A Distributed Model-Free Ride-Sharing Approach for Joint Matching, Pricing, and Dispatching Using Deep Reinforcement Learning

Marina Haliem, Ganapathy Mani, Vaneet Aggarwal, *Senior Member, IEEE*, and Bharat Bhargava, *Life Fellow, IEEE*

***Abstract*— Significant development of ride-sharing services presents a plethora of opportunities to transform urban mobility by providing personalized and convenient transportation while ensuring the efficiency of large-scale ride pooling. However, a core problem for such services is route planning for each driver to fulfill the dynamically arriving requests while satisfying given constraints. Current models are mostly limited to static routes with only two rides per vehicle (optimally) or three (with heuristics) (Alonso-Mora *et al.*, 2017), at least in the initial allocation while not ascertaining that opposite-direction rides are not grouped together. In this paper, we present a dynamic, demand aware, and pricing-based vehicle-passenger matching and route planning framework that (1) dynamically generates optimal routes for each vehicle based on online demand, pricing associated with each ride, vehicle capacities and locations. This matching algorithm starts greedily and optimizes over time using an insertion operation, (2) involves drivers in the decision-making process by allowing them to propose a different price based on the expected reward for a particular ride as well as the destination locations for future rides, which is influenced by supply-and-demand computed by the Deep Q-network. (3) allows customers to accept or reject rides based on their set of preferences with respect to pricing and delay windows, vehicle type and carpooling preferences. These (1-3) in tandem with each other enforce grouping rides with the most route-intersections together. (4) Based on demand prediction, our approach re-balances idle vehicles by dispatching them to the areas of anticipated high demand using deep Reinforcement Learning (RL). Our framework is validated using millions of trips extracted from the New York City Taxi public dataset; however, we consider different vehicle types and designed customer utility functions to validate the setup and study different settings. Experimental results show the effectiveness of our approach in real-time and large scale settings.**

Manuscript received June 30, 2020; revised February 15, 2021, May 24, 2021, and June 30, 2021; accepted July 5, 2021. Date of publication August 4, 2021; date of current version November 29, 2021. This work was supported in part by the Office of the Assistant Secretary of Defense for Health Affairs under Contract W81XWH-18-1-0769, in part by the National Science Foundation under Grant CNS-1618335, and in part by CISCO. This article was presented in part at ACM CSCS 2020 [1]. The Associate Editor for this article was E. I. Vlahogianni. *(Corresponding author: Vaneet Aggarwal.)*

Marina Haliem and Bharat Bhargava are with the Department of Computer Science, Purdue University, West Lafayette, IN 47907 USA (e-mail: mwadea@purdue.edu; bbshail@purdue.edu).

Ganapathy Mani was with the Department of Computer Science, Purdue University, West Lafayette, IN 47907 USA. He is now with Qualcomm, San Diego, CA 92121 USA (e-mail: manig@purdue.edu).

Vaneet Aggarwal is with the School of Industrial Engineering and the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: vaneet@purdue.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TITS.2021.3096537, provided by the authors.

Digital Object Identifier 10.1109/TITS.2021.3096537

***Index Terms*— Ride-sharing, deep reinforcement learning, vehicle routing, pricing, deep Q-network, pooling, shared mobility, route planning.**

I. 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. Given tens of millions of users per month [3], these mobility-on-demand (MoD) services introduce a new paradigm in urban mobility —ride-sharing or ride-splitting. These ride-hailing services, when adopting shared mobility, can provide an efficient and sustainable way of transportation [4]. With higher usage per vehicle, its service can reduce traffic congestion, environmental pollution, as well as energy consumption, thereby enhance living conditions in urban environments [5]. The growth in demand for these services coupled with the rise of self-driving technology points towards a need for a fleet management framework that accommodates both the drivers' and customers' preferences in an optimal and sustainable manner. 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 a limitation on the money to spend on a particular ride as well as time constraints on reaching the destination. On the other hand, the 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 a low demand area. Thus, a reliable framework is needed to identify trade-offs between drivers' and customers' needs and make a compromised decision that is favorable to both. Our proposed intelligent transportation system is driven by the objective of maintaining service levels for customers while accommodating both drivers' and customers' preferences. Thus, it has the potential to revolutionize modern transportation systems.

The key contributions, in this paper, can be summarized as:

- We present a novel dynamic, Demand-Aware and Pricing-based Matching and route planning (DARM) framework that is scalable up to the maximum capacity per vehicle in the initial assignment phase. In the optimization phase, this algorithm takes into account the near-future demand as well as the pricing associated with each ride in order to improve the route-planning by eliminating rides heading towards opposite directions and applying insertion operations to vehicles' current routes.

- In addition to our matching and route-planning (DARM) framework, we integrate a novel Distributed Pricing approach for Ride-Sharing -with pooling- (DPRS) framework where, based on their convenience, customers and drivers get to weigh-in on the decision-making of a particular ride. The key idea is that the passengers are offered price based on the additional distance given the previous matched passengers thus prioritizing the passengers which will have intersections in the routes to be grouped together.
- In the DPRS framework, drivers are allowed to propose a price based on the location of the ride that accounts for the reward of DQN based on the destination location. Similarly, customers can either accept or reject rides based on their pricing and timing thresholds, vehicle type, and number of people to share a ride with.
- Our joint (DARM + DPRS) framework increases the profit margins of both customers and drivers, and the profits are also fed back to the reinforcement learning utility functions that influence the Q-values learnt using DQN for making the vehicles' dispatch decisions. The optimization problem is formulated such that our novelty framework minimizes the rejection rate, customers' waiting time, vehicles' idle time, the total number of vehicles to reduce traffic congestion, fuel consumption, and maximizes the vehicle's profit.
- We simulate the ride-sharing system[1] using real-world dataset of New York City's taxi trip records (15 million trips) [38]. Experimental results show that our novel Joint (DARM + DPRS) framework provides 10 times more profits for drivers when compared to various baselines, while maintaining waiting times of < 1 min. for customers. Besides, we show a significant improvement in fleet utilization, utilizing only 50% of the vehicles allowed by the system to fulfill $\approx 96\%$ of the demand, in contrast to baselines that utilize $> 80\%$ of allowed vehicles to serve $< 60\%$ of the demand.

The rest of this paper is organized as follows: The detailed related work is provided in Appendix A. Section II describes the overall architecture of our framework as well as the model parameters. Section III, explains our dynamic, demand-aware, and pricing-based matching and route planning. In Section IV, we provide details for our pricing strategy, including customers' and drivers' utility functions and their decision-making processes. In Section V, we describe the DQN-based approach utilized for dispatching idle vehicles. Simulation setup and experimental results are presented in Section VI. Finally, Section VII concludes the paper.

II. Distributed Joint Matching, Pricing, and Dispatching Framework

We propose a novel distributed framework for matching, pricing, and dispatching in ride-sharing environments using Deep Q-Network (DQN), where initial matchings (that are

[1]The code for this work is available at https://github.itap.purdue.edu/Clan-labs/Dynamic_Matching_RS.

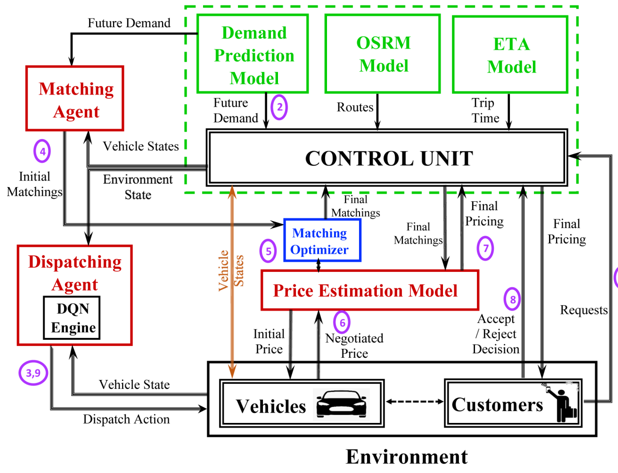


Fig. 1. Overall architecture of the proposed framework.[2]

decided in a greedy fashion) are then optimized in a distributed manner (per vehicle) in order to meet the vehicle's capacity constraints as well as minimize customers' extra waiting time and driver's additional travel distance. This framework involves customers and drivers (will be referred to as Agents henceforth) 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 they first enter the market, or when they spend a long time being idle (searching for a ride). Vehicles' dispatch decisions are made in parallel, since drivers learn the location updates of other vehicles in real-time (e.g., GPS), so it is unlikely for two drivers to take actions at the same exact time. Therefore, our algorithm learns the optimal policy for each agent independently as opposed to centralized-based approaches [32].

A. Model Architecture

Figure 1 shows the basic components of our joint framework and the interaction steps (in purple) between them. We assume that the central control unit is responsible for: (1) 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. (2) The control unit also has some internal components that help manage the ride-sharing environment such as: (a) the estimated time of arrival (ETA) model used to calculate and continuously update the estimated arrival time. (b) The Open Source Routing Machine (OSRM) model used to generate the vehicle's optimal trajectory to reach a destination, and (c) the (Demand Prediction) model used to calculate the future anticipated demand in all zones. We adopt these three models from [32]; whose details and their relevant calculations are provided in Appendix B. For every time step, first, the ride requests are input to the system (Step 1 in Fig. 1) along with the heat map for supply and demand (which involves demand prediction

[2]Enlarged figure is provided in Fig. 7 in Appendix D.

in the near future) [Step 2 in Fig. 1]. Then, based on the predicted demand, vehicles adopt a dispatching policy using DQN, where they get dispatched to zones with anticipated high demand (Step 3). This step not only takes place when vehicles first enter the market (lines 3-4 in Algorithm 1), but also when they experience large idle durations (at the end of every time step, this gets checked for - lines 23-24 (Step 9)). Then, each vehicle receives the updated environment state from the control unit and performs initial greedy vehicle-passenger(s) matching (Step 4 in Fig. 1), where one request (or more) gets assigned to the nearest vehicle based on its maximum passenger capacity. Next, communicating with the Price Estimation model, each vehicle calculates the corresponding initial pricing associated with each request (Step 5). Afterward, each vehicle executes its matching optimizer module that performs an insertion-based route planning (Step 6). In this step, vehicles reach their final matchings list by dealing with their initial matchings list in the order of their proximity, performing an insertion operation to its current route plan (as long as this insertion satisfies the capacity, extra waiting time, and additional travel distance constraints to guarantee that serving this request would yield a profit). Using the expected discounted reward learnt from DQN (in step 3), and the ride's destination, vehicles weigh their utility based on the potential hotspot locations, and propose new pricing for the customer (at the end of steps 5 and 6). This takes place on a customer-by-customer basis, where a vehicle upon inserting a customer into its current route plan, proposes to him/her the new price (Step 7). Then, the customer has the ability to accept or reject based on their own independent utility function (Step 8 in Fig. 1). Finally, upon receiving the customer's decision, the driver either confirms the addition of this ride to its route plan or removes it. Algorithm 1 shows the overall flow of our framework. Note that lines (7-22) that correspond to steps (3-9) take place in parallel at each vehicle.

The proposed model is distributed in the sense that each vehicle solves its own DQN and utilizes the output Q-values to make matching, pricing, route planning, and dispatching decisions, without communicating with other vehicles in its vicinity. However, a vehicle would consider the locations of nearby vehicles while making its independent decisions. So, a vehicle can communicate with the control unit, as needed, to request new information of the environment (prior to making decisions) or update its own status (after any decision).

B. Model Parameters and Notations

We built a ride-sharing simulator to train and evaluate our framework. We simulate New York City as our area of operation, where the map is divided into multiple non-overlapping regions, a grid with each 1 square mile being taken as a zone. This allows us to discretize the area of operation and thus makes the action space—where to dispatch the vehicles—-tractable. This discretization prevents our state and action space from exploding thereby making implementation feasible. We use $m \in \{1, 2, 3, \cdots, M\}$ to denote the city's zones, and n to denote the number of vehicles. A vehicle is marked as *available* if it has any remaining seating capacity. Available vehicles in zone i at time slot t is denoted $v_{t,i}$. We optimize our algorithm over T time steps, each of duration Δt. Idle vehicles make decisions on where on the map to head-to to serve the demand at each time step $\tau = t_0, t_0 + \Delta t, t_0 + 2(\Delta t), \ldots, t_0 + T(\Delta t)$ where t_0 is the start time. Below, we present our model parameters and notations:

Algorithm 1 Joint RideSharing Framework

```
1: Initialize vehicles' states $X_0$ at $t_0$.
2: for $t \in T$ do
3:     Fetch all ride requests at time slot t, $D_t$.
4:     Fetch all vehicles that entered the market in time slot t, $V_{new}$.
5:     Dispatch $V_{new}$ to zones with anticipated high demand - Algorithm 4
6:     Fetch all available vehicles at time slot t, $V_t$.
7:     for each vehicle $V_j \in V_t \ldots$ do
8:         Obtain initial matching $A_j$ using Algorithm 2 in III-A.
9:         for each ride request $r_i \in A_j \ldots$ do
10:            Obtain initial price $P_{init}(r_i)$ using (2).
11:            Perform route planning using Algorithm 3 in III-B.
12:            Obtain $S'_{V_j}[r_i]$ based on cost($V_j, S'_{V_j}[r_i]$).
13:            Update trip time $T_i$ based on $S'_{V_j}[r_i]$ using ETA model.
14:            Calculate final price $P(r_i)$ based on $S'_{V_j}[r_i]$ using (3).
15:            Get customer $i$'s decision $C_d^i$ on $P(r_i)$ using (4) & (5).
16:            if $C_d^i == 1$ then
17:                Update $S_{V_j} \leftarrow S'_{V_j}[r_i]$.
18:            else
19:                Insert $r_i$ to $D_{t+1}$
20:            Update the state vector $s_t$.
21:         Retrieve next stop from $S_{V_j}$.
22:         Head to next stop (whether a pickup or a dropoff).
23:     Fetch all idle vehicles with Idle_duration > 10 minutes, $V_{idle}$.
24:     Dispatch $V_{idle}$ to zones with anticipated high demand - Algorithm 4
25:     Update the state vector $s_t$.
```

1) *Demand:* We denote the number of requests for zone m at time t as $d_{t,m}$. The future pick-up request demand in each zone is predicted through a historical distribution of trips across the zones [39], and is denoted by $D_{t:T} = (\overline{\boldsymbol{d}}_{\boldsymbol{t}}, \ldots, \overline{\boldsymbol{d}}_{\boldsymbol{t+T}})$ from time t_0 to $t + T$.
2) *Vehicle States:* We use $X_t = \{x_{t,1}, x_{t,2}, \ \ldots \ , x_{t,N}\}$ to denote the N vehicles' status at time t. $x_{t,n}$ tracks vehicle n's state variables at time step t. For a given vehicle, we keep track of its: (1) current location/zone V_{loc}, (2) current capacity V_C, (3) type V_T, (4) maximum capacity C^V_{max}, (5) Pick-up time for each passenger, and (6) 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 $V_C < C^V_{max}$.
3) *Supply:* At each time slot t, the supply of vehicles for each zone is projected to future time $\tilde{t}$. $d_{t,\tilde{t},m}$ 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. This information can be ascertained using the ETA prediction for all vehicles. Consequently, we can predict the number of vehicles in each zone, 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.

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).

III. DARM Framework for Matching and Route Planning

NP-Hardness: The ride-sharing assignment problem is proven to be NP-hard in [20] as it is a reduction from the

3-dimensional perfect matching problem (3DM). In 3DM, given a number of requests with source and destination locations and a number of available vehicle locations, the task is to assign vehicles to requests. However, in [20], the authors limit this allocation to only two requests sharing the same vehicle at a time using an approximation algorithm that is 2.5 times the optimal cost. They approach this problem by pairing requests first greedily, and then using bipartite graphs to match each vehicle to one pair of requests, while assigning the maximum number of requests with the minimum total cost. Heuristic approaches for 3 rides per vehicles have been considered in [2]. In our approach, we don't limit matching to only two-three requests; instead, we go as far as the maximum capacity of a vehicle allows (satisfying the capacity constraint), which significantly boosts the acceptance rate of passengers.

Our dynamic, demand-aware, and pricing-based DARM framework goes through two phases:

Algorithm 2 Greedy Assignment

```
1: **Input:** Available Vehicles $V_t$ with their locations $loc(V_j)$ such
   that: $V_j \in V_t$, Ride Requests $D_t$ with origin $o_i$ and destination $d_i$
   associated with $r_i \in D_t$.
2: **Output:** Matching decisions $A_j$ for each $V_j \in V_t$
3: **Initialize** $A_j = [\,]$, $V^j_{\text{capacity}} = V^j_C$ for each $V_j \in V_t$.
4: **for** each $r_i \in D_t \ldots$ **do**
5:     **Obtain** locations of candidate vehicles $V_{\text{cand}}$, such that:
       $|loc(V_j) - o_i| \leq 5\,km^2$ **AND** $(V^j_{\text{capacity}} + |r_i|) \leq C^{V_j}_{max}$.
6:     **Calculate** trip time $T_{j,i} \in T_{\text{cand},i}$ from each $loc(V_j) \in V_{\text{cand}}$
       to $o_i$ using the ETA model.
7:     **Pick** $V_j$ whose $T_{j,i} = \text{argmin}(T_{\text{cand},i})$ to serve ride $r_i$.
8:     **Push** $r_i$ to $A_j$
9:     **Update** $loc(V_j) \leftarrow o_i$
10:    **Increment** $V^j_{\text{capacity}} \leftarrow V^j_{\text{capacity}} + |r_i|$
11: **Return** $A_t = [A_j, A_{j+1}, \ldots, A_n]$, where $n = |V_t|$.
```

A. Initial Vehicle-Passenger(s) Assignment Phase

The initial assignment is represented in Algorithm 2. In this phase, each vehicle having the knowledge of the future demand (fed from the control unit) $D_{t:t+T}$ at each zone, the vehicles' status vectors X_t including their current locations as well as the origin o_i and destination d_i locations for each request r_i, performs a greedy matching operation. This is where each request r_i gets assigned to the nearest available vehicle to it, satisfying the capacity constraints. In other words, we define the capacity constraint to be: the number of all requests assigned to vehicle V_j is less than its maximum capacity $C^{V_j}_{max}$ at any time. At the end of this phase, each vehicle V_j has a list of initial matchings $A_j = [r_1, r_2, \ldots, r_k]$, where $k \leq C^{V_j}_{max}$ assuming that each request has only one passenger. Assume the passenger count per request is $|\ r_i\ |$, and the vehicle V_j arrives at location z. Then, to check the capacity constraint in $O(1)$ time, we define vehicle V_j's current capacity $V^j_C[z]$ that refers to the total capacity of the requests that are still on-board of V_j when it arrives at that location z as follows:

$$V^j_C[z] = \begin{cases} V^j_C[z-1] + |\ r_i\ | & \text{if } z == o_i \\ V^j_C[z-1] - |\ r_i\ | & \text{if } z == d_i \end{cases} \tag{1}$$

B. Distributed Optimization Phase

Our demand-aware route planning problem is a variation of the basic route planning problem (which is NP-hard) for shareable mobility services [5], [40] by setting $\alpha = 1$ and $\beta = 0$. Further, the existing literature proved that there is no optimal method to maximize the total revenue for the basic route planning problem (which is reducible to our DARM problem) using neither deterministic nor randomized algorithms [5], [22]. Thus, the same applies to our DARM problem. However, several studies show that Insertion is an effective approach to greedily deal with the shared mobility problem. We propose an insertion based framework, similar to the idea in [28], to optimize our matching framework. However, in [28], authors group together close-by requests without considering if their destinations are in opposite directions, while our approach mitigate this problem. Also, [28] use denwick tree to speed up their operations while we utilize our OSRM module as will be explained below. Note that, in our framework, this optimization step is impacted by the pricing decisions that are made based on the Q-values learnt from our DQN. In this aspect, our approach can achieve better results in a relatively long time period using the near future predicted demand (that is part of the DQN input) to overcome the short-sightedness problem of the basic insertion algorithms.

In DARM, we follow the idea of searching each route and locally optimally inserting new vertex (or vertices) into a route. In our problem, there are two vertices (i.e., origin o_i and destination d_i) to be inserted for each request r_i. We define the insertion operation as: given a vehicle V_j with the current route S_{V_j}, and a new request r_i, the insertion operation aims to find a new feasible route S'_{V_j} by inserting o_i and d_i into S_{V_j} with the minimum increased cost, that is the minimum extra travel distance, while maintaining the order of vertices in S_{V_j} unchanged in S'_{V_j}. Specifically, for a new request r_i, the basic insertion algorithm checks every possible position to insert the origin and destination locations and return the new route such that the incremental cost is minimized. To present our cost function, we first define our distance metric, where given a graph G we use our OSRM engine to pre-calculate all possible routes over our simulated city. Then, we derive the distances of the trajectories (i.e., paths) from location a to location b to define our graph weights. Thus, we obtain a weighted graph G with realistic distance measures serving as its weights. We extend the weight notation to paths as follows: $w(a_1, a_2, \ldots, a_n) = \sum_{i=1}^{n-1} w(a_i, a_{i+1})$.

Thus, we define the cost associated with each new potential route/path $S'_{V_j} = [r_i, r_{i+1}, \ldots, r_k]$ to be the cost$(V_j, S'_{V_j}) = w(r_i, r_{i+1}, \ldots r_k)$ resulting from this specific ordering of vertices (origin and destination locations of the k requests assigned to vehicle V_j). Besides, we derive the cost of the original route to calculate the increased costs for every new route $S'_{V_j}[r_i]$. To illustrate, assume A_j for vehicle V_j has only two requests r_x and r_y, its location is $loc(V_j)$ and its current route has r_x already inserted as: $[loc(V_j), o_x, d_x]$. Then, V_j picks S'_{V_j} of inserting r_y into its current route, such that: cost$(V_j, S'_{V_j}[r_y]) = \min[w(loc(V_j), o_x, o_y, d_x, d_y)$, $w(loc(V_j), o_y, o_x, d_x, d_y)$, $w(loc(V_j), o_y, o_x, d_y, d_x)$, $w(loc(V_j), o_x, o_y, d_y, d_x)$, $w(loc(V_j), o_x, d_x, o_y, d_y)$, $w(loc(V_j), o_y, d_y, o_x, d_x)]$. Note that the last two optional routes complete one request before serving the other, hence they do not fit into the *ride-sharing* category. However, we still consider them as we optimize for the fleet's overall profits and total travel distance. Also, note that these two routes will still serve both requests and thus would not affect the overall acceptance rate of our algorithm. They may just increase the customers' waiting time a little, however, we show in our results that the customers' waiting time is very reasonable. Note that if this was the first allocation made to this vehicle, then the first request will be just added to its currently empty route. Otherwise, it will be dealt with (like all requests in the list) by following the insertion operation above. Then, the cost of serving all k requests in matching A_j is in turn defined as: $\text{cost}(V_j, A_j) = \sum_{r_i \in M_j} \text{cost}(V_j, S'_{V_j}[r_i])$. Finally, this phase works in a distributed fashion where each vehicle minimizes its travel cost, following Algorithm 3. This distributed procedure goes on a customer-by-customer basis as follows:

- Each vehicle V_j receives its initial matchings list A_j, and an initial price P_{init} (as explained in Section IV-A) associated with each request r_i in that list. This initial matchings list is sorted ascendingly based on proximity to vehicle V_j.
- Then, each vehicle V_j considers each $r_i \in A_j$, in order of proximity, inserting into its current route. For each request r_i, the vehicle arrives at the minimum $\text{cost}(V_j, S'_{V_j}[r_y])$ of insertion into its current route (as described above).
- Now, given the initial price associated with this request P_{init} and the new route $S'_{V_j}[r_y]$ (which may involve detours to serve this request), the vehicle can re-calculate the pricing to account for any extra distance, by feeding the new distance into Eq. (2) in Section IV-A.
- Afterwards, drivers will then modify the pricing based on the Q-values of the driver's dispatch-to locations. Using the DQN dispatch policy on a regular basis, drivers have gained insight about which destinations can yield him/her a higher profit. So, they weigh in on their utility function and propose a new pricing to the customer (explained in Section IV-B).
- Finally, the customer(s) can accept or reject based on his/her utility function as explained in Section IV-C. If a customer accepts, the vehicle updates its route S_{V_j} to be $S'_{V_j}[r_y]$, otherwise S_{V_j} remains unchanged. The vehicle then proceeds to the next customer and repeats the process. Rejected requests will be fed back into the system to be considered in the matching process initiated in the next timestep for other/same vehicles.

The key idea in the proposed matching algorithm is that the pricing of the passenger depends on the distance in the route based on the previous matched passengers. Thus, if the new passenger is going in the opposite direction, the distance will be larger leading to higher price, and we expect that the passenger would likely not accept the high price as compared to another vehicle that is going towards that direction. Of course, if the customer is willing to pay the high price, he/she will be matched. Thus, the increased pricing prioritizes passengers to be matched if their routes have intersections as opposed to if they are going in opposing directions.

Complexity Analysis: The complexity of the insertion operation is discussed in Appendix C. Here we discuss checking the route feasibility in $O(1)$. For a route to be feasible, for each request r_i in this route, o_i has to come before d_i. Therefore,

Algorithm 3 Insertion-Based Route Planning

```
1: Input: Vehicle $V_j$, its current route $S_{V_j}$, a request $r_i = (o_i, d_i)$ and
   weighted graph G with pre-calculated trajectories using OSRM model.
2: Output: Route $S'_{V_j}$ after insertion, with minimum cost($V_j, S'_{V_j}$).
3: if $S_{V_j}$ is empty then
4:     $S'_{V_j} \leftarrow [loc(V_j), o_i, d_i]$.
5:     cost($V_j, S'_{V_j}$) = $w(S'_{V_j})$.
6:     Return $S'_{V_j}$, cost($V_j, S'_{V_j}$)
7: Initialize $S''_{V_j} = S_{V_j}$, $Pos[o_i]$ = NULL, cost$_{min} = +\infty$.
8: for each $x$ in 1 to $|S_{V_j}|$ do
9:     $S^x_{V_j}$ := Insert $o_i$ at $x-th$ in $S_{V_j}$.
10:    Calculate cost($V_j, S^x_{V_j}$) = $w(S^x_{V_j})$.
11:    if cost($V_j, S^x_{V_j}$) < cost$_{min}$ then
12:        cost$_{min} \leftarrow$ cost($V_j, S^x_{V_j}$).
13:        $Pos[o_i] \leftarrow x$, $S''_{V_j} \leftarrow S^x_{V_j}$.
14: $S'_{V_j} = S''_{V_j}$, cost$_{min} = +\infty$.
15: for each $y$ in $Pos[o_i]+1$ to $|S''_{V_j}|$ do
16:    $S^y_{V_j}$ := Insert $d_i$ at $y-th$ in $S''_{V_j}$.
17:    Calculate cost($V_j, S^y_{V_j}$) = $w(S^y_{V_j})$.
18:    if cost($V_j, S^y_{V_j}$) < cost$_{min}$ then
19:        cost$_{min} \leftarrow$ cost($V_j, S^y_{V_j}$).
20:        $S'_{V_j} \leftarrow S^y_{V_j}$, cost($V_j, S'_{V_j}$) $\leftarrow$ cost$_{min}$.
21: Return $S'_{V_j}$, cost($V_j, S'_{V_j}$)
```

to further reduce the computation needed, we first find the optimal position $Pos[o_i]$ to insert o_i and then, find the optimal position $Pos[d_i]$ to insert d_i we only consider positions starting from $Pos[o_i]+1$. Therefore, we never have to check all permutations of positions, we only check n^2 options in the ride-sharing environment as we check the route feasibility in $O(1)$ time. This is further reflected in the capacity constraint defined in phase 1, where we borrow the idea of defining the smallest position to insert origin without violating the capacity constraint from [20] as $Ps[l]$, the capacity constraint of vehicle V_j will be satisfied if and only if: $Ps[l] \leq Pos[o_i]$. Here, we need to guarantee that there does not exist any position $l \in (Pos[o_i], Pos[d_i])$ such that: $V^j_C[l] \geq (| r_{i-1} | - | r_i |)$, other than $Ps[l]$ to satisfy $Ps[l] \leq Pos[o_i]$ and thus abide by the capacity constraint.

IV. Distributed Pricing-Based Ride-Sharing (DPRS)

In this section, we explain our distributed pricing strategy that is built on top of our distributed ride-sharing environment. In our simulator setup, we consider various vehicle types with varying capacity, mileage, price per mile-distance, price per waiting-minute, and base price for driver per trip denoted B_j. B_j serves as the local minimum earning for the driver per trip.

A. Initial Pricing

Initially, each vehicle calculates a price for each of its requests, 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 the weights of the n edges that constitute the vehicle's optimal route from its current location to origin o_i and then to destination d_i. This route is obtained through the insertion operation in Algorithm 3, as the route that minimizes the DARM cost function after inserting request r_i and subtracting the cost of the original route before insertion. Thus, in the optimization step, the new cost($V_j, S'_{V_j[r_i]}$) is plugged into the equation to get the updated pricing.

- 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 the vehicle j's capacity $V^j_C[o_i]$ when it reaches the origin o_i location of this request r_i subtracted from its capacity when it reaches the destination d_i, $V^j_C[d_i]$. Thus, we define $V^j_C[r_i] = \mid V^j_C[d_i] - V^j_C[o_i] \mid$.
- The cost for fuel consumption associated with this trip, denoted by [distance travelled$*(P_{gas}/M_j)$,] where P_{gas} represents the average gas price, and M^j_V 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 pricing equation for request r_i is represented as:

$$P_{init}[r_i] = B_j + \left[\omega^1 * \frac{cost(V_j, S_{V_j[r_i]})}{V^j_C[r_i]}\right] + \left[\omega^2 * \left(\frac{cost(V_j, S_{V_j[r_i]})}{V^j_C[r_i]} * (\frac{P_{gas}}{M^j_V})\right)\right] - \left[\omega^3 * T_i\right] \tag{2}$$

where 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. Our proposed algorithm will first use the initial price according to cost($V_j, S_{V_j[r_i]}$) and notify the vehicle (or driver), who will then modify the pricing based on the updated route cost($V_j, S'_{V_j[r_i]}$) as well as the Q-values of the driver's dispatch-to location (explained in Section IV-B).

B. Vehicles' Proposed Pricing

The core intuition behind assessing the cost/benefit of picking up a passenger is in having knowledge over the supply-demand distribution over the city. This is learnt by each driver through the dispatch policy that they follow once he/she enters the market. This dispatch policy aims to provide 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 Section V). As a result of running such a policy every dispatch cycle (set to 5 minutes in our simulation), the driver gains the necessary insight about how the supply-demand is distributed over the city, and thus can make informed decisions on the pricing strategy that can yield him a higher profit. This decision-making process is captured as follows: With the knowledge of the expected discounted sum of rewards (Q-values) across the map, the vehicle:

- Ranks destinations on the map, in a descending manner, according to the expected discounted sum of rewards, which is obtained using the DQN's Q values. This rank is denoted α.
- Dynamically maintains a list of highest ranked λ regions on the map, denoted as L. This list represents the potential hotspots on the map that are anticipated to maximize the driver's profits, and thus are *desired zones*. It is continuously updated whenever the vehicle runs its dispatch policy.
- After the route planning optimization step, the driver re-calculates the initial pricing $P_{init}(r_i)$ using the updated route S'_{V_j}, by plugging cost($V_j, S'_{V_j[r_i]}$) (after subtracting the cost of the original route before insertion) into Eq. (2). This is done to account for any detours required to serve this new request r_i, and thus enforces requests going in the same direction to be matched together.
- Knowing the request location $loc(r_i)$, if $\in L$, the driver uses the initial pricing suggested for this trip, denoted $P_{init}(r_i)$.
- Otherwise, it would indicate that driver might end up in the middle of nowhere (i.e., region with low demand), and thus receives no more requests or at least drives idle for a long distance. Instead of just rejecting the request, the driver suggests a higher price to the customer to make up for the cost incurred due to mobilizing to a region of low demand. The price increase is influenced by both the rank of the destination as well as the driver's own base price per trip B_j as in Eq. (3).

$$P(r_i) = \begin{cases} P_{init}(r_i) & \text{if } loc(r_i) \in L \\ P_{init}(r_i) + & \\ [P_{init}(r_i) * \frac{\alpha_{loc(r_i)}}{2} * B_j] & \text{otherwise} \end{cases} \tag{3}$$

C. Customers' Decision Function

Upon the vehicle's proposed price it becomes the customer's turn to make his/her own decision according to his/her set of preferences. In our algorithm, we consider various preferences for each customer that constitutes their utility function:

- Tolerance in waiting time: whether the customer is in a hurry and how much delay can be tolerated: denoted as delay/waiting time of trip i: T_i.
- Preference in car-pooling: whether the customer is willing to share this ride or would rather take the ride alone even if it means a higher price. This is captured based on the current capacity of vehicle j, denoted by V^j_C.
- Preference of vehicle type for their trip: whether he/she is willing to pay more in exchange for a more luxurious vehicle. The type of vehicle j is denoted by V^j_T.

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

$$U_i = \left[\omega^4 * \frac{1}{V^j_C}\right] + \left[\omega^5 * \frac{1}{T_i}\right] + \left[\omega^6 * V^j_T\right] \tag{4}$$

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:

$$C_d^i = \begin{cases} 1 & \text{if } U_i > P(r_i) - \delta_i \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

V. Distributed DQN Dispatching Approach

We utilize a distributed DQN dispatch policy to re-balance idle vehicles to areas of predicted high demand and profits over the city, where they can better serve the demand and maximize their profits. We utilize a reinforcement learning framework, with which we can learn the probabilistic dependence between vehicle actions and the reward function thereby optimizing our objective function. Idle vehicles get dispatched when they first enter the market or when they experience large idle times throughout our simulation.

At every time step t, idle vehicles observe the state of the environment, $s_{t,n}$, and perform inference on their trained DQN to predict a future reward r_t associated with each dispatch-to location on their action space $a_{t,n}$ over the map. 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, i.e., $\sum_{j=t}^{\infty} \eta^{j-t} r_j(a_t, s_t)$, where $\eta < 1$ is a time discount factor. This ultimately improves the fleet utilization. The overall flow of this framework is explained in Algo. 4. Lines $6-8$ of the algorithm specifically describe the best action that a given vehicle V_j infers from the trained Q-network given the state $s_{t,n}$ and set of possible actions $a_{t,n}$.

In our algorithm, we define the reward r_t as a weighted sum of different performance components that reflect the objectives of our DQN agent. 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. The reward will be learnt from the environment for individual vehicles and then leveraged to optimize their decisions. Thus, the overall system objective is optimized at each vehicle in the distributed transportation network. Below, we explain the state, action, and reward for our dispatch policy:

State Space: The state variables are utilized to reflect the environment status and thus influence the reward feedback to the agents' actions. We combine all the environment data explained in Section II-B: (1) (X_t: that keeps track of vehicles states: *current zone of vehicle v, available seats, pick-up time, destination zone of each passenger.* (2)$V_{t:t+T}$: Supply prediction of the number of available vehicles at each zone for T time slots ahead, (3)$D_{t:t+T}$: Demand prediction at each zone for T time slots ahead. Thus, the state space at time t is captured by three tuples combined in one vector as $s_t = (X_t, V_{t:t+T}, D_{t:t+T})$. At each vehicle-request assignment, the simulator engine updates the state space tuple with the expected pick-up time, source, and destination data. The three-tuple state variables s_t are passed as an input to the DQN input layer which consequently outputs the best action to be taken.

Action Space: a_t^n denotes the action taken by vehicle n at time step t. In our simulator, the vehicle can move (vertically or horizontally) at most 7 cells, it can move to any of the 14 vertical (7 up and 7 down) and 14 horizontal (7 left and 7 right) cells and hence the action space is limited to these cells. This results in a 15×15 action space $a_{t,n}$ for each vehicle as a vehicle can move to any of these cells or remain in its own cell. After the vehicle decides on which cell to go to using DQN, it uses the shortest optimal route to reach its next stop.

Reward: The reward function (in Eq. (6)) is a weighted sum of the following terms: (1) $C_{t,n}$: number of customers served by vehicle n at time t, (2) dispatch time, $T_{t,n}^D$, taken by vehicle n at time t to go to zone m or take detours to pick up extra requests. This term discourages the agent from picking up additional orders without considering the delay for on-board passengers. (3) $T_{t,n}^E$ denotes the sum of additional time vehicle n takes at time t to serve additional passengers, (4) profit for vehicle n at time t: $\mathbb{P}_{t,n}$, and (5) $\max(e_{t,n} - e_{t-1,n}, 0)$ this term addresses the objective of minimizing the number of vehicles at time t to improve vehicle utilization.

$$r_{t,n} = \beta_1 C_{t,n} + \beta_2 T_{t,n}^D + \beta_3 T_{t,n}^E + \beta_4 \mathbb{P}_{t,n} + \beta_5[\max(e_{t,n} - e_{t-1,n}, 0)] \tag{6}$$

We define the derivation of each component of our reward function in Appendix D-B. 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. The details of learning these Q-values associated with the action space is provided in Appendix D-C, and the architecture of our Deep Q-Network is presented in Appendix D-A.

While the primary role of the DQN is to act as a means of dispatching idle vehicles, it contains useful signals on future anticipated demand that is utilized by other components of our method including DARM matching and DPRS Pricing. Note that the profits term added to 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 the supply-demand distribution over the city which is essential in making knowledgeable decisions when it comes to ranking their potential hotspots, and thus making the corresponding route planning and pricing decisions (Section III and Section IV-B, respectively).

Algorithm 4 Dispatching Using DQN

```
1: Input: $X_t, V_{t:t+T}, D_{t:t+T}$.
2: Output: Dispatch Decisions.
3: Fetch all idle vehicles $\leftarrow V_{Idle}$.
4: for each vehicle $V_j \in V_{Idle}$ do
5:     Construct a state vector $s_{(t,n)} = (X_t, V_{t:t+T}, D_{t:t+T})$.
6:     Push state vector to the Deep Q-Network.
7:     Get the best dispatch action $a_{(t,j)} = argmax[Q(s_{(t,n)}, a, \theta)]$.
8:     Get the destination zone $Z_{(t,j)}$ based on action $a_{(t,j)}$.
9:     Update dispatch decisions by adding $(j, Z_{(t,j)})$.
10: Return Dispatch Locations $\forall\ V_j \in V_{Idle}$
```

VI. Experimental Results

A. Simulator Setup

In our simulator, we used the road network of the New York Metropolitan area along with a real public dataset of taxi trips in NY [38]. For each trip, we obtain the pick-up time, passenger count, origin location, and drop-off location. We use this trip information to construct travel requests demand prediction model as well. 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 accompanying 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. Note that, not all vehicles are populated at once, they are deployed incrementally into the market by each time step t. We also defined a reject radius threshold for a customer request. Specifically, if there is no vehicle within a radius of 5km to serve a request, it is rejected. This simulator hosts each deep reinforcement learning agent which acts as a ridesharing vehicle that aims to maximize its reward: Eq. (6).

B. DQN Training and Testing

The fleet of autonomous vehicles was trained in a virtual environment that simulates urban traffic. We consider the data of June 2016 for training and one week from July 2016 for evaluations. For each experiment, we trained our DQN neural networks using the data from the month of June 2016 for 20k epochs, which corresponds to a total of 14 days, and used the most recent 5000 experiences as a replay memory. In Appendix D-C, Fig. 6 shows the convergence of average Q-max during training. Upon saving Q-network weights, we retrieve the weights to run testing on an additional 8 days from the month of July which corresponds to 10k epochs. Thus, $T = 8 \times 24 \times 60$ steps, where $\Delta t = 1$ minute. Also, we use *Python* and *Tensorflow* to implement our framework. Each vehicle has a maximum working time of 21 hours per day, after which it exits the market. To initialize the environment, we run the simulation for 20 minutes without dispatching the vehicles. Finally, 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 $\omega^6 = 4$.

C. Performance Metrics

We breakdown the reward and utility functions, and investigate the performance for various baselines. Recall that we want to minimize the components of our reward in Eq. (6).

- **Accept rate:** we note that the supply-demand mismatch is reflected in our simulation through this metric. Accept rate is defined as the ratio of successful pick-ups by the fleet to the total number of requests made in a given time slot. A high accept rate is a characteristic of a reliable mode of transportation. With a high acceptance rate, our fleet is able to fulfill the passengers' transportation demands.
- **Cruising (idle) time:** this metric represents the time at which a vehicle is neither occupied nor gaining profit but still incurring gasoline cost.
- **Occupancy Rate:** this metric captures the utilization rate of the fleet of vehicles, it keeps track of how many vehicles are deployed from the fleet to serve the demand. By minimizing the number of occupied vehicles, we achieve better utilization of individual vehicles in serving the demand. Given that (i) all baselines are catering to a similar volume of pickup orders, and (ii) if all baselines are achieving a similar accept rate, a lower occupancy rate indicates that a fleet is able to minimize the number of vehicles on the street to serve the requests. Note that, in Fig. 3, we also show the utilization of each individual vehicle (i.e., percentage of time the vehicle is occupied while in duty).
- **Waiting Time:** this metric captures the time customers had to wait till they get picked up by a vehicle (shown in seconds). We note that wait time is an important metric for customer convenience with mobility-on-demand services.
- **Profit:** this metric represents the net profit of each driver per hour of service, where the cost incurred by fuel consumption is subtracted from the revenue. Note that, the Q-values depend on the pricing since the decisions made by customers and drivers impact the reward function through the profit term. With high net profits, our framework is able to find a common ground that is profitable to both parties.
- **Travel Distance:** this metric shows the number of kilometers traveled by each vehicle per hour of service, which gives a good reflection of the cost incurred by vehicles due to serving multiple ride requests.

D. Baselines

We compare our proposed framework (with dispatching, ride-sharing, our novel DPRS pricing strategy, and DARM approach for matching and route planning) against the following baselines to emphasize the distinct impact of each component:

- No Dispatch, No Ride-sharing, No Pricing Strategy, Greedy Matching (!D, !RS, !PS, GM): In this setting, vehicles don't get dispatched to areas with anticipated high demand, no matter how long they stay idle. Ride-sharing (pooling) is not allowed, every vehicle serves only one request at a time. Also, initial pricing is used and is accepted by both drivers and customers by default. For matching, only the greedy initial matching is applied, no optimization takes place.
- No Dispatch with Ride-sharing but No Pricing Strategy, and with Greedy Matching (!D, RS, !PS, GM): similar to (!D, !RS, !PS) except that ride-sharing (pooling) is allowed, where vehicles can serve more than one request altogether.
- Dispatch with No Ride-sharing and No Pricing Strategy with Greedy Matching (D, !RS, !PS, GM): Here, vehicles are dispatched when idle but, ridesharing isn't allowed as [32].
- Dispatch with Ridesharing but No Pricing Strategy, and with Greedy Matching (D, RS, !PS, GM): similar to (D, !RS, !PS, GM), but with ride-sharing allowed, as in DeepPool [29].
- Dispatch with Ridesharing and Pricing Strategy, but with Greedy Matching (D, RS, PS, GM): similar to (D, RS, !PS, GM), but with applying our DPRS pricing strategy where customers and drivers are involved in the decision-making process. However, only greedy matching is adopted here.

The proposed baselines aim to evaluate the effectiveness of each component of our framework. Our proposed joint method incorporates both insertion-based route-planning and pricing strategy to involve both customers and drivers in the decision-making. As compared to the above baselines that don't adopt DARM, we hypothesize that our (DARM+DPRS) would be

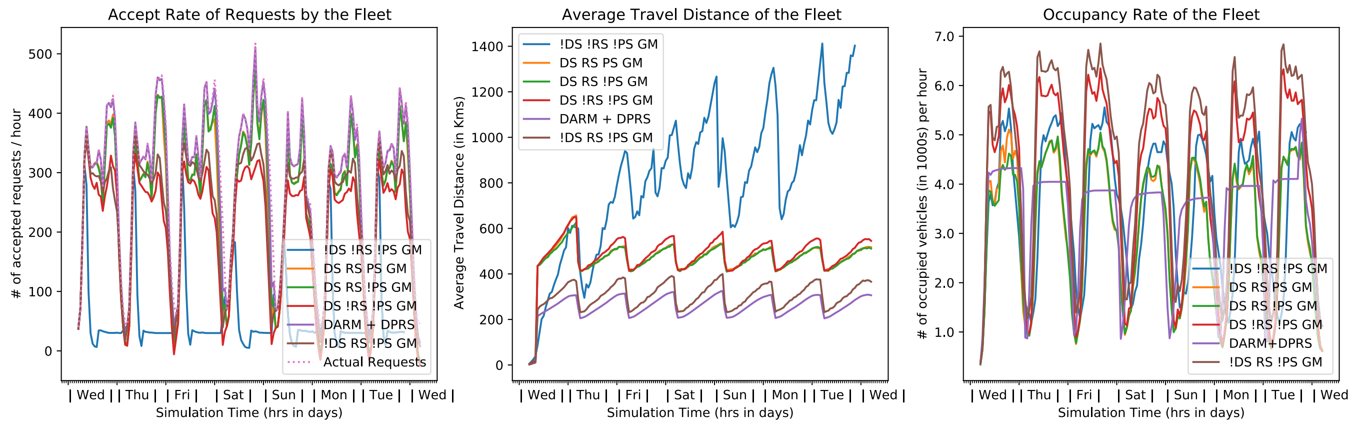


Fig. 2. Performance metrics of the proposed algorithm and the baselines.[3]

a more effective approach. Given that the core intuition of DARM is to group together rides that share route intersections to their destinations as opposed to rides heading to opposite-direction-destinations, we expect improvements in the number of rides served, profits, travel distance, and occupancy rate.

Moreover, we have included baselines that do not consider DPRS, where Pricing Strategy is not adopted (!PS) to observe the effectiveness of our pricing framework. In these scenarios, involving drivers and customers in the decision-making process has the potential to burden the system causing an increase in the rejection rate and the number of vehicles utilized to serve the demand. However, we hypothesize that our DPRS together with our dispatch policy will be able to establish the balance and reach solutions that are profitable to both passengers and drivers. To validate our hypothesis, we investigate both the net profits of drivers as well as the passengers' waiting times. In addition, as compared to DPRS only baseline (D, RS, PS, GM), our joint framework (DARM + DPRS) is expected to further improve the fleet utilization, idle time and overall travel distance. In addition, we also include baselines that don't adopt dispatching policy where vehicles only mobilize according to the pickup locations of their requests as opposed to learning the supply-demand distribution of the city and mobilizing accordingly when they experience idle time. Comparing against this baseline shows the impact of our dispatch policy on improving profits, fleet utilization, accept rate, travel distance, and waiting times.

Finally, since [29] showed that (D, RS, PS, GM) performs better than the dispatch with minimum distance (DS-mRS) approach [20] and the Centralized Receding Horizon Control (cRHC) approach [41] in terms of idle times, waiting times and fleet utilization, we do not consider these baselines. In (DS-mRS), ride-sharing is allowed where two riders are assigned to one vehicle so that the total driving distance is minimized. However, in cRHC, the dispatch actions are taken to maximize the expected reward in a centralized manner as opposed to our distributed approach.

We include baselines that don't adopt ridesharing to benchmark how much of the improvement in the aforementioned performance metrics is attributed to ride pooling instead of serving one ride at a time, as opposed to the improvement due to the deployment of the other components of our framework (Dispatching Policy, DPRS and DARM). Note that DARM works in tandem with DPRS, as the pricing decisions impact the route planning procedure (i.e. pricing-based matching) and thus the improvement of profits, waiting times, idle times as well as travel distance is a result of deploying both approaches.

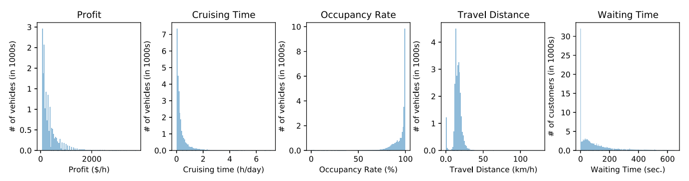
E. Results Discussion

We validate the necessity of each component of our framework using computational results. From our simulation, we observe that the hypothesis for each baseline comparison has been supported for the most part by our experimental results. In Fig. 2, we investigate the overall performance of our proposed framework in comparison to all other baselines. We show the actual number of requests as the dotted pink line. We observe that our joint DARM + DPRS framework ranks highest in the acceptance rate per hour (almost coincide with the actual requests curve) of $> 96\%$, followed by all the ridesharing-based baselines (at around 80%), while the non-ridesharing and the non-dispatching baselines come at the bottom of the list (with $< 55\%$). Clearly, our joint (DARM+DPRS) approach boosts the acceptance rate with an increase of $> 15\%$ when compared to DPRS only [(D RS PS GM) baseline] and DeepPool [(D RS !PS GM) baseline] and $> 40\%$ when compared to non-dispatching and/or non-ridesharing baselines. This proves our hypothesis that this improvement of 40% is achieved by our joint (DARM + DPRS), while 15% is fully attributed to our DARM approach (which makes use of DPRS in its optimization phase).

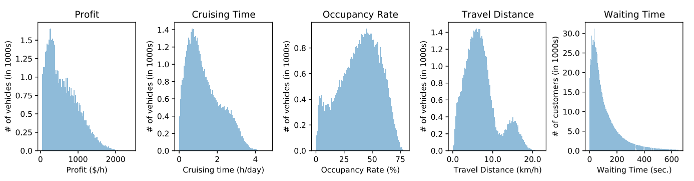
Contradictory to the expectation that involving drivers and customers in the decision-making process would increase the rejection rate and the number of vehicles utilized to serve the demand, our joint (DARM+DPRS) has significantly low rejection rate. To further analyze the rejection rate due to DPRS, we observe that the rejection rate made by 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.

It is worth noting that the proposed method is able to deliver more requests while minimizing the number of occupied vehicles. Fig. 2 shows a utilization/occupancy rate of around two-thirds of that of the rest of the baselines, saving one-third

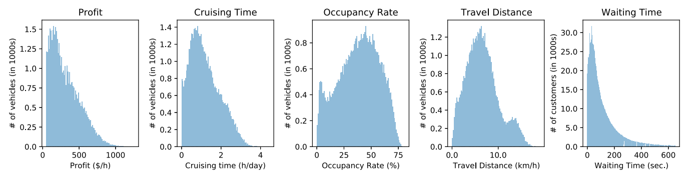
[3]Enlarged Figure is provided in Fig. 8 in Appendix D.



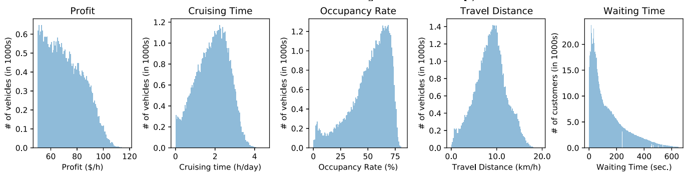
(a) Performance Metrics of our Joint Framework "DARM + DPRS"



(b) Performance Metrics of (D, RS, PS, GM) Baseline "DPRS with Greedy Matching"



(c) Performance Metrics of (D, RS, !PS, GM) Baseline "Deep_Pool in [29]"



(d) Performance Metrics of (D, !RS, !PS, GM) Baseline "Dynamic Taxi Fleet Management in [32]"

Fig. 3. Histograms of performance metrics for the proposed algorithm and the baselines.[4]

of the vehicles for serving new incoming requests which would -in turn- further increase the acceptance rate. Also, we had set the maximum number of vehicles in our simulator to 8000, our approach utilizes only half of them (4000) to serve the demand with an acceptance rate $>$ 96% With the DPRS only [(D RS PS GM) baseline] and DeepPool [(D RS !PS GM) baseline], this increases to just below 5000 vehicles. Again, an improvement of 1000 vehicles, only attributed to adopting our DARM approach. Without our dispatching policy, the number of occupied vehicles reaches more than 6000 that only covered 60% of the demand. While, as expected, without ridesharing, more than 7500 vehicles are utilized to serve only 60% of the demand. Since our DARM + DPRS performs significantly better than DeepPool in both metrics; this makes DARM + DPRS superior to DeepPool, DS-mRS, and cRHC baselines.

We can further support our hypothesis by looking at the average travel distance of the fleet, we can see it is exploding (around 1300km) with the non-dispatching non-ridesharing baseline, where the vehicle only serves one request at a time and is at the risk of encountering large cruising time while looking for a ride, without taking the right dispatch action to a zone where new requests can be found. Just by adding ridesharing, travel distance dramatically decreases to around 500km [(!DS RS !PS GM) baseline] saving $>$ 60%, but in this case, the accept rate is only 55%. However, when adopting our dispatch policy [(DS RS !PS GM) baseline], travel distance reaches 700km saving $>$ 50%, while serving 80% of the demand. Finally, with joint (DARM+DPRS), the overall travel distance falls at 300km, saving $>$ 80% while serving $\approx$ 96% of the demand. Therefore, a 30% decrease in the overall travel distance is attributed to our DARM+DPRS framework.

Clearly, non-dispatching and non-ridesharing algorithms are shown to have poor utilization of resources, as they use a higher number of vehicles to serve the less amount of demand. Therefore, we exclude them due to their poor performance in Fig. 2., and we take a closer look at the ride-sharing based baselines that adopt a dispatch policy. Figure 3 shows that the average profits for the drivers have significantly increased over time as compared to the other three 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 make up for any extra encountered cost. This implies both the drivers and customers are achieving a compromise that is profitable and convenient to them. Specifically, the profits for our DARM + DPRS framework is almost double that of DPRS only [(D RS PS GM) baseline], and $3-4$ times that of DeepPool [(D RS !PS GM) baseline]. Without ride-sharing, profits are 10 times less than the average profit with our joint framework.

We will further show that our framework not only enhances the overall fleet utilization, but also the utilization of each individual vehicle (i.e., percentage of time the vehicle is occupied). Similarly, for the travel distance metric. Fig. 3a shows that our joint framework minimizes the cruising time, where vehicles are idle, and thus minimizes the extra travel distance as well as extra gasoline cost. On average, vehicles' idle time is within a minimal range $(1-2)$ hours for DARM + DPRS framework. Knowing that vehicles' working time is at most 21 hours per day, we observe that $>$ 85% of the allowed vehicles per day, experience idle time $<$ 2 hours, which is $<$ 10% of their total working time. This metric is almost doubled for all other three baselines. This proves that most of the improvement in fleet utilization is due to our DARM framework. This is also reflected in the occupancy rate metric, which is defined as the percentage of time where vehicles are occupied while on duty. Fig. 3a shows that $6k-8k$ vehicles are between $80\%-100\%$ occupied, and hence proves that our framework significantly improves the utilization of each individual vehicle as well as the whole fleet.

However, Fig. 3 shows that the average travel distance of DARM + DPRS is slightly higher, ranging between $10-30$ km per hour, compared to $5-25$ km for the other baselines. Most of the 4000 vehicles deployed by (DARM+DPRS) fall within $10-20$ km which is very reasonable given the $15-40\%$ extra demand that they serve. Since DARM + DPRS provides significantly higher profits for drivers, it comes at the cost of a slight increase in travel distance, which is an advantage of DARM + DPRS. Note that, the two policies with the lowest travel distance in Fig 3c, and 3d are not involving vehicles or customers in the decision-making process which explains why vehicles have

[4]Enlarged figure is provided in Fig. 9 in Appendix D.

lower travel distances on average. However, we can observe that the non-ridesharing protocols are not efficient as they result in lower profit margins and higher customers' waiting time. Moreover, we emphasize that non-dispatching protocols yield higher idle time for the vehicles as they might spend a large amount of time being idle and they never get dispatched to higher demand areas. In contrast, non-ride-sharing protocols yield lower idle time, but they are still inefficient as vehicles spend more time on duty while serving a lower number of customers than the ride-sharing protocols.

Compared to both DeepPool [(D RS !PS GM) baseline] and DPRS only [(D RS PS GM) baseline], the waiting time per request is significantly lower for DARM + DPRS approach. As shown in Fig. 3, the waiting time for customers reduces overtime to < 1 minute. On average, the response time (time till customer is picked up) of our framework is < 200 sec (≈ 3 minutes), which is almost half that of the other three policies. Note that, the framework in Figure 3d is a non-ridesharing framework which should have had a lower response time as the vehicles directly head to the customer to be served without having to pickup other customers on the way. However, our framework shows a similar response time, with a majority of customers experiencing waiting time ≈ 1 minute.

VII. Conclusion

In this paper, we detailed two novel approaches—Demand-Aware and Pricing-based Matching and route planning (DARM) framework and Distributed Pricing approach for Ride-Sharing with pooling (DPRS)—that generate ideal routes on-the-fly and involve both customers as well as drivers in the decision-making processes of ride acceptance/rejection and pricing. Agents' decision-making process is informed by utility functions that aim to achieve the maximum profit for both drivers and customers. The utility functions also account for the fuel costs, waiting time, and passenger's spending power to compute the reward. These novel DARM and DPRS methodologies are also integrated via a Deep Q-network (DQN) based dispatch algorithm where the profits influence the dispatch and the Q-values impact the pricing and thus matching. Contradictory to an expected high rejection rate when agents are given a choice to reject rides, experimental results show that the rejection rate is significantly low for (DARM + DPRS) framework. Given the maximum number of vehicles (8000 vehicles) populated in the simulation, our framework only uses 50% of the vehicles to accept and serve the demand of up to 96% of the requests. When compared with no ride-sharing baselines, our framework provides 10 times more profits. Experiments also show that vehicle idle time (cruising without passengers) is reduced to under two hours and 80% - 100% of the vehicles are occupied all the time. Our model-free DARM + DPRS framework can be extended to large-scale ride-sharing protocols due to distributed decision making for different vehicles reducing the decision space significantly.

Extension of this work to consider travel-time uncertainty where riders can change their ride information on-the-fly [17], and to compute global plans for a flexible ridesharing with different objectives [42] is left as future work. Additional future directions of extending this work are: including capabilities of a joint delivery system for passengers and goods as in [43] (considered in part in [44]), or using multi-hop routing of passengers as in [45] and transit services as in [33] for efficient fleet utilization.

References

[1] M. Haliem, G. Mani, V. Aggarwal, and B. Bhargava, "A distributed model-free ride-sharing algorithm with pricing using deep reinforcement learning," in *Proc. Comput. Sci. Cars Symp.*, Dec. 2020, pp. 1–10.

[2] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," *Proc. Nat. Acad. Sci. USA*, vol. 114, no. 3, pp. 462–467, Jan. 2017.

[3] M. Iqbal. (2020). Uber revenue and usage statistics. BusinessOfApps. [Online]. Available: https://www.businessofapps.com/data/uber-statistics/

[4] J. Lazarus *et al.*, "Shared automated mobility and public transport," in *Road Vehicle Automation 4*. Springer, 2018, pp. 141–161.

[5] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu, "A unified approach to route planning for shared mobility," *Proc. VLDB Endowment*, vol. 11, no. 11, p. 1633, 2018.

[6] U. Ritzinger, J. Puchinger, and R. F. Hartl, "Dynamic programming based metaheuristics for the dial-a-ride problem," *Ann. Oper. Res.*, vol. 236, no. 2, pp. 341–358, 2016.

[7] X. Zhan, X. Qian, and S. V. Ukkusuri, "A graph-based approach to measuring the efficiency of an urban taxi service system," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 9, pp. 2479–2489, Sep. 2016.

[8] S. C. Ho, W. Szeto, Y.-H. Kuo, J. M. Y. Leung, M. Petering, and T. W. Tou, "A survey of dial-a-ride problems: Literature review and recent developments," *Transp. Res. B, Methodol.*, vol. 111, pp. 395–421, May 2018.

[9] N. Agatz, A. Erera, M. Savelsbergh, and X. Wang, "Optimization for dynamic ride-sharing: A review," *Eur. J. Oper. Res.*, vol. 223, no. 2, pp. 295–303, 2012.

[10] P. Santi, G. Resta, M. Szell, S. Sobolevsky, S. H. Strogatz, and C. Ratti, "Quantifying the benefits of vehicle pooling with shareability networks," *Proc. Nat. Acad. Sci. USA*, vol. 111, no. 37, pp. 13290–13294, 2014.

[11] A. Tafreshian and N. Masoud, "Trip-based graph partitioning in dynamic ridesharing," *Transp. Res. C, Emerg. Technol.*, vol. 114, pp. 532–553, May 2020.

[12] X. Wang, N. Agatz, and A. Erera, "Stable matching for dynamic ride-sharing systems," *Transp. Sci.*, vol. 52, no. 4, pp. 850–867, Aug. 2018.

[13] N. A. Agatz, A. L. Erera, M. W. Savelsbergh, and X. Wang, "Dynamic ride-sharing: A simulation study in metro Atlanta," *Transp. Res. B, Methodol.*, vol. 45, no. 9, pp. 1450–1464, 2011.

[14] X. Liang, G. H. D. A. Correia, K. An, and B. van Arem, "Automated taxis' dial-a-ride problem with ride-sharing considering congestion-based dynamic travel times," *Transp. Res. C, Emerg. Technol.*, vol. 112, pp. 260–281, Mar. 2020.

[15] A. Kleiner, B. Nebel, and V. A. Ziparo, "A mechanism for dynamic ride sharing based on parallel auctions," in *Proc. IJCAI*, 2011, pp. 266–272.

[16] C. Zhang, J. Xie, F. Wu, X. Gao, and G. Chen, "Pricing and allocation algorithm designs in dynamic ridesharing system," *Theor. Comput. Sci.*, vol. 803, pp. 94–104, Jan. 2020.

[17] Y. Li and S. H. Chung, "Ride-sharing under travel time uncertainty: Robust optimization and clustering approaches," *Comput. Ind. Eng.*, vol. 149, Nov. 2020, Art. no. 106601.

[18] Y. Hu, Y. Yao, and W. S. Lee, "A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs," *Knowl.-Based Syst.*, vol. 204, Sep. 2020, Art. no. 106244.

[19] Z. Liu, Z. Gong, J. Li, and K. Wu, "Mobility-aware dynamic taxi ridesharing," in *Proc. IEEE 36th Int. Conf. Data Eng. (ICDE)*, Apr. 2020, pp. 961–972.

[20] X. Bei and S. Zhang, "Algorithms for trip-vehicle assignment in ride-sharing," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 1–7.

[21] J. Wang *et al.*, "Demand-aware route planning for shared mobility services," *Proc. VLDB Endowment*, vol. 13, no. 7, pp. 979–991, Mar. 2020.

[22] M. Asghari, D. Deng, C. Shahabi, U. Demiryurek, and Y. Li, "Price-aware real-time ride-sharing at scale: An auction-based approach," in *Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, Oct. 2016, pp. 1–10.

[23] P. Cheng, H. Xin, and L. Chen, "Utility-aware ridesharing on road networks," in *Proc. ACM Int. Conf. Manage. Data*, May 2017, pp. 1197–1210.

[24] B. Cici, A. Markopoulou, and N. Laoutaris, "Designing an on-line ride-sharing system," in *Proc. 23rd SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, Nov. 2015, pp. 1–4.

[25] M. Ota, H. Vo, C. Silva, and J. Freire, "STaRS: Simulating taxi ride sharing at scale," *IEEE Trans. Big Data*, vol. 3, no. 3, pp. 349–361, Sep. 2017.

[26] R. S. Thangaraj, K. Mukherjee, G. Raravi, A. Metrewar, N. Annamaneni, and K. Chattopadhyay, "Xhare-a-ride: A search optimized dynamic ride sharing system with approximation guarantee," in *Proc. IEEE 33rd Int. Conf. Data Eng. (ICDE)*, Apr. 2017, pp. 1117–1128.

[27] H. Wang and H. Yang, "Ridesourcing systems: A framework and review," *Transp. Res. B, Methodol.*, vol. 129, pp. 122–155, Nov. 2019.

[28] Y. Xu, Y. Tong, Y. Shi, Q. Tao, K. Xu, and W. Li, "An efficient insertion operator in dynamic ridesharing services," *IEEE Trans. Knowl. Data Eng.*, early access, Apr. 8, 2020, doi: 10.1109/ICDE.2019.00095.

[29] A. O. Al-Abbasi, A. Ghosh, and V. Aggarwal, "DeepPool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 12, pp. 4714–4727, Dec. 2019.

[30] R. Zhang and M. Pavone, "Control of robotic mobility-on-demand systems: A queueing-theoretical perspective," *Int. J. Robot. Res.*, vol. 35, nos. 1–3, pp. 186–203, Jan. 2016.

[31] S. Ma, Y. Zheng, and O. Wolfson, "Real-time city-scale taxi ridesharing," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 7, pp. 1782–1795, Jul. 2015.

[32] T. Oda and C. Joe-Wong, "MOVI: A model-free approach to dynamic fleet management," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 2708–2716.

[33] T.-Y. Ma, S. Rasulkhani, J. Y. J. Chow, and S. Klein, "A dynamic ridesharing dispatch and idle vehicle repositioning strategy with integrated transit transfers," *Transp. Res. E, Logistics Transp. Rev.*, vol. 128, pp. 417–442, Aug. 2019.

[34] J. Jung, R. Jayakrishnan, and J. Y. Park, "Dynamic shared-taxi dispatch algorithm with hybrid-simulated annealing," *Comput.-Aided Civil Infrastruct. Eng.*, vol. 31, no. 4, pp. 275–291, 2016.

[35] N. Alisoltani, M. Zargayouna, and L. Leclercq, "A multi-agent system for real-time ride sharing in congested networks," in *Agents and Multi-Agent Systems: Technologies and Applications*. Singapore: Springer, 2020.

[36] M. Ramezani and M. Nourinejad, "Dynamic modeling and control of taxi services in large-scale urban networks: A macroscopic approach," *Transp. Res. C, Emerg. Technol.*, vol. 23, pp. 41–60, Jan. 2017.

[37] M. M. Vazifeh, P. Santi, G. Resta, S. H. Strogatz, and C. Ratti, "Addressing the minimum fleet problem in on-demand urban mobility," *Nature*, vol. 557, no. 7706, pp. 534–538, May 2018.

[38] NYC. (2019). *NYC Taxi and Limousine Commission-Trip Record Data*. [Online]. Available: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[39] D. C. Wyld, M. A. Jones, and J. W. Totten, "Where is my suitcase? RFID and airline customer service," *Marketing Intell. Planning*, vol. 23, no. 4, pp. 382–394, 2005.

[40] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *Proc. IEEE 29th Int. Conf. Data Eng. (ICDE)*, Apr. 2013, pp. 410–421.

[41] F. Miao *et al.*, "Taxi dispatch with real-time sensing data in metropolitan areas: A receding horizon control approach," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 2, pp. 463–478, Apr. 2016.

[42] V. Armant and K. N. Brown, "Fast optimised ridesharing: Objectives, reformulations and driver flexibility," *Expert Syst. Appl.*, vol. 141, Mar. 2020, Art. no. 112914.

[43] K. Manchella, A. K. Umrawal, and V. Aggarwal, "FlexPool: A distributed model-free deep reinforcement learning algorithm for joint passengers and goods transportation," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 4, pp. 2035–2047, Apr. 2021.

[44] K. Manchella, M. Haliem, V. Aggarwal, and B. Bhargava, "PassGoodPool: Joint passengers and goods fleet management with reinforcement learning aided pricing, matching, and route planning," 2020, *arXiv:2011.08999*. [Online]. Available: http://arxiv.org/abs/2011.08999

[45] A. Singh, A. Alabbasi, and V. Aggarwal, "A distributed model-free algorithm for multi-hop ride-sharing using deep reinforcement learning," *IEEE Trans. Intell. Transp. Syst.*, early access, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9477304, doi: 10.1109/TITS.2021.3083740.

[46] (2018). *OpenStreetMaps*. [Online]. Available: https://www.openstreetmap.org

Marina Haliem received the B.Sc. and M.Sc. degrees in computer science from The British University in Egypt, Cairo, Egypt (validated from Loghbourough University, U.K.), in 2015 and 2018, respectively. She is currently pursuing the Ph.D. degree with Purdue University, IN, USA. Her research interests are in the areas of machine learning, AI, deep reinforcement learning, and the Internet of Things.

Ganapathy Mani received the B.E. degree in electrical and computer engineering from Anna University, India, in 2008, the M.S. degree in computer science from The George Washington University in 2012, and the Ph.D. degree from Purdue University in 2020. He is currently a Senior Engineer with Qualcomm, CA, USA. His research interests are in the areas of machine learning, deep learning, and cybersecurity.

Vaneet Aggarwal (Senior Member, IEEE) received the B.Tech. degree in electrical engineering from IIT Kanpur, India, in 2005, and the M.A. and Ph.D. degrees in electrical engineering from Princeton University, NJ, USA, in 2007 and 2010, respectively. Since January 2015, he has been with Purdue University, West Lafayette, IN, USA, where he is currently an Associate Professor. He was a Senior Member of Technical Staff Research at AT&T Labs-Research, NJ (2010–2014), Adjunct Assistant Professor at Columbia University, NY (2013–2014), and VAJRA Adjunct Professor at IISc Bangalore (2018–2019). He received Princeton University's Porter Ogden Jacobus Honorific Fellowship in 2009, the 2017 IEEE Jack Neubauer Memorial Award, and the 2018 Infocom Workshop HotPOST Best Paper Award. His current research interests are in communications and networking, cloud computing, and machine learning.

Bharat Bhargava (Life Fellow, IEEE) received the B.S. degree in mathematics from Punjab University in 1966, the B.E. degree in electrical and computer engineering from the Indian Institute of Science in 1969, and the Ph.D. degree in electrical engineering from Purdue University in 1974. He is currently a Professor with the Department of Computer Science, Purdue University. His recent work is on intelligent autonomous systems, security, data analytics, and machine learning.