

Demand-Responsive Zone Generation for Real-Time Vehicle Rebalancing in Ride-Sharing Fleets

Alberto Castagna¹ and Maxime Guériau¹ and Giuseppe Vizzari² and Ivana Dusparic¹

Abstract. Enabling Ride-sharing (RS) in existing Mobility-on-demand (MoD) systems allows to reduce the operating vehicle fleet size while achieving a similar level of service. This however requires an efficient vehicle to multiple requests assignment, which is the focus of most RS-related research, and an adaptive fleet rebalancing strategy, which counter-acts the uneven geographical spread of demand and relocates unoccupied vehicles to the areas of higher demand. Existing research into rebalancing generally divides the system coverage area into predefined geographical zones, however, this is done statically at design-time and can limit their adaptivity to evolving demand patterns. To enable dynamic, and therefore more accurate rebalancing, this paper proposes a Dynamic Demand-Responsive Rebalancer (D2R2) for RS systems. D2R2 uses Expectation-Maximization (EM) clustering to determine relocation zones at runtime. D2R2 re-calculates zones at each decision step and assigns them relative probabilities based on current demand. We demonstrate the use of D2R2 by integrating it with a Deep Reinforcement Learning multi-agent RS-enabled MoD system in a fleet of 200 vehicle agents serving 10,000 trips extracted from New York taxi trip data. Results show a more fair workload division across the fleet without loss of performance with respect to waiting time and distribution of passengers per vehicle, when compared to baselines with no rebalancing and static pre-defined equiprobable zones.

1 INTRODUCTION

Mobility-on-Demand (MoD) systems are gaining popularity over privately owned vehicles and public transportation due to reduced prices and shorter overall journey times [7]. Recent work suggests that RS-enabled MoD systems can achieve similar level of service using fewer vehicles, by better optimizing: (i) vehicle to multiple requests assignment [2] and (ii) rebalancing empty vehicles to fit real-time demand [24].

Vehicle to request matching has been widely investigated. Offline methods relying on constraint solving [3, 4] can design an optimized plan that is then executed by the vehicle. Online methods involve matching to requests dynamically and has so far been addressed using constraint solving methods [2] and agent-based models [1, 8, 9, 25].

However, fleet rebalancing in MoD systems has been less investigated while shown to have a strong impact on level of service of RS-enabled systems [24]. As depicted in Figure 1 created by extracting requests from New York Taxi data [21] on two different periods,

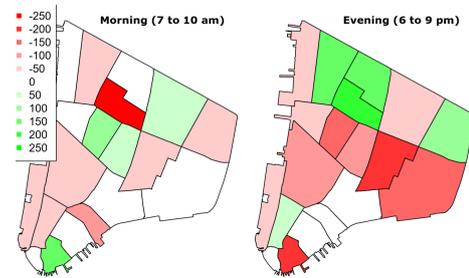


Figure 1: Observed demand imbalance in New York Taxi dataset [21] trips between morning (7-10am) and evening (6-9pm) peak hours in the south part of Manhattan on Tuesday, February 2nd 2016

mobility demand is changing over time and distribution of requests is uneven. This can lead to an unbalanced fleet distribution for RS-enabled MoD systems [2], as illustrated in Figure 2, where most of the demand is concentrated in the top area while majority of vehicles are located in the opposite side after finishing their last trip, where fewer new customers are requesting for a ride.

Adaptively following (or even preventing) changes in the demand spatial patterns can improve the perceived level of service from the perspective of the use of the MoD system, and also, assuming a joined fleet in which human drivers can participate with their vehicles, its capability to consistently enable drivers to meet the actual demand while optimizing vehicles usage and increase the ROI (return on investment), assuming that the participation to the RS system would imply a subscription cost (due to setup and operating costs of the system).



Figure 2: Example of an unbalanced fleet distribution where demand (requests in red) location differs from available supply (vehicles)

To address this issue, existing work define rebalancing strategies that consist in relocating vehicles according to past or current demand. Rebalancing can be achieved using pre-defined location per vehicles (station-based relocation) or by defining a set of areas (also called zones) each vehicle can be sent to. Rebalancing approaches can rely on a static [9, 1, 8, 25, 24, 19] or a dynamic partition of the network [15] to split it into zones. Rebalancing vehicles using a

¹ School of Computer Science and Statistics, Trinity College Dublin, Ireland, emails: acastagn@tcd.ie, maxime.gueriau@scss.tcd.ie, ivana.dusparic@scss.tcd.ie

² University of Milano - Bicocca, Italy, email: giuseppe.vizzari@unimib.it

dynamic partition is expected to better track changes in mobility demand *e.g.*, caused by temporary network disruptions, special events concentrating demand temporarily and long-term city developments affecting observed patterns.

In this paper, we propose a Dynamic Demand-Responsive Rebalancer (D2R2) which, using Expectation-Maximization (EM) clustering technique, generates a dynamic set of rebalancing zones and computes relocating probability per zone from current demand trend every time a vehicle needs to relocate. Novelty of our approach is twofold: first, boundaries and number of relocating areas are computed when required and second, rebalancing probability for each zone is allocated dynamically using only unserved requests data available in real-time. Therefore, D2R2 does not require data collection and no learning phase is required, enabling our proposal to operate from the beginning while being responsive to current demand.

We evaluate D2R2 in an implementation of a Multi-Agent Reinforcement Learning (MARL) ride-sharing enabled MoD system where 200 vehicle agents serve 10,000 ride requests in lower Manhattan road network. Requests have been generated from the open NYC taxi dataset [21] to be representative of real demand patterns. Results show that coupling D2R2 with ride-sharing enhances performance from a single vehicle perspective, and improves the overall balance of the distribution of requests across the fleet. At the cost of few more kilometres travelled empty for rebalancing, the performance at the fleet level confirms the overall efficiency of our demand-responsive rebalancing strategy.

2 RELATED WORK

Rebalancing for MoD can be categorized in the approaches relying on static rebalancing zones [1, 8, 9, 24, 25] and dynamic zones [15]. In static rebalancing zone generation, geographical coverage of relocation zones is predefined at design time. For example, in [9], NYC Lower Manhattan area is divided in predefined zones which do not change over time. Each vehicle, using RL, learns and decides at each time step whether to relocate to one of the neighbouring zones or to stay in its current zone. In [8], rebalancing areas in Austin, Texas, are defined by partitioning the area in 2-mile by 2-mile square blocks. Block balance is calculated for each zone, capturing the excess or deficit of vehicles within the block in relation to supply and expected travel demand. Blocks with a negative balance try to gather vehicles from neighbourhood where there is a surplus, and expected travel demand is estimated from historical data and current requests. In [1], zones are defined using a fine-grained but also static grid. In [25], if a vehicle is idling, it can rebalance based on its local knowledge: according to the demand distribution in surrounding areas, it decides to rebalance to a neighbouring zone or not. The work presented in [24] partitions the area into rebalancing zones according to the road network layout. Zones are defined such as that for each region r_i , exists a zone which allows to reach r_i within the established maximum travel time. Idle vehicles are rebalanced by taking into account travel time, to limit empty travels, and future demand, estimated from current demand, to avoid an excess of vehicles in the same area. However, the zones, once defined at the start, do not change based on traffic or fleet conditions. Majority of the approaches allow rebalancing only for idle (and empty) vehicles, while [1] and [9] mix rebalancing with RS assignment, and allow RS pick-ups from neighbouring zones, shortening waiting time for passengers, but increasing their travel time.

The only current work that uses dynamic zone generation for rebalancing is presented in [15]; rebalancing zones are computed using

a clustering algorithm. A k-means clustering is applied on virtual requests generated by a distribution defined on historical data. Therefore the coverage and the size of the zones can change, but the total number of zones remains fixed. This approach is the closest related work, however, we allow different number of zones based on different density of requests. Furthermore, our approach relies on real time data rather than historical to have a better respond to dynamic demand.

Table 1 summarizes existing work on rebalancing for MoD systems, categorized by four main characteristics: *Analysed data*, which can be historical, real-time or both, depending on what kind of data is rebalancing based; *Dynamic # zones*, *i.e.*, whether the number of rebalancing zones can change over time; *RB empty*, whether the vehicles relocate only when empty or can relocate as a part of RS assignment; and *Dynamic boundaries*, whether the area covered by each rebalancing zone can adapt dynamically.

Table 1: Characteristics of existing rebalancing algorithms

	Analysed data	Dynamic # zones	RB only empty	Dynamic boundaries
Wen, 2017 [25]	Real-time	✗	✓	✗
Fagnant, 2017 [8]	Real-time Historical	✗	✓	✗
Alonso-Mora, 2018 [24]	Real-time	✗	✓	✗
Alabbasi, 2019 [1]	Historical	✗	✗	✗
Yang, 2019 [15]	Historical	✗	✓	✓
Guériaux, 2020 [9]	Real-time Historical	✗	✗	✗
This paper	Real-time	✓	✓	✓

We observe that the main issue of reviewed research is the low adaptability to demand changes (daily, seasonal, or more long-term ones resulting from new city developments), as the addition of new city areas/zones, or changing their granularity, requires system redesign.

With respect to request assignment, multiple algorithms are used in the literature to match riders and drivers (or vehicles). For example, [3, 4, 2] use integer programming to optimize the objective function for the optimal matching. RL-based approaches [9, 10, 1] are the closest to our approach, in which agents explore by themselves possible solutions without having any prior knowledge. A full review of vehicle assignment algorithms is out of scope of this paper as our contribution focuses on rebalancing, nevertheless, it is worth mentioning that, even though we illustrate D2R2 application in conjunction with an RL-based vehicle assignment, R2D2 is designed to be independent from the assignment algorithm used in the MoD system.

3 BACKGROUND

This section introduces the background information needed to understand D2R2 design and implementation: Reinforcement Learning (RL) used for vehicle assignment problem and expectation maximization (EM) with model selection criterion for rebalancing.

3.1 Deep Reinforcement Learning

RL is a branch of machine learning in which an agent learns autonomously by trial-and-error to map actions to the current environment state, by receiving a positive or negative reward for their execution [23]. The goal of the agent is to learn actions that maximize

the long term cumulative reward. RL iterates three tasks: at each time step an agent obtains the perception of the environment and maps it to a state s from its overall state space. Based on past experience, it can select an action a from the action space. The agent then, at timestep t receives a reward $r_t = R(s_t, a_t)$ which expresses how good was the selected action.

In most of real-world scenarios, the environment space is complex or continuous, making it intractable to handle all possible state-action pairs. To overcome this issue, RL has been combined with deep neural networks to approximate states, giving rise to a range of Deep RL techniques. The approach we choose for our implementation is Proximal Policy Optimization (PPO) [22], which is simpler to implement and tune without affecting the performance when compared to other state-of-the-art Deep RL approaches. PPO uses a novel objective function, formed by three terms, which is maximized each iteration:

$$L_t^{CLIP+VF+S}(\Theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\Theta) - c_1 L_t^{VF}(\Theta) + c_2 S[\pi_\Theta](s_t) \right] \quad (1)$$

Θ is the policy's parameter vector. The first term L^{CLIP} is the clipped objective function defined in Equation 2. c_1 is a coefficient, defined between 0.5 and 1, applied to $L^{VF} = (V_\Theta(s_t) - V_t^{target})^2$, which computes the squared-error loss of V , the learned state-value function, compared to the target value at time t . Last term S is the entropy bonus, used to ensure sufficient exploration which is regulated by c_2 , ranging from 0 to 0.01. S of a stochastic policy π_Θ refers to state at time t .

$$L^{CLIP}(\Theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\Theta)\hat{A}_t, \text{clip}(r_t(\Theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t) \right] \quad (2)$$

In the L^{CLIP} objective function definition (Equation 2), the first term inside the \min is the surrogate objective with a conservative policy iteration which is clipped by the second term. \hat{A} is an estimator of the advantage function shown in Equation 3 and $r_t(\Theta)$ denotes the probability ratio $\frac{\pi_\Theta(a_t|s_t)}{\pi_{\Theta_{old}}(a_t|s_t)}$ that expresses the difference between current and old policy, which is clipped if difference falls out of boundaries by ϵ , a small hyper-parameter which weighs distance from new policy in respect to the old.

$$\hat{A}_t = \sigma_t + (\gamma\lambda)\sigma_{t+1} + (\gamma\lambda)^2\sigma_{t+2} + \dots + (\gamma\lambda)^{(T-t+1)}\sigma_{t-1} \quad (3)$$

where, $\sigma_t = r_t + \gamma V(s_{t+1}) - V(s_t)$

During training, PPO method collects a sequence of samples of length T from the environment and then estimates the advantage \hat{A}_t for the complete sequence. Finally, several epochs of optimization on L^{CLIP} are performed on the same batch, to maximize gathered experiences.

While we are not aware of any application of PPO in a ride-sharing problem, it shows good performance in many other applications with similar characteristics, such as portfolio management [14], robot control [12, 18, 16] or simulation and games [5, 26], which motivated our decision to use it as basis for our implementation.

3.2 Clustering

Expectation-Maximization (EM) is an iterative algorithm to find the *maximum-likelihood* estimates of parameters by computing probabilities of cluster memberships across several probability distributions [20]. EM consists of two steps, *Expectation* (E step), which

computes the complete log-likelihood of data memberships in clusters, and *Maximization* (M step), which, by maximizing the computed likelihood, updates the hidden parameters θ related to normal distribution, hence mean, variance and prior probability for each cluster.

E step: hidden parameters are initialized by some random values or, more likely, computed from data points. Then, expectation of the complete log-likelihood, conditioned by observed samples and current estimation of θ , is computed. Expected value is:

$$Q(\theta; \theta(t)) = \mathbb{E} \left[\sum_k \ln(p_y(y_k; \theta|X; \theta(t))) \right] \quad (4)$$

where θ is the unknown parameter vector and the term inside the logarithm expresses the conditional probability of a datapoint to belong to a cluster k given the observed samples X and value of θ at the previous step.

M step: computes the next $(t + 1)$ -th estimation of the unknown parameter vector by maximising the expected value $Q(\theta; \theta(t))$ obtained from previous step.

$$\theta(t + 1) : \frac{\partial Q(\theta; \theta(t))}{\partial \theta} = 0 \quad (5)$$

EM algorithm terminates when difference between expectation at time t and time $t - 1$, obtained from Equation 4, is smaller than a threshold ϵ .

Once terminated, the likelihood indicates how good our model fits data. However, this parameter alone does not take into account overfitting and the number of clusters; in fact likelihood could be maximized with each datapoint belonging to a different cluster. An option to validate and select a model is by using Bayesian Information Criteria (BIC) [6]. It prevents overfitting by taking into account number of clusters. BIC is computed through Equation 6, where number of free parameters, k , depends on number of clusters. BIC measure weighs the number of free parameters with the number of samples available. It looks for the true model among the set of candidates.

$$\text{BIC} = \ln(n)k - 2 \ln(\hat{L}) \quad (6)$$

Where \hat{L} is the maximized value of the likelihood function, result of Equation 4, n is the number of data points within a dataset and k is the number of free parameters to be estimated.

We have opted to use EM over other clustering techniques because it computes clusters by estimating normal distribution with their parameters, which underlies data. By doing that, EM enables clusters to have different shapes unlike other clustering methods which tends to find clusters with comparable areas by working directly on data points.

4 DYNAMIC DEMAND-RESPONSIVE REBALANCER

This section describes the main contribution of our paper, a Demand-Responsive Zone Generation for Real-Time Vehicle Rebalancing (D2R2) in RS fleets. We first introduce our unpublished RS system using multi-agent Deep Reinforcement Learning, and then our novel rebalancer.

4.1 Ride-Sharing using Deep Reinforcement Learning

We designed a multi-agent decentralized algorithm for RS applied to a fleet composed of 5-seater autonomous vehicles for a MoD system,

which is model-free and designed to be replicable to any city in the world. To take a decision, agent implements PPO [22], introduced in Section 3. Each agent controls a vehicle, taking an action at each step by evaluating its internal state and perception, without communicating or coordinating with other vehicles. Once an agent completes its action a next step can begin. Agents evaluate and decide of an action in a sequential order. At each time step, each agent perceives the environment and decides of the next action: to pick-up a ride, to drop-off passengers or to rebalance. Finally, it updates its learning process. This cycle is described in Alg. 1.

Algorithm 1: Controller for a single vehicle V

Parameters: V vehicle, a action, r request, PPO model

```

1 Perceive and act ( $V$ )
2   update vehicle perception and status
3    $a \leftarrow PPO.getAction([V.perception, V.status])$ 
4   if  $a$  is parked then
5     if ( $V.queue \wedge V.perception$ ) are empty then
6       rebalance( $V$ ) // Alg. 2
7        $reward \leftarrow -0.01$ 
8     end
9     else  $V.wait()$ 
10    if  $V.queue$  is empty then  $reward \leftarrow -0.3$ 
11    else  $reward \leftarrow -0.5$ 
12  else if  $a$  is drop-off then
13    if  $V.queue$  is empty then return  $-10$ 
14     $V.destination \leftarrow \arg \min_{r_i \in V.queue} Supply(v, r_i)$ 
15     $detourRatio \leftarrow V.goToDestination()$ 
16    if  $detourRatio \leq max\_detourRatio$  then  $reward \leftarrow 5$ 
17    else  $reward \leftarrow 5 - (detourRatio - 1)$ 
18  else if  $a$  is pick-up
19    if  $\exists r$  associated to  $a \wedge V.freeSeats \geq r.passengers$ 
20    then
21       $V.pickUp(r)$ 
22      if  $size(V.queue) == 1$  then  $reward \leftarrow 1$ 
23      else  $reward \leftarrow 2$  // doing rs
24    end
25    else  $reward \leftarrow -10$ 
26  end
27   $PPO.update(reward, a)$ 

```

The agent’s internal state is composed by the vehicle position, represented by latitude-longitude pair, its destination, and the number of vacant seats. For an empty vehicle, destination is void, and if a vehicle is serving one or more requests, its destination matches to that of the request r_i that can be served the quickest, as shown in line 14 in Alg. 1. The vehicle location on the road network is updated every time that a new position is reached, either destination or pick-up point.

Perception P is composed of the three closest requests that an agent could serve. A request r is available to a vehicle v if v has enough empty seats to accommodate the number of passengers associated with the request (ranging from 1 to 6), and the total waiting time for r , i.e., the delay between request being created and estimated passengers pick-up time, is less than the maximum time allowed (set to 15 minutes). All customers, who have waited more than 15 minutes leave the system without being served, and the request is recorded as not served.

Perception is defined as the aggregation of states of re-

quests perceived by an agent, $P: \{S_{r,1}, S_{r,2}, S_{r,3}\}$. Where, $S_{r,i}: [r_{Pos}^i, r_{Dest}^i, r_{Passengers}^i]$ represents the state of the i -th request perceived by an agent. Each request consists of a pick-up location, the destination and number of passengers.

Vehicles can choose between 5 actions, organized in three categories: (1) *drop-off*, in which an agent serves a request by driving the passenger(s) to their destination; (2) *park*, in which an agent waits one minute being parked. (3) *pick-up*, an agent drives to a pick-up point of the selected request. Pick-up action has three variations, pick-up first, second or third request from the perception set. Once an agent selects a pick-up action, is first checked whether in perception corresponds a request and then if the vehicle can accommodate the new passenger(s), line 19 in Algorithm 1. When the vehicle perception is empty and it is not serving any requests, it is enabled to rebalance as shown at line 6 in Algorithm 1. We further discuss rebalancing in next section.

Rewards associated with actions are also shown in Algorithm 1. An agent get a negative reward of -10 for attempting to try to pick-up a request while it does not have enough free seats. Otherwise, the best (+5) and the potential worst reward are related to the same action: when a vehicle is doing a drop-off while carrying passenger from several requests, if the travel time for passengers to reach their destination exceed the estimated travel time without RS by 30% or more, then the reward is reduced according to the total additional detour distance travelled.

4.2 Rebalancer - D2R2

D2R2 rebalancer can be used with different MoD systems, however for illustration we describe its implementation as combined with the Deep RL ride-sharing request assignment strategy presented in previous section. Rebalancing is triggered when a vehicle is not serving any requests and it has no further requests to serve in its neighbourhood. D2R2 aims to dispatch vehicles efficiently and dynamically according to current demand, preventing fleet unbalance, which in turn can result in longer passenger waiting times, or an increased number of unserved requests. D2R2 infers relocating zones and computes their associated probabilities (Eq. 7) for a vehicle to be relocated into:

$$p_r(v, z_i) = \frac{|R_i|}{|R|} \quad (7)$$

where z_i is the i_{th} zone, R_i is the set of pending requests within current zone, and R is the set of pending requests across all zones.

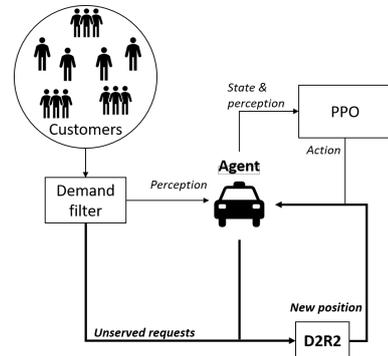


Figure 3: Ride-sharing framework with D2R2 integrated

R2D2 framework is illustrated in Figure 3, depicting the rebalancing module and an agent behaviour. The demand filter has a double

role: it selects three requests for agents perception and also filters demand for rebalancing, according to used time frame. D2R2 takes as input pending requests available at current time. All previous unsatisfied requests and future requests (estimated or scheduled), are not taken into account, as illustrated in Figure 4.

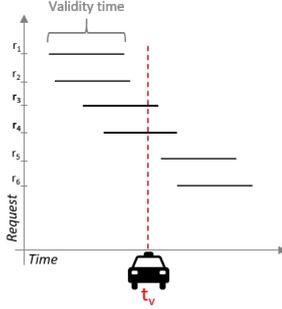


Figure 4: Example of pending requests at time t_v . Only requests active at time t_v , (i.e., r_3 , r_4) are taken into account when rebalancing

Algorithm 2 describes the procedure applied to relocate an idle vehicle to a new position. First, R2D2 generates new clusters, based on pending requests, following Algorithm 3. For a single rebalancing task, several runs of EM occur, producing different number of clusters. The algorithm then selects the model which better represents the data by choosing the clustering model which minimizes BIC measure (line 7 in Algorithm 3). The vehicle is then rebalanced to a zone within the selected clustering model according to a weighted random selection defined over a probability distribution among clusters and computed in Equation 7 (lines 3-4 in Algorithm 2). Among different approaches we preferred a weighted random selection to enable vehicles exploring different zones while rebalancing, avoiding all the vehicles to rebalance in a similar area.

Algorithm 2: Single vehicle relocating by D2R2

Result: relocates a vehicle V to a new position

```

1 rebalance ( $V$ )
2    $clusters, relocatingProbability \leftarrow$ 
    $updatingClusters(reqsAvailable, V.time)$  // Alg 3
    $rnd \leftarrow generate\ random\ value \in [0, 1]$ 
3    $i \leftarrow 0$ 
4   do  $i++$  while  $rnd < relocatingProbability[i]$ 
5    $V.destination \leftarrow clusters[i].getCenter()$ 
6    $V.driveToDestination()$ 

```

Each agent therefore, at each timestep, takes an action using its policy trained though PPO and, when needed, relocates by using EM as described in this section.

5 SIMULATION

We evaluate the performance of D2R2 in an RL-based RS system by using a fleet of 200 5-seater vehicles controlled by autonomous agents. Requests have been extracted from NYC city taxi data, and filtered to include only requests within lower Manhattan in New York City. Our evaluation is performed using data for the evening peak hour (from 6 to 9 pm), and consists of 10,000 requests. These 10,000 requests have been generated by aggregating trips from 50 consecutive Tuesdays between July 2015 and June 2016 (to represent a typical weekday demand pattern). Passengers by request, during evening

Algorithm 3: Defining relocating zones for rebalancing

Result: Given a set of pending requests ($reqsAvailable$) and a time, finds relocating zones $C : \{c_1, c_2, \dots, c_n\}$ with associated probabilities $P : \{p_1, p_2, \dots, p_n\}$

Parameters: V vehicle, r request, p probability

```

1 updatingClusters ( $reqsAvailable, V.time$ )
2   foreach  $r \in reqsAvailable$  do
3     if  $V.time - r.timeBegin \leq TIMEFRAME$  then
4        $queueReqs.append(r)$ 
5     end
6   end
7    $clusters \leftarrow min\_bic_{k=10}^{50}(EM(queueReqs, k))$ 
8    $prob \leftarrow 0$ 
9   foreach  $c \in clusters$  do
10     $prob \leftarrow prob + \frac{c.size()}{queueReqs.size()}$ 
11     $relocatingProb[c] \leftarrow prob$ 
12  end
13 return  $clusters, relocatingProb$ 

```

peak time, are distributed as follows: 71.95% of demand is composed by a single passenger request, 13% by two, 4.1% by three, 2.28% by four, 5.3% by 5 and the remaining 3.37% by 6.

Agents learning stage is performed by running multiple rounds of single-vehicle training. Only a single vehicle v_t is allowed to explore the environment at a given time t and can perceive remaining requests that the previous vehicle v_{t-1} could not serve.

This emulates a multi-vehicle concurrent exploration without competition between vehicles in serving customers. All the experience gained by all of the vehicle agents during training is gathered into a single learning process. In this way, all vehicles update the same learning process, to optimize the use of acquired knowledge and speed up the overall learning process. However, if one vehicle fails to do the update, or is not available to serve requests in a particular location, the others can continue seamlessly. Once training is completed, knowledge is replicated to all vehicles of the fleet. This allows new vehicles/agents to join the fleet without carrying out any additional training.

Travel information for vehicles is estimated using the Open Source Routing Machine (OSRM) [17], which, given two longitude-latitude coordinates, estimates the distance and travel time driving on the shortest route from origin and destination. Distance and time are computed according to a 24-hours snap-shoot acquired from a real-world scenario.

Agents, which are implemented through Tensorforce [13], use PPO with a deep neural network with a dense topology. Input layer has 20 neurons since an agent can perceive at most 3 requests and for each request is taken number of passengers and position with destination, as latitude-longitude coordinates, while the output layer has 5 neurons, one for each action. Between input and output layer lie three hidden layers, each of them composed of 32 neurons. The clipping ratio is set to 0.2 and the discount factor to 0.99. We used Adam [11] optimizer, with a learning rate of $1e-3$ and an entropy regularization set to 0.01.

PPO batching capacity is 100, and we executed 10 iterations over the batches of the PPO objective. The model was trained during 20 episodes, and each episode is composed of 10 rounds in which a vehicle serves requests. Each episode does not have a fixed number of iterations, as it terminates when a given agent does not have any more requests to serve. During training, rebalancing is disabled to

avoid unpredictable bias as it relies on a random weighted choice to select the new zone in which a vehicle is relocated into. For this reason integrating rebalancing as an action would have affected agents learning, requiring more exploration.

6 EVALUATION RESULTS AND ANALYSIS

To evaluate the performance of D2R2 rebalancing strategy, we propose 5 different scenarios, as presented in Table 2.

Table 2: Specification of evaluated scenarios

	Scenarios	RB	RS
Baselines	Base - no RB, no RS	no	no
	RS only	no	yes
	RS with fixed zones RB	yes*	yes
D2R2	D2R2 RB only	yes	no
	D2R2 RB and RS	yes	yes

* Vehicle randomly chooses the rebalancing zone.

The first baseline scenario (*Base - no RB, no RS*) imitates the behaviour of a simple MoD system without ride-sharing or rebalancing. In the second baseline scenario (*RS only*) ride-sharing alone is enabled. Third baseline is a MoD with both ride-sharing and rebalancing, but the number, size, and boundaries of zones is fixed (similarly to existing related work) and the zone for rebalancing is chosen randomly. We also evaluate two variations of D2R2. (*D2R2 RB only*) uses D2R2 without ride-sharing while (*D2R2 RB and RS*) uses both ride-sharing and rebalancing. This combination of scenarios allows to evaluate the benefits of rebalancing overall as well as the benefits of D2R2-based rebalancing specifically.

For all simulations, we assumed each vehicle to have a capacity of 5 passengers, therefore requests with more passengers are ignored by vehicles. This means that, based on the dataset used, the maximum level of service the fleet can reach is serving 96.63% of the 10,000 requests.

6.1 Evaluation metrics

We rely on commonly used measures adopted by related work [1, 2, 9, 24] to evaluate the performance of D2R2 in the proposed scenarios. The overall performance of the MoD fleet is assessed by analysing the number/percentage of served requests and the percentage of requests that involved ride-sharing (as opposed to occupancy of the vehicle only by one or more passengers from a single request). From the passenger perspective, we evaluate the average detour ratio ($\overline{D_r}$) (*i.e.*, the percentage of extra travel time used to facilitate ride-sharing) and the average waiting time (\overline{wt}). $\overline{D_r}$ is obtained by comparing actual time spent to reach the passenger destination serving other RS requests, and the expected travel time for a direct trip between the passenger origin and destination. \overline{wt} represents the average time passengers have to wait between creating a request and being picked-up. The maximum waiting time per request is limited to 15 minutes and after this the request is discarded from the system and flagged as unserved.

From the vehicles perspective, we recorded the distribution of the number of passengers per vehicle in the fleet and computed its variance (σ_{pass}). We also recorded the average distance travelled per vehicle ($\overline{d_t}$).

6.2 Simulation results

This section presents the simulation results for all the scenarios.

Level of Service From the system perspective, as reported in Table 3, each scenario shows a different level of service. Scenario *Base - no RB, no RS*, which models a classical taxi service, serves around 74% of requests, and *D2R2 RB only* serves 95% of the requests. All other scenarios (*RS only*, *RS with fixed zones RB*, and *D2R2 RB and RS*) achieve maximum level of service possible with 5-seater cars, of 96.63% requests. The results show that by enabling ride-sharing in combination with D2R2 rebalancing, decreases number of request ride-share (68% vs 80%) when compared to *RS only*. However, detour ratio is decreased (1.5 vs 1.7), which means that average time needed for passengers to reach their destination is decreased in *D2R2 RB and RS*.

Passenger waiting time Figure 5a shows passenger waiting times for each scenario. By enabling ride-sharing in the baseline scenario (*RS with fixed zones RB*) or D2R2 (*D2R2 RB and RS*), we observed a significant reduction when compared to *Base - no RB, no RS*. *RS only* shows lower overall waiting times (3.761 minutes) when compared to *D2R2 RB only* (4.218 min). However, ride-sharing can generate additional delay for passengers as vehicles are following a detour to serve more requests as shown in Table 3. Rebalancing in addition can limit the additional travelled distance due to detours as the vehicle relocates in zones with closer requests that can be matched. Hence, *D2R2 RB only* results as the best option for passengers with respect to travel time.

Passenger Distribution We can observe in Figure 5b that in *Base - no RB, no RS* many vehicles are driving with only a few passengers. These vehicles, once serving one or few requests, may end up in an area of the network that is empty of any further request. Enabling rebalancing or ride-sharing can prevent them from staying idle and help vehicle to find new requests. This can be observed from scenarios *D2R2 RB only* and *RS only*, where further improvements are achieved by enabling ride-sharing and rebalancing, as all vehicles serve a similar number of passengers. In particular, *D2R2 RB and RS* seems to converge to a higher average value when compared to baseline rebalancer *RS with fixed zones RB*. Moreover, as shown in Table 3, the variance of the number of served passengers is also lower: the combinations of these leads to conclude that the ROI for the individual vehicle/agent, member of this network, is more consistent and stable (for a fixed number of vehicles serving shared rides).

Vehicle mileage Distance travelled by agents is depicted in Figure 5c. Base scenario *Base - no RB, no RS* shows that around one fourth of vehicles are travelling only a few kilometres, confirming they only serve a few requests and then stay idle in an area with no further demand. Also, since the number of served requests varies by scenario, an important difference in terms of distance travelled was recorded. We further investigate travelled distance in Table 3. From the Table, we can confirm that enabling rebalancing adds additional travel distance for empty vehicles.

6.3 Discussion

Simulation results show that enabling ride-sharing alone is enough to satisfy all of the requests possible to serve, when the baseline *Base - no RB, no RS* only serves 74%. We showed that D2R2 improves the average and individual performance of all vehicles when enabled in combination with ride-sharing (*D2R2 RB and RS*) compared to *Base - no RB, no RS*. Passenger waiting time for (*D2R2 RB and RS*)

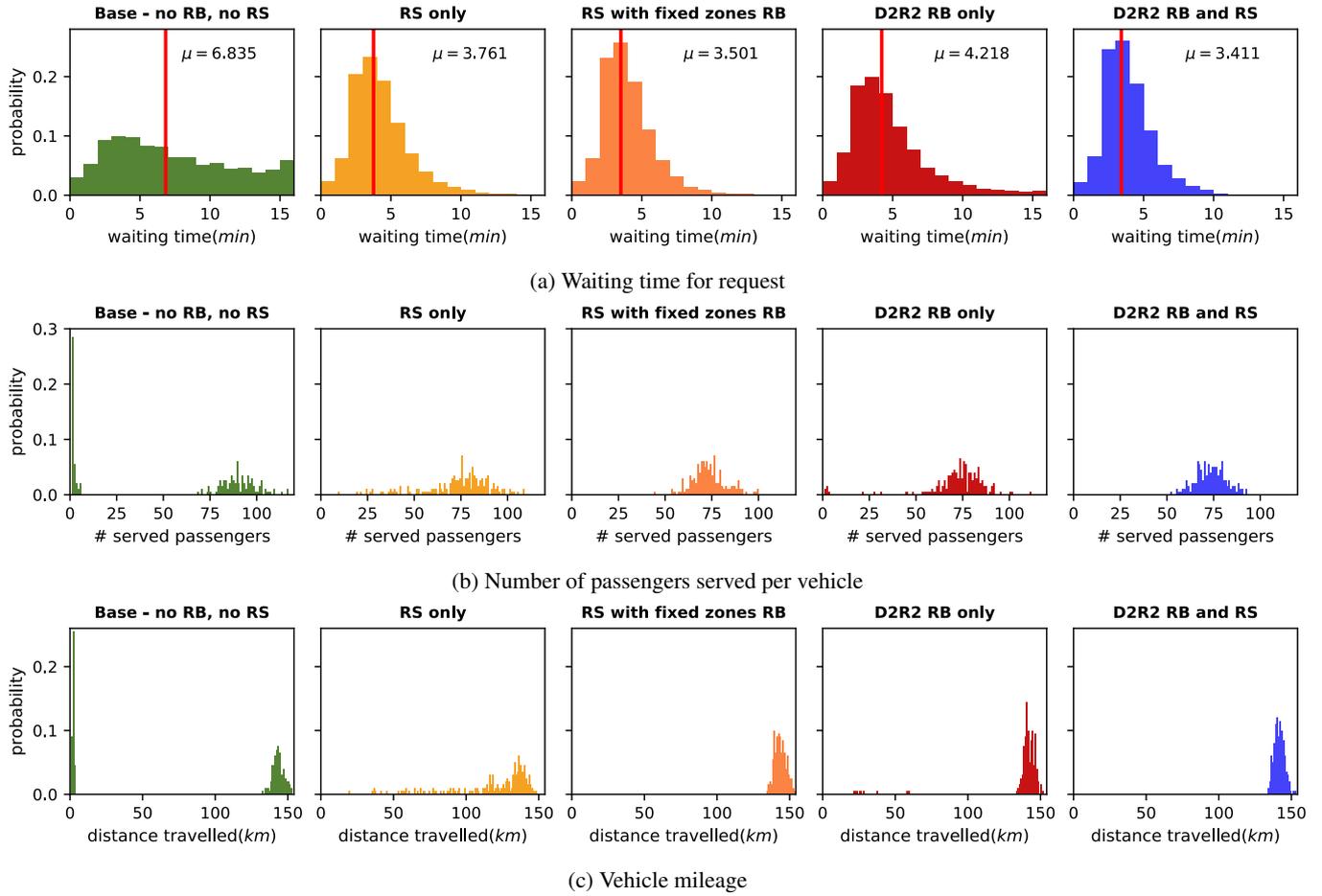


Figure 5: Comparison of simulation results for the implemented scenarios: waiting time (a), passengers distribution per vehicle (b) and distance travelled per vehicle (c)

Table 3: All values refer to 10,000 requests served by a fleet composed by 200 5-seater vehicles

Scenarios	%Served Reqs.	%RS	\overline{wt} (sec)	σ pass	\overline{d}_t (km)	\overline{D}_r
Base	74.21	0	410	1926	88	1
RS only	96.63	80	226	331	118.1	1.7
RS with fixed zones RB	96.63	70	210	82	143.6	1.5
D2R2 RB only	95	0	253	274	137	1
D2R2 RB and RS	96.63	68	204	63	141.2	1.5

is almost halved when compared to a normal taxi cab service (Scenario *Base - no RB, no RS*). However, the main observed advantage of D2R2 (with RB and RS) is that the workload that each vehicle has to carry out seems to better converge to a global average value (Figure 5b), resulting in fairer workload distribution. This is in contrast to other scenarios, where we observed that a few agents are contributing more than the others and some vehicles can be under-utilized, serving only a few requests. The closest in terms of workload distribution to our approach is (*RS with fixed zones RB*), however the variance is still larger at 82 passengers vs 63 in our approach (as σ_{pass} in Table 3 shows). Interestingly, fairness is not a metric existing work considered. Our results show that this should be included as a standard measure when evaluating ride-sharing systems, along side other system, user and vehicle metrics. We also observe that approaches with rebalanc-

ing (*RS with fixed zones RB* and both R2D2 scenarios) generate additional travelled distance: total mileage is slightly increased due to empty vehicles travelling as a part of relocation process.



Figure 6: Sample outcome of D2R2 clustering

To illustrate the zone outcomes of D2R2 and how does it differ from fixed zone clustering, we here show a snapshot of the number, size, and shape of the clusters it generated for a particular vehicle v at time t (Figure 6). In this instance, 10 relocation zones were computed by D2R2. Cluster centres are represented by a cross and colours intensity indicates its rebalancing probability. However, the number of clusters throughout the simulation, based on the demand

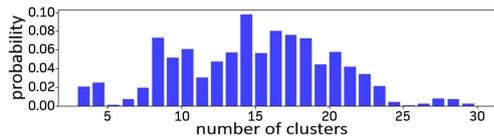


Figure 7: D2R2 clusters distribution during the simulated scenario

at any particular time, ranged from 4 to 30, as illustrated in Figure 7, with majority being in range between 13 and 20.

7 CONCLUSION AND FUTURE WORK

This paper presents a Dynamic Demand-Responsive Rebalancer (D2R2), a novel vehicle rebalancing algorithm for ride-sharing mobility-on-demand systems. Unlike existing approaches which use a fixed number and fixed geographical division of the network in zones to relocate empty vehicles, D2R2 uses EM clustering to dynamically generate zones. D2R2 enables zones to be dynamic in terms of their number, position and boundaries. First, rebalancing zones are identified by analysing real-time pending requests at each time step from each vehicle perspective. The zone to which a vehicle rebalances is then selected, among the defined zones according to a probability distribution defined over the zones. Thus, idle vehicles are spread across the area rather than rebalanced to the same zone(s). D2R2 effectiveness is shown by integrating it with 200 RL-based ride-sharing vehicles, which serve 10,000 ride-sharing requests in the lower Manhattan area. We compare D2R2 to approaches with no rebalancing and fixed-zone rebalancing, and observe a more fair workload division across the fleet when using D2R2, indicating a more accurate rebalancing strategy, without loss of performance with respect to waiting time and distribution of passengers per vehicle.

This work can be expanded in multiple directions. To verify its general applicability, it should be integrated with other RS approaches, and evaluated on other road network maps and datasets. In terms of improving the underlying PPO learning process, vehicles could be enabled with further online learning, to fine-tune their behaviours to new request patterns as they arise. Rebalancing could be further improved by taking into account vehicle position, for estimating travel time, when selecting the cluster to which to relocate. Thus, probability computed by Equation 7 would be conditioned by vehicle position and then normalized. Additionally to be more precise, it could integrate real-time traffic congestion data. It could further be integrated with existing approaches, as reviewed in Related Work, which use historical data to predict future demand, and tune cluster probabilities accordingly.

ACKNOWLEDGEMENTS

This publication has emanated from research supported in part by a research grant from Science Foundation Ireland (SFI) under Grant Number 18/CRT/6223, Center for Research Training in Artificial Intelligence.

REFERENCES

- [1] Abubakr Alabbasi, Arnob Ghosh, and Vaneet Aggarwal, ‘Deepool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning’, (03 2019).
- [2] Javier Alonso-Mora, Samitha Samaranyake, Alex Wallar, Emilio Frazzoli, and Daniela Rus, ‘On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment’, *Proceedings of the National Academy of Sciences*, **114**(3), 462–467, (2017).
- [3] Vincent Armant and Kenneth N. Brown, ‘Minimizing the driving distance in ride sharing systems’, *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, 568–575, (2014).
- [4] Vincent Armant, Nahid Mabub, and Kenneth N. Brown, ‘Maximising the number of participants in a ride-sharing scheme: Mip versus cp formulations’, *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, 836–843, (2015).
- [5] Trapit Bansal, Jakub W. Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch, ‘Emergent complexity via multi-agent competition’, *ArXiv*, **abs/1710.03748**, (2017).
- [6] Gerda Claeskens and Nils Lid Hjort, *The Bayesian information criterion*, 70–98, Cambridge Series in Statistical and Probabilistic Mathematics, Cambridge University Press, 2008.
- [7] Regina R Clewlow and Gouri Shankar Mishra, ‘Disruptive transportation: The adoption, utilization, and impacts of ride-hailing in the united states’, Technical report, (2017).
- [8] Daniel Fagnant and Kara Kockelman, ‘Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in austin, texas’, *Transportation*, **45**, (08 2016).
- [9] Maxime Guériau and Ivana Dusparic, ‘Samod: Shared autonomous mobility-on-demand using decentralized reinforcement learning’, in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1558–1563. IEEE, (2018).
- [10] Maxime Guériau, Federico Cugurullo, Ransford A. Acheampong, and Ivana Dusparic, ‘Shared autonomous mobility-on-demand: Learning-based approach and its performance in the presence of traffic congestion’, *IEEE Intelligent Transportation Systems Magazine*, (01 2020).
- [11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.
- [12] William Koch, Renato Mancuso, Richard West, and Azer Bestavros, ‘Reinforcement learning for uav attitude control’, *ACM Trans. Cyber-Phys. Syst.*, **3**(2), (February 2019).
- [13] Alexander Kuhnle, Michael Schaarschmidt, and Kai Fricke. Tensorflow: a tensorflow library for applied reinforcement learning. Web page, 2017.
- [14] Zhipeng Liang, Kangkang Jiang, Hao Chen, Junhao Zhu, and Yanran Li, ‘Deep reinforcement learning in portfolio management’, *ArXiv*, **abs/1808.09940**, (2018).
- [15] Yang Liu and Samitha Samaranyake, ‘Proactive rebalancing and speed-up techniques for on-demand high capacity vehicle pooling’, *CoRR*, (2019).
- [16] Guilherme Lopes, Murillo Ferreira, Alexandre Simões, and Esther Colombini, ‘Intelligent control of a quadrotor with proximal policy optimization reinforcement learning’, pp. 503–508, (11 2018).
- [17] Dennis Luxen and Christian Vetter, ‘Real-time routing with openstreetmap data’, in *19th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2011, November 1-4, 2011, Chicago, IL, USA, Proceedings*, pp. 513–516, (2011).
- [18] A. Rupam Mahmood, Dmytro Korenkevych, Gautham Vasan, William Ma, and James Bergstra, ‘Benchmarking reinforcement learning algorithms on real-world robots’, in *CoRL*, (2018).
- [19] Katarzyna Marczuk, Harold Soh, Carlos Lima Azevedo, Der-Horng Lee, and Emilio Frazzoli, ‘Simulation framework for rebalancing of autonomous mobility on demand systems’, *MATEC Web of Conferences*, **81**, 01005, (01 2016).
- [20] T. K. Moon, ‘The expectation-maximization algorithm’, *IEEE Signal Processing Magazine*, **13**(6), 47–60, (Nov 1996).
- [21] NYC Taxi and Limousine Commission. Tlc trip record data, 2020.
- [22] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov, ‘Proximal policy optimization algorithms’, *CoRR*, **abs/1707.06347**, (2017).
- [23] Richard S. Sutton and Andrew G. Barto, *Introduction to Reinforcement Learning*, MIT Press, Cambridge, MA, USA, 1st edn., 1998.
- [24] A. Wallar, M. Van Der Zee, J. Alonso-Mora, and D. Rus, ‘Vehicle rebalancing for mobility-on-demand systems with ride-sharing’, in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4539–4546, (Oct 2018).
- [25] Jian Wen, Jinhua Zhao, and Patrick Jaillet, ‘Rebalancing shared mobility-on-demand systems: A reinforcement learning approach’, in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 220–225. IEEE, (2017).
- [26] Yunqi Zhao, Igor Borovikov, Jason Rupert, Caedmon Somers, and Ahmad Beirami, ‘On multi-agent learning in team sports games’, (2019).