Deep Reinforcement Learning for Multi-Driver Vehicle Dispatching and Repositioning Problem

John Holler*†, Risto Vuorio*†, Zhiwei Qin‡, Xiaocheng Tang‡, Yan Jiao‡, Tiancheng Jin†,
Satinder Singh†, Chenxi Wang‡ and Jieping Ye‡
*Equal Contribution, †University of Michigan, ‡Didi Chuxing
{johnholl, riv, jintia, baveja}@umich.edu
{qinzhiwei, xiaochengtang, yanjiao, wangchenxi, yejieping}@didiglobal.com

***Abstract*—Order dispatching and driver repositioning (also known as fleet management) in the face of spatially and temporally varying supply and demand are central to a ride-sharing platform marketplace. Hand-crafting heuristic solutions that account for the dynamics in these resource allocation problems is difficult, and may be better handled by an end-to-end machine learning method. Previous works have explored machine learning methods to the problem from a high-level perspective, where the learning method is responsible for either repositioning the drivers or dispatching orders, and as a further simplification, the drivers are considered independent agents maximizing their own reward functions. In this paper we present a deep reinforcement learning approach for tackling the full fleet management and dispatching problems. In addition to treating the drivers as individual agents, we consider the problem from a system-centric perspective, where a central fleet management agent is responsible for decision-making for all drivers.**

***Index Terms*—reinforcement learning, ride-sharing, fleet management, order dispatching**

I. Introduction

The order dispatching and fleet management system at a ride-sharing company must make decisions both for assigning available drivers to nearby passengers (hereby called orders) and for repositioning drivers who have no nearby orders. These decisions have short-term effects on the revenue generated by the drivers and driver availability. In the long term they change the distribution of drivers across the city, which in turn has a critical impact on how well future orders can be served. Provident algorithmic solutions, which account for the short term and long-term consequences of their decisions can improve the quality of service of the ride-sharing platforms and are thus an important area of research.

Recent works [1], [2] have successfully applied Deep Reinforcement Learning (RL) techniques to dispatching problems, such as the Traveling Salesman Problem (TSP) and the more general Vehicle Routing Problem (VRP) [3], however they have primarily focused on *static* (*i.e.* those where all orders are known up front) and/or *single-driver* dispatching problems. In contrast to these problems, the fleet management and order dispatching problem ride-sharing platforms face has multiple drivers and dynamically changing supply and demand conditions. We refer to this dynamic dispatching and fleet management problem as the Multi-Driver Vehicle Dispatching and Repositioning Problem (MDVDRP).

VRPs and other problems similar to the MDVDRP are studied in the field of combinatorial optimization. Exactly solving instances of these problems at the scale of real-world environment is computationally intractable [4]. To deal with the intractability, heuristic solutions, which produce approximate solutions in polynomial time, are often used [5], [6]. The planning problem presented by the MDVDRP is related to the VRPs, but the complexity comes from the dynamic nature of the assignment scenario rather than the intractability of computing the exact solution. Drivers and orders appear in the assignment system at random points in time. In this dynamic assignment setting, assignment decisions are made based on the current driver-order situation, without exact information about future orders. A high performing assignment solution needs to account for unknown future supply and demand conditions.

In realistic instances of the MDVDRP, the decision-making continues 24 hours a day and may involve thousands of drivers and tens of thousands of customers. Accounting for the spatially and temporally varying supply and demand conditions makes hand-crafting heuristic solutions to these scenarios challenging. In this paper, we explore a deep reinforcement learning approach to the MDVDRP. The contributions of our work can be summarized as follows. (i) We introduce a new reinforcement learning problem: the MDVDRP, which is motivated by the needs of real-world ride-sharing platforms. (ii) We propose a novel network architecture for decision-making in a realistic problem setting with variable sized observation and action spaces. (iii) We provide empirical analysis of value-based and actor-critic methods on instances of the MDVDRP, including instances based on real-world data.

II. Related Work

Recent machine learning approaches to dispatching and routing problems operate according to an encoding-decoding scheme, where information is first processed into a fixed-sized representation, and then actions are decoded from this representation [1], [7]. Machine learning approaches to the more general problem family of combinatorial optimization problems are surveyed in [8]. Pointer networks [9] offer an approximate solution to traveling salesman problems by encoding cities (in our terminology, orders) with a recurrent network, and then producing a solution by sequentially "pointing" to orders using an attention mechanism [10]. The network is trained with a supervised loss function by imposing a fixed ordering rule on decoded orders. Bello *et al*. propose training an architecture similar to the pointer networks with policy gradients instead of a supervised loss. Using policy gradients allows them to dispense with the fixed ordering of the outputs during the decoding phase [1]. Similarly, we use reinforcement learning to train our networks. An alternative to pointer networks is explored in [11], where graph convolutional networks are used for solving routing problems. We follow an architecture related to [2], which uses an attention mechanism for encoding the inputs. We depart from their architecture in two ways. First, we replace the input attention layers with layers that compute their output elementwise. Second, we remove the recurrent network used in the decoder.

In practice, order dispatching and fleet management problems are often solved with heuristic solutions, which construct an approximation to the true problem by ignoring the spatial extent, or the temporal dynamics, or both, and solve the approximate problem exactly. One example of a heuristic solution to the order dispatching problem is the myopic pickup distance minimization (MPDM), which ignores temporal dynamics and always assigns the closest available driver to a requesting order [12]. In Local Policy Improvement [13], handcrafted heuristics are combined with a machine learning method by summarizing supply and demand patterns into a table and then using the learned patterns to account for future gains in the real-time planner of the dispatching solution. A fully machine learning based approach to the dispatching problem is presented in [14], where deep Q-learning is used for learning dispatching strategies from the perspective of a single driver. We take these developments a step further and learn fleet management and order dispatching strategies end-to-end using reinforcement learning.

Another thread of related work comes from the multi-agent reinforcement learning literature. Specifically, our single-driver training approach is analogous to the "independent Q-learning" training approach [15]. Independent Q-learning has been leveraged successfully in a number of problems including resource allocation problem for elevator control [16], traffic signal control [17], and fleet management [18]. Multi-agent reinforcement learning has also been investigated for order dispatching in [19], where a grid-based algorithm is used. Instead of operating on a grid, our method is based on continuous coordinates, which makes it more ready for deployment in the real-world.

III. Method

A. MDVDRP as a Reinforcement Learning Environment

The experiments presented in this paper are conducted in a ride-sharing simulation environment. The environment is designed to capture the dynamic supply and demand situation in ride-sharing platforms. The environment represents customers as orders, which start and end in some coordinates. Assigning a driver to the order immediately yields a reward proportional to the price of the order. Drivers are represented as points in the 2-D space and they can move by serving orders or repositioning without an order. The drivers and orders are generated following a Poisson process with parameters depending on each scenario we consider.

Decision points are triggered whenever a driver becomes active in the system, finishes an order or finishes a repositioning action. Variable amounts of time may have passed between the decision points when the simulator polls the policy for actions. The simulation scenarios considered in the experiments mostly depict the situation where there are more customers demanding rides than there are drivers to serve them. In this setting, it is natural to assume that only a single driver is available for actions at any decision point. This assumption simplifies the design of the policies by removing the need to choose which driver to assign an order to. To prevent multiple drivers polling for actions at exactly the same moment, we add random noise with small variance to the duration of the reposition actions. The actions available to the RL agent at each decision point are with respect to the driver that is currently available in the environment. These actions include assigning the driver to nearby orders as well as actions to reposition the driver to other areas.

Assigning a driver to an order removes the order from the system and makes the driver unavailable for instructions until the order has been completed. At order completion the driver is relocated to the order destination and made available for assignments. Orders have a limited time window within which they are valid. After the validity window has passed, the orders will be removed from the system.

The policies may choose to move the drivers to different directions for a fixed amount of time by selecting reposition actions. The number of available reposition actions includes actions that move the driver into one of the eight cardinal directions for a fixed period of time and a stationary action.

The observation of the agents consists of the environment time, the driver for which action is currently being selected and all drivers and orders currently in the simulation. At time t there is a collection of orders $o_t^i \in \mathcal{O}_t$, drivers $d_t^j \in \mathcal{D}^t$, with exactly one *available driver* $d_t^{selected}$. The state is given to the neural network as $s_t = (\mathcal{O}_t, \mathcal{D}_t, d_t^{selected}, t)$. The orders are presented as 6 dimensional vectors consisting of the starting and ending coordinates, price, and time waiting. Time waiting is the difference between the current time and the creation time. A driver is represented by a 6 dimensional vector: the coordinates of the driver location, x and y components of its reposition direction, time to order completion, and time to reposition completion. If the driver is serving an order, its location is set to the ending location of the order it is servicing. If the driver is repositioning the driver location is updated at each timestep, the reposition direction shows which way the driver will move during the next timestep.

To limit the number of orders considered by the policy at each timestep, we impose a *broadcasting radius* d_{bcast} on the order assignment. This means that drivers may be only paired with orders if they are within d_{bcast} units of the driver. Otherwise, the driver may only take a repositioning action.

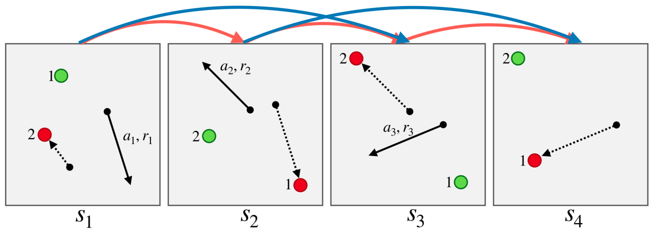


Fig. 1. The above image shows a trajectory with four timesteps in an MDVDRP instance with two drivers. The currently available driver is green, dispatched driver is red, and the order that the available driver accepts at time t is a_t and has price r_t. The accepted order at time t is labeled by its action name and price, (a_t, r_t) and travels from the solid black dot to the terminal arrow. Driver-centric transitions are indicated by blue arrows above state, e.g. transition (s_1, a_1, r_1, s_3), which is driver-centric with respect to driver 1. System-centric transitions are indicated by red arrows e.g. transition(s_1, a_1, r_1, s_2), which transitions from a state where driver 1 is available to a state where driver 2 is available.

The repositioning actions are not available to the drivers when there are orders within broadcasting radius of them.

B. Reward Settings

The objective of the algorithms in the MDVDRP problem is to maximize the cumulative reward defined by the environment. The environment rewards the agent for each assigned order with a reward the size of the order price. Reposition actions yield no reward. We consider two alternative reward specifications corresponding to *driver-centric* and *system-centric* perspectives. An visual overview of these concepts is presented in Figure 1

In the driver-centric approach, we consider the MDVDRP a reinforcement learning problem from the perspective of the individual driver. In this setting, each driver is maximizing their own expected revenue and there are no incentives for co-operation. The trajectories taken by each driver in the environment are collected separately and the discounted returns are computed on the individual trajectories. The individual trajectories consist of timesteps that are consecutive from the perspective of the driver but not necessarily from the perspective of the environment as other drivers may have taken actions between the actions of any single driver.

From the point of view of the ride-sharing platform, the dispatching and repositioning problem is not about individual drivers maximizing their own revenue but rather about the platform maximizing the combined revenue across all drivers. Therefore, it is important to consider optimizing the policies from the perspective of the whole system. In this system-centric approach, the expected cumulative reward across all drivers is being maximized. This leads to the trajectories experienced by the policy consisting of timesteps that are consecutive from the perspective of the environment. For example, if the driver i acts on the timestep t_1 receiving reward r_1 and driver j takes an action on the timestep t_2, the training algorithm will consider the timesteps t_1 and t_2 consecutive.

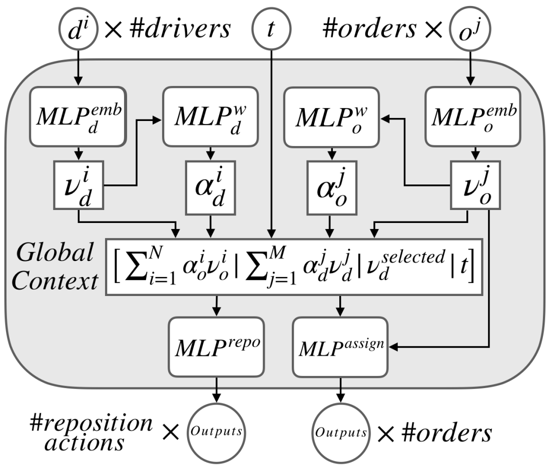


Fig. 2. Network Architecture. Driver and order embeddings ν_d^i, ν_o^j and their corresponding weights α_d^i, α_o^j are computed by MLP^{emb} and MLP^w respectively. A fixed sized global representation of the orders and drivers is computed by concatenating the sums of the weighted embeddings with the selected driver embedding and the environment time. The fixed number of outputs for the reposition actions are computed by MLP^{repo} using the global context as an input. MLP^{assign} computes one output for each order using the global context and each order embedding as an input.

C. Neural Network Architecture

We propose a neural network architecture for RL in environments with variable sized observation and action spaces. An overview of the proposed network architecture is presented in Figure 2. A learned pooling mechanism allows the network to compute a fixed-sized global representation of the inputs, which enables relating the features of each individual input to the global state. In the environments we consider, the number of actions depends on the number of orders in the observation as described in III-A. We compute network outputs, one for each action, in a manner similar to the weight computation in the attention mechanism [10].

The network first computes order embeddings ν_o^i, order pooling weights α_o^i, driver embeddings ν_d^j, and driver pooling weights α_d^j. The embeddings are length 128 vectors computed by MLP_d^{emb} and MLP_o^{emb}, both of which have one hidden layer of size 128 and ReLU activations. The scalar pooling weights α are computed from the embeddings by MLP_d^w and MLP_o^w. Both have hidden layer size 128 and tanh activation, and sigmoid output activation. The global context vector is computed as $\left[\sum_{i=1}^N \alpha_o^i \nu_o^i | \sum_{j=1}^M \alpha_d^j \nu_d^j | \nu_d^{selected} | t\right]$, where $[a|b]$ denotes concatenation and t denotes time.

MLP^{assign}, which has one hidden layer of size 64 and ReLU activation, outputs values for the assignment actions. MLP^{assign} takes each order embedding concatenated with the global context separately as input and produces a scalar output for each input. The repositioning actions are computed by MLP^{repo} with the same architecture as MLP^{assign}. MLP^{repo} has one output for each reposition action and uses the global context as its input.

D. Training Algorithms

We train the proposed network architecture on the MDV-DRP using two modern reinforcement learning algorithms: Deep Q-Networks (DQN) [20] and Proximal Policy Optimization (PPO) [21]. We chose these two algorithms because they represent widely used, modern methods in off-policy and on-policy reinforcement learning and many of the recent successes in deep reinforcement learning have been achieved using one of the two algorithms. A short description of the key ideas of the algorithms is given in the following. For more details we refer the reader to the corresponding papers.

DQN is a value-based, off-policy reinforcement learning algorithm. The Q-network estimates state-action values meaning it gives numeric estimates to the expected future return for each action available in the current state. On each iteration i, it optimizes the objective function

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(s,a,r,s')}\left[(r + \gamma \arg\max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2\right] \quad (1)$$

where (s, a, r, s') are states, actions, rewards and next states sampled from the environment, γ is a discount factor, $Q(s, a; \theta_i)$ is a neural network parameterized by θ_i approximating the action-value function, θ_i^- are the target network parameters and θ_i are the learning network parameters. The DQN algorithm collects experience from the environment, stores the experience in a replay buffer and updates the learning network parameters by performing stochastic gradient descent on the loss function. The target network is used to compute an action-value estimate in the next state s'. In order to improve the stability of the learning network update, the target network parameters are copied from the learning network every couple of hundred update steps. Actions are selected by taking the $\arg\max$ of the Q-network output, except with probability ϵ a random action is chosen in order to make the algorithm explore. When evaluating the policies exploration is no longer needed and the epsilon can be set to a zero.

PPO is an on-policy actor-critic reinforcement learning algorithm [21]. Unlike DQN, PPO directly optimizes the policy using the policy gradient method. In our case, the parameters of the policy are estimated by a neural network. During training time the actions are chosen by sampling from the categorical distribution parameterized by the policy. At evaluation time, we select the actions with the maximum probability. The policy gradient objective is given by

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A(\tau)\right] \quad (2)$$

where $\pi_\theta(a_t|s_t)$ is the action distribution defined by the policy, θ are the parameters of the policy, $\tau = (s_0, a_0, r_0, ...s_T, a_T, r_T)$ are trajectories sampled from the environment, and $A(\tau)$ is the advantage function. We estimate the advantage function using the generalized advantage estimation method [22], which requires requires computing a state-value function estimate called the critic. The critic is learned alongside the policy using the same data. On-policy reinforcement learning algorithms require new samples for each update they compute, otherwise they risk deviating too far from the previous policy, which may harm the policy performance. Since sample collection can be expensive in reinforcement learning, updating on each batch of data multiple times would be beneficial. In order to enable multiple updates on the same data, different variants of the PPO algorithm either clip or penalize the objective in order to prevent the policy from diverging too far from the previous one. We use the clipping variant, which clips the objective based on the probability ratio between the current and the previous policy to remove the incentive to diverge too far.

As the simulation runs in continuous time, the time between consecutive timesteps may vary. Both algorithms use a discounting factor γ to compute the discounted return, which accounts for the future rewards. In continuous time settings, the discounting factor for each timestep is $\gamma^{t'-t}$, where t is the time at the current observation and t' in the next.

We found empirically that training using the system-centric rewards is more challenging than using the driver-centric rewards. We found the use of n-step Q-learning [23] helpful for stabilizing the training of DQN when training on system-centric rewards. In all of our DQN experiments with system-centric reward we use 20-step Q-learning.

IV. Experiments

We investigate the learning behavior of the proposed network architecture in ride-sharing environments implemented using the simulator described in III-A. We present results in five environments to test and illustrate the dispatching and repositioning strategies learned the proposed approach. The environments are illustrated in Figures 3, 4. The geography of the environments is presented as a rectangle with the longer side length set to one. The drivers move at speed 0.1 and the broadcasting radius is set to 0.3.

We compare the learned policies against two kinds of baselines: myopic revenue maximization (MRM) and myopic pickup distance minimization (MPDM) [12]. MRM always assigns the highest value order to the closest available driver. MPDM assigns orders to drivers in the order of shortest distance first. Since the baselines do not account for repositioning, we test three variants of the baselines with different repositioning heuristics. In the simple variants, broadcasting distance is ignored and drivers can be assigned to orders across the region. The drivers stay where they drop orders off until they are assigned to the next order. In the *random* variants of MRM and MPDM, if a driver has no orders the within broadcast distance, then a repositioning action is selected randomly. The *demand* variants of MRM and MPDM apply a simple heuristic on repositioning, by moving the drivers towards the nearest order when an order is available but not within broadcasting distance. If there is no order available, the demand variants take random reposition actions.

We use the same neural network architecture for all variants of the learning algorithms, except for the extra output layer

TABLE I

EXPERIMENTAL RESULTS. THE ALGORITHMS WERE EVALUATED MULTIPLE TIMES DURING THE TRAINING AT FIXED INTERVALS AND THE RESULTS OF THE BEST EVALUATION STEP ACROSS ALL RANDOM SEEDS IS REPORTED ± STANDARD ERROR. THE REPORTED VALUES ARE AVERAGES OVER 5 EPISODES FOR HOT COLD AND REGIONAL DOMAINS AND OVER 20 EPISODES FOR HISTORICAL ORDERS DOMAIN. ALL THE SCORES THAT ARE WITHIN STANDARD ERROR OF THE BEST SCORE ARE BOLDED IN THE TABLE.

	Hot Cold		Regional		Historical Orders
Algorithm	High Demand	Low Demand	High Demand	Low Demand	
MRM-simple	5359 ± 8	5189 ± 14	3597 ± 6	2964 ± 22	40001 ± 128
MPDM-simple	5917 ± 6	5713 ± 20	4258 ± 8	3328 ± 28	37960 ± 127
MRM-random	797 ± 30	949 ± 41	2150 ± 30	3039 ± 30	49563 ± 309
MPDM-random	1006 ± 35	1004 ± 50	4203 ± 19	3103 ± 33	46763 ± 241
MRM-demand	5351 ± 9	5449 ± 9	2161 ± 28	3262 ± 16	48805 ± 463
MPDM-demand	5883 ± 17	5658 ± 15	4252 ± 8	$\mathbf{3343 \pm 16}$	46635 ± 539
PPO System-Centric	$\mathbf{7954 \pm 17}$	5801 ± 29	4744 ± 2	$\mathbf{3372 \pm 33}$	50094 ± 162
DQN System-Centric	6323 ± 168	5278 ± 77	4735 ± 12	2831 ± 104	48532 ± 71
PPO Driver-Centric	7861 ± 23	5767 ± 10	4888 ± 2	3208 ± 16	53029 ± 45
DQN Driver-Centric	7883 ± 3	$\mathbf{5855 \pm 8}$	$\mathbf{5006 \pm 15}$	$\mathbf{3349 \pm 6}$	$\mathbf{53255 \pm 130}$

TABLE II

DISTRIBUTE DOMAIN WITH 20 DRIVERS

Algorithm	50/50 Served %	80/20 Served %
Optimal	100%	100%
Uniform Optimal	50%	80%
PPO System-Centric	$100 \pm 0.0\%$	$93 \pm .54\%$
DQN System-Centric	$95 \pm .11\%$	$80 \pm 3.42\%$
PPO Driver-Centric	$96 \pm .13\%$	$92 \pm .72\%$
DQN Driver-Centric	$96 \pm .13\%$	$92 \pm .72\%$

implementing the critic for PPO, which does not exist in the model used for DQN. We set the discount factor $\gamma = 0.99$ for both algorithms. For DQN we use batch size of 32, replay buffer size 20000 and update on every timestep with learning rate 0.0001. The exploration rate ϵ starts from 0.99 and we anneal it by 0.01 on each episode to the final value 0.1. We update the target network every 100 steps. The PPO policy and value functions are updated 20 times every epoch using 4000 consecutive timesteps as the training data. We use generalized advantage estimation (GAE) [22] and set the GAE hyperparameter $\lambda = 0.95$. We aim to keep hyperparameters unchanged between the different environments but make slight changes for Distribute Domain and the real data based domains. Those changes are explained in the relevant sections. To handle the large numbers of drivers and orders in the Historical Statistics domain, we use a parallel implementation of PPO and suitable hyperparameters, which are explained in IV-E.

The results from the experiments can be found in tables I and II. We present training curves for the Hot-Cold, Regional and real data based environments in Figures 5, 6 and 7.

A. Regional Domain

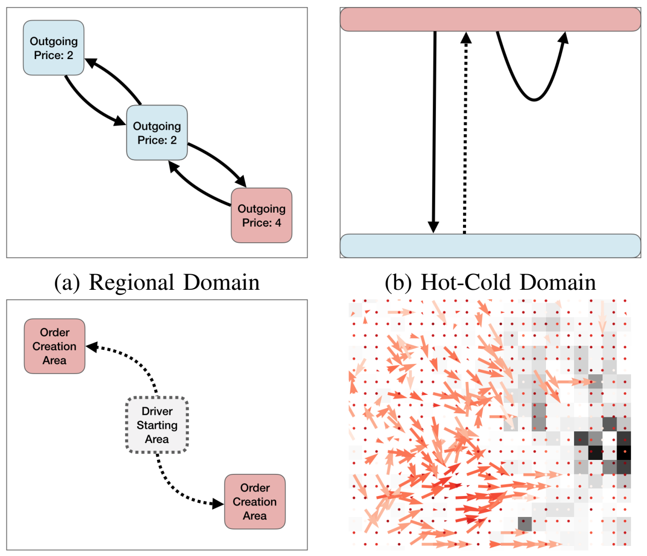
In our first experiment, we consider the Regional Domain, which illustrates how a simple price differential can be exploited by learned policies but is missed by the myopic dispatching and repositioning approaches. Intuitively, in this domain, a good policy dedicates enough drivers to fully serve the high-reward orders and serves the other orders with the remaining drivers. Myopic policies, which ignore the long-term effects of their decisions, will fail to exploit the available extra reward from serving the high-reward orders. The high-level concept of Regional Domain is presented in Figure 3. In the Regional Domain, there are three regions: left, center, and right. One quarter of all orders go from the center to the upper-left region, one quarter from center to bottom-right, one quarter from upper-left to center, and one quarter from bottom-right to center. All orders yield a reward of 2 except those that go from right to center, which yield a reward of 4.

In the high demand version of the Regional Domain, the proposed method outperforms the baselines by over a 15% margin between the best performing baseline (MPDM-demand) and the best learned policy (Driver-Centric DQN). To identify the source of the difference between the learned policies and the baselines, we look at a policy trained with PPO on driver-centric rewards. The learned policy serves 70 times more orders from the high reward region than the low reward region and achieves significantly higher score than the baselines, which suggests that it has learned to exploit the higher priced orders to its advantage.

The low demand variant proves to be challenging for the learning algorithms. Here the demand is low enough that over commitment to the high reward area can become a problem as is the case for Driver-Centric PPO, which ends up allocating 8 times more idle drivers to the high reward area and as a consequence has 8% lower ratio of total orders served compared to MPDM-demand. This policy leads to weak performance, and is in fact outperformed by the simple and demand variants of MPDM.

B. Hot-Cold Domain

The Hot-Cold domain can be thought of as a ride-sharing scenario with a busy area of downtown ("hot region") sur-



(a) Regional Domain (b) Hot-Cold Domain

(c) Distribute Domain (d) Historical Orders Domain

Fig. 3. Conceptual drawings of the illustrative domains considered in the experiments. The colored regions represent areas where orders are generated. The solid arrows depict order starting and ending locations and the dotted paths depict paths the drivers need to reposition along in order to get to the orders. Best viewed in color. (a) **Regional domain** tests whether the policies can learn to exploit price differences between otherwise similar orders. Each square is labeled with its outgoing order value. (b) **Hot-Cold Domain** seeks to evaluate the algorithms ability to learn allocating drivers to orders which minimize driver idle time. All orders begin in the red bar, with their positions generated uniformly randomly. For the destination, a fair coin is flipped to decide whether the order ends in hot or cold, and then the exact position is sampled uniformly randomly in the designated region. (c) **Distribute Domain** requires the policies to anticipate that the orders are going to be created in the two distant regions and allocate correct number of drivers to each. Figure (d) represents an order generation pattern defined by the historical data and the corresponding reposition strategy learned by the policy for an hour long time window in **Historical Data Domain**. The graph is composed of 2d histogram depicting the order generation pattern and arrow plots depicting the average reposition movements in each histogram bin. The intensity of the arrow color corresponds to the number of drivers who have been in each cell during the time window.

rounded by suburbs with less traffic ("cold region"). In the Regional domain, the advantage of traveling to the region with higher prices is clear; it is directly tied to the price of orders found in that region (4 vs. 2). In the Hot-Cold domain, the agent must learn a more subtle advantage. Order pickup locations are located uniformly along the top edge of the simulation area. Half of the orders end uniformly along the bottom edge of the area and half end uniformly in the hot region. Order price is given by the Euclidean distance from order pickup to order drop-off locations. Despite orders to the cold region having higher price (since they are longer on average), it is generally more advantageous for drivers to stay in the hot region, since there they can quickly pick up new orders. In other words, the advantage is entirely *temporal*. An illustration of the Hot-Cold domain is presented in Figure 3.

The results in Table I suggest that learning to balance serving the orders to hot and cold regions is straightforward for the learning algorithms and all learning algorithms outperform all baselines in the high demand variant of the environment. As with the Regional domain, the challenges of training DQN with system-centric rewards are apparent in the low demand Hot-Cold domain, where it is outperformed by multiple baselines.

The margin in favor of the learned policies is narrower in the low demand variant. In the low demand variant of the environment, the policies trained with driver centric rewards the ratio of orders served is approximately double compared to the high demand variant (50% vs 25%). The higher served ratio suggests that there is less freedom of choice between hot-to-hot and hot-to-cold orders in the low demand regime, which in turn reduces the potential for improvement over the naive policies.

In all of the experiment configurations for Regional domain and Hot-Cold domain, we found DQN to be more sample efficient than PPO. DQN was trained for 1000 episodes and PPO often took two to three times as many to achieve similar performance. Actor-critic methods are known to be less sample efficient than value-based methods and another contributing factor may be the relatively low learning rate we used for PPO.

C. Distribute Domain

While Hot-Cold domain tests an important aspect of learning - namely, the ability of the agents to reposition drivers to locations where they can pick up new orders, this repositioning behavior is quite simple in that it is *uniform across drivers*. This means that each individual driver can always be repositioned in the same manner (*i.e.* "if in cold region, go to hot region"). In order to test whether our methods can learn non-uniform repositioning behavior, we introduce a class of "distribution environments" where drivers must be repositioned so as to match their spatial distribution with a fixed future order distribution. A Distribute domain operates in two phases. In the first phase, the environment resets to a state with k drivers and no orders, so drivers may only reposition during this phase. In the second phase, k orders appear according to a fixed spatial distribution, and drivers can match to orders if they are within a given broadcast radius d_{bcast}. The second phase only lasts long enough to allow drivers to reposition one more time before all orders cancel and the environment is reset. Each order matching action receives $+1$ reward. Order destinations are placed away from start locations so that each driver may only serve one order per episode. As a result, the episodic return is proportional to the number of orders served, so we may interpret the episode score as a measure of how well the agent arranges driver supply in phase 1 with order demand in phase 2.

In our experiments, the distribution of orders always consists of two small patches in the top left and bottom right parts of the unit square, refer to Figure 3 for visualization. The order start locations are sampled uniformly within each patch. The total number of orders in each patch is fixed across episodes, and we denote it fractionally. An even order split between patches (*e.g.* 10 orders in both patches) is denoted $50/50$. If

80 percent of orders are in the first patch and 20 percent are in the second patch, we denote it as $80/20$.

We found that training successful policies on the Distribute Domains requires more random exploration than any of the other environments. For DQN, we set the final $\epsilon = 0.2$. For PPO, we follow an annealing scheme similar to DQN where we anneal the entropy coefficient from 0.7 to 0.01 over the first 2000 epochs.

Results for distribute domains with 20 drivers are presented in Table II. We include the optimal served percentage (which is 100 %) and the "uniform optimal" served percentage. This quantity reflects the maximum score one can obtain if the repositioning behavior is uniform across drivers. The results between driver and system-centric variants are mixed. While system-centric PPO achieves a slight advantage over the driver-centric variant, the relationship is flipped for DQN.

D. Historical Orders Domain

Solving the MDVDRP is motivated by the real-world need of ride-sharing platforms to improve their marketplace for drivers and passengers. To test our learning algorithms in a more realistic setting, we consider an environment where the order generation scheme is based on historical order data from the GAIA dataset [24]. The dataset provides gps records for all orders served during a 30-day period in the city of Chengdu in China. To limit computational demands, we choose a subset of consisting of 10% of the orders in the dataset. The orders in the subset all start and end in an area from the dataset that has approximately twenty thousand orders per day. We then create 30 *order generation schemes*, which correspond to the 30 days in the dataset. Each episode in the environment corresponds to a randomly sampled day from the dataset. The orders generated during the episode appear exactly in the coordinates and at the time defined by the data. We use a fixed number of drivers (100), a 2 kilometre broadcast radius and a fixed speed (40 km/h). Histograms of the order start locations in the area under consideration are presented in Figure 4.

Similarly to the Hot-Cold domain, the reward for serving an order is the distance from the order start location to the drop-off location. Following from the reward definition, the total reward each driver receives is directly correlated with the total time they spend serving orders and thus inversely correlated with the time spend between orders.

Driver-centric DQN and PPO are both capable of learning strong dispatching and repositioning policies, which outperform all of the baselines by over 6%. The learning curves for the experiments in this domain are presented in Figure 7. The historical data domain also demonstrates the importance of redistributing drivers during their downtime. The worst performing algorithms on the historical data domain are the *simple variants* of MPDM and MRM, which do not use reposition actions at all. In contrast, the learned policies actively redistribute drivers back to the populated areas as shown in Figure Figure 3 (d).

Finally, we observe that both DQN and PPO have trouble learning competitive policies on this larger scale problem when trained on the system-centric rewards. We hypothesize that this is due to having more drivers complicating the already challenging credit assignment problem. With more drivers simultaneously acting in the environment, the time between decision points becomes smaller and the effective discount factor approaches one due to the way it is computed. Training with a discount factor close to one can make the training unstable. To mitigate this effect we use a smaller discount factor $\gamma = 0.9$.

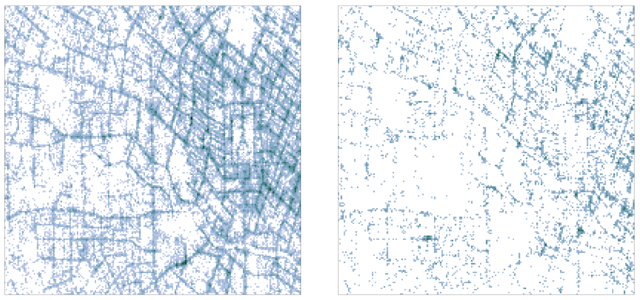


Fig. 4. 2d histograms depicting the order creation in the *Historical Data Domain*. Darker colors indicate more orders requested in the cell according to a power law coloring scheme. The histograms colors have been determined by normalizing the histograms separately for visual clarity. In **(a)** the histogram corresponds to the 6 most active hours on all of the 30 days during which the dataset has 294279 orders for the region under consideration. The 6 quietest hours are similarly displayed in **(b)**, which corresponds to 21807 orders in the dataset.

E. Historical Statistics Domain

Finally, we test our method in another MDVDRP instance based on the GAIA dataset. In the Historical Statistics domain, both drivers and orders are generated from a Poisson process. Having the number of drivers vary with the time of the day takes the simulation another step closer to the realistic setting.

To build the simulator from this data, we first covered the city in a square 20 by 20 grid, and extracted Poisson parameters $\kappa_{x,y,t}$ where x is an order start tile, y is an order end tile, and t is the time of day in hours. This results in $400 \times 400 \times 24 = 3.84$ million parameters which we use specify an order generation process. The order generation parameters for the most active and the most quiet hours in the day averaged over all 30 days. The price of an order is determined by the trip distance.

We determine the driver generation parameters using a similar approach as we used for the order parameters. A driver is counted to have become active when it serves its first order during the day. The drivers remain active for six hours after they have been created. In the initial testing of the environment, we found that using the raw driver generation parameters results in enough drivers being created that even a randomly initialized policy is able to achieve above 99% ratio of orders served. The dataset only records orders which were served by drivers, so it seems natural that the served ratio using the unscaled parameters would be high. The approximations we make about the duration drivers remain active and the speed of the drivers could also be contributing to the high served ratio. In order to set up the environment such that there is room for improvement, we scale the driver generation parameters down to 7%. The driver generation parameters are further scaled down by a factor of 0.5, which we also apply to the order generation parameters, to lower the computational demands.

To handle the large scale of the domain with close to 100000 orders per day served by over a 1000 drivers, we use a parallelized implementation of PPO. Using parallel environments allows us to sample shorter trajectories in each environment to achieve the same update batch size. Sampling fewer consecutive transitions between each update speed up the training. We use 10 parallel environments and correspondigly reduce the steps per epoch hyperparameter to 400. We also raise the policy learning rate to 0.0005 and value function learning rate to 0.001.

The training curves from the experiments are presented in Figure 7. Training with either system and driver-centric approaches results in policies that clearly outperform the baselines. The best performing learned policy is the driver-centric PPO, which achieves the evaluation score of 286135 ± 1089. For system-centric PPO the evaluation score is 261059 ± 184 and the best performing baseline, MRM-random, scores 252656 ± 280. The driver-centric approach outperforms the system-centric by a margin of close to 10%. It is evident from the evaluation score and the learning curves that learning with the system-centric rewards is challenging, possibly even more so as in the Historical Orders domain.

V. Conclusion

We performed a detailed empirical study of reinforcement learning approaches to multi-driver vehicle dispatching and repositioning problems. We studied driver-centric and system-centric reward definitions and trained policies using DQN and PPO algorithms. We found DQN to be more sample efficient than PPO. On the other hand PPO was better able to learn on system-centric rewards. Central to all of our approaches was the network architecture we presented, which leverages a global representation of state processed using attention mechanisms. We found that, while one can construct environments where the system-centric approach is superior, typically driver-centric is better or at least competitive with the system-centric approach. Furthermore we applied these methods to environments built using real dispatching data, and found that driver-centric approach is able to consistently beat myopic dispatching and repositioning strategies.

References

[1] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.

[2] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takac, "Reinforcement learning for solving the vehicle routing problem," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 9839–9849. [Online]. Available: http://papers.nips.cc/paper/8190-reinforcement-learning-for-solving-the-vehicle-routing-problem.pdf

[3] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management science*, vol. 6, no. 1, pp. 80–91, 1959.

[4] P. Toth and D. Vigo, *The vehicle routing problem*. SIAM, 2002.

[5] J.-F. Cordeau, M. Gendreau, G. Laporte, J.-Y. Potvin, and F. Semet, "A guide to vehicle routing heuristics," *Journal of the Operational Research society*, vol. 53, no. 5, pp. 512–522, 2002.

[6] D. Pisinger and S. Ropke, "A general heuristic for vehicle routing problems," *Computers & operations research*, vol. 34, no. 8, pp. 2403–2435, 2007.

[7] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L.-M. Rousseau, "Learning heuristics for the tsp by policy gradient," in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, W.-J. van Hoeve, Ed. Cham: Springer International Publishing, 2018, pp. 170–181.

[8] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *arXiv preprint arXiv:1811.06128*, 2018.

[9] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2692–2700.

[10] V. Mnih, N. Heess, A. Graves *et al.*, "Recurrent models of visual attention," in *Advances in neural information processing systems*, 2014, pp. 2204–2212.

[11] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *International Conference on Learning Representations (ICLR)*, 2019.

[12] L. Zhang, T. Hu, Y. Min, G. Wu, J. Zhang, P. Feng, P. Gong, and J. 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*. ACM, 2017, pp. 2151–2159.

[13] Z. Xu, Z. Li, Q. Guan, D. Zhang, Q. Li, J. Nan, C. Liu, W. Bian, and J. Ye, "Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 905–913.

[14] Z. Wang, Z. Qin, X. Tang, J. Ye, and H. Zhu, "Deep reinforcement learning with knowledge transfer for online rides order dispatching," in *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2018, pp. 617–626.

[15] C. Claus and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 1998, 1998, pp. 746–752.

[16] R. H. Crites and A. G. Barto, "Elevator group control using multiple reinforcement learning agents," *Machine learning*, vol. 33, no. 2-3, pp. 235–262, 1998.

[17] S. El-Tantawy and B. Abdulhai, "An agent-based learning towards decentralized and coordinated traffic signal control," in *13th International IEEE Conference on Intelligent Transportation Systems*, Sep. 2010, pp. 665–670.

[18] T. Oda and C. Joe-Wong, "Movi: A model-free approach to dynamic fleet management," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 2708–2716.

[19] M. Li, Z. Qin, Y. Jiao, Y. Yang, Z. Gong, J. Wang, C. Wang, G. Wu, J. Ye *et al.*, "Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning," 2019.

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

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[22] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[23] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[24] Didi Chuxing, "Didi launches gaia initiative to facilitate data-driven research in transportation," URL: http://www.didichuxing.com/en/press-news/bgcitgfc.html, 2017.

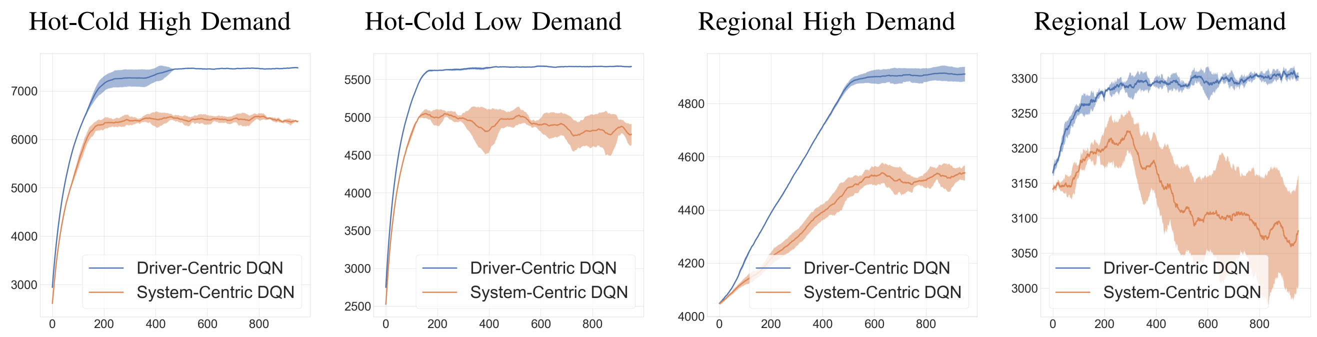


Fig. 5. **DQN training curves on the illustrative environments.** The curves represent average values across 4 random seeds. The shaded area is the standard deviation across the seeds. The curves have been smoothed with a sliding window of width 50. The vertical axis represents the total driver income. The number of training episodes is on the horizontal axis.

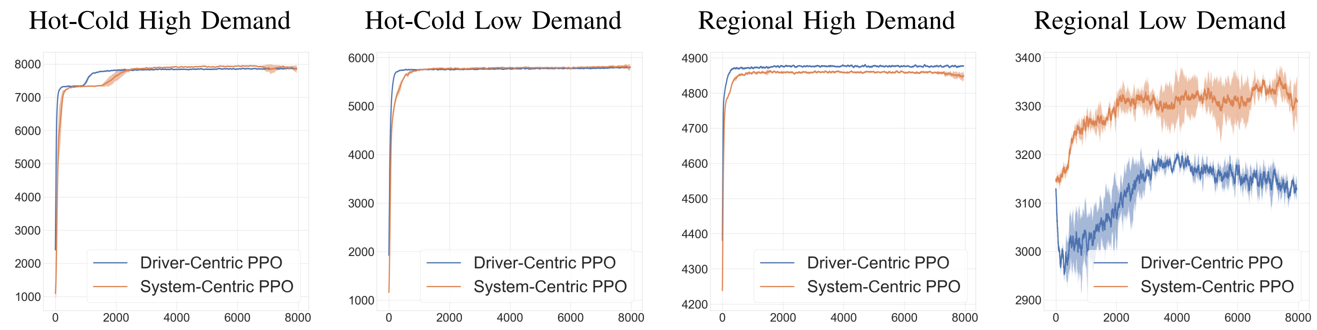


Fig. 6. **PPO training curves on the illustrative environments.** The curves represent average values across 4 random seeds. The shaded area is the standard deviation across the seeds. The curves have been smoothed with a sliding window of width 500. The vertical axis represents the total driver income. The number of update epochs is on the horizontal axis.

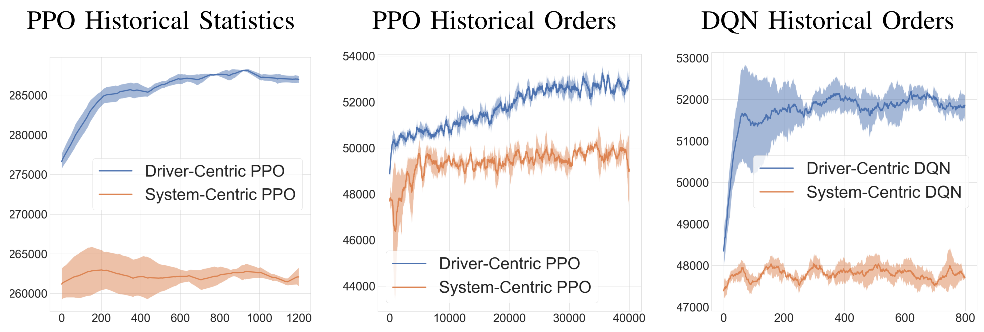


Fig. 7. **Training curves on real-world data domains.** The curves represent average values across 4 random seeds. The shaded area is the standard deviation across the seeds. The curves have been smoothed with a sliding window of width 500 for PPO and 50 for DQN. The vertical axis represents the total driver income. The horizontal axis represents the number of update epochs for PPO plots and number of training episodes for DQN. Driver-centric rewards learn effectively in these larger scale environments, whereas both algorithms have problems optimizing policies with system-centric rewards.