Neural Approximate Dynamic Programming for On-Demand Ride-Pooling

Sanket Shah, Meghna Lowalekar, Pradeep Varakantham
School of Information Systems, Singapore Management University
{sankets, pradeepv}@smu.edu.sg, meghnal.2015@phdcs.smu.edu.sg

Abstract

On-demand ride-pooling (e.g., UberPool, LyftLine, GrabShare) has recently become popular because of its ability to lower costs for passengers while simultaneously increasing revenue for drivers and aggregation companies (e.g., Uber). Unlike in Taxi on Demand (ToD) services – where a vehicle is assigned one passenger at a time – in on-demand ride-pooling, each vehicle must simultaneously serve multiple passengers with heterogeneous origin and destination pairs without violating any quality constraints. To ensure near real-time response, existing solutions to the real-time ride-pooling problem are myopic in that they optimise the objective (e.g., maximise the number of passengers served) for the current time step without considering the effect such an assignment could have on assignments in future time steps. However, considering the future effects of an assignment that also has to consider what combinations of passenger requests can be assigned to vehicles adds a layer of combinatorial complexity to the already challenging problem of considering future effects in the ToD case.

A popular approach that addresses the limitations of myopic assignments in ToD problems is Approximate Dynamic Programming (ADP). Existing ADP methods for ToD can only handle Linear Program (LP) based assignments, however, as the value update relies on dual values from the LP. The assignment problem in ride pooling requires an Integer Linear Program (ILP) that has bad LP relaxations. Therefore, our key technical contribution is in providing a general ADP method that can learn from the ILP based assignment found in ride-pooling. Additionally, we handle the extra combinatorial complexity from combinations of passenger requests by using a Neural Network based approximate value function and show a connection to Deep Reinforcement Learning that allows us to learn this value-function with increased stability and sample-efficiency. We show that our approach easily outperforms leading approaches for on-demand ride-pooling on a real-world dataset by up to 16%, a significant improvement in city-scale transportation problems.

1 Introduction

On-demand ride-pooling, exemplified by UberPool, LyftLine, GrabShare, etc., has become hugely popular in major cities with 20% of all Uber trips coming from their ride-pool offering UberPool (Heath 2016; Gessner 2019). Apart from reducing emissions and traffic congestion compared to Taxi/car on-Demand (ToD) services (e.g., UberX, Lyft), it benefits all the stakeholders involved: (a) Individual passengers have reduced costs as these are shared by overlapping passengers; (b) Vehicles make more money per trip as multiple passengers (or passenger groups) are present; (c) For the centralized entity (like Uber, Lyft etc.) more customer requests can be satisfied with the same number of vehicles.

Underlying these on-demand ride-pooling services is the Ride-Pool Matching Problem (RMP) (Alonso-Mora et al. 2017; Bei and Zhang 2018; Lowalekar, Varakantham, and Jaillet 2019). The objective in the RMP is to assign groups of user requests to vehicles that can serve them, online, subject to predefined quality constraints (e.g., the detour delay for a customer cannot be more than 10 minutes) in such a way that a quality metric is maximised (e.g., revenue). The RMP reduces to the taxi/car-on-demand (ToD) problem when the capacity, i.e., the maximum number of simultaneous passengers (with different origin and destination pairs) that can be served by a vehicle, is 1. In this paper, we consider the most general version of the RMP, in which batches of requests are assigned to vehicles of arbitrary capacity, and present a solution that can scale to practical use-cases involving thousands of locations and vehicles.

Past research in solving RMP problems can be categorized along four threads. Past work along the first thread employs traditional planning approaches to model the RMP as an optimisation problem (Ropke and Cordeau 2009; Ritzinger, Puchinger, and Hartl 2016; Parragh, Doerner, and Hartl 2008). The problem with this class of approaches is that they don't scale to on-demand city-scale scenarios. The second thread consists of approaches that make the best greedy assignments (Ma, Zheng, and Wolfson 2013; Tong et al. 2018; Huang et al. 2014; Lowalekar, Varakantham, and Jaillet 2019; Alonso-Mora et al. 2017). While these scale well, they are myopic and, as a result, do not consider the impact of a given assignment on future assignments. The third thread, consists of approaches that use Reinforcement Learning (RL) to address the myopia associated with approaches from the second category for the ToD problem (Xu et al. 2018; Lin et al. 2018; Li et al. 2019; Wang et al. 2018; Verma et al. 2017). Past RL work for the ToD problem cannot be extended to solve the RMP, however, because it relies heavily on the assumption that vehicles can

only serve one passenger at a time.

Lastly, there has been work in operations research that uses the Approximate Dynamic Programming (ADP) framework to solve the ToD problem (Powell 2007) and a special case of a capacity-2 RMP (Yu and Shen 2019). In ADP, matching is performed using a learned value function that estimates the future value of performing a certain matching. There are two major reasons for why past ADP approaches for solving the ToD problem cannot immediately be applied to the RMP, however. Firstly, the value function approximation in past work is linear and its update relies heavily on the assumption that matching can be modelled as a Linear Program (LP); this does not hold for the RMP with arbitrary vehicle capacities. Secondly, in the RMP, passengers may be assigned to a partially filled vehicle at each time step. This results in a complex state space for each vehicle that is combinatorial (combinations of requests already assigned) in the vehicle capacity (more details in Section 4).

We make three key contributions in this paper. First, we formulate the arbitrary capacity RMP problem as an Approximate Dynamic Programming problem. Second, we propose Neural ADP (NeurADP), a general ADP method that can learn value functions (approximated using Neural Networks) from ILP based assignment problems. Finally, we bring together techniques from Deep Q-Networks (DQN) (Mnih et al. 2015) to improve the stability and scalability of NeurADP.

In the experiments, we compare our approach to two leading approaches for the RMP on a real-world dataset (NYYellowTaxi 2016). Compared to a baseline approach proposed by (Alonso-Mora et al. 2017), we show that our approach serves up to 16% more seen requests across different parameter settings. This translates to a relative improvement of 40% over the baseline.

2 Background: Approximate Dynamic Programming (ADP)

ADP is a framework based on the Markov Decision Problem (MDP) model for tackling large multi-period stochastic fleet optimisation problems (Powell 2007) such as ToD. The problem is formulated using the tuple $\langle S, A, \xi, T, O\rangle$:

S: denotes the system state with s_t denoting the state of system at decision epoch t.

A: denotes the set of all possible actions [1] (which satisfy the constraints on the action space) with A_t denoting the set of possible actions at decision epoch t. $a_t \in A_t$ is used to denote an action at decision epoch t.

ξ: denotes the exogenous information – the source of randomness in the system. For instance, this would correspond to demand in ToD problems. ξ_t denotes the exogenous information (e.g., demand) at time t.

T: denotes the transition function which describes how the system state evolves over time.

O: denotes the objective function with $o_t(s_t, a_t)$ denoting the value obtained on applying action a_t on state s_t.

[1] We use action and decision interchangeably in the paper.

In an MDP, system evolution happens as $(s_0, a_0, s_1, a_1, s_2, ...)$. However, in an ADP, the evolution happens as $(s_0, a_0, s_0^a, \xi_1, s_1, a_1, s_1^a, \cdots, s_t, a_t, s_t^a, \cdots)$, where s_t denotes the pre-decision state at decision epoch t and s_t^a [2] denotes the post-decision state (Powell 2007). The transition from state s_t to s_{t+1} depends on the action vector a_t and the exogenous information ξ_{t+1}. Therefore,

$$s_{t+1} = T(s_t, a_t, \xi_{t+1})$$

Using post-decision state, this transition can be written as

$$s_t^a = T^a(s_t, a_t); s_{t+1} = T^\xi(s_t^a, \xi_{t+1})$$

Let $V(s_t)$ denotes the value of being in state s_t at decision epoch t, then using Bellman equation we get

$$V(s_t) = \max_{a_t \in A_t} (O(s_t, a_t) + \gamma \mathbb{E}[V(s_{t+1})|s_t, a_t, \xi_{t+1}])$$

where γ is the discount factor. Using post-decision state, this expression can be broken down into two parts:

$$V(s_t) = \max_{a_t \in A_t} (O(s_t, a_t) + \gamma V^a(s_t^a)) \quad (1)$$

$$V^a(s_t^a) = \mathbb{E}[V(s_{t+1})|s_t^a, \xi_{t+1}] \quad (2)$$

The advantage of this decomposition is that Equation 1 can be solved using an LP in fleet optimisation problems. The basic idea in any ADP algorithm is to define a value function approximation around post-decision state, $V^a(s_t^a))$ and to update it by stepping forward through time using sample realizations of exogenous information (i.e. demand in fleet optimisation that is typically observed in data). Please refer to Powell (Powell 2007) for more details.

3 Ride-pool Matching Problem (RMP)

In this problem, we consider a fleet of vehicles/resources $\mathcal{R}$ with random initial locations, travelling on a predefined road network $\mathcal{G}$. Passengers that want to travel from one location to another send requests to a central entity that collects these requests over a time-window called the decision epoch Δ. The goal of the RMP is to match these collected requests U^t to empty or partially filled vehicles that can serve them such that an objective $\mathcal{O}$ is maximised subject to constraints on the delay $\mathcal{D}$. These delay constraints $\mathcal{D}$ are important because customers are only willing to trade being delayed in exchange for a reduced fare up to a point. In this paper, we consider the objective $\mathcal{O}$ to be the number of requests served.

We provide a formal definition for the RMP using the tuple $\langle \mathcal{G}, \mathcal{U}, \mathcal{R}, \mathcal{D}, \Delta, \mathcal{O}\rangle$:

$\mathcal{G}$: $(\mathcal{L}, \mathcal{E})$ is a weighted graph representing the road network. Along the lines of (Alonso-Mora et al. 2017), $\mathcal{L}$ denotes the set of street intersections and $\mathcal{E}$ defines the adjacency of these intersections. Here, the weights correspond to the travel time for a road segment. We assume that vehicles only pick up and drop people off at intersections.

[2] Here a is just used to indicate that it is post decision state and it does not correspond to any specific action.

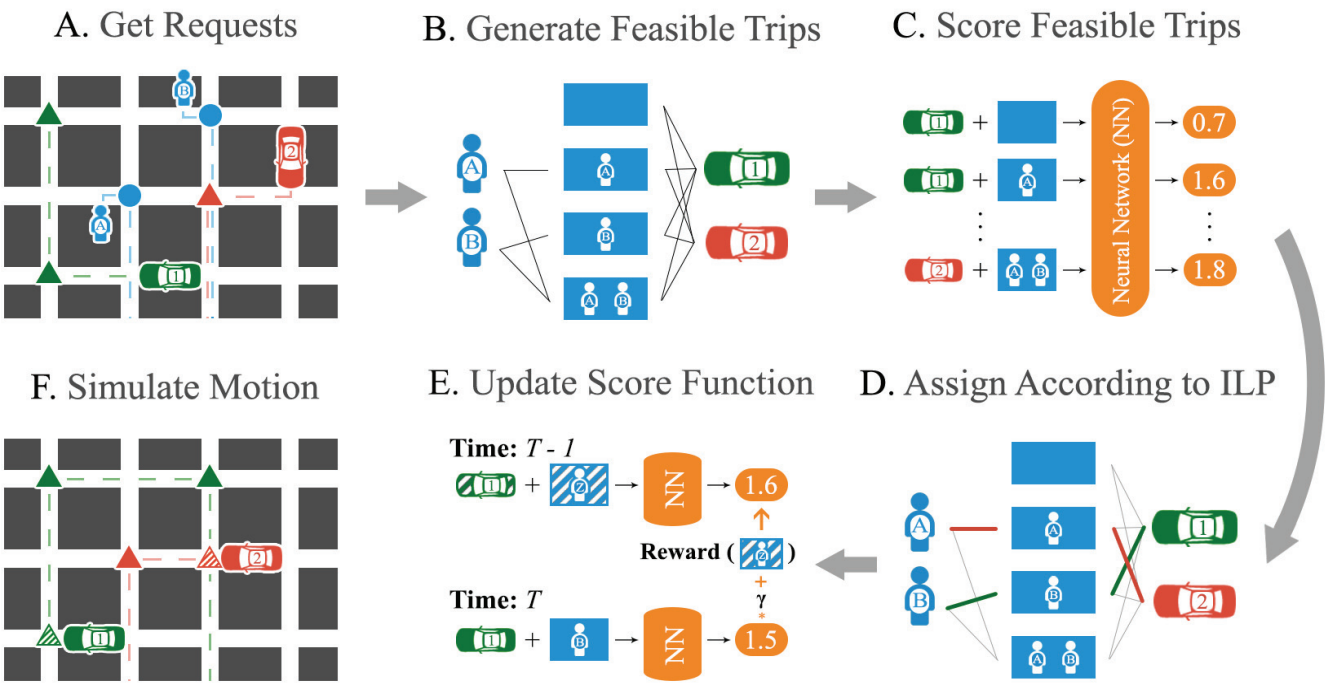

Figure 1: **Schematic outlining our overall approach.** We start with a hypothetical $\mathcal{G}, \mathcal{U}$ and $\mathcal{R}$ in **(A)**. The grid represents a road network. The blue people and circles correspond to user requests and the nearest street intersection that they're mapped to respectively. The blue dotted lines represent the shortest path between the pick-up and drop-off points of a request. The red and green triangles correspond to existing pick-up/drop-off points for the red and green vehicles respectively. The dotted lines describe their current trajectory. In **(B)** we map the requests and their combinations to vehicles that can serve them under the constraints defined by $\mathcal{D}$ to create feasible actions using the approach presented in (Alonso-Mora et al. 2017). In **(C)**, we score each of these feasible actions using our Neural Network Value Function. In **(D)**, we create a mapping of requests to vehicles that maximises the sum of scores generated in (C) using the Integer Linear Program (ILP) in Table 1. In **(E)**, we use this final mapping to update the score function (Section 4). In **(F)**, we simulate the motion of vehicles until the next epoch either based on their current trajectories or a re-balancing strategy. This process then repeats for the next decision epoch.

$\mathcal{U}$: denotes the set of user requests. $\mathcal{U} = \times_t \mathcal{U}_t$ is the combination of requests that we observe at each decision epoch t. Each request $u_t^j \in \mathcal{U}_t$ is represented by the tuple: $\left\langle o_t^j, e_t^j, t \right\rangle$, where $o_t^j, e_t^j \in \mathcal{L}$ denote the origin and destination and t denotes the arrival epoch of the request.

$\mathcal{R}$: denotes the set of resources/vehicles. Each element $i \in \mathcal{R}$ is represented by the tuple $\left\langle c^i, p^i, L^i \right\rangle$. c^i denotes the capacity of the vehicle, i.e., the maximum number of passengers it can carry simultaneously, p^i its current position and L^i the ordered list of locations that the vehicle should visit next to satisfy the requests currently assigned to it.

$\mathcal{D}$: $\{\tau, \lambda\}$ denotes the set of constraints on delay. τ denotes the maximum allowed pick-up delay which is the difference between the arrival time of a request and the time at which a vehicle picks the user up. λ denotes the maximum allowed detour delay which is the difference between the time at which the user arrived at their destination in a shared cab and the time at which they would have arrived if they had taken a single-passenger cab.

Δ: denotes the decision epoch duration.

$\mathcal{O}$: represents the objective, with $\mathcal{O}_t^i$ denoting the value obtained by serving request i at decision epoch t. The goal of the online assignment problem is to maximize the overall objective over a given time horizon, T.

4 NeurADP: Neural Approximate Dynamic Programming

Figure 1 represents the overall framework used for solving the RMP. As shown in the figure, the framework executes 6 steps at each decision epoch to assign incoming user requests to available vehicles. Existing myopic approaches only execute steps (A), (B), (D) and (F). The crucial steps (C) and (E) help in maximising the expected long-term value of serving a request rather than its immediate value. To learn this long-term value, we model the RMP problem using ADP and use deep neural networks to learn the value functions of post-decision states.

In this section, we first indicate key challenges that preclude direct application of existing ADP methods. We next provide the ADP model for the RMP problem and describe

our contributions in using neural function approximations for scalable and effective policies in RMP.

Departure From Past Work

Approximate Dynamic Programming has been used to model many different transportation problems such as fleet management (Simao et al. 2009), ambulance allocation (Maxwell et al. 2010) etc. While we also model our RMP problem using ADP, we cannot use the solutions from past work for the following reasons:

1. ***Non-trivial generation of feasible actions:*** In using ADP to solve the ToD problem, the action for a single empty vehicle is to match a single request. Computing the feasible set of requests for a vehicle is a straightforward and the best action for all vehicles together can then be computed by solving a Linear Program (LP). In the case of the RMP, multiple requests can be assigned to a single empty or partially filled vehicle. Generating the set of feasible actions, in this case, is complex and real-time solutions to this problem have been the key challenge in literature on myopic solutions to ride-pooling. In this paper, we use the approach proposed by (Alonso-Mora et al. 2017) to generate feasible actions for a single vehicle (Section 4) and then use an Integer Linear Program (ILP) to choose the best action (Table 1) over all vehicles.
2. ***Inability to use LP-duals to update the value function:*** Past work in ADP for ToD (Simao et al. 2009) uses the dual values of the matching LP to update the parameters of their value function approximation. However, choosing the best action in the RMP requires solving an Integer Linear Program (ILP) that has bad LP-relaxations. As a result, we cannot use duals to update our value function. Instead, we show the connection between ADP and Reinforcement Learning (RL), and use the more general Bellman update used in RL to update the value function (Section 4).
3. ***Curse of Dimensionality:*** Past work in ADP for transportation problems addresses the curse of dimensionality by considering the value function to be dependent on a small set of hand-crafted attributes (e.g., aggregated number of vehicles in each location) rather than on the states of a large number of vehicles. Hand-crafting of state attributes is domain-specific and is incredibly challenging for a complex problem like RMP, where aggregation of vehicles is not a feasible attribute (as each vehicle can have different number of passengers going to multiple different locations). Instead, we use a Neural Network based value function to automatically learn a compact low dimensional representation of the large state space.
4. ***Incorporating Neural Network value functions into the optimisation problem:*** Past work in ADP for ToD uses linear or piece-wise linear value function approximations that allow for the value function to be easily integrated into the matching LP. Non-linear value functions (such as neural networks) cannot be integrated in this way, however, as they would make the overall optimisation program non-linear. In Section 4), we address this issue by using a two-step decomposition of the value function that allows it to be efficiently integrated into the ILP as constants.
5. ***Challenges of learning a Neural Network value function:*** In Deep Reinforcement Learning literature (Mnih et al. 2015), it has been shown that naive approaches to approximating Neural Network value functions are unstable. Additionally, training them requires millions of samples. To address this, we propose a combination of methodological and practical solutions in Section 4.

The combination of using a Neural Network value function (instead of linear approximations) and updating it with a more general Bellman update (instead of LP-duals) represents a general alternative to past ADP approaches that we term Neural ADP (NeurADP).

Approximate Dynamic Programming Model for the RMP

We model the RMP by instantiating the tuple in Section 2.

S: The state of the system is represented as $s_t = (r_t, u_t)$ where r_t is the state of all vehicles and u_t is contains all the requests waiting to be served. A vehicle $r \in \mathcal{R}$ at decision epoch t is described by a vector $r_t^i = (p^i, t, L^i)$ which represents its current trajectory. Specifically, it captures the current location (p^i), time(t) and an ordered list of future locations (along with the cut-off time by which each must be visited) that the vehicle has to visit (L^i) to satisfy the currently assigned requests. Each user request j at decision epoch t is represented using vector $u_t^j = (o^j, e^j)$ which captures its origin and destination.

A: For each vehicle, the action is to assign a group of users from the set $\mathcal{U}_t$ to it. These actions should satisfy:

1. ***Constraints at the vehicle level*** - satisfying delay constraints $\mathcal{D}$ and vehicle capacity constraints
2. ***Constraints at the system level*** - Each request is assigned to at most one vehicle.

Handling exponential action space: To reduce the complexity, feasible actions are generated in two steps. In the first step, we handle vehicle-level constraints by generating a set of feasible actions (groups of users) for each vehicle. To do this efficiently, we first generate an RTV (Request, Trip, Vehicle) graph using the algorithm by Alonso et.al. (Alonso-Mora et al. 2017). Along with feasible actions, this generation process also provides the routes that the vehicle should take to complete each action. We use $\mathcal{F}_t^i$ to denote the set of feasible actions generated for vehicle i at decision epoch t.

$$\mathcal{F}_t^i = \{f^i \mid f^i \in \cup_{c'=1}^{c^i} [\mathcal{U}]^{c'}, PickUpDelay(f^i, i) \leq \tau, \\ DetourDelay(f^i, i) \leq \lambda\}$$

To ensure that system-level constraints are satisfied, we solve an ILP that considers all vehicles and requests together. Let $a_t^{i,f}$ denote that vehicle i takes action f at

AssignmentILP(t):

$$\max \quad \sum_{i} \sum_{f \in \mathcal{F}_t^i} o_t^{i,f} * a_t^{i,f} + V^i(T^{i,a}(r_t^{i,a}, f)) * a_t^{i,f} \quad (6)$$

subject to Constraints (3) - (5)

Table 1: Optimization Formulation for assignment of vehicles to feasible actions

decision epoch t. Then, the decision variables $a_t^{i,f}$ need to satisfy following constraints:

$$\sum_{f \in \mathcal{F}_t^i} a_t^{i,f} = 1 ::: \forall i \in \mathcal{R} \quad (3)$$

$$\sum_{i \in \mathcal{R}} \sum_{f \in \mathcal{F}_t^i; j \in f} a_t^{i,f} \leq 1 ::: \forall j \in \mathcal{U}_t \quad (4)$$

$$a_t^{i,f} \in \{0, 1\} ::: \forall i, f \quad (5)$$

Constraint (3) ensures that each vehicle is assigned a single action and constraint (4) ensures that each request is a part of, at most, one action. Together, they ensure that a request can be mapped to at most one vehicle.

We use A_t denote the set of all actions that satisfy both individual and system-level constraints at time t and $a_t \in A_t$ to denote a feasible action in this set[3].

ξ: As in previous work, exogenous information ξ_t represents the user demand that arrives between time $t-1$ and t.

T: The transition function T^a defines how the vehicle state changes after taking an action. In the case of the RMP, all user requests that are not assigned are lost (Alonso-Mora et al. 2017). Therefore, the user demand component of post-decision state will be empty, i.e., $u_t^a = \phi$.

$$T^a(s_t, a_t) = r_t^a \quad (7)$$

Here, r_t^a denotes the post decision state of the vehicles. We use $T^{i,a}(s_t^i, a_t^i) = r_t^{i,a}$ to denote the transition of individual vehicles. At each decision epoch, based on the actions taken, $(p^i, t, L^i)\, \forall i$ are updated and are captured in $r_t^{i,a}$. Each vehicle has a fixed path corresponding to each action and as a result the transition above is deterministic.

O: When vehicle i takes a feasible action f at decision epoch t, its contribution to the objective is $o_t^{i,f}$. For the objective of maximizing the number of requests served, $o_t^{i,f}$ is the number of requests that are part of a feasible action f (0 for the null action a_ϕ). The objective function at time t is as follows:

$$o_t(s_t, a_t) = \sum_{i \in \mathcal{R}} \sum_{f \in \mathcal{F}_t^i} o_t^{i,f} * a_t^{i,f}$$

[3]At every time step, we augment A_t with a *null* action a_ϕ. This allows a vehicle to continue along its trajectory without being assigned a passenger.

Algorithm 1: NeurADP (N, T)

```
Initialize: replay memory M, Neural value function
  V(with random weights θ)
for each episode 1 ≤ n < N do
    Initialize the state s_0^n by randomly positioning vehicles.
    Choose a sample path ξ^n
    for each step 0 ≤ t ≤ T do
        Compute the feasible action set F_t based on s_t^n.
        Solve the ILP in Table 1 to get best action a_t^n.
        (Add the Gaussian noise for exploration.)
        Store (r_t^n, F_t) as an experience in M.
        if t % updateFrequency == 0 then
            Sample a random mini-batch of experiences from M
            for each experience e do
                Solve the ILP in Table 1 with the information from experience e to get the objective value y^e
                for each vehicle i do
                    Perform a gradient descent step on (y^{e,i} − V(r_t^{i,n}))^2 with respect to the network parameters θ
        Update: s_t^{a,n} = T^a(s_t^n, a_t^n), s_{t+1}^n = T^ξ(s_t^{a,n}, ξ_{t+1}^n)
```

Value Function Decomposition

Non-linear value functions, unlike their linear counterparts, cannot be directly integrated into the matching ILP. One way to incorporate them is to evaluate the value function for all possible post-decision states and then add these values as constants. However, the number of post-decision states is exponential in the number of resources/vehicles.

To address this, we propose a two-step decomposition of our overall value function that converts it into a linear combination over individual value functions associated with each vehicle[4]:

- ***Decomposing joint value function based on individual vehicles' value functions:*** We use the fact that we have rewards associated with each vehicle to decompose the joint value function for all vehicles' rewards into a sum over the value function for each vehicle's rewards. The proof for this is straightforward and follows along the lines of (Russell and Zimdars 2003).

$$V(r_t^a) = \sum_i V^i(r_t^a)$$

- ***Approximation of individual vehicles' value functions:*** We make the assumption that the long-term reward of

[4]As mentioned in equation (7), the post-decision state only depends on the vehicle state. Therefore, $V(s_t^a) = V(r_t^a)$.

a given vehicle is not significantly affected by the specific actions another vehicle makes in the current decision epoch. This makes sense because the long-term reward of a given vehicle is affected by the interaction between its trajectory and that of the other vehicles and, at a macro level, these do not change significantly in a single epoch. This assumption allows us to use the pre-decision, rather than post-decision, state of other vehicles.

$$V^i(r_t^a) = V^i(\left\langle r_t^{i,a}, r_t^{-i,a} \right\rangle) \approx V^i(\left\langle r_t^{i,a}, r_t^{-i} \right\rangle)$$

Here, -i refers to all vehicles that are not vehicle i. This step is crucial because the second term in the equation above r_t^{-i} can now be seen as a constant that does not depend on the exponential post-decision state of all vehicles.

Therefore, the overall value function can be rewritten as:

$$V(r_t^a) = \sum_i V^i(\left\langle r_t^{i,a}, r_t^{-i} \right\rangle)$$

We evaluate these individual V^i values for all possible $r_t^{i,a}$ and then integrate the overall value function into the ILP in Table 1 as a linear function over these individual values. This reduces the number of evaluations of the non-linear value function from exponential to linear in the number of vehicles.

Value Function Estimation for NeurADP

To estimate the value function V over the post-decision state, we use the Bellman equation (decomposed in the ADP as equation (1) and (2)) to iteratively update the parameters of the function approximation. In past work (Simao et al. 2009), the parameters of a linear (or piece-wise linear) value function were updated in the direction of the gradient provided by the dual values at every step. Hence, the LP-duals removed the need to explicitly calculate the gradients in the case of a linear function approximation.

Given that we use a neural network function approximation and require an ILP (rather than an LP), we cannot use this approach. Instead, we use standard symbolic differentiation libraries (Abadi et al. 2015) to explicitly calculate the gradients associated with individual parameters. We then update these parameters by trying to minimise the L2 distance between a one-step estimate of the return (from the Bellman equation) and the current estimate of the value function (Mnih et al. 2015), as shown in Algorithm 1.

Overcoming challenges in Neural Network Value Function Estimation

In this section, we describe how we mitigate the stability and scalability challenges associated with learning neural network value functions through a combination of methodological and practical methods.

Improving stability of Bellman updates: It has been shown in Deep Reinforcement Learning (DRL) literature that using standard on-policy methods to update Neural Network (NN) based value function approximations can lead to instability (Mnih et al. 2015). This is because the NN expects the input samples to be independently distributed while consecutive states in RL and ADP are highly correlated. To address these challenges, we propose using off-policy updates. To do this, we save the current state and feasible action set for each vehicle $\forall_i (s_t^i, \mathcal{F}_t^i)$ during sample collection. Then, offline, we score the feasible actions using the value function and use the ILP create the best matching. Finally, we update the value function of the saved post-decision state with that of the generated next post-decision state. This is different from experience replay in standard Q-Learning because the state and transition functions are partly known to us and choosing the best action, in our case, involves solving an ILP. In addition to off-policy updates, we use standard approaches in DRL like using a target network and Double Q-Learning (Van Hasselt, Guez, and Silver 2016).

Addressing the data scarcity: Neural Networks typically require millions of data points to be effective, even on simple arcade games (Mnih et al. 2015). In our approach, we address this challenge in 3 ways:

- Practically, we see that in the RMP, the biggest bottleneck in speed is in generating feasible actions. To address this, as noted above, we directly store the set of feasible actions instead of recomputing them for each update.
- Secondly, we use the same Neural Network for the value function associated with each of the individual vehicles. This means that a single experience leads to multiple updates, one for each vehicle.
- Finally, we use Prioritised Experience Replay (Schaul et al. 2015) to reuse existing experiences more effectively.

Practical simplifications: Finally, based on our domain knowledge, we introduce a set of practical simplifications that makes learning tractable:

- Instead of using one-hot representations for discrete locations, we create a low-dimensional embedding for each location by solving the proxy-problem of trying to estimate the travel times between these locations.
- During training, we perform exploration by adding Gaussian noise to the predicted V_i values (Plappert et al. 2017). This allows us to more delicately control the amount of randomness introduced into the training process than the standard ϵ-greedy strategy.
- We don't use the pre-decision state of all the other vehicles to calculate the value function for a given vehicle (as suggested in Section 4). Instead, we aggregate this information into the count of the number of nearby vehicles and provide this to the network, instead.

The specifics of the neural network architecture and training can be found in the supplementary file.

5 Experiments

The goal of the experiments is to compare the performance of our NeurADP approach to leading approaches for solving the RMP on a real-world dataset(NYYellowTaxi 2016) across different RMP parameter values. The metric we use to compare them is the service rate, i.e., the percentage of

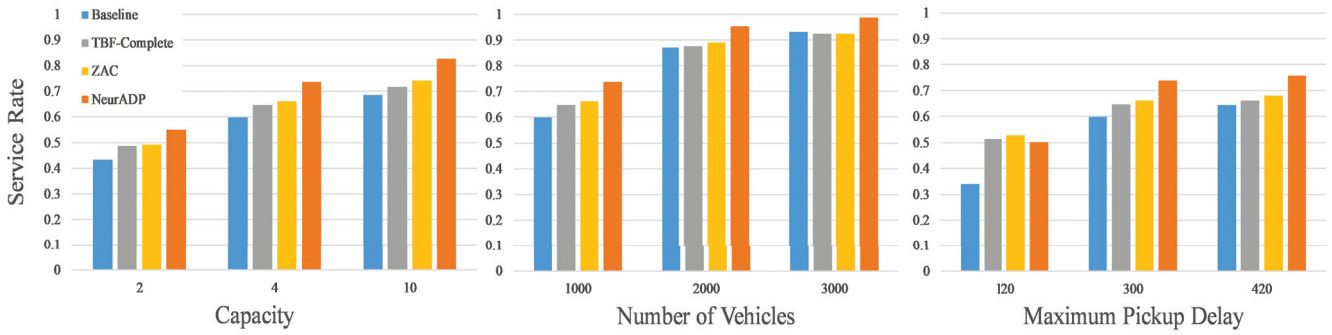

Figure 2: The three graphs benchmark our performance across 3 sets of parameter values - c^i, τ and $|\mathcal{R}|$ respectively (from left to right). In each case, we start with the prototypical configuration of τ = 300 seconds, c^i = 4 and $|\mathcal{R}|$ = 1000 and vary the chosen parameter.

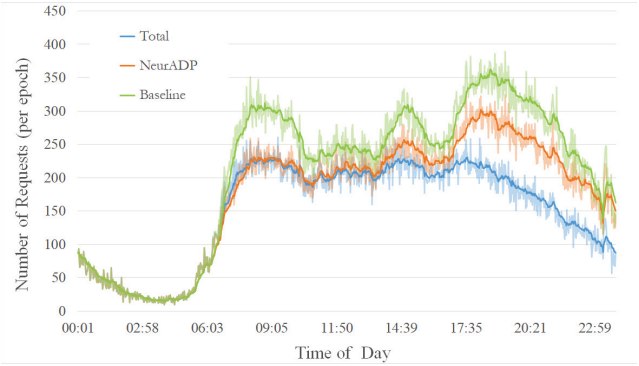

Figure 3: The graph compares the number of requests served as a function of time. The bold lines represent a moving average of the actual values (represented by the lighter lines). This graph corresponds to the configuration λ = 300sec, c^i = 10 and $|\mathcal{R}|$ = 1000 on 4 April 2016

total requests served. Similar to (Alonso-Mora et al. 2017; Lowalekar, Varakantham, and Jaillet 2019), we vary the following parameters: the maximum allowed waiting time τ from 120 seconds to 420 seconds, the number of vehicles $|\mathcal{R}|$ from 1000 to 3000 and the capacity c^i from 2 to 10. The value of maximum allowable detour delay λ is taken as $2*\tau$. The decision epoch duration Δ is taken as 60 seconds.

We compare NeurADP against the following algorithms:

- **ZAC** – ZAC algorithm by (Lowalekar, Varakantham, and Jaillet 2019).
- **TBF-Complete** – Implementation of (Alonso-Mora et al. 2017) taken from (Lowalekar, Varakantham, and Jaillet 2019).
- **TBF-Heuristic (Baseline)** – This is our implementation of Alonso et.al.'s (Alonso-Mora et al. 2017) approach [5].

To disentangle the source of improvement in our approach, we introduce TBF-Heuristic which we refer to as the baseline. This uses a fast insertion method, that mirrors the implementation in NeurADP, to generate feasible actions. This is important because training requires a lot of samples and the key bottleneck in generating samples is the generation of feasible actions. While this process is completely parallelisable in theory, our limited academic computing resources do not allow us to leverage this. Therefore, comparing against this baseline allows us to measure the impact using future information has on solution quality.

Setup: The experiments are conducted by taking the demand distribution from the publicly available New York Yellow Taxi Dataset (NYYellowTaxi 2016). The experimental setup is similar to the setup used by (Alonso-Mora et al. 2017; Lowalekar, Varakantham, and Jaillet 2019). Street intersections are used as the set of locations $\mathcal{L}$. They are identified by taking the street network of the city from openstreetmap using osmnx with 'drive' network type (Boeing 2017). Nodes that do not have outgoing edges are removed, i.e., we take the largest strongly connected component of the network. The resulting network has 4373 locations (street intersections) and 9540 edges. Similar to earlier work (Alonso-Mora et al. 2017), we only consider the street network of Manhattan as a majority ($\sim$75%) of requests have both pickup and drop-off locations within it.

The real-world dataset contains data about past customer requests for taxis at different times of the day and different days of the week. From this dataset, we take the following fields: (1) Pickup and drop-off locations (latitude and longitude coordinates) - These locations are mapped to the nearest street intersection. (2) Pickup time - This time is converted to appropriate decision epoch based on the value of Δ. The travel time on each road segment of the street network is taken as the daily mean travel time estimate computed using the method proposed in (Santi et al. 2014). The dataset contains on an average 322714 requests in a day (on weekdays) and 19820 requests during peak hour.

We evaluate the approaches over 24 hours on different days starting at midnight and take the average value over 5 weekdays (4 - 8 April 2016) by running them with a single instance of initial random location of taxis [6]. NeurADP is

[5] Please refer to supplementary file for a complete list of differences in the implementation.

[6] All experiments are run on 24 core - 2.4GHz Intel Xeon E5-2650 processor and 256GB RAM. The algorithms are im-

trained using the data for 8 weekdays (23 March - 1 April 2016) and it is validated on 22 March 2016. For the experimental analysis, we consider that all vehicles have identical capacities.

Results: We now compare the results of our approach, NeurADP, against past approaches. Figure 2 shows the comparison of service rate between NeurADP and existing approaches. As shown in the figure, NeurADP consistently beats all existing approaches across different parameters. Here are the key observations:

- **Effect of changing the tolerance to delay, τ:** NeurADP obtains a 16.07% improvement over the baseline approach for $\tau = 120$ seconds. The difference between the baseline and NeurADP decreases as τ increases. The lower value of τ makes it difficult for vehicles to accept new requests while satisfying the constraints for already accepted requests. Therefore, it is more important to consider future requests while making current assignments when τ is lower, leading to a larger improvement.
- **Effect of changing the capacity, c^i:** NeurADP obtains a 14.03% gain over baseline for capacity 10. The difference between the baseline and NeurADP increases as the capacity increases. This is because, for higher capacity vehicles, there is a larger scope for improvement if the future impact of making an assignment is taken into account.
- **Effect of changing the number of vehicles, $|\mathcal{R}|$:** The difference between the baseline and NeurADP decreases as the number of vehicles increase. This is because, in the presence of a large number of vehicles, there will always be a vehicle that can serve the request. As a result, the quality of assignments plays a smaller role.

For the specific case of $\tau = 120$, NeurADP does not outperform ZAC and TBF-Complete because they use a more complex feasible action generation which allows them to leverage complex combinations of requests and their ordering. This becomes important as the delay constraints become stricter. If NeurADP is implemented with the complete search for feasible action generation, we expect it to outperform ZAC and TBF in this case as well.

We further analyse the improvements obtained by NeurADP over baseline by comparing the number of requests served by both approaches at each decision epoch. Figure 3 shows the total number of requests available and the number of requests served by the baseline and our approach NeurADP at different decision epochs. As shown in the figure, initially at night time when the demand is low both approaches serve all available demand. During the transition period from low demand to high demand period, the baseline algorithm starts to greedily serve the available requests without considering future requests. On the other hand, NeurADP ignores some requests during this time to serve more requests in future. This allows NeurADP to serve more requests during peak time.

plemented in python and optimisation models are solved using CPLEX 12.8. The setup, code and supplementary file are available at https://github.com/sanketkshah/NeurADP-for-Ride-Pooling.

The approach can be executed in real-time settings. The average time taken to compute each batch assignment using NeurADP is less than 60 seconds (for all cases) [7].

These results indicate that using our approach can help ride-pooling platforms to better meet customer demand.

6 Conclusion

On-demand ride-pooling has become quite popular in transportation (through services like UberPool, LyftLine, etc.), food delivery (through services like FoodPanda, Deliveroo, etc.) and in logistics. This is a challenging problem as we have to assign each (empty or partially filled) vehicle to a group of requests. Due to the difficulty of making such assignments online, most existing work has focussed on myopic assignments that do not consider the future impact of assignments. Through a novel combination of approaches from ADP, ride-sharing and Deep Reinforcement Learning, we provide an offline-online approach that trains offline on past data and provides online assignments in real-time. Our approach, NeurADP improves the state of art by up to 16% on a real dataset. To put this result in perspective, typically, an improvement of 1% is considered a significant improvement on ToD for an entire city (Xu et al. 2018; Lowalekar, Varakantham, and Jaillet 2019).

7 Acknowledgements

This research was supported by the Singapore National Research Foundation through the Singapore-MIT Alliance for Research and Technology (SMART) Centre for Future Urban Mobility (FM).

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Alonso-Mora, J.; Samaranayake, S.; Wallar, A.; Frazzoli, E.; and Rus, D. 2017. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences* 201611675.

Bei, X., and Zhang, S. 2018. Algorithms for trip-vehicle assignment in ride-sharing.

Boeing, G. 2017. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems* 65:126–139.

[7] 60 seconds is the decision epoch duration considered in the experiments

Gessner, K. 2019. Uber vs. lyft: Who's tops in the battle of u.s. rideshare companies. https://www.uber.com/en-GB/newsroom/company-info/.

Heath, A. 2016. Inside uber's quest to get more people in fewer cars. https://www.businessinsider.com/uberpool-ridesharing-could-be-the-future-of-uber-2016-6/.

Huang, Y.; Bastani, F.; Jin, R.; and Wang, X. S. 2014. Large scale real-time ridesharing with service guarantee on road networks. *Proceedings of the VLDB Endowment* 7(14):2017–2028.

Li, M.; Jiao, Y.; Yang, Y.; Gong, Z.; Wang, J.; Wang, C.; Wu, G.; Ye, J.; et al. 2019. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. *arXiv preprint arXiv:1901.11454*.

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

Lowalekar, M.; Varakantham, P.; and Jaillet, P. 2019. ZAC: A zone path construction approach for effective real-time ridesharing. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019.*, 528–538.

Ma, S.; Zheng, Y.; and Wolfson, O. 2013. T-share: A large-scale dynamic taxi ridesharing service. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, 410–421. IEEE.

Maxwell, M. S.; Restrepo, M.; Henderson, S. G.; and Topaloglu, H. 2010. Approximate dynamic programming for ambulance redeployment. *INFORMS Journal on Computing* 22(2):266–281.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

NYYellowTaxi. 2016. New york yellow taxi dataset. http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.

Parragh, S. N.; Doerner, K. F.; and Hartl, R. F. 2008. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft* 58(1):21–51.

Plappert, M.; Houthooft, R.; Dhariwal, P.; Sidor, S.; Chen, R. Y.; Chen, X.; Asfour, T.; Abbeel, P.; and Andrychowicz, M. 2017. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*.

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

Ritzinger, U.; Puchinger, J.; and Hartl, R. F. 2016. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research* 54(1):215–231.

Ropke, S., and Cordeau, J.-F. 2009. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science* 43(3):267–286.

Russell, S. J., and Zimdars, A. 2003. Q-decomposition for reinforcement learning agents. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 656–663.

Santi, P.; Resta, G.; Szell, M.; Sobolevsky, S.; Strogatz, S. H.; and Ratti, C. 2014. Quantifying the benefits of vehicle pooling with shareability networks. *Proceedings of the National Academy of Sciences* 111(37):13290–13294.

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

Simao, H. P.; Day, J.; George, A. P.; Gifford, T.; Nienow, J.; and Powell, W. B. 2009. An approximate dynamic programming algorithm for large-scale fleet management: A case application. *Transportation Science* 43(2):178–197.

Tong, Y.; Zeng, Y.; Zhou, Z.; Chen, L.; Ye, J.; and Xu, K. 2018. A unified approach to route planning for shared mobility. *Proceedings of the VLDB Endowment* 11(11):1633–1646.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*.

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

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

Xu, Z.; Li, Z.; Guan, Q.; Zhang, D.; Li, Q.; Nan, J.; Liu, C.; Bian, W.; and Ye, J. 2018. 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*, 905–913. ACM.

Yu, X., and Shen, S. 2019. An integrated decomposition and approximate dynamic programming approach for on-demand ride pooling. *IEEE Transactions on Intelligent Transportation Systems*.