A Deep Value-based Policy Search Approach for Real-world Vehicle Repositioning on Mobility-on-Demand Platforms

Yan Jiao[1], **Xiaocheng Tang**[1], **Zhiwei (Tony) Qin**[1]*, **Shuaiji Li**[1], **Fan Zhang**[2], **Hongtu Zhu**[2], **Jieping Ye** [2]
[1]DiDi Research America, Mountain View, CA 94943
[2]Didi Chuxing, Beijing, China
{yanjiao,xiaochengtang,qinzhiwei,shuaijili,feynmanzhangfan,zhuhongtu,yejieping}@didiglobal.com

Abstract

We present a new practical framework based on deep reinforcement learning and decision-time planning for real-world vehicle repositioning on ride-hailing (also known as mobility-on-demand, MoD) platforms. Our approach learns the driver-perspective state-value function using a batch training algorithm with spatiotemproal deep value networks. The optimal repositioning action is generated on-demand through value-based policy search, which combines planning and bootstrapping with the value networks. We benchmark our algorithm with baselines in an MoD simulation environment to demonstrate its superiority in improving the income rate. We have also designed and run a real-world experiment program with regular drivers on a major MoD platform. We have observed positive results on key metrics comparing our method with a group drivers who performed idle-time repositioning based on their own experience and knowledge.

1 Introduction

Mobility-on-demand (MoD), or ride-hailing platforms have gained incredible popularity worldwide, thanks to the rising urban population and the consequent need for more availability of on-demand transportation. An MoD platform connects people with travel needs to drivers with vacant vehicles, greatly tapping into a larger supply pool and reducing the waiting time for getting a ride. A more efficient MoD system offers better user experience for both driver and passenger groups: Drivers would be able to generate higher income through reduced idle time. Passengers would enjoy shorter waiting time before their trips get fulfilled. Vehicle repositioning [10, 5, 8] is an important lever to reduce driver idle time and increase the overall efficiency of an MoD system, by proactively deploying idle vehicles to a specific location in anticipation of future demand at the destination or beyond. It is also an integral operational component of an autonomous MoD system, where vehicles are fully managed by the dispatching algorithm. On the other hand, order dispatching [17, 16, 13, 9, 4] matches idle drivers (vehicles) to open trip orders. From the view of spatiotemporal movement, order dispatching can be thought as a special case of vehicle repositioning through fulfilling trip orders.

In this paper, we consider the problem of repositioning for a small group of vehicles (relative to the entire vehicle population) on an MoD platform. Our objective is to learn an optimal policy that maximizes an individual driver's daily income rate, measured by income-per-hour (IPH). Solutions to this problem have important business use cases, for example, (a) an intelligent driver assistant for those who are new to an MoD platform to help them quickly ramp up by providing learning-based idle-time cruising strategies, and (b) management of an autonomous vehicle fleet within a large-scale

*corresponding author

Presented at Deep RL Workshop, NeurIPS 2020.

MoD system. In such scenarios, each vehicle can be thought as acting independently, since its actions are unlikely to affect the environment, i.e. the overall demand-supply condition.

Many of the works on vehicle repositioning or taxi dispatching are under the setting of autonomous MoD, where a fleet of autonomous vehicles are deployed in an MoD system and fully managed by a controller. *Model predictive control* (MPC) and *receding horizon control* (RHC) [2, 7, 19, 6] are a popular class of methods that involve repeatedly solving a mathematical program using predictions of future costs and system dynamics over a moving time horizon to choose the next control action. Previous works are all grid-based and using discrete time buckets with a short planning horizon due to computational complexity. A number of recent works use reinforcement learning (RL) and deep RL to learn vehicle repositioning (or fleet management) policies, such as dynamic programming [10] and Monte Carlo learning [15]. Through training within a grid-based simulation environment, DQN [8, 1], multi-agent RL [5], and hierarchical RL [3] have been proposed for the repositioning problem, and in some cases, jointly with order dispatching.

In designing the solution framework for the vehicle repositioning problem, there are a few factors for consideration. First of all, the algorithm has to be practical for implementation on real-world production system and for real-time execution. That means we would not be able to make significant simplification assumptions that many some prior works have done. In particular, grid-based algorithms with low granularity and those require end-to-end training in a simulation environment are hard to deploy in our setting. Second, our objective is to maximize an agent's daily income rate, which is long-term reward optimization on the scale of trips. MPC with long horizon is expensive to solve. A coarse discretization for the time would render the solution hard to implement and execute. On the other hand, RL, which focuses on long-term values, is well-suited for such objectives. Third, data from regular MoD system is usually incomplete regarding idle-time repositioning. It is thus hard to learn a state-action value function for repositioning directly from data.

Considering the factors discussed above, we have developed a solution framework that combines offline batch RL and decision-time planning for guiding vehicle repositioning. We model the problem within a semi-Markov decision process (semi-MDP) framework [12], which optimizes a long-term cumulative reward (daily income rate) and models the impact of temporally extended actions (repositioning movements) on the long-term objective through state transitions along a policy. We learn the state value function using tailored spatiotemporal deep value networks trained within a batch RL framework with dual policy evaluaton. We then use the state-value function and learned knowledge about the environment dynamics to develop a value-based policy search algorithm for real-time vehicle repositioning, which is a type of decision-time planning algorithm [11] and can be easily plugged into a *generalized policy iteration* framework [11] for continuous improvement.

1.1 Contributions

Our contributions in this paper come in two folds: 1. We have developed a practical framework for vehicle repositioning on MoD platforms. Our method does not require a simulation environment for training, can work on incomplete trajectory data, and is sufficiently adaptive for production deployment and real-time execution. 2. We have implemented our proposed solution on a major MoD platform and developed a real-world experiment program with carefully designed incentive and operational schemes that allow successful testing of the repositioning algorithm. We observe positive empirical results and report practical experience from this program, which we believe is the first real-world deployment of an RL-based vehicle repositioning algorithm on an MoD platform.

2 Problem Formulation

We start from the environment dynamics. The driver, when idle, can be assigned to a nearby trip order by the MoD platform. Order dispatching/matching takes place in a batch fashion typically with a time window of a few seconds [13, 17, 18]. Trip fee is collected upon the completion of the trip. After dropping off the passenger, the driver becomes idle. If the idle time exceeds a threshold of L minutes (typically five to ten minutes), the driver performs repositioning by cruising to a specific destination, incurring a non-positive cost. If the driver is to stay around the current location, he/she stays for L

minutes before another repositioning could be triggered. During the course of any repositioning, the driver is still eligible for order assignment. [2]

Mathematically, we model this process by a semi-MDP with the agent being the driver as follows. The driver's *state* s contains basic spatiotemporal information of location l and time t, and it can include additional supply-demand contextual features.[3] The eligible actions for the agent include both vehicle repositioning and order fulfillment (as a result of order dispatching). These actions are temporally extended, so they are *options* in the context of a semi-MDP and are denoted by o. A basic repositioning task is to go towards a destination in one of the neighboring grid cells of the driver, including the current cell. The time duration of a repositioning or order fulfillment option is τ_o. The price of the trip corresponding to an option is $p_o > 0$. The cost of a repositioning option is $c_o \leq 0$. Hence, the immediate reward of a transition is $r = c_o$ for repositioning and $r = p_o$ for order fulfillment. The corresponding estimated version of τ_o, p_o, and c_o are $\hat{\tau}_o$, $\hat{p}_o$, and $\hat{c}_o$, respectively. An *episode* of this semi-MDP runs till the end of a day, that is, a state with its time component at midnight is terminal. We denote the repositioning and the order dispatching policies separately by π_r and π_d, and the joint policy by $\pi := (\pi_r, \pi_d)$. In this paper, we focus on learning the repositioning policy π_r and assume that π_d is exogenous and fixed. At any decision point only the repositioning options need to be considered. Order dispatching is executed automatically following a given dispatching policy π_{d0}. The state-option value function is denoted by $Q^{\pi_r}(s, o)$, with the understanding that it is also associated with π_{d0}. $\hat{Q}$ denotes the approximation of the Q-function. In the following sections, we develop a planning method to compute $\hat{Q}(s, o)$ for a particular s so that the repositioning agent would be able to select the best movement at each decision point. Our objective is to maximize the daily cumulative income rate of a driver, which is the ratio of the total price of the trips completed during a day and the total online hours logged by the driver.

3 Learning State Values and Environment

We assume the following environment model per the dynamics and learn the state values associated with the semi-MDP above. At state s, the probability of the driver being dispatched is $p_d^{(s)}$. The probability of being idle within the time interval of L minutes is $p_{id}^{(s)} = 1 - p_d^{(s)}$, upon which repositioning is triggered next, and the driver can either stay around or move to another location. If order dispatch takes place, the driver is relocated to the destination of the trip. The estimated time interval for transitioning from the current state s_0 to the target state s_i is $t_{0i} := \Delta t(s_0, s_i)$. Transitions for both vehicle repositioning and order fulfillment are deterministic, given the current state and option. This is illustrated in Figure 1a.

We decompose the state value function into four components by conditioning on whether the driver is being dispatched or not, e.g.,

$$V(s) = p_d^{(s)} V(s|dispatch) + p_{id}^{(s)} V(s|idle), \tag{1}$$

where $V(s|dispatch)$ and $V(s|idle)$ are the corresponding long-term value function conditioned on whether the associated driver is being dispatched or not at state s.

To facilitate discussions, let us use a different option definition from the option o introduced in Section 2. Consider, for now, simply a binary option b under the same semi-MDP framework with 0 indicating idle and 1 as dispatched. $V(s|dispatch)$ and $V(s|idle)$ can then be represented as $V(s|b = 1)$ and $V(s|b = 0)$, respectively. The dispatch probability $p_d^{(s)}$ is equivalent to $\pi^b(b = 1|s)$, given the behavior policy under the option b as π^b,

Learning of (1) can be done by evaluating the behavior policy on the observed transactions collected from the MoD platform. In particular, we apply variance reduction techniques to the joint learning of both $V(s)$ and the conditional value function $V(s|b)$. Here we directly learn $V(s)$ from data instead of computing it using (1) in order to minimize the errors due to estimations and compositions. The dispatch probabilities are estimated separately after applying negative sampling to drivers' trajectories. Next we describe the approach in details.

[2]For modeling purpose, we assume that the driver can always complete a repositioning task before the platform would match him/her to an order, but the policy is not restricted by this assumption in any way.

[3]In our experiments, only the basic spatiotemporal features are used.

3.1 Dual Policy Evaluation

Notice that $V(s|b)$ is equivalent to the state-option value function $Q^{\pi^b}(s, b)$ under the binary policy π^b. Evaluation of $V(s|b)$ can thus be done using standard TD algorithms like SARSA [11]. In this work we propose Dual Policy Evaluation (DPE) that prevents stochasticity in the policy from increasing variance. It does so by jointly learning $V(s|b)$ and $V(s)$ while basing the update of $V(s|b)$, not on the next-state value function $V(s'|b')$, but on its expected value $V(s')$. Formally, consider the k-step transition from s_0 to s_k by applying the option b. We can write down the k-step Bellman equation as follows,

$$V(s_0|b) \leftarrow \frac{R_b(\gamma^k - 1)}{k(\gamma - 1)} + \gamma^k V(s_k),\ b \in \{0, 1\}, \tag{2}$$

where the value function $V(s_k)$ in the RHS is updated using the standard value iteration, e.g., $V(s_0) \leftarrow \frac{R_b(\gamma^k-1)}{k(\gamma-1)} + \gamma^k V(s_k)$ which is the same target as (2) but with inputs unconditioned on the option b. R_b is the reward from the option which is either the cost of idle movement (≤ 0) or the trip fee (> 0) depending on whether the option b is 0 or 1. The time discounting technique proposed in [13] is applied to the reward as in (2). The γ is the discount factor between 0 and 1 and $k \geq 1$ is the transition steps.

The update equations (2) is, in essence, similar to expected SARSA [14]. The main difference is that expected SARSA uses empirical samples to approximate the expected value while DPE does so by directly learning a separate function approximator. To see that, note that $V(s)$ can be considered as the marginal expectation of the conditional value function $V(s|b)$, e.g., $V(s_k) = E_b(V(s_k|b))$. Hence the RHS of the updates in (2) is directly using the expectation, e.g., $\frac{R_b(\gamma^k-1)}{k(\gamma-1)} + \gamma^k E_b(V(s_k|b))$ while expected SARSA uses empirical samples $\mathcal{S}_k$ starting from the state s_k to approximate the corresponding expectation, e.g., $\frac{R_b(\gamma^k-1)}{k(\gamma-1)} + \frac{1}{|\mathcal{S}_k|}\gamma^k \sum_{b' \in \mathcal{S}_k} V(s_k|b')$. The overhead of learning two policies is minimum in this case since both $V(s|b)$ and $V(s)$ are required for the value-based policy search as described in Section 4.

Similar to [13], we use a neural network to represent the value function, but the training is different since we now maintain and update both the conditional value network $V(s|b)$ and the marginalized one $V(s)$. We employ the same state representation and training techniques as introduced by [13]. Besides, for the conditional network we engage a separate embedding matrix to encode the binary option and use the multiplicative form to force interactions between the state features and the option embedding. Both $V(s|b)$ and $V(s)$ share the same state representation but have a separate branch for the output. An algorithm statement of DPE can be found in Supplementary Material.

3.2 Dispatch Probabilities

We estimate the dispatching probability $p_d^{(s)} = \pi^b(b = 1|s)$ by maximizing its log-likelihood on the marketplace transaction data. To generate the training data we collect drivers' historical trajectories including the descriptions of completed trips as well as the online and offline states. We use the states when the driver receives the trip request as the positive examples indicating the option b being 1. For the negative examples we are unable to enumerate all possibilities considering the limited observed trajectories and the system complexity. To that end we perform negative samplings. The negative examples we use for training are drivers' starting states of idle transaction in-between orders as well as the states when they become active or inactive. The training is done using one-month driver trajectories. Experiments on hold-out datasets show that the learned estimator achieves AUC of 0.867 ± 0.010 across multiple days and cities. Detailed results are presented in Supplementary Material.

4 Value-based Policy Search (VPS)

With the deep value network learned within the training framework in Section 3, the optimal action is selected using decision-time planning when repositioning is triggered. Specifically, we use our environment model to carry out planning of potentially multiple steps from the particular state s_0, the state of the agent when repositioning is triggered. This allows us to evaluate at run-time the state-option pairs associated with s_0 and all the available repositioning options at that state and select the best move for the next step. The next time repositioning is triggered, the same planning process is repeated.

We want to estimate $Q^*(s_0, o)$ associated with the optimal policy π_r^* and the given π_{d0}, so that $o^* = \arg\max_o \hat{Q}^*(s_0, o)$ gives the approximate optimal repositioning at decision-time with respect to a given dispatch policy. The one-step expansion per the environment model in Section 3 writes $Q^*(s_0, o) = r^{(0,1)} + (V^*)^{(t_{01})}(s_1)$, where $r^{(0,1)} \le 0$ is the repositioning cost from s_0 to s_1, and s_1 is the state after repositioning o, with location l_1 and time $t_0 + t_{01}$. V^* is the state-value function associated with π_r^* (and the given π_{d0}). To make the time component of the input explicit, $(V^*)^{(t_{01})}$ is the same value function with time component t_{01} ahead of the decision-time t_0. *All value functions with a future time component are assumed to be properly discounted without clogging the notation.* The discount factor is $\gamma^{(t-t_0)}$, where t is discretized time component for the input state, and t_0 is the time for current decision point. The duration that incurs cost r is Δt, and the starting time for the cost is t. The cost used in this paper is also properly discounted: $r \leftarrow \frac{\gamma^{(t-t_0)} r(\gamma^{\Delta t}-1)}{\Delta t(\gamma-1)}$.

In practice, we use the state-value function V learned in Section 3 to replace V^*, writing $V^{(t_{01})}$ concisely as $V_1^{(t_{01})}$. Then,

$$\hat{Q}^*(s_0, o) = r^{(0,1)} + V_1^{(t_{01})}. \tag{3}$$

That is, that the one-step expansion renders a greedy policy by selecting the repositioning movement leading to the next-state with the highest value given by the state value networks, V. We also notice that $\hat{Q}^*(s_0, o)$ is in fact $Q^{\pi_0}(s_0, o)$ in this case because V is computed by policy evaluation on historical data generated by π_0. Hence, finding the optimal option to execute by $o^* = \arg\max_o \hat{Q}^*(s_0, o)$ is essentially one-step policy improvement in generalized policy iterations.

We can use our environment model to expand $\hat{Q}^*$ further:

$$\hat{Q}^*(s_0, o) = r^{(0,1)} + p_d^{(1)} \hat{V}^*(s_1|dispatch) + p_{id}^{(1)} \hat{V}^*(s_1|idle). \tag{4}$$

$p_d^{(1)}$ is the dispatch probability at s_1, and we use the conditional value networks discussed in Section 3, denoted by $\tilde{V}_1^{(t_{01})}$, for $\hat{V}^*(s_1|dispatch)$. When the driver is idle, the immediate next option has to be a reposition, so we have $\hat{V}^*(s_1|idle) = \max_j \hat{Q}^*(s_1, o_j)$, where o_j is the second-step reposition option. $\hat{Q}^*(s_0, o)$ can be recursively expanded, eventually written in terms of the given estimated state-value function.

A two-step expansion is that

$$\begin{aligned}\hat{Q}^*(s_0, o) &= r^{(0,1)} + p_d^{(1)} \tilde{V}_1^{(t_{01})} + p_{id}^{(1)} \max_j \hat{Q}^*(s_1, o_j) \\ &= r^{(0,1)} + p_d^{(1)} \tilde{V}_1^{(t_{01})} + p_{id}^{(1)} \max\left\{ \max_{j \neq 1} r^{(1,j)} + V_j^{(t_{0j})}, V_1^{(t_{01}+L)} \right\}.\end{aligned} \tag{5}$$

For a three-step expansion, we further write

$$\hat{Q}^*(s_1, o_j) = r^{(1,j)} + p_d^{(j)} \tilde{V}_j^{(t_{0j})} + p_{id}^{(j)} \max\left\{ \max_{k \neq j} r^{(j,k)} + V_k^{(t_{0k})}, V_j^{(t_{0j}+L)} \right\}. \tag{6}$$

In the above equations, $t_{0j} := t_{01} + t_{1j}, t_{0k} := t_{01} + t_{1j} + t_{jk}$, are the total ETA of two-step and three-step repositions respectively. Figure 1b summarizes the process of our value-based policy search.

We can interpret the above decision-time planning through path value approximation. For repositioning, not only the destination cell is important, the route is important as well, since the driver may be matched to orders along the way, generating income. Our approach can be viewed as computing the expected values of n-step look-ahead repositioning paths from the current spatiotemporal state, selecting the optimal one, and then executing its first step. An n-step look-ahead path also corresponds to a path from the root to a leaf node with a depth of $2n$ in Figure 1b. We show in Section 4.1 that selecting the max-value path is equivalent to $o^* = \arg\max_o \hat{Q}^*(s_0, o)$ in terms of the action executed. The new data generated by the current learned policy is collected over time and can be used to update the state-value function, which in turn updates the policy through decision-time planning.

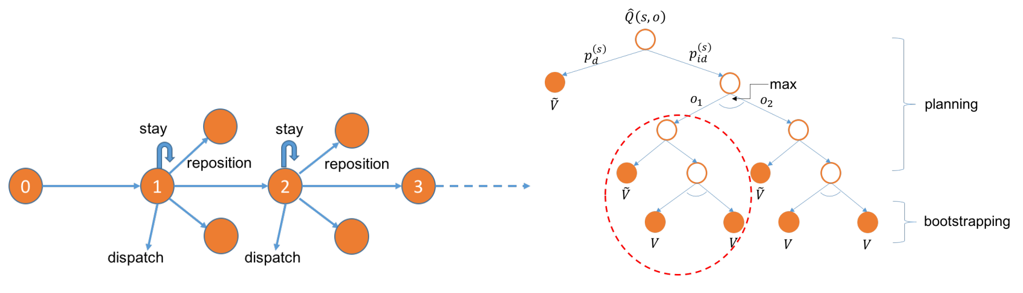


Figure 1: (a) Environment dynamics model. Any order dispatch happening during 'reposition' or 'stay' is assumed to take place at the next node. (b) Planning + bootstrapping perspective of the proposed algorithm. At the leaf nodes, bootstrapping is used through the given state-value function V. The subtree within the red loop illustrates the back-up of values.

4.1 Practical Implementation

Our implementation of the policy search algorithm (VPS) is based on the path-value perspective instead of directly implementing (3) to (6). The reason is that the latter does not allow batch inference of the state value networks, which is crucial for fast computation. We first show the equivalence of the two approaches taking the three-step expansion as an example.

By convention, for the case of staying at the same location ($j = k$), $r^{(j,k)} = 0$ and $t_{0k} = t_{0j} + L$. Then, we have $\max\left\{\max_{k\neq j}\left\{r^{(j,k)} + V_k^{(t_{0k})}\right\}, V_j^{(t_{0j}+1)}\right\} = \max_k\left\{r^{(j,k)} + V_k^{(t_{0k})}\right\}$. Through further transformation, we get

$$\begin{aligned}
\hat{Q}^*(s,o) &= r^{(0,1)} + p_d^{(1)}\tilde{V}(s_1) + p_{id}^{(1)}(\max_j \hat{Q}^*(s_1,o_j)) \\
&= r^{(0,1)} + p_d^{(1)}\tilde{V}(s_1) + p_{id}^{(1)} \max_j \quad \left\{r^{(1,j)} + p_d^{(j)}\tilde{V}(s_j) + p_{id}^{(j)} \max_k \left\{r^{(j,k)} + V_k^{(t_{0k})}\right\}\right\} \\
&= r^{(0,1)} + p_d^{(1)}\tilde{V}(s_1) + p_{id}^{(1)} \max_{j,k} \quad \left\{r^{(1,j)} + p_d^{(j)}\tilde{V}(s_j) + p_{id}^{(j)}(r^{(j,k)} + V_k^{(t_{0k})})\right\} \\
&= \max_{j,k}\left\{r^{(0,1)} + p_d^{(1)}\tilde{V}(s_1) \quad + p_{id}^{(1)}(r^{(1,j)} + p_d^{(j)}\tilde{V}(s_j) + p_{id}^{(j)}(r^{(j,k)} + V_k^{(t_{0k})}))\right\}. \qquad (7)
\end{aligned}$$

Equation (7) shows that the state-option value that we use for decision-time planning is the maximum expected value of the three-step paths with o being the first step. Our algorithm based on (7) is split into two phases. First, we generate all the paths of a certain length originating at the current grid cell using breadth-first-search. Second, we calculate the value of each path and select the first step of the path which has the maximum value as our repositioning action. Keeping two separate phases has the benefit of allowing batch inference of the state value networks model in the second phase. We will now describe the details of each phase.

Path generation Each city is divided into hexagon cells, each covering an equal size area. In each grid cell, we use the pick-up points (from an in-house map service) as the candidate repositioning destinations. Some of those cells might be "inaccessible"(e.g. lakes, rivers, mountains) or might be missing well-known pick-up points (e.g. low population density areas). These cells are excluded from our search.

The main parameter for the path generation is the maximum path length that our algorithm should consider. We call this parameter the *expansion depth* (also from the perspective of (5) and (6)). Our algorithm then uses a breadth-first strategy to generate paths of a maximal length no longer than the expansion depth. We have found that a small depth works the best in practice. Larger depth values would result in accumulation of errors introduced by model assumption and estimation, which outweighs the benefits of the models. We also note that with bootstrapping the state values, the path values account for an horizon far beyond the expansion depth. Because of the presence of invalid cells, it may happen that no generated path reaches the search depth. In the infrequent case where no cell is valid within the search depth distance to the origin cell, we expand our breadth-first search to the city limit to find the closest valid cells to the current cell and use those as repositioning candidates.

Step selection Once a set of paths has been generated, depending on the length reached, our algorithm applies a different formula to each path in order to calculate its value, e.g., (7) being applied to the paths of length three, and (3) being applied to unit-length paths. Finally, the algorithm returns as reposition action the first step of the path which has the maximum value.

We have developed a highly efficient implementation of VPS that has met the strict production requirement on latency for our real-world experiment. Since the vehicle repositioning requests can be processed independently, the proposed algorithm is horizontally scalable.

5 Experiments

We describe our simulation and real-world experiments in this section and discuss empirical results and observations. Additional experiment set-up details and numerical results can be found in Supplementary Material.

5.1 Simulation Environments

We have developed a realistic simulation environment for evaluating vehicle repositioning policies based on prior works on order dispatching for MoD platforms [13, 17]. This environment captures all the essential elements of a large-scale MoD system. We emphasize that our proposed algorithms do not require the simulation environment for training. In this environment, one can specify a given number of vehicles to follow a particular repositioning policy π_r. A fixed order dispatching policy π_d assigns trip orders to the drivers in batch windows, and minimizes the within-batch total pick-up distance. In addition, order dispatching can interrupt a repositioning action.

5.2 Simulation Results

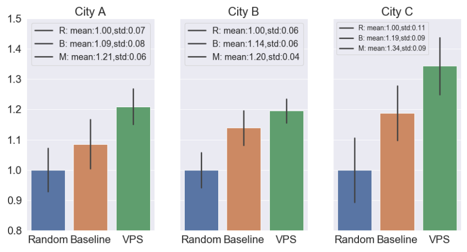
We benchmarked our algorithm in the simulation environment generated by MoD trip data from three mid-sized cities, which we denote by A, B, and C. Ten drivers are selected for each experiment out of hundreds of drivers in the environment. We chose three different days, and for each city and each day, we use five random seeds, which give different initial states for the experiment. We compare three algorithms: *Random*: This policy randomly selects a grid cell from the six neighbouring cells and the current location as the repositioning destination. *Baseline*: This is the implementation of (3), which is one-step policy improvement of the generalized policy iterations. *VPS*: This is the value-based policy search algorithm described in Section 4. An expansion depth of two is chosen by benchmarking different expansion depths (see Supplementary Material).

We computed the average income rate across three days for each random seed. We then compare for the three methods the mean and standard deviation of the five random seeds results. We can see from Figure 2a that both VPS and Baseline consistently outperform Random across the three cities. Moreover, VPS yields significantly higher income rates than Baseline, with 12%, 6%, and 15% improvement in the mean for each of the three cities respectively.

5.3 Real-world Experiment

We designed and ran a field experiment program on a major MoD platform to evaluate the performance of our proposed repositioning algorithm (VPS) in the real-world environment. The goal is to compare VPS with human expert repositioning strategies, which is significantly more challenging than the simulation baselines. Human drivers have obtained the domain knowledge through experience and the real-time supply-demand information provided by the app.

Unlike operating on an autonomous MoD platform, testing a vehicle repositioning algorithm with human drivers in our case requires additional consideration of the way the repositioning recommendations are delivered and the drivers' willingness to accept and follow the recommendations, because the drivers within the experiment program were on a voluntary basis in terms of executing the repositioning tasks, due to various practical constraints. We designed a hierarchical incentive program



(a)

City	Group	Mean	Median	+% Mean	+% Median
A	human	1	0.9634	-	-
A	algo	1.003	0.9653	0.3	0.2
B	human	1	0.9958	-	-
B	algo	1.061	1.0303	6.1	3.5
C	human	1	0.9677	-	-
C	algo	1.131	1.0030	13.1	3.7

(b)

Figure 2: (a) Simulation experiment results comparing income rates of Random, Greedy (Baseline) and VPS policy using the multi-driver simulation environment. Results are normalized with respect to the mean Random. (b) Daily income rate comparison for regular drivers without preferential order matching in the real-world experiment, normalized w.r.t. the means of the *human* group.

to ensure that the drivers are online for a sufficient period of time, and they follow the repositioning recommendations as closely as possible. Details of the program set-up are in Supplementary Material.

We recruited 61 experienced drivers across the same three cities as in the simulation experiments to participate in our experiment, which lasted for two weeks. The drivers were randomly divided into two groups of similar sizes. The *algo* group received repositioning recommendations from our proposed algorithm. The *human* group did not receive any recommendation, and they were free to determine their own cruising plan while idle. Order dispatching was handled by the platform and was exogenous to both groups.

5.4 Experiment Results

First, we look at the income rates across the three cities. We have found that the advantage of algorithmic repositioning appears most significant for the group of regular drivers without preferential order matching.[4] Figure 2b shows daily income rate comparison between the *algo* group and the *human* group by city. Cities B and C both show significant improvement in income rate, with a difference of 6.1% and 13.1% in the mean respectively. There is also a relative improvement of more than 3% in the median for both cities. Over the entire group of drivers in the program, the relative improvement of the *algo* group over the *human* group is up to 2.6% across the three citie. We surmise that regular drivers without preferential order matching were more likely to be in need of idle-time cruising guidance as they have a higher chance of becoming idle during the day, thus offering larger room for improvement through algorithmic repositioning.

Since our participation requirements to the drivers were voluntary and incentive-based, all the drivers might not have accepted or completed all the repositioning tasks that our algorithm issued. We analyzed the *algo* group data with more than five hours of daily online time and compared daily income rates of the drivers who followed repositioning instructions closely to those of whom failed. We have found that those drivers following the algorithmic repositioning faithfully enjoy an average income rate 9% (4% for median resp.) higher than those not, corroborating the effectiveness of our algorithm. A participants survey after the experiment program showed that the overwhelming majority of the drivers rated the program positively and would like to participate again.

6 Conclusion and Future Direction

We have presented a new vehicle repositioning algorithm designed for production deployment on MoD platforms. This method combines deep value-networks and decision-time planning via graph search for long-term value maximization and effective accommodation of practical constraints. In addition to simulation experiments, we designed and ran a real-world experiment program on a major MoD platform to benchmark our approach against human expert strategies. With positive results obtained from both simulation and real-world experiments, we are working further along the direction

[4]Preferential order matching is typically applied to drivers with high service scores as incentives.

of real-world group vehicle repositioning (i.e. fleet management), which requires coordination and can potentially be applied to city-scale driver populations.

Broader Impact

As is typical for ride hailing applications, driver income is highly influenced by the experience and knowledge of the drivers. New drivers may have a tough time at the beginning. With the help of the proposed method, their initial income could increase significantly. This will improve the driver onboarding experience and, consequently, driver retention. Higher income also help improve driver satisfaction, which often translates into higher service quality, improving the ridership experience for passengers as well. In terms of potential negative impact, the main risk comes from having an AI-powered system: there could be unintended edge cases or biases brought in from the training data. Therefore, continuous feedback from the operations team and the drivers in the pilot programs is highly desirable.

Aknowledgement

The authors would like to thank Jing He, Jing Chen, Yuanfang Wang, Shaohui Su, Chunyang Liu, Zhe Xu, Ben Wang, Bin Liu, Chengyi Shi, the DiDi regional operations teams, and other sister engineering teams for their support and efforts in product, platform, and operations for the experiment program of this research.

References

[1] J. Holler, R. Vuorio, Z. Qin, X. Tang, Y. Jiao, T. Jin, S. Singh, C. Wang, and J. Ye. Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem. In J. Wang, K. Shim, and X. Wu, editors, *2019 IEEE International Conference on Data Mining (ICDM)*, pages 1090–1095. Institute of Electrical and Electronics Engineers, Washington, DC, 2019.

[2] R. Iglesias, F. Rossi, K. Wang, D. Hallac, J. Leskovec, and M. Pavone. Data-driven model predictive control of autonomous mobility-on-demand systems. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7. IEEE, 2018.

[3] J. Jin, M. Zhou, W. Zhang, M. Li, Z. Guo, Z. Qin, Y. Jiao, X. Tang, C. Wang, J. Wang, et al. Coride: Joint order dispatching and fleet management for multi-scale ride-hailing platforms. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1983–1992, 2019.

[4] N. Korolko, D. Woodard, C. Yan, and H. Zhu. Dynamic pricing and matching in ride-hailing platforms. *Available at SSRN*, 2018.

[5] K. Lin, R. Zhao, Z. Xu, and J. Zhou. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1774–1783, 2018.

[6] F. Miao, S. Han, S. Lin, J. A. Stankovic, D. Zhang, S. Munir, H. Huang, T. He, and G. J. Pappas. Taxi dispatch with real-time sensing data in metropolitan areas: A receding horizon control approach. *IEEE Transactions on Automation Science and Engineering*, 13(2):463–478, 2016.

[7] J. Miller and J. P. How. Predictive positioning and quality of service ridesharing for campus mobility on demand systems. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1402–1408. IEEE, 2017.

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

[9] E. Ozkan and A. Ward. Dynamic matching for real-time ridesharing. *Available at SSRN 2844451*, 2017.

[10] Z. Shou, X. Di, J. Ye, H. Zhu, H. Zhang, and R. Hampshire. Optimal passenger-seeking policies on e-hailing platforms using markov decision process and imitation learning. *Transportation Research Part C: Emerging Technologies*, 111 (February):91–113, February 2020.

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

[12] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

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

[14] H. van Seijen, H. van Hasselt, S. Whiteson, and M. Wiering. A theoretical and empirical analysis of expected sarsa. In *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 177–184, March 2009.

[15] T. Verma, P. Varakantham, S. Kraus, and H. C. Lau. Augmenting decisions of taxi drivers through reinforcement learning for improving revenues. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*, 2017.

[16] Z. Wang, Z. Qin, X. Tang, J. Ye, and H. Zhu. Deep reinforcement learning with knowledge transfer for online rides order dispatching. In *International Conference on Data Mining*. IEEE, 2018.

[17] 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*, pages 905–913. ACM, 2018.

[18] 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*, pages 2151–2159. ACM, 2017.

[19] R. Zhang, F. Rossi, and M. Pavone. Model predictive control of autonomous mobility-on-demand systems. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1382–1389. IEEE, 2016.