Efficient Ridesharing Dispatch Using Multi-Agent Reinforcement Learning

Brian Fogelson*, Hansal Shah*, Oscar de Lima*, Ting-Sheng Chu*
{bfogels,shansal,oidelima,timchu}@umich.edu

May 17, 2020

Abstract

With the advent of ride-sharing services, there is a huge increase in the number of people who rely on them for various needs. Most of the earlier approaches tackling this issue required handcrafted functions for estimating travel times and passenger waiting times. Traditional Reinforcement Learning (RL) based methods attempting to solve the ridesharing problem are unable to accurately model the complex environment in which taxis operate. Prior Multi-Agent Deep RL based methods based on Independent DQN (IDQN) learn decentralized value functions prone to instability due to the concurrent learning and exploring of multiple agents. Our proposed method based on QMIX is able to achieve centralized training with decentralized execution. We show that our model performs better than the IDQN baseline on a fixed grid size and is able to generalize well to smaller or larger grid sizes. Also, our algorithm is able to outperform IDQN baseline in the scenario where we have a variable number of passengers and cars in each episode. Code for our paper is publicly available at https://github.com/UMich-ML-Group/RL-Ridesharing.

1 Introduction

Ridesharing services are becoming widely popular with services like Uber and Lyft. A recent study [1] estimated that the ridesharing economy will increase from $14 Billion in 2014 to $335 Billion by 2025. With this increasing demand, dispatching taxis efficiently has become a widely researched problem. Solving this problem is important since it has a two-fold advantage: helping increase customer satisfaction with shorter waiting times and increasing revenue for taxi services by completing more rides. Early works aimed to improve ridesharing efficiency [2, 3] but require handcrafted cost functions for estimating travel times and user waiting times. Clearly, these approaches suffer from inaccuracies when used in the dynamic real-world environment in which taxis operate. This complexity of the real world can be modeled more accurately using Reinforcement Learning (RL) based approaches.

In recent years, RL has achieved great success in modelling and solving extremely complex problems [4, 5]. Naturally, there has been work in the past to use RL for efficient ridesharing [6, 7, 8]. However these traditional methods are unable to accurately model the complicated dynamics involved in a real world environment. With the advent of Deep Reinforcement learning (DRL) [9, 10]

* indicates equal contribution

it has been possible to achieve super-human performance levels on previously intractable problems. In light of such advances, we propose a Multi-agent DRL based approach to solve the Ridesharing problem. [11, 12] use Independent DQN (IDQN) [13] based approaches which learn decentralized value functions or policies. These methods are prone to instability due to the non-stationarity of the environment induced by simultaneously learning and exploring agents. We use the approach of QMIX adopted from [14] to achieve centralized training and decentralized execution. This allows the network to be trained with a global value function and enables the choice of greedy actions for each agent corresponding to its individual value function.

In summary, we propose a multi-agent RL approach based on QMIX [14], which achieves better quantitative results than IDQN based methods. Additionally, we model the problem effectively using a First Come First Serve (FCFS) principle in which passengers requesting the ride first are given higher priority over others, which should reduce passenger waiting time. By selecting a car for the passenger with the highest priority at a given time, the size of the action space is reduced. Through experiments conducted in this project, it is demonstrated that a network trained on a fixed grid sizes can generalize well on other smaller or larger grid sizes and is able to perform better than the baseline when the number of passengers and cars are varied between each episode.

2 Related Work

Multi-Agent RL (MARL) has been considered an important problem in RL with rich history of works [15, 16]. Earlier methods of work consisted of mainly tabular based methods, but more recent work is moving towards deep learning based methods [13, 17, 18] which can tackle high-dimensional state and action spaces. One relevant approach to this work is that of IDQN [13] where two agents interact with each other only through the reward. Although these approaches achieve decentralization, they are prone to instability which arises due to the non-stationarity of the environment induced by simultaneously learning and exploring agents. Attempts have been made in the past to alleviate some of these issues. [19, 20] address learning stability but still learn decentralized value functions, whereas [21, 22, 17] adopt centralized learning of joint actions but require substantial communication between agents during execution. QMIX [14] provides centralized training with decentralized execution. QMIX employs a network that estimates joint action-values as a complex non-linear combination of per-agent values that condition only on local observations. We thus adopt a method based on QMIX for our solution.

Ridesharing, the focus of this paper has also been researched extensively in the past. Traditional methods [6, 7, 8] are unable to model the complex dynamics of the real world accurately. Algorithms proposed in [23, 24, 25] require routing recommendations to be provided to drivers to maximize company profits. Multi-agent IDQN based approaches [12, 11] suffer from the disadvantages mentioned earlier. The approach that we adopt, based on QMIX [14], achieves superior performance over other MARL methods. This allows our network to perform well in realistic scenarios with variable number of passengers and cars in each episode. Generalizing capabilities were also achieved with a model trained on a fixed grid size, which performed better than baseline methods when tested on a smaller or a larger grid size.

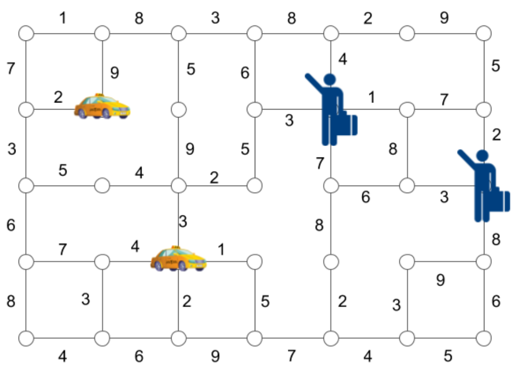

Figure 1: An example of simulation environment with randomly assigned road cost.

3 Methodology

In this section we outline the environment developed for this project, the Greedy and IDQN baselines implemented, and our proposed method based on QMIX [14].

3.1 Environment

In order to evaluate and compare the proposed algorithm, a simulated environment is implemented. To avoid a complicated implementation with a real-world map, a grid map is used. The nodes of the grid map correspond to different intersections while the edges between them correspond to different roads. Every road has a cost corresponding to the amount of time it takes a car to cross it. The costs are representative of factors including different traffic conditions. The costs are randomly assigned by the environment and cannot be observed by any algorithm. An illustration of the grid map is shown in Figure 1.

3.2 Greedy Baseline

A non-learning-based greedy algorithm is implemented as a baseline method and will be compared to various reinforcement learning methods. The greedy algorithm follows a first-come-first-serve (FCFS) strategy. As a result, passengers who requests the car earlier are given higher priority. Every passenger is paired to a car naively according to the distance. The Manhattan distance is used due to the nature of the grid map setting. No knowledge of the costs of the roads is used to make this decision. The performance of this greedy algorithm has been examined and the results are shown in Section 4. The time to finish dropping off all passengers is then measured for multiple combinations of cars and passengers.

3.3 Independent Deep Q-Learning (IDQN)

To dispatch cars to passengers in an efficient way, a reinforcement learning algorithm is introduced. Q-Learning has been proven to outperform previous state-of-the-art algorithms in scenarios similar to our problem, as shown in [11, 12]. Therefore, Q-Learning will be used as our baseline reinforcement learning algorithm.

The problem is defined as follows. The maximum number of cars and passengers on the map is defined as C_{max} and P_{max}. The state of each car on the map is defined as $s_c = (c_x, c_y)$ where c_x and c_y are the coordinates of the car. The state of each passenger on the map is defined as $s_p = (p_x, p_y, d_x, d_y)$ where p_x and p_y represent the coordinates of the pick up point, and d_x and d_y represent the coordinates of the drop off point for the passenger. Two binary indicator vectors, I_p of length P_{max} and I_c of length C_{max}, are used to tell the network which cars and passengers are in the environment. The observation of the environment is then $(s_c^1, ..., s_c^{C_{max}}, I_c, s_p^1, ..., s_p^{P_{max}}, I_p) \in \mathcal{S}$. If the i-th car or the j-th passenger do not show up on the map, then $c_x^i, c_y^i, p_x^j, p_y^j, d_x^j$ and d_y^j are set to 0. Each passenger is paired with a car as part of the overall action.

Deep Q-learning uses deep neural networks parameterized by ϕ to represent the action value function of an agent [9]. To make the algorithm more computationally efficiency, the action value of every passenger for every action is computed at the same time by the same network. Therefore, the output of this network is a matrix whose rows represent the passengers and whose columns represents the car to which each passenger can be matched. The value of the matrix at (i, j) is the Q-value where the i-th passenger takes the action of pairing with the j-th car. The dimension of the network output $\in \mathrm{R}^{P_{max} \times C_{max}}$.

Each passenger is paired with the car which has the maximum Q-value in the row, following the FCFS order. Each passenger gets their own reward, which is (-1) $\times$ (the passenger's waiting time). By doing so, when the reward is maximized, the passenger's waiting time is minimized. An episode starts by randomizing the state of every car and passenger on the grid map. When we test the scenario with variable agents, we also randomize the number of cars and passengers between episodes. All passengers are paired with cars at the beginning of the episode and the episode ends when all passengers are dropped off. A step in this implementation is equivalent to an episode, so when we store a transition tuple (s, a, r) in the replay memory we don't include the next state. The implication is that we also don't learn a target network. We use the Huber loss shown in eq. (1) [26] since it is robust to outliers when the estimates of the action-value are noisy and can prevent exploding gradients in some cases [27].

$$H(x) = \begin{cases} \frac{1}{2}x^2 & |x| \leq 1 \\ |x| - \frac{1}{2} & otherwise \end{cases} \tag{1}$$

The loss function is then:

$$L(\phi) = \frac{1}{B \times P_{max}} \sum_{i=1}^{B} \sum_{n=1}^{P_{max}} H(r_n^i - Q_n(s, a_n, \phi)) \tag{2}$$

where r_n is the reward for the n^{th} passenger, Q_n is the action-value for the n^{th} passenger given the choice if action a_n, and B is the batch size.

3.4 QMIX: Monotonic Value Factorization

While methods like IDQN and our greedy baseline try to maximize the reward of each agent independently, QMIX [14] promotes coordination between agents by having a shared global reward R_{tot} and learning a joint action-value function $Q_{tot}(s, a)$ where s is the observation of the environment and a is the joint-action of all the passengers. Consistency between the individual and global action-values is guaranteed by ensuring that the joint-action that maximizes the global action-value is equal to the set of actions that maximize the individual action-values as shown in equation (3).

$$\arg\max_a Q_{tot}(s,a) = \begin{bmatrix} \arg\max_{a_1} Q_1(s,a_1) \\ \arg\max_{a_2} Q_2(s,a_2) \\ \vdots \\ \arg\max_{a_{P_{max}}} Q_{P_{max}}(s,a_{P_{max}}) \end{bmatrix} \tag{3}$$

Having a global state-action value allows every passenger to choose actions greedily with respect to their individual action-values Q_a. Also, if the condition in equation (4) is satisfied, finding the $\arg\max_a Q_{tot}$ becomes trivial.

We can ensure monotonicity with a constraint on the relationship between the global action-value Q_{tot} and the action-value for each passenger Q_p:

$$\frac{\partial Q_{tot}}{\partial Q_p} \geq 0, \forall p \leq P_{max} \tag{4}$$

Our implementation has an agent network, a mixing network, and 2 hypernetworks [28]. The overall structure of the network can be seen in Figure 2. In the original QMIX implementation [14], each agent had their own agent network whose output was their value function $Q_p(s,a)$. For computational efficiency and simplicity we decided to use a shared network for all passengers like the one shown in section 3.3.

When optimizing our model, the mixing network takes the action-values from the selected actions in the replay buffer as input and mixes them monotonically to produce Q_{tot}. The weights of the mixing network, but not the biases, are restricted to be non-negative. Dugas et al. [29] showed that this allows any monotonic function to be approximated arbitrarily closely. We force the weights of the mixing network to be non-negative by using a separate hyperparameter network to produce them on every layer. The input to these hyperparatmeter networks is the state of the environment and the output is a vector which is reshaped to the appropriate size for that layer. We use a separate hyperparameter network for the weights of each layer and for each bias. The weights, but not the biases, are passed through an absolute activation function. The hyperparameter networks for the biases have one layer, except for the one used in the last layer of the mixing network which has 2 layers and a ReLU non-linearity. This architecture is shown in Figure. 2.a.

We train using the following loss:

$$L(\theta) = \frac{1}{B}\sum_{i=1}^{B} H(R^i_{tot} - Q_{tot}(s,a,\theta)) \tag{5}$$

where θ are parameters of the agent network, the mixing network and the hyperparameter networks together. H is Huber loss function defined in equation 1. B is the batch size and R_{tot} is the global reward for the episode which is equal to $(-1) \times$ (duration of the episode).

QMIX works better than other methods including *Value Decomposition Networks* [30] that also satisfy Equation 3, because it is able to represent a larger family of monotonic functions. In addition, full factorisation of the action-value function is not necessary to extract the decentralized policies.

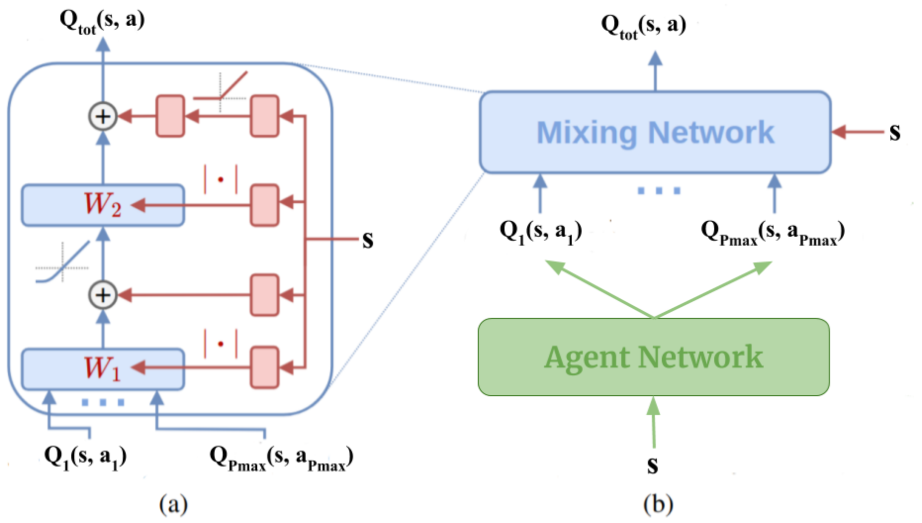

Figure 2: (a) Mixing network structure. In red are the hypernetworks that produce the weights and biases for mixing network layers shown in blue. (b) The overall QMIX architecture. Adapted from Rashid et al. [14]

4 Experimental Results

In this section, several implementation details and experiment results are presented, and the simulation environment mentioned in Section 3.1 is used. The experiments are divided into three parts. First, the model is trained on a 100×100 grid map with different combinations of cars and passengers and tested on the same map. In the second experiment, the performance on variable number of cars and passengers is evaluated, meaning numbers of cars and passengers is randomly assigned between episodes. Finally, a model trained on a 100×100 map is tested on different map sizes with different combinations of cars and passengers in order to test how the learnt strategies generalize to other map sizes.

For each experiment, four different methods are used for comparing. The first one is Random, in which passengers are randomly assigned cars. The methods: greedy, IDQN, and QMIX are mentioned in section 3. Hyperparameters and some brief explanations and design choices for IDQN and QMIX are mentioned in section 7. P indicates the number of passengers in the experiment and C stands for the number of cars. The values in the table are the average steps to drop off all passengers over $1,000$ episodes. A shorter episode time means better performance because it corresponds to all passengers being dropped off sooner.

4.1 Fixed Number of Passengers and Cars with a 100 $\times$ 100 Map

In the first experiment conducted, the number of cars and passengers were fixed for all episodes for a single configuration, and the grid size was set to 100 x 100. Different combinations of numbers of cars and passengers were compared in this fixed configuration. QMIX outperformed all of the other methods for each of the configurations tested. When there is a large number of cars and passengers such as in $P = 25, C = 20$; QMIX is 18.9% faster than greedy and 9.8% faster than IDQN. When there are more passengers than cars, such as in $P = 7, C = 2$, QMIX was 8.7% better

100 x 100 Map Size						
Method	**P=7,C=2**	**P=10,C=10**	**P=11,C=13**	**P=9,C=4**	**P=10,C=2**	**P=25,C=20**
Random	3386.248	2210.87	2089.87	2958.861	4644.59	2962.79
Greedy	3526.959	2208.55	2089.63	3072.806	4934.91	3173.54
DQN	3306.884	2102.65	2046.65	2763.201	4847.97	2853.66
QMIX (ours)	**3218.542**	**2042.44**	**1951.378**	**2724.029**	**4357.72**	**2573.24**

Table 1: Average episode duration for different configurations of passengers and cars

	10 x 10 Map Size	**500 x 500 Map Size**
Method	**Pmax= 10, Cmax= 10**	**Pmax= 20, Cmax= 20**
Random	209.03	13700.14
Greedy	201.85	13871.07
DQN	199.43	13462.82
QMIX(ours)	**195.66**	**12812.79**

Table 2: Average episode duration for a variable number of cars and passengers

than greedy and 2.7% faster than IDQN. When there were more cars than passengers such as in $P = 11, C = 13$, QMIX performed 6.6% faster than Greedy and 4.7% faster than IDQN.

4.2 Variable Number of Cars and Passengers

In the second experiment conducted, the number of cars and passengers was allowed to vary in between episodes. The average episode duration over 1000 testing episodes can be seen in Table 2. For each episode, the number of cars and passengers was sampled randomly from 1 to a predetermined maximum number of cars C_{max} and passengers P_{max}.

This is a more difficult and more realistic problem than a fixed number of cars and passengers since the strategy must adapt to more changing environmental factors including the number of agents. Nevertheless, QMIX was able to outperform the other methods in each of the configurations tested. For a 10×10 grid map with a maximum 10 cars and passengers, QMIX was 6.4% better than Random, and 1.9% better than IDQN. For the 500×500 grid map with a max of 20 cars and passengers, QMIX performed 7.6% better than greedy and 4.8% better than IDQN.

4.3 Generalization to Different Grid Sizes

In the third experiment conducted, the models trained in the first experiment grid were tested on different grid sizes to see how well the strategies learned on a 100×100 grid generalize to different sizes. The results obtained can be seen in Table 3.

For nearly every configuration, QMIX generalized to the other sizes tested better than any other method. For example, for $P = 7, C = 2$ on a 10×10 grid, the generalized QMIX performed 9.2% better than greedy and 2.3% better than the generalized IDQN. Additionally, for $P = 25, C = 20$ on a 500×500 grid map, the generalized QMIX performed 20.3% better than greedy and 12.7% better than the generalized IDQN. Even in the cases where IDQN generalized better than QMIX, QMIX was still comparable and was never more than 2.5% slower than IDQN.

10 x 10 Map Size						
Method	**P=7,C=2**	**P=10,C=10**	**P=11,C=13**	**P=9,C=4**	**P=10,C=2**	**P=25,C=20**
Random	337.3	215.64	208.77	287.57	449.38	291.62
Greedy	348.7	209.07	199.75	303.86	474.44	287.84
DQN	323.9	**201.28**	197.53	**262.40**	454.27	285.62
QMIX(ours)	**316.5**	206.46	**197.32**	265.10	**417.91**	**283.06**
500 x 500 Map Size						
Method	**P=7,C=2**	**P=10,C=10**	**P=11,C=13**	**P=9,C=4**	**P=10,C=2**	**P=25,C=20**
Random	17092.4	10860.21	10428.24	14715.82	23303.64	14820.33
Greedy	17251.2	10720.69	10905.83	15582.50	24491.44	16046.29
DQN	16473.1	10139.36	9968.44	13571.88	23688.50	14649.64
QMIX(ours)	**16274.3**	**10120.57**	**9835.75**	**13548.92**	**21871.99**	**12784.23**

Table 3: Training on a 100x100 grid and testing on other grid sizes

This result shows that QMIX can be trained on one map size and tested on another map size, either larger or smaller, and still perform fairly well. It was also able to generalize to other sizes better than IDQN.

5 Conclusion

In this paper, we propose a new MARL approach based on QMIX [14] which allows us to perform centralized training with decentralized execution. We compare our proposed method against the Random, Greedy as well as the IDQN method through various experiments. QMIX was able to complete the rides faster than the other methods for each configuration on a 100×100 grid with a fixed number of cars and passengers. Additionally, QMIX performed better than the other methods for a varying number of cars and passengers. Finally, once trained with a fixed number of cars and passengers, QMIX was able to generalize to other map sizes better than the other methods for nearly every configuration. Future directions for extending this work include considering a more realistic model for the weights in our maps from real-world datasets and also having a modifiable grid map where connections can be removed or added dynamically.

6 Author Contributions

All authors contributed equally to the work. Everyone met together multiple times per week to work on the conception of the problem statement and implementation of the simulation environment. After implementing an initial environment, the group members split into two groups of two working in parallel on implementing the four methods in two different environment situations. When one of this environments trained more efficiently, Ting-Sheng Chu began making the environment compatible with RLPYT. Oscar de Lima and Hansal Shah worked on implementing the QMIX and IDQN algorithms. Brian Fogelson and Hansal Shah began to test the networks with different configurations and different hyperparameters to evaluate performance. Then the whole group began running tests plus finalizing the presentation and paper together.

References

[1] Niam Yaraghi and Shamika Ravi. The current and future state of the sharing economy. *Available at SSRN 3041207*, 2017. 1

[2] Niels Agatz, Alan Erera, Martin Savelsbergh, and Xing Wang. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, 223(2):295–303, 2012. 1

[3] Ming Zhu, Xiao-Yang Liu, Feilong Tang, Meikang Qiu, Ruimin Shen, Wennie Shu, and Min-You Wu. Public vehicles for future urban transportation. *IEEE transactions on intelligent transportation systems*, 17(12):3344–3353, 2016. 1

[4] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016. 1

[5] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017. 1

[6] Gregory A Godfrey and Warren B Powell. An adaptive dynamic programming algorithm for dynamic fleet management, i: Single period travel times. *Transportation Science*, 36(1):21–39, 2002. 1, 2

[7] Gregory A Godfrey and Warren B Powell. An adaptive dynamic programming algorithm for dynamic fleet management, ii: Multiperiod travel times. *Transportation Science*, 36(1):40–54, 2002. 1, 2

[8] Chong Wei, Yinhu Wang, Xuedong Yan, and Chunfu Shao. Look-ahead insertion policy for a shared-taxi system based on reinforcement learning. *IEEE Access*, 6:5716–5726, 2017. 1, 2

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 1, 4

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 1

[11] Abubakr O Al-Abbasi, Arnob Ghosh, and Vaneet Aggarwal. Deeppool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 20(12):4714–4727, 2019. 2, 3

[12] Ishan Jindal, Zhiwei Tony Qin, Xuewen Chen, Matthew Nokleby, and Jieping Ye. Optimizing taxi carpool policies via reinforcement learning and spatio-temporal mining. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1417–1426. IEEE, 2018. 2, 3

[13] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4), 2017. 2

[14] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *ICML 2018: Proceedings of the Thirty-Fifth International Conference on Machine Learning*, July 2018. 2, 3, 4, 5, 6, 8

[15] Erfu Yang and Dongbing Gu. Multiagent reinforcement learning for multi-robot systems: A survey. Technical report, tech. rep, 2004. 2

[16] Lucian Bu, Robert Babu, Bart De Schutter, et al. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008. 2

[17] Peng Peng, Ying Wen, Yaodong Yang, Quan Yuan, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2017. 2

[18] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Thirty-second AAAI conference on artificial intelligence*, 2018. 2

[19] Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2681–2690. JMLR. org, 2017. 2

[20] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1146–1155. JMLR. org, 2017. 2

[21] Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored mdps. In *Advances in neural information processing systems*, pages 1523–1530, 2002. 2

[22] Jelle R Kok and Nikos Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7(Sep):1789–1828, 2006. 2

[23] Jason W Powell, Yan Huang, Favyen Bastani, and Minhe Ji. Towards reducing taxicab cruising time using spatio-temporal profitability maps. In *International Symposium on Spatial and Temporal Databases*, pages 242–260. Springer, 2011. 2

[24] Desheng Zhang, Tian He, Yunhuai Liu, Shan Lin, and John A Stankovic. A carpooling recommendation system for taxicab services. *IEEE Transactions on Emerging Topics in Computing*, 2(3):254–266, 2014. 2

[25] Meng Qu, Hengshu Zhu, Junming Liu, Guannan Liu, and Hui Xiong. A cost-effective recommender system for taxi drivers. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 45–54, 2014. 2

[26] Peter J. Huber. Robust estimation of a location parameter. *Ann. Math. Statist.*, 35(1):73–101, 03 1964. 4

[27] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 4

[28] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016. 5

[29] Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating functional knowledge in neural networks. *Journal of Machine Learning Research*, 10(Jun):1239–1262, 2009. 5

[30] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017. 5

7 Appendix

We describe the optimization parameters we used for our experiments.

7.1 Hyperparameters

Hyperparameters such as hidden size, learning rate, hidden size of the mixing network, batch size, epsilon decay, and number of episodes needed to be tuned. To keep results consistent, the hidden size, learning rate, batch size, epsilon decay, and number of episodes were kept consistent across both QMIX and IDQN. Both RL algorithms trained for 50000 episodes with a learning rate of .001, a hidden size of 128, a batch size of 128, and an epsilon decay of $20,000$. Additionally, the loss curves and episode duration curves, such as the one seen in Figure 3 over the episodes indicated that $50,000$ was an appropriate number of episodes. Epsilon decay affects how quickly a network learns, and having this value of epsilon decay resulted in a good balance between exploration and exploitation.

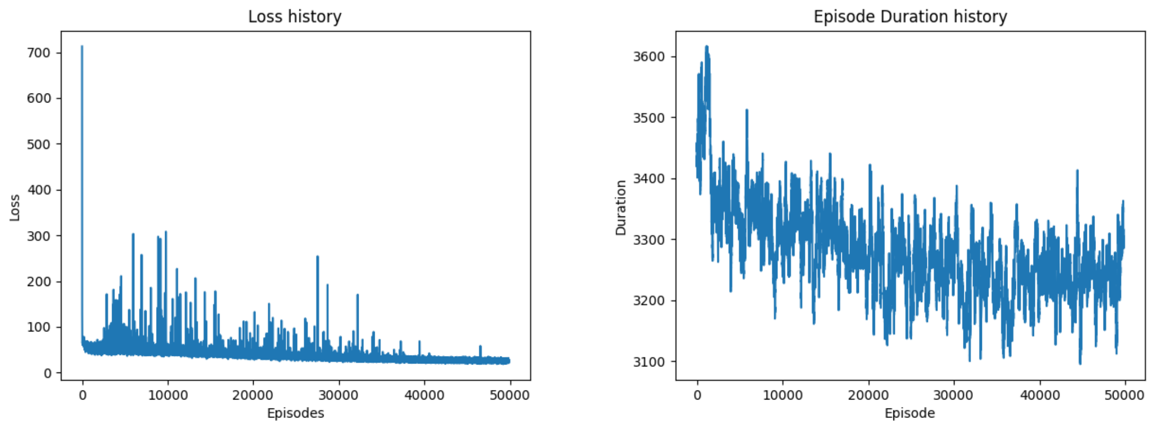

Figure 3: Huber Loss and Episode Durations over 50,000 episodes of QMIX for 4 passengers and 9 cars on a 100×100 grid