Heterogeneous Multi-Agent Deep Reinforcement Learning for Traffic Lights Control

Jeancarlo Arguello Calvo, Ivana Dusparic

School of Computer Science and Statistics, Trinity College Dublin
arguellj@tcd.ie, ivana.dusparic@scss.tcd.ie

Abstract. Reinforcement Learning (RL) has been extensively used in Urban Traffic Control (UTC) optimization due its capability to learn the dynamics of complex problems from interactions with the environment. Recent advances in Deep Reinforcement Learning (DRL) have opened up the possibilities for extending this work to more complex situations due to it overcoming the curse of dimensionality resulting from the exponential growth of the state and action spaces when incorporating fine-grained information. DRL has been shown to work very well for UTC on a single intersection, however, due to large training times, multi-junction implementations have been limited to training a single agent and replicating behaviour to other junctions, assuming homogeneity of all agents.
This study proposes the usage of Independent Deep Q-Network (IDQN) to train multiple heterogeneous agents, which is a more realistic scenario given heterogeneity of junction layouts in the city. We use Dueling Double Deep Q-Networks (DDDQNs) with prioritized experience replay to train each individual agent separately for its own conditions. We enrich this approach with fingerprinting to disambiguate the age of the data sampled from the replay memory to mitigate non-stationarity effects resulting from other agents affecting the environment.
Our IDQN approach is evaluated on three connected heterogeneous junctions in low and high traffic conditions, implementing different combinations of standard and prioritized experience replay and fingerprinting. Results show that IDQN is suitable approach to optimization in heterogeneous UTC with the best performance achieved by the combination of IDQN with prioritized experience replay but without fingerprinting.

1 Introduction

Traffic congestion is often caused by an inefficient control of traffic lights on intersections, i.e., settings not sufficiently adapted to particular set of conditions. In recent years several AI-based approaches have been investigated in order to optimize traffic flow in junctions. The most promising technique has been Reinforcement Learning (RL) due to its capacity to learn the dynamics of complex problems without any human intervention. Different RL implementations have been applied in Urban Traffic Control (UTC), both on single junctions/agents as well as on multiple collaborating junctions/agents. The main issue with RL approaches is the curse of dimensionality that arises from the exponential growth

of the state and action spaces because of the number of intersections. Combining RL with Neural Networks led to a method called Deep Reinforcement Learning (DRL) which enhances hugely the performance of RL for large scale problems. DRL techniques have demonstrated to work very well for traffic lights in single agent environments. Nonetheless, when the problem scales up to multiple intersections the need for coordination becomes more complex, and as a result, the latest studies take advantage of the similarity of agents in order to train several agents at the same time. However, assuming homogeneous junctions is not realistic as a city has a large range of different layouts of intersections.

This paper proposes a solution for heterogeneous multi-junction UTC scenario. Our approach is based on the Independent Deep Q-Network (IDQN) to achieve learning on multiple agents, with each agent using Dueling Double Deep Q-Network (DDDQN) enriched with fingerprinting to mitigate non-stationarity present in multi-agent environment. We evaluate the approach in widely used open source traffic simulator SUMO, and show it can successfully optimize the traffic flows on three heterogeneous junctions.

The rest of this paper is organized as follows: Section 2 reviews other work in DRL in UTC. Chapter 3 presents the techniques our approach is based on as well as the design specifics of our proposed IDQN technique. Chapter 4 presents the simulation environment, experiment design, results and their analysis, while Section 5 concludes the paper discussing avenues for future work.

2 Applications of RL in UTC

RL has been extensively applied in UTC, with some of the most advanced approaches addressing multi-agent multi-objective heterogenous scenarios [3]. To enable using image snapshosts of the intersection as input, while dealing with the curse of dimensionality, more recent work started exploring DRL and Convolutional Neural Networks (CNN) solutions in single agent scenarios [12, 9]. Multi-agent DRL work is currently limited to solutions for homogeneous multi-agent by taking advantage of the similarity in order to train only one neural network which can be used for all agents indistinctly. Approach presented in [17] used transfer learning to transfer the learnt policy across the agents, while in [10] only one agent is trained at each training episode, and the others react based on their previously learned policies. However, in the cases where junctions are heterogenous, the knowledge is not directly reusable, and agents need to be trained separately to account for the potential impact of other agents' actions on its own environment. In the following sections we introduce the details of our approach addressing these issues.

3 Design of IDQN approach to UTC

This paper proposes the usage of DRL to deal with the curse of dimensionality, that traditional RL approaches encounter, by using Neural Networks. To train multiple heterogeneous agents, it uses Independent Q-Learning (IQL), where

each agent learns independently and simultaneously its own policy, treating other agents as part of the environment. However, the environment becomes nonstationary from the point of view of each agent, as it involves the interaction with other agents who are themselves learning at the same time, ruling out any convergence guarantees. The technique we used is Dueling Double Deep Q-Networks (DDDQN), which relies in a component called experience replay memory in order to stabilize and improve the learning. (We have used Dueling Networks, Double DQN and Prioritized Experience Replay to improve the training over standard DQN). However, experience replay technique is incompatible with non-stationary environments. In order to combine the experience replay memory and IQL, we used a technique of fingerprinting which stabilizes the memory against the non-stationarity. This fingerprint disambiguates the age of the data sampled from the replay memory.

In this section we describe all the techniques underlying our approach, as well as present specific ways in which these are combined in our approach. We then detail the specifics of all RL components in our implementation, namely the design of the state space, action space, and reward function, and deep neural network architecture.

3.1 Deep Reinforcement Learning

Reinforcement learning (RL) learns optimal actions for specific environment conditions by trial-and-error learning to maximize the long term reward signal [13].

At each time step t, the agent perceives a state s_t in state space S from which it selects an action a_t in the action space A by following a policy π. The agent receives a reward r_t when it transitions to the state s_{t+1} according to the environment dynamics, the reward function $R(s_t,\ a_t,\ s_{t+1})$ and the transition function $T(s_t,\ a_t,\ s_{t+1})$. Discount factor $\gamma \in [0, 1]$ is applied to the future rewards.

RL can be either model-based or model-free (where the transition and the reward functions are unknown). The most common model-free technique is Q-Learning, where RL agents learn Q-values which are functions of state-action pair that returns a real value: Q: S x $A \rightarrow \mathbb{R}$. Thus, the policy is represented as:

$$\pi(s) = \arg\max_{a \in A} Q(s, a) \tag{1}$$

where the Q-value can be learned by using Q-learning updates [19]:

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha[R(s, a) + \max_{a \in A} Q(s', a')] \tag{2}$$

where $0 < \alpha \leq 1$ is a learning rate.

In situations with a large state and action spaces it is unfeasible to learn Q value estimates for each state and action pair independently as in standard tabular Q-Learning. Therefore, DRL models the components of RL with deep neural networks. There are multiple variations and extensions of the basic DRL model, so in the following subsections we describe the methods our approach utilizes to develop DRL algorithm applicable in heterogeneous UTC scenarios.

Deep Q Networks [11] proposed Deep Q-Networks (DQN) as a technique to combine Q-Learning with deep neural networks. RL is known to be unstable or even to diverge when a non-linear function approximator such as a neural network is used to represent the Q value. DQN addresses these instabilities by using two insights, experience replay and target network.

DQN parameterizes an approximate value function Q(s, a; θ_i) using Convolutional Neural Networks, where θ_i are the weights of the network at iteration i. The experience replay stores the agent's experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time step t in a dataset $D_t = e_1, \ldots, e_t$ pooled over many episodes into a replay memory. Then, mini batches of experience drawn uniformly at random from the dataset (s, a, r, s) $\sim$ U(D) are applied as Q-updates during the training. The Q-learning update at iteration i follows the loss function:

$$L_i(\theta_i) = E_{(s,a,r,s)\sim U(D)}\left[\left(r + \gamma \max_{a'} Q\left(s', a'; \theta_i^-\right) - Q\left(s, a; \theta_i\right)\right)^2\right] \quad (3)$$

where θ_i are the parameters of the Q-network at iteration i and θ_i^- are the target network parameters. The target network parameters are only updated with the Q-network parameters every C steps and are held fixed between individual updates.

Double DQN The max operator in standard Q-learning and DQN uses the same values both to select and to evaluate an action. It is known the this maximization sometimes produces to learn unrealistically high action values which tends to prefer overestimated values over underestimated values, resulting in overoptimistic value estimations. To prevent this, Double Q-learning decouples the selection and the evaluation [7].

For DQN architectures is not desired to fully decoupled the target as in [7] because the target network provides a intuitive option for the second value function, without having to include extra networks. For that reason, [6] propose to evaluate the greedy policy according to the online network, but using the target network to estimate its value given as result the Double DQN (DDQN) algorithm:

$$Y_t^{DDQN} = R_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta_t); \theta_i^-) \quad (4)$$

Prioritized Experience Replay [14] presented prioritized experience replay which detaches agents from considering transitions with the same frequency that they are experienced. Prioritized replay samples more frequently transitions from which there is a high expected learning progress, as measured by the magnitude of their temporal-difference (TD) error. It samples transitions with probability p_t relative to the last encountered absolute TD error:

$$p_t \propto \left|\left(R_{t+1} + \gamma_{t+1} \max_{a'} Q\left(s', a'; \theta_i^-\right) - Q\left(s, a; \theta_i\right)\right)\right|^{\omega} \quad (5)$$

Where ω is a hyper-parameter that determines the pattern of the distribution. New transitions are pushed into the replay buffer memory with maximum priority, providing a bias towards recent transitions.

Dueling Network Dueling Network is a technique proposed by [18] which computes separately the value V(s) and advantage A(s, a) functions that are represented by a duelling architecture that consists of two streams where each stream represents one of these functions. These two streams are combined by an convolutional layer to produce an estimate of the state-action value Q(s, a). The dueling network automatically produces separate estimates of the state value and advantage functions without supervision. Besides that, it can learn which states are valuable, without having to explore the consequence of each action for each state. Dueling network is defined with the equation:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A\left(s,a;\theta,\alpha\right) - \frac{1}{|\mathcal{A}|}\sum_{a'} A\left(s,a;\theta,\alpha\right) \right) \quad (6)$$

By using this approach, Dueling DQN achieves faster training over DQN.

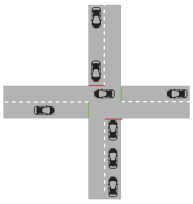
Multi-agent learning: Independent DQN In a multi-agent setting, multiple agents can collaborate directly by exchanging Q-values or jointly deciding on actions, or can operate independently in the shared environment, such as in the case of Independent Q-Learning (IQL)[16]. Independent DQN (IDQN) is an extension of IQL for DRL environments using DQN, where each agent a observes the partial state s_t^a, selects an individual action u_t^a, and receives a team reward, r_t shared among all agents. [15] combines DQN with independent Q-learning, where each agent a independently and simultaneously learns its own Q-function $Q^a(s, u^a; \theta_i^a)$ [4]. Since our setting is partially observable, IDQN can be implemented by having each agent conditioned on its action-observation history, i.e., $\mathrm{Q}_a(\tau_a, \mathrm{u}_a)$. In DRL, this can be implemented by given to each agent a DQN on its own observations and actions.

Fingerprinting in IQL A key component of DQN is the experience replay memory. Unfortunately, the combination of experience replay with IQL is problematic because the non-stationarity introduced by IQL results in data in experience replay memory no longer indicating the current dynamics in which the agent is learning. To address this issue, [5] introduced a fingerprint technique, which associates the data in replay memory with the age of the data, i.e., where in training trajectory does it originate from. It gradually changes over training time in order to allow the model to generalise across experiences in which the other agents execute policies as they learn.

3.2 State Representation

In our approach, the state is a image-like representation of the current state of the simulator environment (Figure 1a), similar to the concept used in [11].

The state consists of two matrices of 64x64: (1) a binary matrix P for vehicle positions (Figure 1b), and (2) a matrix S for vehicle speeds (Figure 1c). These matrices are based on those used in previous works such as [9, 10, 17].



(a) Simulation Environment State

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

(b) Position Matrix

$$\begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.4 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.8 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

(c) Speed Matrix

Fig. 1: State Representations

The locations are calculated by mapping the continuous space of the simulated environment into a discretized environment by creating a grid with cells of size C. The matrix P is a binary matrix where one indicates the presence of a vehicle and zero the absence of a vehicle (Figure 1b). The matrix S indicates the speed of the vehicle in the same cell position where a vehicle was calculated to be located in the matrix P (Figure 1c). The speed is represented as a percentage of the maximum allowed speed that it is computed by dividing the current vehicle's speed by the maximum allowed speed.

Additionally, a fingerprint is defined as: Let -a be the other agents of agent a, such that we can include the importance weights of the prioritized experience replay and the TD-Errors vectors sampled from the other agents into the observation function as θ_{-a} and TD_{-a} respectively. Given that, the new observation function is O'(s) = { O(s), $\theta_{-}-a$, TD_{-a} }. They are included in a matrix F.

3.3 Action Space

The action space varies depending of the structure of the intersection. For example, in a four road intersection the action space is defined as A = {NS, EW, NST; EWT} where NS stands for turning green North-South roads, EW stands for turning green East-West roads, NST stands for turning green North-South right turning, and EWT stands for turning green East-West right turning.

The duration of each phase is 1 time step, but at each time step the phase can be extended. Additional yellow phase is added during a fixed period of 3 time steps when the previous action is different than the current chosen action. This middle yellow phase reduces the risk of collisions.

3.4 Reward Function

Let $w_{i,t}$ be the i$^{\text{th}}$ vehicle's waiting time at time step t, and W_t the total cumulative waiting time for all the vehicles in the observation scope of the road network at time step t as shows in equation 7. The reward function is formulated in equation 8. Agent receives a positive reward from a range (0.0, 1.0] where the agent's reward loses are proportional to the cumulative waiting time at time step t. Thus, the agent must keep short waiting time in order to receive higher scores.

$$W_t = \sum_i w_{i,t} \tag{7}$$

$$r_t = \begin{cases} \frac{1.0}{W_t}, & \text{if cummulative waiting time } \mathrm{W}_t \text{ is greater than 0} \\ 1.0, & \text{otherwise} \end{cases} \tag{8}$$

3.5 Deep Neural Network Architecture

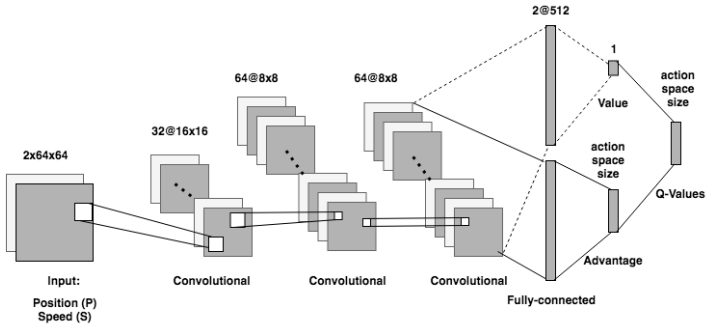


Fig. 2: The architecture of the Deep Neural Network.

The neural network is the same as used in [18]. The Figure 2 illustrates the Deep Neural Network. Every agent has its own Neural Network copy in order to allow it to learn its own local policy since every agent is trained independently by using IDQN.

4 Evaluation

In this section we present an evaluation of IDQN as a proposed solution for multi-agent DRL for heterogeneous agents in traffic light control using the traffic simulator SUMO [8]. We use OpenAI's baseline framework [2] to implement our DDDQN with prioritized experience replay. OpenAI Baselines is a set of implementations of RL in Python which uses TensorFlow library [1].

4.1 Set up and parameters

We run the test in the network layout shown in Figure 3, which consists of three heterogeneous junctions.

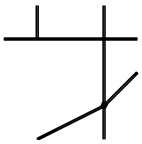


Fig. 3: Network Layout for Multi-Agent Experiments

The table 1 describes the hype-parameters used. Every episode corresponds to 1 hour of simulation time which is 3600 time steps, where every time step is 1 simulated second.

Table 1: Multi-agent evaluation hyper-parameters

Parameter	Value
Episodes	1000
Time steps	3600000
Pre train time steps	2500
Learning rate	0.0004
Exploration ϵ	$1.0 \rightarrow 0.01$
Time steps from starting ϵ to ending ϵ	360000
Target network update	5000
Prioritization exponent α	0.6
Prioritization importance sampling β	$0.4 \rightarrow 1.0$
Discount factor γ	0.99
Replay Memory size M	30000
Minibatch size B	32

We select two metrics that have been used in several DRL studies for UTC. Every metric is taken per episode. The metrics are the following:

- **Cumulative Reward**. The reward r_t taken on every time step t is accumulated until the episode is finished.
- **Average Waiting Time**. This is the sum of all W_t divided by the number of time steps in the episode. SUMO defines the waiting time for a vehicle as the time (in seconds) spent with a speed below 0.1 m/s since the last time it was faster than 0.1 m/s.

We implement and evaluate the following combinations of IDQN, experience replay, and fingerprinting, to assess impact of each component on the performance:

- **IDQN without experience replay (PEMR Disabled)**, therefore the agent cannot store experiences.
- **IDQN with prioritized experience replay (PEMR)**.
- **IDQN with prioritized experience replay and fingerprint (PEMR + FP)**. This is the proposed IDQN technique with the fingerprint to disambiguate the age of experience replays.

We evaluate the approach against the **fixed time (FT)** baseline, in which order of the phases is fixed and preconfigued; phases are actuated in a round robin manner.

Every technique is tested in the following traffic conditions:

- **Low traffic load**. 1300 to 1600 cars are simulated per 3600 time steps. With efficient traffic control the scenario is expected to result in smooth traffic flow, but any control inefficiencies will result in longer queues.
- **High traffic load**. 1900 to 2500 cars are simulated per 3600 time steps. This scenario represents the traffic load in peak times.

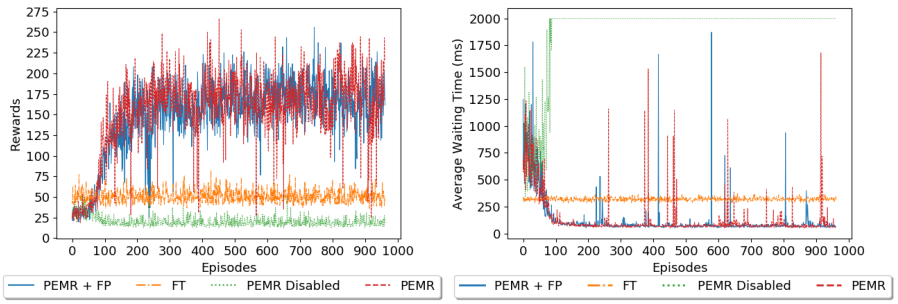
4.2 Results and analysis

The Figure 4 presents the results of the experiments executed in low and high traffic load respectively.

As can be seen from the Figure 4a, in low traffic conditions, *FT* gets a constant reward range that varies extremely low around 50 due to it not adapting its performance to the traffic conditions. *PEMR Disabled* also gets very low rewards, even worse than the FT, as it never successfully learns to adapt. Therefore, disabling the experience replay is not useful for our problem. Finally, *PEMR* and *PEMR + FP*, show identical performance, getting 400% bigger rewards than *FT*. They both learn very well in spite of the non-stationarity, increasing the rewards from around 25-100 in the exploration phase up to around 200-250 in the exploitation phase. They reach their optimal point at 300 episodes, where they get to keep a stable rewarding. Based on this similarity of results, we deduce that the fingerprint in low traffic loads is not helping.

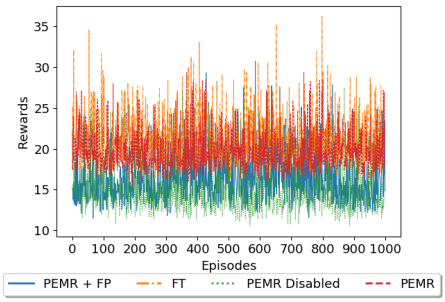
As illustrated in figure 4c, in high traffic conditions, *FT* actually performs well in terms of the reward, and in line with *PEMR*, with both outperforming *PEMR Disabled* and *PEMR + FP* by around 25%. We conclude that the extra layer of the fingerprint reduces the performance of the technique under intense traffic loads. Moreover, none of the DRL technique learn sufficiently over the episodes. These results indicate that the agents need longer training time for such a high load.

The Figure 4b presents the waiting time for low traffic. *FT* waiting time follows the same pattern as its rewards. *PEMR Disabled* produces extremely high

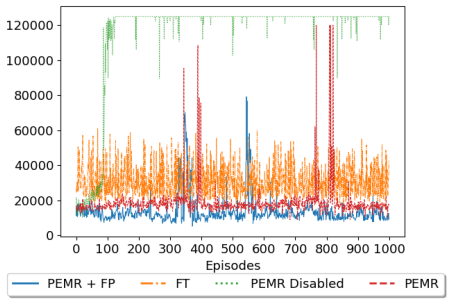


(a) Low Traffic: Cumulative Rewards

(b) Low Traffic: Average Waiting Time



(c) High Traffic: Cumulative Rewards



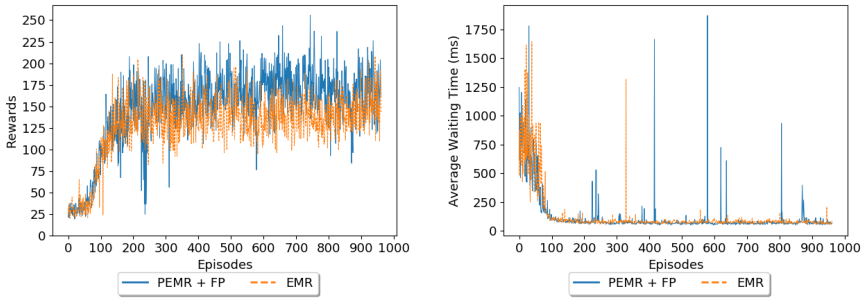
(d) High Traffic: Average Waiting Time

Fig. 4: IDQN Experiments - Results per Episode

waiting times. Similarly to their performance with respect to rewards, *PEMR* and *PEMR + FP* get similar performance, with very low waiting times which are approximately 300% less than *FT*. From these results we can confirm that the fingerprinting is not helping, as observed when analyzing the rewards. In high traffic load, the results for waiting time follow the same pattern as for the rewards in high load: *PEMR + FP* slightly outperforms *PEMR*. Overall, no DRL approach learns a good performance for high load in the amount of time given for the training.

To further analyse the results, we have looked into unexpectedly bad performance of fingerprinting. One possibility is that the fingerprint might not be able to further improve the performance as prioritized experience replay is already good enough to deal with the non-stationarity without adding fingerprints. To test this hypothesis, we run an experiment using only **normal experience replay (EMR)**. The figure 5 shows this additional experiment which compares *EMR* against *PEMR + FP*. Both techniques learn well, but *PEMR + FP* obtains higher rewards outperforming *ERM* by around 45% in the highest point of *PEMR + FP*. Figure 5b illustrates how they get a similar performance in waiting time, but *EMR* is more stable than *PEMR + FP* by not producing

peeks. We conclude the Prioritized Experience Replay helps to deal with the non-stationarity by getting bigger rewards, which indicates that *PEMR* has better performance than *EMR*. These results prove that *PEMR* can be good enough to deal with the non-stationarity. Potentially *PEMR + FP* could be improved by selecting a better fingerprint, but we leave further investigation into this for future work.



(a) Cumulative Rewards per Episode (b) Average Waiting Time per Episode

Fig. 5: Low Traffic IDQN Experiments with Standard ERM

5 Conclusion and future work

This paper presented IDQN as the first approach in literature to addresses heterogeneous multi-agent DRL in UTC. We evaluated IDQN's performance with different configurations in low and high traffic loads. Our experiments showed that IDQN is a suitable techniques for heterogeneous multi-agent UTC environments which can deal with the non-stationarity. The best results were obtained in the low traffic load. The high traffic load requires further investigation, by either fine-tuning hyperparamters to high load or allowing for the longer training time. We demonstrated that the experience replay is mandatory to learn efficiently, but that the fingerprint we chose did not enhance the prioritized experience replay. Therefore, further investigation is needed to evaluate if different fingerprints could improve the performance. Further, our technique should be evaluated against standard RL approaches, to investigate if more fine-grained sensor data enabled by the use of DRL approaches improves the performance over standard approaches.

References

1. Abadi et al, G.B.: TensorFlow: A System for Large-Scale Machine Learning TensorFlow: A system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16). pp. 265–284 (2016)

2. Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y.: OpenAI Baselines. https://github.com/openai/baselines (2017)
3. Dusparic, I., Cahill, V.: Autonomic multi-policy optimization in pervasive systems. ACM Transactions on Autonomous and Adaptive Systems (2012)
4. Foerster, J.N., Assael, Y.M., de Freitas, N., Whiteson, S.: Learning to Communicate with Deep Multi-Agent Reinforcement Learning. CoRR abs/1605.0 (2016)
5. Foerster, J.N., Nardelli, N., Farquhar, G., Torr, P.H.S., Kohli, P., Whiteson, S.: Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning. CoRR abs/1702.0 (2017)
6. van Hasselt, H., Guez, A., Silver, D.: Deep Reinforcement Learning with Double Q-Learning. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. pp. 2094–2100. AAAI'16, AAAI Press (2016)
7. Hasselt, H.V., Group, A.C., Wiskunde, C.: Double Q-learning. Nips pp. 1–9 (2010)
8. Krajzewicz, D., Erdmann, J., Behrisch, M., Bieker, L.: Recent Development and Applications of {SUMO - Simulation of Urban MObility}. International Journal On Advances in Systems and Measurements 5(3&4), 128–138 (dec 2012)
9. Liang, X., Du, X., Wang, G., Han, Z.: Deep Reinforcement Learning for Traffic Light Control in Vehicular Networks. Ieee Transactions on Vehicular Technology 1(Xx), 1–11 (2018), `https://arxiv.org/pdf/1803.11115.pdf`
10. Liu, M., Deng, J., Xu, M., Zhang, X., Wang, W.: Cooperative Deep Reinforcement Learning for Traffic Signal Control. 23rd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), Halifax 2017 (2017)
11. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature 518(7540), 529–533 (2015)
12. Mousavi, S., Schukat, M., Howley, E.: Traffic light control using deep policy-gradient and value-function-based reinforcement learning. IET Intelligent Transport Systems 11(7) (2017)
13. Richard S. Sutton and Andrew G. Barto: Reinforcement Learning, Second Edition An Introduction. MIT Press, second edi edn. (2018)
14. Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. CoRR abs/1511.05952 (2015), `http://arxiv.org/abs/1511.05952`
15. Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., Vicente, R.: Multiagent Cooperation and Competition with Deep Reinforcement Learning. arXiv pp. 1–12 (2015), `http://arxiv.org/abs/1511.08779`
16. Tan, M.: Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. In: Machine Learning Proceedings 1993, pp. 330–337. Morgan Kaufmann Publishers Inc. (1993)
17. Van Der Pol, E., Oliehoek, F.A.: Coordinated Deep Reinforcement Learners for Traffic Light Control. NIPS'16 Workshop on Learning, Inference and Control of Multi-Agent Systems (Nips) (2016)
18. Wang, Z., de Freitas, N., Lanctot, M.: Dueling Network Architectures for Deep Reinforcement Learning. arXiv (9), 1–16 (2016)
19. Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning 8(3-4) (1992)