-function is more compact than state value function and Q-function in our problem. Notice that the MDP of our problem has an exponential number of states as $R_{t,l}$ can take any subsets of the potential rides starting from time t at location l. Therefore, computing either state value function or Q-value function would be inefficient in both memory and computation time. In contrast, CST-values are only relevant to the vehicle's location and time and we only need polynomial-sized space to store the values and it can be computed in polynomial time.

4.2 SOLVING STAGE 2

In this section, we present our solution approach for Stage 2. When there is only one vehicle in the system, we design an algorithm DPDA (Dynamic Programming based Dispatch Algorithm) with the aid of the CST-function, 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 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 the CST-function $\mathsf{CST}(t, l|r)$. It first calculates the CST-function $\mathsf{CST}(t_a, l_a|\tilde{r}_c^t)$ 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 $\tilde{r}_c^t$ (line 5). Next, we show that Algorithm 2 induces an optimal policy for the MDP.

Theorem 1. *Algorithm 2 induces an optimal policy.*

Algorithm 1 Calculate $\mathsf{CST}(t, l|r)$

if $l = o_r$ and $t = t_r$ **then**
  return 0
for $d \in D(t, l|r)$ **do**
  Calculate $\mathsf{CST}(t + \delta(l, d), d|r)$
Denote $\{a_i\}$ the sequence of $d \in D(t, l|r)$ in decreasing order of $V_{l,d,t} + \mathsf{CST}(t + \delta(l, d), d|r)$
$d^* \leftarrow \arg\max_{d \in D(t,l)} \mathsf{CST}(t + \delta(l, d), d|r)$
$j \leftarrow$ the largest index of $\{a_i\}$ such that $V_{l,a_j,t} + \mathsf{CST}(t + \delta(l, a_j), a_j|r) > \mathsf{CST}(t + \delta(l, d^*), d^*|r)$
$p \leftarrow 1$
$F \leftarrow 0$
for $i = 1$ to j **do**
  $F \leftarrow F + p \cdot \Pr[X_{l,a_i,t} \geq 1](V_{l,a_i,t} + \mathsf{CST}(t + \delta(l, a_i), a_i|r))$
  $p \leftarrow p \cdot (1 - \Pr[X_{l,a_i,t} \geq 1])$
$F \leftarrow F + p \cdot \mathsf{CST}(t + \delta(l, d^*, d^*)|r)$
$\mathsf{CST}(t, l|r) \leftarrow F$

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

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 $\mathsf{CST}(t_a, l_a|\tilde{r}_c^t)$
5: $a^* = \arg\max_{a \in \mathcal{A}(s)} \mathcal{V}_s^a + \mathsf{CST}(t_a, l_a|\tilde{r}_c^t)$
6: **return** a^*

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 $\tilde{r}_c^t$. 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 1, solving the overall MDP is equivalent to solving the local MDP corresponding to $\tilde{r}_c^t$ (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 CST 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

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

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 the multi-vehicle case, as shown in Algorithm 3, for each vehicle we sequentially run DPDA (line 6) and update a virtual demand distribution represented by $h(X_w \geq i)$ (line 7) and recompute the CST-function (line 6) assuming $\Pr[X_w > i] = h(X_w \geq i)$, the updated virtual demand distribution, after each call of DPDA. Indeed, h serves as an approximation of the updated marginal probability $\Pr[X_w \geq i]$ after a vehicle is assigned to a ride request. Note that when a vehicle is assigned to a request, it not only changes the virtual distribution of trips starting from the current time step, but also in the future time steps because the assigned vehicle can serve future demands after it completes the current ride. So the key is in the update of h, which is done following equation (1). Intuitively, we first get p_w, the probability that a request of type w will be served by the vehicle c which is just assigned (in the last iteration) to a ride request, assuming it is the only vehicle in the system, and then update the distribution. p_w is in fact a byproduct of the computation of CST-function in the last iteration. We defer the pseudocode of the distribution update to Appendix B.

In addition, in line 2 of Algorithm 2, we do not fix the choice of vehicle sequences. In experiments, we investigate how the choice of vehicle sequences impact the outcome, specifically the variance of values gained by each vehicle, since the variance relates to the fairness of a dispatching algorithm which is a practical concern in many ride-hailing platforms with self-interested drivers.

Algorithm 3 DPDA-SU

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

4.3 SOLVING 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.

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 notion in online algorithm analysis. Given an input instance $\mathcal{I}$, we denote $\mathsf{OPT}(\mathcal{I})$ and $\mathsf{ALG}(\mathcal{I})$ as the optimal offline solution and the solution of an online algorithm on $\mathcal{I}$. We say the online algorithm is γ-competitive, if $\frac{\mathbb{E}[\mathsf{OPT}(\mathcal{I})]}{\mathbb{E}[\mathsf{ALG}(\mathcal{I})]} \leq \gamma$ 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 proportional to the trip distance. Thus, w.l.o.g, we simply set v_r equal to $\delta(o_r, d_r)$. 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 2. *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 $4\mu - 1$.*

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, proving that the bound is tight.

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 3. *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.*

The proofs of Theorem 2 and 3 are deferred to Appendix. Next, we take into consideration the Stage 2 on-demand requests, which are assumed to follow the distribution $Pr[X_w \geq i]\, \forall w \in \mathcal{W}, i \in \mathbb{N}$.

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 CST-function $\mathsf{CST}(t, l|r)$. More specifically, let r_0 and r_1 be the accepted scheduled 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

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

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

$$E_1 = \quad \mathsf{CST}(t_{r_0} + \delta(o_{r_0}, d_{r_0}), d_{r_0} \mid r) + v_r + \mathsf{CST}(t_r + \delta(o_r, d_r), d_r \mid r1)$$

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 coming 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 and $\Delta_c(r)$ could be negative. As we will see in the experiments in Section 6, variants of BESTSCORE that treat the case where $\Delta_c(r) \leq 0$ differently 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 [Karp *et al.*, 1990], we assign each vehicle a weight $\mathsf{rank}(k) = e^{\alpha k}$ 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.*

5 EXPERIMENTS

In this section, we demonstrate the effectiveness of the proposed algorithms. 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.

5.1 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 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 (details in Appendix). 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. We set $(\tilde{t'_c}, \tilde{o'_c}) = (\tilde{t_c}, \tilde{o_c})$.

5.2 EXPERIMENT SETUP

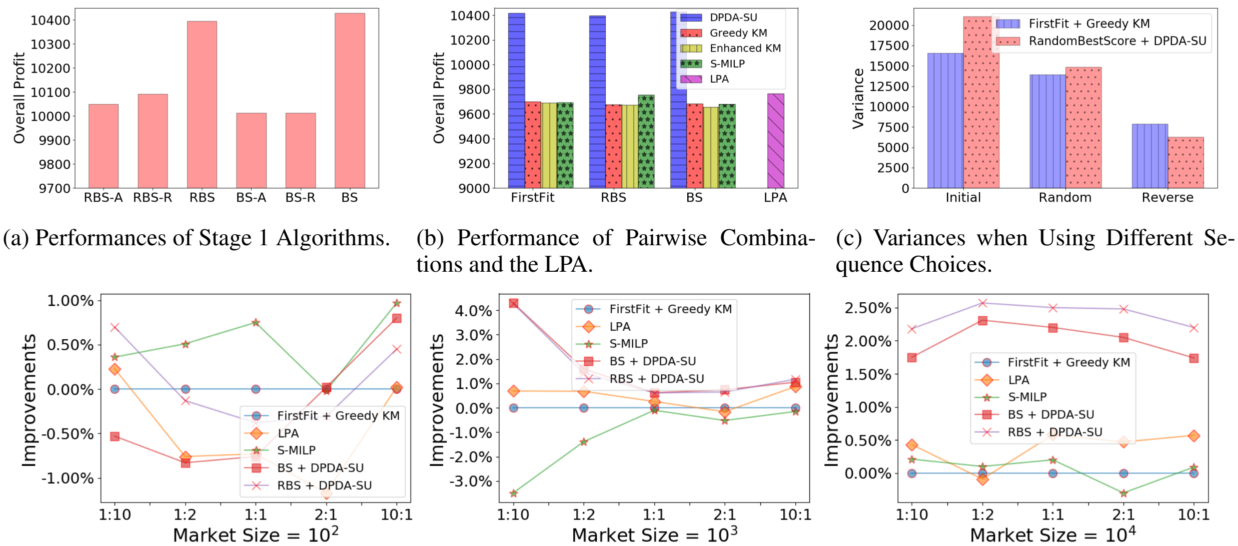
All experiments are done on an i7-6900K@3.20GHz CPU with 128GB memory. We introduce the default global setup for all the experiments. The duration of a time step is set to 1 minute and we have $24 \times 60 = 1440$ time steps in each iteration. Next, we sample on-demand requests from the historical distribution 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.

5.3 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 (Greedy-KM and Enhanced KM), a learning and planning based algorithm (LPA), and a sampling-based mixed integer linear programming (S-MILP) algorithm. Greedy-KM dispatches requests myopically considering only their values. Enhanced KM is an extension of Greedy-KM with the CST value. The LPA is adapted from [Xu *et al.*, 2018] to handle the hard constraints brought in by the scheduled requests and we implement it with slight changes of the setting in [Xu *et al.*, 2018]. In [Lowalekar *et al.*, 2018], 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 [Lowalekar *et al.*, 2018] by adding scheduled requests as constraints in the MILP. In our experiments, the number of samples is set to 10. The details of those baseline algorithms are provided in Appendix.

5.4 RESULTS

First, we combine DPDA-SU for Stage 2 with BESTSCORE (BS), RANDOMBESTSCORE (RBS) and their variants for Stage 1, to evaluate the performance of all combinations of our proposed algorithms. We 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. We also define the two variants of RANDOMBESTSCORE, RANDOMBESTSCORE-R and RANDOMBESTSCORE-A, in a similar way to the two variants of BESTSCORE. For the multi-vehicle case, the results are shown in Figure 1a. 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 I. 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 1b. We can conclude that when one of FIRSTFIT, BESTSCORE, RANDOMBESTSCORE is fixed for Stage 1, DPDA-SU always outperforms Greedy-KM,



(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 1: Experimental Results

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 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 1b. We choose the LPA and S-MILP as the other two baselines.

Figure 1d 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 $\frac{\mathbb{E}[\mathsf{OPT}(I)]}{\mathbb{E}[\mathsf{ALG}(I)]}$ 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 $\mathbb{E}[\text{RANDOMBESTSCORE}(I)]$. The offline optimal value for each instance $\mathsf{OPT}(\mathcal{I})$ can be calculated by a flow-based approach, as described in Appendix E. 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 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. 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

	DPDA-SU+BESTSCORE	LPA
Stage 1	2.2%	3.3%
Stage 2	52.3%	56.0%
Overall	50.0%	53.5%

Table 2: Reject Rates of Different Stages.

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 1c. One can see that when applying Reverse, our algorithm RANDOMBESTSCORE with DPDA-SU leads to a lower variance than FIRSTFIT combined with Greedy-KM.

We also investigate how the CST-value changes as the number of vehicles increases. We provide the details and result in Appendix J.

In real world, it could be bad service to reject the scheduled requests, so we evaluate the index of reject rate of DPDA-SU with BESTSCORE and the LPA. In Table 2, we show that in Stage 1 and overall, the reject rates of our algorithm is lower than the LPA.

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 3. 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 for each time step. We assume no scheduled requests in Stage 1. For the markets with 1000, 3000, 5000 idle vehicles, we test the running time respectively, and the results are summarized in Table 3. 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. We then increase the number of regions to 200 and obtain the centers of each region using the same clustering algorithm.

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

Table 3: Running Time (in seconds) with Different Number of Vehicles and Space Complexities of Different Algorithms.

Following the request distribution, we generate 1128 on-demand requests at each time step within the rush hour (9PM - 10PM). In this case, our algorithms can compute the results for each time step within 0.35 seconds using 115.2GB of memory.

6 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.

The model can be applied to or further extended for problems with relaxed assumptions. First, our work can be applied to problems with patient requests, which can be treated as duplicated requests when there is only one driver. Second, our framework can be extended to the case where each scheduled request becomes available at least μ time before departure, where μ is the longest possible trip time. In this case, at each time step, we first deal with the newly-received scheduled requests before processing the on-demand requests and computing the CST-function. Third, our algorithms can also deal with uncertainties in travel time, i.e., $\delta(u, v)$'s are not the same at different time steps. We could handle these uncertainties by replacing $\delta(u, v)$ with $\delta(u, v, t)$ in the algorithms, where $\delta(u, v, t)$ is the shortest time to travel from u to v that depends on t. For further investigation, our work can be integrated with work on last-mile routing to handle actual road networks.

Acknowledgement

Fei Fang is funded in part by Carnegie Mellon Universitys Mobility21, a National University Transportation Center for Mobility sponsored by the US Department of Transportation. The contents of this report reflect the views of the authors only.

References

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

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

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

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

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

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

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

[Bertsimas *et al.*, 2018] Dimitris Bertsimas, Patrick Jaillet, and Sébastien Martin. Online vehicle routing: The edge of optimization in large-scale applications. 2018.

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

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

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

[Chen *et al.*, 2017] Mengjing Chen, Weiran Shen, Pingzhong Tang, and Song Zuo. Optimal vehicle dispatching schemes via dynamic pricing. *arXiv preprint arXiv:1707.01625*, 2017.

[Cheng *et al.*, 2014] Shih-Fen Cheng, Duc Thien Nguyen, and Hoong Chuin Lau. 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*, pages 1505–1506. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[ComfortDelGro Inc., 2018] ComfortDelGro Inc. Corporate Profile - ComfortDelGro. `https://www.comfortdelgro.com/web/guest/corporate-profile`, 2018.

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

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

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

[Faye and Watel, 2016] Alain Faye and Dimitri Watel. Static dial-a-ride problem with money as an incentive: Study of the cost constraint. 2016.

[Fiat *et al.*, 2018] Amos Fiat, Yishay Mansour, and Lior Shultz. Flow equilibria via online surge pricing. *arXiv preprint arXiv:1804.09672*, 2018.

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

[Howard, 1960] Ronald A Howard. Dynamic programming and markov processes. 1960.

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

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

[Kim, 2011] Taehyeong Kim. *Model and algorithm for solving real time dial-a-ride problem*. PhD thesis, 2011.

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

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

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

[Ma *et al.*, 2019] Hongyao Ma, Fei Fang, and David C. Parkes. Spatio-temporal pricing for ridesharing platforms. In *Proceedings of the 2019 ACM Conference on Economics and Computation*, EC '19. ACM, 2019.

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

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

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

[Nedregård, 2015] Ida Nedregård. The integrated dial-a-ride problem-balancing costs and convenience. Master's thesis, NTNU, 2015.

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

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

[Tong *et al.*, 2017] Yongxin Tong, Yuqiang Chen, Zimu Zhou, Lei Chen, Jie Wang, Qiang Yang, Jieping Ye, and Weifeng Lv. 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*, pages 1653–1662. ACM, 2017.

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

[Xu *et al.*, 2018] Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, and Jieping Ye. 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*, pages 905–913. ACM, 2018.

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

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