A Distributed Model-Free Ride-Sharing Algorithm with Pricing using Deep Reinforcement Learning

Marina Haliem
mwadea@purdue.edu
Purdue University

Ganapathy Mani
manig@purdue.edu
Purdue University

Vaneet Aggarwal
vaneet@purdue.edu
Purdue University

Bharat Bhargava
bbshail@purdue.edu
Purdue University

ABSTRACT

Modern-day ride-sharing platforms leave out drivers and customers in the decision-making process of the rides in terms of vehicle-customer matching as well as pricing. We propose a model-free Distributed Pricing-based Ride-sharing with pooling (DPRS) framework with reinforcement utility functions for both customers and drivers. The framework allows (1) drivers to choose their convenient ride based on the expected reward for this ride as well as the destination locations for future rides influenced by the supply-demand computed by the Deep Q-network, (2) customers to accept or reject rides based on their preferred pricing window, timing preferences, type of the vehicle, and convenient number of people to car pool with, (3) customer to be added to the ride queue if she/he rejects the price initiated by the driver, and (4) Influencing vehicle-passenger matching and dispatching based on prices through reinforcement learning (RL). Through our simulation of multi-agent ride-sharing with pooling platform, we show that performance of the platform significantly improved in terms of accept rate, profits of both the customers and drivers, and reduction of travel distance as well as idle time in between rides for drivers with similar profits, when compared to the state of the art ride-sharing settings that don't consider pricing strategies or potential hotspot locations.

CCS CONCEPTS

• **Computer systems organization** → **Real-time systems**; • **Computing methodologies** → **Multi-agent planning**.

KEYWORDS

Deep Reinforcement Learning, Neural Networks, Car Pooling, Mobility on Demand, Multi-agent, Intelligent Transporartion

ACM Reference Format:
Marina Haliem, Ganapathy Mani, Vaneet Aggarwal, and Bharat Bhargava. 2020. A Distributed Model-Free Ride-Sharing Algorithm with Pricing using Deep Reinforcement Learning. In *Computer Science in Cars Symposium (CSCS '20), December 2, 2020, Feldkirchen, Germany.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3385958.3430484

1 INTRODUCTION

Advanced user-centric ride-hailing services such as Uber and Lyft are thriving in urban environments by transforming urban mobility through convenience in travel to anywhere, by anyone, and at anytime. These ride-hailing services can have positive impact on traffic congestion, personal mobility, environmental pollution, and energy consumption, thereby living conditions in urban environments [Schrank et al. 2019], [Hennessy and Wiesenthal 2019]. Even though the pooling services provide customized personal service to customers, both the drivers and the customers are largely left out in deciding what is best for them in terms of their conveniences and preferences. It is challenging to introduce customer and driver conveniences into the framework. For example, a customer may have limitation on the money she/he could spend for a particular ride as well as time constraints on reaching the destination. On the other hand, driver may not be willing to accept the customer's convenient fare as it may negatively affect his/her profits since the final destination may be in low demand area. Thus, a robust framework is needed to identify trade-offs between the drivers' and the customers' needs, and make a compromised decision that is favorable to both.

It is a non-trivial optimization problem to include a pricing strategy in ride-sharing (with pooling) where customers and drivers can weigh in on the decision (customers can take or leave a ride based on their convenience, and drivers can propose more profitable price) since both may have significantly different preferences. These unpredictable, and sometimes erratic, preferences will end up costing drivers more because the ride cancelling rate may go up. This increases the idle time—empty or semi-filed vehicles searching for a ride—of vehicles. Increase in ride cancellation will end up overloading the system with customers waiting for too long to get an ideal ride. It will also increase the travel distances for each vehicle with reduced number of customers carpooling per vehicle. Thus adding pricing strategy has the potential to burden the system and increase losses for customers and drivers in terms of time and money. This prospect raises interesting research questions: (1) is it possible to have a distributed pricing framework where drivers and customers can make decisions individually and attain a compromise that provides profits to both parties?, (2) given convenience constraints of customers and drivers, is it possible to reduce the rejection rate of rides through customer utility function and reduce the customer wait time?, and (3) given driver's profit margin and customer's price threshold (i.e, maximum he/she is willing to pay for the ride), is it possible to satisfy both customer as well as the driver and increase the average earnings of the fleet?

This paper utilizes the dispatch of idle vehicles using a Deep Q-learning (DQN) framework as in [Alabbasi et al. 2019], and we add the profit term in the reward function so that the output expected discounted rewards (Q-values) associated with each action, becomes a good reflection of the expected earnings gained from perfroming this action. Depending on the DQN, we decide on the pricing by vehicles as well as the acceptance or rejection of rides by customers. The goal of our approach is to utilize optimal dispatching provided by the pre-defined model, influence the customer and vehicle utility functions to achieve convenience, maximize the profits for both and reduce the customers' waiting time, travel time, and idle driving. We identify the following as our major contributions:

- We integrate our novel distributed pricing approach in the ride-sharing (with pooling) framework where, based on their convenience, customers and drivers get to weigh-in on the decision-making of a particular ride. Also, the ride-sharing decisions impact pricing, and vice versa. This approach is built on top of a distributed model-free approach for matching and dispatching vehicles in large-scale systems, DeepPool [Alabbasi et al. 2019].
- In this framework, drivers are allowed to propose a price based on the location of the ride (source and destination) that account for the reward of DQN based on the destination location. Similarly, customers can either accept or reject rides based on their pricing threshold, timing preference, type of vehicle, and number of people to share a ride with.
- Our framework increases the profit margins of both customers and drivers using the reinforcement learning utility functions that are influenced by Q-values learnt using DQN for making the vehicles' dispatch decisions. The optimization problem is formulated such that our novelty framework tries to minimize the rejection rate, customers' waiting time, vehicles' idle time, total number of vehicles to reduce traffic congestion and fuel consumption.
- Through experiments using real-word dataset of New York City's taxi trip records [NYC.gov 2019] (15 million trips), we simulate the ride-sharing system with distributed pricing strategy. We show that our novel DPRS framework increases profits for customers and drivers when compared to various settings (ride-sharing, no ride-sharing, dummy agent without dispatching, DQN agent with dispatching) for similar rejection rate and travel distance. This is challenging as the overall pricing decisions and the acceptance rate greatly impact the ride-sharing decisions and vice versa, as the two are not independent.

The code for this project is available at [1]. The rest of the paper is organized as follows: Section 2 provides literature review related to our paper. Section 3 explains the architecture and the main components of our DPRS framework, as well as the strategy for distributed pricing. In Section 4, we give a descritopn of how the DQN dispatching algorithm works. Section 5 provides the experimental results. Finally, Section 6 concludes this paper.

[1]https://github.itap.purdue.edu/Clan-labs/RS_Pricing

2 RELATED WORK

Ride-sharing is a widely studied problem in the Artificial Intelligence community. But majority of those approaches are model-based approaches [Kleiner et al. 2011], [Zhang and Pavone 2016], [Ma et al. 2014], where pickup locations, destinations, and travel time are all predetermined before coming up with a dispatching policy. They anticipate that the computed dispatching policy would improve the performance of the system. These models cannot be deployed in highly dynamic environments and they do not easily adapt when the size of the states increase. In some ride-sharing solutions, for example [Bei and Zhang 2018] incentives play a central role in computing dispatching policies. Ride-sharing has been studied as pick up and delivery problem (PDP) where the number of miles travelled by vehicles is reduced, by carpooling customers [Lu 2015]. Requests for rides are generated by square block with uniform distribution. Graph theoretical models have also been used in implementing ride-sharing. In [Ta et al. 2017], drivers and customer ride requests are assumed as nodes in a bigraph where vehicles and requests are managed by computing the max(weighted matching). Equipped with real-world data, the authors in [Jauhri et al. 2017] have designed customer requests and variation in destination location, and travel time using a graph. In [de Lira et al. 2015], the matching algorithm uses user utility function that is leveraged for customer-vehicle matching. In [Zhang et al. 2016], the disadvantages and advantages of taxi ride-pooling mode are studied by simulation. Experimental results show that the ride-pooling can decrease customer's cost significantly at the same time increasing the profit for the vehicle driver. In addition, passenger capacity is improved considerably. Similarly, several studies show that learning from historical taxi fleet data helps organizing the whole fleet while reducing the customer wait-time as well as drivers' wait-time [Zhao et al. 2016]. In [Oda and Joe-Wong 2018], DQN methodology is used for adaptive dynamic fleet management and proves that distributed framework is more scalable than the centralized framework. However, this approach does not consider ride-splitting and is only used for dispatching vehicles per request at a time. The authors, in [Alabbasi et al. 2019], provided the first model-free approach for ride-sharing with pooling based on RL. However, DeepPool neither incorporates a pricing strategy, nor accommodates customers' and drivers' convenience. It primarily focuses on dispatching and customer-vehicle matching using a greedy matching policy. To the best of our knowledge, this is the first work that introduces a model-free approach for a distributed pricing-based ride-sharing where customers and drivers can weigh in their ride preferences, influencing the decision making of ride-sharing platforms.

In this paper, we approach the pricing-based ride-sharing (with pooling) problem through a model-free technique for ride-sharing with ride-pooling. In contrast to model-based approaches in literature [Zhang and Pavone 2016] [Ma et al. 2014] [Kleiner et al. 2011] [Bei and Zhang 2018], our proposed approach can adapt to dynamic distributions of customer and driver preferences. The authors, in [Alabbasi et al. 2019], provided the first model-free approach for ride-sharing with pooling based on RL. However, DeepPool neither incorporates a pricing strategy, nor accommodates customers' and drivers' conveniences. It primarily focuses on dispatching idle vehicles and customer-vehicle matching using a greedy matching policy.

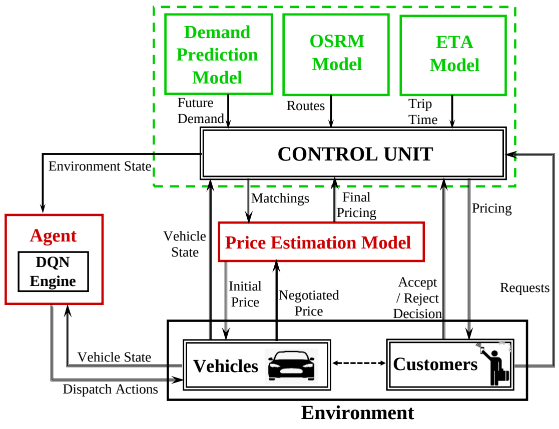

Figure 1: DPRS Architecture

To the best of our knowledge, this is the first work that introduces a model-free approach for a distributed pricing-based ride-sharing where customers and drivers can weigh in their ride preferences, influencing the decision making of ride-sharing platforms.

3 DISTRIBUTED PRICING-BASED RIDE-SHARING (DPRS)

We propose a novel Distributed Pricing for Ride-sharing (DPRS) framework using Deep Q-Network (DQN), where customers and drivers (will be referred to as Agents henceforth) are involved in the decision-making process. They learn the best pricing actions based on their utility functions that dynamically change based on each agent's set of preferences and environmental variables. Moreover, vehicles learn the best future dispatch action to take at time step t, taking into consideration the locations of all other nearby vehicles, but without anticipating their future decisions. Note that, vehicles get dispatched to areas of anticipated high-demand either when just enter the market, or when they spend a long time being idle (searching for a ride). Vehicles' dispatch decisions are made in parallel. To guarantee distribution, updates are performed sequentially so that vehicles can take other vehicles' actions into account when making their own decisions. However, each vehicle does not influence the future. Therefore, our algorithm learns the optimal policy for each agent independently as opposed to centralized-based approaches such as the setting in [Oda and Joe-Wong 2018]. actions of other vehicles, thus limiting the coordination among them.

Figure 1 shows the basic components of DPRS and the interactions between them. We assume that the control unit is responsible for: (1) making the initial matching decisions, based on the proximity of vehicles to ride requests, (2) maintaining the states such as current locations, current capacity, destinations, etc., for all vehicles. These states are updated in every time step based on the dispatching and matching decisions. (3) Control unit also has some internal components that help manage the ride-sharing environment such as: the estimated time of arrival (ETA) model used to calculate and contentiously update the estimated arrival time. The OSRM model used to generate the vehicle's optimal trajectory to reach a destination, and demand prediction models used to calculate the future anticipated demand in all zones. First, the ride requests are input to the system along with the heat map for supply and demand. Then, the control unit performs greedy vehicle-passenger matching where a request is assigned to the nearest vehicle, calculates the corresponding initial pricing and notifies the driver. Vehicles adopt a dispatching policy using DQN, where they get dispatched to zones with anticipated high demand when they experience large idle duration or when they newly enter the market. Using the expected discounted reward learnt from DQN and the ride's destination, vehicles weigh their utility based on the potential hotspot locations, and propose a new pricing for the customer. A Customer has the right to accept or reject based on his/her own independent utility function. A Vehicle communicates with the control unit, as needed, to request new information of other vehicles (prior to making a dispatch or price decision) or update its own status (after any decision).

For the DQN dispatch, we use the framework as in [Alabbasi et al. 2019], and add the profit in the reward function. The details are provided Section 4 and further explained in Appendix A. In thi section, we focus on the pricing algorithm, and involving both the drivers and the customers in the decision-making process.

3.1 Initial Pricing

In our algorithm, we consider various vehicle types: hatch-back, sedan, luxury, and van. Each of which has different capacity, mileage, and base price for driver per trip denoted by B_j. B_j serves as the local minimum earning that the driver gains per trip. Also, based on the vehicle type, the pricing per mile distance varies as well as the pricing per waiting minute. Initially, the system suggests a price for each vehicle-customer matching, taking into consideration several factors:

- The total trip distance, i.e., the distance till pickup plus the distance from pickup to drop off. Note that, this distance is composed of k routes/edges that constitute the vehicle's optimal path to the destination, denoted: $D_i = \sum_{i=1}^{k} r_i$.
- Number of customers who share travelling a trip distance (whether all or part of it, which can be determined from the vehicle's path). For simplicity, we denote it by vehicle j's capacity V_C^j at time t.
- The cost for fuel consumption associated with this trip, denoted by $D_i * (P_G/M_j)$, where P_G represents the average gas price, and M_V^j denotes the mileage for vehicle j assigned to trip i.
- The waiting time experienced by the customer (or customers) associated with trip i till pickup, denoted T_i.

The overall price initialization equation is represented as:

$$B_j + \left[\omega^1 * \frac{D_i}{V_C^j}\right] + \left[\omega^2 * \left(\frac{D_i}{V_C^j} * (P_G/M_V^j)\right)\right] - \left[\omega^3 * T_i\right] \quad (1)$$

where ω^1, ω^2, and ω^3 are the weights associated with each of the factors affecting the price calculation. ω^1 is the price per mile distance according to the vehicle type. ω^2 is set to 1 as it doesn't change across vehicles, what changes is the mileage in this factor. Finally, ω^3 is the price per waiting minute that is influenced by the vehicle type, it is negative here as we want to minimize the waiting time for the customer. Note that, our framework runs in two modes. A customer can select either to carpool or not. If the customer decides not to carpool, the vehicle will not pickup any extra customers until this customer drops off, and thus V_C^j will always be 1. However, for ride-sharing customers: the path for trip i, D_i is composed of k routes: $D_i = \sum_{i=1}^{k} r_i$. So, for each customer, D_i will be the sum of distances of the shared routes among other customers, with V_C^j set to the number of customers to ride-share with. Then, the price for the remaining non-shared distance, if any, will be calculated with D_i equals to the sum of distances of all un-shared routes and $V_C^j = 1$. The sum of both will constitute the total initial pricing. This initial pricing gets updated for on-board passengers whenever a vehicle picks up an additional customer (as the V_C^j will now increase), where all the overlapping (shared) distances are taken into consideration (when constructing D_i) and thus price may get reduced. Our proposed algorithm will first use the initial price and notify the driver, who will then modify the pricing based on the Q-values of the driver's dispatch-to location.

3.2 Vehicles' Proposed Pricing

Each driver follows a dispatch policy once he/she enters the market. This dispatch policy provides him/her with the best next dispatch action to make, which is predicted after weighing the expected discounted rewards (Q-values) associated with each possible move on the map using DQN (described in Appendix A.3 and A.4). As a result of running such a policy every dispatch cycle (which is set to 5 minutes), the driver gains insight about how the supply-demand is distributed over the city, and thus can make informed decisions on which destinations can yield him a higher profit. The driver's decision-making process is formulated as follows:

- Based on the knowledge of the expected discounted reward —Q-values —associated with the action of going to each location on the map (i.e. using DQN to learn the Q-values associated with each action), the driver can order the destinations descendingly and assign each of them a rank α, representing where it falls in that ordered list. The driver dynamically maintains this ranking and continuously updates it whenever the dispatch policy runs. For flexibility, the driver also has a tolerance rate, λ (which is a percentage), that he/she uses to decide on the size of the *desired zones* list.
- Upon receiving the initial price, and the request pickup location, the driver retrieves the ranking α associated with various destinations. After that, he adds the highest ranking λ locations to a set of *desired zones*, denoted L. Then, if the request's location is among that desired set L, the driver uses the initial pricing suggested for this trip, denoted $P_{init}(R_i)$.
- Otherwise, it would indicate that he/she might end up in the middle of nowhere (region with low demand), and thus receives no more requests or at least drives idle a long distance. Instead of just rejecting the request, he/she can suggest a price to the customer that is slightly higher by a factor influenced by both the rank of the destination as well as his/her own base price per trip B_j.

Let $P(R_i)$ represents the final price suggested by driver j for the customer associated with request/trip i, the driver's price decision is as follows:

$$P(R_i) = \begin{cases} P_{init}(R_i) & \text{if } loc(R_i) \in L \\ P_{init}(R_i) + [P_{init}(R_i) * \frac{\alpha_{loc(R_i)}}{2} * B_j] & \text{otherwise} \end{cases} \tag{2}$$

3.3 Customers' Decision Function

After the driver makes a decision regarding the price associated with the trip, it becomes the customer's turn to make his own decision according to his/her set of preferences. In our algorithm, we consider various preferences for each customer:

- Whether the customer is in a hurry or not or how much delay can he/she tolerate. This is taken into consideration in the customer's utility and denoted by delay / waiting time of trip i: T_i.
- Whether the customer prefers car-pooling or would rather take the ride alone even if it means a higher price. This is captured in the utility equation based on the current capacity of vehicle j assigned to trip i, denoted by V_C^j.
- Whether he/she prefer a certain type of vehicle for their trip, and whether he/she is willing to par more in exchange for a more luxurious vehicle. The type of vehicle j assigned to trip i is denoted by V_T^j.

Based on all these factors, the customer's utility for request/trip i is formulated as:

$$U_i = \left[\omega^4 * \frac{1}{V_C^j}\right] + \left[\omega^5 * \frac{1}{T_i}\right] + \left[\omega^6 * V_T^j\right] \tag{3}$$

where ω^4, ω^5, and ω^6 are the weights associated with each of the factors affecting the customer's overall utility. To add more flexibility, we introduce a customer's compromise threshold δ_i to represent how much the customer i is willing to compromise in the decision-making process. Finally, the decision of customer i to accept or reject, denoted by C_d^i, after receiving the final price $P(R_i)$ for the trip i is as follows:

$$C_d^i = \begin{cases} 1 & \text{if } U_i > P(R_i) - \delta_i \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Upon customer's acceptance, no further action is required, and the process continues normally. However, upon rejection a new matching process is initiated to match this request to another vehicle that better meets their preferences.

3.4 DPRS Algorithm

In this section, we explain our proposed pricing-based ride-sharing (DPRS) algorithm in detail. Recall the ride-sharing framework architecture shown in Figure 1. The full algorithm is shown in Algorithm 1, and can be summarized as follows:

(1) First, for each newly entered vehicle to the system, the agent (i.e., vehicle) determines its action a_t, and gains insight about the supply-demand situation over the city using its Q-Network agent [see lines 6 - 14]. Every vehicle V_j selects the action that maximizes its own reward, i.e., taking the *argmax* of the DQN-network output. The output of the DQN are the Q-values associated with every possible move that the vehicle can make at time step t, the input is the environment state vector s_t.
(2) Second, using our algorithm, the vehicles get matched to one or more requests, and initial pricing is calculated depending on whether it is on ride-sharing mode or not. Then, the proposed pricing strategy is used by vehicles takes place, which would be accepted by customers if it is bounded by their utility [see lines 16 - 39]. In our algorithm, we adopt a greedy matching

policy, where requests initially get matched to the nearest available vehicle. As rejections from customers are received, our algorithm tries to accommodate their preferences, instead of losing customers once they reject the initial assigned vehicle. We avoid further rejections that would result in further delay for the customer by running the matching policy again but feeding it with the learnt customer preferences. So, the customer does not get directly matched to the second nearest vehicle, as that vehicle might still not accommodate the customer's needs, and thus gets rejected as well. After acceptance from both parties, vehicle traverses to the pickup locations of passengers using the shortest optimal path in the road network graph, and eventually to drop them off at their destinations. In the ride-sharing mode, upon picking up additional customers, the price for other on-board passengers gets reduced, taking into consideration all overlapping distances [see lines 26 - 29].

(3) Finally, dispatching happens again for vehicles whose idle time T_I exceeds the maximum dispatch cycle (set to 10 minutes), as our model tries to maximize utilization, it dispatches those idle vehicles to zones of anticipated high demand [see lines 42 - 45]. Finally, each vehicle updates its status at the DQN based on the chosen actions (lines 39, 41, and 46).

4 DISTRIBUTED DQN DISPATCHING APPROACH

We utillize this framework in order to re-balance vehicles over the city to better serve the demand. At the beginning of every time step t, vehicles that newly enter the system at time t, are dispatched to areas of anticipated high demand follwoing this approach. Moreover, at the end of every time step, we check for vehicles whose idle duration exceeds 10 minutes, and we apply this technique to dispatch them to high-demand areas to better utilize our respurces. The overall flow of this framework is explained in Algorithm 2.

4.1 DQN Design and Model Parameters:

We build a ride-sharing simulator to train and evaluate our framework. We simulate New York City as our area of operation and the area is divided into multiple non-overlapping zones (or regions), each of which is 1 square mile. This allows us to discretize the area of operation and thus makes the action space—where to dispatch the vehicles—tractable. We use $m \in \{1, 2, 3, ..., M\}$ to denote the city's zones. We optimize our algorithm over T time steps, each of duration Δt. Here, we present the model parameters and notations:

(1) *Demand:* We denote the number of requests for zone m at time t as $d_{t,m}$. $d_{t,\tilde{t},m}$ is the number of vehicles that are currently unavailable at time t but will become available at time $\tilde{t}$ as they will drop-off customer(s) at region m. $d_{t,\tilde{t},m}$ can be estimated using the estimated time of arrival (ETA) model in [Alabbasi et al. 2019]. We denote the predicted future demand from time t_0 to time $t + T$ at each zone as $D_{t:T} = \{\bar{d}_t, ..., \bar{d}_{t+T}\}$. Such information can be reached by maintaining the state vectors described next in 2.

(2) *State Vector:* The state variables are utilized to reflect the environment status. We use $X_t = \{x_{t,1}, x_{t,2}, \ldots, x_{t,N}\}$ to denote the N vehicles' status at time t. $x_{t,n}$ is a vector that represents vehicle n's state variables at time step t such as: its current location V_{loc}, its current capacity V_C, its type V_T, its maximum capacity C^V_{max}, the time at which a passenger was picked

Algorithm 1 DPRS Algorithm

```
1: Inputs: Evironemnt State Vector $s_t$, Ride Requests, map-based locations.
2: Outputs: Matching, Dispatching and Pricing decisions based on learnt Q-values
3: Construct an initial state vector $s_t = (X_t, V_{t:t+T}, D_{t:t+T})$.
4: for $t \in$ Total Time . . . do
5:     Fetch all ride requests at time slot t
6:     Fetch all vehicles that entered the market in time slot t, $V_{new}$
7:     for each $V_j$ $in$ $V_{new}$ . . . do
8:         Run the DQN dispatch policy in Algorithm 2.
9:         Get the best dispatch action $a^j_t$ for each new vehicle $V_j$ from the Q-network.
10:        Find the shortest path to the dispatch location of vehicle $V_j$.
11:        Generate the trajectory of vehicle $V_j$.
12:        Get the list of q-values associated with map locations from the Q-network.
13:        Store a ranking $\alpha$ associated with each of the map locations.
14:        Create set $L$ of top $\lambda$ "desired" locations that gives highest utility.
15:    end for
16:    Fetch all vehicles whose $V_s \neq$ OFF_DUTY && $V_C \neq C^V_{max}$
17:    for each ride request $R_i \in$ time slot t . . . do
18:        Run the matching policy to assign vehicle $V_j$ to request/trip $R_i$ .
19:        Calculate the travel time till pickup $T_i$ using the ETA-model in [Alabbasi et al. 2019].
20:        Calculate initial price $P_{init}(R_i)$ for this request/trip.
21:        Send $P_{init}(R_i)$ to Vehicle $V_j$.
22:        Get $P(R_i)$ from vehicle $V_j$ after it evaluates given $loc(R_i)$ using [2].
23:        Send $P(R_i)$, $V^j_C$, and $V^j_T$ to the customer associated with request $R_i$.
24:        Get customer's decision $C^i_d$ after it evaluate its utility using [3].
25:        if $C^i_d ==$ True then
26:            if $V^j_C \neq 0$ then
27:                Update price for other on-board passengers.
28:            end if
29:            Update vehicle $V_j$ capacity as more requests can be assigned to it.
30:            Estimate the travel time using ETA model in [Alabbasi et al. 2019].
31:            Find the shortest path to the pickup location.
32:            Generate the trajectory of vehicle $V_j$.
33:            if $C^i$ does not prefer ride-sharing then
34:                Disable ride-sharing for Vehicle $j$.
35:            end if
36:        else
37:            Go to step 22. {if rejected > 3 times, reject the request.}
38:        end if
39:        Update the state vector $s_t$.
40:    end for
41:    Send the state vector $s_t$ to the Q-network.
42:    Fetch all vehicles whose $V_s ==$ IDLE &&$T_I > 10$ minutes, $V_{idle}$.
43:    for each available vehicle $V_n \in a_t$ ... do
44:        Run the DQN dispatch policy in Algorithm 2.
45:        Repeat steps 9 - 14.
46:    end for
47:    Update the state vector $s_t$.
48: end for
```

up, and the destination of each passenger. A vehicle is considered available if at least one of its seats is vacant that is, if and only if $V_C < C^V_{max}$. A vehicle becomes unavailable when all its seats are occupied or if it will not consider taking an extra passenger. Let $\gamma_{j,t}$ be a binary decision variable which is 1 if vehicle V_j decides to serve customers, otherwise it is 0. Only available vehicles can be dispatched in our algorithm. Let the number of available vehicles at region m at time slot t be $v_{t,m} = \sum_{n=1}^{N} \gamma_{n,t,m}\{\forall$ vehicle $n \in$ zone $m\}$. Using the vehicle's state information, we can predict the time slot at which that vehicle will become available (if it is currently unavailable). Thus, for a set of dispatch actions at time t, we can predict the number of vehicles in each zone for T time slots ahead, from time t_0 to time $t+T$, denoted by $V_{t:t+T}$ which serves as our predicted supply in each zone for T time slots ahead. An improvement upon our dispatching policies can be achieved by anticipating the demand in every zone through historical weekly/daily distribution of trips across zones [Wyld 2005]. Combining all this data, we have defined a three tuple that captures the environment updates at time t as $s_t = (X_t, V_{t:t+T}, D_{t:t+T})$. When a set of new ride requests arrive at the system, we can retrieve from the environment all the state elements, combined in one vector s_t. Also, when a passenger's request becomes accepted, we append the customer's expected pickup time, source, and destination to s_t as well. These variables change in real time according to the environment variations and demand/supply dynamics. However, our framework keeps track of all these rapid changes and seeks to make the demand, d_t, $\forall t$ and supply v_t, $\forall t$ close enough (i.e., mismatch between them is zero).

(3) *Action:* a^n_t denotes the action taken by vehicle n at time step t. This action has two parts: a)First, if the vehicle still has vacant seats, it decides whether to accept new passengers or to only serve the on-board customers, and b) if it makes the decision of accepting new customers or if it were initially totally empty, it needs to decide on which zone to head to at time step t. Naturally, a fully occupied vehicle cannot serve any additional customers. Finally, if a vehicle decides to only serve its existing on-board passengers, it uses the shortest optimal route to reach the destinations of its customers.

(4) *Reward:* having explained all of the above factors, at every time step t, the DQN agent obtains a representation for the environment, s_t, and a reward r_t that will be explained in 4.2. Based on this information, the agent takes an action that directs the vehicle (that is either idle or recently entered the market) to different dispatch zone where the expected discounted future reward is maximized, i.e.,

$$\sum_{k=t}^{\infty} \eta^{k-t} r_k(a_t, s_t) \tag{5}$$

where $\eta < 1$ is a time discounting factor. In our algorithm, we define the reward r_k as a weighted sum of different performance components that reflect the objectives of our DQN agent, which is thoroughly explained in 4.2 The reward will be learnt from the environment for individual vehicles and then leveraged by the DPRS optimizer to optimize its decisions.

Algorithm 2 Dispatching using DQN

```
1: Input: $X_t$, $V_{t:t+T}$, $D_{t:t+T}$.
2: Output: Dispatch Decisions
3: Construct a state vector $s_{t,n} = (X_t, V_{t:t+T}, D_{t:t+T})$.
4: Get the best dispatch action $a_{t,n} = argmax[Q(s_{t,n}, a, \theta)]$ for all
   vehicles $V_n$ using the Q-network.
5: Get the destination zone $Z_{t,j}$ for each vehicle $j \in V_n$ based on action
   $a_{t,j} \in a_{t,n}$
6: Update dispatch decisions by adding $(j, Z_{t,j})$
7: Return $(n, Z_{t,n})$
```

Our framework keeps track of the rapid changes of all these variables and seeks to make the demand, d_t, $\forall t$ and supply v_t, $\forall t$ close enough (mismatch between them is zero). Note that, by ride-sharing we mean ride-sharing with pooling in our model.

4.2 DQN Dispatch Agent

At every time step t, the DQN agent obtains a representation for the environment, $s_{t,n}$, and calculates a reward r_t associated with each dispatch-to location in the action space $a_{t,n}$. Based on this information, the agent takes an action that directs the vehicle to different dispatch zone where the expected discounted future reward is maximized as in equation (5). In our algorithm, we define the reward r_k as a weighted sum of different performance components that reflect the objectives of our DQN agent. The reward will be learnt from the environment for individual vehicles and then leveraged by the agnet/optimizer to optimize its decisions. The decision variables are i) Dispatching of an available vehicle in zone m, $V_j \in v_{t,m}$ to another zone at time slot t, ii) if a vehicle V_j is not full, decide $\gamma_{j,t}$ its availability for serving new customers at time slot t. If the vehicle is full, then $\gamma_{j,t} = 0$. If it is empty, it will serve new passengers whose requests generate within the vehicle V_j's current region at time t.

We define the overall objectives of the dispatcher, where our dispatch policy aims to (1) minimize the supply-demand mismatch: (diff$_t$), (2) minimize the dispatch time: T^D_t (i.e., the expected travel time of vehicle V_j to go zone m at time step t), (3) minimize the extra travel time a vehicle takes for car-pooling compared to serving one customer: Δt, (4) maximize the fleet profits P_t, and (5) minimize the number of utilized vehicles: e_t. Each of these objectives is represented with a corresponding term, and the DQN overall reward function is represented as a weighted sum of these terms as follows:

(1) Minimize the supply-demand mismatch, recall that $v_{t,m}$, and $\bar{d}_{t,m}$ denotes the number of available vehicles, and the anticipated demand respectively at time step t in zone m. We want to minimize their difference over all M zones, therefore, we get:

$$\text{diff}_t = \sum_{m=1}^{M} (\bar{d}_{t,m} - v_{t,m}) \tag{6}$$

The reward will be learnt from the environment for individual vehicles, therefore, we map this term for individual vehicles. When vehicle serves more requests, the difference between supply and demand is minimized, and helps satisfy the demand of the zone it is located in. Therefore, we can get the total number of customers served by vehicle n at time step t:

$$\mathrm{C}_{t,n} = \sum_{m=1}^{M} v_{t,m}^{n} \quad (\text{where } v_{t,m} = 1 \text{ when } v_{t,m} < \bar{d}_{t,m})$$
$$\text{where } \sum_{m=1}^{M} v_{t,m}^{n} = 1 \; (\gamma_{n,t,m} \in \{0,1\} \text{ where } \; n \in v_{t,m}) \quad (7)$$

(2) Minimize the dispatch time, which refers to the expected travel time of vehicle V_j to go zone m at time step t, denoted by $h_{t,m}^{j}$. We calculate this time from the location of vehicle V_j at time t which is already included in the state variable $X_{t,j}$. Idle vehicles get dispatched to different zones (where anticipated demand is high) than their current zones (even if they do not have any new requests yet), in order to pick up new customers in the future. Since we want to minimize over all available vehicles N over all zones M within time t, we get the total dispatch time, T_t^D as follows:

$$T_t^D = \sum_{n=1}^{N} \sum_{m=1}^{M} h_{t,m}^{n} \quad \{\forall\, n \in v_{t,m}\} \tag{8}$$

For individual vehicles, considering the neighboring vehicles' locations while making their decision, we get for vehicle n at time step t:

$$T_{t,n}^D = \sum_{m=1}^{M} h_{t,m}^{n} \quad \{\text{where } \; n \in v_{t,m}\} \tag{9}$$

(3) Minimize the difference in times that the vehicle would have taken if it only serves one customer and the time it would take for car-pooling. For vehicles that participate in ride-sharing, an extra travel time may be incurred due to (1) either taking a detour to pickup an extra customer or (2) after picking up a new customer, the new optimal route based on all destinations might incur extra travel time to accommodate the new customers. This will also imply that customers already on-board will encounter extra delay. Therefore, that difference in time needs to be minimized, otherwise both customers and drivers would be disincentivized to car-pool. Let t' be the total time elapsed after the passenger l has requested the ride, $t_{n,l}$ be the travel time that vehicle n would have been taken if it only served rider l, and $\tilde{t}_{n,l}$ be the updated time the vehicle n will now take to drop off passenger l because of the detour and/or picking up a new customer at time t. Note that $\tilde{t}_{n,l}$ is updated every time a new customer is added. Therefore, for vehicle n, rider l at time step t, we want to minimize: $\xi_{t,n,l} = t' + \tilde{t}_{n,l} - t_{n,l}$. But for vehicle n, we want to minimize over all of its passengers, thus: $\sum_{l=1}^{\cup_n} \xi_{t,n,l}$, where $\cup_n$ is the total number of chosen users for pooling at vehicle n till time t. Note that $\cup_n$ is not known apriori, but will be adapted dynamically in the DQN policy. It will also vary as the passengers are picked or dropped by vehicle n. We want to optimize over all N vehicles, therefore, the total extra travel time can be represented as:

$$\Delta t = \sum_{n=1}^{N} \sum_{l=1}^{\cup_n} \xi_{t,n,l}. \tag{10}$$

For individual vehicles, extra travel time for vehicle n at time step t becomes:

$$T_{t,n}^E = \sum_{l=1}^{\cup_n} \xi_{t,n,l}. \tag{11}$$

(4) Maximize the fleet profits. This is calculated as the average earnings E_t minus the average cost of all vehicles. Cost is calculated by dividing the total travel distance of vehicle V_j by its mileage, and multiplied by the average gas price P_G. Therefore, the average profits for the whole fleet can be represented as:

$$\mathbb{P}_t = \sum_{n=1}^{N} E_{t,n} - \left[\frac{D_{t,n}}{M_V^n} * P_G \right] \tag{12}$$

But, since we are estimating the reward for individual vehicles, we get for vehicle n at time step t, the average profits becomes:

$$\mathbb{P}_{t,n} = E_{t,n} - \left[\frac{D_{t,n}}{M_V^n} * P_G \right] \tag{13}$$

(5) Minimize the number of utilized vehicles/resources. We capture this by minimizing the number of vehicles that become active from being inactive at time step t. Although we are minimizing the number of active vehicles in time step t, if the total distance or the total trip time of the passengers increase, it would be beneficial to use an unoccupied vehicle instead of having existing passengers encounter a large undesired delay. Let $e_{t,n}$ represent whether vehicle n is non-empty at time step t. The total number of vehicles that recently became active at time t is given by:

$$e_t = \sum_{n=1}^{N} \left[\max(e_{t,n} - e_{t-1,n}, 0) \right] \tag{14}$$

Having defined all our objective terms, we represent the DQN reward function as a weighted sum of these terms as follows:

$$r_t = -\left[\beta_1 \mathrm{diff}_t + \beta_2 T_t^D + \beta_3 \Delta t \right] + \beta_4 \mathbb{P}_t - \beta_5 e_t \tag{15}$$

Note, from equation 5, that we maximize the discounted reward over a time frame. The negative sign here indicates that we want to minimize the terms within the bracket.

Note that weights $\beta_1, \beta_2, \beta_3, \beta_4$ and β_5 depend on the weight factors of each of the objectives. Further, we maximize the discounted reward over a time frame, and the negative sign here indicates that we want to minimize the terms within the function. Finally, note that the reward for vehicle n is 0 if it decides to only serve the passengers on-board (if, any). Therefore, we focus on the scenario where vehicle n decides to serve a new user and it is willing to take a detour at time t. In this case, the reward $r_{t,n}$ for vehicle n at time slot t is represented equation 16, where the objectives above are mapped to: (1) $C_{t,n}$: number of customers served by vehicle n at time t, (2), (3) dispatch time and extra travel time are the same, denoted by: $T_{t,n}^D$, and $T_{t,n}^E$. (4) average profit for vehicle n at time t, $\mathbb{P}_{t,n}$. In this case, the reward $r_{t,n}$ for vehicle n at time t is represented by:

$$\begin{aligned} r_{t,n} = r(s_{t,n}, a_{t,n}) = \beta_1 C_{t,n} + \beta_2 T_{t,n}^D + \beta_3 T_{t,n}^E + \beta_4 \mathbb{P}_{t,n} \\ + \beta_5 [\max(e_{t,n} - e_{t-1,n}, 0)] \end{aligned} \tag{16}$$

In equation 16, the last term captures the status of vehicle n where $e_{t,n}$ is set to 1 if vehicle n was empty and then becomes occupied at time t (even if by one passenger), however, if it was already occupied and just takes a new customer, $e_{t,n}$ is 0. The intuition here is that if an already occupied vehicle serves a new user, the congestion and fuel costs will be less when compared to when an empty vehicle serves that user. Note that if we make β_3 very large, it will disincentivize passengers and drivers from making detours to serve other passengers, Thus, the setting becomes similar to the one in [Oda and Joe-Wong 2018], where there is no carpooling.

Note that the additional profits term P_t integared with the reward function makes the output expected discounted rewards (Q-values) associated with each possible move on the map, a good reflection of the expected earnings gained when heading to these locations. This gives drivers an insight about how the supply-demand is distributed over the city, and assists them in making knowledgeable and informed decisions when it comes to ranking their desired go-to locations that can yield them higher profits (potentail hotspots), and thus making the corresponding pricing decisions (Section 3.2).

4.3 DQN Architecture

The output of the DQN represents the Q-value for each possible movement/dispatch. In our simulator, the service area is divided into 43x44, cells each of size 800mx800m. The vehicle can move (vertically or horizontally) at most 7 cells, and hence the action space is limited to these cells. A vehicle can move to any of the 14 vertical (7 up and 7 down) and 14 horizontal (7 left and 7 right). This results in a 15x15 map for each vehicle as a vehicle can move to any of the 14 cells or it can remain in its own cell. The input to the neural network consists of the state representation, demand and supply, while the output is the Q-values for each possible action/move (15 moves). The input consists of a stack of four feature planes of demand and supply heat map images each of size 51x51. In particular, first plane includes the predicted number of ride requests next 30 minutes in each region, while the three other planes provide the expected number of available vehicles in each region in 0; 15 and 30 minutes. Before passing demand and supply images into the network, different sizes of average pooling with stride (1, 1) to the heat maps are applied, resulting in 23 x 23 x 4 feature maps. The first hidden layer convolves 16 filters of size 5x5 followed by a rectifier non-linearity activation. The second and third hidden layers convolve 32 and 64 filters of size 3x3 applied a rectifier non-linearity. Then, the output of the third layer is passed to another convolutional layer of size 15 x 15 x 128. The output layer is of size 15 x 15 x 1 and convolves one filter of size 1x1. Since reinforcement learning is unstable for nonlinear approximations such as the neural network, due to correlations between the action-value, we use experience replay to overcome this issue. Since every vehicle runs its own DQN policy, the environment during training changes over time from the perspective of individual vehicles.

4.4 Learning Expected Discounted Reward Function —Q-values —

In our algorithm, we use reinforcement learning to learn the reward function stated in (16) using DQN. Through learning the probabilistic dependence between the action and the reward function, we learn the Q-values associated with the probabilities $P(r_t \mid a_t, s_t)$ over time by feeding the current states of the system. Instead of assuming any specific structure, our model-free approach learns the Q-values dynamically using convolutional neural networks whose architecture is described in 1. Deep queue networks are utilized to dynamically generate optimized values. This technique of learning is characterized by its high adaptability to dynamic features in the system, which is why it is widely adopted in modern decision-making tasks. The optimal action-value function for vehicle n is defined as the maximum expected achievable reward. Thus, for any policy π_t we have:

$$Q^*(s,a) = max_\pi \ \mathbb{E} \left[\sum_{k=t}^{\infty} \eta^{k-t} r_{k,n} \mid (s_{t,n} = s, a_{t,n} = a, \pi_t) \right] \tag{17}$$

where $0 < \eta < 1$ is the discount factor for the future. If η is small (large, resp.), the dispatcher is more likely to maximize the immediate (future, resp.) reward. At any time slot t, the dispatcher monitors the current state s_t and then feeds it to the neural network (NN) to generate an action. In our algorithm, we utilize a neural network to approximate the Q function in order to find the expectation.

For each vehicle n, an action is taken such that the output of the neural network is maximized. The learning starts with no knowledge and actions are chosen using a greedy scheme by following the Epsilon-Greedy method. Under this policy, the agent chooses the action that results in the highest Q-value with probability $1 - \epsilon$, otherwise, it selects a random action. The ϵ reduces linearly from 1 to 0.1 over T_n steps. For the n^{th} vehicle, after choosing the action and according to the reward $r_{t,n}$, the Q-value is updated with a learning factor σ as follows:

$$\begin{aligned} Q^{'}(s_{t,n}, a_{t,n}) \ \leftarrow \ & (1-\sigma)\, Q(s_{t,n}, a_{t,n}) \\ & + \sigma \left[r_{t,n} + \eta \max_a Q(s_{t+1,n}, a) \right] \end{aligned} \tag{18}$$

Similar to ϵ, the learning rate σ is also reduced linearly from 0.1 to 0.001 over 10000 steps. We note that an artificial neural network is needed to maintain a large system space. When updating these values, a loss function $L_i(\theta_i)$ is used to compute the difference between the predicted Q-values and the target Q-values, i.e.,

$$L_i(\theta_i) \ = \ \mathbb{E} \left[\left((r_t \ + \ \eta \max_a Q(s,a;\bar{\theta_i})) \ - Q(s,a;\theta_i) \right)^2 \ \right] \tag{19}$$

where $\theta_i, \bar{\theta_i}$, are the weights of the neural networks. This above expression represents the mean-squared error in the Bellman equation where the optimal values are approximated with a target value of $r_t \ + \ \eta \max_a Q(s,a;\bar{\theta_i})$, using the weight $\bar{\theta_i}$ from some previous iterations.

The Q-values are then used to decide on the best dispatching action to take for each individual vehicle. Since the state space is large, we don't use the full representation of s_t, instead a map-based input is used to alleviate this massive computing. Note that, the Q-values depend on the pricing since the decisions made by customers and drivers impact the reward function.

5 SIMULATION AND EVALUATION RESULTS

5.1 Simulator Setup

Our simulator is created based on real public dataset of taxi trips in Manhattan, New York city [NYC.gov 2019]. We start by populating vehicles over the city, randomly assigning each vehicle a type and an initial location. According to the type assigned to each vehicle, we set the accompanied features accordingly such as: maximum capacity, mileage, and price rates (per mile of travel distance ω^1, and per waiting minute ω^3). We initialize the number of vehicles, to 8000, allowing a portion of vehicles to enter the market in every time step t. We consider the data of June 2016 for training, and one week from July 2016 for evaluations. We trained our DQN neural networks using the data from June 2016 for 10000 epochs and used the most recent 5000 experiences as a replay memory. For each trip, we obtain the pick-up time, location, passenger count, and drop-off locations. We use this trip information to construct travel demand prediction model. Further, we use *Python* and *Tensorflow* to implement our DPRS framework. To initialize the environment, we run the simulation for 20 minutes without dispatching the vehicles. Further, we run the simulator for 8 days, and thus T = 8*24*60 steps, where Δt = 1 minute. Further, we set $\beta_1 = 10$, $\beta_2 = 1$, $\beta_3 = 5$, $\beta_4 = 12$, $\beta_5 = 8$, $\lambda = 10\%$, $\omega^4 = 15$, $\omega^5 = 1$, and $\omega^6 = 4$.

5.2 Evaluation Metrics and Baselines:

We breakdown the reward as well as the drivers' and customers' utilities and investigate the performance for different baselines. Recall that we want to minimize the components of our reward function: supply-demand mismatch, average travel distance per vehicle, and number of used vehicles captured by utilization rate. We note that the supply-demand mismatch is reflected in our simulation through a reject rate metric. Recall that a request is rejected if there is no vehicle around in a range of $5km^2$ that is available to serve the request (denoted as: Non-matched), or if it was rejected by a customer or driver (denoted as: User Rejected). We analyze the impact of both types of rejections. Also, the metric of idle time represents the time at which a vehicle is not occupied while still incurring gasoline cost and not gaining revenue. Further, the drivers' average profits, and the customers' waiting time are also evaluated. We compare our proposed DPRS Framework (with dispatching, ride-sharing and pricing strategy) against four baslines where the dispatching, ride-sharing, or pricing (denoted as D, RS and PS respectively) are omitted and marked (!) accordingly as follows:

- No Dispatch, No Ride-sharing, No Pricing Strategy, (!D, !RS, !PS): In this setting, vehicles don't get dispatched to areas with anticipated high demand, no matter how long they stay idle. Ride-sharing is not allowed, every vehicle serves only one request at a time. Also, initial pricing is accepted by both drivers and customers, by default.
- No Dispatch with Ride-sharing but No Pricing Strategy (!D, RS, !PS): similar to (!D, !RS, !PS) except that ride-sharing is allowed, where vehicles can serve more than one request altogether.
- Dispatch with No Ride-sharing and No Pricing Strategy (D, !RS, !PS): Here, vehicles are dispatched when idle but, ride-sharing is not allowed similar to the setting in [Oda and Joe-Wong 2018].
- Dispatch with Ridesharing but No Pricing Strategy (D, RS, !PS): similar to (D, !RS, !PS), but here ride-sharing is allowed, similar to DeepPool in [Alabbasi et al. 2019].

5.3 Experimental Results

Involving drivers and customers in the decision-making process was expected to increase both the rejection rate, and number of vehicles utilized to serve the demand. However, even after adding the distributed pricing based on the Q-Network that gives more control to the customers and drivers, the ride-sharing framework has significantly low rejection rate. Figure 2 shows that DPRS performs almost equally well as DeepPool, where dispatching and ride-sharing are considered. This implies both the drivers and customers are achieving a compromise that is profitable and convenient to them. We can observe that in rush hours, non-matched requests peaks while user rejected requests remains minimal.

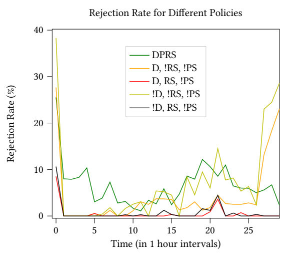

Figure 2: Rejection Rate

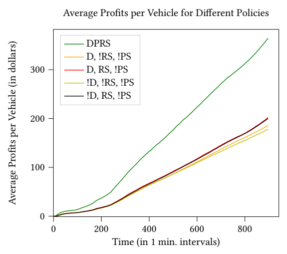

Figure 3: Average Profits

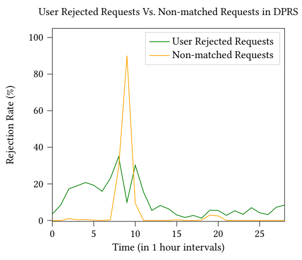

Figure 4: Reject Types

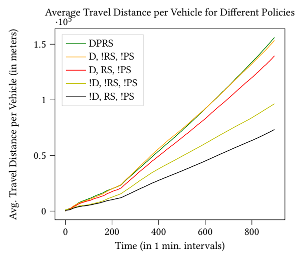

Figure 5: Avg. Distance

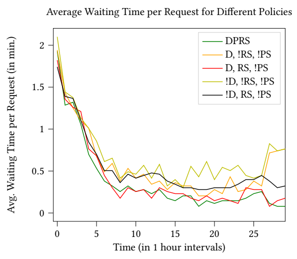

Figure 6: Avg. Wait Time

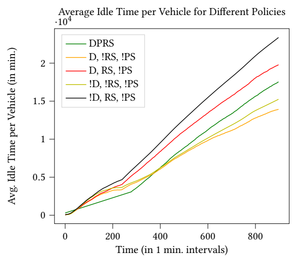

Figure 7: Avg. Idle Time

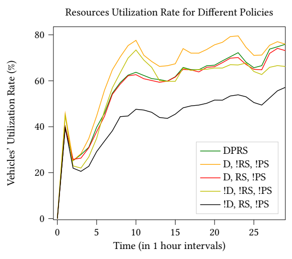

Figure 8: Utilization Rate

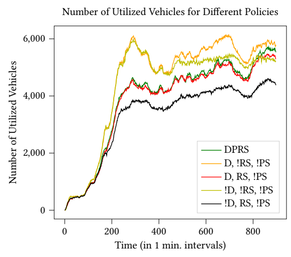

Figure 9: Number of Utilized Vehicles

Also, DPRS enhances the utilization of vehicles (i.e. makes use of every available seat, while minimizing the total number of vehicles in use). Figure 3 shows the average profits for the drivers has significantly increased over time compared to all the other four protocols. Thus, quantifying the individual drivers' preferred zones based on the learnt reward using DQN, could guarantee them a significant improvement in earnings that, in turn, helped them make up for any extra encountered cost and boost their profits. To further analyze the rejection rate, Figure 4 shows that in DPRS the rejection rate made by the customers (i.e. when a customer weighs in and rejects a ride) is fairly close to the naturally encountered rejection rate that occurs due to the unavailability of vehicles within the request's vicinity.

We have shown in Figure 3 that DPRS yields a significant improvement in drivers' earnings due to quantifying their individual preferred zones based on the learnt reward using DQN. However, Figure 5 shows that the average travel distance of DPRS is a little higher than the DeepPool ride-sharing framework. Since DPRS provides significantly higher profits for drivers, if it comes at the cost of a slight increase in travel distance; that becomes an advantage of DPRS. Note that, the two policies with the lowest travel distance in Figure 5, are non-dispatching which explains why vehicles have lower travel distance on average. However, we can observe that these non-dispatching protocols as well as the non-ride-sharing protocols are not efficient as they result in lower profit margins (Figure 3), and higher customers' waiting time (Figure 6).

Moreover, Figure 7 shows that non-dispatching protocols yield higher idle time for vehicles as they might spend large amount of time being idle and they never get dispatched to higher demand areas. On contrast, non-ride-sharing protocols yield lower idle time, but they are still inefficient as vehicles spend more time on duty while serving lower number of customers than the ride-sharing protocols. Compared to DeepPool framework, the waiting time per request is lower for DPRS approach. As shown in Figure 6, it reduces over time to less than a minute. On average, vehicles' idle time between requests is within a minimal range for DPRS. Figure 7 shows that the average idle time for DPRS is considerably less than that of DeepPool framework as well, which also proves better overall utilization of vehicles.

Figure 8 shows that DPRS edges higher than 3 other policies, while the only higher policy is not adopting ride-sharing, and therefore it requires higher number of vehicles to serve the same demand. Figure 9 supports our hypothesis, showing that DPRS utilizes around the same number of vehicles as DeepPool, while at the same time, involving both customers and drivers in the decision-making process. Clearly, non-dispatching and non-ride-sharing algorithms are shown to have poor utilization of resources, as they use higher number of vehicles to serve the same amount of demand. Since involving customers and drivers in the decision-making process was supposed to boost both rejection rate and number of vehicles utilized to serve the demand, while DPRS performs equally well as DeepPool in both metrices; this makes DPRS superior to DeepPool.

6 CONCLUSION

Integrating the preferences of customers and drivers in terms of pricing and convenience in ride-sharing is a non-trivial problem. In this paper, we presented a novel distributed pricing-based ride-sharing (DPRS) framework that (1) utilizes deep reinforcement learning methodologies, (2) integrates dispatching vehicles when idle to areas with anticipated high demand, and (3) vehicle-passenger matching. The framework uses customer and driver utility functions, impacted by deep Q-learning strategy. Since our approach is distributed, utility functions of both customers and drivers compute the best pricing actions independently and influence the ride-sharing decisions based on their convenience. Through experiments, we show that our framework reduces the customers' fare and waiting time, the vehicle's idle time, and at the same time increases drivers' average profits while maintaining low rejection rate. Since DPRS is a model-free approach, it is applicable for large-scale ride-sharing systems. Extension of this work to include capabilities of a joint delivery system for passengers and goods as in [Manchella et al. 2020], or using multi-hop routing of passengers as in [Singh et al. 2019] for efficient fleet utilization is left as a future work.

ACKNOWLEDGMENTS

This research is supported, in part, by DARPA [2] and AFRL [3] under contract number W911NF2020003 [4].

REFERENCES

A. Alabbasi, A. Ghosh, and V. Aggarwal. 2019. DeepPool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning. In *EEE Transactions on Intelligent Transportation Systems*, Vol. 20.12.

X. Bei and S. Zhang. 2018. Algorithms for trip-vehicle assignment in ride-sharing. In *Thirty-Second AAAI Conference on Artificial Intelligence.*

Vinícius Monteiro de Lira, Valeria Cesario Times, Chiara Renso, and Salvatore Rinzivillo. 2015. ComeWithMe: An activity-oriented carpooling approach. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems.* 2574–2579.

D. A. Hennessy and D. L. Wiesenthal. 2019. Traffic congestion, driver stress, and driver aggression. In *Aggressive Behavior: Official Journal of the International Society for Research on Aggression*, Vol. 25.6. 409–423.

Abhinav Jauhri, Brian Foo, Jerome Berclaz, Chih Chi Hu, Radek Grzeszczuk, Vasu Parameswaran, and John Paul Shen. 2017. Space-time graph modeling of ride requests based on real-world data. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence.*

A. Kleiner, B. Nebel, and V. A. Ziparo. 2011. A mechanism for dynamic ride sharing based on parallel auctions. In *ITwenty-Second International Joint Conference on Artificial Intelligence.*

Wei Lu. 2015. *Optimization and mechanism design for ridesharing services.* Ph.D. Dissertation.

Shuo Ma, Yu Zheng, and Ouri Wolfson. 2014. Real-time city-scale taxi ridesharing. *IEEE Transactions on Knowledge and Data Engineering* 27, 7 (2014), 1782–1795.

Kaushik Manchella, Abhishek K Umrawal, and Vaneet Aggarwal. 2020. FlexPool: A Distributed Model-Free Deep Reinforcement Learning Algorithm for Joint Passengers & Goods Transportation. *arXiv preprint arXiv:2007.13699* (2020).

NYC.gov. 2019. NYC Taxi and Limousine Commission-Trip Record Data. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

T. Oda and C. Joe-Wong. 2018. MOVI: A Model-Free Approach to Dynamic Fleet Management. In *arXiv preprint arXiv:1804.04758.*

D. Schrank, B. Eisele, and T. Lomax. 2019. 2019 urban mobility report. *Texas A&M Transportation Institute* (2019). https://mobility.tamu.edu/umr/

Ashutosh Singh, Abubakr Alabbasi, and Vaneet Aggarwal. 2019. A distributed model-free algorithm for multi-hop ride-sharing using deep reinforcement learning. *arXiv preprint arXiv:1910.14002* (2019).

Na Ta, Guoliang Li, Tianyu Zhao, Jianhua Feng, Hanchao Ma, and Zhiguo Gong. 2017. An efficient ride-sharing framework for maximizing shared route. *IEEE Transactions on Knowledge and Data Engineering* 30, 2 (2017), 219–233.

D. Wyld. 2005. Where is my suitcase? RFID and airline customer service. *Marketing Intelligence & Planning Journal* 23 (2005), 382–394.

Desheng Zhang, Tian He, Shan Lin, Sirajum Munir, and John A Stankovic. 2016. Taxi-passenger-demand modeling based on big data from a roving sensor network. *IEEE Transactions on Big Data* 3, 3 (2016), 362–374.

R. Zhang and M. Pavone. 2016. Control of robotic mobility-on-demand systems: a queueing-theoretical perspective. In *The International Journal of Robotics Research*, Vol. 35(1-3). 186–203.

Kai Zhao, Denis Khryashchev, Juliana Freire, Claudio Silva, and Huy Vo. 2016. Predicting taxi demand at high spatial resolution: Approaching the limit of predictability. In *2016 ieee international conference on Big data (big data).* IEEE, 833–842.

[2]the Defense Advanced Research Projects Agency

[3] the Air Force Research Laboratory

[4]The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, AFRL, or the U.S. Government.