5-2022

Hierarchical value decomposition for effective on-demand ride pooling

Hao JIANG
Singapore Management University, haojiang.2021@phdcs.smu.edu.sg

Pradeep VARAKANTHAM
Singapore Management University, pradeepv@smu.edu.sg

Citation

JIANG, Hao and VARAKANTHAM, Pradeep. Hierarchical value decomposition for effective on-demand ride pooling. (2022). *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), Virtual Conference, May 9-13*. 1-8.
Available at: https://ink.library.smu.edu.sg/sis_research/7096

Hierarchical Value Decomposition for Effective On-demand Ride-Pooling

Jiang Hao
Singapore Management University
Singapore, Singapore
haojiang.2021@phdcs.smu.edu.sg

Pradeep Varakantham
Singapore Management University
Singapore, Singapore
pradeepv@smu.edu.sg

ABSTRACT

On-demand ride-pooling (e.g., UberPool, GrabShare) services focus on serving multiple different customer requests using each vehicle, i.e., an empty or partially filled vehicle can be assigned requests from different passengers with different origins and destinations. On the other hand, in Taxi on Demand (ToD) services (e.g., UberX), one vehicle is assigned to only one request at a time. On-demand ride pooling is not only beneficial to customers (lower cost), drivers (higher revenue per trip) and aggregation companies (higher revenue), but is also of crucial importance to the environment as it reduces the number of vehicles required on the roads.

Since each vehicle has to be matched with a combination of customer requests, the matching problem in ride pooling is significantly more challenging. Due to this complexity, most existing solutions to ride-pooling problem are myopic in that they either ignore future impact of current matches or the impact of other taxis in the expected revenue earned by a taxi. In this paper, we build on an approximate dynamic programming framework to consider impact of other taxis on the value of a taxi (expected revenue earned until end of horizon) through a novel hierarchical value decomposition framework. On a real world city scale taxi data set, we show a significant improvement of up to 10.7% in requests served compared to existing best method for on-demand ride pooling.

KEYWORDS

Ride Pooling; Neural Approximate Dynamic Programming; Reinforcement Learning; Value Decomposition

ACM Reference Format:
Jiang Hao and Pradeep Varakantham. 2022. Hierarchical Value Decomposition for Effective On-demand Ride-Pooling. In *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), Online, May 9–13, 2022*, IFAAMAS, 8 pages.

1 INTRODUCTION

Taxi/car on Demand (ToD) services (e.g., UberX, Lyft, Grab) are servicing more and more requests everyday in most major cities around the world. ToD services not only provide a comfortable means of transport for customers, but also are good for the environment as it reduces usage of private vehicles and enables sharing of vehicles over time (while being used to serve one request at any one point in time). A further improvement of ToD is on-demand ride pooling (e.g., UberPool, LyftLine, GrabShare etc.), where vehicles are shared not only over time but also in space (on the taxi/car). On-demand ride pooling reduces the number of vehicles required, thereby reducing emissions and traffic congestion compared to Taxi/car on-Demand (ToD) services (e.g., UberX, Lyft). This is achieved while providing benefits to all the stakeholders involved: (a) Individual passengers have reduced costs due to sharing of space; (b) Drivers make more money per trip as multiple passengers (or passenger groups) are present; (c) For the aggregation company (like Uber, Lyft etc.) more customer requests can be satisfied with the same number of vehicles.

The underlying model employed to represent on-demand ride-pooling services is the Ride-Pool Matching Problem (RMP) [1, 2, 6]. In the RMP, the goal is to assign groups of customer requests to vehicles that can serve them, subject to service constraints (e.g., the detour delay for a customer cannot be more than 10 minutes) in such a way that a quality metric is maximised (e.g., number of requests or revenue). The RMP is equivalent to the ToD problem when the vehicle capacity (maximum number of simultaneous customer requests that can be served by a vehicle) is restricted to 1. Our focus is on solving RMP problems at city scale involving thousands of locations and multi-capacity (capacity > 1) vehicles.

In ToD, vehicles have to be assigned to at most one request and hence can be represented as a bipartite matching problem. RMP on the other hand is significantly more challenging as vehicles have to be assigned to request combinations thereby requiring matching on a tripartite graph of requests, trips (combinations of requests) and vehicles. Some of the early approaches to solving ride pooling problems have focussed on using optimisation and planning methods [11, 15, 16]. Such approaches are typically offline and cannot scale to large number of agents and requests. Most existing research of relevance in solving RMP problems at city scale has focused on myopic approaches [1, 3, 6, 8, 18] that take the best matching decisions for the current time step. Unfortunately, such approaches do not consider the impact of current assignment on future matches.

Another research thread of relevance is to employ Reinforcement Learning (RL) for matching problems [4, 5, 19–21]. It scales well and considers future impact. However, these approaches are focused on ToD setting and cannot be extended to RMP due to the need for matching on a tripartite graph (with request combinations) instead of a bipartite graph. Recently, there have been approaches [7, 17] that account for future impact of current matches, by maintaining a future value (expected reward) for each vehicle. Neural Approximate Dynamic Programming, NeurADP provides the best results so far on city scale RMPs and it employs Deep Learning representations with Approximate Dynamic Programming to calculate future value of each vehicle if a request is assigned to it. However, the

Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), P. Faliszewski, V. Mascardi, C. Pelachaud, M.E. Taylor (eds.), May 9–13, 2022, Online. © 2022 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

value decomposition framework employed is oblivious to presence of other agents.

In this paper, our key contribution is a novel HIerarchical ValuE decompoSition (HIVES) approach that considers the impact of other agents and their actions while computing the future value of an individual vehicle. More importantly, HIVES employs a hierarchical mixing network to compute joint value from individual vehicle values. Value decomposition [13, 14] has been the leading approach to solve cooperative Multi-Agent Reinforcement Learning (MARL) problems. The key distinctions between HIVES and existing value decomposition approaches for cooperative MARL problems are two fold: (i) our focus is on centralized training and centralized execution problems, as opposed to centralized training and decentralized execution problems in cooperative MARL; (ii) we have to consider RMP problems with thousands of agents (vehicles), as opposed to tens of agents in cooperative MARL. So, scalability requirements are significantly higher in our case.

We then compare HIVES to NeurADP, the current best solver [17] for RMP on a benchmark real-world dataset [10]. We show that our approach outperforms NeurADP across different parameter settings by up to 10.7%. To provide perspective, in real-world settings of Taxi/car on demand services, even a 0.5% improvement is considered a significant one [21].

2 RIDE-POOL MATCHING PROBLEM(RMP)

In RMP, we have a set of vehicles $\mathcal{R}$ spread out at different locations on a road network, $\mathcal{G}$. For each time window, Δ, we collect requests, $\mathcal{U}$ and assign them to different vehicles so as to maximise the objective $\mathcal{J}$. This maximization problem is constrained by the delay constraints, $\mathcal{D}$.

Formally, RMP can be formulated as the tuple:

$$\langle \mathcal{G}, \mathcal{U}, \mathcal{R}, \mathcal{D}, \Delta, O \rangle$$

- $\mathcal{G}$: denotes the underlying graph of the road network where intersections, $\mathcal{I}$ are the vertices in the graph and roads connecting the intersections, $\mathcal{E}$ are the edges. Without loss of generality, we assume that pickups typically happen at intersections (as any pickup can be matched to a nearby intersection).
- $\mathcal{U}$: denotes the set of user requests. $\mathcal{U}_t$ is the set of all requests collected in an epoch t, with $\mathcal{U} = \cup_t \mathcal{U}_t$. Each request u_t^j in $\mathcal{U}_t$ is represented using the tuple $\left\langle q_t^j, e_t^j, t \right\rangle$, where q_t^j denotes the start location, e_t^j denotes the destination, t denotes the arrival time of the request.
- $\mathcal{R}$: denotes the set of vehicles or resources. Each vehicle r^i in $\mathcal{R}$ is represented using the tuple $\left\langle c^i, l^i, \mathcal{L}^i \right\rangle$, where c^i denotes the capacity of the vehicle, i.e., maximum seats in each vehicle, l^i denotes the current location of the vehicle, $\mathcal{L}^i$ denotes the list of locations to go for the assigned requests to be satisfied. It is null if no request is assigned to vehicle i.
- $\mathcal{D}$: considers two delay constraints. One is the maximum allowed pick-up delay δ, i.e., the difference between arrival time of request and pick-up time of vehicle. Another is the maximum allowed detour delay λ, i.e., the difference between the estimated travel time from origin to destination in single-passenger taxi and sharing taxi.
- Δ : denotes the length of each time window.
- $\mathcal{J}$: denotes the objective. We use $\mathcal{J}_t^{i,f}$ to denote the value vehicle i achieves in epoch t on serving request combination f. The goal of RMP is to optimize the objective (e.g., served request/waiting time) in a specific time horizon $\mathcal{T}$.

3 BACKGROUND: NEURADP FOR SOLVING RMP

In this section, we describe the Neural Approximate Dynamic Programming algorithm [17], the leading approach for solving RMP.

Figure1 provides the overall flow. The process starts from a predefined road network $\mathcal{G}$ and randomized request set $\mathcal{U}$ and agent/vehicle set $\mathcal{R}$. In step (A), we initialize the state by approximating each agent to its nearest road intersection. The blue dotted line in the figure denotes the path from the pick up location to the drop off location of the request. Two triangles denotes the existing pick up/drop off locations of the vehicle while the black/grey dotted line denotes their current trajectory. In (B), we map the combination of requests and agents restricted by constraint $\mathcal{D}$ and generate feasible actions for each vehicle using approach in [1]. In (C), we score all the feasible actions using individual neural networks and do the assignment using an Integer Linear Program (ILP) in (D) according to the scores from (C). In (E), we use the best action chosen by the ILP to update the neural network based individual value functions. In (F), the agents move to execute the ride pool assignment.

Approximate Dynamic Program (ADP) is similar to a Markov Decision Problem (MDP) with the key difference that the transition uncertainty is extrinsic to the system and not dependent on the action. The ADP problem for RMP is formulated using the tuple $\langle S, A, \xi, T, \mathcal{J} \rangle$, where :

- S : The state of the system is represented as $s_t = (r_t, u_t)$ where r_t is the state of all vehicles and u_t contains all the requests waiting to be served. The state is obtained in Step A of Figure 1.
- A : At each time step there are a large number of requests arriving to the taxi service provider, however for an individual vehicle only a small number of such requests are reachable. The feasible set of request combinations for each vehicle i at time t, $\mathcal{F}_t^i$ is computed in Step B of Figure 1:

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

- ξ : denotes the exogenous information – the source of randomness in the system. This would correspond to the user requests or demand. ξ_t denotes the exogenous information at time t.
- T : denotes the transitions of system state. In an ADP, the system evolution happens as

$$(s_0, a_0, s_0^a, \xi_1, s_1, a_1, s_1^a, \cdots, s_t, a_t, s_t^a, \cdots),$$

where s_t denotes the pre-decision state at decision epoch t and s_t^a denotes the post-decision state [12]. The transition from state s_t to s_{t+1} depends on the action vector a_t and the exogenous information ξ_{t+1}. Therefore,

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

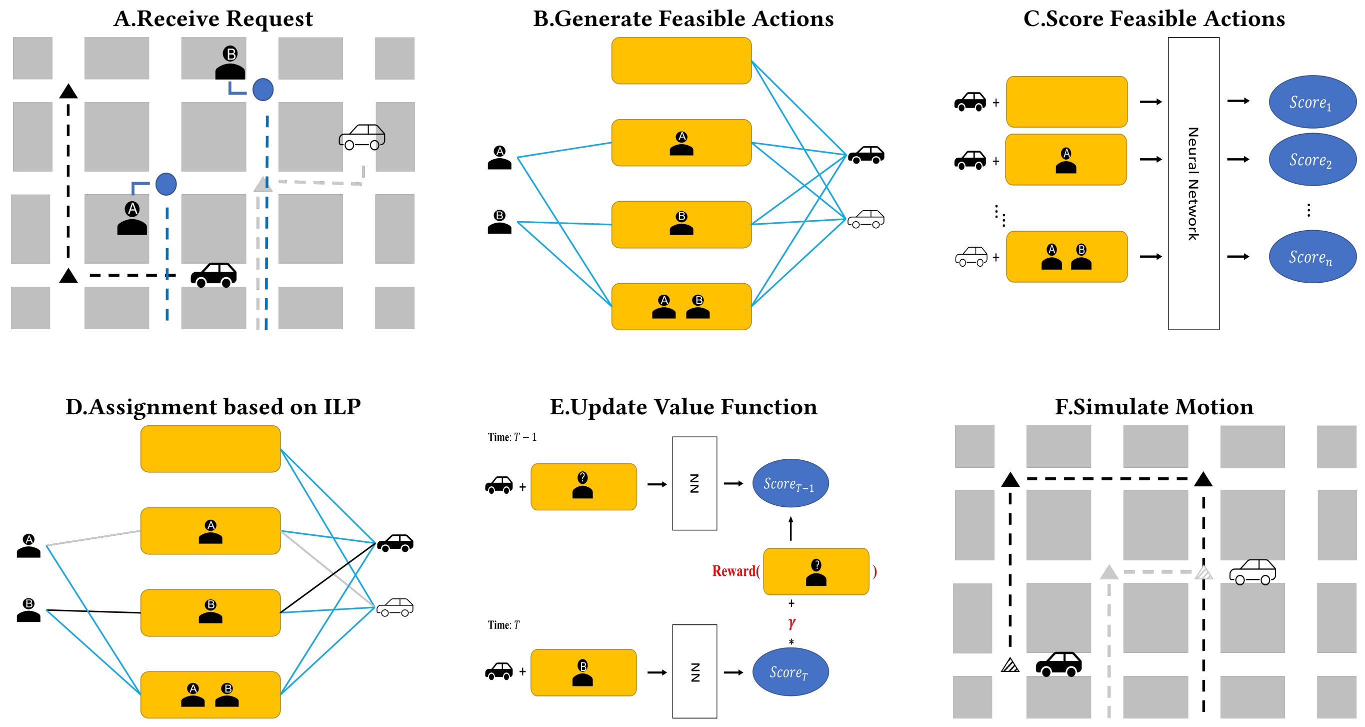

Figure 1: Outline of NeurADP approach for RMP

It should be noted that $T^a(.,.)$ is deterministic as uncertainty is extrinsic to the system.

$\mathcal{J}$: denotes the reward function and in RMP, this will be the revenue from a trip.

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

$$V(s_t) = \max_{z_t \in A_t} (\mathcal{J}(s_t, z_t) + \gamma \mathbb{E}[V(s_{t+1})|s_t, a_t, \xi_{t+1}]) \tag{2}$$

where γ is the discount factor. Using post-decision state, this expression breaks down nicely:

$$V(s_t) = \max_{z_t \in A_t} (\mathcal{J}(s_t, z_t) + \gamma V^a(s_t^a)) \tag{3}$$

$$V^a(s_t^a) = \mathbb{E}[V(s_{t+1})|s_t^a, \xi_{t+1}] \tag{4}$$

The advantage of this two-step value estimation is that the maximization problem in Equation 3 can be solved using a Integer Linear Program (ILP) with constraints on joint action space. Joint actions, $z_t \in \mathcal{A}_t$ across vehicles have to satisfy matching constraints: (i) each vehicle, i can only be assigned at most one request combination, f; (ii) at most one vehicle, i can be assigned to a request j; and (iii) a vehicle, i can be either assigned or not assigned to a request combination. Let $z_t^{i,f}$ denote the decision variable that indicates whether vehicle i takes action f (a combination of requests) at decision epoch t.

$$\sum_{f \in \mathcal{F}_t^i} z_t^{i,f} = 1 ::: \forall i \in R \tag{5}$$

$$\sum_{i \in \mathcal{R}} \sum_{f \in \mathcal{F}_t^i; j \in f} z_t^{i,f} \leq 1 ::: \forall j \in U_t \tag{6}$$

$$z_t^{i,f} \in \{0, 1\} ::: \forall i, f \tag{7}$$

As for the objective of ILP, the goal is to maximize the overall expected reward. The future value, $V^a(s_t^a)$ is non-linear and non-linear value functions, unlike their linear counterparts, cannot be directly integrated into the objective of ILP. One way to incorporate them is to evaluate the value function for all possible post-decision states and then add these values as constants. However, the number of post-decision states is exponential in the number of resources/vehicles.

To that end, Shah *et al.* [17] introduced a two-step decomposition of the joint value function, where the joint value function can be decomposed into individual values functions,

$$V^a(s_t^a) = \sum_i V^{i,a}(s_t^{i,a})$$

This allows NeurADP to get around the *combinatorial explosion of the post-decision state of all vehicles.*

Thus, the overall ILP is given by:

$$\max \sum_{i} \sum_{f \in \mathcal{F}_t^i} \left[o_t^{i,f} + V^{i,a}(T^{i,a}(s_t^i, f)) \right] \cdot z_t^{i,f} \tag{8}$$

Subject to constraints in equations 5, 6 and 7

The individual value function $V^{i,a}(.)$ is a neural network that is updated by stepping forward through time using sample realizations of exogenous information (i.e. demand observed in data) and best action computed by the ILP. In the objective, $V^{i,a}$ values (Step C of Figure 1) for all possible $s_t^{i,a}$ (from the individual value neural network) and then integrate the overall value function into the ILP as a linear function over these individual values. This reduces the number of evaluations of the non-linear value function from exponential to linear in the number of vehicles.

4 HIERARCHICAL VALUE DECOMPOSITION (HIVES)

The source of the problems with NeurADP is the following assumption on the joint value function:

$$V^a(s_t^a) = \sum_i V^{i,a}(s_t^{i,a})$$

First, joint value is obtained by taking a sum of individual values. Second, $V^{i,a}(.)$ depends only on the state and action of agent i and not on other agents. Our approach, HIVES addresses these issues by making the following major changes.

(1) We employ a hierarchical mixing neural network that combines individual values to get better estimate of the joint value.
(2) Individual value neural network takes as input not only state and action of agent i but pre-decision state of neighboring agents, i.e., $V^{i,a}(s_t^{i,a}, \cup_{j \in N_i} s_t^j)$

4.1 Hierarchical Mixing

First, we focus on providing a mechanism for combining the individual values of agents, so that they are able to compute joint value more accurately. QMix [14] has provided a mixing neural network in case of cooperative MARL problems, for centralized training and decentralized execution settings. Given we are interested in the less restrictive centralized execution settings, we can potentially utilize a similar mixing network dependent on individual values and joint state for combining individual values. Figure 2 provides the QMix network adapted for solving RMP.

The mixing network is a feed-forward neural network that takes the agent values as input and mixes them monotonically, producing the values of $V^a(.)$, as shown in Figure 2. The key implication of monotonicity in our setting is that:

$$\arg\max_{a_t \in A_t} V^a(T^a(s_t, a_t)) = \begin{pmatrix} \arg\max_{f^1 \in \mathcal{F}_t^1} V^{1,a}(T^{1,a}(s_t^1, f^1)) \\ \arg\max_{f^2 \in \mathcal{F}_t^2} V^{2,a}(T^{2,a}(s_t^2, f^2)) \\ \vdots \\ \vdots \end{pmatrix}$$

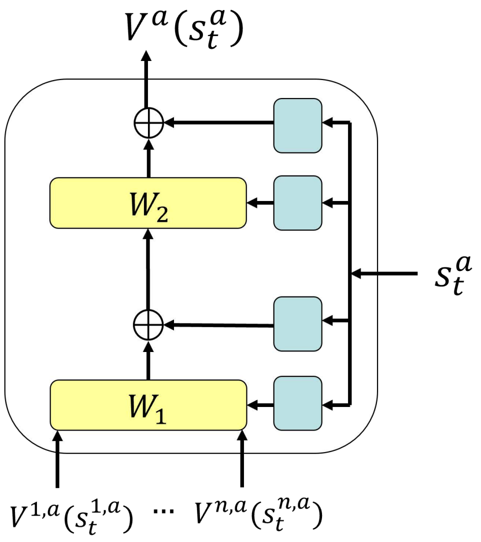

Figure 2: Hierarchical mixing network structure for solving RMP

Intuitively, this implies that the best joint action for $V^a(.)$ will maximize the individual values, $V^{i,a}(.)$ as well. In other words,

$$\frac{\partial V^a(s_t^a)}{\partial V^{i,a}(s_t^{i,a})} \geq 0, \forall i$$

It also implies that the weights (but not the biases) of the mixing neural network are restricted to be non-negative. This allows the mixing network to approximate any monotonic function arbitrarily closely.

Unfortunately, such an approach is more relevant in the context of problems with a few agents. This is because the network has to learn the contribution of each agent value to the joint value in the context of the state space, and when the number of agents increases to thousands, the network is unable to learn the contributions of individual agents effectively. We were able to verify this experimentally as well. QMix was unable to estimate the joint value any better than basic addition of individual vehicle values.

Thus, to ensure QMix is able to better estimate joint values, the challenge is to ensure that a mixing network does not combine values from a few agents. It is feasible to achieve this by considering a hierarchical mixing network, where the bottom level mixers combine values of individual agents belonging to a cluster into cluster values and the top level mixer combines the cluster values from different clusters into the overall joint value. Figure 3 provides the hierarchical mixing network we employ.

The key missing component to operationalize hierarchical mixing is the definition of clusters and which agents go into which clusters. RMP has two characteristics that make it easier to identify and cluster agents at any point in time:

- Agents that are nearby spatially are more probable to compete over the same set of requests and hence would have some level of dependency.

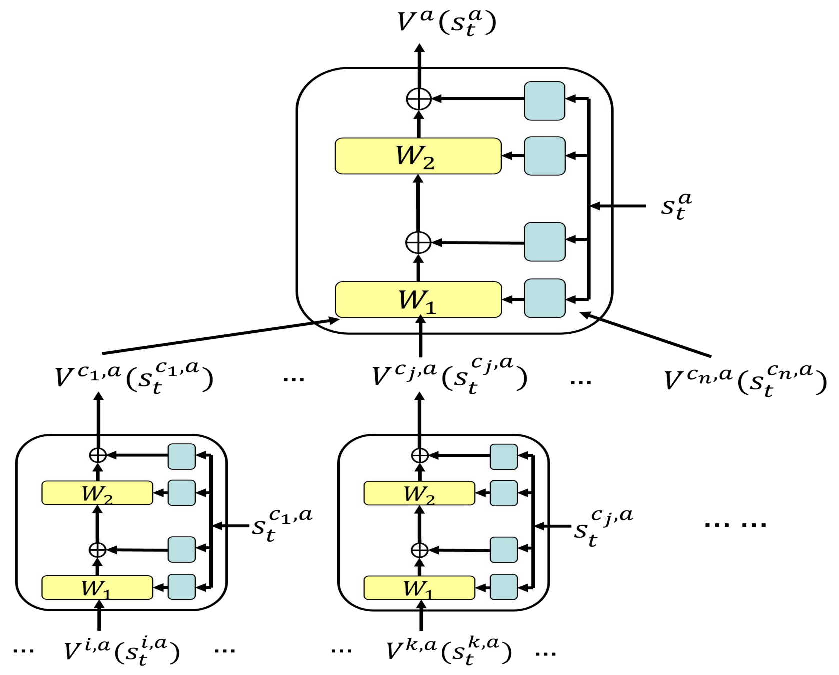

Figure 3: Hierarchical mixing network for solving RMP

- Agents/vehicles do not have identity, i.e., they are all homogenous.

Due to these characteristics, we can cluster the intersections in the road network (to capture spatial dependencies) and consider agents at a time step in an intersection cluster as neighboring agents. However, it should be noted that the agents in a cluster keep changing at each time step, as agents move between clusters while serving requests. Given the homogeneity of agents, it only matters how many agents are present and not which specific agents. At time t, the value of an agent placed at an intersection belonging to cluster c_k will be mixed with other agents belonging to the cluster. Figure 3 provides the architecture for this hierarchical decomposition.

After considering neighbours in individual value function, we input them together with the global environment state into HIVES. Algorithm 1 denotes how HIVES works after that while Algorithm 2 denotes QMIX operation in each QMIX network in HIVES.

Algorithm 1 HIVES()

```
1: **Input**: Individual value functions - $\{V^{i,a}(s_t^{i,a})\}_{i\leq|R|}$; Global environment state - $s_t^a$
2: **Initialize**:(1) Divide global environment state $s_t^a$ to $j$ cluster - $s_t^{c_j,a}$. (2) Classify agents to clusters $j$.
3: **for** each cluster $j$ **do**
4:     **for** agent $k$ in cluster $j$ **do**
5:         $V^{j,a}(s_t^{j,a}) = QMIX(V^{k,a}(s_t^{k,a}), s_t^{j,a})$
6: $V^a(s_t^a) = QMIX(V^{j,a}(s_t^{j,a}), s_t^a)$
7: **return** Joint Value Function $V^a(s_t^a)$
```

Algorithm 2 QMIX()

```
1: **Input**: Number of input values - $n$ ; Value functions – $\{V^{c_k,a}(s_t^{c_k,a})\}_{k\leq n}$; Environment state $s_t^{c,a}$
2: $Q = [V^{c_1,a}(s_t^{c_1,a}), ..., V^{c_n,a}(s_t^{c_n,a}), s_t^{c,a}]^T$
3: $Q_1 = w_1 Q^T + b_1$
4: $Q_2 = ReLu(Q_1)$
5: $Q_3 = w_2 Q_2^T + b_2$
6: $Q_4 = ReLu(Q_3)$
7: **return** Joint Value Function $Q_4$
```

It should be noted that the since we put together multiple QMix modules together, we have, $\forall s_t^a, s_t^{i,a}$:

$$\frac{\partial V^a(s_t^a)}{\partial V^{i,a}(s_t^{i,a})} = \frac{\partial V^a(s_t^a)}{\partial V^{c_j,a}(s_t^{c_j,a})} \cdot \frac{\partial V^{c_j,a}(s_t^{c_j,a})}{\partial V^{j_k,a}(s_t^{j_k,a})}$$

Both higher level and bottom level QMix modules have monotonicity property, so $\frac{\partial V^a(s_t^a)}{\partial V^{c_j,a}(s_t^{c_j,a})} \geq 0$ and $\frac{\partial V^{c_j,a}(s_t^{c_j,a})}{\partial V^{j_k,a}(s_t^{j_k,a})} \geq 0$, therefore:

$$\frac{\partial V^a(s_t^a)}{\partial V^{i,a}(s_t^{i,a})} \geq 0$$

Hence, the action maximizing joint value will maximize individual values as well with HIVES.

4.2 Neighbourhood based Individual Value

In RMP problems, vehicles are competing for the same demand, so future value of a vehicle will be dependent not only on its actions but also on the state of other nearby vehicles. As we demonstrate in

Table 1: Performance Improvement Provided by HIVES over NeurADP

	Variants	NeurADP	HIVES	Percentage Improvement
Capacity	2	163465	181023	10.741
	4	239842	255562	6.554
	10	252556	265237	5.021
Vehicles	500	187933	196301	4.453
	1000	239842	255562	6.554
	1500	297281	304615	2.467
	2000	324007	327276	1.009
pickup delay	60	144998	150632	3.886
	120	179902	190280	5.769
	300	239842	255562	6.554
	420	240753	260612	8.248

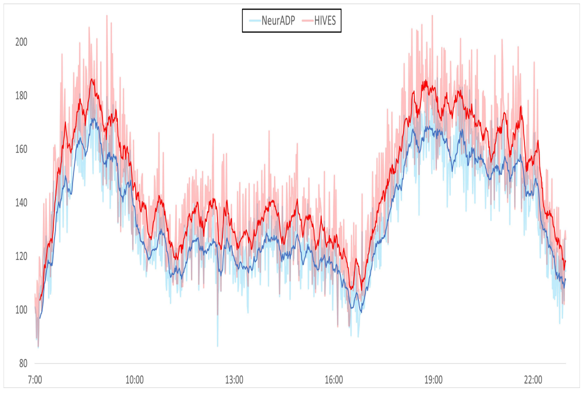

Figure 4: All Day Results

our experimental results, the independence of future value on other agents that is assumed in NeurADP can have a significant impact on overall performance. During the order assignment process, if there are many neighbours close to a single vehicle, its reward will be affected as the new arrived requests can be assigned to its neighbours. Thus, the reward of a given vehicle will be affected by the pre-decision state of neighbours instead of the post-decision state. That is to say, we consider $V^{i,a}(s_t^{i,a}, \cup_{j \in N_i} s_t^j)$ instead of just $V^{i,a}(s_t^{i,a})$.

While it is possible to consider many neighbours, the training of individual value network can become quite challenging. To balance the training performance and complexity, we employ the following mechanisms to restrict the neighbour set:

- As new requests will only be assigned to empty or partly filled vehicles, we only consider vehicles with capacity left as neighbours. While there exist some vehicles with full capacity which will finish its order in next time step, we also consider them as the 'potential' neighbours in our experiment.
- To search for neighbours of each vehicle, we assume all vehicles in the same cluster as neighbours with each other so that for each vehicle we search its cluster first. If the number of neighbours we set is lower than the number of neighbours of single vehicle, we greedily choose the closest vehicles as the set of neighbours.
- We do not consider vehicles which are more than a certain distance away to be neighbours. This will ensure that we only consider neighbours who add value.

As we will demonstrate in our experimental results, most of the performance gains are obtained by considering just a couple of neighbours.

5 EXPERIMENT

The goal of the experiments is to compare the performance of HIVES with NeurADP [17]. **Setup:** The experiment is performed with publicly available New York Yellow Taxi Dataset (NYYellowTaxi 2016). The experiment setup is similar to the original system[17]. We ignore some locations in the graph without outgoing edges and only consider areas with most requests' pick-up and drop-off locations in it. The final network has 4373 locations and 9540 edges. The capacity c^i was varied from 2 to 10, the total number of vehicles was varied from 500 to 2000 and the maximum pickup delay, δ from 120 seconds to 420 seconds. The maximum detour time λ equals to 2δ. The time window is always set as 60 seconds.The dataset contains about 300,000 requests each day and about 20 thousand requests during peak hour. We initialize the vehicles with randomized locations and same capacity. NeurADP model is trained for 8 weekdays (23 March – 1 April) and we use the average number of requests served during 4 – 8 April as the test result.

The number of agents in each cluster were set to be not more than 20, similar to the previous QMix experiments in StarCraftII. The number of neighbours for step 2 was limited to maximum of 3 while evaluating the impact of neighbourhood based Individual Value network.

The metric we use to compare the performance in experiments is the number of served requests (averaged over 3 runs of the algorithms).

Overall Results: We first compared NeurADP with the overall HIVES algorithm that combines hierarchical mixing and neighborhood based individual value. Table 1 provides the overall results for different settings of capacity, vehicle and pickup delay. Here are the key observations:

(1) HIVES outperformed NeurADP consistently across all settings. To provide perspective, even a 0.5%-1% is considered a significant improvement in city scale taxi on demand problems [21]. Here, our average improvement over all settings was 5.56%, maximum of 10.741% and minimum of 1.009%.
(2) As capacity increased from 2 to 10 keeping vehicles constant at 1000 and pickup delay at 300, the percentage improvement dropped from 10.741% to 5.021%. This is expected, as there is more supply available to cater to the same amount of demand.
(3) When number of vehicles was increased keeping capacity at 4 and pickup delay at 300, the highest improvement in service rate came at 1000 vehicles. When number of vehicles was 2000, the improvement was the least at 1.009%.

	Variants	NeurADP	HIVES (only Neighborhood based Individual Value) Number of Neighbours Considered 1	2	3
	2	163465	179725(9.947%)	179643(9.896%)	179675(9.916%)
Capacity	4	239842	253432(5.666%)	253710(5.782%)	253752(5.800%)
	10	252556	259703(2.830%)	263227(4.225%)	263274(4.244%)
	1000	239842	253432(5.666%)	253710(5.782%)	253752(5.800%)
Vehicles	1500	297281	302129(1.631%)	302012(1.591%)	302237(1.667%)
	2000	324007	324489(0.149%)	324605(0.185%)	324617(0.188%)
	120	179902	187932(4.464%)	188827(4.961%)	188831(4.963%)
pickup delay	300	239842	253432(5.666%)	253710(5.782%)	253752(5.800%)
	420	240753	258659(7.437%)	258676(7.445%)	258695(7.452%)

Table 2: Impact of Neighborhood

(4) Contrary to our expectation, as pickup delay was increased from 60 to 420 seconds, the performance improvement increased from 3.886% to 8.248% with capacity at 4 and number of vehicles at 1000. We expected that NeurADP can make up for wrong estimates in a joint value, when there is a larger pickup delay.

Figure 4 provides the all day result for the smallest case of 500 vehicles, where the improvement is not that substantial. At night HIVES and NeurADP provide roughly the same performance, so we focus during the day from 7:00 AM to 12:00 midnight. As can be seen, HIVES provide a consistent improvement at all points and not just few surges of better performance.

We have two enhancements in HIVES: (i) hierarchical mixing; and (ii) neighborhood based individual value. Here, we study the improvement provided by the two individually.

Ablation Analysis 1: Impact of neighborhood based individual value: Table 2 provides the impact of only the neighborhood based individual value network. Here are the key observations:

(1) Most of the improvement obtained by HIVES can be attributed to this aspect of our algorithm, except in case of 2000 vehicles and capacity 4 with pickup delay of 300.
(2) Considering even one neighbour seems to provide a significant improvement in almost all the settings.
(3) Only in the case of capacity 10 and 1000 vehicles, a second neighbor provided a significant improvement. Considering a third neighbour was not useful in any case.

Ablation Analysis 2: Impact of hierarchical mixing: Table 3 provides the results for evaluating the impact of only hierarchical mixing. On average, hierarchical mixing provided a constant improvement over around 0.67% on all settings. This was not a substantial improvement, but we believe this is because of vehicles changing clusters quite often. We leave this investigation for future work.

Ablation Analysis 3: Impact of combining hierarchical mixing and neighborhood based individual value: We observe that the sum of improvements provided by each of the enhancements individually is slightly less than the overall performance of the HIVES algorithm that puts both the enhancements together. This is a good outcome that indicates that the two enhancements are combining together well.

6 CONCLUSION

On-demand ride-pooling has been popular in many areas, such as taxi sharing, food delivery, goods transportation, ambulance allocation[9]. This is a challenging problem where combinations of demand requests have to be considered. Due to the challenging nature of the problem, existing approaches have either employed myopic heuristics that do not consider future impact of current matches or ignore the impact of other agents on the future value of a vehicle. We develop a HIerarchical ValuE decompoSition (HIVES) approach that addresses these issues with existing work. Heuristic approaches typically perform quite well in taxi on demand problems at city scale and hence even 0.5%-1% improvements in performance are considered to be substantial. Some details can further be improved in our approach, we will try a 3-level hierarchical network to do more clustering and add more uncertainty to the system, such as customers are likely to cancel the order due to different elements in our future work. Our HIVES approach is able to improve on existing best method by up to 10.7%, a truly significant improvement in performance.

ACKNOWLEDGMENTS

This research/project is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG2-RP-2020-017).

REFERENCES

[1] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. 2017. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences* 114, 3 (2017), 462–467.
[2] Xiaohui Bei and Shengyu Zhang. 2018. Algorithms for trip-vehicle assignment in ride-sharing. In *Thirty-second AAAI conference on artificial intelligence.*
[3] Yan Huang, Ruoming Jin, Favyen Bastani, and Xiaoyang Sean Wang. 2013. Large scale real-time ridesharing with service guarantee on road networks. *arXiv preprint arXiv:1302.6666* (2013).
[4] Minne Li, Zhiwei Qin, Yan Jiao, Yaodong Yang, Jun Wang, Chenxi Wang, Guobin Wu, and Jieping Ye. 2019. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. In *The World Wide Web Conference.* 983–994.

	Variants	NeurADP	HIVES (only hierarchical mixing)	Percentage Improvement
Capacity	2	163465	164589	0.688
	4	239842	241503	0.693
	10	252556	254318	0.698
Vehicles	1000	239842	241503	0.693
	1500	297281	299282	0.673
	2000	324007	326145	0.660
pickup delay	120	179902	181176	0.708
	300	239842	241503	0.693
	420	240753	242385	0.678

Table 3: Impact of Hierarchical Mixing

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

[6] Meghna Lowalekar, Pradeep Varakantham, and Patrick Jaillet. 2019. ZAC: A zone path construction approach for effective real-time ridesharing. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 29. 528–538.

[7] Meghna Lowalekar, Pradeep Varakantham, and Patrick Jaillet. 2020. Competitive ratios for online multi-capacity ridesharing. *arXiv preprint arXiv:2009.07925* (2020).

[8] Shuo Ma, Yu Zheng, and Ouri Wolfson. 2013. T-share: A large-scale dynamic taxi ridesharing service. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 410–421.

[9] Matthew S Maxwell, Mateo Restrepo, Shane G Henderson, and Huseyin Topaloglu. 2010. Approximate dynamic programming for ambulance redeployment. *INFORMS Journal on Computing* 22, 2 (2010), 266–281.

[10] NYYellowTaxi. 2016. New york yellow taxi dataset. http://www.nyc.gov/html/tlc/html/about/triprecorddata.shtml

[11] Sophie N Parragh, Karl F Doerner, and Richard F Hartl. 2008. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft* 58, 2 (2008), 81–117.

[12] Warren B Powell. 2007. *Approximate Dynamic Programming: Solving the curses of dimensionality.* Vol. 703. John Wiley & Sons.

[13] Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. 2020. Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:2006.10800* (2020).

[14] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2018. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*. PMLR, 4295–4304.

[15] Ulrike Ritzinger, Jakob Puchinger, and Richard F Hartl. 2016. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research* 54, 1 (2016), 215–231.

[16] Stefan Ropke and Jean-François Cordeau. 2009. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science* 43, 3 (2009), 267–286.

[17] Sanket Shah, Meghna Lowalekar, and Pradeep Varakantham. 2020. Neural approximate dynamic programming for on-demand ride-pooling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 507–515.

[18] Yongxin Tong, Yuxiang Zeng, Zimu Zhou, Lei Chen, Jieping Ye, and Ke Xu. 2018. A unified approach to route planning for shared mobility. *Proceedings of the VLDB Endowment* 11, 11 (2018), 1633.

[19] Tanvi Verma, Pradeep Varakantham, Sarit Kraus, and Hoong Chuin Lau. 2017. Augmenting decisions of taxi drivers through reinforcement learning for improving revenues. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 27.

[20] Zhaodong Wang, Zhiwei Qin, Xiaocheng Tang, Jieping Ye, and Hongtu Zhu. 2018. Deep reinforcement learning with knowledge transfer for online rides order dispatching. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 617–626.

[21] Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, and Jieping Ye. 2018. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 905–913.