Delft University of Technology, Bachelor Seminar of Computer Science and Engineering

Q-value reuse between state abstractions for traffic light control

Emanuel Kuhn, Jinke He, Rolf Starre, Frans Oliehoek

Delft University of Technology

Abstract

Previous research has in reinforcement learning for traffic control has used various state abstractions. Some use feature vectors while others use matrices of car positions. This paper first compares a simple feature vector consisting of only queue sizes per incoming lane to a matrix of car positions. Then it investigates if knowledge can be transferred from a simple agent using the feature vector abstraction to a more complex agent that uses the position matrix abstraction.We find that training cannot be sped up by first training an agent with the feature vector abstraction and then reusing this Q-function to train an agent with the position matrix abstraction. The simple agent does not take considerably fewer samples to converge, and the total time needed to first train the simple agent and then transfer exceeds the time needed to train the complex agent from scratch.

1 Introduction

Traffic congestion is very costly, in the EU the cost of road congestion is estimated to be equivalent to 1% of GDP [1]. Reinforcement learning is a promising method for improving traffic light controllers.

Reinforcement learning methods can take a lot of time to train. Thus approaches have been sought to reduce this burden. One approach, transfer learning, incorporates previously learned knowledge in learning a new policy[2]. Transfer learning is used to speed up learning of a target task that is similar to a source task for which an agent has already been trained. As an example, an agent trained to climb a hill in 2D can speed up training an agent to climb a hill in 3D[3]. Another approach, curriculum learning, is motivated by the idea that a difficult task can be learned more easily by first learning easier subtasks[4]. The goal is thus to reduce the total training time by using an agent curriculum instead of learning the task directly. Curriculum learning approaches include changing the tasks and environment such that subtasks get progressively harder as each subsequent agent gets trained, building on the knowledge of the previous one. Transfer learning is used in curriculum learning to transfer the knowledge between the agents trained on the different subtasks. A different curriculum learning approach varies only the internal representation of the agent instead of varying the environment[5].

Instead of changing the task at hand, this research will attempt to learn traffic control policies more effectively by varying the state abstraction over the training time. To do this an agent with a simple state abstraction is trained first, then the Q-function of this agent is reused to train an agent with a more complex state abstraction. As simple state abstraction a vector of queue sizes is used, while as complex abstraction a matrix of car positions is used.

1.1 Organisation

First background on Reinforcement learning and Q-value reuse is provided in section 2. Section 3 presents the used DQN architecture, the environment and the evaluation method. Then section 4 presents the and compares the simple and complex state abstractions. The results of Q-value reuse are discussed in section 5. Sections 6 discusses ethical aspects and reproducibility. Section 7 provides a discussion of the results and section 8 concludes and gives suggestions for further work.

2 Background

2.1 Reinforcement learning

In reinforcement learning the environment is usually modeled as a Markov Decision Process (MDP). A MDP consists of a set of states S, as set of actions A, transition probabilities $P(s_{t+1}|s_t, a_t)$, and a reward function $R(s_t, a_t, s_{t+1})$. The transition probabilities specify the probability that the next state is s_{t+1} given the current state and action. The reward function assigns a reward to each transition. The goal is then to learn a policy $\pi(s_t) = a_t$, that takes action a_t in state s_t, which maximizes the the expected discounted reward $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ with r_t the reward received at time t and γ the factor by which future rewards are discounted.

One method to solve a MDP is Q-learning [6]. In Q learning a Q value is assigned to each state action combination. The optimal Q value function then encodes an optimal policy by selecting the best action in each state: $\pi^*(s) = max_a Q^*(s, a)$. The Q values can be learned iteratively through the following update: $Q_{i+1}(x_t, a_t) = (1 - \alpha)Q_i(x_t, a_t) + \alpha(r_t + \gamma Q_t(x_{t+1}, a_{t+1}))$ with the learning rate α controlling how much the Q values are changed each iteration.

2.1.1 DQN

As learning the tabulated Q values is not feasible for a large or continuous state space, a function approximator (FA) is used to represent the Q values. A popular choice is to use a deep neural network as FA, this technique is referred to as a Deep Q Network[7] (DQN).

Instead of updating the Q values directly, the weights of the neural network are updated by optimizing a loss function. Mnih et al[7] also proposed to use experience replay[8] to train a DQN. Using experience replay, transitions observed by the agent are not used to update the Q values directly, but are instead stored in the replay memory. Each step a mini-batch of transitions is sampled from the replay memory and this mini-batch is used to train the DQN. A side effect of experience replay that will be noticeable in the graphs

presented in this paper is that the agent only starts training when the replay memory is full. The agent's behaviour is thus random until it has experienced enough environment steps to fill the memory.

2.2 Q-value Reuse

Transfer learning is a concept in in Reinforcement Learning based on the idea that knowledge learned in one task can be useful in learning another task[9]. Instead of transferring between different tasks this paper explores if transfer learning can also be useful for transferring knowledge between agents which use different state abstractions, but otherwise learn the same task.

Q-value Reuse is a specific transfer learning approach, where the Q-values of the source task are reused in the target task[9]. This is accomplished by embedding a copy of the source task's Q-function approximator in the target agent. The Q-value of the target agent is then a combination of the reused Q-function and the target task's function approximator:

$$Q(s,a) = Q_{sourceFA}(\chi_X(s), \chi_A(a)) + Q_{targetFA}{}^{1}(s,a) \tag{1}$$

Here $\chi_X(s)$ and $\chi_A(a)$ are the intertask mappings that respectively map a state and action in the target task to a state and action in the source task. Intertask mappings can be defined by hand using domain knowledge[9]. In our case defining the intertask mappings are trivial as the environment can provide the current state in both the source and target state abstraction, and the actions of the source and target task are identical.

3 Methodology

This section will first go over the architecture of the DQN agents. Then it will show the scenario used, and the metrics used for evaluation.

3.1 DQN architecture

Two different architectures are used. One with convolution layers for matrix state abstraction and one with only fully-connected layers for the feature vector abstraction.

For the matrix based state representation the architecture proposed by Mnih et al [7] is used. This architecture was used previously for traffic control by Van der Pol[10]. The input of this architecture consists of a width $\times$ height $\times$ frames matrix, with frames the number of stacked observations. There are three hidden convolution layers each followed by ReLu[11] activation, and one hidden fully-connected layer of width 512 also followed by ReLu activation. The output consist of a fully-connected linear layer with one output for each action.

For the vector based agent this architecture is modified by removing the convolution layers. The input is now of size number of features $\times$ frames. The input layer is followed by one fully connected hidden of width 512 with ReLu activation. The output layer is a fully-connected linear layer with one output per action.

[1]Note that $Q_{sourceFA}$ and $Q_{targetFA}$ are the source and target task's Q-function approximators, and $Q_{targetFA}$ should not be confused with target network of a DQN.

3.2 SUMO Environment

In order to simulate traffic SUMO[12] is used as the environment. SUMO simulates individual cars in a scenario, and exposes an API to control the traffic lights.

The scenario used in this paper is a single intersection, with traffic generated from two directions. This one intersection scenario was chosen, as initial results suggested that even in this simple scenario there is a performance difference between the tested state representations.

In the scenario traffic load is parameterized by the car probability (car_pr). The chosen car_pr of 0.8 means that at every time step there is an 80% chance of adding a car route. This relatively high traffic load was chosen as it seemed that at this traffic load the behaviour of the traffic light had the largest influence on performance.

Figure 1 shows a screenshot of SUMO running the scenario, giving an indication of the traffic load and size of the used scenario. The snapshot was taken in the middle of an episode, with the light controlled by SUMO's default fixed phase plan.

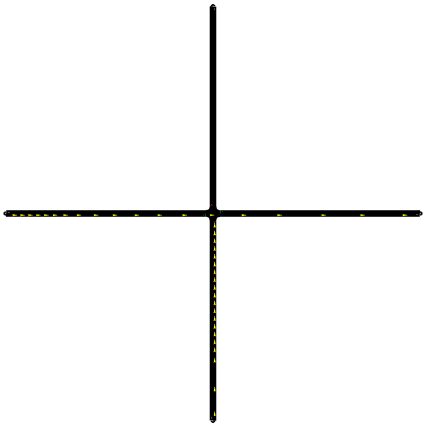


Figure 1: SUMO scenario

3.3 Action space

The same action space as used by Van der Pol[10] is used. This action space consists of the choice between the two green phases: either the horizontal lane or the vertical lane gets a green light. The yellow phases are handled by the environment: whenever the agent chooses a different action than in the last time step, the environment first sets the lights to yellow and only switches after the yellow duration.

3.4 Evaluation

For simplicity the results of training episodes, including the effects of epsilon greedy exploration are evaluated. In order to also get information about the greedy performance of the agents, the behaviour of the trained agent is inspected manually for some comparisons. The plots show the mean along with the standard error of the mean. A moving average is used to smooth out the stochasticity and make analysis easier.

3.4.1 Metrics

The abstractions are compared both in terms of the episodic return achieved by the agent and the number of timesteps needed to finish an episode. The number of steps acts as a proxy for throughput, as it is the number of steps an agent needs to let the same amount of cars[2] pass through the intersection.

[2]Actually only the average number of cars stays the same, as cars are routed with a probability. Using a moving average in the plots smooths the effect of this stochasticity.

4 State abstractions for Traffic control

To answer if first learning the task of controlling an intersection with a simple state abstraction and then transferring this knowledge to learn the same task with a more complex state abstraction can speed up learning, it first needs to be shown whether there is a performance difference between the abstractions.

4.1 SimpleState

As simple state abstraction a vector containing only queue size for each incoming lane was chosen. This abstraction will be referred to as SimpleState. This seemed to be the smallest representation that contains information about the amount of traffic per lane. Intuitively this amount of information should not be enough to control a traffic light optimally. Firstly the agent does not know the exact location of the cars, and thus cannot anticipate on a gap in the traffic. Secondly without knowing the current traffic light state the agent also cannot flip the phase as it does not know what the current phase is. Nonetheless it would be interesting to find out if this agent can still learn some policy, and if this policy can be used as a heuristic to learn a policy using a state representation containing more information.

Previous work by Li et al[13] also used only queue size per lane in their state abstraction. Their action space differed however from the one used in this paper. Their action space consisted of the choice to either stay on the current traffic phase, or switch to the other phase. Furthermore they imposed a minimum green duration and only executed the switch action after this minimum time. This action space simplifies the problem compared to the approach taken in this paper, as the agent only needs to learn when the state should be switched, and not which phase should be chosen.

Figure 2: SimpleState convergence for learning different rates

One notable thing to note is that the learning rate had a large influence on if an agent using SimpleState converged. Initially a learning rate of 2.5×10^{-4}, taken from the fine tuned agent by Van der Pol[10] was used for both MatrixState and SimpleState. As shown in figure 2, the SimpleState agent only converged with a lower learning rate of 2.5×10^{-5}.

4.2 MatrixState

As complex state abstraction we use the matrix based state representation proposed by Van der Pol[14]. This abstraction consists of a matrix of ones at positions with a car, and zeros at positions without a car. The same matrix also contains the traffic light information. Each traffic light color has a unique value associated with it. For each traffic light the value value corresponding to its state is then added to the matrix at the position of the traffic light.

As a lower learning rate had a large positive effect for

Figure 3: MatrixState convergence for learning different rates

SimpleState, an experiment with lower learning rate
for MatrixState was carried out. Figure 3 shows that
while both learning rates resulted in the same asymptotic performance, the lower learning rate seemed to converge slightly quicker.

4.3 Queue size and phase

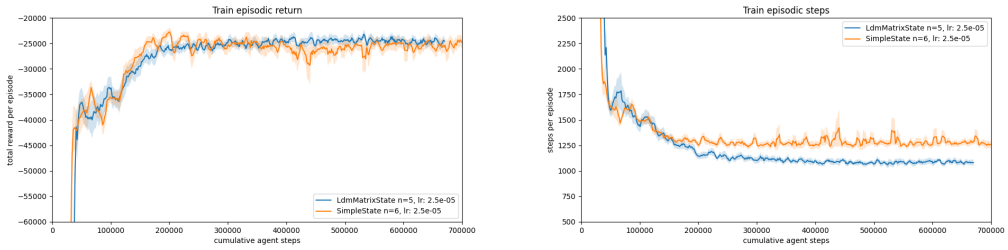
A state abstraction containing both the queue sizes per lane and the current traffic light phase is also evaluated. Including this state representation in the comparison will show if the lack of the current traffic light state can explain the performance difference between the SimpleState and the MatrixState.

The current traffic phase is encoded as one hot vector. Each element in this vector corresponds to a traffic phase configuration. Only the element corresponding to the current phase is one and the other elements are zero. The queue-size-phase state abstraction then consists of a vector of queue sizes appended with the one hot vector encoding of the current phase.

This abstraction is similar to that used by Rasheed et al [15]. Their state representation also includes the time since the last phase switch, for each neighbouring intersection the sum of queue sizes, and the current rainfall intensity. The last two features are however not relevant for the scenario used in this paper.

4.4 Performance comparisons

4.4.1 Matrix state vs SimpleState



(a) Episodic return

(b) Number of steps

Figure 4: SimpleState vs MatrixState

Figure 4 shows the results of comparing the SimpleState abstraction and the MatrixState abstraction. Surprisingly the episodic return of both state abstractions are similar. There is a clear difference in the number of steps however, indicating that the policy learned by SimpleState is worse in terms of actually controlling traffic.

The SimpleState agent does not converge in fewer environment steps than the MatrixState agent, at least when both use the lower learning rate of 2.5×10^{-5}.

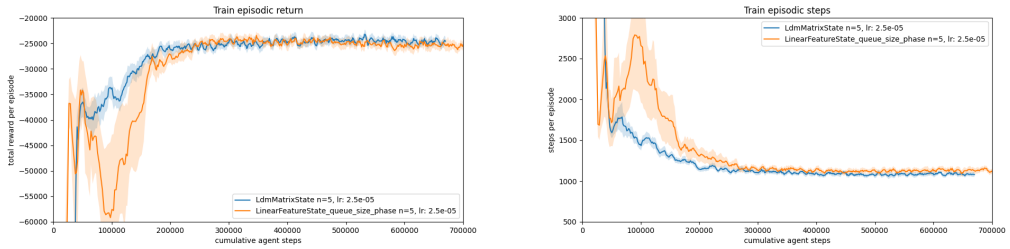
Manual inspection of the policies also shows that the MatrixState agent behaves reasonably. The traffic light switches when one queue gets exhausted of cars. Then the light only switches back when the other queue exhausts.

In contrast, the SimpleState agent behavior did not look very reasonable. The solution the agent found to not knowing the current phase, seemed to be to favour one direction and only switch when this direction gets very empty. Then as soon as the favoured direction filled up slightly the phase was switched to favoured direction. This lead to often only one car at a time being let through the unfavoured lane.

4.4.2 Queue size and traffic phase

Figure 5 shows the performance of using queue-size-phase, the abstraction consisting of the current traffic phase in addition to queue sizes per lane. It seems that adding the current phase to the state in addition to the queue sizes improved performance to the level of the MatrixState baseline.

Manual inspection showed that the asymptotic behaviour of queue-size-phase is similar to that of MatrixState. It did however take more training steps for queue-size-phase to perform as well as MatrixState. The MatrixState agent behaved almost as well after 300k training steps as it did after 600k training steps. The queue-size-phase agent however performed suboptimally during the first 600k episodes. The agent after 600k training steps also sometimes seemed to not recognize a single car. The last car of the episode would sometimes get stuck because the phase wasn't switched.



(a) Episodic return

(b) Number of steps

Figure 5: queue-size-phase vs MatrixState

5 Q-value Reuse

In this section the results of reusing a learned SimpleState Q-function to train an agent using the MatrixState abstraction will be discussed. The Q-value reuse experiment is repeated for source policies trained for different amounts of time. This to show the correlation between the number of steps trained with the source state abstraction and the training curve of the target abstraction.

5.1 Source policy

A simple state agent was trained for 200 episodes. During training the weights of the model were saved after every 50k training steps. Additionally the untrained weights at 0 training steps and the final weights after 290k training steps were also saved for the Q-value reuse experiment.

5.2 Evaluation metrics

There are many metrics to evaluate the performance of transfer learning methods[9]. These metrics can be divided into two groups: metrics that treat the time spent on the source task as a sunken cost and metrics that take the full training time into account. The metric that will be used to answer the research question if the total training time can be shortened by first learning a simpler abstraction, is one that takes the full training time into account: the time-to-threshold measure. Time-to-threshold asks what total training time is needed to reach a threshold performance. Taylor et al[3] also argue that in a transfer setting explicitly guided by a human with the aim to shorten total learning time, as was the aim of this paper, a metric that takes total training time into account is most sensible.

Metrics which treat learning the source task as a sunken cost include asymptotic performance, jumpstart, and comparing the total reward under the learning curve. Although a jumpstart is visible in most training curves, this initial high performance is not a very useful metric for Q-value reuse. This because at the start of training with Q-value reuse the new Q-function is equal to the reused Q-values plus the output of a randomly initialized DQN. Thus the initial policy will be close to the policy of the reused agent and does not give a useful indication. Total reward is however an interesting secondary metric, as it can show if learning is improved or hindered by Q-value transfer.

5.3 Results

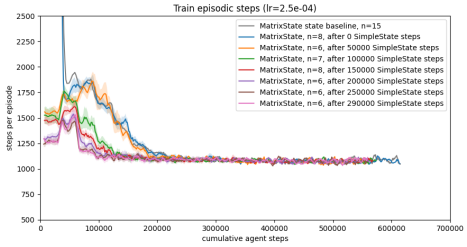
5.3.1 Transfer from SimpleState to MatrixState

Figure 6 shows the learning curves of MatrixState with Q-value reuse of a SimpleState agent trained for different amounts of steps. The experiment was performed using two different learning rates for the target MatrixState agent. Both show similar behaviour, and seem to converge around the same time.

The general observation seems to be that the initial performance is high for the first 30k steps during which no training occurs. Then there is a large drop in performance, after which the performance increases again. The size of this drop seems to depend on the amount of steps spent learning in the source task.

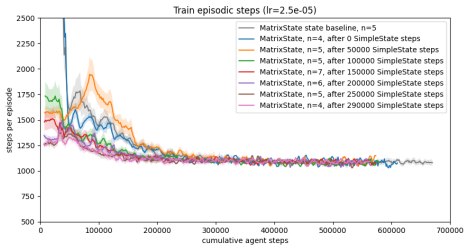
Furthermore all learning curves for which the source task was trained at least 100k steps seem to have a larger total reward caused by the reward increasing faster after the dip compared to the baseline. Q-value reuse of the SimpleState agent thus does have a positive effect on the training time of the MatrixState agent.

(a) Episodic return (lr=2.5×10^{-4})



(b) Number of steps (lr=2.5×10^{-4})

(c) Episodic return (lr=2.5×10^{-5})



(d) Number of steps (lr=2.5×10^{-5})

Figure 6: Q value reuse SimpleState to MatrixState without source cost

An explanation for the dip could be that the policies the different agents converge to differ a lot as noted in section 4.4.1. That the policies differ implies that the Q-values also differ. It could be that during this dip the policy slowly changes from the one SimpleState converged to to the one MatrixState converges to.

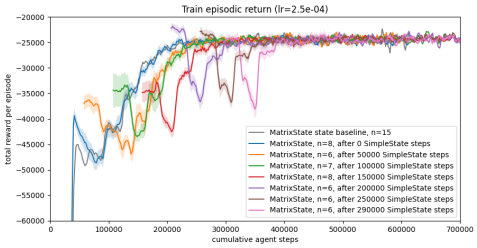
This does not however explain why the dip is shorter with a source policy trained for longer. It could be that a longer trained source policy is more consistent in terms behaviour, which might make it easier to learn the target DQN. This since the Q-value reuse equation can be rewritten to show that the target agent should learn the difference between optimal Q values function and the reused Q-function:

$$Q_{targetFA}(s,a) = Q(s,a) - Q_{sourceFA}(\chi_X(s), \chi_A(a)) \tag{2}$$

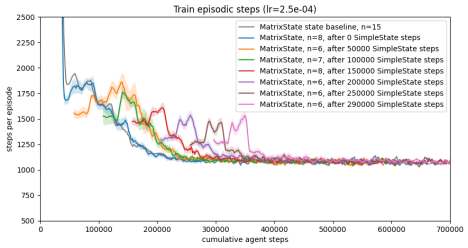
Thus implying that a more consistent source Q-function would lead to the the target Q-function being easier to learn.

Another hypothesis is motivated by that Q-value reuse can be considered a type of reward shaping[9]. It might be reasonable to assume that a better source policy as shaping reward is more useful than a bad one. Then again, the policy the SimpleState agent converges to the end is also pretty bad, but still good enough to provide a positive effect compared to training without it.

Looking at figure 7, it is clear that after the drop, there exists no threshold performance



(a) Episodic return



(b) Number of steps

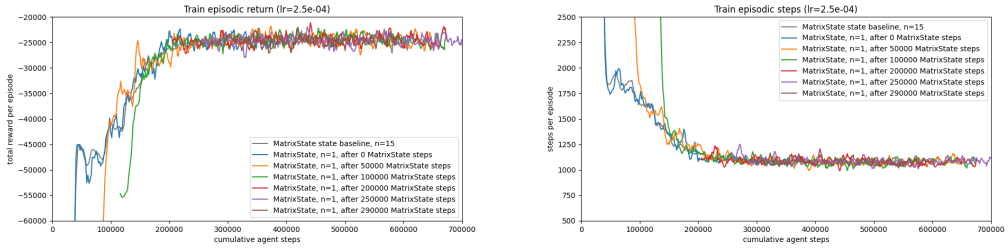
Figure 7: Q value reuse SimpleState to MatrixState with source cost

that is reached earlier by the agent employing Q-value reuse compared to the baseline agent trained without Q-value reuse.

5.3.2 Transfer from MatrixState to MatrixState

As sanity check and to show that Q-value reuse does not have an inherent performance penalty when applied in this context, the transfer experiment is also carried out from the matrix based representation.

Figure 8 shows that the training curves with Q-value reuse are surprisingly close to the baseline. The curves with reuse seem to only take roughly 50k agent steps before continuing along the same trajectory as the baseline, the outlier being the curve transferred after 150k agent steps. This is slightly surprising as only the Q-value is transferred and the weights of the new Q-function are all randomly initialized. One might expect that having to relearn the convolution filter in the middle of training would have a negative impact on the number of samples needed, this seems to not have had a large effect.



(a) Episodic return

(b) Number of steps

Figure 8: Q value reuse MatrixState to MatrixState

6 Responsible Research

6.1 Ethical concerns

The experiment were conducted using simulations and no data collected from the real world was used. Thus there seem to not be any relevant ethical or privacy concerns with the experiments conducted in this paper.

If traffic control through reinforcement learning algorithms were to be used in the real world there would be ethical concerns that need addressing. Traffic is inherently something dangerous because of the high speeds cars move at. Currently most machine learning and reinforcement learning algorithms can't give any guarantees as to how they will behave. As a faulty traffic light could cause accidents, necessary save guards should be put in place before employing a RL algorithm in this context.

6.2 Reproducibility

Care went into correctly collecting the results presented in this paper. Each episode was seeded uniquely such that every seed was only used for one episode. These seeds were

collected as part of the results and are available on request along with the results and code used in this paper. The results should thus be fully reproducible.

7 Discussion

The curriculum learning approach of first learning the task of traffic light control with a simple state representation, and then using this knowledge to speed up learning with a more complex state abstraction was not successful. The main reason for this is that the assumption that a simpler state representation might converge to a policy quicker did not hold.

The comparison of queue-size-phase vs MatrixState showed the performance of a simple state abstraction can be similar to using a matrix of car positions. This result is consistent with the findings of Genders et al[16] which found that performance in terms of throughput and queue size was equal across state abstraction. A limitation of both their work and this paper is that the scenario used is a single intersection. Future work could explore if the performance stays equal in more complex environments.

8 Conclusions and Future Work

We found that using an agent using queue sizes as state representation can be trained to perform traffic control if the state representation also includes the current traffic phase. Both using only the queue sizes and using queue size and current phase did not result in faster learning compared to using a matrix as state representation.

Q-value reuse seems to be effective in transferring knowledge between different state abstractions. Even using the suboptimal Q-value function trained using only the queue sizes and not the current phase as source Q-function, training performance of the target agent which uses the position matrix as state was still improved compared to training from scratch. This scheme did not result in lower total training time however.

Future work could look into extending Q-value reuse to a multiagent setting. Whereas previous work tackles a multi agent scenario by explicitly creating a factor graph and applying a belief propagation algorithm to optimize for the joint global action[14][17], another approach could be to use Q-value reuse via TVITM (transfer via inter-task mapping). The Q-values of individual intersections can be mapped to the global task, and the global agent could try to learn the difference between the reused local Q values and the globally optimal Q value. It should be noted that this would still only be feasible with a relatively small number of agents, as a global action space explodes quickly in size with more agents.

References

[1] Panayotis Christidis, J Nicolás Ibañez Rivas, et al. Measuring road congestion. *Institute for Prospective and Technological Studies, Joint Research Centre, Brussels*, 2012.

[2] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

[3] Matthew E Taylor. *Transfer in Reinforcement Learning Domains*. Springer Berlin Heidelberg, 2009.

[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[5] Wojciech Czarnecki, Siddhant Jayakumar, Max Jaderberg, Leonard Hasenclever, Yee Whye Teh, Nicolas Heess, Simon Osindero, and Razvan Pascanu. Mix & match agent curricula for reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1087–1095. PMLR, 10–15 Jul 2018.

[6] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[7] 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.

[8] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[9] Matthew E Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(Sep):2125–2167, 2007.

[10] Elise van der Pol. Deep reinforcement learning for coordination in traffic light control. Master's thesis, University of Amsterdam, 2016.

[11] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[12] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. Sumo–simulation of urban mobility: an overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.

[13] L. Li, Y. Lv, and F. Wang. Traffic signal timing via deep reinforcement learning. *IEEE/CAA Journal of Automatica Sinica*, 3(3):247–254, July 2016.

[14] Elise Van der Pol and Frans A Oliehoek. Coordinated deep reinforcement learners for traffic light control. *Proceedings of Learning, Inference and Control of Multi-Agent Systems (at NIPS 2016)*, 2016.

[15] Faizan Rasheed, Kok-Lim Alvin Yau, and Yeh-Ching Low. Deep reinforcement learning for traffic signal control under disturbances: A case study on sunway city, malaysia. *Future Generation Computer Systems*, 2020.

[16] Wade Genders and Saiedeh Razavi. Evaluating reinforcement learning state representations for adaptive traffic signal control. *Procedia computer science*, 130:26–33, 2018.

[17] Lior Kuyer, Shimon Whiteson, Bram Bakker, and Nikos Vlassis. Multiagent reinforcement learning for urban traffic control using coordination graphs. In *Joint European*

Conference on Machine Learning and Knowledge Discovery in Databases, pages 656–671. Springer, 2008.