A Value-based Dynamic Learning Approach for Vehicle Dispatch in Ride-Sharing

Cheng Li[1,2], David Parker[1] and Qi Hao[2,3]

***Abstract*— To ensure real-time response to passengers, existing solutions to the vehicle dispatch problem typically optimize dispatch policies using small batch windows and ignore the spatial-temporal dynamics over the long-term horizon. In this paper, we focus on improving the long-term performance of ride-sharing services and propose a deep reinforcement learning based approach for the ride-sharing dispatch problem. In particular, this work includes: (1) an offline policy evaluation (OPE) based method to learn a value function that indicates the expected reward of a vehicle reaching a particular state; (2) an online learning procedure to update the offline trained value function to capture the real-time dynamics during the operation; (3) an efficient online dispatch method that optimizes the matching policy by considering both past and future influences. Extensive simulations are conducted based on New York City taxi data, and show that the proposed solution further increases the service rate compared to the state-of-the-art far-sighted ride-sharing dispatch approach.**

I. Introduction

In ride-sharing, a key goal is to assign and route vehicles to serve as many requests as possible. The most common approach is batch assignment, which uses dispatch windows to allocate multiple requests at the same time. It is well suited for real-world ride-sharing production systems and enables a "global" optimization for each dispatch window [1], [2]. However, because of the lack of future information in the subsequent batch windows, an assignment decision that is "optimal" now may cause potential efficiency losses in the future and leaves room for further optimizations [3], [4]. Fig. 1 shows an example that, instead of choosing the red schedule with the maximum immediate reward, the best policy is to assign the blue one to the vehicle now and adjust the schedule according to the appearance of newly revealed requests to get maximum total reward.

Meanwhile, the rapid development of online ride hailing services (e.g., Uber, Lyft, DiDi and Grab) provides rich information on urban mobility patterns, which has led to research on the far-sighted vehicle dispatch problem [5]–[7]. To leverage transit information to learn urban mobility patterns and improve transportation efficiency, the following technical challenges need to be addressed:

*This work is partially supported by the Shenzhen Fundamental Research Program (No:JCYJ20200109141622964) and the Intel ICRI-IACV Research Fund (CG#52514373).

[1]School of Computer Science, University of Birmingham, Birmingham, UK {cxl1776, d.a.parker}@cs.bham.ac.uk

[2]Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, 518055, China haoq@sustech.edu.cn (Corresponding author: Qi Hao)

[3]Research Institute of Trustworthy Autonomous System, Southern University of Science and Technology, Shenzhen, China

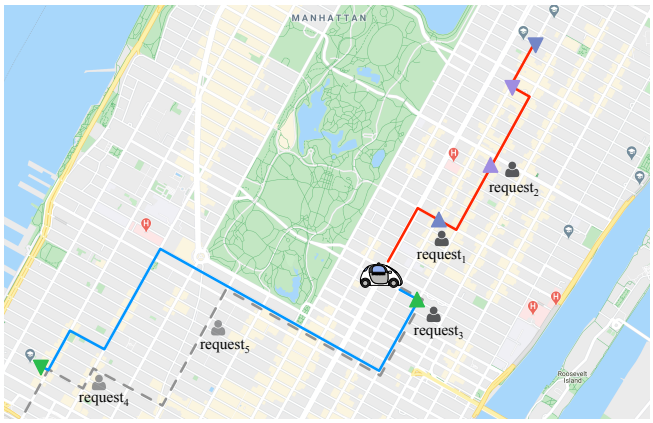

Fig. 1: An example with one vehicle, three revealed requests and two requests that will arrive in the future but are not known at present. Triangles (△) and inverted triangles (▽) represent the origins and destinations, respectively. When myopically optimizing over the current epoch, the vehicle will go towards the top right of the map to serve requests 1 and 2 and earn a reward of 2. However, if it chooses to serve request 3 and instead travels towards the bottom left of the map, it will eventually earn a reward of 3.

1) *Learning Efficiency.* Applying extensive learning exploration on real world deployments is inefficient and may lead to unintended behavior. Operating in simulations requires an accurate simulator that is typically hard to build and may slow down the learning process due to simulation computations.
2) *Real-Time Fluctuations.* Besides the systematic patterns of mobility (e.g., people are more likely to travel to residential areas during the evening peak hours), the non-stationary dynamics (e.g., people may go to different restaurants on different days), which cannot be easily learned from historical data, also needs to be captured.
3) *Value Deviation.* The learned mobility pattern is an approximation of the value of real-world dynamics. If the dispatch policy is solely based on learned values, there will be deviations that need to be corrected.

Most existing work on the far-sighted vehicle dispatch problem does not allow ride-sharing [5]–[7]. Some work studies double-occupancy fleets, but they formulate assignment decisions with no more than five candidate options [8], [9], which cannot be directly extended to higher occupancy ride-sharing and therefore cannot fully take advantages of ride-sharing services.

In this paper, we study the long-term optimization of ride-sharing and present a learning-based approach that is flexible

in adjusting the dispatch policy based on real-time dynamics to maximize the objective. The proposed approach models the vehicle dispatch problem as a Markov Decision Process (MDP), combines offline evaluation and online learning to obtain a value function that captures the spatial-temporal dynamics of ride-sharing trips, and employs a re-optimization strategy to mitigate the deviation between the learned value and the actual value. To summarize our contributions:

1) We develop an offline policy evaluation (OPE) based method to capture systematic mobility patterns, using a neural network to learn a value function from historical vehicle movement data.
2) We adopt an online learning procedure to continuously update the offline learned value function to handle non-stationary dynamics, using only the states of vehicles in the current operating epoch.
3) We combine the learned value function with the re-optimization dispatch method in [2] to optimize the objective over a long horizon, including both recent and upcoming epochs.
4) We conduct simulation experiments on real world large-scale taxi datasets to evaluate our proposed approach and analyze the impact of long-term optimization.

II. Related Work

Ride-sharing holds great promise for improving urban mobility efficiency and is becoming an emerging academic research problem. A study in New York City demonstrates that, using pairwise ride-sharing, up to 80% of taxi rides and 40% of total travel time can be saved [10]. To enable high occupancy ride-sharing, a scalable batch assignment algorithm is developed in [1] to utilize the full capacities of vehicles. The efficiency of the fleet is further improved in [2] by exploring all possible ride-sharing combinations in real time and dynamically altering the assignment policy to keep it "optimal" at any given time. However, to provide a good passenger experience, the above approaches involve using small batch periods and are myopic in nature.

There has been some research on leveraging knowledge about the future to dispatch vehicles in a more far-sighted way. They can be categorized into two groups: forecast-based dispatch and value-based dispatch. The former usually partitions the road network into zones and estimates the future demand distribution over the zones at each time step. The predicted travel demand is then used to generate a set of artificial requests to compute sophisticated matching and scheduling policies [11]–[13]. However, considering future requests in the computing process yields higher computational complexity that requires tighter heuristics to solve the problem, which would in turn reverse the improvement achieved by lookahead. Moreover, results in [13] show that, when properly trained, value-based dispatch obtains higher service rates than forecast-based dispatch.

Value-based dispatch learns a value function, which evaluates the potential long-term return of assignments, to assist in finding the best matching policy. It does not affect the computational complexity of online matching and scales well for large fleets. Most of the literature on value based dispatch has focused on unit capacity vehicle fleets. They model the movement of each vehicle as a semi-Markov decision process and adopt the deep reinforcement learning framework to estimate the state-action value function of the vehicle [5]–[7]. Only a few studies consider pairwise ride-sharing [8], [9], but the action space is not granular enough for high occupancy ride-sharing.

A similar work to this paper is presented in [4], which allows a vehicle to be shared by more than two requests. However, the simulator-reliant training process is inefficient and takes around one week to learn a value function [13]. Moreover, solely using a value function trained on historical data may not handle dynamic environments well.

III. Problem Statement

A. Definitions

We assume a central dispatcher computes vehicle-request matches in a rolling horizon framework, with time window length ΔT. It operates a fleet of m vehicles. At each planning epoch, it receives a set of n new requests and allocates them to vehicles to maximize the service rate, i.e., percentage of requests served. A trip Γ denotes that a group of requests is merged with ride-sharing. The order in which a vehicle picks up and drops off requests in a trip is called a schedule $l = \{q, \ldots, o_1, \ldots, d_1, \ldots, o_2, \ldots, d_{n_\Gamma}\}$, where q is the vehicle's current location, o_i and d_i are the origins and destinations of requests. A schedule must satisfy two constraints: (1) the number of requests on board cannot exceed the capacity κ of the vehicle, (2) the waiting time (the difference between when it is on board and when it is submitted) and travel delay (the difference between when it arrives via ride-sharing and when it will arrive if travelling alone) of each request cannot exceed thresholds Ω and Λ, respectively.

B. An MDP Formulation

We model the vehicle dispatch problem as an MDP, with an agent representing an individual vehicle. In this framework, a vehicle interacts with an environment over a sequence of discrete time steps $t \in \{0, 1, \ldots, n_{end}\}$, where n_{end} is the terminal time, e.g., end of an operating hour or day. At each time step t, the vehicle receives a state s_t, based on which it selects an action a_t. It then receives a scalar reward r_{t+1} and transitions to s_{t+1}, according to environmental dynamics. The goal of a vehicle is to maximize the expected discounted return $G_t = \sum_{k=t+1}^{n_{end}} \gamma^{k-t-1} \cdot r_k$, where $0 < \gamma \leq 1$ is the discount rate. The main components of the MDP are as follows:

State. The state of a vehicle consists of the spatial status l_t, the associated temporal status ζ_t, the real world time stamp u_t and the exogenous information w_t. Formally, it is defined as $s_t = (l_t, \zeta_t, u_t, w_t)$. The temporal status ζ_t represents the remaining delay time when the vehicle visits each location in schedule l_t, reflecting how much further deviation from the current path the vehicle can take to pick up a new passenger. For example, assuming a vehicle is travelling to location d_1 to drop off a passenger, the deadline to arrive is 400 sec

and the travel time is 200 sec, then the vehicle has 200 sec (i.e., the remaining delay time) to make a detour to pick up a new passenger. Passengers that require more than 200 sec to pick up will not be feasible for the vehicle. The time stamp u_t indicates the time scale across the entire operating horizon of the ride-sharing system and is independent of the algorithmic time t. The exogenous information w_t consists of two scalars: the number of new requests at the current dispatch epoch and the number of nearby vehicles at the location $q = l_t[0]$.

Action. The eligible actions for a vehicle include all possible candidate schedules $F_{i,t} = \{l_1, l_2, \dots\}$ (i stands for the vehicle id), where each one indicates a candidate vehicle-trip match that assigns the requests included in the trip to the vehicle. Formally, an action defined as $a_t = l_{t+1}$. Note that a vehicle has $|F_{i,t}|$ actions. There are basically two types of actions. The first type is to add (or remove) one or more requests to (from) the vehicle and replace its schedule with the action schedule. The other type is "remain the same" which leaves the vehicle to follow its current schedule. Executing action a_t means replacing the schedule of the vehicle with l'_{t+1} and transitioning to l_{t+1}, where l'_{t+1} is the status of l_{t+1} at time t.

Reward function. The reward is the number of requests added to the vehicle when it takes an action a_t. Formally, it is defined as $R(s_t, a_t) = (|l'_{t+1}| - |l_t|)/2$. The output value of $R(s_t, a_t)$ is denoted by r_{t+1}. The "remain the same" action produces a reward of 0 (i.e., $r_{t+1} = 0$). Note that r_{t+1} could be negative if requests that are assigned to the vehicle at $t-1$ are re-assigned to other vehicles at t.

State transition. When a vehicle is in state s_t and takes an action a_t, its transition to state s_{t+1} is denoted by $s_{t+1} = f(s_t, a_t, w_{t+1})$. However, in practice, the decisions have to be made at t before the realization of the exogenous information w_{t+1} (arriving between t and $t+1$), so as not to affect the passenger experience. Therefore, a post-decision state s^a_t [14] (also called afterstate [15]) is introduced to split the transition function into two components: $s^a_t = f^{(1)}(s_t, a_t)$ and $s_{t+1} = f^{(2)}(s^a_t, w_{t+1})$. The post-decision state is a segregation of deterministic and stochastic information about the future and explicitly captures the most recent state.

A common way to solve an MDP is to find a value function that measures potential future rewards to guide the interaction of the agent. As the full dynamics of a ride-sharing system are not known to the dispatcher when deciding the matching between vehicles and requests at time step t, we choose to learn a post-state value function that incorporates the most recent information we have. This would produce a more efficient method than learning an action-value function [15], [16].

C. Method Overview

Fig. 2 illustrates the proposed framework, which optimizes the objective of ride-sharing over a long horizon while maintaining an efficient online matching procedure that yields a short response time. It has three components:

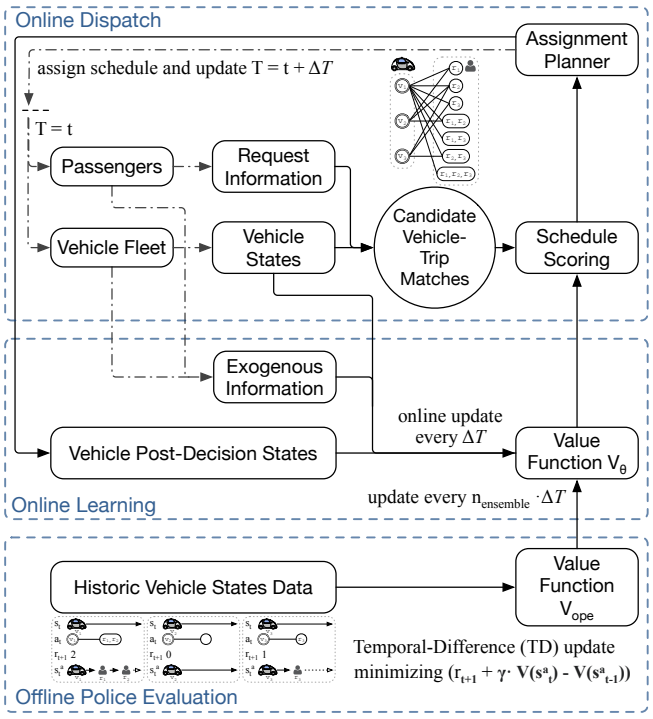

Fig. 2: Framework of the long-term optimization logic.

OPE. Given a set of historical state transitions collected from vehicles, denoted by $\{(s^a_{i,t-1}, r_{i,t+1}, s^a_{i,t})\}$ (i stands for vehicle id), the dispatcher performs a policy evaluation with a neural network to learn a post-decision state value function $V_{ope}(s^a)$, by applying the Bellman squared error and Temporal Difference (TD) update.

Online Dispatch. The dispatcher uses the candidate generation method in [2] to compute actions for all vehicles. It then employs the learned value function to match vehicles and requests in a far-sighted way, by scoring candidate matches (i.e., candidate schedules) with the sum of instant rewards and long-term expectations. The dispatcher also considers re-optimization by adjusting the matching between vehicles and previously received requests with the most recent knowledge about the system, so as to extend the matching horizon for a more "global" optimization.

Online Learning. During the online dispatch procedure, a new set of state transitions is generated at each epoch, which reflects the real-time dynamics of the current operational state. The dispatch leverages these real-time transitions to continuously update the value function, so that it can quickly adapt to any fluctuations in the system.

IV. Value-based Vehicle Dispatch Scheme

We assume the dispatcher is equipped with a matching policy that maximizes the objective function at each dispatch epoch and remains unchanged over the training period. The value function is trained for an individual vehicle, under the assumption that all vehicles are homogeneous and have a common goal of maximizing the global service rate. Learning a shared value function, not a separate network for

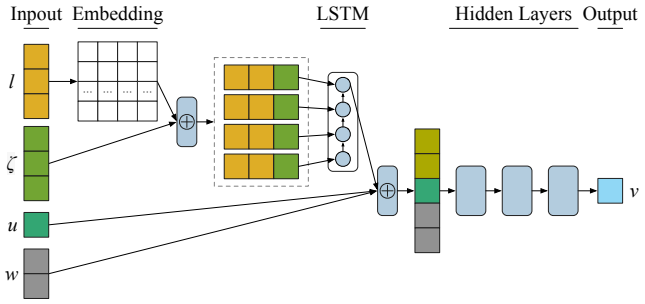

Fig. 3: Structure of the value network.

each vehicle, utilizes the historical data in a more efficient way and requires a simpler training pipeline [5], [17].

A. Offline Policy Evaluation With Neural Networks

As we do not distinguish individual vehicles, a complete historical transition tuple can be presented as $(s^a_{t-1}, w_t, s_t, a_t, r_{t+1}, s^a_t, w_{t+1}, s_{t+1})$. If $t = 0$, we set $s^a_{t-1} = s_0$. Let $V(s)$ denote the state-value function of a vehicle, the Bellman equation can be written as:

$$\begin{aligned} V(s_t) &= \mathbb{E}\{r_{t+1} + \gamma \cdot G_{t+1} | s = s_t\} \\ &= \mathbb{E}\{r_{t+1} + \gamma \cdot V(s_{t+1})\} \end{aligned} \tag{1}$$

Considering the transition from s_t to s_{t+1} and applying the TD method [15], the state-value function is updated as:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma \cdot V(s_{t+1}) - V(s_t)] \tag{2}$$

where $\alpha > 0$ is the learning rate, s_t is the current state and s_{t+1} is the next state of the vehicle following the schedule indicated by a_t. Using the post-decision state and letting $V_{ope}(s^a)$ denote the post-state value function learned by OPE, Eq. (1) is decomposed into two steps:

$$V(s_t) = \mathbb{E}\{r_{t+1} + \gamma \cdot V_{ope}(s^a_t)\} \tag{3}$$

$$V_{ope}(s^a_t) = \mathbb{E}\{V(s_{t+1}) | s^a_t\} \tag{4}$$

The advantage of this decomposition is that the right hand side of Eq. (3) is deterministic, which makes it easier to use deterministic algorithms [18]. Similar to Eq. (2), the post-decision state value function is updated as:

$$\begin{aligned} V_{ope}(s^a_{t-1}) \leftarrow & V_{ope}(s^a_t) \\ & + \alpha[r_{t+1} + \gamma \cdot V_{ope}(s^a_t) - V_{ope}(s^a_{t-1})] \end{aligned} \tag{5}$$

which is similar to Eq. (2), except that the value function is updated around the previous post-decision state.

In ride-sharing systems, both the state space and action space grow exponentially, especially with regard to the number of state variables. To overcome the curse of dimensionality, a neural network is used to approximate the post-decision state value function, which is inspired by [4] and represented in Fig. 3. The network takes a post-decision state $s^a_t = f^{(1)}(s_t, a_t) = (l_{t+1}, \zeta_{t+1}, u_{t+1}, w_t)$ as input and outputs a scalar as the estimation of the value. It uses an embedding layer to learn the geographical feature, a Long Short-Term Memory (LSTM) layer to learn the sequence-level temporal feature, and a hidden layer to combine the spatial-temporal features with the exogenous information.

Algorithm 1 OPE with Neural Network

Input : A prioritized replay buffer filled with historical state transitions $D_{offline} = \{(s^a_{t-1}, r_{t+1}, s^a_t)\}$, the number of iterations n_{iter}, the discount factor γ and the target smoothing factor ρ.

Output: The post-decision state value function $V_{ope}(s^a)$.

1: $V_{ope}, \theta_{ope} \leftarrow InitializeValueNetwork()$;
2: $\hat{V}_{ope}, \hat{\theta}_{ope} \leftarrow InitializeValueNetwork()$;
3: **for** $i \leftarrow 1$ to n_{iter} **do**
4: $\quad B \leftarrow SampleMiniBatch(D_{offline})$;
5: $\quad \mathcal{L} = 1/|B| \cdot \sum_{j=1}^{|B|}(r_{j,t+1} + \gamma \cdot \hat{V}_{ope}(s^a_{j,t}) - V_{ope}(s^a_{j,t-1}))^2$;
6: $\quad \theta_{ope} \leftarrow GradientDescent(\mathcal{L}, \theta_{ope})$;
7: $\quad \hat{\theta}_{ope} \leftarrow \rho \cdot \theta_{ope} + (1-\rho) \cdot \hat{\theta}_{ope}$;

We collect a set of historical transition tuples and store them in a replay buffer $D_{offline}$. Each tuple $(s^a_{t-1}, r_{t+1}, s^a_t)$ represents that a vehicle transitions from s^a_{t-1} to s^a_t within one time step and receives a reward of r_{t+1}. We use Double Q-Learning [19] and TD error to train the network $V_{ope}(s^a)$ offline. The main procedure is described in Algorithm 1. It uses prioritized experience replay [20] to make the training more efficient by sampling important transitions more frequently, where the importance of a transition is measured by the magnitude of its TD error. Function $InitializeValueNetwork$ returns a network using the structure in Fig. 3 with random weights, where θ_{ope} is the weights of $V_{ope}(s^a)$, and $\hat{V}_{ope}(s^a_t)$ is the target network that is designed to reduce overestimation [21]. Function $SampleMiniBatch$ returns a small set of transitions using prioritized sampling, which is then used to compute the mean-squared loss $\mathcal{L}$ and perform a gradient descent step to update the network weights θ_{ope} (line 6). In the last line of the algorithm, a soft update [22] is applied to the target network weights $\hat{\theta}_{ope}$, which tries to improve the stability of learning by changing the target network very slowly.

The resultant value function $V_{ope}(s^a)$ captures the general systematic patterns of the supply-demand conditions from historical data, and will serve as the basis for the online value function during operation (i.e., online dispatching).

B. Online Learning With Value Ensemble

To capture the non-stationary dynamics in real time, we present a new value network $V_{ol}(s^a)$ that uses the transitions of vehicles in the current dispatching epoch. The main difference between $V_{ol}(s^a)$ and $V_{ope}(s^a)$ is that the online value function $V_{ol}(s^a)$ cares more about the spatial dynamics of the supply-demand conditions, and fixes time to t_{net} to utilize the sparse online transition data more efficiently.

Considering that at time step t, after the matching policy has been made, we will have an online state transition $(s_{i,t}, a_{i,t}, r_{i,t+1}, s^a_{i,t})$ for each vehicle $i \in [1, 2, \ldots, m]$. We allow the dispatcher to make a backup of the transitions at time step $t-1$, and use it to generate an online buffer $D_{online} = \{(s^a_{i,t-1}, r_{i,t+1}, s^a_{i,t})\}$. Note that $|D_{online}| = m$.

Algorithm 2 Online Learning With Value Ensemble

Input : The offline learned value network V_{ope} with its weights θ_{ope}, the discount factor γ, the value ensemble step size $n_{ensemble}$, the value ensemble factor ψ and the target smoothing factor ρ.

1: $V_{ol}, \theta_{ol} \leftarrow InitializeValueNetwork()$;
2: $\hat{V}_{ol}, \hat{\theta}_{ol} \leftarrow InitializeValueNetwork()$;
3: $t_{net} \leftarrow 0$;
4: **for** $t \leftarrow 0$ to n_{end} **do**
5: **if** t mod $n_{ensemble} = 0$ **then**
6: $\theta_{ol} \leftarrow \psi \cdot \theta_{ope} + (1-\psi) \cdot \theta_{ol}$;
7: $t_{net} \leftarrow t$;
8: $D_{online} \leftarrow SolveMatchingProblem(V_{ol})$;
9: $\mathcal{L} = 1/m \cdot \sum_{i=1}^{m} (r_{i,t+1} + \gamma \cdot \hat{V}_{ol}(\Psi(s^a_{i,t}, t_{net})) - V_{ol}(\Psi(s^a_{i,t-1}, t_{net})))^2$;
10: $\theta_{ol} \leftarrow GradientDescent(\mathcal{L}, \theta_{ol})$;
11: $\hat{\theta}_{ol} \leftarrow \rho \cdot \theta_{ol} + (1-\rho) \cdot \hat{\theta}_{ol}$;

The one-step TD update for each vehicle's transition is slightly different from Eq. (5), given by:

$$V_{ol}(\Psi(s^a_{i,t-1}, t_{net})) \leftarrow V_{ol}(\Psi(s^a_{i,t}, t_{net})) + \alpha[r_{t+1} + \gamma \cdot V_{ol}(\Psi(s^a_{i,t}, t_{net})) - V_{ol}(\Psi(s^a_{i,t-1}, t_{net}))] \quad (6)$$

where a new function $\Psi(\cdot)$ is introduced to fix the time of the states. As an example, for $s^a_{t-1} = (l_t, \zeta_t, u_t, w_{t-1})$ and $s^a_t = (l_{t+1}, \zeta_{t+1}, u_{t+1}, w_t)$, we have $\Psi(s^a_{t-1}, t_{net}) = (l_t, \zeta_t, u_{t_{net}}, w_{t-1})$ and $\Psi(s^a_t, t_{net}) = (l_{t+1}, \zeta_{t+1}, u_{t_{net}}, w_t)$.

The main procedure for online training is presented in Algorithm 2, which is a joint procedure with online dispatching. The algorithm initializes $V_{ol}(s^a)$ with random weights θ_{ol} at the beginning of operations and updates it in two processes: (1) periodically update its weights using a weighted ensemble with a snapshot of the offline learned network $V_{ope}(s^a)$ (line 6); (2) update its weights using real-time transitions in D_{online} at each dispatch epoch (line 8-10). The ensemble factor $\psi > 0$ is a hyperparameter similar to the target smoothing factor ρ. It determines how much importance we place on historical and immediate experience, by adjusting the weighting between the offline trained network $V_{ope}(s^a)$ and the online learned network $V_{ol}(s^a)$ [7].

The resultant online value function $V_{ol}(s^a)$ is actually an augmentation of $V_{ope}(s^a)$, by continuously updating the value function using the most recent information to capture the real-time variations of the moment. It relies on the ensemble with $V_{ope}(s^a)$, as only partial vehicle trajectories are available for online training, which makes it hard to learn a global value function. Also, solely using online learning suffers from sample-inefficiency.

C. Re-Optimization and Value-Based Allocating

Function $SolveMatchingProblem$ in Algorithm 2 solves a maximum matching problem. Considering a fleet of m vehicles, a set of previously matched yet not served requests REQ_{prev}, a set of newly submitted requests REQ_{new}, and a set of candidate action pools $\{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_i\}$ that have been computed for each vehicle, it is formulated as an integer linear program:

$$\underset{x_{i,j}, \epsilon_k}{\text{argmin}} \quad \sum_{i=1}^{m} \sum_{j=1}^{|\mathcal{F}_i|} x_{i,j} \cdot Q(s_i, a_j) \quad (7)$$

$$s.t. \quad \sum_{j=1}^{|\mathcal{F}_i|} x_{i,j} = 1, \quad \forall i \in [1, 2, \ldots, m] \quad (8)$$

$$\sum_{i=1}^{m} \sum_{j=1}^{|\mathcal{F}_i|} x_{i,j} \cdot \Theta_{a_j}(k) + \epsilon_k = 1, \quad \forall k \in REQ_{all} \quad (9)$$

$$\epsilon_k = 0, \quad \forall k \in REQ_{prev} \quad (10)$$

where $Q(s_i, a_j) = R(s_i, a_j) + \gamma \cdot V_{ol}(s^a_i)$, $\Theta_{a_j}(k)$ is an function indicating whether action a_j serves k ($\Theta_{a_j}(k) = 1$) or not ($\Theta_{a_j}(k) = 0$), and $REQ_{all} = REQ_{prev} \cup REQ_{new}$.

Binary variable $x_{i,j} \in \{0, 1\}$ indicates whether an action a_j (i.e., a candidate schedule) is selected ($x_{i,j} = 1$) or not ($x_{i,j} = 0$) for vehicle i. Binary variable ϵ_k indicates whether a request k is matched ($\epsilon_k = 0$) or not ($\epsilon_k = 1$). Constraint (8) ensures that each vehicle selects exactly one action, constraint (9) ensures that each request is matched to at most one vehicle. While assigning vehicles to new incoming requests, the dispatcher also considers adjusting the matches of previous requests (i.e., re-optimization) based on the latest knowledge of the supply-demand conditions (including both revealed and estimated). At the same time, constraint (10) ensures the passenger experience, by guaranteeing that no previously matched requests will be rejected even if rejecting them yields a higher objective value.

The re-optimization procedure adjusts the previous matching, both by considering new requests and by updating the weights of actions with the latest value function $V_{ol}(s^a)$. When producing the matching policy at time step t, each candidate vehicle-trip match (i.e., the action) is scored with the expected future reward (i.e., $R(s_t, a_t) + \gamma \cdot V_{ol}(s^a_t)$) that might have potential prediction bias. Then, at time step $t+1$, while using TD error to update the value function, the dispatcher also implicitly uses a Bellman-style update of the form $R(s_t, a_t) + R(s_{t+1}, a_{t+1}) + \gamma \cdot V_{ol}(s^a_{t+1})$, to adjust the score of each candidate match, to alter the matching policy.

V. Experimental Study

We evaluate the proposed far-sighted dispatcher using taxi trip data in Manhattan [23] and compare it to the state-of-the-art. All algorithms are implemented and run on the same machine for a fair comparison.

A. Simulation Details

The experiments are conducted using data from two days: 25th and 26th May 2016. The value network $V_{ope}(s^a)$ is trained using data from nine days (3rd-5th, 10th-12th and 17th-19th May 2016) and validated on 24th May 2016. The selected days are Tuesday, Wednesday and Thursday and follow a similar request pattern [7]. Simulations are run for the hour with peak demand (19:00-20:00), where the average

TABLE I: Parameter settings (defaults in bold).

Parameters	Settings
Vehicle Capacity κ	2, 4, **6**, 8
Fleet Size m	1200, **1500**, 1800
Maximum Waiting Time Ω (sec)	180, **300**, 420
Batch Period ΔT (sec)	10, **30**, 60, 120

number of requests is 18,789. The road map and travel times of Manhattan are produced using the method proposed in [10]. The main experimental parameters are summarized in Table I, where we vary the supply and demand conditions to evaluate the algorithms. Besides, the maximum travel delay is set to be twice the maximum wait time $\Lambda = 2\Omega$. The hyperparameters of the value network are tuned by running the validation simulation and then fixed when running the evaluation simulation. Specifically, we set the discount factor to $\gamma = 0.95$, the learning rate to $\alpha = 0.01$, the target smoothing factor to $\rho = 0.05$, the value ensemble step size to $n_{ensemble} = 600\ sec/\Delta T$ and the value ensemble factor to $\psi = 0.9$. When running the simulation, each request will be considered by the dispatcher for up to five epochs before rejecting it for not finding a feasible match. The results are averaged over five experiment runs.

B. Algorithm Comparison

We examine and compare the service rates of the following algorithms. The key difference between them is how many batch windows are optimized, as shown in Fig. 4.

- LTO (Long-Term Optimization): The method proposed in this paper that optimizes the objective for all available requests over a long horizon.
- LTO-Basic: A method that emulates the one presented in [4], which is the state-of-the-art far-sighted ride-sharing dispatch approach. It only optimizes the objective for newly received requests, with a value function estimating the expected reward in the future. It is obtained by removing the online learning and re-optimization components of LTO.
- OSP (Optimal Schedule Pool): The method from [2] that maximizes the objective for all available requests, but does not have any knowledge about the future.
- Baseline: A method that only optimizes the current dispatch epoch by maximizing the objective for newly received requests. It is a simplified version of OSP that does not allow re-optimization.

C. Results

The average service rates for the 25th and 26th May, under different parameters, are shown in Fig. 5. In general, by considering a longer planning horizon and real-time dynamics, LTO always achieves the highest service rates.

Vehicle Capacity. With higher vehicle capacity, the ride-sharing system has more seats and thereby serves more requests. But the new increased seats are bundled with existing trips and require the dispatcher to capture a more complex mobility pattern when the capacity is larger than 4. The performance of OSP is robust as the re-optimization strategy

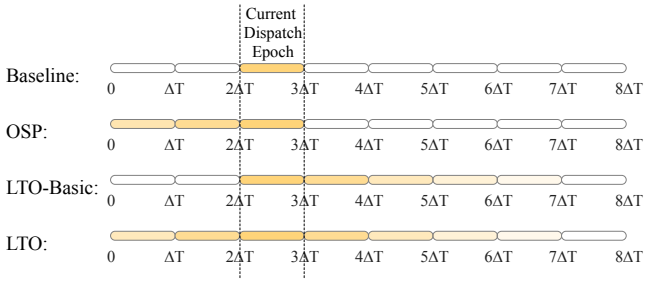

Fig. 4: A schematic comparison of how many dispatch epochs are considered for optimization by different algorithms. The epochs considered are marked in light orange, and the intensity of the color indicates how much the information within that epoch contributes to the optimization.

always explicitly uses all the precisely revealed information. But value-based approaches suffer from a decrease in the accuracy of estimation that comes with the increase in the complexity of the supply-demand conditions. When the capacity increases from 4 to 8, LTO's improvement relative to Baseline drops by 0.42%, from 7.33% to 6.91%. However, the drop for LTO-Basic is 1.01%, from 3.48% to 2.47%. This indicates that, by employing online learning and re-optimization, LTO can learn a better value function and correct for prediction bias.

Fleet Size. When the fleet size increases, the system has more independent seats, which gives more flexibility to the dispatcher and reduces the dispatcher's reliance on sophisticated algorithms. As the number of vehicles increases from 1200 to 1800, the lift in LTO-Basic relative to Baseline decreases from 3.62% to 0.48%, which is consistent with the results in [4]. The reason for this may be that a far-sighted dispatcher will tend to ignore immediate rewards and save empty seats for future orders. When the number of vehicles is sufficient ($m = 1800$), most of the future requests are quickly picked up by vehicles and there are not many left. It is also possible that, because the value network is trained on 1500 vehicles, when the number of vehicles increases, which actually increases the supply and decreases the average reward of vehicles, the value becomes overestimated and inaccurate. Using online learning to update the value network, the lifts in LTO relative to OSP are 4.04% and 1.26% when using 1200 and 1800 vehicles, respectively, indicating that LTO learns a better value function than LTO-Basic.

Maximum Waiting Time. The increase in travel delay constraint allows for more detours and therefore makes it easier to carry more passengers. Thus, all algorithms achieve higher service rates as the delay tolerance increases. When the constraint increases from 180 sec to 420 sec, the improvement yielded by LTO-Basic relative to Baseline increases from 2.65% to 3.17%. Enhanced by online learning and re-optimization, LTO further improves on the basis of LTO-Basic by 2.03%, 4.63% and 5.17%, when the time constraints are 180 sec, 300 sec and 420 sec. Overall, the trends in different algorithms' performance, as the constraint becomes looser, are similar to what happens when changing vehicle capacities. When the constraint is tight ($\Omega = 180\,\text{sec}$), LTO-

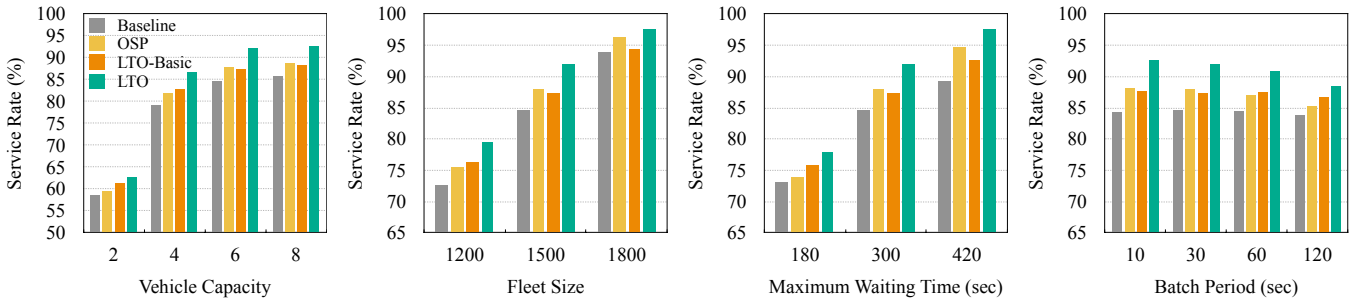

Fig. 5: A comparison of services rates (%) for varying vehicle capacities, fleet sizes, values of maximum waiting time constraint and lengths of batch period. In each case, we start with the default configuration and vary the chosen parameter.

TABLE II: A comparison of service rate (%) during the peak hour for using online learning.

Date of Experimental Data	LTO	LTO Without Online Learning
25th May 2016	91.80	91.32
26th May 2016	92.34	92.08

TABLE III: A comparison of service rate (%) of LTO during the peak hour for varying discount factors.

Date of Experimental Data	Discount Factor γ			
	0.90	0.95	0.97	0.99
25th May 2016	90.65	91.80	91.44	90.25
26th May 2016	91.53	92.34	91.97	90.87

Basic performs better than OSP. But when the constraint becomes looser, LTO-Basic has a significant drop in service rate compared to OSP and becomes worse than it. The performance of LTO shows that online learning can slow the drop. This is because passengers are more likely to be pooled together, which produces a more complex scheduling problem that requires the dispatcher to be more sophisticated.

Batch Period. When the window length increases, more requests are received per epoch, but the response time of passengers also grows and in turn actually leads to a tighter maximum waiting time constraint. The service rates of OSP and LTO decrease by 2.93% and a 4.22%, respectively, when the window length is increased from 10 sec to 120 sec. This is because, through re-optimization, they are actually using a batch period as large as possible to perform matching by considering all available requests. For them, a larger batch window would only reduce the time available for detours and produce worse performance. For LTO-Basic and Baseline, because they optimize only the current batch window, receiving more new orders offsets the drop in detour time and makes their service rates drop by less than 1%.

To analyse the role of online learning, we observe the difference when this component of LTO is removed. Table II shows the corresponding service rates over the two days. Online learning yields an average improvement of 0.37% and achieves greater improvements on the 25th, which indicates that mobility patterns can vary on different days and that the dispatcher can benefit from adapting the value function to real-time fluctuations. The results in Table II show that online learning only contributes 0.48% and 0.26% to the

TABLE IV: A comparison of computation time (sec) of different procedures during the peak hour for varying vehicle capacities ($m = 1500,\ \Omega = 300\,\text{sec},\ \Delta T = 30\,\text{sec}$).

Procedures	Vehicle Capacity κ			
	2	4	6	8
Scoring Process in OSP	0.05	0.07	0.07	0.07
Scoring Process in LTO	2.71	2.71	2.75	2.77
Online Training in LTO	4.12	4.18	4.21	4.26

performance of LTO on the 25th and 26th, suggesting that the supply-demand conditions may vary very little during the peak hour. We then examine the role of online learning during 00:00 - 08:00, where the change in the number of requests is greater than during the peak hour (19:00-20:00), and find that the contributions of online learning to the service rates increase to 1.57% and 1.21% on the 25th and 26th, respectively. This suggests that greater improvements can be achieved through online learning when real-time fluctuations are high. Typically, a 0.5% gain can be considered significant on real taxi systems [5].

We further investigate the impact of varying discount factors. Table III shows the service rates of LTO over the two days. In general, the discount factor implicitly determines the number of future epochs we take into consideration when computing the matching policy. A small discount factor results in a short-sighted strategy, whereas a large one provides a long-sighted goal. We see that the service rate rises when γ is increased from 0.9 to 0.95, making it more far-sighted. But the rate falls if we continue to increase γ. This might be because a very large discount factor induces the dispatcher to optimize the objective over the next few hours, which is longer than the length of our experiments.

Finally, we investigate the computation times of the scoring process using the value network and the online training process. Table IV shows the comparison for varying capacities. OSP only needs to retrieve the already calculated travel delays from the candidate schedule pool, so it takes very little time for scoring. Although LTO takes tens of times more computation time for scoring, the time cost is less than 3 sec. Therefore, LTO is able to run in real-time, i.e., the total computation time is less than 30 sec when the dispatch window is 30 sec. Note that the computation of OSP is around 10 sec [2]. As for the time consumed by the online

training process, since it can run separately from dispatching, it can meet the requirements of real-time deployment as long as the computation time is shorter than the dispatch window.

D. Discussion

Considering a longer horizon when computing a matching policy results in better performance than Baseline. Solely optimizing over all possible past epochs, OSP produces an increase of up to 5.39% ($\Omega = 420\,\text{sec}$). LTO-Basic, solely optimizing over the future epochs, yields an increase of up to 3.41% ($\Delta T = 10\,\text{sec}$). While optimizing over both the future and the past epochs, LTO achieves an improvement over Baseline by up to 8.42% ($\Delta T = 10\,\text{sec}$). Learning a good state value function can improve the performance of a dispatcher. Besides, due to the highly dynamic nature of mobility and the unavoidable prediction bias, the ability to continuously alter the matching policy online is also very important. Moreover, updating the value function during online dispatch has the potential to further improve performance. By employing online learning and re-optimization, LTO captures real-time supply-demand conditions and corrects for deviations with newly revealed requests. Compared to OSP and LTO-Basic, LTO improves the service rate by up to 4.61% ($\kappa = 4$) and up to 5.17% ($\Omega = 420\,\text{sec}$), respectively. Moreover, as not relying on simulators, the training process of the value function of LTO can be done in serval hours, which is much more efficient than that of LTO-basic.

VI. Conclusion

In this paper, we have proposed a value-based approach for vehicle dispatch in ride-sharing. Our work aims to investigate the role of optimization over a long horizon, including both the past and the future. The developed offline learning method can efficiently learn a value function that captures general urban mobility patterns from historical data. The developed online updating procedure can quickly adapt the offline learned value function to real-time dynamics of the ride-sharing system during operation. The value function is embedded into an online dispatcher that uses re-optimization for better performance. Numerical experiments show that, by optimizing over a longer horizon and adapting to real-time dynamics, the proposed method yields a higher service rate than the state-of-the-art (up to 5.17% at peak hour). Future work will investigate the role of idle vehicle rebalancing and dynamic fleet sizing.

References

[1] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," *Proceedings of the National Academy of Sciences*, vol. 114, no. 3, pp. 462–467, 2017.

[2] C. Li, D. Parker, and Q. Hao, "Optimal online dispatch for high-capacity shared autonomous mobility-on-demand systems," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 779–785, IEEE, 2021.

[3] Z. Qin, X. Tang, Y. Jiao, F. Zhang, Z. Xu, H. Zhu, and J. Ye, "Ride-hailing order dispatching at didi via reinforcement learning," *INFORMS Journal on Applied Analytics*, vol. 50, no. 5, pp. 272–286, 2020.

[4] S. Shah, M. Lowalekar, and P. Varakantham, "Neural approximate dynamic programming for on-demand ride-pooling," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 507–515, 2020.

[5] Z. Xu, Z. Li, Q. Guan, D. Zhang, Q. Li, J. Nan, C. Liu, W. Bian, and J. 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*, pp. 905–913, 2018.

[6] X. Tang, Z. Qin, F. Zhang, Z. Wang, Z. Xu, Y. Ma, H. Zhu, and J. Ye, "A deep value-network based approach for multi-driver order dispatching," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1780–1790, 2019.

[7] X. Tang, F. Zhang, Z. Qin, Y. Wang, D. Shi, B. Song, Y. Tong, H. Zhu, and J. Ye, "Value function is all you need: A unified learning framework for ride hailing platforms," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 3605–3615, 2021.

[8] I. Jindal, Z. T. Qin, X. Chen, M. Nokleby, and J. Ye, "Optimizing taxi carpool policies via reinforcement learning and spatio-temporal mining," in *2018 IEEE International Conference on Big Data (Big Data)*, pp. 1417–1426, IEEE, 2018.

[9] X. Yu and S. Shen, "An integrated decomposition and approximate dynamic programming approach for on-demand ride pooling," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 9, pp. 3811–3820, 2019.

[10] P. Santi, G. Resta, M. Szell, S. Sobolevsky, S. H. Strogatz, and C. Ratti, "Quantifying the benefits of vehicle pooling with shareability networks," *Proceedings of the National Academy of Sciences*, vol. 111, no. 37, pp. 13290–13294, 2014.

[11] J. Alonso-Mora, A. Wallar, and D. Rus, "Predictive routing for autonomous mobility-on-demand systems with ride-sharing," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3583–3590, IEEE, 2017.

[12] M. Tsao, D. Milojevic, C. Ruch, M. Salazar, E. Frazzoli, and M. Pavone, "Model predictive control of ride-sharing autonomous mobility-on-demand systems," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6665–6671, IEEE, 2019.

[13] M. Lowalekar, P. Varakantham, and P. Jaillet, "Zone path construction (zac) based approaches for effective real-time ridesharing," *Journal of Artificial Intelligence Research*, vol. 70, pp. 119–167, 2021.

[14] W. B. Powell, *Approximate Dynamic Programming: Solving the curses of dimensionality*, vol. 703. John Wiley & Sons, 2007.

[15] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[16] M. Szubert and W. Jaśkowski, "Temporal difference learning of n-tuple networks for the game 2048," in *2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–8, IEEE, 2014.

[17] Y. Jiao, X. Tang, Z. T. Qin, S. Li, F. Zhang, H. Zhu, and J. Ye, "Real-world ride-hailing vehicle repositioning using deep reinforcement learning," *Transportation Research Part C: Emerging Technologies*, vol. 130, p. 103289, 2021.

[18] W. B. Powell, "What you should know about approximate dynamic programming," *Naval Research Logistics (NRL)*, vol. 56, no. 3, pp. 239–249, 2009.

[19] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[20] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[22] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[23] N. Taxi and L. Commission, "Tlc trip record data." `https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page`, 07 2021. Accessed: 2022-01-19.