Transportation Research Part C 115 (2020) 102626

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

Dispatch of autonomous vehicles for taxi services: A deep reinforcement learning approach

Chao Mao[a,c], Yulin Liu[a,*], Zuo-Jun (Max) Shen[a,b,c]

[a] Department of Civil and Environmental Engineering, University of California Berkeley, Berkeley, CA 94720, USA
[b] Department of Industrial Engineering and Operations Research, University of California Berkeley, Berkeley, CA 94720, USA
[c] Tsinghua-Berkeley Shenzhen Institute (TBSI), Tsinghua University, Shenzhen 518055, China

ARTICLE INFO

Keywords:
Taxi dispatching
Demand rebalancing
Deep reinforcement learning
Actor-critic algorithm
Policy-gradient
Autonomous vehicles
Ride sharing

ABSTRACT

In this paper, we define and investigate a novel model-free deep reinforcement learning framework to solve the taxi dispatch problem. The framework can be used to redistribute vehicles when the travel demand and taxi supply is either spatially or temporally imbalanced in a transportation network. While previous works mostly focus on using model-based methods, the goal of this paper is to explore the policy-based deep reinforcement learning algorithm as a model-free method to optimize the rebalancing strategy. In particular, we propose an actor-critic algorithm with feed-forward neural networks as approximations of both policy and value functions, where the policy function provides the optimal dispatch strategy and the value function estimates the expected costs at each time stamp. Our numerical studies show that the algorithm converges to the theoretical upper bound with less than 4% optimality gap, whether the system dynamics are deterministic or stochastic. We also investigate the scenario where we consider user priority and fairness, and the results indicate that our learned policy is capable of producing a superior strategy that balances equity, cancellation, and level of service when user priority is considered.

1. Introduction

In the past decade, we have seen rapid expansion of ride sharing services and the emerging development in self-driving technologies. We believe that over the coming decades, ride sharing companies such as Uber and Lyft may aggressively begin to use shared fleets of electric and self-driving cars that could be summoned to pick up passengers and shuttle them to offices and stores. One major operational challenge such systems might encounter, however, is the imbalance of supply and demand. The travel patterns are asymmetric both spatially and temporally, thus causing the vehicles to be clustered in certain regions at certain times of day, and customer demand may not be satisfied in time. The primary goals of our research, therefore, is to come up with a ***centralized***, yet efficient, operational strategy such that it can: (1) deal with the imbalanced demand and supply of the ride sharing service; (2) guarantee the level of service in both high and low demanding areas and times; and (3) ensure the equity of the users.

For traditional taxi fleets, ride sharing operators such as Uber and Lyft usually use pricing incentives (e.g. surge pricing, spatial pricing) to help redistribute vacant vehicles and attract drivers towards regions with high demand. Zha et al. (2018) build a model based on a geometric matching framework and investigate the effects of spatial pricing and its regulation in ride-sourcing markets. Banerjee et al. (2015) build a queueing-theoretic economic model to study the value of dynamic pricing in ride sharing platforms, and

* Corresponding author at: 107D McLaughlin Hall, University of California, Berkeley, Berkeley, CA, 94720, USA.
E-mail addresses: chaomao@berkeley.edu (C. Mao), liuyulin101@berkeley.edu (Y. Liu), maxshen@berkeley.edu (Z.-J.M. Shen).

https://doi.org/10.1016/j.trc.2020.102626
Received 4 September 2019; Received in revised form 18 March 2020; Accepted 20 March 2020
Available online 10 April 2020
0968-090X/

they find that while the performance of the system under any dynamic pricing strategy is not better than that under the optimal static pricing policy, dynamic pricing is much more robust to fluctuations in the system parameters compared to static pricing. On the other hand, however, studies have been conducted to question the effectiveness of dynamic pricing in real-world ride sourcing market. Based on real data collected from Uber's smartphone application, Chen et al. (2015) observe that on a micro-scale, surge prices have a strong negative impact on passenger demand, and a weak positive impact on car supply. The authors also argue that Uber's reliance on discrete surge areas introduces unfairness into its system – two users standing a few meters apart may receive dramatically different surge multipliers.

From the perspective of autonomous taxi fleet, however, the operator can directly control all the vehicles and thus make *centralized* decisions on how to dispatch the vacant vehicles at any time of interest. Lots of model-based methods have been proposed in the open literature to optimize the strategy that controls the fleet. Pavone et al. (2012) develop a real-time rebalancing policy based on a fluid model of the system, in which they assume that all the pickups and drop-offs happen at a set of stations. The authors show that under their rebalancing policy, every station will reach an equilibrium in which there are excess vehicles and no waiting customers. Similarly in Volkov and Aslam (2012), Volkov et al. establish an analytical Markov-based framework to describe an urban transportation network, in which there are discrete pickup and drop-off locations, and they propose a practical redistribution policy and show that it performs favorably in light of different optimization criteria. Zhang and Pavone (2016) propose a queueing model for the control of autonomous mobility-on-demand (MOD) system, and they show that an optimal open-loop policy can be found by solving a linear program, based on which they develop a closed-loop real-time rebalancing policy and then apply it to a case study of New York. In more recent studies, Iglesias et al. (2019) further extended the analysis of the MOD system in Zhang and Pavone (2016) by introducing a Baskett-Chandy-Muntz-Palacios (BCMP) queuing-theoretical framework, in which the vehicle repositioning problem has been formulated as a non-linear optimization problem. The authors cast the original problem into a linear program by assuming infinite fleet supply, and provided rigorous mathematical proofs to the asymptotic property of infinite supply system. With the approximated framework, this work enables the analysis of probabilistic distribution of the fleet, and can also be greatly used to synthesize both vehicle routing and rebalancing policies. Braverman et al. (2019) continued the thread of BCMP approach to the empty-car repositioning problem in the ride sharing system. In their work, the authors proposed a (non-linear) fluid-based optimization problem and proved that when fleet supply grows to infinity, the routing policy from the program is asymptotically optimal when the system is at its steady state. This policy, while under somewhat unrealistic assumptions, gives a valueable asymptotically tight upper bound to a ride sharing system. Shou et al. (2019), on the other hand, proposed a Markov Decision Process to determine the optimal sequential passenger-seeking strategy for e-hailing drivers, in which dynamic programming and Monte Carlo simulation have been employed. Such framework is deemed to be especially beneficial to the autonomous e-hailing platform as it can significantly improve the system level reward and vehicle utilization.

While the above studies rely on steady-state formulations and their control policies are time-invariant, there is another branch of study that incorporates demand forecasting and utilizes model predictive control (MPC) method. Miao et al. (2016) present a receding horizon control (RHC) framework for large-scale taxi dispatching system. In the study, they utilize both historical and real-time GPS and occupancy data to build demand models, and apply predicted models and sensing data to decide dispatch locations for vacant taxis considering multiple objectives. Zhang et al. (2016) present a model predictive control (MPC) approach to optimize vehicle scheduling and routing in an autonomous MOD system. At each optimization step, the vehicle scheduling and routing problem is formulated as a mixed integer linear program (MIP), and their case study shows that the MPC method outperforms previous time-invariant control strategies. While in Miller and How (2017), Miller and How present a predictive positioning algorithm which uses customer arrival rate information to position vehicles at key nodes in a MOD network graph in order to minimize the expected customer waiting time. Later, Iglesias et al. (2018) propose a MPC algorithm that leverages short-term demand forecasts based on historical data to compute rebalancing policies, and their algorithm is built on a formulation that computes the optimal rebalancing strategy and the minimum feasible fleet size for a given travel demand. With simulations based on real data from DiDi Chuxing, they show that the approach scales well for large systems.

While most previous works focus on model-based methods to solve for the optimal rebalancing strategy, these methods have three main drawbacks. First of all, model-based methods usually use a finite set of parameters to represent the network dynamics, such as the customer demands and travel times, which need to be estimated before solving the model. Second, in these methods, strong assumptions, such as Markovian property and constant arrival rate, have to be made on the network, which are typically difficult to validate in the real-world networks. Third, some of the proposed methods suffer from the curse of dimensionality and therefore do not scale well in large systems. With these concerns, the aim of this paper is to explore model free methods to solve for the rebalancing strategy of the autonomous taxi fleet. Specifically, we will incorporate the framework of Reinforcement Learning (RL), which is primarily concerned with how to obtain an optimal policy when the model of Markov decision process (MDP) is not available. Therefore, instead of relying on any prior information of the model, the RL learning agent will interact with the transportation network and update the control strategy directly. Recently, there are a lot of studies working on the application of RL methods in the domain of transportation engineering, and interested readers may refer to other RL applications such as (Prashanth and Bhatnagar, 2011; Li et al., 2016; Abdulhai et al., 2003) for traffic signal control, (Mao and Shen, 2018) for adaptive routing, and (Zhu and Ukkusuri, 2014; Walraven et al., 2016) for traffic management. Among these studies, we found the study by Lin et al. (2018) most relevant to our research. In their work, the authors have formulated a fleet management problem for the ride-sharing platform using multi-agent framework, and they have proposed a contextual multi-agent RL algorithm that incorporates first training and then coordinating multiple homogeneous single agents. However, we would like to mention that their work, though tackles a similar problem in our study, has a significantly different goal such that their policy is ***decentralized*** that may sacrifice system level optimality. In this paper, however, we define and investigate a variate of actor-critic algorithm, which belongs to the family of policy

gradient algorithms (Sutton et al., 2000), to learn a *centralized* policy that optimizes the system level returns.

The remainder of this paper is organized as follows. In Section 2, we formulate the problem and present our basic assumptions. In Section 3, we first introduce the framework of the actor-critic method and the adaptation we have made; then we propose an integer program to derive the theoretical upper bound of the total rewards we can obtain if we assume the dynamics of the system are deterministic and known to us. In the meantime, we present two more realistic scenarios where we consider stochasticity, cancellation and passenger priority. Section 4 presents the experimental results and implications of the RL methods under different scenarios. Section 5 offers the conclusions and directions for future research.

2. Problem formulation

In this section, we first formulate the taxi repositioning problem, and then introduce the environmental setup in our subsequent learning process.

Suppose we are providing taxi services to a service region within a certain period of time. We first assume that the regions have been discretized into a set $\mathcal{N}$ of disjoint zones, each of which can be represented by a node (e.g., a "station") in a directed graph. We further assume the service time is represented by discrete time intervals of size Δt. The period of time under consideration is denoted as $\mathcal{T} = [1, 2, ..., T]$. For simplicity, we assume the discrete time interval Δt is small, and all the deployment of vehicles happen at the end of each time interval. Therefore, if a waiting passenger is not served at time t, he/she will have to wait until at least $t + 1$ to be served, in other words, no dispatching arrangement will happen between t and $t + 1$. And all serving vehicles will become available again after travelling for some time intervals and dropping passengers at the destination zone. And for now, we further assume we will not lose any passengers even if we let them wait for a long time.

At the end of each time interval t, therefore, we need to decide the number of vehicles to be dispatched from zone $i \in \mathcal{N}$ to zone $j \in \mathcal{N}$, which is denoted as x_{ijt}. Suppose the number of waiting passengers who want to travel from zone i to zone j at time interval t is p_{ijt}, and the number of available (empty) vehicles at zone i is v_{it}. Once x_{ijt} has been determined, the number of passenger can be served from zone i to j at this time interval t, denoted by y_{ijt}, can be determined by $y_{ijt} = \min(x_{ijt}, p_{ijt})$. Namely, if $x_{ijt} < p_{ijt}$, then the supply is less than the demand, and only a portion of the passenger calls can be answered; and if $x_{ijt} \geqslant p_{ijt}$, we have dispatched more vehicles than the current number of waiting passengers. Therefore, not only all the passengers' requests can be satisfied, but we also let some empty vehicles to travel from i to j to meet the future potential demand in zone j.

From the passenger's perspective, we assume there are costs associated with the waiting time experienced by all the passengers. Let δ_{ijt} be the cost of letting one customer who wants to go from i to j wait for one unit of time interval Δt at time t. Thus the total waiting time costs from time t to $t + 1$ is $\sum_{i,j\in\mathcal{N}} \left(p_{ijt} - y_{ijt}\right)\cdot\delta_{ijt}$. From the system operation's perspective, there are costs resulting from repositioning the empty vehicles between different zones. We assume the cost of repositioning an empty vehicle from zone i to zone j at time t is c_{ijt}. Thus the total repositioning costs at time t is $\sum_{i,j\in\mathcal{N}} \left(x_{ijt} - y_{ijt}\right)\cdot c_{ijt}$.

With the above assumptions and formulations, we then define the state space and action space of the problem under the structure of reinforcement learning. The state space is defined as:

$$\mathbf{S} \coloneqq \{(t, P_t, V_t): t \in \mathcal{T}, P_t \in \mathbb{R}_+^{n\times n}, V_t \in \mathbb{R}_+^{n}\} \tag{1}$$

Each state $\mathbf{s}_t = (t, P_t, V_t)$ is characterized by the current time interval t, the waiting passenger demand vector $P_t = \{p_{ijt}: i \in \mathcal{N}, j \in \mathcal{N}\}$, and the available vehicle count vector $V_t = \{v_{it}: i \in \mathcal{N}\}$. The corresponding action space for state s is defined as:

$$\mathbf{A}(\mathbf{s}_t): \left\{x_{ijt}: \sum_{j\in\mathcal{N}} x_{ijt} = v_{it},\ i \in \mathcal{N}, j \in \mathcal{N}, x_{ijt} \in \mathbb{R}_+\right\} \tag{2}$$

The action $\mathbf{a}_t$ at time interval t is characterized by x_{ijt}, $i, j \in \mathcal{N}$, which is the number of available vehicles to be dispatched from zone i to zone j at the end of time interval t.

The reward we obtain from time t to $t + 1$ would be the negative of the costs we have incurred, which consists of the waiting time costs and the costs of repositioning empty vehicles, and is formulated by Eq. (3).

$$r\left(\mathbf{s}_t, \mathbf{a}_t\right) = -\sum_{i,j\in\mathcal{N}} \left[\left(p_{ijt} - y_{ijt}\right)\delta_{ijt} + \left(x_{ijt} - y_{ijt}\right)c_{ijt}\right] \tag{3}$$

Therefore, the objective of our dispatching system is to provide the optimal vehicle dispatch strategy at the lowest expected total operational costs (a.k.a., highest expected total rewards). Let $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ denote the policy (a.k.a. dispatching strategy) we are interested in, which produces an action $\mathbf{a}_t$ given the state $\mathbf{s}_t$. Notice that the policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ is parametrized by θ that can be optimized by solving the following problem.

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \sum_{t=1}^{T} \{\mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}[r(\mathbf{s}_t, \mathbf{a}_t)]\} \tag{4}$$

To solve the above dynamic sequential decision problem, we make the following three assumptions. First of all, to avoid the curse of dimensionality that arises from discrete state and action space, we allow both the state and action vectors to be continuous, specifically, p_{ijt}, v_{it} and x_{ijt} are all continuous variables. With such continuous state and action space, however, value-based reinforcement learning algorithms such as deep Q-learning (Mnih et al., 2013) become intractable. Therefore, in this paper, we mainly focus on policy-based reinforcement learning methods. Lastly, our feasible action space is state-dependent, i.e. the number of vehicles dispatched from one zone should equal to the number of available vehicles within that zone.

3. Methods

Considering the above challenges, in this section, we first introduce vanilla policy gradient algorithms. Second, we discuss the actor-critic algorithm, which belongs to the family of policy gradient methods however is much more efficient than the vanilla methods. We also introduce the adaptations we have made to the actor critic algorithm to enforce feasible actions. Third, we derive the theoretical upper bound of the total rewards we can get if we assume the dynamics of the system are deterministic and known to us. Finally, we introduce two different scenarios, in which more realistic settings such as cancellation and stochastic demand profiles are considered.

3.1. Vanilla policy gradient algorithms

Policy gradient methods are policy-based reinforcement learning techniques that rely on optimizing parametrized policies with respect to the total expected return (long-term cumulative reward) by gradient descent algorithms. They do not suffer from many of the problems that traditional value-based reinforcement learning methods might have, such as the complexity arising from continuous states and actions. The general idea of policy gradient is that, by generating samples of trajectories (sequences of tuples of state, action and reward) from the environment based on the current policy function, we can collect the rewards associated with different trajectories, then we can update our parametrized policy function such that high-reward paths will become more likely and low-reward paths become less likely. One advantage of policy gradient methods is their strong convergence property, which is naturally inherited from gradient descent methods.

Suppose we have a differentiable policy function $\pi_\theta(\mathbf{a}|\mathbf{s})$, where $\mathbf{a}$ is the action, $\mathbf{s}$ is the state, and θ represents the actor's parameters to be updated and learned. Further suppose our reward function is $r(\mathbf{s}, \mathbf{a})$, then from Eq. (3), the optimal policy can be obtained by:

$$\theta^{\star} = \underset{\theta}{\operatorname{argmax}} J(\theta) = \underset{\theta}{\operatorname{argmax}} \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[r(\tau)] = \underset{\theta}{\operatorname{argmax}} \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right]$$

where τ denotes a trajectory of the $(\mathbf{s}_t, \mathbf{a}_t)$ tuples sampled from the policy $\pi_\theta(\mathbf{a}|\mathbf{s})$.

Instead of solving the above problem analytically, the family of policy gradient algorithms apply the gradient descent algorithms and sampling techniques to iteratively improve the performance of the policy due to the fact that the gradient of the objective function $J(\theta)$ (Eq. (5)) can be approximated by Eq. (6) (Sutton and Barto, 2018), where the tuples of $(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$ are samples generated by the up-to-date policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\left(\sum_t \nabla_\theta \log\pi_\theta\left(\mathbf{a}_t\middle|\mathbf{s}_t\right)\right)\left(\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right)\right] \tag{5}$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_t \nabla_\theta \log\pi_\theta\left(\mathbf{a}_{i,t}\middle|\mathbf{s}_{i,t}\right)\right)\left(\sum_t r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right)\right)\right] \tag{6}$$

The vanilla policy gradient algorithm (a.k.a., REINFORCE algorithm (Sutton and Barto, 2018; Williams, 1992)) is summarized in Algorithm 1. Within each iteration, step 1 samples state-action-reward tuples and steps 2 and 3 perform policy update. The process has an intuitive appeal such that samples with higher rewards are more likely to be sampled and those "good" samples will further improve the policy updating.

Algorithm 1. Vanilla policy gradient algorithm

Procedure Vanilla Policy Gradient
Initialize: A differentiable policy function $\pi_\theta(\mathbf{a}|\mathbf{s})$
Initialize: θ
for each episode **do**
1. sample $\{\mathbf{s}_{i,t}, \mathbf{a}_{i,t}, r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}), \mathbf{s}_{i,t+1}\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it in the simulator)
2. calculate $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}[(\sum_t \nabla_\theta \log\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}))(\sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}))]$
3. $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$
end for
end procedure

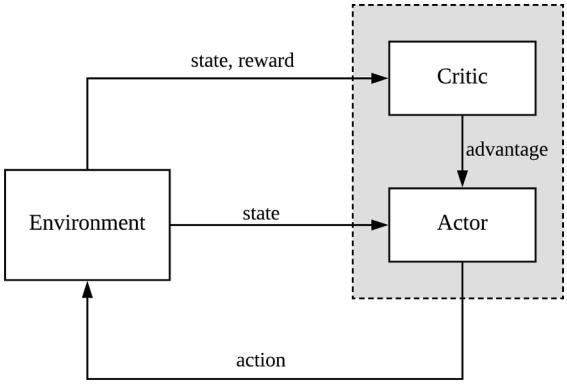

Fig. 1. Schematic overview of actor-critic algorithm.

3.2. Actor-critic algorithm

In previous sections we have shown that the essence of policy gradient algorithm is to increase the probabilities of "good" actions by a "sampling-updating" loop. Unfortunately, in practice the sampled rewards usually have very large variances, which induces large variance of the estimated gradients. Therefore, the vanilla policy gradient method is prone to be unstable and inefficient.

Actor-critic methods solve those issues by adding a value function ("critic") (Grondman et al., 2012) to the vanilla policy gradient algorithm. During the training process, the policy (a.k.a., "actor") generates a trajectory of samples as in Algorithm 1. However, before updating the policy, the value function approximates the expected rewards and updates itself using the generated samples. The value function is then used to update the policy's parameters in the direction of performance improvement. Since the critic is modeled by a bootstrap method, it reduces the variance so the learning is more stable than vanilla policy gradient methods.

Fig. 1 below shows the schematic structure of an actor-critic algorithm. The learning agent consists of a critic agent and an actor agent. The actor is responsible for generating actions based on the states of the environment, and the critic evaluates the value function of the current policy by observing the feedback (states and rewards) from the environment. Then the information of advantage (the improvement when compared with the average value of the current state) is sent to the actor to improve the current policy function unit.

Following the notations in previous sections, suppose we have a policy function (actor) $\pi_\theta(\mathbf{a}|\mathbf{s})$. Further denote the value function (critic) corresponding to policy $\pi_\theta(\mathbf{a}|\mathbf{s})$ as $V_\phi^\pi(\mathbf{s})$, which is parameterized by ϕ. Then the actor-critic algorithm works as in Algorithm 2.

Algorithm 2. Batch actor-critic algorithm

Procedure Actor-critic
Initialize: A differentiable policy function $\pi_\theta(\mathbf{a}|\mathbf{s})$
Initialize: θ **for** each episode **do**

1. sample $\{\mathbf{s}_{i,t}, \mathbf{a}_{i,t}, r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}), \mathbf{s}_{i,t+1}\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it in the simulator)
2. fit $\widehat{V}_\phi^\pi(\mathbf{s})$ to sampled reward sums
3. evaluate the advantage as $\widehat{A}^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) = r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \widehat{V}_\phi^\pi(\mathbf{s}_{i,t+1}) - \widehat{V}_\phi^\pi(\mathbf{s}_{i,t})$
4. calculate $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^N [\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \cdot \widehat{A}^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})]$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

end for
end procedure

In the algorithm, step 2 updates the value function using the generated samples and is later used to calculate the advantage $\widehat{A}^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$ in step 3, which estimates how much better off the corresponding action is compared to the average value function we have estimated. Therefore, by multiplying the gradient of the policy function with the advantage in step 4, the algorithm updates the parameters such that high-reward actions become more likely. Also notice that the value function $\widehat{V}_\phi^\pi(\mathbf{s})$ in step 2 is estimated by a bootstrap method, which significantly reduces the variance of the parameter estimation.

Although we can choose any parametric functions for the policy function $\pi_\theta(\mathbf{a}|\mathbf{s})$ and critic function $V_\phi^\pi(\mathbf{s})$, in this study, we pick feed forward dense neural networks to represent both functions due to their flexibility. However, in our problem formulation, the number of vehicles dispatched from each zone should be non-negative and sum up to the total number of available vehicles in that zone. Therefore, a special output transformation function (activation) has to be employed to the policy neural network to enforce such constraints. To be more specific, suppose the output vector from the policy network $\pi_\theta(\cdot|\mathbf{s})$ is $\mathbf{a}$, and each element a_{ij} corresponds to the edge between the origin zone i and the destination zone j. Then we can view a_{ij} as some weight factor and apply the following transformation function to get the number of vehicles to dispatch from zone i to zone j:

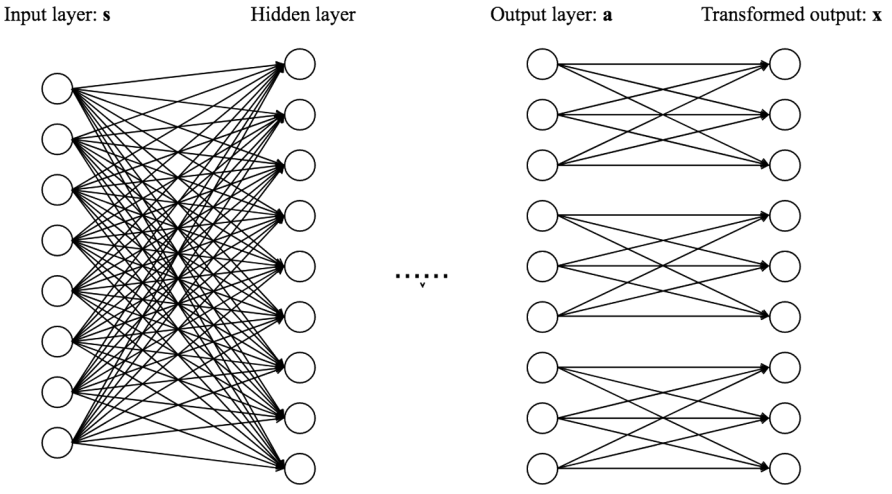

Fig. 2. Post-processing of action function outputs.

$$x_{ij} = v_i \cdot \frac{|a_{ij}|}{\sum_{k \in \mathcal{N}} |a_{ik}|}$$

Then the resulting actor vector **x** is a feasible action given the current state vector **s**. The above procedure is summarized in Fig. 2.

3.3. Theoretical bound - an integer program

Nomenclature

Parameters	
δ_{ijt}	the cost of letting customer who wants to go from i to j wait for one unit of time at time t
λ_{ijt}	number of ride requests from zone i to zone j at time t
τ_{ijt}	the travel time from zone i to zone j at time t
c_{ijt}	the cost of repositioning an empty vehicle from zone i to zone j at time t
n_i	the initial number of vacant vehicles in zone i
Sets	
$\mathcal{A}_{it} = \{(j, t'): t' + \tau_{jit'} = t\}$	set of departure zones and times that would let vehicle arrive in zone i at time t
Variables	
p_{ijt}	number of outstanding passengers that are waiting to go from zone i to zone j at time t
v_{it}	number of vehicles available in zone i at the beginning of time t
x_{ijt}	number of vehicles that are dispatched from zone i to zone j at time t
y_{ijt}	number of passengers going from zone i to zone j that are served at time t

In this section, we aim to look at the case where we have full information on the travel demand and the system dynamics, both of which are deterministic and known. We can then formulate the dispatching problem as an optimal control problem to maximize the total expected rewards and solve the optimal dispatching strategy. Then we compare the resulting rewards with the rewards we obtain from the model-free reinforcement learning method. The definition of the integer program is as formulated into problem **P1**.

$$\textbf{P1}: \quad \min_{x,y,p} \sum_{i,j \in \mathcal{N}} \sum_{t \in \mathcal{T}} \left(p_{ijt} - y_{ijt}\right)\delta_{ijt} + \left(x_{ijt} - y_{ijt}\right)c_{ijt}$$

$$\text{s. t.} \quad y_{ijt} = min(x_{ijt}, p_{ijt}) \ \forall\, i, j \in \mathcal{N}, t \in \mathcal{T} \tag{1.1}$$

$$p_{ij(t+1)} = p_{ijt} - y_{ijt} + \lambda_{ij(t+1)} \ \forall\, i, j \in \mathcal{N}, t \in \mathcal{T}) \tag{1.2}$$

$$v_{it} = \sum_{j \in \mathcal{N}} x_{ijt} \ \forall\, i \in \mathcal{N}, t \in \mathcal{T} \tag{1.3}$$

$$v_{i0} = n_i \ \forall\, i \in \mathcal{N} \tag{1.4}$$

$$v_{it} = \sum_{(j,t') \in \mathcal{A}_{it}} x_{jit'} \ \forall\, t > 0, i \in \mathcal{N} \tag{1.5}$$

$$p_{ij0} = \lambda_{ij0} \ \forall\, i, j \in \mathcal{N} \tag{1.6}$$

$$x_{ijt} \in \mathbb{N}^+ \ \forall\ i, j \in \mathcal{N}, t \in \mathcal{T} \tag{1.7}$$

In the formulation, the objective function consists of the waiting time costs for the passengers and the costs of repositioning empty vehicles. The first constraint (1.1) states that the number of served customers at any time is either equal to the number of waiting customers (when enough vehicles are dispatched) or the number of dispatched vehicles (when supply of vehicles is less than customers). Constraint (1.2) ensures the conservativeness of the passenger demand, and constraints (1.3) and (1.5) enforce that the number of arriving vehicles must equal to the number of departing vehicles. We assume that at the beginning of the planning horizon, the initial number of available vehicles and waiting passengers in each zone are given, as shown in constraints (1.4) and (1.6). Constraint (1.7) ensures the decision variables to be non-negative integers.

Notice that in our formulation, we treat idling vehicles in the same zone as a special case of repositioning. However, we can alter the parameters δ_{ijt} to reflect any realistic scenarios. For example, parking in the downtown area can be very expensive during daytime, thus we can impose higher δ value to encourage the vacant vehicles to move out of the area during that time.

3.4. Different scenarios: consider user priority

One assumption we make in the above programs is that we will not lose any passengers regardless of their waiting time. In other words, the optimal dispatching strategy does not differentiate passengers who have already waited for a long time and who just start waiting. As a result, in our dispatching system, even if there are some areas where passengers' requests have been dismissed for a long period of time, we might still send vehicles to other areas where we can maximize the total rewards of the system. Such an assumption, however, can hardly be true and is far from ideal in the real world. Passengers who have waited for a long time, or *impatient passenger*, might just cancel their requests and thus induce higher costs. Therefore, we need to take into account users' different tolerance of waiting time, in which impatient passengers have higher waiting costs. Specifically, we have considered two scenarios in this paper. The first scenario, which assumes deterministic demand profile, only distinguishes two types of passengers and does not consider cancellation. The second scenario, on the other hand, considers a more realistic case where the demand and initial number of vehicles follow some distributions. In the meantime, we further distinguish passengers by using a linearly increasing waiting costs function capped by a maximal waiting time by which users will cancel their request if exceeded. A detailed mathematical illustration is presented below.

Scenario I

For simplicity, we let the waiting time penalty δ distinguishes *patient passengers* and *impatient passengers* by setting it to be a two-valued function of the waiting time ω. Specifically, a customer who travels from zone i to zone j at time t, if his/her total waiting time is ω, then the cost of letting him/her wait for one more unit of time is formulated by Eq. (7), where Ω is a predetermined threshold that specifies the maximal tolerance of waiting time to be an *impatient passenger*. We can imagine that if $\delta^2_{ijt} \gg \delta^1_{ijt}$, we are imposing a very large penalty if we let customers wait more than Ω units of time, thus we can ensure some level of service with regard to the maximum waiting time.

$$\delta_{ijt}(\omega) = \begin{cases} \delta^1_{ijt} & , \text{if } \omega \leqslant \Omega \\ \delta^2_{ijt} & , \text{if } \omega > \Omega \end{cases} \tag{7}$$

To solve the above problem with user priority, we modify the state space and the reward function as follows. For any origin destination pair (i, j) at any time t, the number of waiting passengers p_{ijt} can be divided into two groups: p^1_{ijt} and p^2_{ijt}, where p^1_{ijt} is the number of outstanding passengers who have waited less than or equal to Ω units of time and p^2_{ijt} is the number of those who have waited more than Ω. We call those p^1_{ijt} passengers the *patient passengers*, and those p^2_{ijt} passengers the *impatient passengers*.

Given x_{ijt}, i.e. the number of vehicles to dispatch from zone i to j at time t, we still have $y_{ijt} = \min(x_{ijt}, p_{ijt})$, where y_{ijt} is the number of customers that can be served. Since "impatient passengers" have higher priority, the number of "impatient passengers" will become as $p^2_{ijt} - \min\left(p^2_{ijt}, y_{ijt}\right)$, and the number of "patient passengers" will turn into $p^1_{ijt} - \max\left(0, y_{ijt} - p^2_{ijt}\right)$. Therefore the associated waiting time cost at this time is: $\sum_{i,j\in\mathcal{N}} \left[p^2_{ijt} - \min\left(p^2_{ijt}, y_{ijt}\right)\right]\cdot\delta^2_{ijt} + \left[p^1_{ijt} - \max\left(0, y_{ijt} - p^2_{ijt}\right)\right]\cdot\delta^1_{ijt}$, and the repositioning cost stays the same, i.e. $\sum_{i,j\in\mathcal{N}}\left(x_{ijt} - y_{ijt}\right)\cdot c_{ijt}$. Now the new state space is defined as:

$$\mathbf{S} := \{(t, P^1_t, P^2_t, V_t) | t \in \mathcal{T}, P^1_t \in \mathbb{R}^{n\times n}_+, P^2_t \in \mathbb{R}^{n\times n}_+, V_t \in \mathbb{R}^n_+\}$$

Each state $\mathbf{s} = (t, P^1_t, P^2_t, V_t)$ is characterized by the current time interval t, the demand vector for those "patient passengers" $P^1_t = \left\{p^1_{ijt}: i \in \mathcal{N}, j \in \mathcal{N}\right\}$, the demand vector for those "impatient passengers" $P^2_t = \left\{p^2_{ijt}: i \in \mathcal{N}, j \in \mathcal{N}\right\}$, and the available vehicle count vector $V_t = \{v_{it}: i \in \mathcal{N}\}$, while the action space stays the same.

Scenario II

Previous cases have assumed that the demand profile and initial vehicle distribution are both deterministic across the time-of-day and OD pairs, however, we are also interested in a more realistic case where we allow stochasticity and randomness. Therefore, in this scenario, we first relax the assumption of deterministic demand profile and vehicle distributions. Specifically, we let the OD demand $\lambda_{ijt} \sim \mathcal{D}_\lambda(\theta_1)$, and initial vehicle number $n_i \sim \mathcal{D}_n(\theta_2)$, where $\mathcal{D}_\lambda$ and $\mathcal{D}_n$ are the pre-specified distributions parameterized by respectively

θ_1 and θ_2.

Second, we further generalize the **unit** waiting time cost function $\delta_{ijt}(\omega)$ to be a piecewise constant function (with m pieces) of waiting time ω, as in Eq. (8), where α_k's and Ω_k's are predetermined values that indicate respectively the level of waiting cost and a set of thresholds to specify how "impatient" the passengers are; $\mathbb{I}_{\Omega_k}(\omega)$ is an indicator function of Ω_k as in Eq. (9).

$$\delta_{ijt}(\omega) = \sum_{k=1}^{m} \alpha_k \cdot \mathbb{I}_{\Omega_k}(\omega) \tag{8}$$

$$\mathbb{I}_{\Omega_k}(\omega) = \begin{cases} 1 & \text{, if } \Omega_k \leqslant \omega < \Omega_{k+1} \\ 0 & \text{, otherwise} \end{cases} \tag{9}$$

Notice that to guarantee the monotonicity, we also let $0 \leqslant \Omega_1 \leqslant \Omega_2 \leqslant ... \leqslant \Omega_m \leqslant \Omega_{m+1}$, and $\alpha_1 \leqslant \alpha_2 \leqslant ... \leqslant \alpha_m$.

Third, we introduce a new type of cost – cancellation cost – by eliminating passengers when their waiting time exceeds some thresholds. Let Φ denote the maximal tolerance of waiting time, then we define the cancellation cost for a passenger who *requests* a travel from zone i to zone j at time t and has waited for ω to be:

$$\eta_{ijt}(\omega) = \begin{cases} \phi & \text{, if } \omega \geqslant \Phi \\ 0 & \text{, if } \omega < \Phi \end{cases} \tag{10}$$

In the above equation, ϕ is a predetermined fixed value that ascribes the cancellation cost. Also notice that we let $\phi \geqslant \alpha_m$ and $\Phi \geqslant \Omega_{m+1}$ to guarantee monotonicity.

Similar to the Scenario I, we have modified the state space, the transition function, and the reward function to adapt to our actor-critic algorithm. First of all, passengers who have requested service from zone i to j at time t, p_{ijt}, will be divided into $m + 1$ groups: p_{ijt}^1, p_{ijt}^2, ..., p_{ijt}^m, and l_{ijt}, where p_{ijt}^k, $1 \leqslant k \leqslant m$ is the number of passengers who have waited for ω units of time that satifies $\Omega_k \leqslant \omega < \Omega_{k+1}$, and l_{ijt} is the number of passengers who would cancel their requests. Accordingly, the new state space is defined as:

$$\mathbf{S} := \{(t, L_t, V_t, P_t^1, P_t^2, ..., P_t^m) | t \in \mathcal{T}, L_t \in \mathbb{R}_+^{n \times n}, V_t \in \mathbb{R}_+^n, P_t^k \in \mathbb{R}_+^{n \times n}, k \in \{1, 2, ..., m\}\}$$

Each state $\mathbf{s} = (t, P_t^k, C_t, V_t)$ is characterized by the current time interval t, the demand vector for waiting passengers with different waiting times $P_t^k = \left\{p_{ijt}^k : i \in \mathcal{N}, j \in \mathcal{N}\right\}$, in which $\forall\ k \in \{1, 2, ..., m\}$, the vector for cancelled requests $L_t = \{l_{ijt} : i \in \mathcal{N}, j \in \mathcal{N}\}$, and the available vehicles vector $V_t = \{v_{it} : i \in \mathcal{N}\}$.

Second, we have adapted the transition function by introducing a queue for the waiting passengers. Specifically, given the number of vehicles dispatched from zone i to j at t, the number of served passengers (y_{ijt}), and waiting passengers (p_{ijt}^k) will respectively be:

$$y_{ijt} = \min\left(x_{ijt}, p_{ijt} - l_{ijt}\right) p_{ijt}^{m'} = p_{ijt}^m - \min\left(p_{ijt}^m, y_{ijt}\right) p_{ijt}^{k'} = p_{ijt}^k - \max\left(0, y_{ijt} - \sum_{r=k+1}^{m} p_{ijt}^r\right), \forall\ k \in \left\{1, 2, ..., m-1\right\}$$

Accordingly, the total costs **at this** timestamp is:

$$z_t = z_t^{wt} + z_t^{cxl} + z_t^{rep} = \sum_{i,j \in \mathcal{N}} \sum_{k'=1}^{m} \alpha_k \cdot p_{ijt}^{k'} + \sum_{i,j \in \mathcal{N}} \phi \cdot l_{ijt} + \sum_{i,j \in \mathcal{N}} c_{ijt} \cdot \left(x_{ijt} - y_{ijt}\right)$$

where z_t^{wt}, z_t^{cxl}, z_t^{rep} are respectively the total waiting time costs, cancellation costs, and repositioning costs.

4. Case study

In this section, we first present the experimental setup of the case study, then we demonstrate the performance of the above actor-critic method, followed by the comparison with both the vanilla policy gradient method and the theoretical upper bound derived by the integer programming problem. Lastly, we discuss the implications of the two more realistic scenarios where we consider passengers' different tolerance of waiting time, i.e., user priority.

4.1. Experimental setup

Generic settings

In our case study, we build simulators that resemble the current yellow cab service and transportation demand patterns in the New York Manhattan area, however with the assumption that all taxis are autonomous vehicles and can be dispatched in a *centralized* manner. Our goal, therefore, is to find an optimal taxi dispatching strategy using the proposed actor-critic algorithm with such simulators.

To setup simulators, We first obtained a geo-spatial dataset that partitions the Manhattan area into 64 zones by the NYC TLC (Taxi & Limousine Commission), as shown in Fig. 3 and Fig. 4. Second, we obtained a yellow taxi trip dataset from the NYC TLC within the period of June 2016 to see how the travel demand varies across regions and along the day. Each observation is an actual taxi trip that records the pick-up and drop-off times/locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts in the New York City. To uncover the spatio-temporal traffic patterns, we estimate the travel demand for each zone

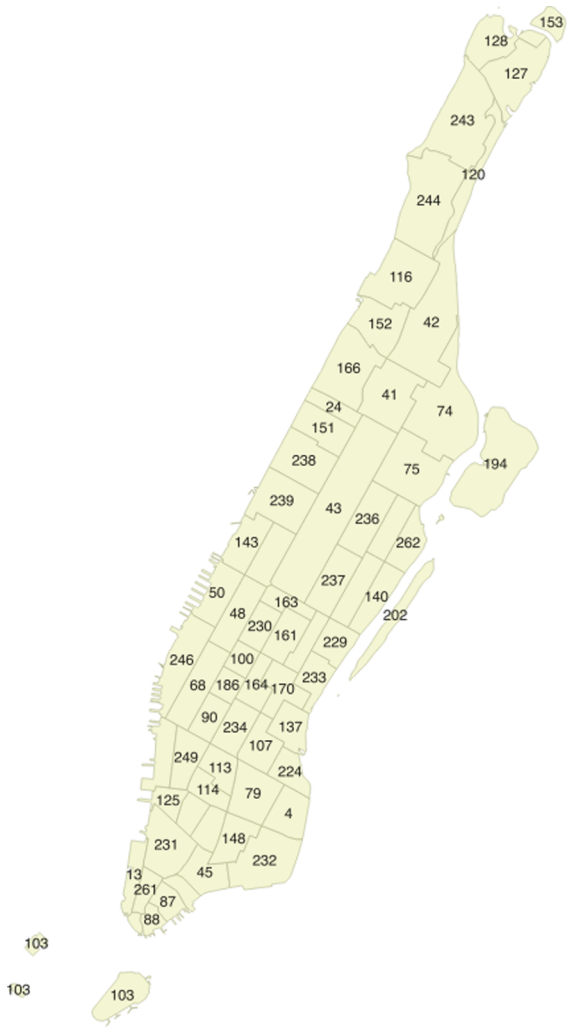

Fig. 3. Manhattan taxi zone partition.

in Fig. 4 throughout a day and across the month by respectively aggregating its hourly departure and arrival trips. Figs. 5 and 6 show the traffic patterns for two representative zones 116 and 161, of which the former is an uptown residential area and the latter represents the midtown business area. In each figure, the blue and green curves show respectively the average amount of departure and arrival trips for a particular hour of the day in June, and the upper and lower caps identify the 95% confidence interval. While both figures confirm the significant imbalance of traffic demand, there exists obvious spatial difference with respect the demand distribution. Zone 161, whilst the central business area, has significantly higher but more concentrated demands than zone 116 across a day. Temporally, zone 116's arrival peak is around midnight and departure peak is around 7 am, while zone 161 has an arrival peak at around 7 am and multiple departure peaks after 12 pm.

In order to reduce the computational burden of the simulation, we have the following three simplifications. First of all, we aggregate the taxi zones into larger zones that yields a smaller network. In particular, we partition the region into 8 service zones as shown in Fig. 7. Second, we divide the morning peak (6 am - 10 am) into 16 time intervals with fixed interval size of 15 min, and we calculate the average number of ride requests for every time interval on an average day with the data. Notice that in our simulation, the data only helps us come up with the daily demand distribution to reflect the "imbalance" nature in real traffic networks. Third, we assume by the end of each day, all vehicles will return to their origins so the initial number of vehicles in each zone will be the same at the beginning of different days. Note that the first two simplifications are solely for the purpose of reducing computational time and numerical validation of our approach. And the third assumption is to ensure that at the beginning of each episode (each day), the initial state would be the same or at least similar, which is essential for the convergence of total return in RL algorithms' learning process. Our methods, however, can be generalized to any size of network and time intervals given enough computational power.

Without loss of generality, all other parameters, such as travel times between any pair of zones and waiting costs, are set by the authors manually. Notice that the waiting time penalty is much higher than the penalty of repositioning empty vehicles so that we can achieve higher level of service for the customers. And our goal is to decide the number of vehicles to dispatch between each pair

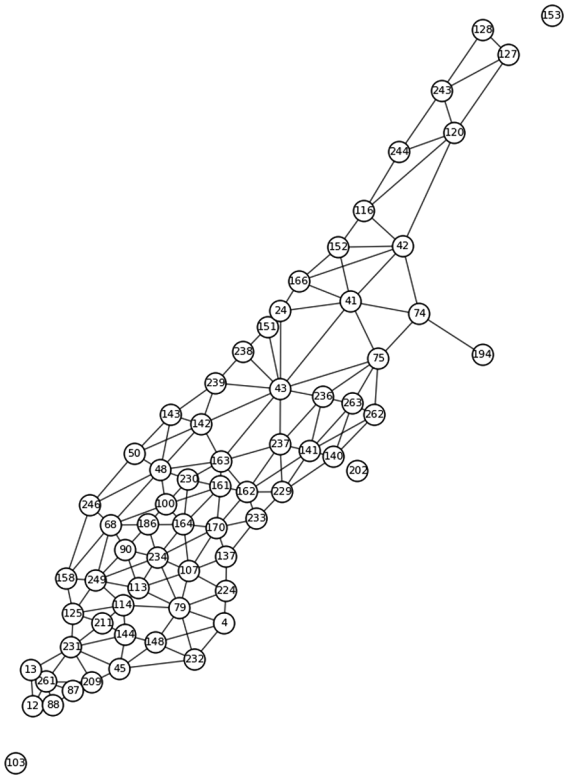

Fig. 4. Network by representing each zone with a node.

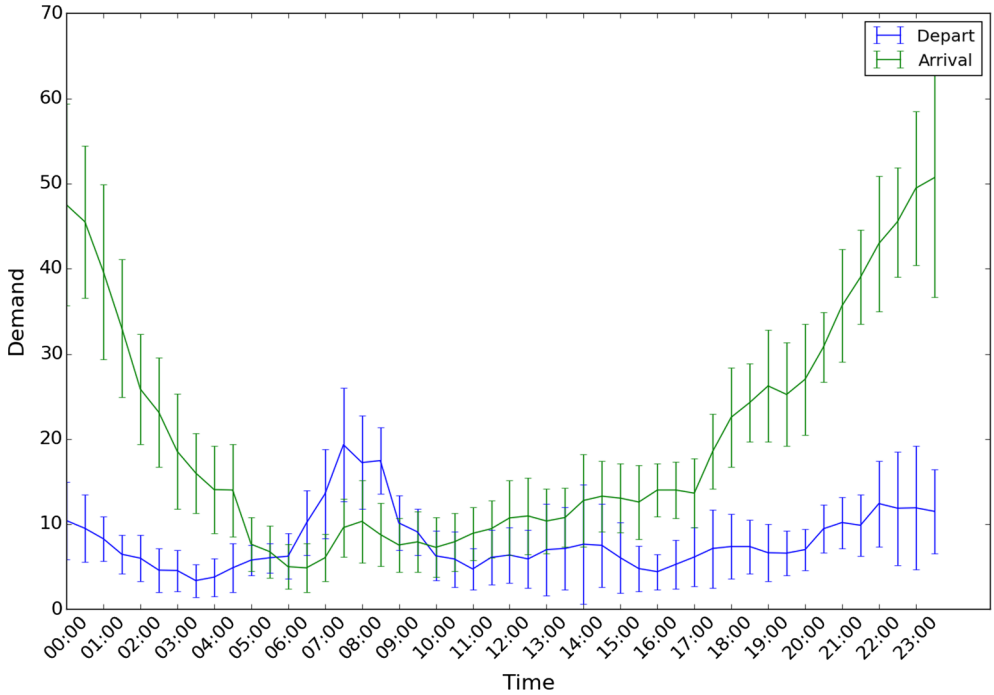

Fig. 5. Arrival and departure rates for zone 116.

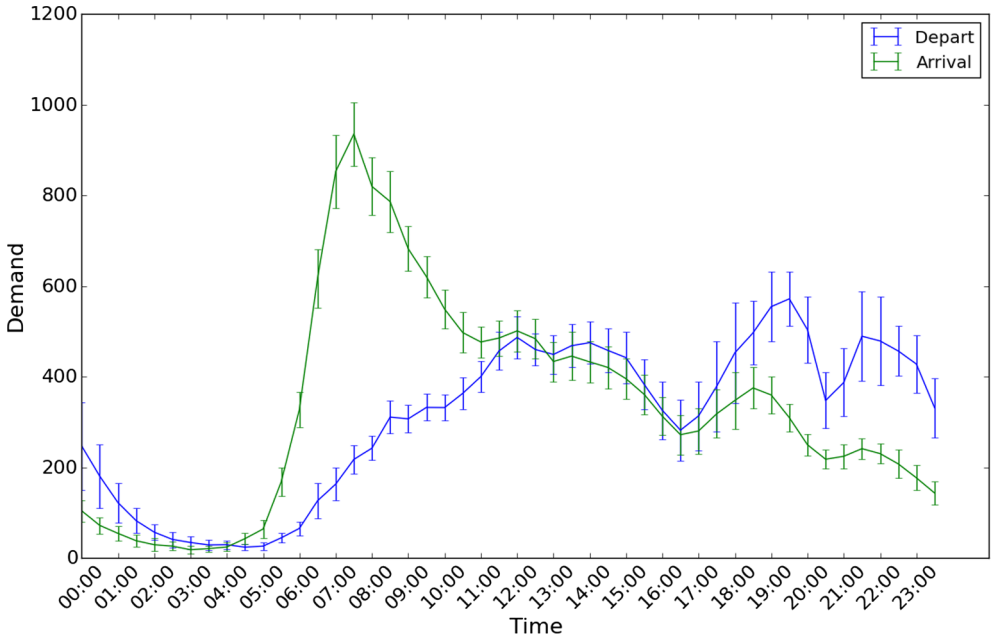

Fig. 6. Arrival and departure rates for zone 161.

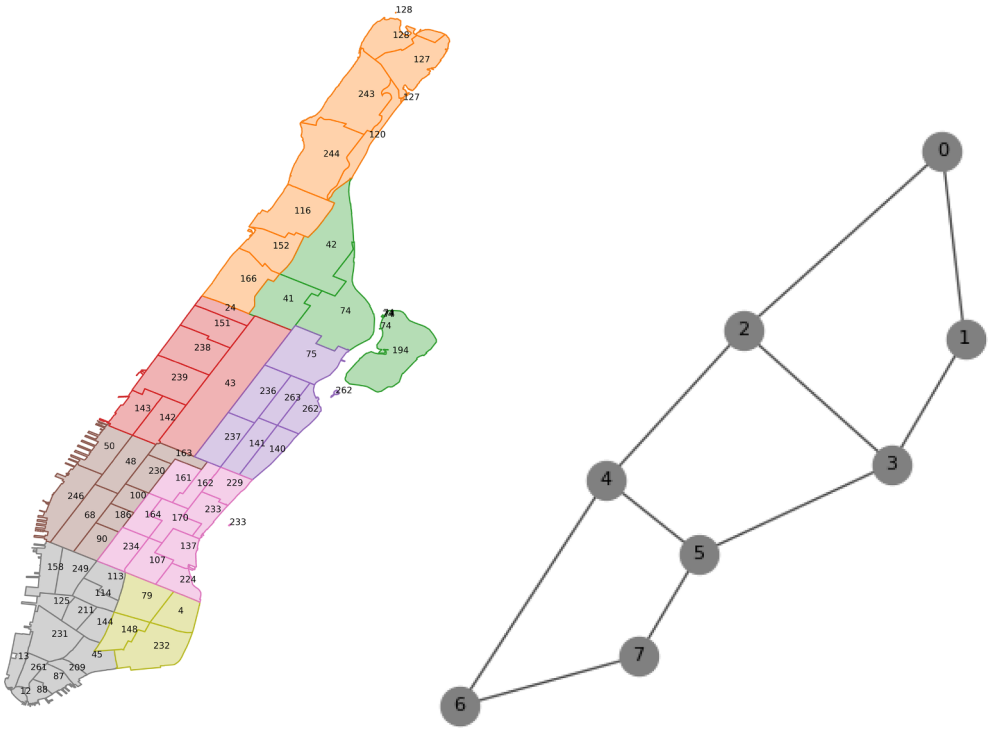

Fig. 7. Partition of Manhattan into 8 zones.

of zones at the beginning of each time interval.

Scenario specific settings – Scenario I

As we have discussed in Section 3.4, if we would like to provide higher level of service to the customers, we need to impose higher penalty on the waiting time of those passengers who have already waited for a long time. In this scenario where we distinguish two types of passengers, we set the threshold $\Omega = 1$ in Eq. (7), i.e. those who have waited more than $\Omega = 1$ time interval are treated as "impatient passengers" and others are "patient passengers". To compare the behaviors of the optimal policies with and without considering user priority, we have conducted two experiments using the proposed actor-critic algorithms. The first experiment

considers the user priority and therefore the waiting time penalty configuration has been set as $\delta^2_{ijt} = 15 \cdot \delta^1_{ijt}$: penalty on the waiting time of "impatient passengers" is much higher; the second experiment (base case), on the other hand, does not distinguish "impatient passengers" from "patient passengers" and therefore, we let $\delta^2_{ijt} = \delta^1_{ijt}$.

Scenario specific settings – Scenario II

In this scenario, we have determined three sets of parameters: (1) demand profile $\mathcal{D}_\lambda(\theta_1)$ and vehicle distributions $\mathcal{D}_n(\theta_2)$; (2) unit waiting time cost function $\delta_{ijt}(\omega)$; and (3) cancellation cost function $\eta_{ijt}(\omega)$.

First of all, we have assumed that the demand profile for each OD pair (i, j) at a specific time t follows a Gaussian distribution, of which the mean and variance are derived from the historical NYC TLC data discussed in the generic settings. Namely, $\lambda_{ijt} \sim \mathcal{N}(\mu_{ijt}, \sigma_{ijt})$ where μ_{ijt} and σ_{ijt} are historical means and standard deviations. Similarly, we have assumed that the initial number of vehicles follows a multinomial distribution where the probabilities of each zone are proportions to the aggregated historical demand at each zone.

Second, we set the parameters in Eqs. ()()()(8)–(10) such that the simulator can distinguish four different types of passengers: "patient passengers" who have only waited for one time interval, "impatient passengers" who have waited for respectively two and three time intervals, and passengers who would cancel their requests after waiting for more than three time intervals. Therefore, in Eqs. (8) and (9), we have set $m = 3$, $\Omega_1 = 1$, $\Omega_2 = 2$, $\Omega_3 = 3$, $\Omega_4 = 4$, $\alpha_1 = \delta^1_{ijt}$, $\alpha_2 = 5 \cdot \delta^1_{ijt}$, $\alpha_3 = 7 \cdot \delta^1_{ijt}$ to ensure a increasing marginal waiting cost (see Eq. (7) for the definition of δ^1_{ijt}). Accordingly, we set the cancellation rules in Eq. (10) as $\Phi = 4$, $\phi = 25 \cdot \delta^1_{ijt}$ so that a cancellation would receive a higher penalty than the waiting costs.

4.2. Model performance

In our case study, we have conducted a set of experiments, including a benchmark vanilla policy gradient algorithm, with different training configurations to see how different model specifications affect our learned policies. Our neural networks' architectures are summarized in Table 1. Other generic parameters, such as learning rate and trajectory batch size, have been set respectively to be 5×10^{-5} and 512 for all experiments. Last but not least, in order to compare the performance of the actor critic algorithm with the theoretical upper bound we have derived in Section 3.3, we set the travel demand to be deterministic in our simulator, i.e. from day to day there are a fixed number of passengers who need to travel between each pair of zones at a certain time of day. Thus we can solve for the optimal dispatching strategy based on the integer programming (IP) model we have formulated. A more general case with stochastic demand will be later discussed in Section 4.3.

After distributing all five experiments on an Amazon EC2 *p2. xlarge* instance for 30,000 epochs of training process, we first visualize the training process by plotting the average return at each epoch in Fig. 8. In the figure, experiment 1 fails to converge and experiment 5 just starts to reaching its plateau, while experiments 2, 3, and 4 all have plateaued out after 30,000 epochs. However, experiment 2 has a far better convergence rate than 3 and 4. In the meantime, the final converged values with their standard deviation and the training speeds per epoch for the five experiments suggest that with similar training speed, experiment 2 is able to train a better policy with higher average return. Those observations suggest that while the actor critic algorithm is obviously better than the vanilla policy gradient, a deeper neural network architecture performs better than a shallow neural network. Second, comparing the plateaued total rewards of experiment 2 with the theoretically optimal upper bound, the actor critic algorithm converged to around -5.02×10^5, while IP model reports the optimal value of -4.86×10^5 – the optimality gap is around 3.4%.

4.3. Policy robustness

To test the robustness of the performance of the proposed actor critic algorithm, we further allow stochasticity in the travel demand realization. In particular, we prepare two different travel demand profiles, which we term as outbound and inbound profile respectively, and on each day we randomly pick, with equal probability, one travel demand profile for the transportation network. While we keep the outbound demand profile the same as the one we use for experiments 1–5, we transpose its OD matrix and reduce the demand by 5% to serve as the inbound demand profile. This tries to mimic a lower traffic circumstances during weekends/holiday periods. Our actor-critic learner, however, has no prior information about this setup, and we train our algorithm using the same training configurations with experiment 2 shown in Table 1. At the same time, we use the proposed IP model in Section 3.3 to

Table 1
Training Configurations.

Exp.	Algorithm	Policy network architecture	Value network architecture	Convergence	Last epoch Return
1	Vanilla policy gradient	4 layers with size {128, 128, 128, 128}	None	Not converged	–
2	Actor critic	4 layers with size {128, 128, 128, 128}	4 layers with size {128, 128, 128, 128}	Fast convergence 1.91 s/epoch	$-5.02 \times 10^5 \pm 523.1$
3	Actor critic	3 layers with size {128, 128, 128}	3 layers with size {128, 128, 128}	Medium convergence 1.80 s/epoch	$-5.06 \times 10^5 \pm 488.9$
4	Actor critic	2 layers with size {128, 128}	2 layers with size {128, 128}	Slow convergence 1.74 s/epoch	$-5.12 \times 10^5 \pm 656.6$
5	Actor critic	1 layer with size {128}	1 layer with size {128}	Not converged 1.64 s/epoch	–

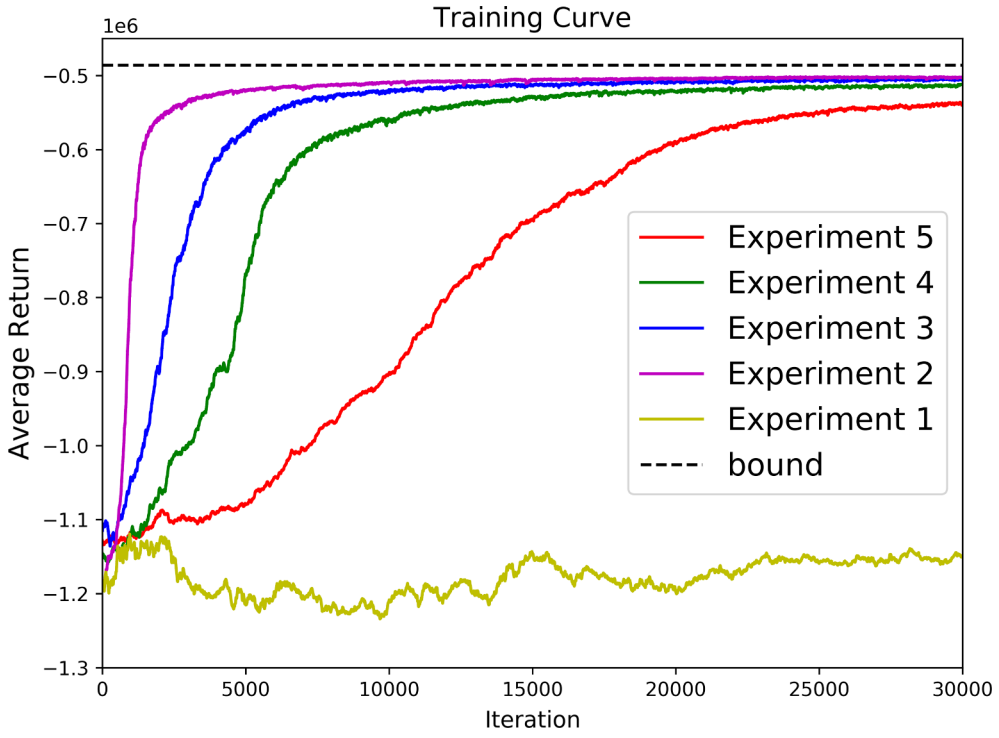

Fig. 8. Training Curve of Actor-Critic Method Under Deterministic Demand.

respectively solve the optimal returns under outbound and inbound demand profiles. Notice that the true optimal should be found by solving a stochastic integer program, and our IP approach only gives an upper and a lower bound of the optimal return. However, we leave the stochastic programming problem as a future work since it is not the main scope of the paper.

We compare the actor-critic training curve with the two bounds in Fig. 9. As can be seen, the optimal solution we get for outbound demand profile is around -4.86×10^5, and for inbound profile it is around -4.32×10^5. Our proposed actor critic algorithm, with the same network configuration as experiment 2 in Table 1, reports the final converged total rewards to about $-4.49 \times 10^5 \pm 409.7$, which is close to the upper bound. Also notice that the true optimal value should be between the converged return and the upper bound. Therefore, when the travel demand is stochastic and unknown to us beforehand, the actor critic method, although may not given the theoretical optimal, still provides satisfying results. In real traffic networks, however, since the network dynamics (demand,

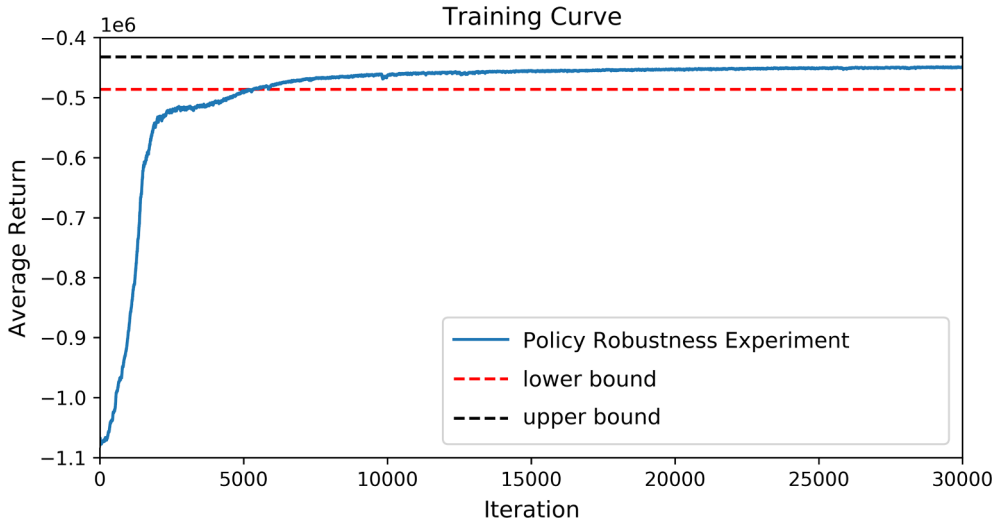

Fig. 9. Training Curve of Actor-Critic Method Under Stochastic Demand.

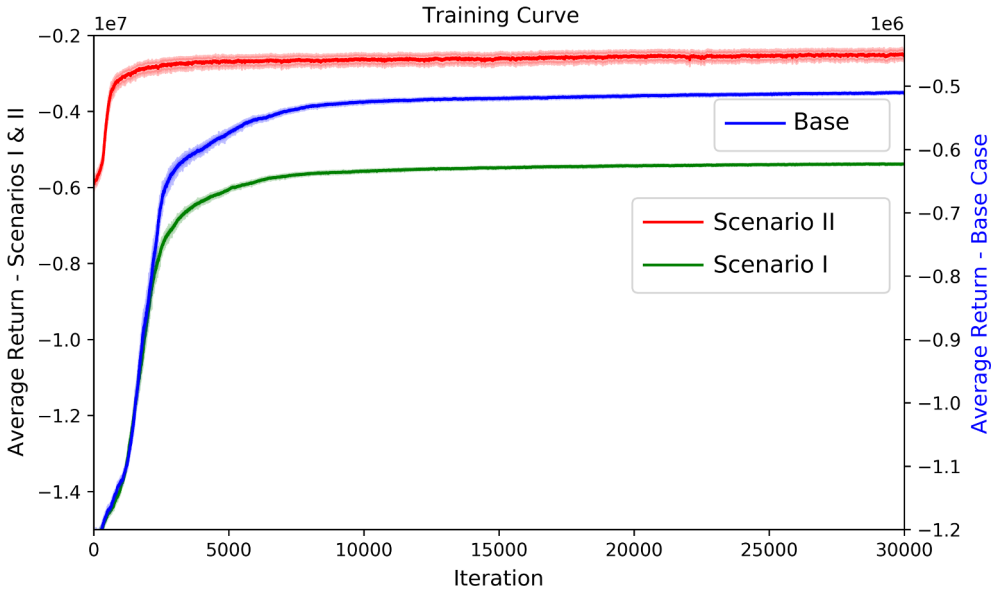

Fig. 10. Training Curve of the User Priority Scenarios.

travel time etc.) are highly complex and stochastic, establishing closed-form models to reflect such complicated dynamics would be challenging. In this case, the proposed model free reinforcement learning method (i.e., actor critic) is an efficient alternative way to solve for reliable and close-to-optimal solutions.

4.4. Consider user priority – Scenario I

In these experiments, we use the same training specifications as described in Section 4.2 and have trained our policy networks until convergence. Fig. 10 shows the average returns for our experiments as the training proceeded. For the sake of current interest, we only focus on the green and blue curves, which respectively represent the training processes for the experiments with (i.e., scenario I) and without (i.e., base case) considering the user priority, and the green and blue curves use the left and right hand side y axes, respectively. Though plateaued at different values, which is due to the different waiting cost settings, both experiments achieve convergence at about the same rate. Second, We keep track of the total waiting times of all the passengers and the "impatient passengers". The results are summarized in Table 2, where the numbers after the plus/minus signs are the standard deviations of the corresponding waiting time. As can be seen, when we consider user priority and choose to first serve those "impatient passengers", the total waiting time of all the passengers is around 5.25×10^4, which is 4% higher than the case without user-priority. However, the waiting time of those "impatient passengers" is significantly smaller (5.42×10^4 vs 5.74×10^4), which matches our expectation. If we do not differentiate between passengers based on their waiting time, the algorithm learns an optimal policy that would benefit the system the most, or "system optimal", even if it makes "impatient passengers" wait longer. However when we want to guarantee level of service and grant priorities to those "impatient passengers", our actor critic algorithm, though still learns a case-specific optimal policy, yields a "sub-optimal" policy that sacrifices the system optimality but drastically reduces those extreme long waiting time.

To further understand the spatial effect of considering user-priority, we run (feed-forward pass) both learned optimal dispatching policies with and without user priority on the same typical day, and compare the average waiting time per each OD pair throughout a day. The results are summarized in the heat maps shown in Fig. 11, where the first two subplots (a) and (b) show respectively the cases with and without user priority, subplot (c) shows the difference in average waiting time (subtract subplot 1 from subplot 2), and subplot (d) shows the total OD demand profile for the evaluation day. For each heat map, the two axes represent the indices of the simplified network nodes shown in Fig. 4. Both waiting time heat maps (a and b) confirm that the average waiting time in high-demand OD pair (e.g., zone 4 to zone 5) is significantly lower than that in low-demand OD pair (e.g., zone 0 to zone 1), which

Table 2
Waiting time comparison with and without user-priority.

	Total waiting time of all passengers	Total waiting time of impatient passengers
With user-priority ($\delta^2_{ijt} = 15 \cdot \delta^1_{ijt}$)	$5.25 \times 10^4 \pm 41.7$	$3.42 \times 10^4 \pm 32.0$
Without user-priority ($\delta^2_{ijt} = \delta^1_{ijt}$)	$5.04 \times 10^4 \pm 44.6$	$3.74 \times 10^4 \pm 61.6$

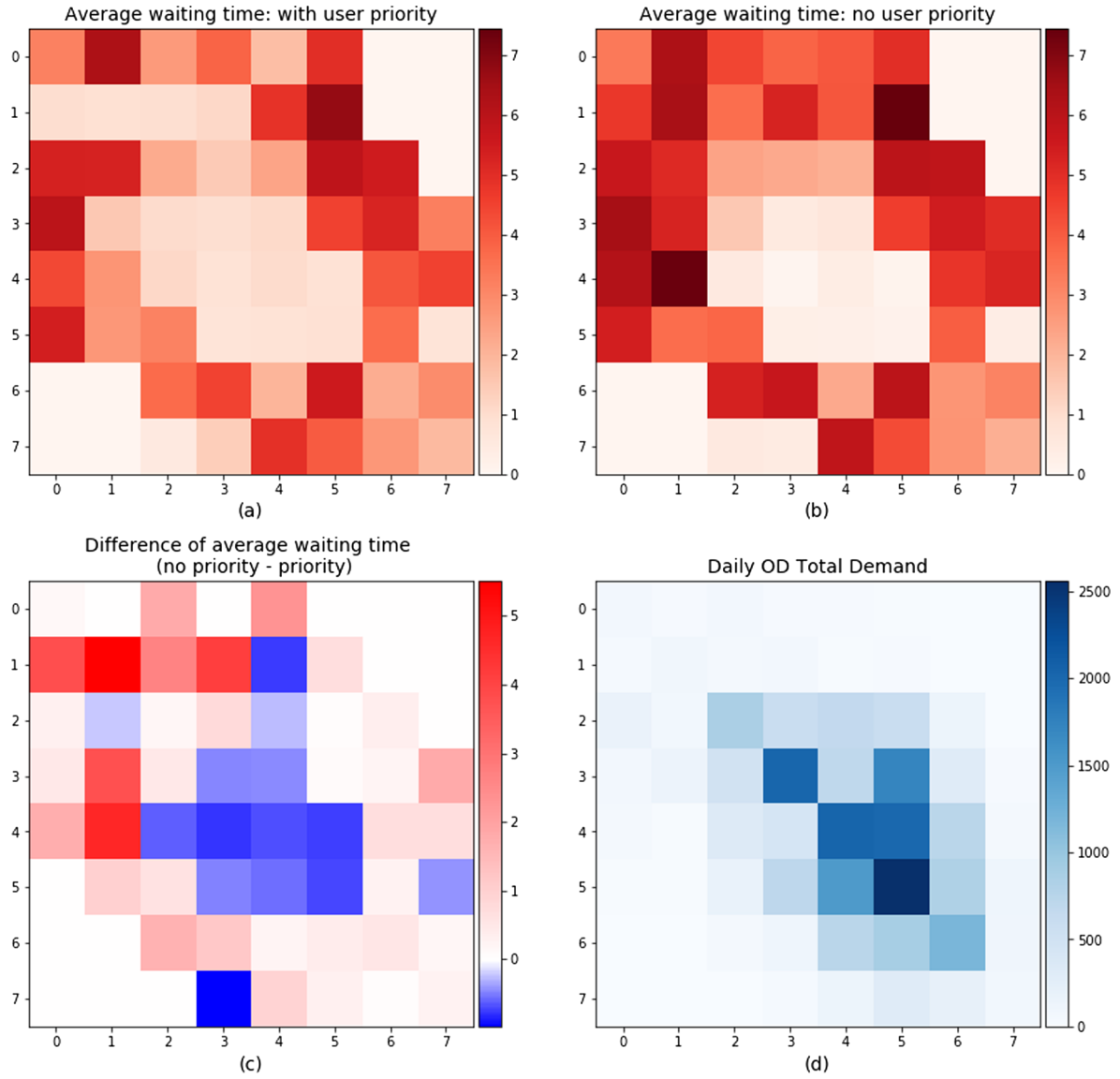

Fig. 11. Heat maps of average passenger waiting time and OD demand.

suggests that the learned policies successfully identify these "hot" areas and dispatch more vehicles to those locations to reduce the waiting time. However, those passengers in the less popular areas would suffer from longer waiting time. And such unfairness can be mitigated by considering user priority. More specifically, the first heat map shows a smaller standard deviation (1.97 vs. 2.39) with respect to the waiting time of all OD pairs, which implies that by considering user priority, our proposed algorithm produces a more equitable policy that reduces the variance of waiting time across passengers and zones. This conjecture is further proved by subplot (c) of Fig. 11, where the average waiting times for less-demanding OD pairs (e.g., zone 1 to zones 0 – 3) are significantly higher in the case without considering user priority, and vice versa for high-demanding OD pair (e.g., zone 4 - zone 5).

4.5. Consider user priority – Scenario II

Using the same training specification as in the first scenario, our policy converges after 30,000 training epochs. We first observe that in Fig. 10, of which the red curve shows the training process for the scenario II, the experiment converges after around 10,000 epochs. Second, for each experiment in the figure, we have plotted the standard deviation of returns within each epoch – each value in the plot is surrounded by a three-standard-deviation band. We notice that the red curve demonstrates higher training variances than the other two experiments, which is due to the fact that scenario II incorporates stochasticity including the Gaussian distributed demand profile and multinomial distributed initial vehicle numbers.

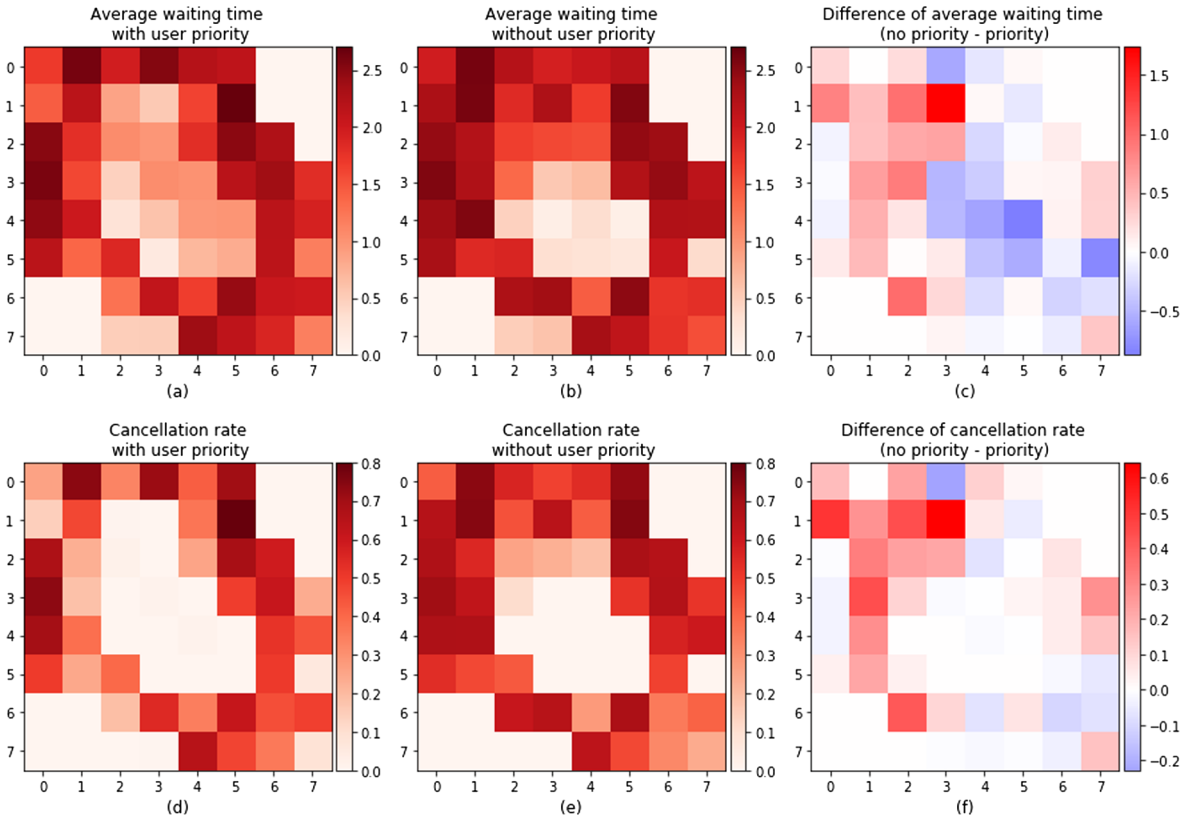

Fig. 12. Heat maps of average passenger waiting time and cancellation rate.

Third, we run the feed forward pass using the learned optimal policies with and without considering the user priority, and compare the resulting average passenger waiting time and cancellations. In the feed forward run, we control the inputs to both feed forward runs the same, including the vehicle distributions and the demand profiles (we use the same OD demand profile as in Fig. 11(d)). Notice that we also let both runs to have passengers cancel their requests even though the policy without user priority does not explicitly consider such case during the training. However, such controlled experiments allow us to address the question that in a realistic world, would a policy with the treatment of user priority dominate the one without? And if so, by how much? Fig. 12 summarizes the results. In the figure, subplots (a) and (b) demonstrate the average waiting time with and without considering the user priority, respectively. Subplot (c) shows the difference between (a) and (b) in which positive values are shown in reds and negative values are shown in blues. Subplots (d) and (e) show the cancellation rate, which is defined as the ratio of the accumulated cancelled requests by each OD pair to each OD demand across a day. Subplot (f) illustrates the difference between the cancellation rates. We first compare the subplots (d) – (f) and observe that the cancellation rates for both policies are significantly lower in the central areas, or "hot" areas, than those less popular areas such as $1 \rightarrow 5$. On the other hand, however, the policy with user priority achieves noticeably better cancellation rates in many OD pairs than the other policy, of which the effect is mostly announced in the less popular OD pairs – the largest difference is as high as 60%. Then we compare subplots (a) – (c). While the implications for the average waiting time largely agree with those concluded from scenario I in Section 4.4, the difference of the two policies are less significant: the policy with user priority does produce a more fair dispatching strategy with lower waiting times in less-demanding areas but the magnitudes are smaller. However, this is caused by that passengers who waited for more than four time intervals cancel their requests. In fact, since the policy with user priority has significantly lower cancellation rate, it serves more passengers than the policy without, while still being able to achieve a comparable waiting time. Therefore, such policy is deemed as a superior policy that balances well with respect to fairness and level of service.

5. Conclusion

In this paper, we present a deep reinforcement learning (RL) approach for the problem of dispatching autonomous vehicles for taxi services. In particular, we propose an actor-critic framework with deep neural networks as approximations for both the actor and critic functions. Second, we derive the theoretical upper bound of the total costs if we assume the dynamics of the system are deterministic and known to us beforehand. Third, we implement our actor-critic method and apply it to a simplified transportation network based on the New York yellow taxi services. We conduct a set of experiments to compare the effectiveness of the proposed actor-critic algorithm with deep neural networks against vanilla policy gradient method and shallow neural networks, which is a surrogate of the approximate

dynamic programming method. The results suggest that the proposed algorithm outperforms the other algorithms with respect to convergence rate and quality. Fourth, we test the robustness of the proposed algorithm by allowing stochastic traffic demand profile, the experiment suggests that our RL method can always converge to a value close to the true optimal. Lastly, we have investigated two more realistic scenarios where we have considered user priority, passenger cancellations, and stochasticity to both take account the extreme passenger waiting time and the random nature of demand and vehicular distributions. Our case study shows that with the consideration of user priority, the total waiting time of all passengers will increase, however the total waiting time of "impatient passenger" reduces, when comparing to the case without user priority. We also show that with user priority, the optimal policy is more equitable across the passengers and OD pairs. In the scenario where we consider cancellation, we conclude that the policy with user priority is more superior than the policy without with respect to user fairness, cancellation rate, and level of service.

This paper leaves a lot of extensions for future research. First of all, our current experiments are conducted on a rather simplified transportation network, however, it has long been a challenge to adapt a *centralized policy* to large scale experiments with deep feed forward neural networks due to the growing state/action space. However, one may tackle this issue by embedding the space and action into recurrent neural networks (RNN) (Hochreiter and Schmidhuber, 1997) or graph neural networks (GNN) (Scarselli et al., 2008). Besides, one can also incorporate the multi-agent RL framework to give a *decentralized* solution, as in Lin et al. (2018), to balance the trade off between system optimality and running efficiency. Second, our current experiments only considered single mode (autonomous vehicles), while in the near future the vehicle fleet will most likely be made up of a mixture of human-driven vehicles and autonomous vehicles. Therefore, we can adapt our framework to a mixed fleet case by incorporating a larger state and action space. Third, for profitable companies such as Uber and Lyft, our algorithm could also combine the surge pricing strategy for the human-driving vehicles with the dispatch strategy for the autonomous vehicles. Fourth, incorporating explicit demand forecasting into the proposed RL framework (as in the model predictive control framework) could be another interesting extension, and we may expect this extra information to improve the system level expected returns and convergence speed. Lastly, we could consider the effect of ride sharing on our dispatch strategy. Currently we assume that different trips are independent from each other and should be served with different vehicles. However, if we allow ride sharing and trip matching, we would expect a more cost-efficient policy to be learned by dispatching less vehicles to regions where there are more trips matched together.

References

Abdulhai, Baher, Pringle, Rob, Karakoulas, Grigoris J., 2003. Reinforcement learning for true adaptive traffic signal control. J. Transport. Eng. 129 (3), 278–285.

Banerjee, Siddhartha, Riquelme, Carlos, Johari, Ramesh, 2015. Pricing in ride-share platforms: a queueing-theoretic approach.

Braverman, Anton, Dai, Jim G., Liu, Xin, Ying, Lei, 2019. Empty-car routing in ridesharing systems. Oper. Res. 67 (5), 1437–1452.

Chen, Le, Mislove, Alan, Wilson, Christo, 2015. Peeking beneath the hood of uber. In: Proceedings of the 2015 Internet Measurement Conference. ACM, pp. 495–508.

Grondman, Ivo, Busoniu, Lucian, Lopes, Gabriel A.D., Babuska, Robert, 2012. A survey of actor-critic reinforcement learning: standard and natural policy gradients. IEEE Trans. Syst., Man, Cybernet., Part C (Appl. Rev.) 42 (6), 1291–1307.

Hochreiter, Sepp, Schmidhuber, Jürgen, 1997. Long short-term memory. Neural Comput. 9 (8), 1735–1780.

Iglesias, Ramon, Rossi, Federico, Wang, Kevin, Hallac, David, Leskovec, Jure, Pavone, Marco, 2018. Data-driven model predictive control of autonomous mobility-on-demand systems. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp. 1–7.

Iglesias, Ramon, Rossi, Federico, Zhang, Rick, Pavone, Marco, 2019. A bcmp network approach to modeling and controlling autonomous mobility-on-demand systems. Int. J. Robot. Res. 38 (2–3), 357–374.

Li, Li, Lv, Yisheng, Wang, Fei-Yue, 2016. Traffic signal timing via deep reinforcement learning. IEEE/CAA J. Autom. Sin. 3 (3), 247–254.

Lin, Kaixiang, Zhao, Renyu, Zhe, Xu, Zhou, Jiayu, 2018. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1774–1783.

Mao, Chao, Shen, Zuojun, 2018. A reinforcement learning framework for the adaptive routing problem in stochastic time-dependent network. Transport. Res. Part C: Emerg. Technol. 93, 179–197.

Miao, Fei, Han, Shuo, Lin, Shan, Stankovic, John A., Zhang, Desheng, Munir, Sirajum, Huang, Hua, He, Tian, Pappas, George J., 2016. Taxi dispatch with real-time sensing data in metropolitan areas: a receding horizon control approach. IEEE Trans. Autom. Sci. Eng. 13 (2), 463–478.

Miller, Justin, How, Jonathan P., 2017. Predictive positioning and quality of service ridesharing for campus mobility on demand systems. In: 2017 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp. 1402–1408.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, Riedmiller, Martin, 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

Pavone, Marco, Smith, Stephen L., Frazzoli, Emilio, Rus, Daniela, 2012. Robotic load balancing for mobility-on-demand systems. Int. J. Robot. Res. 31 (7), 839–854.

Prashanth, L.A., Bhatnagar, Shalabh, 2011. Reinforcement learning with function approximation for traffic signal control. IEEE Trans. Intell. Transp. Syst. 12 (2), 412–421.

Scarselli, Franco, Gori, Marco, Tsoi, Ah Chung, Hagenbuchner, Markus, Monfardini, Gabriele, 2008. The graph neural network model. IEEE Trans. Neural Networks 20 (1), 61–80.

Shou, Zhenyu, Di, Xuan, Ye, Jieping, Zhu, Hongtu, Hampshire, Robert, 2019. Where to find next passengers on e-hailing platforms?-a markov decision process approach. arXiv preprint arXiv:1905.09906.

Sutton, Richard S, Barto, Andrew G, 2018. Reinforcement Learning: An Introduction. MIT press.

Sutton, Richard S., McAllester, David A., Singh, Satinder P., Mansour, Yishay, 2000. Policy gradient methods for reinforcement learning with function approximation. In: Advances in Neural Information Processing Systems, pp. 1057–1063.

Volkov, Mikhail, Aslam, Javed, Rus, Daniela, 2012. Markov-based redistribution policy model for future urban mobility networks. In: Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on. IEEE, pp. 1906–1911.

Walraven, Erwin, Spaan, Matthijs T.J., Bakker, Bram, 2016. Traffic flow optimization: a reinforcement learning approach. Eng. Appl. Artif. Intell. 52, 203–212.

Williams, Ronald J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. 8 (3–4), 229–256.

Zha, Liteng, Yin, Yafeng, Zhengtian, Xu., 2018. Geometric matching and spatial pricing in ride-sourcing markets. Transport. Res. Part C: Emerg. Technol. 92, 58–75.

Zhang, Rick, Pavone, Marco, 2016. Control of robotic mobility-on-demand systems: a queueing-theoretical perspective. Int. J. Robot. Res. 35 (1–3), 186–203.

Zhang, Rick, Rossi, Federico, Pavone, Marco, 2016. Model predictive control of autonomous mobility-on-demand systems. In: 2016 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp. 1382–1389.

Zhu, Feng, Ukkusuri, Satish V., 2014. Accounting for dynamic speed limit control in a stochastic traffic environment: a reinforcement learning approach. Transport. Res. Part C: Emerg. Technol., vol. 41, pp. 30–47.