Proceedings of Machine Learning Research vol 211:1–13, 2023 5th Annual Conference on Learning for Dynamics and Control

Hybrid Multi-agent Deep Reinforcement Learning for Autonomous Mobility on Demand Systems

Tobias Enders TOBIAS.ENDERS@TUM.DE
Technical University of Munich, Germany

James Harrison JAMESHARRISON@GOOGLE.COM
Google Research, Brain Team, California, USA

Marco Pavone PAVONE@STANFORD.EDU
Stanford University, California, USA

Maximilian Schiffer SCHIFFER@TUM.DE
Technical University of Munich, Germany

Editors: N. Matni, M. Morari, G. J. Pappas

Abstract

We consider the sequential decision-making problem of making proactive request assignment and rejection decisions for a profit-maximizing operator of an autonomous mobility on demand system. We formalize this problem as a Markov decision process and propose a novel combination of multi-agent Soft Actor-Critic and weighted bipartite matching to obtain an anticipative control policy. Thereby, we factorize the operator's otherwise intractable action space, but still obtain a globally coordinated decision. Experiments based on real-world taxi data show that our method outperforms state of the art benchmarks with respect to performance, stability, and computational tractability.

Keywords: hybrid learning and optimization, multi-agent learning, deep reinforcement learning, autonomous mobility on demand

1. Introduction

Mobility on demand (MoD) systems, in which a fleet of free-floating vehicles serves customers' ad hoc requests for point-to-point transportation, have transformed urban mobility in recent years. Companies like Uber, Lyft, and DiDi, made MoD more accessible compared to taxi-based ride hailing services. Autonomous vehicles will further transform MoD systems; besides much lower prices, a major benefit of autonomous MoD (AMoD) is its improved potential for advanced control strategies, as a central operator obtains full control over the entire fleet. This transformation changes the fleet operator's control problem substantially: MoD operators focus primarily on revenue maximization, as human drivers' income is (almost) a fixed cost that dominates mileage-dependent operational cost. Contrarily, AMoD operators focus on the maximization of their operating profit, because operational costs dominate their total cost balance. In this context, the central operator can leverage its full knowledge about the system state and fleet control to make improved (proactive) dispatching decisions, i.e., request to vehicle assignment and rejection, to maximize its profit.

Since an operator does not have profound knowledge about future trip requests, it faces an online decision-making problem in a stochastic environment. Hence, it is promising to apply deep reinforcement learning (DRL) to this problem. However, AMoD systems entail many vehicles and trip

© 2023 T. Enders, J. Harrison, M. Pavone & M. Schiffer.

requests, such that an operator's action space is very large and possibly time-varying as the number of requests changes over time. It is thus infeasible to apply off-the-shelf single-agent DRL. To solve this problem, we propose a novel combination of a multi-agent DRL algorithm with optimization-based centralized decision-making through weighted bipartite matching. This hybrid algorithm combines the advantages of multi-agent approaches, DRL, and combinatorial optimization.

1.1. Related Work

To keep this literature overview concise, we focus on literature for controlling (autonomous) MoD systems in the following. For a review of multi-agent DRL, we refer to Gronauer and Diepold (2022) and further elaborate on how we build on the multi-agent DRL literature in Section 3.

Classical approaches for dispatching and explicit rebalancing decisions focused on greedy or hand-crafted feature-based policies (Liao, 2003; Zhang et al., 2017), queueing theoretical approaches (Zhang and Pavone, 2016), and model predictive control (MPC) (Alonso-Mora et al., 2017).

Recently, many works applied DRL in the context of (autonomous) MoD, often including or purely focusing on explicit rebalancing (e.g., Jiao et al., 2021; Gammelli et al., 2021; Skordilis et al., 2022; Liang et al., 2022). Contrarily, other works focused on DRL for non-myopic dispatching, which entails an implicit rebalancing decision that avoids additional costs due to empty driving. Early approaches (cf. Xu et al., 2018; Wang et al., 2018) were shown to be inferior to at least one of the subsequent works: Li et al. (2019) proposed a mean field multi-agent actor-critic algorithm. Tang et al. (2019) used bipartite matching based on learned V-values. Zhou et al. (2019) combined a multi-agent Deep Q-Network with minimization of the Kullback-Leibler divergence (KL-divergence) between the vehicle and the request distribution. Finally, Sadeghi Eshkevari et al. (2022) described how DiDi recently rolled out DRL for dispatching in practice.

Since these works strive to improve the operations of today's MoD systems, they aim at maximizing the drivers' revenue or the number of orders served, rather than at maximizing the profit of an AMoD system. While Xu et al. (2018); Wang et al. (2018); Tang et al. (2019); Sadeghi Eshkevari et al. (2022); Liang et al. (2022) also use a combination of multi-agent DRL and weighted matching, they all employ value-based algorithms. Contrarily, we use an actor-critic algorithm, enabling more advanced strategies to mitigate problems arising from multi-agent learning, in particular, decentralized actors with centralized critics, see Section 3. By using Soft Actor-Critic (SAC) (Haarnoja et al., 2018), we can nevertheless benefit from the improved sample-efficiency of off-policy algorithms.

1.2. Contributions

To the best of our knowledge, we are the first to consider the problem of making proactive dispatching decisions for a profit-maximizing AMoD system operator with DRL. We propose a novel method that combines multi-agent SAC with centralized final decision-making through weighted matching. We perform experiments based on real-world data and, similar to related works, benchmark our method against a greedy policy. In addition, we are the first to compare our method against an MPC approach. We show that our method outperforms the greedy policy on all instances by up to 5%. Moreover, we outperform the MPC approach in most cases. Our DRL method shows a significantly more stable performance across varying instances, while MPC may perform arbitrarily bad—in single cases up to 60% worse than the greedy policy. An extended version of this paper, including an appendix, is available at `https://arxiv.org/abs/2212.07313`. Our code can be found at `https://github.com/tumBAIS/HybridMADRL-AMoD`.

2. Problem Formulation: Markov Decision Process

We consider a profit-maximizing operator who centrally controls a fixed-size fleet of vehicles to serve customer trip requests revealed over time within an operating area. The operator can accept or reject requests and dispatches accepted requests to vehicles. These decisions must be made in real-time and immediately, i.e., the operator cannot defer requests to a later time step, as customers are not willing to wait for feedback. If the operator accepts a request, customers must be picked up within a known maximum waiting time $\omega^{\max} \in \mathbb{N}_0$ after the request was placed. We formalize this control problem as a Markov decision process (MDP) as follows.

Preliminaries. We consider a discrete time horizon $\mathcal{T} = \{0, 1, ..., T\}$. During one time step, multiple requests can enter the system. The operator makes one decision per time step for multiple requests simultaneously, which allows to optimize over a batch of requests. We represent the operating area as a graph $G = (V, E)$ with weight vectors ${}^e\boldsymbol{w} = \left({}^e w^1, {}^e w^2\right) \in \mathbb{R}_{>0} \times \mathbb{N}$, denoting the distance (${}^e w^1$) of and the time steps (${}^e w^2$) to traverse an edge $e \in E$. The nodes of G may represent, e.g., the centers of zones into which the operating area is divided.

States. We describe the system state at time $t \in \mathcal{T}$ by $\boldsymbol{S}_t = \left(t, \left({}^t\boldsymbol{r}^i\right)_{i\in\{1,...,R_t\}}, \left(\boldsymbol{k}_t^j\right)_{j\in\{1,...,K\}}\right)$, with R_t being the variable number of new requests ${}^t\boldsymbol{r}^i$, $i \in \{1, ..., R_t\}$, at time step t, and K vehicles $\boldsymbol{k}_t^j$, $j \in \{1, ..., K\}$. A request $\boldsymbol{r} = (\omega, o, d)$ consists of a waiting time $\omega \in \mathbb{N}_0 \cup \emptyset$, an origin $o \in V$, and a destination $d \in V \setminus \{o\}$; ω tracks the elapsed time from request placement to pickup, where we set $\omega \leftarrow \emptyset$ at pickup. We denote a vehicle by $\boldsymbol{k} = (v, \tau, \boldsymbol{r}^1, \boldsymbol{r}^2)$, with position $v \in V$ and the number of time steps $\tau \in \mathbb{N}_0$ left to reach this position. Here, v can either be the current node if the vehicle idles or the next node that will be reached if the vehicle travels. Furthermore, slightly abusing notation, a vehicle can have at most two assigned requests $\boldsymbol{r}^1, \boldsymbol{r}^2$. Assigning more requests to one vehicle is unreasonable for realistic trip lengths and maximum waiting times. We denote the position of vehicle $\boldsymbol{k}_t^j$ by ${}^j v_t$ and denote other components of the vehicle vector likewise.

Actions. The action space describing feasible decisions of the operator is

$$\begin{aligned}\mathcal{A}\left(\boldsymbol{S}_t\right) = \Bigg\{ \left(a_t^1, ..., a_t^{R_t}\right) \Bigg|\, & a_t^i = 0 \ \vee \ \left(a_t^i = j \in \{1, ..., K\} \ \wedge \ {}^j\boldsymbol{r}_t^2 = \emptyset\right) \ \forall\, i \in \{1, ..., R_t\}, \\ & \sum_{i=1}^{R_t} \mathbb{1}\left(a_t^i = j\right) \leq 1 \ \ \forall j \in \{1, ..., K\} \Bigg\}. \end{aligned} \tag{1}$$

The operator can take one decision a_t^i per request ${}^t\boldsymbol{r}^i$, $i \in \{1, ..., R_t\}$, either rejecting it ($a_t^i = 0$), which means that the request leaves the system, or assigning it to vehicle $\boldsymbol{k}^j$ ($a_t^i = j$), which is only possible if the vehicle does not already have two assigned requests, i.e., if ${}^j\boldsymbol{r}_t^2 = \emptyset$ holds. The final condition in (1) implies that at most one new request is assigned to each vehicle in each time step, which is a realistic simplification facilitating the application of a matching algorithm. The central operator's action space size is of order $(K + 1)^{R_t}$.

Transitions. We first describe the action-dependent transition from the pre-decision to post-decision state. Then, we describe the transition from the post-decision state to the next pre-decision state, which is independent of the action and only determined by the system dynamics.

A reject decision has no impact on the state. When $a^i = j$, we add the request to the vehicle state, i.e., if ${}^j\boldsymbol{r}^1 = \emptyset$, then ${}^j\boldsymbol{r}^1 \leftarrow {}^t\boldsymbol{r}^i$, and ${}^j\boldsymbol{r}^2 \leftarrow {}^t\boldsymbol{r}^i$ otherwise.

The following transitions apply to all vehicles: if the vehicle picks up a customer, i.e., if $\boldsymbol{r}^1 \neq \emptyset \ \wedge \ \tau = 0 \ \wedge \ v = o\left(\boldsymbol{r}^1\right)$, where $o\left(\boldsymbol{r}^1\right)$ denotes the origin of request $\boldsymbol{r}^1$, then $\omega\left(\boldsymbol{r}^1\right) \leftarrow \emptyset$. If the vehicle moves between two nodes, i.e., if $\tau > 0$, then $\tau \leftarrow \tau - 1$. If the vehicle is at a node but moves to serve a request, i.e., if $\tau = 0 \ \wedge \ \boldsymbol{r}^1 \neq \emptyset$, then v is replaced by the next node v' on the vehicle's route to serve the request, going to origin or from origin to destination, and $\tau \leftarrow {}^{(v,v')}w^2 - 1$. If a vehicle drops off a customer before the next decision is made, i.e., if $\boldsymbol{r}^1 \neq \emptyset \wedge \omega\big(\boldsymbol{r}^1\big) = \emptyset \wedge \tau = 0 \wedge v = d\big(\boldsymbol{r}^1\big)$, we shift requests: $\boldsymbol{r}^1 \leftarrow \boldsymbol{r}^2$ and $\boldsymbol{r}^2 \leftarrow \emptyset$. We increment the waiting times $\omega \neq \emptyset$ of requests that have not been picked up yet, i.e., $\omega \leftarrow \omega + 1$, where ω refers to $\omega\big(\boldsymbol{r}^1\big)$ and/or $\omega\big(\boldsymbol{r}^2\big)$. Moreover, independent of the vehicles' states, customers place new requests, i.e., $\big({}^t\boldsymbol{r}^i\big)_{i\in\{1,\ldots,R_t\}}$ is replaced by $\big({}^{t+1}\boldsymbol{r}^i\big)_{i\in\{1,\ldots,R_{t+1}\}}$. Note that we do not know the underlying time-dependent probability distribution which generates new requests, but we can simulate the resulting requests by replaying historic data. We assume that the new requests arrive independently of the state-action history, such that the Markov property holds. Finally, $t \leftarrow t+1$.

Rewards. Since the operator maximizes its profit and fixed costs are independent of the control problem, our reward function focuses on the operating profit, which is the revenue from serving requests minus operational costs, e.g., for fuel and maintenance. The operator obtains the revenue for a request $\boldsymbol{r}$ when a vehicle picks up the request within the maximum waiting time. The revenue is given by a function $\mathrm{rev}(\boldsymbol{r}) \in \mathbb{R}_{>0}$, representing the operator's pricing model. For improved readability, we express the profit components as functions of the post-decision state $\boldsymbol{S}_{t^+}$ and write t for t^+. Then, the total revenue at time t is

$$\mathrm{Rev}(\boldsymbol{S}_t) = \sum_{j=1}^{K} \mathbb{1}\Big({}^j\boldsymbol{r}_t^1 \neq \emptyset \ \wedge \ {}^j\tau_t = 0 \ \wedge \ {}^jv_t = o\big({}^j\boldsymbol{r}_t^1\big) \ \wedge \ \omega\big({}^j\boldsymbol{r}_t^1\big) \leq \omega^{\max}\Big) \cdot \mathrm{rev}\big({}^j\boldsymbol{r}_t^1\big)\,.$$

When a vehicle starts to move from v to v', the operator incurs operational costs $c \in \mathbb{R}_{>0}$ per distance unit, as commonly assumed (see, e.g., Bösch et al., 2018). Thus, the total cost at time t is

$$\mathrm{Cost}(\boldsymbol{S}_t) = c \cdot \sum_{j=1}^{K} \mathbb{1}\Big({}^j\tau_t = 0 \ \wedge \ {}^j\boldsymbol{r}_t^1 \neq \emptyset\Big) \cdot {}^{({}^jv_t,\, {}^jv_t')}w^1.$$

The total profit at time t^+ is $\mathrm{Profit}(\boldsymbol{S}_{t^+}) = \mathrm{Rev}(\boldsymbol{S}_{t^+}) - \mathrm{Cost}(\boldsymbol{S}_{t^+})$. Note that $\boldsymbol{S}_{t^+}$ is a function of $\boldsymbol{S}_t$ (pre-decision) and $\boldsymbol{a}_t \in \mathcal{A}\left(\boldsymbol{S}_t\right)$, such that we write $\mathrm{Profit}(\boldsymbol{S}_{t^+}) = \mathrm{Profit}(\boldsymbol{S}_t, \boldsymbol{a}_t)$.

The AMoD operator wants to find a policy $\pi\left(\boldsymbol{a}_t|\boldsymbol{S}_t\right)$ that maximizes the expected total reward over all time steps, given the initial state $\boldsymbol{S}_0$:

$$\mathrm{Profit}^*(\boldsymbol{S}_0) = \max_{\pi} \mathbb{E}_{(\boldsymbol{S}_t,\boldsymbol{a}_t)\sim\pi}\left[\sum_{t=0}^{T-1} \mathrm{Profit}\left(\boldsymbol{S}_t, \boldsymbol{a}_t\right) \,\middle|\, \boldsymbol{S}_0\right].$$

To do so, we propose a hybrid DRL algorithm in the following section.

3. Method: Multi-agent Soft Actor-Critic with Global Matching

Analyzing our problem setting, we identify two key requirements to develop an algorithm that constructs an effective control policy: first, it should leverage information patterns that can be observed from historic trip data to make non-myopic decisions. Second, it should be scalable to a realistic system size to coordinate a large number of vehicles and requests. To account for the second requirement, we formalize the centralized dispatching of vehicles to requests as a bipartite matching problem (BMP). This BMP should be weighted to allow for non-myopic dispatching decisions, anticipating the downstream impact of decisions in a stochastic environment. The choice of weights

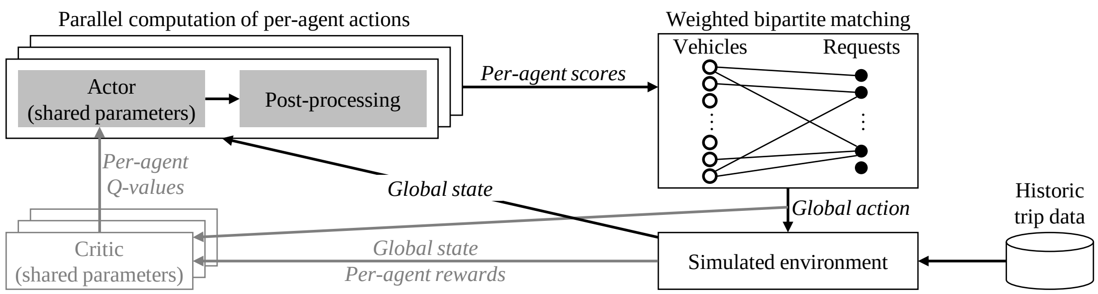

Figure 1: Overview of our method (gray text refers to parts which we use only during training).

heavily impacts the policy's performance. Therefore, we use DRL to parameterize these weights, as it accounts for the downstream impact of decisions in stochastic environments by design and allows to extract and use information from historic data—thus, covering the first requirement.

However, single-agent DRL is not suitable for our problem setting, as the central operator's action space scales exponentially with the number of vehicles and requests per time step and becomes intractable very quickly. Thus, we leverage multi-agent DRL to factorize the action space at the price of increased complexity, caused by having to coordinate the actions of multiple DRL agents to finally take a centralized decision. Our hybrid algorithm combines the advantages of multi-agent DRL with those of combinatorial optimization: we use DRL agents as estimators to compute non-myopic weights, serving as the input to a weighted bipartite matching algorithm, which then makes a globally optimal and coordinated decision.

3.1. Overview

Figure 1 provides an overview of our method in which we leverage a DRL algorithm to parameterize a weighted bipartite matching to take anticipatory global dispatching decisions. To obtain a weight for each request-vehicle combination, we consider each combination as one agent. We represent these agents by an actor network, which we train using the SAC algorithm (Haarnoja et al., 2018). To obtain the weights for our BMP, we post-process the actors' outputs, such that from the perspective of the DRL agents and the computation of policy parameter gradients, post-processing and matching are part of the environment.

Since the DRL agents are a means to factorize the action space of the central operator, rather than "real" individual agents, they can observe the global system state. Although they should take cooperative decisions that eventually benefit the central operator's profit, the agents observe their own (egoistic) rewards, not the global system reward, to avoid a credit assignment problem (e.g., Agogino and Tumer, 2004) that otherwise occurs, in particular with many agents. Here, we enforce coordination of the agents through the BMP and note that varying the credit assignment scheme remains an interesting question for future work.

Individual rewards imply a need for per-agent critic values. From one agent's perspective, the other agents are part of the environment. Since we train all agents concurrently, their policies change simultaneously and the perceived environment is non-stationary. To mitigate this, the critic gets the other agents' actions as an additional input ("centralized critic"), such that the policy evaluation can explicitly account for other agents' behavior (e.g., Lowe et al., 2017; Iqbal and Sha, 2019).

All agents represent a request-vehicle combination and are thus homogeneous. Accordingly, they can share parameters and we need only one actor and one critic network for all agents (cf. Iqbal and Sha, 2019). We can train those centrally, i.e., the total parameter update is given by the sum of per-agent updates. Still, the forward pass of the actor network is independent across agents, allowing for decentralized and parallelized execution, which is important for scalability.

Our method can handle a variable number of requests and thus a variable global action space size, as we can use neural networks with parameter sharing for any number of agents in parallel and the BMP does not require a fixed number of requests. The same holds true for the vehicles, such that the system size when testing may differ from training (see appendix).

We provide details on the individual components of our method in the following.

3.2. Per-agent Post-processing and Global Matching

The actor network parameterizes a categorical probability distribution over the two actions that can be taken for a request-vehicle combination: reject or accept. We post-process the actor output per agent to transform it into a per-agent score, that we then use in the global weighted matching.

Algorithm 1 defines the post-processing. First, we mask infeasible actions by setting the accept probability p_a to zero if the vehicle already has two assigned requests. Then, we sample a reject/assign decision from the masked probability distribution; when testing, we instead take the argmax of the probabilities. A reject decision ($\delta = 0$) at the per-agent level implies a request-to-vehicle reject decision at the global level, such that we set the respective score to zero. For an accept decision, we use the accept probability as score[1], such that a higher probability leads to a higher score for the weighted matching. An accept decision at the per-agent level does not always imply an accept decision at the global level, as the matching might assign the request to a different vehicle.

Algorithm 1: Per-agent post-processing

Input: $p_\mathrm{r}, p_\mathrm{a} \in [0,1]$ s. t. $p_\mathrm{r} + p_\mathrm{a} = 1$; $\boldsymbol{k}^j$
Output: score s
if ${}^j\boldsymbol{r}^2 \neq \emptyset$ **then** $p_\mathrm{r} \leftarrow 1$, $p_\mathrm{a} \leftarrow 0$ `// reject if already two assigned requests`
if *training* **then** $\delta \sim \mathrm{Categorical}\big((p_\mathrm{r}, p_\mathrm{a})\big)$ `// sample` $\delta \in \{0,1\}$ `when training`
else $\delta \leftarrow 0$ if $p_\mathrm{r} \geq p_\mathrm{a}$, $\delta \leftarrow 1$ if $p_\mathrm{r} < p_\mathrm{a}$ `// argmax when testing`
if $\delta = 0$ **then** $s \leftarrow 0$ `// score is zero if rejected`
else $s \leftarrow p_\mathrm{a}$ `// score is accept probability if accepted`

We use all agents' scores to create a bipartite graph, with vehicles and requests as nodes, and edges between all vehicle and request nodes for which we obtain per-agent accept decisions (i.e., $s > 0$). The edges' weights correspond to the respective scores. We solve the resulting maximum weighted BMP (formally defined in the appendix) using the Hungarian algorithm (Kuhn, 1955) to get a globally coordinated decision, where each request is assigned at most once.

1. At first sight, it might seem more intuitive to let the actor parameterize a continuous distribution, from which we can take a sample to directly obtain the score, and/or have separate outputs for the reject/accept probability and the score. We tested both approaches, but empirically observed that they perform worse than the variant described here. If we choose one of those approaches, we cannot compute the terms in the loss functions (see Section 3.3) which are an expectation w.r.t. the policy, i.e., $\pi_\phi(a \mid s, i)^T \cdot (\ldots)$, since the action space is not (purely) categorical anymore. Then, we need to sample to estimate the expectation, as in the version of SAC for continuous action spaces, which increases the variance. We hypothesize that this harms the algorithm's performance and explains our empirical observation.

3.3. Multi-agent Soft Actor-Critic

SAC is an entropy-regularized, off-policy actor-critic algorithm. It trains a stochastic policy $\pi\left(\boldsymbol{a}_t|\boldsymbol{S}_t\right)$ with entropy maximization, incentivizing exploration through a random policy:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{(\boldsymbol{S}_t,\boldsymbol{a}_t)\sim\pi}\left[\sum_{t=0}^{T-1} \text{Profit}\left(\boldsymbol{S}_t, \boldsymbol{a}_t\right) + \alpha\, H(\pi\left(\,\cdot\,|\,\boldsymbol{S}_t\right))\right].$$

The entropy of the policy is defined as $H(\pi\left(\,\cdot\,|\,\boldsymbol{S}_t\right)) = -\mathbb{E}_{\boldsymbol{a}_t\sim\pi}\log\pi\left(\boldsymbol{a}_t|\boldsymbol{S}_t\right)$ and the entropy coefficient $\alpha \in \mathbb{R}_{\geq 0}$ is a hyperparameter that controls the exploitation/exploration trade-off. While SAC was originally developed for continuous action spaces in Haarnoja et al. (2018), it can also be applied to discrete actions (Christodoulou, 2019).

We chose to use an actor-critic algorithm to enable our multi-agent approach of decentralized actors with centralized critics as explained in Section 3.1. Since exploration is paramount for our problem setting, we use SAC, which lets us explicitly tune how much the policy explores.

We parameterize the actor network with parameters ϕ and the critic with θ. SAC uses two critic networks, $Q \in \left\{Q^1, Q^2\right\}$, as well as corresponding target networks with parameters $\bar{\theta}$, which are an exponential moving average of the primary parameters θ. Based on the multi-agent approach as described in Section 3.1, the loss function for the actor with shared parameters is

$$J_\pi(\phi) = \mathbb{E}_{s\sim D}\left[\sum_i \pi_\phi(a\,|\,s,i)^T \cdot \left(\alpha\log\pi_\phi(a\,|\,s,i) - \min_{j\in\{1,2\}}\left\{Q_\theta^j\left(a\,|\,s,i,\bar{a}_{-i}\right)\right\}\right)\right].$$

Here, we use a simplified notation for improved readability: we denote a transition by $(s, \bar{a}, r, s')$, with global states s, s', global action $\bar{a}$ (after the matching), and rewards r. For agent i, r_i denotes its reward, $\bar{a}_{-i}$ is the global action except for agent i's action, and a is a per-agent action (reject/assign), such that $\pi_\phi(a\,|\,s,i) \in [0,1]^2$ and $Q_\theta^j\left(a\,|\,s,i,\bar{a}_{-i}\right) \in \mathbb{R}^2$. We sample states (or transitions) from the replay buffer D and denote the discount factor by γ. For the actor loss, we do not sample the global action from the replay buffer, but compute it based on the state s and the current policy, as in Iqbal and Sha (2019). For each of the two critics $Q \in \left\{Q^1, Q^2\right\}$, the loss function is

$$J_Q(\theta) = \mathbb{E}_{(s,\bar{a},r,s')\sim D}\left[\sum_i \frac{1}{2}\left(Q_\theta\left(a\,|\,s,i,\bar{a}_{-i}\right)\Big|_{\bar{a}_i} - y_i\right)^2\right], \text{ with}$$

$$y_i = r_i + \gamma\cdot\pi_\phi\left(a'\,\middle|\,s',i\right)^T \cdot \left(\min_{j\in\{1,2\}}\left\{Q_{\bar{\theta}}^j\left(a'\,\middle|\,s',i,\bar{a}'_{-i}\right)\right\} - \alpha\log\pi_\phi\left(a'\,\middle|\,s',i\right)\right).$$

Here, the notation $\big|_{\bar{a}_i}$ means "evaluated at $\bar{a}_i$", i.e., of the two Q-values that we compute for the two possible actions of agent i, we use the one corresponding to the global decision $\bar{a}$. The term after γ is the V-value estimate for s' based on $\bar{\theta}$, for which we compute the next global action $\bar{a}'$ with the current policy. The number of requests and thus the number of agents can change between subsequent time steps. This poses a numerical problem for the critic loss computation, which requires the same number of agents for s and s'. We solve this problem by amending the requests in s' when saving a transition to the replay buffer and provide details on this in the appendix.

The actor network obtains all vehicle states and requests for which a decision must be made in the current time step as an input. To deal with these (potentially) many inputs, we train a single request embedding and a single vehicle embedding to encode all requests and vehicles, respectively.

To account for the variable number of requests and to let each agent focus on the parts of the input that are important for this particular agent, we equip the neural network with an attention mechanism (cf. Holler et al., 2019; Kullman et al., 2022). Together with the request and vehicle embeddings for the agent and additional features, we pass the context computed by the attention mechanism to a sequence of feedforward layers. The critic network has the same architecture, but receives the global action as an additional input. We remove the action of the agent from this input, since the critic outputs Q-values for both possible actions. Further details on the neural networks, e.g., a formal description of the attention mechanism and hyperparameters, can be found in the appendix.

4. Experiments

To validate our method, we perform experiments based on historic taxi data that is publicly available for New York City (NYC TLC, 2015). We use a hexagon grid for spatial discretization and consider two different instances: one with 11 small zones (approx. 500 meters distance between neighboring zones) and one with 38 large zones (approx. 1 km distance), both in Manhattan. We consider the time interval from 8:30 am to 9:30 am during morning rush hour as one episode. Our data set contains data for 245 different dates in 2015, which we split into 200 training dates, 25 validation dates, and 20 test dates. We use a time step size of one minute and choose revenue and cost parameters such that a vehicle that serves a customer without empty driving achieves an operating profit margin of 10%. We consider different numbers of vehicles to simulate different degrees of supply shortage; only cases with supply shortage are interesting, as the operator can serve each request immediately in the case of infinite supply, such that a myopic policy would be sufficient. For additional details on the data set, system setup, and hyperparameters, we refer to the appendix.

To benchmark our method, we compare its test performance against two "classical", non RL-based, algorithms: a greedy policy and an MPC approach. The greedy policy accepts any request that can be served with a positive profit (accounting for the cost for empty driving to the pickup location) and rejects all others. It is a reasonable choice when there is no reliable estimate of future requests. If we have such an estimate, it is promising to apply MPC (see, e.g., Alonso-Mora et al., 2017). We adapt this approach to our setting, using a request distribution estimate for mixed-integer-based receding-horizon optimization. Details on both benchmarks can be found in the appendix.

5. Results and Discussion

We provide plots illustrating the training process in the appendix. Figure 2 summarizes the performance of greedy, MPC, and our RL method on the test data for all considered instances. On average, our RL method always outperforms the greedy policy, by up to 5% over the 20 test dates. For individual dates, RL outperforms greedy by up to 17%. It performs by at most 6% worse than greedy for less than 20% of the individual dates. MPC is (substantially) worse than greedy and RL in many cases, although it sometimes outperforms the RL method. This means that MPC can provide a benefit in certain situations, but comes with an unstable performance across instances, which limits its practical applicability. In particular, MPC does not perform well in situations where there is a large shortage of vehicles, which are handled well by our RL method. Thus, our method provides a stable alternative, that always achieves at least the greedy performance and outperforms it by a substantial margin in many cases. Note that the order of magnitude of this performance improvement is significant for our application area (cf. Sadeghi Eshkevari et al., 2022). Given the large

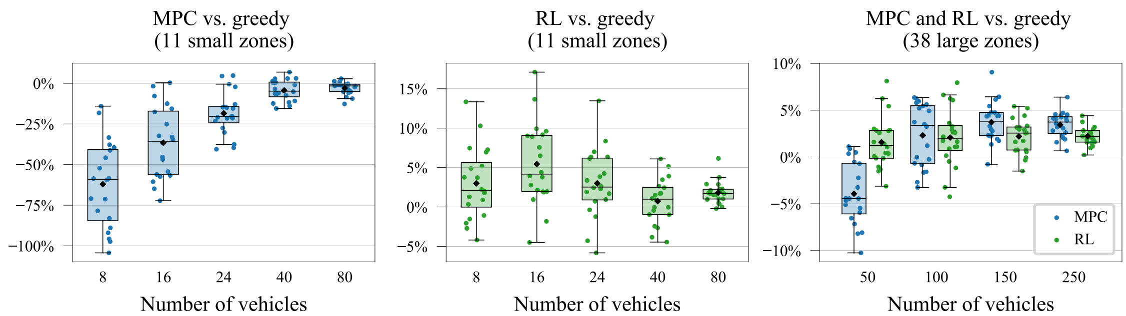

Figure 2: Test performance of MPC and our RL method compared to greedy (greedy is 0%, values $< 0\%$ indicate a performance worse than greedy). Each dot represents one test date.

scale at which AMoD systems operate, the seemingly small percentage improvements translate into significant monetary value for the operator.

Figure 3 shows the performance of MPC and RL for the instance with 38 large zones and different amounts of training data, i.e, different estimation qualities of the request probability distribution. Since our problem setting excludes fixed costs, additional resources, i.e., vehicles, are free of charge. Thus, the greedy policy performs better with very few or many vehicles, compared to instances with a medium number of vehicles. With few vehicles, many requests are available for each vehicle, such that vehicles are rarely idle or drive without a customer. With many vehicles, most requests can be served quickly without empty driving because vehicles are usually available. Consequently, with sufficient training data, for both MPC and our RL approach, the performance gain vs. greedy is largest for a medium number of vehicles. However, with few vehicles, MPC performs worse than greedy, as it is not robust against mistakes when sampling future requests. Such errors have a larger effect with fewer vehicles. With less training data, the performance gain of our method decreases by about one percentage point, but it remains reliably better than greedy. For all instances except the 250 vehicles case, the performance loss is much larger for MPC; it is not always able to sustain a performance better than greedy, even for instances where it outperforms greedy with more training data. With 250 vehicles, there are many resources free of charge, such that the mistakes made by MPC have such a small effect that it is robust against a poor estimation quality. Based on these observations, we conclude that our RL method is more robust against a poor estimation quality due to insufficient training data than MPC. These results might seem surprising, as RL is in general not very sample-efficient—although SAC has better sample-efficiency than most policy gradient-based algorithms, since it is an off-policy algorithm and uses a replay buffer. However, for our problem setting, less training data does not mean that the RL agents must learn from fewer samples, as the available training data can be replayed multiple times in the simulated environment. The performance loss that we observe for the RL method is more likely due to the decreasing diversity of the training data to which the RL agents are exposed, leading to less generalization.

Finally, a major advantage of our RL method over MPC is its shorter computational time during execution. We can train the network parameters offline in advance and easily scale the online execution, because the per-agent actor computations are fast and straightforward to parallelize. Figure 4 shows that although we solve a combinatorial optimization problem in each time step, the

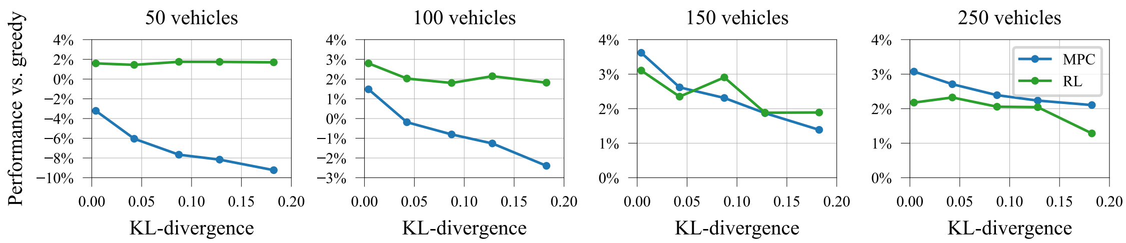

Figure 3: Mean test performance of MPC and RL vs. greedy as a function of the KL-divergence between the true request distribution and the distribution estimated from the training data (with different amounts of training data, resulting in different KL-divergence values). We use the request distribution estimated from the real data for 38 large zones as the true distribution and run the experiments with synthetic data sampled from this distribution.

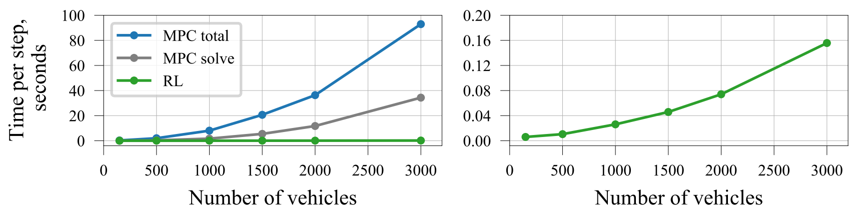

Figure 4: Time to compute one action (mean over 60 steps, based on 38 large zones, number of requests scaled with number of vehicles). MPC solve refers to solving the mixed integer program (MIP), MPC total also includes the generation of the MIP instance in each step.

computational time of our method is very short even for large system sizes. On the other hand, the computational time of MPC increases quickly for large instances: for 3000 vehicles, the action computation for a single step takes more than 30 seconds with MPC, while our RL method (including the matching) takes less than 0.2 seconds. Thus, the practical application of MPC at scale is greatly limited by its computational time, while our RL method can be scaled to much larger system sizes.

6. Conclusion

We consider the dispatching problem of a profit-maximizing AMoD operator with centralized control over a fleet of autonomous vehicles, who accepts (and serves) or proactively rejects requests in real-time. To solve this problem, we use a combination of multi-agent SAC with centralized final decision-making through weighted matching. Our experiments based on real-world data show that our method outperforms two strong benchmarks on most problem instances, that it is stable across these instances and robust against a poor estimation of the request distribution, and that it can be easily scaled to large system sizes. In future work, we will investigate the use of global instead of ego rewards. Furthermore, we will extend our framework to more complex use cases, e.g., dispatching and charge scheduling.

Acknowledgments

We thank Gerhard Hiermann for his help with the implementation of the MIP for the MPC-based benchmark. The work of Tobias Enders was supported by a fellowship within the IFI program of the German Academic Exchange Service (DAAD). This work was supported by the German Research Foundation (DFG) under grant no. 449261765.

References

Adrian K. Agogino and Kagan Tumer. Unifying temporal and structural credit assignment problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 2004.

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

Patrick M. Bösch, Felix Becker, Henrik Becker, and Kay W. Axhausen. Cost-based analysis of autonomous mobility services. *Transport Policy*, 64:76–91, 2018.

Petros Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

Daniele Gammelli, Kaidi Yang, James Harrison, Filipe Rodrigues, Francisco C. Pereira, and Marco Pavone. Graph neural network reinforcement learning for autonomous mobility-on-demand systems. In *Proceedings of the 60th IEEE Conference on Decision and Control (CDC)*, 2021.

Sven Gronauer and Klaus Diepold. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, 55(2):895–943, 2022.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

John Holler, Risto Vuorio, Zhiwei Qin, Xiaocheng Tang, Yan Jiao, Tiancheng Jin, Satinder Singh, Chenxi Wang, and Jieping Ye. Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem. In *2019 IEEE International Conference on Data Mining (ICDM)*, 2019.

Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Yan Jiao, Xiaocheng Tang, Zhiwei Qin, Shuaiji Li, Fan Zhang, Hongtu Zhu, and Jieping Ye. Real-world ride-hailing vehicle repositioning using deep reinforcement learning. *Transportation Research Part C: Emerging Technologies*, 130, 2021.

Harold W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1–2):83–97, 1955.

Nicholas D. Kullman, Martin Cousineau, Justin C. Goodson, and Jorge E. Mendoza. Dynamic ride-hailing with electric vehicles. *Transportation Science*, 56(3):775–794, 2022.

Minne Li, Zhiwei Qin, Yan Jiao, Yaodong Yang, Jun Wang, Chenxi Wang, Guobin Wu, and Jieping Ye. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. In *The World Wide Web Conference*, 2019.

Enming Liang, Kexin Wen, William H. K. Lam, Agachai Sumalee, and Renxin Zhong. An integrated reinforcement learning and centralized programming approach for online taxi dispatching. *IEEE Transactions on Neural Networks and Learning Systems*, 33(9):4742–4756, 2022.

Ziqi Liao. Real-time taxi dispatching using global positioning systems. *Communications of the ACM*, 46(5):81–83, 2003.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017.

NYC TLC. Trip record data. Online, 2015. URL https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page.

Soheil Sadeghi Eshkevari, Xiaocheng Tang, Zhiwei Qin, Jinhan Mei, Cheng Zhang, Qianying Meng, and Jia Xu. Reinforcement learning in the wild: Scalable RL dispatching algorithm deployed in ridehailing marketplace. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.

Erotokritos Skordilis, Yi Hou, Charles Tripp, Matthew Moniot, Peter Graf, and David Biagioni. A modular and transferable reinforcement learning framework for the fleet rebalancing problem. *IEEE Transactions on Intelligent Transportation Systems*, 23(8):11903–11916, 2022.

Xiaocheng Tang, Zhiwei Qin, Fan Zhang, Zhaodong Wang, Zhe Xu, Yintai Ma, Hongtu Zhu, and Jieping 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*, 2019.

Zhaodong Wang, Zhiwei Qin, Xiaocheng Tang, Jieping Ye, and Hongtu Zhu. Deep reinforcement learning with knowledge transfer for online rides order dispatching. In *2018 IEEE International Conference on Data Mining (ICDM)*, 2018.

Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, and Jieping Ye. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

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

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

Ming Zhou, Jiarui Jin, Weinan Zhang, Zhiwei Qin, Yan Jiao, Chenxi Wang, Guobin Wu, Yong Yu, and Jieping Ye. Multi-agent reinforcement learning for order-dispatching via order-vehicle distribution matching. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019.