Intelligent signalized intersection management for mixed traffic using Deep Q-Learning

Master thesis submitted to Delft University of Technology

in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

in **Complex Systems Engineering and Management**

Faculty of Technology, Policy and Management

by

Chantal Schneider

Student number: 4930835

To be defended in public on August 21st, 2020

Graduation committee

Chairperson:	Dr. M.E. Warnier, Systems Engineering Section
First Supervisor:	Dr. J.A. Annema, Transport and Logistics Section

EXECUTIVE SUMMARY

Traffic congestion has negative economic, environmental and societal impacts. In 2017, US commuters aggregated a national travel delay of 8.8 billion hours, which translated to a congestion cost of 179 billion dollars. 3.3 billion gallons of fuel were wasted in traffic jams, which increased air pollution and CO2 emissions. Furthermore, higher congestion levels were linked to higher accident frequencies. Since traffic volumes are ever-increasing, congestion is expected to worsen by another 60% until 2030.

One of the causes of urban congestion are badly timed traffic signal controllers. In the US alone, traffic signals were estimated to cause 5-10% of all traffic delays or 295 million vehicle-hours. To improve traffic signal plans, research efforts have been made to create self-adaptive traffic controllers, i.e. controllers which adapt in real-time to the current traffic demand.

To observe the current traffic situation, it was proposed to use connected vehicle technology. Vehicle connectivity enables vehicles to communicate vehicle trajectory information (e.g. speed, position, planned route), as well as information on road and traffic conditions in real-time with the traffic signal controller or other vehicles. The controller can gather data from all vehicles around the intersection and use this to create optimal signal plans.

Within literature, different types of methods to create self-adaptive traffic signal controllers were found. In this thesis, it was chosen to focus on deep Q-learning models (DQN), which are a type of reinforcement learning model. Reinforcement learning (RL) models are able to overcome shortcomings of other methods: they are applicable in real-time and they do not require researchers to create a complex mathematical model (i.e. they are model-free).

Past research on self-adaptive controllers has mostly assumed that all of the vehicles are connected. Yet, at least for the close future, this remains an unrealistic assumption. Instead, traffic will consist of a mixture of regular vehicles (RVs) and connected vehicles (CVs). Up to date, not many studies investigated whether self-adaptive controllers in general, and deep Q-learning based controllers in particular, are able to control traffic signals efficiently even under different CV-penetration rates. This thesis aims to alleviate this gap by investigating whether DQN-based traffic signal controllers are able to reduce traffic congestion for mixed traffic scenarios, and what design choices have to be made to build such a controller. As such, this thesis aimed to answer the research question: *Can deep Q-learning models be used to control signalized intersections in mixed traffic scenarios such that traffic congestion is reduced?*

In order to answer this research question, it was first investigated what decisions have to be made when building a new DQN controller, and how past studies have made these decisions. This thesis conducted a systematic literature review in which the most important design choices regarding agent design (state, action and reward representations, model extensions), traffic environments (network topologies, traffic scenarios) and model evaluation (base cases, KPI) were reviewed.

Another part of the review focused on reinforcement learning in mixed traffic scenarios. Less research has been done on this topic. For mixed traffic situations, two types of agents were found in literature: vanilla DQN agents and recurrent DQN agents. The results show that even for low CV-penetration rates, both model types can successfully reduce traffic congestion compared to traditional traffic signal controllers.

Yet literature gaps remain. Currently, no clear best practices exist that could guide researchers on how to design new agents. Many studies do not or only partially describe why they made certain design choices and how they calibrated their agents, which makes it difficult for future researchers to learn from their experiences. Overall, it turns agent design and fine-tuning into a trial-and-error process.

This thesis contributed to mitigating this problem by applying a more structured approach to agent design. It was decided to build two types of deep Q-learning controllers, one vanilla DQN agent and one recurrent DQN agent. First, two base agents were designed which combined design choices from previous studies in novel ways. Yet, since the agents would be tested on different types of scenarios than in those previous studies, it would not be guaranteed that these choices would be optimal. To solve this, each of the unclear design choices were investigated in experiments. Within these experiments, each parameter was investigated using a new base model. In this base model, all settings were identical, apart from the parameter under investigation. This allowed assessing how the different alternatives for every parameter impacted both agent stability and performance. The alternatives which resulted in the best stability and highest performance were chosen. Based on this result, a new base agent could be designed, which could then be used to experiment with the next parameter.

Doing this, agents in this thesis were calibrated in a systematic manner. Yet, when comparing the results of the calibration experiments to literature, it was found that the optimal design choices and parameters were often different than in previous studies. This indicates that optimal parameters for an agent depend to a large extent on the overall agent design and also on the training and testing environment (e.g. traffic scenarios, intersection layout). This makes it difficult to derive more general best-performing settings. In the end, fine-tuning the agents remained a time-consuming trial-and-error process.

After the two agents were fine-tuned, they were evaluated using a microscopic traffic simulator. The goal was to investigate how stable the agents' performance is, how well the agents can efficiently control traffic signals under different traffic scenarios (low constant, medium constant, high constant and dynamic traffic) and CV-penetration rates (between 10% and 100%), how robust agents are to changes in penetration rates, and how the vanilla and recurrent agents differ. To be able to benchmark the two agents' performances, they were compared to a traditional fixed-time controller.

It was found that the designed vanilla agent was unable to reduce traffic congestion in mixed traffic scenarios. Many hypotheses were unsupported: neither agent stability nor agent performance increased with higher penetration rates and vanilla agents were unable to outperform fixed-time controllers.

Recurrent agents however performed very well, with all hypotheses from section 8.1.3 being supported. They were relatively stable and performed well even under low penetration rates. The experiments also showed that higher penetration rates led to more stability and higher performance across all traffic scenarios. Additionally, the recurrent agent was found to be robust to penetration rate changes: the recurrent agent trained under 50% CV-penetration was able to perform well for CV-penetration rates between 30%-100%. Furthermore, the recurrent agent was able to outperform the fixed-time controller for most of the penetration rates and scenarios. The required CV-penetration rate to outperform the fixed-time controller was between 0% (medium traffic) and 40% (low and dynamic traffic). This indicates that while the required CV-penetration rate is relatively low across scenarios, no critical transition penetration rate can be determined since this largely depends on the type of scenario. When comparing the scenarios, it was found that for scenarios with fewer cars, the recurrent agent required a higher CV-penetration rate to reach a good level of performance than for scenarios with many cars. Moreover, the recurrent agent was able to perform better in constant traffic scenarios than in dynamic traffic scenarios.

When comparing the results of the mixed traffic experiments to literature, not all findings from past research were supported. A major difference between the results was that in past research, vanilla DQN agents have been shown to be able to successfully control traffic signals even for mixed traffic situations, while the vanilla agent in our study was unable to do so. For the recurrent agent however, both this thesis and past research concluded that recurrent agents are able to significantly outperform fixed-time controllers, even under low penetration rates. Further similarities were that higher penetration rates lead to higher performance, and that recurrent agents are up to a certain degree robust to changes in penetration rates. Unlike past research, this thesis found that for all traffic scenarios agent performance improves with increasing penetration rates. Previous studies found that this was only the case for low and medium traffic scenarios, but not for high traffic scenarios.

Based on the experiments it can be concluded that the designed vanilla DQN agent is unsuitable for mixed traffic control. The recurrent DQN agent on the other hand, performed better than the vanilla agent in terms of stability, performance and robustness, and only the recurrent agent was able to outperform the fixed-time controller for all but the lowest penetration rates. This makes recurrent DQN a promising method for future research on reinforcement learning-based traffic signal control.

In brief, this thesis found that reinforcement learning algorithms could potentially be used to control signalized intersections in mixed traffic scenarios such that traffic congestion is reduced. Nevertheless, this thesis only resulted in a proof-of-concept. Experiments were conducted under simplified conditions and results are subject to several limitations, which may compromise the results. In order to answer the main research question with certainty, recurrent DQN controllers (and reinforcement learning controllers in general), will need to be further developed, calibrated, and tested under more realistic circumstances.

The thesis concludes with several recommendations for practice and future research. Since reinforcement learning models are currently still in the research phase and not ready for real-life implementation, it is currently too early for policy-makers to take action, apart from providing sufficient funding. First, controllers need to become more mature and must be validated on more types of scenarios under more realistic circumstances before they can be considered for real-life implementation.

To make reinforcement learning agents more mature, several recommendations for future research can be made. Future research could for example investigate whether other reinforcement learning models than deep Q-learning could gain better results. Additionally, more research is needed on how agents can be implemented that consider other goals besides traffic efficiency. Also, it is suggested to investigate the robustness of reinforcement learning agents when they are implemented in more realistic conditions.

Furthermore, the design process to build new reinforcement learning agents could be improved. In many published studies, authors did not justify their design choices and did not describe how they calibrated their agents. For new researchers this means that they cannot benefit from these previous experiences, and that agent design remains a time-consuming trial-and-error process. In order to make this process faster, it is recommended to further investigate how different design choices and different parameters impact agent performance and stability. This could for example be done by means of systematic experiments in which the impact of every setting is analyzed separately. To facilitate this process better, it is recommended to develop standardized testing environments and traffic scenarios that allow for more systematic cross-comparisons between studies and benchmarking of well-performing agents.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

(A)NN	(Artificial) neural network
AV	Automated vehicle
CAV	Connected and automated vehicle
CNN	Convolutional Neural Network
CV	Connected vehicle
DQN	Deep Q-network
DRQN	Recurrent Deep Q-network
DTSE	Discrete traffic state encoding
KPI	Key performance indicator
LSTM	Long short term memory
MDP	Markov decision process
MLP	Multi-layer perceptron
MSE	Mean squared error
POMDP	Partially observable Markov decision process
ReLu	Rectified Linear Unit
RL	Reinforcement learning
RV	Regular vehicle
SUMO	Simulation of Urban Mobility
TSC	Traffic signal control
V2I	Vehicle-to-infrastructure
V2P	Vehicle-to-pedestrian
V2V	Vehicle-to-vehicle
V2X	Vehicle-to-everything

List of Notations

General Reinforcement Learning notations

α	Learning rate
$a \epsilon A$	Set of actions
$s \epsilon S$	Set of states
$r \in R$	Set of rewards
t	Time step
$\pi(s)$	Policy / Strategy in state s
γ	Discount factor
T	Transition function
$V(s)$	Value (state value function)
$Q(s, a)$	Q-value (state-action value function)
θ	(Online) neural network weights;
θ^{-}/θ^{T}	Target neural network weights
ε	Exploration rate
β	Target network freeze interval

Conceptual model and KPI specifics

N, S, E, W	North, south, East, West
$NSG, NSLG, EWG, EWLG$	Set of 4 possible actions (North/South advance; North/South Left-turning advance; East/West advance; East/West Left-turning advance)
t	Action time step
D	Delay
T	Simulation time step
d	Distance driven
v	Speed
W	Waiting time
Q	Queue-length
$e \in E$	Set of episodes
$i \in I$	Set of vehicles in episode during the current time step

1. Introduction

1.1 Problem Introduction

Traffic congestion creates far-reaching economic, environmental and social externalities. In 2017, commuters in the US aggregated a national travel delay of 8.8 billion hours, which translated to a congestion cost of 179 billion dollars (Schrank, Eisele, & Lomax, 2019). 3.3 billion gallons of fuel were wasted in traffic jams, which increased air pollution and CO2 emissions. Furthermore, higher congestion levels were linked to higher accident frequencies (G. L. Chang & Xiang, 2003; Marchesini & Weijermars, 2010), thus leading to ten thousand traffic-related fatalities (Florin & Olariu, 2015).

Due to increasing traffic volumes, externalities increased over time: the annual delay due to traffic congestion increased per auto commuter in the US from 20h in 1982, to 54h in 2017, an increase of 270% (Schrank et al., 2019). Since traffic congestion is forecasted to increase an additional 60% by 2030 (Rafter, Anvari, & Box, 2018), economic losses and social and environmental problems are expected to worsen.

One of the causes of congestion are urban intersections. Urban intersections critically impact traffic delays as well as accident rates (Rahmati & Talebpour, 2017). Intersections are bottlenecks for traffic (L. Chen & Englund, 2016), since traffic streams with conflicting paths have to be coordinated, leading to a decrease in road capacity.

To control incompatible traffic flows and guarantee safety at busy urban intersections, traditionally traffic signals are used. Yet, badly timed traffic signal plans (i.e. inappropriate traffic phase sequences and traffic signal splits) can negatively impact traffic flows (Du, Shangguan, Rong, & Chai, 2019). If successive traffic lights are not adjusted to each other, cross-blocking can occur. Cross-blocking means that vehicles at one intersection cannot cross the intersection despite having a green signal, since downstream lanes are completely occupied due to a red signal at the downstream intersection. At the other extreme, green idling may occur. This means that certain lanes of the intersection have a green signal, despite there being no cars that need to cross in those lanes. Both cross-blocking and green idling prevent other cars that currently have a red signal to needlessly wait. Oftentimes congestion at one intersection causes domino effects to other intersections, and thus affects wider network performance (Yau, Qadir, Khoo, Ling, & Komisarczuk, 2017). Traffic signals are especially inefficient at high traffic volumes (Shi, Jiang, & Li, 2016). In the US alone, traffic signals were estimated to cause 5-10% of all traffic delays, or 295 million vehicle-hours (Denney, Curtis, & Olson, 2012).

To improve traffic efficiency, traffic signals must react to traffic demand (Jing, Huang, & Chen, 2017). A recent development believed to aid with this are automated and connected vehicles (L. Chen & Englund, 2016). Vehicle connectivity enables vehicles to communicate vehicle trajectory information (e.g. speed, position, planned route) and information on road and traffic conditions in real-time with the traffic signal controller or other vehicles (Catapult Transport Systems, 2017; L. Chen & Englund, 2016). Based on this data, traffic signal control can be optimized.

Within the field of cooperative intersection management, the final goal is to create intersections in which vehicles communicate with infrastructure and other vehicles to negotiate safe and efficient passing strategies that require no human interventions (L. Chen & Englund, 2016). If a 100% CAV penetration rate were to be reached, all traffic signals and signs could be replaced with intersection management systems that optimize traffic flows based on communicated real-time data (Wuthishuwong & Traechtler, 2017).

Researchers suggest this could make intersections very efficient, thus causing minimal traffic delays (Sousa, Almeida, Coutinho-Rodrigues, & Natividade-Jesus, 2018).

However, a 100% CAV penetration rate is unrealistic within the close future (Overtoom, 2018). Instead, there will be a transition period in which conventional, automated and connected vehicles (i.e. mixed traffic) will co-exist. This requires control strategies that can efficiently and safely manage different vehicle mixtures (Kamal, Hayakawa, & Imura, 2020).

Up until now, many intelligent intersection control strategies were proposed. The idea of using vehicle connectivity to create smart intersections was first proposed by Huang & Miller (2004), and the first intersection controller was created by Dresner and Stone already in 2004. Nevertheless, despite numerous studies in this field, only few studies investigated signal control strategies for mixed traffic.

Additionally, the existing methods are subject to other limitations, as has been documented in various literature reviews (e.g. Chen & Englund, 2016; Florin & Olariu, 2015; Guo, Li, & Xuegang, 2019; Jing et al., 2017; Li, Wen, & Yao, 2014; Namazi, Li, & Lu, 2019; Rios-Torres & Malikopoulos, 2017). Existing control strategies are based on often-unrealistic assumptions, have not yet been extensively validated and their large-scaled impacts remain unresearched.

1.2 Research objective

The second chapter of this thesis discusses relevant background information on traffic signal control and provides more information on the identified research gap. Chapter 3 further scopes down the topic, identifies the main and sub-research questions and presents the research approach.

The main conclusions of these chapters can be summarized as follows: up to date, many intelligent traffic signal control methods have been proposed, but only few studies have investigated whether and how these models can efficiently control traffic signals during the transition period. This thesis aims to alleviate this gap by investigating whether traffic signal controllers are able to reduce traffic congestion for mixed traffic scenarios, and what design choices have to be made to build such a controller.

Since different model types exist, it is not possible to evaluate design choices for all types of models. Instead, it was chosen to focus only on reinforcement learning approaches, which are oftentimes mentioned as a promising traffic signal control approach. More specifically, the most popular type of reinforcement learning-based model, deep Q-learning was chosen.

Finally, the main research question which will be answered within this thesis is as follows:

> ***Can deep Q-learning models be used to control signalized intersections in mixed traffic scenarios such that traffic congestion is reduced?***

1.3 Thesis outline

The thesis is structured as follows: chapter 2 defines core concepts and reviews the field of intelligent traffic signal control. Based on this review, limitations of existing studies are identified, and the research gap and goal of this thesis are introduced. Chapter 3 further scopes down the research topic, presents the research approach and defines the research questions. Chapter 4 presents relevant background knowledge on the chosen control method (reinforcement learning) which will be needed to understand all further design decisions. Chapter 5 presents an in-depth literature review on the state-of-the-art of reinforcement learning in traffic signal control. Based on the findings in this review, chapter 6 develops a conceptual model. Chapter 7 presents the implementation details of the developed controller. In chapter 8, the experiments are described. In this chapter, the methodology is summarized, then hypotheses regarding the expected results are stated, and finally the results of the experiments are analyzed. Chapter 9 discusses the results, evaluates model limitations, suggests possible model improvements and embeds the results in existing literature. Finally, chapter 10 concludes the thesis by answering the research questions, stating the scientific contributions, highlighting the link to CoSEM and providing recommendations for practice and future research.

The code used in this thesis is available on GitHub[1].

[1] https://github.com/chantal000/Deep-QLearning-Agent-for-Traffic-Signal-Control

2. RESEARCH GAPS AND RESEARCH GOAL

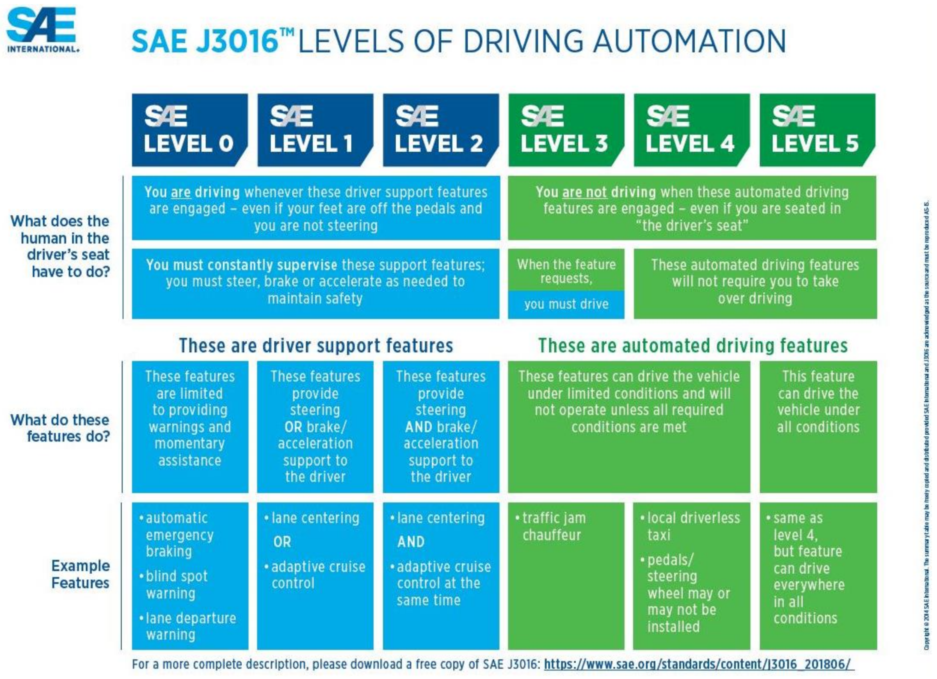
This section aims to identify literature research gaps to define a clearer research goal. To provide the reader with needed background knowledge, first some core concepts are defined, and intersection management is introduced. To identify specific research gaps, a literature review on intelligent signalized intersection management has been conducted. For the literature review, the review methodology, the results and the found research gaps are discussed. Lastly, the research goal is presented.

2.1 Definition of core concepts: connected and automated vehicles

Within literature many terms regarding intelligent vehicles are used interchangeably: connected, autonomous or automated vehicles, smart or intelligent driving, driverless cars or self-driving cars (Catapult Transport Systems, 2017; Elliott, Keen, & Miao, 2019; H. Zhang, Tam, & Shi, 2002). Oftentimes, researchers imply either one of or both of two different technologies: automation and communication. To avoid confusion, the terms used throughout this thesis must be clearly defined. The next section discusses both technologies separately, before discussing their combination.

2.1.1 Automation

The goal of automation is for vehicles to drive fully autonomously, i.e. the car can drive by itself without any input by humans. This goal can be reached by equipping vehicles with an array of sensors such as cameras, radars, LiDAR, lasers, ultrasonic sensors and GPS (Guanetti, Kim, & Borrelli, 2018). However, as controversy around the term "autonomous" exists, the term "driving automation" is preferred (Vagia & Rødseth, 2019).


SAE INTERNATIONAL

SAE J3016™ LEVELS OF DRIVING AUTOMATION

	SAE LEVEL 0	SAE LEVEL 1	SAE LEVEL 2	SAE LEVEL 3	SAE LEVEL 4	SAE LEVEL 5
What does the human in the driver's seat have to do?	**You are driving** whenever these driver support features are engaged – even if your feet are off the pedals and you are not steering			**You are not driving** when these automated driving features are engaged – even if you are seated in "the driver's seat"		
	You must constantly supervise these support features; you must steer, brake or accelerate as needed to maintain safety			When the feature requests, you must drive	These automated driving features will not require you to take over driving	
	These are driver support features			These are automated driving features		
What do these features do?	These features are limited to providing warnings and momentary assistance	These features provide steering **OR** brake/ acceleration support to the driver	These features provide steering **AND** brake/ acceleration support to the driver	These features can drive the vehicle under limited conditions and will not operate unless all required conditions are met		This feature can drive the vehicle under all conditions
Example Features	• automatic emergency braking • blind spot warning • lane departure warning	• lane centering **OR** • adaptive cruise control	• lane centering **AND** • adaptive cruise control at the same time	• traffic jam chauffeur	• local driverless taxi • pedals/ steering wheel may or may not be installed	• same as level 4, but feature can drive everywhere in all conditions

For a more complete description, please download a free copy of SAE J3016: https://www.sae.org/standards/content/J3016_201806/

Copyright © 2014 SAE International. The summary table may be freely copied and distributed provided SAE International and J3016 are acknowledged as the source and must be reproduced AS-IS.

Figure 1 Levels of automation (SAE International, 2019)

The degree of driving automation is directly related to its technology's degree of complexity (Martínez-Díaz, Soriguera, & Pérez, 2019). Different classification schemes to refer to automation levels were proposed, but the dominantly adopted scheme is SAE standard J3016 (SAE International, 2019), as shown in Figure 1. It classified vehicle automation on a scale from level 0 (no automation) to level 5 (full automation). The levels are differentiated by who is responsible for controlling the car (human/vehicle), who supervises the environment (human/vehicle) and the types of driving scenarios (Martínez-Díaz et al., 2019).

In levels 0-2, humans are responsible for driving and must supervise the environment at all times, even if support features are in use (Martínez-Díaz et al., 2019; SAE International, 2019). Level 0 corresponds to manual driving. While warning and momentary assistance systems (e.g. navigation systems, automatic emergency braking) may be present, the human has full control over all lateral (i.e. steering) and longitudinal (i.e. braking, acceleration) car movements. From level 1 upwards, either lateral or longitudinal control is taken over by the vehicle. In automation level 1, driver assistance systems can either support steering or braking/acceleration. Examples are adaptive cruise control, frontal collision avoidance or lane-keeping assistance systems. In automation level 2 both steering and braking/acceleration are supported.

In levels 3-5, the car is responsible for driving and monitoring the environment, allowing the human "driver" to do other tasks. In level 3, the vehicle can handle well-defined driving scenarios (e.g. driving on the highway) but may request the human to take back control once it reaches its capabilities' boundaries (e.g. in unexpected traffic). In level 4 the vehicle can drive autonomously under limited conditions. It will not require the human to take over control. Nevertheless, level 4 automated vehicles cannot handle all traffic scenarios. Level 5 automation (full automation) means the car has full control over all operational (steering, acceleration/braking, monitoring) and tactical decisions (turning, lane changes, signal observance) and can handle all traffic scenarios. The driver will never drive manually (Martínez-Díaz et al., 2019). Level 5 automation is what authors usually refer to when they mention autonomous vehicles.

The use of the term automated vehicles (AVs) in this thesis refers to vehicles that function independently of a human operator in the given traffic scenario (i.e. drive through an intersection). Humans are passengers and do not need to drive. As such, vehicles from automation levels 4-5 are covered.

2.1.2 Connectedness

Connected vehicles (CVs) have technologies to communicate with contributing agents, such as vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I) (e.g. to a traffic controller) or vehicle-to-pedestrians (V2P) communication. The combination of all communication modes is called vehicle-to-everything (V2X) (Bagloee, Tavana, Asadi, & Oliver, 2016; Zeadally, Guerrero, & Contreras, 2019). Propagated messages could include information about road, traffic and weather conditions, routing options or vehicle trajectories, thus enabling a range of services (Zeadally et al., 2019).

Communication between vehicles makes them more aware of their surroundings. It enables e.g. multi-vehicle cooperation, awareness of obstacles outside the vehicle's line of sight, or improved traffic forecasts (Guanetti et al., 2018). Connectedness thus forms the basis for intelligent intersection control.

V2V communications allow vehicles to communicate without a centralized entity (Martínez-Díaz et al., 2019). Instead, vehicles form so-called vehicular ad hoc networks (VANETs) (Zeadally et al., 2019). Core technologies supporting vehicular communication are dedicated short-range communication (DSRC) and 5G. DSRC is the currently used technology. Bandwidth for DSRC has been allocated in both the US and the EU and standards for its use were developed (L. Chen & Englund, 2016). 5G technology is being studied as an alternative, and could in the future be used as well (Guanetti et al., 2018). A detailed review of communication technology is out of scope, but readers may refer to reviews by Guanetti et al. (2018), Martínez-Díaz et al. (2019) and Zeadally et al. (2019).

2.1.3 Types of vehicles

Based on the two technologies, it is possible to create four types of vehicles, as shown in Figure 2. Conventional vehicles are vehicles that cannot communicate with other vehicles or infrastructure. Additionally, they are manually driven by humans, even though some driver assistance features may be present (automation levels 0-2). CVs are human-driven vehicles that have communication technologies that enable V2I and V2V communication. AVs are vehicles that require no human inputs but instead use sensors to monitor and react to their environment. Connected and automated vehicles (CAVs) are capable of both automated driving and communication (Guanetti et al., 2018).

		Level of Automation	
		Not automated	**Automated**
Connectedness	**Unconnected**	Conventional / regular vehicles (RV)	Automated vehicles (AV)
	Connected	Connected vehicles (CV)	Connected and automated vehicles (CAV)

Figure 2 Possible types of vehicle classifications based on their level of automation and connectedness

Note that vehicles equipped with communication technology are not necessarily automated and vice versa. However, likely automated vehicles will increasingly rely on connectivity to reach autonomy, thus a convergence of both technologies is expected (Catapult Transport Systems, 2017).

Penetration rates of both AVs and CVs are expected to increase in the future. A review of studies that forecasted penetration rates of AVs and CVs was done by Overtoom (2018). While forecasts between researchers differ greatly, the studies agree that some fraction of vehicles will be automated or connected. Furthermore, the review concludes that likely most AVs will also be connected. Based on these results, a mix between conventional vehicles and CAVs can thus be expected for the near future. It will take a long time until a penetration rate of 100% CAVs is reached, if ever (Overtoom, 2018).

2.2 Background on Intersection Control

When conflicting vehicle streams meet at intersections, rules are needed to coordinate their passing. Implemented strategies are tradeoffs between traffic flow efficiency, accessibility, safety and environmental factors. Intersections can be unsignalized (e.g. yield or stop signs, roundabouts, priority rules, right-before-left) or signalized (i.e. using traffic lights) (L. Chen & Englund, 2016; Ilgin Guler, Menendez, & Meier, 2014). The focus of this thesis will be on signalized intersections for two reasons. First, traffic signals are usually used to manage busier urban intersections which are more likely to be congested. Second, in mixed traffic scenarios in the transition period, it would not be possible to communicate decisions directly with V2I or V2V to unconnected vehicles. Instead, traffic signals would still be needed to relay these decisions. The next sections will then focus only on signalized intersection control.

2.2.1 Signalized intersection control

Traffic signals sequentially assign the right-of-way for conflicting vehicle streams at intersections via three signals (red, yellow, green) with certain cycle times. The phases and timings are decided by the traffic signal controller (Florin & Olariu, 2015). In general, we can distinguish traditional TSC from intelligent TSC. The following paragraphs introduce these phases briefly.

2.2.1.1 Traditional traffic signal control

Traditional TSC strategies did not directly communicate with vehicles. Nevertheless, these controllers went through several phases of improvement. Commonly they are categorized into three types: fixed-time controllers, actuated controllers and adaptive controllers (Ilgin Guler et al., 2014; L. Li et al., 2014).

Initial traffic controllers were fixed-time controllers. Signal plans used predefined cycles and timings set by administrators (H. J. Chang & Park, 2013). Improved fixed-time controllers allowed to set different control plans at different times during the day (e.g. during morning or evening rush-hour) to facilitate better traffic flows for recurring traffic patterns (H. Zhang et al., 2002). Optimization of traffic control plans occurred offline based on historical traffic data (Florin & Olariu, 2015). Advantages of these static controllers are a simple design, low computing power and the lack of need for expensive hardware. However, fixed-time controllers cannot adapt well to dynamic non-recurring traffic fluctuations such as congestion, accidents, construction or differences in human driving patterns (H. J. Chang & Park, 2013). As such they cannot control traffic optimally, and could even increase congestion (Artimy, 2007).

An improved type of controller are traffic-responsive controllers, such as actuated and adaptive controllers. Traffic-responsive controllers use real-time measurements gathered from infrastructure-based sensors such as inductive loops, magnetic or ultrasonic sensors (H. J. Chang & Park, 2013; Florin & Olariu, 2015; Jing et al., 2017). The controller can dynamically optimize phase sequences, timings and splits based on this data to adapt to current traffic demand. Actuated controllers are usually used for isolated intersection control. Their algorithms use static parameters (e.g. minimum green time, maximum phase length) to react to real-time data (Jing et al., 2017).

Adaptive controllers are used to control intersections in arterials or networks. Data from upstream detectors is used as input for prediction models that estimate vehicle arrivals and queue lengths to optimize traffic signals based on incoming traffic (L. Li et al., 2014). Advantages of traffic-responsive controllers are that they can adapt signal plans in real-time to traffic fluctuations, making them more flexible and efficient. Nevertheless, the sensors can only cover limited local areas and can only gather instantaneous traffic data (when a vehicle passes over it), instead of continuous traffic states (e.g. speed, location, heading) (H. J. Chang & Park, 2013). Thus, it does not allow for route prediction. Since only sampled data is available, roads are modeled in averages or densities, which can critically impact signal efficiency (He, Zheng, Chen, & Guan, 2017). Additionally, infrastructure sensors are expensive to install and maintain, and system performance is severely degraded if a sensor breaks down (Jing et al., 2017).

2.2.1.2 Intelligent traffic signal control

In intelligent traffic signal control, the goal is to create self-adaptive control systems. Self-adaptive control systems adapt signal plans in real-time to match current traffic patterns based on certain control targets and real-time traffic data (Y. Wang, Yang, Liang, & Liu, 2018). Self-adaptive traffic signal control systems can adjust their internal logic and parameters based on their perception of the environment (Abdulhai, Pringle, & Karakoulas, 2003), and are thus able to permanently adapt to changed traffic conditions without human intervention.

In intelligent TSC, the developmental phases are not as clearly distinguished as in traditional TSC, since different researchers are relying on different assumptions and are working on different model types. Yet,

Y. Wang, Yang, Liang, & Liu (2018) attempted to capture different developmental levels of self-adaptive traffic signal control in a framework, with each level making traffic controllers more intelligent. The transition from the lowest to the highest (predicted) development level can be summarized as follows: TSC evolved from offline control to real-time online control, from a low data to a data-rich environment, from isolated intersection control to network control and from empirical control methods to intelligent self-adaptive control. The authors predict that at the highest level of self-adaptive traffic signal controllers, controllers will use data-driven, adaptive and model-free learning algorithms.

In self-adaptive traffic signal control, the controller requires data on the current state of the environment. Usually, this data is gathered via vehicular involvement. Gathering information via communication networks has several advantages compared to infrastructure-based sensors. V2V and V2Icommunication enable a more accurate perception of the environment, infrastructure and traffic. The detection range is significantly larger and gathered information is more detailed. Instead of instantaneous readings and averages, it is possible to sample each connected vehicle's exact trajectory (Q. Guo et al., 2019), thus enabling more accurate traffic estimations. Furthermore, using vehicle communication technology is cheaper than installing sensors in the infrastructure and allows for more flexibility (L. Chen & Englund, 2016; R. Zhang, Ishikawa, Wang, Striner, & Tonguz, 2020).

2.3 Literature Review: Intelligent traffic signal control

In this thesis, intelligent traffic signal control is the topic of interest. The goal of intelligent signal control is to retime signals based on real-time data such that vehicle delays or queue lengths are minimized, traffic flow is smoothened, and emissions are reduced (Jing et al., 2017). To get a better understanding of the field, and to define the specific research topic, knowledge gap and research questions, a literature review has been conducted. The following sections will present this review.

2.3.1 Literature search method

During the literature search, the databases Scopus and Google Scholar were used. The review took place in March 2020. Since in literature different terms are used to describe the same concepts, various keywords were used. The keywords "connected", "cooperative", "autonomous", "automated", "driverless" and "self-driving", were used in combination with "car", "vehicle", "driving" or "intersection". Further keywords were "congestion", "traffic signal control", "traffic efficiency" and "traffic flow". Only English results were included. Due to the large number of results, results were filtered by document type "review". Forward and backward snowballing were applied to find additional relevant review papers.

2.3.2 Literature review results

Many researchers worked on intelligent traffic signal control in recent years, and many approaches were proposed. Since several in-depth reviews on these approaches already exist, the further analysis will be based on these reviews. Table 1 summarizes the review topic, the classification used and the review focus of the nine found reviews. Some important topics mentioned in the reviews are discussed next.

Table 1 Summary of review studies on intelligent traffic signal control

Reference	Review topic	Classification Types	Description
(D. Zhao, Dai, & Zhang, 2012)	Computational intelligence techniques in urban traffic control	Recurrent and non-recurrent congestion; further divided into different computational intelligence strategies	- introduces applications of computational intelligence in TSC to solve traffic recurrent and non-recurrent congestion

(L. Li et al., 2014)	Improving the efficiency of traffic control strategies by using vehicular communications	Isolated intersection control and network-wide traffic control (both with vehicular communication)	- Focus on the controller side - Discusses non-cooperative and cooperative driving - Discusses transition from feedback to feedforward control due to vehicle connectivity - More focus on isolated intersection control strategies - Contrasting design preferences in controller design (model-based predictive control vs simulation-based predictive control; global planning-based control vs local self-organization-based control; control using rich, maybe redundant information vs concise information)
(Florin & Olariu, 2015)	Adaptive traffic signal control using vehicular communication	Level of vehicular involvement (no vehicular involvement, passive involvement, active involvement); further subdivisions in sections	- Taxonomy of adaptive traffic signal control strategies achieved through vehicular communication - Focuses on optimizing traffic signals (from technology perspective) - Discusses most representative papers - Future outlook/vision for traffic signal control
(L. Chen & Englund, 2016)	Cooperative intersection management systems	Signalized and unsignalized intersections (with more sub-categories in unsignalized intersection control)	- Review of cooperative intersection management systems - Main methodologies and techniques used - Focus is on non-signalized intersections - Summary of worldwide projects
(Rios-Torres & Malikopoulos, 2017)	Coordination of CAVs at intersections and merging on highway on-ramps	Centralized and decentralized approaches (each further divided into heuristic rule-based and optimization-based control approaches)	- Review of strategies on intersection and highway ramps coordination using CAVs
(Jing et al., 2017)	Adaptive signal control methods	Isolated and multiple intersection management strategies	- Systematic review of papers, including overview tables (i.a. method, objective function, CV penetration rate, data resource, simulation scenario, simulation platform) - Evaluates studies quality based on systematic and quantifiable criteria - Presents an adaptive traffic signal control framework
(Q. Guo et al., 2019)	Urban traffic signal control with CAVs or mobile sensing data	Deterministic and stochastic approaches, further divides studies for CAV signal control into research topics (driver guidance, actuated control, platoon-based signal control, planning-based signal control, signal-vehicle coupled control, multi-vehicle cooperative driving without signals)	- Reviews methods for estimating traffic flow states and optimize traffic signal timings based on CAVs - Focuses on mixed conventional and CAV traffic - Highlights required related research topics (transit priority control, network control, impacts of CAVs penetration, safety guarantee, implementation requirements for CAV technologies)
(Elliott et al., 2019)	Advances in CAV technology (includes a section on intersection navigation)	Centralized and decentralized approaches	- Reviews state-of-the-art on several CAV-technologies (communications, security, intersection navigation, collision avoidance, and pedestrian detection) needed for the success of CAVs

			- Categorizes intersection navigation into several methods (real-world testing, optimization and control, heuristics), types of intersections (isolated, networks), and other aspects (environmental impacts, human-driver and pedestrian considerations, prioritized vehicles)
(Namazi et al., 2019)	Intelligent intersection management systems using AVs	Signalized intersection control (for pure AV situations and mixed traffic) + unsignalized intersection control (for pure AV traffic); further subdivided by study goals; categorizes studies by applied method	- Systematic review of papers, including overview tables - Compares how well different approaches are evaluated - Identified the intended goals of the reviewed studies (efficiency, safety, ecology, passenger comfort, others, mix of goals) - Identifies the used methods in the reviewed studies (optimization-based, rule-based, machine learning-based, hybrid approaches)

2.3.2.1 Types of control strategies

In intelligent signalized intersection management systems, generally three types of control strategies are possible (Q. Guo et al., 2019; Jing et al., 2017), depending on the available type of technology.

The first method is *intelligent signal scheduling*. This method uses V2I communication to a signal controller. Vehicles are passive participants that send information about their current state (e.g. location, speed, its environment) to a traffic controller. The controller aggregates the vehicle data to create better estimations of current traffic states and uses this to optimize signal phases, sequences and timings (Florin & Olariu, 2015; Q. Guo et al., 2019). This method requires that some vehicles are connected.

The second method is *trajectory control*. Based on CAV's own states and observations of traffic signals and other vehicles, the CAV's arrival time at the intersection can be estimated. V2V allows vehicles to negotiate with traffic controllers and other vehicles to optimize their own vehicle trajectories for efficient passing (L. Chen & Englund, 2016; Florin & Olariu, 2015) before even reaching the intersection. This can be done by calculating passing sequences and by adapting trajectories (via accelerating/decelerating or steering) where necessary (Jing et al., 2017). However, non-CAVs cannot be controlled using this strategy.

Lastly, it is possible to jointly optimize both trajectory and signal control (Rios-Torres & Malikopoulos, 2017). Likely, this strategy can achieve the best traffic control performance (Q. Guo et al., 2019).

2.3.2.2 Types of methods

Most reviews describe the methods used in studies. Some reviews further classify these into groups. Namazi et al. (2019) classified studies as optimization-based, rule-based, machine learning-based or hybrid. According to this review, 44.76% of the included studies (N=105) used optimization-based methods, 40% rule-based methods, 3.8% machine learning methods and 11.43% hybrid methods. Rios-Torres and Malikopoulos (2017) group optimization-based and heuristic rule-based approaches. Chen & Englund (2016) do not classify studies, but they do discuss mathematical optimization in general terms.

Optimization-based methods aim to find values for a set of decision variables that optimize an objective function under a set of constraints (L. Chen & Englund, 2016). Objectives can be singular or a combination of goals. Possible objectives are to maximize efficiency (minimize traffic delay, maximize intersection throughput), maximize safety (minimize collisions, resolve conflicts), minimize the ecological impact (minimize emissions or fuel consumption) or maximize passenger comfort (Namazi et al., 2019). Decision variables can be time slot allocation, passing sequences, or vehicle maneuvers such as braking, accelerating or steering. Constraints can e.g. be safety-related (safe headways, speed limits) or passenger comfort-related (no hard braking, no sudden movements) (L. Chen & Englund, 2016). The advantage of optimization-based methods is that complex situations can be modeled by including different objectives and constraints. Optimization-based methods search for an optimal solution under the respective traffic scenario, thus are adaptive to fluctuations in traffic. Drawbacks are that solution spaces may be non-linear, so search algorithms could get stuck in local optima. Additionally, models can get complex quickly (e.g. more constraints, higher traffic volumes, more complex scenarios), which is computationally expensive to solve. Only few methods were deemed solvable in real-time (Namazi et al., 2019).

(Heuristic) rule-based methods use a set of rules to determine the next action. Rule-based methods are computationally simpler than optimization-based methods, and thus usable in real-time. However, when the complexity of the traffic scenario increases, more rules must be added, which can make the model too complex to guarantee efficient traffic flows. Additionally, rule-based methods are based on a set of stochastic rules, thus cannot guarantee an optimal solution or adapt to traffic. As a consequence, rule-based methods' performances may vary significantly between traffic scenarios (Namazi et al., 2019).

Hybrid methods of optimization and rule-based methods could reduce the computational complexity compared to pure optimization models, while also making the model more adaptive to traffic scenarios. However, up to date little research on how to combine models has been done (Namazi et al., 2019).

Machine learning models could train controllers by letting them handle different traffic scenarios through trial-and-error. Over time, algorithms learn how to respond to scenarios efficiently. This could make machine learning models adaptive to traffic, while also remaining low in complexity. Additionally, no inner workings of traffic scenarios would have to be modeled, making the models simpler than other methods. Machine learning methods have been effective in other fields, yet compared to other methods in TSC have gained less attention (Jing et al., 2017).

2.3.3 Limitations and future research

Most reviews elaborate on the assumptions, simplifications and limitations of their reviewed studies. Table 2 summarizes and categorizes these findings. As can be seen, many assumptions and simplifications were made within the proposed models. The reason for this is that modeling an intersection is a too complex spatio-temporal problem to study all factors at the same time.

Table 2 Assumptions, limitations and simplifications made within reviewed studies

Type	**Limitation / Simplification**	**Identified by**	**Description**
Simplified models	Exclusion of other traffic participants	(L. Chen & Englund, 2016; Elliott et al., 2019; Guanetti et al., 2018; Jing et al., 2017; Namazi et al., 2019)	Only road traffic (specifically cars) is included. Other modes (pedestrians, cyclists, buses, trucks) are excluded.
	Assumption of 100% penetration rate of CV or CAV technology	(L. Chen & Englund, 2016; Elliott et al., 2019; Florin & Olariu, 2015; Q. Guo et al., 2019; Jing et al., 2017; Namazi et al., 2019; Sarker et al., 2020)	Models assume that each vehicle is connected or connected and automated. These models cannot handle conventional vehicles.

	Exclusion of unexpected situations	(L. Chen & Englund, 2016; Elliott et al., 2019)	Models do not consider sudden occurrences (accidents, car breakdowns, power outages, bad weather)
	Unrealistic human behavior	(L. Chen & Englund, 2016)	Models assume rational and uniform human behavior.
	Only unsaturated traffic conditions	(Jing et al., 2017)	Models are not developed to handle oversaturated traffic scenarios.
	Unrealistic driving behaviors	(L. Chen & Englund, 2016)	Models assume that there are no lane changes, no overtaking, no reversing.
	Unrealistic vehicle characteristics	(L. Chen & Englund, 2016; Namazi et al., 2019)	Models do not realistically model vehicle characteristics or car following behaviors. Models assume deterministic and homogeneous characteristics (e.g. length, width, dynamics, quality of sensors, types of algorithms) for all vehicles, rather than stochastic ones.
Simplified validation environment	Simplified traffic conditions	(L. Chen & Englund, 2016; Elliott et al., 2019; Jing et al., 2017; Namazi et al., 2019)	In the validation environment only simple traffic conditions are being tested (e.g. balanced flow rates, geometric intersection)
	Too few traffic scenarios	(Elliott et al., 2019; Jing et al., 2017; Namazi et al., 2019)	Models are validated using too few types of scenarios (e.g. no variation in flow rates, intersection layout)
	Validated in simulation studies	(Q. Guo et al., 2019; Jing et al., 2017)	Models are validated in simulation environments rather than field studies.
Technological assumptions	Assumption of perfect communication	(L. Chen & Englund, 2016; Q. Guo et al., 2019; Namazi et al., 2019)	Models assume perfect information. Data loss, interference, transmission delays are excluded.
	Assumption of perfect sensing technology	(Florin & Olariu, 2015)	Models assume vehicles have perfect information on their speed, location, etc.
	Disregard of privacy and security	(Q. Guo et al., 2019)	Models do not consider data privacy or security.
	All intersections have communication technology installed	(L. Chen & Englund, 2016)	Models assume that V2I communication to all infrastructure is possible.
Network-based	Modeling only isolated intersections	(Elliott et al., 2019; Florin & Olariu, 2015; Q. Guo et al., 2019; Jing et al., 2017; L. Li et al., 2014; Namazi et al., 2019)	Most studies model an isolated intersection, rather than corridors or networks of intersections.
Other	Model not solvable in real-time	(Namazi et al., 2019)	Most models are too computationally complex to be solvable in real-time.

In this thesis, the focus will be mixed traffic modeling. Mixed traffic modeling has been widely mentioned as a critical shortcoming in past reviews, yet it is critical for the nearby future.

2.3.3.1 Mixed traffic

Most intelligent TSC models assume a 100% penetration rate of either CVs or CAVs (L. Chen & Englund, 2016; Elliott et al., 2019; Florin & Olariu, 2015; Q. Guo et al., 2019; Jing et al., 2017; Namazi et al., 2019; Sarker et al., 2020). The review by Namazi et al. (2019) found 93% of the 105 reviewed studies focused only on pure CAVs. Only 7% considered mixed traffic scenarios.

Assuming that all vehicles are autonomous and/or connected makes models simpler. The assumption of 100% AV penetration means the traffic signal controller can control all vehicle's trajectories. The assumption of 100% CV penetration means no traffic estimation models are needed since full information on all vehicle trajectories and states is available (Q. Guo et al., 2019). Estimating the status of unconnected vehicles is complex, and the used estimation method can have big impacts on the controller's performance (Jing et al., 2017). These estimations become even more complex under limited CV-penetration rates. Research on how to do this is just starting to emerge (Q. Guo et al., 2019).

Since in the transition period different mixes of conventional, automated and connected vehicles will be present, TSC methods that work well for different penetration rates are needed. Different penetration rates have been shown to significantly impact a controller's effectiveness (Q. Guo et al., 2019; Jing et al., 2017; Sarker et al., 2020; Y. Wang et al., 2018). Yet, only 30.77% of the reviewed studies by Jing et al. (2017) tested their controller under different penetration rates.

While some of the proposed algorithms also work well for lower penetration rates (e.g. Zheng & Liu, 2017), for most of the proposed controllers traffic efficiency increases significantly only after 25-30% CV-penetration (Q. Guo et al., 2019). The exact critical transition point (i.e. the critical penetration rate) is unknown since many factors influence it. Only recently have studies started to investigate the relation between CV/CAV-penetration and controller performance (e.g. Rios-Torres & Malikopoulos, 2018; Validi, Ludwig, Hussein, & Olaverri-Monreal, 2018).

2.4 Research gap and research goal

The research gap this thesis aims to alleviate can be summarized as follows: in the past, many traffic signal control methods have been proposed, but most methods can only be applied under full CV- or CAV-penetration. Since we are currently entering the transition phase towards connected vehicles, strategies are needed which work under less than 100% CV-penetration. However, only few controllers can be applied under these mixed traffic situations.

Therefore, this thesis aims to alleviate this gap by investigating design choices when building a new traffic signal controller. The goal is to propose and evaluate an intelligent traffic signal control strategy that works under mixed traffic scenarios.

2.5 Summary

This chapter introduced the reader to the field of intelligent signalized intersection management, identified a research gap and presented the research goal.

First, the core concepts of automation and connectedness were introduced. Automated vehicles are vehicles that do not need a human operator to drive or monitor the vehicle in a given traffic scenario. Connected vehicles are vehicles that can communicate with infrastructure (V2I) or other vehicles (V2V).

Then, background knowledge on signalized intersection control was briefly summarized. Traditional signal control methods do not communicate directly with vehicles. Traditional strategies include fixed-time, actuated or adaptive controllers. Intelligent controllers on the other hand aim to control traffic in a self-adaptive way, i.e. in real-time and based on current traffic data.

To scope down the field and identify a research goal, a literature review on intelligent signalized intersection control strategies was conducted. In the review, different types of control strategies and methods were discussed. Additionally, many research limitations were identified. One of the research gaps is that most models assume a 100% CV- or CAV-penetration rate. However, in the close future there will be a transition phase from conventional to automated and/or connected vehicles. Therefore, this thesis aims to contribute to closing this gap by developing and proposing an intelligent signaled intersection management strategy that can operate in the transition period.

3. Research approach

3.1 Scoping: Traffic signal control method

3.1.1 Machine Learning

As described in chapter 2.3.2.2, different types of traffic signal control strategies exist. For each type of method, different theories are used, and many different models have been suggested. This makes it impossible to study all methods simultaneously.

In this thesis, it was chosen to focus on machine learning algorithms. Fewer studies focused on machine learning than on optimization or rule-based methods. Yet, the authors of review articles believe machine learning methods will likely become the future of intelligent TSC (Jing et al., 2017; Namazi et al., 2019; Y. Wang et al., 2018; D. Zhao et al., 2012). Machine learning models can find (near-)optimal signal settings within large stochastic and non-linear systems, which would be too complex for humans to model (D. Zhao et al., 2012). Furthermore, machine learning models are likely able to overcome the shortcomings of other models by improving data collection, traffic state and human behavior prediction and decision-making.

Different types of machine learning algorithms have been applied to intelligent traffic signal control, such as reinforcement learning, fuzzy logic, group intelligence algorithms (e.g. genetic algorithms) and neural networks (Y. Wang et al., 2018; D. Zhao et al., 2012). Yet, both Yau et al., (2017) and Y. Wang et al. (2018) state that fuzzy logic, group intelligence and neural network algorithms alone are not suitable to create self-adaptive traffic signal controllers due to the strict assumptions these algorithms impose. They believe that reinforcement algorithms are the most suitable algorithm for TSC. Thus, it was decided to use reinforcement learning.

3.1.2 Reinforcement Learning

Using reinforcement learning (RL) in TSC has many advantages. RL algorithms are model-free approaches, i.e. they do not require a human to create a precise mathematical model of the system environment. They can learn good or optimal policies without prior knowledge of the environment or external supervision. In other approaches such as optimization or rule-based methods, researchers must a priori model the complete intersection control system including all behavioral rules. However, this can get very complex. Oftentimes it is unclear how setting certain traffic phases will affect traffic at later points in time. For example, it may be a good decision to set a certain signal to green in the short-term, but in the long-term, another choice may have been better. RL agents however can operate in such delayed return environments, making them very suited for adaptive TSC (Rodrigues & Azevedo, 2019). RL agents can learn the best actions just by interacting with their environments in a trial-and-error way. If the agent does an action that improves a predetermined performance indicator (e.g. the cumulative waiting time of all cars in the intersection), then agent gets a reward, otherwise it gets a punishment. As such, the only inputs the modeler needs to specify are the ways that the agent perceives its environment, the actions it can choose and the reward function. This is much simpler than in other methods.

Another advantage of RL is that it allows for real-time control. Other methods such as optimization approaches are too computationally complex (see chapter 2) for real-time use. RL models however are, once fully trained, computationally cheap to use and thus suitable. Furthermore, unlike e.g. rule-based models, RL agents can adapt to dynamically changing environments (e.g. rush hour, demand changes, accidents) (Mannion, Duggan, & Howley, 2016; Yau et al., 2017), as long as similar situations have been encountered during training. Unlike traditional TSC methods, RL allows traffic splits to be shortened or lengthened and traffic phases to be chosen as needed, rather than following fixed cycles (Yau et al., 2017). This allows better adaptation to the current traffic situation, and ultimately to better reduce congestion.

Within the field of reinforcement learning, many different types of models exist. Yet, researching all types of models would be too time-consuming for this thesis. Due to this, it was decided to focus on Deep Q-learning models since these are the most commonly applied type of RL model in TSC literature. Chapter 4 will provide a brief introduction to deep Q-learning models and RL in general.

3.2 Research questions and research flow

Based on the outlined knowledge gap, research goal and the chosen traffic signal control method, the main research question and sub-research questions were formulated.

MAIN: Can deep Q-learning models be used to control signalized intersections in mixed traffic scenarios such that traffic congestion is reduced?

1. **What is the current state-of-the-art in intelligent traffic signal control using deep Q-learning for both homogeneous and mixed traffic scenarios?**
2. **What types of deep Q-learning models would be suitable to design an intelligent traffic signal controller for mixed traffic scenarios?**
3. **How can the deep Q-learning-based traffic signal controllers be calibrated and trained in a systematic manner?**
4. **How can the designed deep Q-learning-based traffic signal controllers be evaluated in order to determine which controller performs better under which circumstances?**
5. **To what extent are the designed deep Q-learning-based traffic signal controllers able to reduce traffic congestion for mixed traffic situations in a robust manner?**
6. **How do the results compare to other literature results?**

The first sub-question requires to conduct a literature review. This is needed to get to know the state-of-the-art, in order to make an informed decisions on how to design a deep Q-learning controller and what design decisions were made in previous research. Based on the results of the first sub-question, at least two suitable deep Q-learning model types can be chosen (sub-question 2).

When designing RL-agents, many design decisions have to be made and many hyperparameters must be chosen. Sub-question 1 already answered what these decisions are and how they influence the final model performance. Yet, calibrating RL-agents remains a trial-and-error task. In sub-question 3, it is investigated how this calibration process can be conducted in a more systematic way. The goal is to create fine-tuned agents which reduce traffic congestion for mixed traffic situations. In this step, the proposed controllers must be implemented in software.

Sub-question 4 is concerned with the methodology of evaluating the designed agents. Based on this methodology, several experiments can be implemented and conducted. The results of these experiments will be used to answer sub-question 5. More specifically, it will be investigated to what extent model types work well for different circumstances (e.g. under what types of traffic scenarios, under which penetration rates), and how the performance of the different models compares to each other.

In sub-question 6 the results of experiments will be compared to results of other studies. A cross-comparison with literature allows us to embed the results in literature, to validate the results and to determine the new scientific contributions of this study.

3.3 Research approach: Modeling and Simulation

For this research, a design and simulation approach was chosen. In the design part, RL-based controllers that can accommodate mixed traffic will be designed. In this thesis, at least two different types of RL controllers will be compared, so that it can be tested which model works well under which traffic scenarios and penetration rates. In order to build the controllers, design choices of other studies will be analyzed and combined in novel ways. Yet, it may not always be clear a priori what design choices and parameters work best for the newly designed models. To solve this, systematic experiments will be conducted in which each of the unknown design choices and different values for each of the parameters will be tested. The goal is to fully calibrate the algorithms such that stable and well-performing RL-agents are created.

Once the RL-models have been built, they are ready to be trained on different traffic scenarios. After training, the algorithms can be evaluated. Both training and training will take place under several different CV/CAV penetration rates, so that it can be investigated how the controllers perform under different types of mixed traffic situations.

Training RL-models is a time-intensive process which requires the RL-model to be exposed to numerous different traffic situations. Due to this, RL-algorithms are trained using microscopic traffic simulations.

Furthermore, RL algorithms are generally also evaluated using microscopic traffic simulations. Using simulations to evaluate traffic signal control strategies has several advantages compared to field studies. To evaluate a strategy, it must be tested under different traffic situations. In simulations, it is possible to do many runs under precisely controlled traffic scenarios, which allows for detailed results. In field simulations however, it is impossible to control all factors exactly (Jing et al., 2017), which could make results less precise. Furthermore, simulation studies are faster, safer and cheaper than field studies.

However, simulations also have disadvantages. Intersections are complex, which means that there are many factors which may significantly impact how well a controller performs. It is also possible that some behavior (especially human behavior) is not modeled realistically. Results may largely depend on the assumptions under which the model has been tested. However, research on traffic simulation is ongoing, and more features have been added over time to make it more realistic (Fellendorf & Vortisch, 2010). Ultimately, simulators are an established way to evaluate newly proposed traffic signal controllers. Once a strategy was proven effective in a simulation study, and once automated and connected vehicles are mature enough for real traffic, field studies can be used as the next validation step.

4. Background on Deep Q-learning Learning

This section introduces the required background knowledge of reinforcement learning and deep Q-learning needed to understand this thesis. For quick introductions for beginners, please refer to (Heberer, 2019; Nicholson, n.d.; Osiński & Budek, 2018; Sharma, 2019; Silver, 2015). For an in-depth introduction, see (Sutton & Barto, 2018).

4.1 Basic Idea of Reinforcement Learning

Reinforcement Learning (RL) is a subset of Machine Learning in which models learn to make sequences of decisions within complex and/or uncertain environments. RL algorithms apply an iterative trial-and-error process to find the best solution for a problem without any external supervision or hints from the modeler on how to solve the problem. Instead, the algorithm solely learns which actions are optimal via the reward the action obtains (similar to how you would train e.g. a pet). If the algorithm performs a good action, it will receive a positive reward. If it performs a bad action, it will receive a punishment (negative reward). This trial-and-error learning process takes place over many iterations. Over time, the algorithm gains experience and learns which actions lead to better results. RL algorithms can correlate immediate actions with both immediate and delayed returns.

There are three basic elements in RL: the agent, the environment and rewards (see Figure 3). The agent is the entity taking actions within its current state at each time step. The environment is the world in which the agent is located. The environment responds to actions by the agent(s) and provides as outputs a new state and a reward for the agent in the new time step. Rewards are incentives or feedback for the agent to measure how good a certain action is.

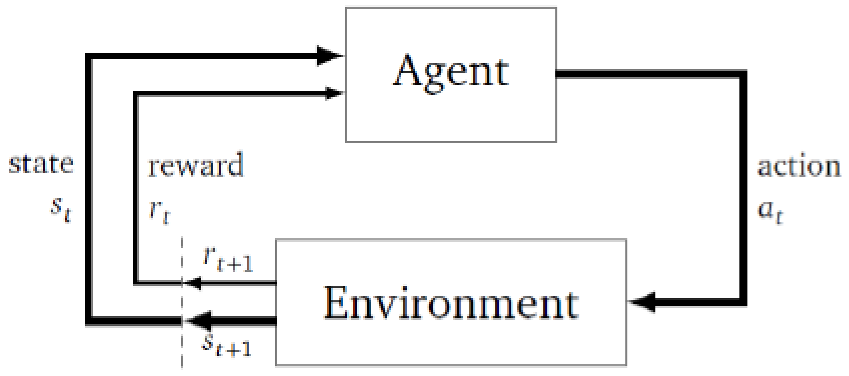


Figure 3 Schematic on Reinforcement Learning as an MDP. Adapted from (Yang, Tan, & Menendez, 2017)

Before proceeding with the formal definition, some terms will be informally defined.

Actions (A). A is the set of all possible actions which the agent can take while in state s. Action sets are discrete and finite. Each round, the agent picks a single action.

State (S). S is the current situation of the environment, i.e. a specific configuration of objects and time.

Reward (R). A reward is an immediate return from the environment based on the last chosen action by the agent. Rewards can be used as feedback by the agent to evaluate the success of its last action.

Policy (π). The strategy of the agent to map states to actions, i.e. to determine its next actions based on its current state. The goal is to find a policy that leads to the best rewards.

Transition function (T). Change in the environment as a result of taking a certain action in a certain state.

Discount factor (γ). Returns from the environment can be immediate or delayed. Delayed rewards are rewards that happen after several state-action combinations are chosen. Since these rewards are less important than immediate rewards, delayed rewards will be discounted by factor γ between 0 and 1. A γ -value of 0 means only immediate rewards are considered, while a γ-value of 1 means that all future delayed rewards are as important as the immediate rewards.

Value (V). Expected long-term return with discount (opposed to short-term reward). $V\pi(s)$ is the expected long-term return of the current state under policy π.

Q-value (also action-value) (Q). Similar to V, but also includes the current action a. $Q\pi(s,a)$ is the long-term return of taking action a under policy π from current state s. It measures how *good* it is to take action a when in state s and following policy π.

4.2 Markov Decision Processes

Formally, RL agents are generally modeled as Markov Decision Processes (MDP). According to Sutton and Barto (2018), MDPs are "classical formalizations of sequential decision making where actions influence not just immediate rewards, but also subsequent situations, or states, and through those future rewards. Thus, MDPs involve delayed reward and the need to tradeoff immediate and delayed reward".

An MDP can be formally defined as a four-tuple $< S, A, R, T >$, where $S = \{s_1, \dots, s_n\}$ is a finite set of states and $A = \{a_1, \dots, a_m\}$ is a finite set of actions. The function $T: S\ x\ A\ x\ S \to [0,1]$ defines the transition function specifying the probability of taking action a in state s and ending up in state s'. The reward for taking action a in state s and ending up in state s' is represented by reward function $R: S\ x\ A\ x\ S \to \mathbb{R}$.

The system fulfills the Markov property if the outcome of an action only depends on the previous state and action, such that

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \dots) = (s_{t+1}|s_t, a_t, r_t) \quad (1)$$

To determine which actions agents will choose when in a certain state, they use policies. Policy π maps from states to actions: $\pi: S \to A$. This means that if the agent is in state s_0 and acts under policy π, it will choose action $a_0 = \pi(s_0)$ and will transition to state s_1. For this transition, the agent will receive reward $r_0 = R(s_0, a_0, s_1)$.

Rewards that are received are a combination of immediate returns and delayed returns. Delayed returns are discounted by a discount factor $\gamma\ \in [0,1]$. The agent's goal is to maximize its expected reward over time. Usually, it should give preference to short-term (immediate) rewards over long-term (delayed) rewards. The total return, the expected discounted cumulative reward over time is calculated as:

$$R_t\ = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = E[\sum_{k=0}^{\infty} \gamma^k r_{t+k}] \quad (2)$$

Two different value functions can be defined: a state value function and a state-action value function. The value of state s under policy π is $V^{\pi}(s)$. It represents the expected return of following policy π when starting in state s. It is defined as

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}[R_t | s_t = s] \\ &= E_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s] \\ &= E_{\pi}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s] \\ &= E_{\pi}[r_t + \gamma V^{\pi}(s_{t+1}) | s_t = s] \\ &= \sum_{s' \in S} T(s, \pi(s), s') \left(R(s, a, s') + \gamma V^{\pi}(s') \right) \end{aligned} \tag{3}$$

where s' denotes the next state. This equation is also known as the Bellman equation. It describes the relationship between the value of state s and its successor states.

Furthermore, a state-action value function $Q^{\pi}(s, a)$ can be defined. It shows the expected value of taking action a while the agent is in state s while following policy π. In other words, it is a measure of how good the state-action pair is. It is defined as

$$\begin{aligned} Q^{\pi}(s, a) &= E[R_t | s_t = s, a_t = a] \\ &= E_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a] \\ &= \sum_{s' \in S} T(s, \pi(s), s') \left(R(s, a, s') + \gamma V^{\pi}(s') \right) \end{aligned} \tag{4}$$

An optimal policy π^* (i.e. a policy that results in expected values equal or greater than any other policies for all states) satisfies

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s') \left(R(s, a, s') + \gamma V^*(s') \right) \tag{5}$$

$$Q^*(s, a) = \sum_{s' \in S} T(s, a, s') \left(R(s, a, s') + \gamma \max_{a' \in A} Q^*(s', a') \right) \tag{6}$$

If T and R are known, the optimal policy can be found e.g. via dynamic programming methods that use the recursion in equation (5). These approaches are known as model-based approaches. These approaches do not require the agent to interact with the environment directly (Y. Li, 2018a). However, in many cases the system is too complex to be able to determine T and R upfront. In these cases, RL algorithms can be applied. These model-free approaches rely on the earlier described trial-and-error processes to sample the underlying MDP to learn the optimal policies.

We can express the optimal value function and policy solely in terms of the Q-function:

$$V^*(s) = \max_{a \in A} Q^*(s, a) \tag{7}$$

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s, a) \tag{8}$$

Now, we do not need to know the transition function T or the reward R explicitly. Instead, the Q-function is sufficient to determine the next action. The Q-function is iteratively estimated and updated.

4.3 Model-free Reinforcement Models

Model-free reinforcement models can be classified as value function-based or policy search-based. Additionally, hybrid actor-critic approaches exist that use both value functions and policy search (Arulkumaran, Deisenroth, Brundage, & Bharath, 2017). Value function methods estimate the value function V^* in an iterative process, until an optimal value function is found. This process is based on the optimality Bellman operator. Policy-search methods do not need a value function model, but directly search for an optimal policy π^*. Given a policy, its value function can be determined via the Bellman equation.

Furthermore, value-based approaches can be divided into *on-policy* and *off-policy* algorithms. In off-policy algorithms, actions may be selected according to non-optimal policies during training. On-policy algorithms however only select policies which are currently estimated to be optimal, i.e. these algorithms only select actions greedily.

In this thesis, it was decided to use deep Q-learning. Deep Q-learning is a type of Q-learning, which is a model-free, value function-based, off-policy RL algorithm. This algorithm will be described next.

4.4 Tabular Q-learning

Traditional Q-learning (Watkins, 1989) uses a lookup table of all possible Q-values of state-action pairs and iteratively updates the Q-values. The updates at each time step t are performed using the following equation:

$$\underbrace{Q_{t+1}(s_t, a_t)}_{\text{new Q-value}} = \underbrace{Q_t(s_t, a_t)}_{\text{old Q-value}} + \underbrace{\alpha}_{\text{learning rate}} \overbrace{\left[\underbrace{r_t + \gamma \underbrace{\max_{a' \in A} Q_t(s_{t+1}, a)}_{\text{estimate of optimal future Q-value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q_t(s_t, a_t)}_{\text{old Q-value}} \right]}^{\text{temporal difference}} \tag{9}$$

The algorithm uses the Bellman equation for the value function update, by weighting the old Q-value and the temporal difference value. The learning rate (or step size) $\alpha \in [0,1]$ determines the speed that new information is being learned. A learning rate of 0 means the algorithm learns nothing, while a value of 1 means that only the most recent information is considered to choose new actions (i.e. all previously learned knowledge would be forgotten).

If each state-action pair is visited infinite times, then Q-learning will converge to an optimal policy. In practice this is not impossible, so algorithms are stopped after a certain fixed number of episodes or when the model's performance improvements are under a certain threshold. Nevertheless, it is not easy to determine a priori for how long a model needs to be trained.

The full Q-learning algorithm is shown in algorithm 1 (adapted from (Pol, 2016)).

Algorithm 1: Tabular Q-learning

```
1: Initialize $\boldsymbol{Q(s, a)}$ randomly for all $\boldsymbol{s \in S, a \in A, i = 1}$
2: for each episode do
3:      Initialize $\boldsymbol{s, a}$
4:      for each step $\boldsymbol{t}$ in episode do
5:           $\boldsymbol{a = \pi(s)}$  // Select a using policy based on current Q, e.g. ε-greedy
6:           Take action $\boldsymbol{a}$
7:           Receive reward $\boldsymbol{r}$, observe new state $\boldsymbol{s'}$
8:           Update Q: $\boldsymbol{Q(s, a) = Q(s, a) + \alpha\left(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)}$
9:           set $\boldsymbol{s = s'}$
10:     end for
11: end for
```

4.5 Exploration and Exploitation

In reinforcement learning, there is a trade-off between exploration and exploitation. Exploration is needed to discover up until then unencountered state-action combinations, to find potentially better/more optimal solutions. Exploitation is needed to use the learned knowledge about which action would be optimal in the current state. Both must be balanced. On the one hand, excessive exploration leads to agents limiting their capacity to receive rewards, therefore reducing their performance. On the other hand, excessive exploitation leads to the agent never learning new potentially better policies (Yau et al., 2017).

Literature has extensively studied this trade-off. Different exploration strategies exist, e.g. ε-greedy, Botzman (also called softmax), UCB. The choice of exploration strategy can affect the performance of the algorithm (Yau et al., 2017).

4.6 Function approximation

A critical problem with tabular Q-learning is the so-called "curse of dimensionality": if the number of state-action pairs becomes too large or even unlimited (for continuous states) problems arise. Too much storage space would be needed to store every state-action pair and the computational costs and learning times would increase exponentially (Arulkumaran et al., 2017; Yau et al., 2017). Additionally, since every state-action pair would have to be visited near infinitely to converge, finding the optimal policy is no longer guaranteed. Tabular Q-learning works well for small problems, however in the real-world state-action spaces are rarely small or finite.

To solve this, function approximation can be applied. It is used to approximates the value of a function and allows us to generalize experiences to other similar (potentially unencountered) state-action pairs. In the case of Q-learning, function approximation can be used to generalize learned Q-values of state-action pairs to predict similar state-action pairs. As such, the algorithm is likely to learn Q-values faster.

Instead of having to store every state-action pair Q-value in a table, the learned function $Q(s, a)$ will be parameterized by a learned weight θ. Different methods can be used to approximate these values. One of them is gradient descent. In gradient descent, θ is updated by minimizing the mean squared error (MSE) between the current $Q(s, a)$ estimate and the true $Q^{\pi}(s, a)$ estimate under policy π (i.e. target). The update is done by taking the derivative of the MSE, which is calculated as follows:

$$MSE(\theta) = \sum_{s \in S} P(s)[\underbrace{Q^{\pi}(s,a,\theta^*)}_{\text{target Q}} - \underbrace{Q_t(s,a,\theta_t)}_{\text{current Q estimate}}]^2 \tag{10}$$

$$\frac{\partial}{\partial \theta_t} MSE(\theta) = 2[Q^{\pi}(s,a,\theta^*) - Q_t(s,a,\theta_t)] \frac{\partial}{\partial \theta_t} Q_t(s,a,\theta_t) \tag{11}$$

Where $P(s)$ is the sampling distribution (i.e. the probability to visit state s under policy π).

However, the target Q^{π} is not known, thus we estimate it by using the reward in the current time step and a discounted estimate of the next state's best Q-value using the current Q_t estimate.

$$Q^{\pi}(s,a,\theta^*) \approx r_t + \gamma \left[\max_{a' \in A} Q_t(s_{t+1}, a', \theta_t) \right] \tag{12}$$

Since it uses the max-operator, equation (12) is an optimistic estimate of the Q-value at time step t.

Now that the Q-value is parameterized by θ, supervised machine learning algorithms can be used to approximate the Q-function.

4.7 Deep Q-learning

In this thesis, it was chosen to study deep Q-learning models. In deep Q-learning, deep (artificial) neural networks ((A)NN) are used to approximate the Q-values (also called DQN: deep Q-networks). NNs are universal approximators that can approximate complex and highly non-linear functions. As such, they can recognize hidden patterns from complex data, making it suitable for problems that are too complex for humans or other data analysis methods (Araghi, Khosravi, & Creighton, 2015). An in-depth explanation of how neural networks work is out-of-scope. For an introduction to neural networks, refer to Bishop (1996). Here it suffices to know that deep neural networks will be used. Deep NNs are NNs which have multiple hidden layers. Adding more hidden layers to the NN allows the NN to approximate more complex functions, at the cost of longer training times. Thus, a tradeoff between the number of hidden layers and training time is needed.

The basic idea is that we want to use NNs to update the Q-values, similar as in equation (9). To update the NN's weights needed to approximate the Q-functions, commonly gradient descent and backpropagation are used. To use this, we need a cost function that measures the difference between the NN's estimated Q-value and the actual Q-value. The goal is to minimize this error function. Since the actual Q-value is unknown, we can again estimate it by using the temporal difference target of equation (9). It represents the total expected reward of all future time steps, including discounted future rewards. Then we can iteratively update the target value.

We can set up our loss function as follows, using the squared error loss:

$$L = \frac{1}{2}\left[Q(s_t, a_t) - \left(r_t + \gamma V(s_{t+1})\right)\right] \tag{13}$$

Now we can perform gradient descent using the derivative of the loss function:

$$\frac{\partial L}{\partial Q(s_t, a_t)} = Q(s_t, a_t) - \left(r_t + \gamma V(s_{t+1})\right) \tag{14}$$

Using this, an update rule for the Q-value can be created:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) - \alpha\big[Q(s_t, a_t) - \big(r_t + \gamma V(s_{t+1})\big)\big] \tag{15}$$

Where the best estimate for the value of the next state is the value of the expected best action in that state:

$$V(s) = \max_{a \in A} Q_{target}(s, a) \tag{16}$$

A basic scheme of DQN is shown in Figure 4. The NN consists of an input layer, a certain amount of hidden layers (minimum of 2 in case of deep learning), and an output layer. The input to the NN is the current state, which consists of multiple measurement values of the environment. The output of the NN are the Q-values which are associated with every possible action (in this scheme 4 actions). In other words, the NN predicts the Q-values for all m actions $a \in A$ in the current state s_t: $Q(s_t, a_0), Q(s_t, a_1), \dots, Q(s_t, a_m)$.

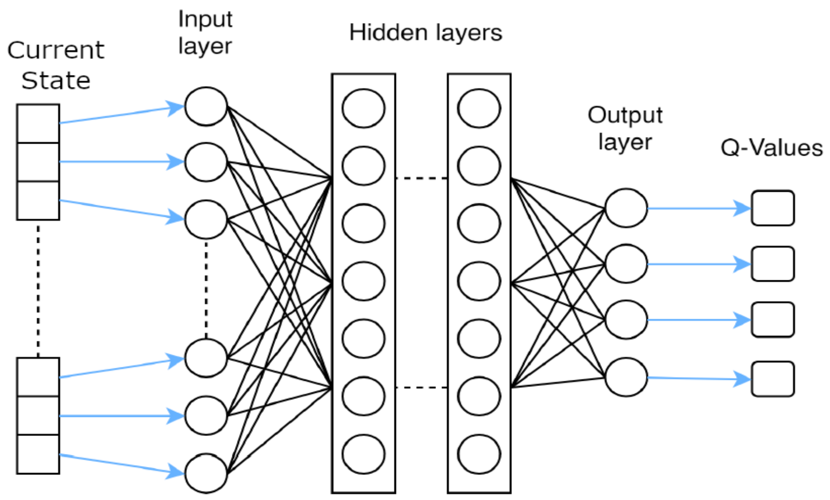


Figure 4 Scheme of Deep Q-learning (adapted from Vidali (2018))

4.8 Convergence

4.8.1 Convergence issues

If we use function approximation approaches such as NNs to estimate our Q-values, convergence to the optimal policy like in tabular Q-learning is no longer guaranteed. Different instabilities or convergence problems may occur (Pol, 2016; Samad, 2020).

Violates i.i.d assumption

Supervised Learning algorithms such a NN assume that samples are independently and identically distributed (i.i.d). However, the next sample of state-action pairs (s_{t+1}, a_{t+1}) is heavily dependent on the previous pair (s_t, a_t). Furthermore, since Q_t is iteratively updated every time step, the sampling distribution of the sample (s_t, a_t, r_t, s_{t+1}) also changes. In other words, samples are neither independent nor identically distributed.

Catastrophic forgetting and moving targets

When using function approximation, Q-values are not updated per single entry in a table, but rather globally via changing the Q-value approximation function (here: the NN). One problem is that of catastrophic forgetting: when new samples are used to update the Q-function, the network could forget earlier learned tasks from older samples (Coşkun, Baggag, & Chawla, 2019).

Furthermore, since as shown in equation (12), both the current $Q(s,a)$ estimate and the target estimate $Q^{\pi}(s,a)$ are used to update the new $Q(s,a)$ estimate. Thus, in every time step the Q-values of a specific state-action pair change. However, since the Q-function is updated globally, this means that also the Q-target estimate changes at the same time, moving the Q-target. This can lead to oscillation of the Q-target and thus destabilize the learning.

4.8.2 Solutions

Different methods have been proposed to solve the convergence issues. The two most commonly used are experience replay and target networks, as first proposed for DQN in the 2015 DeepMind paper (Mnih et al., 2015) that used RL for Atari games. Furthermore, many other add-ons to the so-called "vanilla" or base DQN have been proposed (see e.g. Hessel et al. (2018)), which will be discussed in the next section.

4.8.2.1 Experience replay

Experience replay (Lin, 1992) is a method used to avoid the correlation of samples. *Samples* consist of the tuple (s_t, a_t, r_t, s_{t+1}), where s_t is the current state, a_t is the chosen action, r_t is the reward of choosing action a in state s, and s_{t+1} is the resulting next state. Usually, neural network updates would be performed immediately after each experience (i.e. after obtaining a sample). This is referred to as *online learning*.

Instead, in experience replay, experiences are first stored in the *replay memory* M. Each training step, a randomized group of samples (called a *batch*) are taken from the memory and used to train the network. Batches can consist of a single sample, a subset of samples (called *mini-batches*) or all samples. This process is shown visually in Figure 5. Using experience replay thus ensures that the chosen samples are no longer correlated since the sampling order is random. As such, learning is logically separate from gaining new experiences. This ensures better convergence (Mnih et al., 2015).

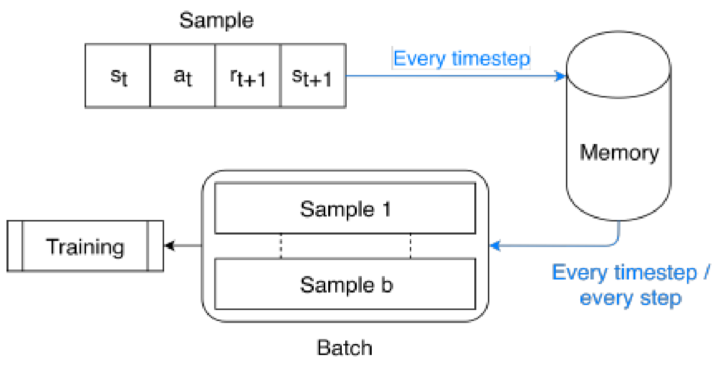


Figure 5 Memory handling before training the NN. Adapted from Vidali (2018).

To use experience replay, a memory is needed. This memory has a certain size, representing how many samples can be stored before having to remove old ones. Usually, once the memory is full, the oldest sample is removed. Additionally, the modeler must determine a batch size, which specifies how many samples are included in each batch. In experience replay, samples may be encountered several times. The higher the batch size and the more often training instances happen, the more often a sample will be encountered. Encountering an experience multiple times is useful, since gaining new experiences can be computationally costly. Since the algorithm works iteratively and only updates Q-values slowly, encountering a sample multiple times is useful. Especially reusing rare but valuable experiences can be beneficial, since it may take long to re-encounter the situation.

However, experience replay also has disadvantages. One drawback is that it requires a lot of memory, which may be costly (Park & Shires, 2019). Another problem is that if the environment changes too much, the old rewards for certain state-action pairs may no longer be accurate. Furthermore, since samples are chosen uniformly, common experiences may be sample more often than rare ones, even though those experiences may be much more critical (e.g. experiences leading to fatal accidents)(Pol, 2016). An improvement for this problem is *prioritized experience replay* (Schaul, Quan, Antonoglou, & Silver, 2016).

4.8.2.2 Target network

To avoid the problem of moving targets, an approach called *target network* can be applied (Mnih et al., 2015). In this method, two separate Q-networks with different θ parameters are kept: an online network $Q(s, a, \theta)$ and a target network $Q(s, a, \theta^T)$. The online network is used to for backpropagating every step, i.e. to use experiences to update parameter θ of the online Q-network. The target network is used to estimate the target Q-values. The target Q-network is kept frozen for several iterations, the so-called freeze interval. Once the freeze interval is over, the two networks are synchronized by copying the values from the online network to the target network. Now the target Q-values are not changing every time step, but only once at the end of the freeze interval. For a schematic of the approach, see Figure 6. Since now the current Q-values and target Q-values are no longer changing simultaneously, the agent's decisions become more stable (R. Zhang et al., 2020).

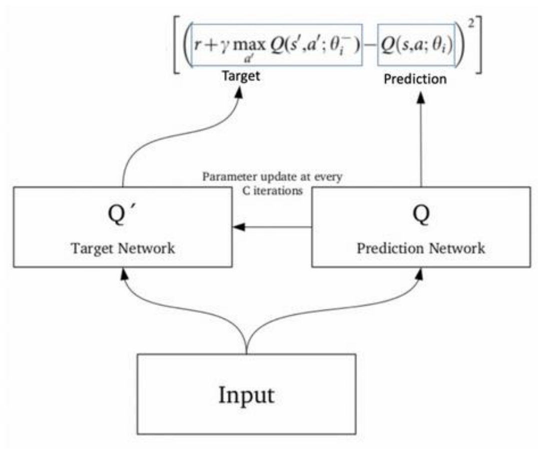

$\left[\left(r+\gamma \max_{a'} Q(s',a';\theta_i^-)\right)-Q(s,a;\theta_i)\right]^2$
Target
Prediction
Parameter update at every C iterations
Q′
Target Network
Q
Prediction Network
Input

Figure 6 Scheme of target network approach. Adapted from Kocabaş (2017).

4.9 Deep Q-learning extensions

As mentioned, many extensions for deep Q-learning (and other reinforcement learning algorithms) have been proposed in literature. These algorithms have shown performance improvements when used standalone, however, it was unclear to what extent these extensions can be combined and how these combinations affect the performance, learning speed and stability of DQN.

Hessel et al. (2018) aimed to study this problem and published the highly cited paper in which they evaluated DQN combined with six extensions (called the *Rainbow agent*). The extensions studied in the paper were double deep Q-learning (Van Hasselt, Guez, & Silver, 2015), prioritized experience replay

(Schaul et al., 2016), dueling networks (Z. Wang et al., 2016), multi-step bootstrap targets (Sutton, 1988), distributional Q-learning (Bellemare, Dabney, & Munos, 2017) and noisy nets (Fortunato et al., 2017). To evaluate the performance of the proposed agent, the in RL commonly used Atari 2600 benchmark was used. The authors evaluated the performance of their Rainbow agent compared to vanilla DQN as well as other state-of-the-art DQN algorithms (which use either one or several of the described extensions). They found that the Rainbow agent outperforms vanilla DQN and all other state-of-the-art agents by a significant margin. Additionally, the authors conducted ablation studies in which they ran the Rainbow agent with one extension removed to quantify how much an extension contributes to Rainbow's performance. It was found that prioritized experience replay and multi-step learning were most crucial for the improved performance compared to DQN. Distributional Q-learning, noisy nets and dueling networks also generally improved the agent's performance, however not in all games. Only double Q-learning did not improve the agent's performance.

Another type of DQN extension is to add recurrence. The most popular variant of recurrent neural networks is to add long short term memory layers (LSTM) (Hochreiter & Schmidhuber, 1997). Recurrent networks allow to model time/sequence-dependent behavior by remembering information on past states in memory. Instead of using only the current state as input, recurrent networks also use their memory of previously seen states. This method is particularly useful in situations in which either decisions not only depend on the current state but also on the past states, or in which the state is only partially observable. Hausknecht & Stone (2017) were the first to use recurrence in deep RL. Specifically, they applied LSTM to DQN agents (resulting in deep recurrent Q-learning; DRQN) and evaluated the models on the Atari benchmark. They compared vanilla DQN and recurrent DQN and found that DRQN performed better than DQN if the state was only partially observable and that DRQN was more robust to changes in the quality of observations. Further improvements to DRQN were made by Kapturowski, Ostrovski, Quan, Munos and Dabney (2019 and Lample & Chaplot (2017).

Many other types of extensions exist, but a more in-depth discussion is out of the scope. For more information on different model types and extensions, refer to Y. Li (2018b) or Sutton and Barto (2018).

4.10 Summary

This chapter has introduced required background knowledge on Q-learning and deep Q-learning. The basic idea is that algorithms are model-free, i.e. they can learn good policies without researchers having to model all internal behaviors. Instead, agents learn by trial-and-error by observing the environment state, choosing actions and getting feedback from the environment on how good or bad their action was.

Traditional Q-learning agents use tabular Q-learning. Yet, this oftentimes suffers from the curse of dimensionality, meaning that the number of state-action pairs quickly grows too large to train. Deep Q-learning solves this problem, but also introduces stability issues. The stability issues can be combatted by applying two extensions: experience replay and target network freezing. A wide range of other extensions have been proposed in literature and could be used to improve model performance or stability.

5. State-of-the-art in Deep Q-Learning in traffic signal control

This section the state-of-the-art in using deep Q-learning in TSC will be reviewed. The focus will only be on Deep Q-learning. Earlier research on using RL in TSC used other, not deep, methods such as tabular Q-learning. However, as described in chapter 2, these methods do not allow to model as complex relations as using neural networks. Other methods than Q-learning could potentially also be applicable for TSC, however since most research focuses on Q-learning, only this method is taken into consideration.

5.1 Literature gap and sub-research goal

The first RL traffic controller was proposed in 1994, however at that time it could not yet be implemented due to computational limitations (Mikami & Kakazu, 1994). The first adaptive controller using RL was proposed in 2003 (Abdulhai et al., 2003). Since then, many authors used RL for adaptive TSC. For a review of RL methods in TSC from 1997-2010, please see the paper by El-Tantawy, Abdulhai and Abdelgawad (2014). For papers until 2016, please see Mannion et al. (2016). In this time, RL approaches in TSC only used tabular Q-learning, thus they were limited in state space and model complexity. To solve this, deep reinforcement learning started becoming popular.

As mentioned, DQN was first used by Mnih et al. in 2015 to play Atari games. Deep RL is a fast-developing field, in which much new research is being published. In recent years, deep RL also became popular in TSC (P. Chen, Zhu, & Lu, 2019). The most popular RL model is deep Q-learning (Y. Wang et al., 2018).

Since the field is developing so quickly, existing reviews are outdated quickly. Several general DQN reviews not specific to TSC were found in literature, such as by Arulkumaran et al. (2017), Y. Li (2018) and Mousavi, Schukat and Howley (2016). Wang, Yang, Liang and Liu (2018) reviewed methods to control self-adaptive TSC systems with a focus on integrating TSC and intelligent transport systems. They conclude that TSC based on multi-agent RL will probably be the future research focus, however they do not review any DQN methods. Lastly, the survey by Yau et al. (2017) reviews the state-of-the-art in RL until 2017 and they specifically review different representations of TSC problems in RL models (i.e. state representations, action space, reward functions). Nevertheless, their review also does not include DQN methods. As can be seen, no reviews focusing on DQN in TSC have been published. Thus, this review will fill this gap.

5.2 Literature Search Method

A systematic review of existing literature was conducted through the databases Scopus, IEEE and Google Scholar. Two different searches were conducted: the first was to find papers that apply DQN to TSC and the second was to find papers that apply RL (not specifically DQN) to TSC in mixed human and connected or automated vehicle environments. The search terms used for each search are shown in Table 3.

Table 3 Search terms used in the literature review on RL approaches in TSC

Key concept	**Keywords**		
	1	2	
Traffic signal control	●	●	Intersection, "traffic light", "traffic signal", "traffic signal control", signal*
Road traffic	●	●	Vehicle, car
Reinforcement learning		●	reinforcement learning
Deep Q-learning	●		DQN, “deep reinforcement learning”, deep Q-learning
Connected or automated vehicles		●	Autonomous, automated, intelligent, driverless, unmanned, cooperative, connected, connect*, communication, cooperat*, V2*, smart
Mixed Traffic		(●)	Non-automated, human-driven, mixed, penetration, partial* detect*, POMDP

All searches were performed on the title, abstract and keywords. To limit results from other transport domains, the keywords “UAV”, “unmanned aerial vehicle”, “aerial”, “underwater” and “air” were excluded. In each case, articles were filtered first by their titles and by screening the abstracts. For the relevant results, the full-body text was used. Lastly, for the selected literature the references were reviewed in included if relevant (‘backward snowballing’).

Only papers that focused on signalized intersection control were included. This excluded studies that focused on non-signalized intersections, on-ramp merging or AV trajectory control. Studies that included pedestrians, lane changes or traffic rule violations were excluded since they are out of the scope of the algorithm that will be designed in this thesis. Other articles that were out of the scope were papers on pedestrian or human driving prediction algorithms, cross-modal learning and human-vehicle interactions. To guarantee quality articles are included, all included papers must describe the implemented RL algorithm, their methodology used (experimental setup, list of parameters) and must provide quantitative experimental results.

A full list of all included papers, as well as the results can be found in Appendix C. Due to page size limitations, it could not be included in the main body of the paper.

5.3 State-of-the-art of Deep Q-learning in traffic signal control

In the next sections, the findings will be outlined and discussed by topic. Note that all discussions assume that traffic drives on the right side of the road, but conclusions can easily be adjusted to left-side driving.

5.3.1 Network topologies

Many different types of network topologies can be chosen for the traffic model. In general, it is possible to study either isolated intersections, multiple intersections in an arterial or a network of intersections. Furthermore, it is possible to study either synthetic topologies or real-world topologies (Yau et al., 2017). Synthetic layouts allow for better control of the environment, but real-world intersections allow to better adapt the TSC to the specific requirements for that intersection.

In the reviewed literature, many papers focused on isolated intersections. All of these papers used synthetic intersection layouts of bidirectional 4-way intersections. Researchers chose different numbers of lanes for the legs: between 1 and 4. Recently, more interest is being shown in studying coordinated signal control in arterial or networks.

5.3.2 Possible movements and traffic phases

When looking at the single intersection level, the modeler must decide which movements are permitted for vehicles and how traffic phases should be assigned (Yau et al., 2017). Since nearly all papers assumed 4-way intersections, the discussion will only be based on these.

Concerning the allowed movements, the following options were found:

- only through traffic (i.e. driving straight without turns)
- through traffic and turns
- movements as in the real-world (through traffic, turns, potentially U-turns or one-way streets).

The possible traffic (green) phases are related to this. The options found here are either two traffic phases, 4 traffic phases or more than 4 traffic phases.

Some researchers decided to not allow turns. (e.g. P. Chen et al., 2019; L. Li, Lv, & Wang, 2016; Mousavi, Schukat, & Howley, 2017; Nawar, Fares, & Al-Sammak, 2019; Pol & Oliehoek, 2016; Wei, Zheng, Yao, & Li, 2018; Wu, Kong, & Fan, 2019; R. Zhang et al., 2020). In these models, only two phases were used: North-South green and East-West green. This was done to simplify the model; however, it reduced the models' real-world applicability. Often these models were meant as proofs-of-concept for newly proposed ideas.

The other popular choice was to use 4 green phases. While other green phase options would be possible (see (Yau et al., 2017)), the common choice was to use the option shown in Figure 7. Two traffic phases allow through and right-turning traffic and two traffic phases allow left-turning traffic.

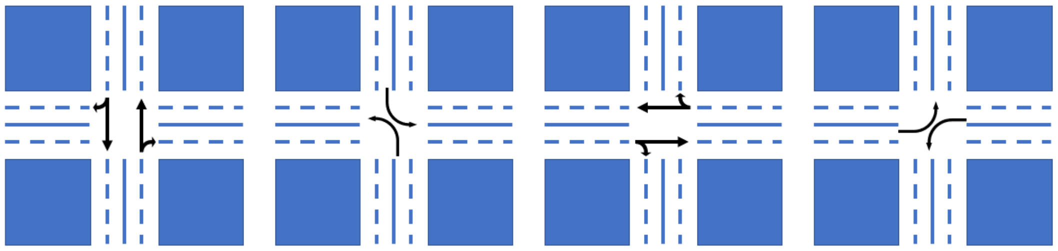


Figure 7 Commonly chosen combination of green phases

A few papers even added as many as 8 phases of compatible movements, see e.g. (M. Guo, Wang, Chan, & Askary, 2019; Kim, Jung, Kim, & Lee, 2019).

Ultimately, the choice of allowed traffic movements and green signals will influence for what kind of intersections the model will be applicable. In practice, the choice of intersection will depend on the local traffic situation. Nevertheless, adding more possible green phases to the model can make the controller more flexible, but it will also become harder to train the agent. A trade-off will have to be made.

5.3.2.1 Additional phase constraints

Many modelers add transitions between phases. Some researchers add a few seconds of yellow time after switching phases (e.g. S. Wang et al., 2019; Wei et al., 2018). Additionally, some models also include a few seconds of all-red phases during transitions (e.g. Coşkun et al., 2019). The purpose of yellow and red phases is to allow vehicles to clear the intersection in order to increase safety. It also makes the model more similar to real-life.

Some researchers added extra logic to the allowed phases, a so-called "sanity check" (R. Zhang et al., 2020). This involves adding a minimum and/or maximum duration for green phases. This means the controller may not switch phase until the minimum green duration has passed and must switch phase if the maximum duration was reached. Minimum phase lengths are implemented to ensure that at least one vehicle can pass during a green phase. Maximum phase lengths are implemented to ensure fairness between lanes since some reward functions could lead to shorter queues becoming less prioritized and having to wait infinitely long compared to longer queues. Other authors however argue that imposing minimum and maximum durations a priori is not a necessity since the algorithm would develop such policies by itself (Genders & Razavi, 2018).

5.3.3 States

The state representation is an important input for RL models. The state represents the agent's decision-making factors, i.e. it represents everything the agent knows about the current situation at time t and the agent must decide which action to take based only on this knowledge. Within the literature, many different state representations were found. States can either be singular or consist of several different sub-states (i.e. measures for different factors combined into one overall state) (Yau et al., 2017). Additionally, the state of one intersection may include information about other surrounding intersections. The following (sub)state representations were found in DQN literature:

- **Queue size:** Number of vehicles waiting/halting within a leg or lane. To determine which vehicles are waiting, a maximum speed has to be defined (e.g. 0.1 m/s). All vehicles below this speed are counted as waiting in the queue, all others are not. Used e.g. by M. Guo et al. (2019) and L. Li et al. (2016).
- **Vehicle position:** Physical position of vehicles within the intersection. Represented via discrete traffic state encoding (DTSE). In DTSE, lane segments of length l beginning at the stop line are split into discrete cells of length c (Genders & Razavi, 2016). All cells are combined into a state vector or matrix. For DTSE, the size of the cell is critical: if it is too long, the individual vehicle dynamics are lost, but if it is too short, the computational cost will increase drastically. Generally, the cell length is chosen slightly larger than the average car. In the case of vehicle positions, binary DTSE is used: if a vehicle is inside the cell, the cell will be set to 1, else it will be 0.
- **Distance to the nearest vehicle at each approach:** Distance [m] from the stop line to the closest vehicle that is driving towards the intersection for each leg. If no vehicle is approaching on a leg, the maximum distance is used. Representation used by R. Zhang et al. (2020).
- **Vehicle density:** Can be measured per leg (e.g. R. Zhang et al., 2020), per lane (e.g. Genders & Razavi, 2016) or via DTSE (Zeng, Hu, & Zhang, 2018). Is encoded either as an absolute number of vehicles on the leg/lane, as a number of vehicles per area or as $\frac{\#vehicles}{\max \#vehicles}$.
- **Speed.** Speed of vehicles at time t [m/s]. Can be either measured as average speed for the whole leg or lane (e.g. Genders & Razavi, 2018) or represented via DTSE per vehicle (e.g. T. Zhao & Wang, 2019). Speeds can be absolute or normalized with respect to the maximum allowed speed.
- **Delay.** Delay [s] for vehicles from entering the intersection to leaving the intersection (i.e. crossing the stop line). Measured as the difference between the time needed to cross the intersection at the maximum allowed speed and the time needed to actually cross the intersection. Delay can be measured per vehicle or as average per leg over time. Used for example in (Kim et al., 2019).
- **Vehicle waiting time.** Similar to vehicle delay. Measures the time [s] that vehicles have spent under a certain speed threshold (e.g. under 0.1 m/s). The waiting time per vehicle can either be reset to 0 every time the vehicle starts moving or kept as a cumulative value. Nawar et al. (2019) for example used the vehicle waiting time normalized to the maximum waiting time in the network.
- **Vehicle emissions.** CO2, NOx or other emissions created by vehicles. Used e.g. by Kim et al. (2019).
- **Red/green/yellow timing:** elapsed time [s] since the beginning of the red/green/yellow phase for a specific leg or lane. See for example in (Shabestary & Abdulhai, 2019).
- **Yellow phase indicator:** binary value which is set to 1 if there currently is a yellow phase, 0 if not. See (R. Zhang et al., 2020).
- **Current traffic phase:** Combination of green signals currently activated. Within the state representation, phases can be encoded differently. Option 1 is to simply assign numbers to traffic phases (phase 1, phase 2, phase 3, …) and to simply use the phase number for the state. Option 2

is to one-hot encode phases. Here a vector with the same length as the number of traffic phases is created. Every entry in this vector is set to 0, except the current phase which is set to 1 (see e.g. (Zeng et al., 2018)). Option 3 is to use DTSE and to encode per lane (e.g. Nawar et al., 2019) or for every cell if they currently have a green phase (1) or not (0) (e.g. Genders & Razavi, 2016). Option 4 is to use +/- signs in another indicator. R. Zhang et al. (2020) for example encoded the current phase by making the detected car count a positive number if that leg had a green phase, and a negative number if it had a red phase. Apart from encoding the current phase, in some cases, researchers also encode the next phase (e.g. Wei et al., 2018). Note that this is only possible if the traffic phases are in a fixed order.

- **Current time.** Elapsed time [s; min; h; d; m] since a specified time. Different representations are possible, such as encoding the current hour of the day, day in the week, or month of the year. R. Zhang et al. (2020) for example encoded the current time in hours since midnight, discretized into 24 steps of one hour.
- **Raw pixel snapshot.** Camera snapshot of the full traffic situation at the current time. Uses all pixels as input. Used for example by Mousavi et al. (2017).

As can be seen, there are many different possibilities for creating a state representation. Earlier RL methods such as tabular Q-learning only allowed for lower dimensional state representations, and usually required a certain amount of discretization. The most common choices in state representations for tabular Q-learning were queue length, delay or flow rates (Genders & Razavi, 2016; Mannion et al., 2016; Yau et al., 2017). No data at the individual vehicle level could be included. The developments of DQN now allow researchers to include rich high-dimensional data (Liang, Du, Wang, & Han, 2018). Deep Q-learning however removed these restrictions, leading to authors proposing very detailed state representations, often at the individual vehicle level, with many authors proposing to use DTSE.

Some authors argue that the controller can make better decisions the more data it has available (R. Zhang et al., 2020). They argue that if not all information is included in the state, relevant decision factors may be missing, thus the model would not be able to create optimal policies.

Three studies find that using higher-dimensional state representations outperforms RL-algorithms which only use queue size as state. Mousavi et al. (2017) find that using raw pixel inputs from a SUMO snapshot as state leads to a 67-73% reduction in queue length and cumulative delay compared to using a shallow network RL-algorithm that uses queue length as state. Genders & Razavi (2016) also compare a high-dimensional state DQN to a shallow NN RL-algorithm. As state representation they use a combination of a one-hot encoded traffic phase vector with a 2-layered DTSE that includes the vehicle position and speed. They found that the higher-state representation led to an 88% reduction in average cumulative delay, a 66% reduction in queue length, a 20% reduction in average travel time and a similar throughput. Shabestary and Abdulhai (2019) compared DQN using 2-layer DTSE (representing vehicle position and vehicle speed) to tabular Q-learning and found that the higher-dimensional state representation led to a 23.4% reduction in intersection travel time, 39.3% reduction for in queue time, and 36.0% shorter queue lengths with 42.3% fewer variations. As can be seen, higher-dimensional state representations led to significant improvements in various performance indicators compared to only using queue length. Genders & Razavi (2018) and Gao, Shen, Liu, Ito, & Shiratori (2017) speculate that this is because if a model only uses the queue length as state representation, this would ignore all vehicles which are still driving (so they would not be counted as a member of the queue). Shabestary & Abdulhai (2019) provide similar reasoning: they explain that if a controller only uses queue length as input, it cannot distinguish between

100 slow-moving vehicles or an empty lane. Once a lane with a long queue gets a green phase, all these vehicles suddenly start moving slowly. However, since they are now above the cutoff speed to be counted as “in a queue”, they now suddenly become “invisible” to the controller. As such, different environmental situations could be represented by the same state representation but gain completely different rewards even if the same actions are picked. This can lead to instability.

Similar reasoning can be used against using average traffic flows. Average traffic flows use historical traffic data to calculate e.g. average speeds or delays over a certain time interval. The problem is that these are just approximations of the current traffic states, and useful information could be left out (Genders & Razavi, 2016). In case of using average travel delays, the problem is that these kinds of metrics can only be gathered once vehicles have left the intersection, leading to only a delayed representation of the environment (Gao et al., 2017).

Additionally, many metrics such as queue length and average vehicle flows are not metrics which can be measured directly, because they need to be preprocessed and abstracted by experts using prior knowledge (Shabestary & Abdulhai, 2019). To use queue lengths for example, experts must specify a speed threshold. For average flows, assumptions about e.g. vehicle lengths must be made. This type of abstraction and discretization causes a loss of information which may lead to problems.

For these reasons, many authors argue against using discretization or abstraction by experts. Instead, they suggest using high-dimensional states such as raw pixel images or DTSE, so that the deep RL-algorithm can extract and learn the relevant features by itself, without prior knowledge (Gao et al., 2017; Genders & Razavi, 2016; Mousavi et al., 2016).

Other authors argue that certain state representations would not work in practice since this data would either be impossible to obtain in real-time or it would require too expensive sensors (Choe, Baek, Woon, Kong, & Member, 2018; Coşkun et al., 2019; Genders & Razavi, 2018; Horsuwan & Aswakul, 2019; Mousavi et al., 2017; Rodrigues & Azevedo, 2019; S. Wang et al., 2019; T. Wu et al., 2019). Many of these authors work under the assumption that the traffic environment is only observable via infrastructure sensors, such as induction loops, cameras or radar. However, as stated in chapter 2, the rise in connected vehicles could solve this problem. Using vehicle communication could enable the traffic controller to gather individual vehicle-based real-time traffic information, without the need for expensive infrastructure sensors (Liang et al., 2018; R. Zhang et al., 2020).

Nevertheless, the fact that higher-dimensional state representations always gain significantly better results is not undisputed in literature. Genders & Razavi (2018) compared three different levels of details in state representations using the same asynchronous advantage actor-critic RL-algorithm with the same rewards, traffic situation, training parameters and model parameters for all three cases. For each of the three state representations, the authors included the one-hot encoded current traffic phase and the time spent in that phase. For the low-resolution state, they used leg occupancy and average leg speed. For the middle-resolution state, lane queue lengths and lane density were used. In the high-resolution state, a 1-layer DTSE of vehicle positions was used. After running the experiments, they found that there was no difference in traffic throughput between the different controllers and only a 9% difference in queue length between the high and the low/medium-state representations. Only the level of delay was significantly affected: the middle- and high-resolution states led to a reduction of the average vehicle delay of 21 and 25% respectively. The authors conclude that in many cases, lower- or medium-resolution state representations may be sufficient. Yet, they mention that the lack of significantly better performance of the high-resolution controller could have been caused by the fact that the experiments only used a shallow NN, rather than a deep network.

We can conclude that it is not completely clear which state representations would be most suitable. In general, researchers have gained good results when using individual vehicle-based representations (such as position and speed DTSE), especially when in combination with current traffic phase information and elapsed time (Fang, Chen, & Liu, 2019; Gao et al., 2017; Pol & Oliehoek, 2016; Shabestary & Abdulhai, 2019; Wei et al., 2018; Zeng et al., 2018; Zeng, Hu, & Zhang, 2019). Ultimately, adding more information will likely improve the agent's performance (as long as only raw unabstracted data is used), but it will also increase the agent's training time. A trade-off will have to be made.

5.3.4 Actions

The action space represents what changes the traffic controller can make. Within the literature, two types of actions were specified:

- **Pick next traffic phase.** In this action representation, the controller chooses the next traffic phase for a certain number of time steps (usually for the next 1 to 10s). The choice of traffic phases is determined by the modeler (see section 5.3.2). For this action representations, traffic phases are acyclic, meaning they do not appear in any fixed order.
- **Pick traffic phase split.** In this representation, traffic phases happen in a fixed order; the controller can only choose the duration of the current traffic phase. In some models, the controller can extend the current phase for a certain number of time steps or decide to switch to the next one (e.g. Zeng et al., 2018). In other models, the duration of a phase is determined at the beginning of the phase and cannot be extended (e.g. Liang et al., 2018). In some cases, the controller may even decide to spend 0s in a certain traffic phase, effectively skipping the phase and making the phases acyclic (e.g. Zeng et al., 2018).

Both types of action representations were found numerous times in literature. Researchers which use the first type argue that these controllers can more dynamically adapt to traffic, since the controller has free choice of all phases without being constrained by predetermined orders. Proponents of the second type argue that having a fixed order is more predictable for humans and increases safety (Choe et al., 2018).

Note that in case of models that only have two phases, effectively both types of action representations become the same.

5.3.5 Rewards

Rewards are the measure by which agents determine how good or bad a certain action in a certain state was. As such, rewards can either be positive (rewards) or negative (punishments). The agent chooses actions such that expected rewards are optimized. Like for the state representation, there are many different metrics for rewards in literature. Some authors use only one metric, while others use a weighted reward function consisting of two or more metrics. The following metrics were found in DQN literature:

- **Queue length.** See section 5.3.3 on states. For an example see (L. Li et al., 2016).
- **Vehicle delay.** See section 5.3.3. Used e.g. by R. Zhang et al. (2020).
- **Vehicle travel time.** Similar to delay, but instead of using the difference with the optimal travel time, the actual time to pass the intersection is used. Used for example by Wei et al. (2018).
- **Vehicle waiting time.** See section 5.3.3. Used e.g. by Nawar et al. (2019).

- **Number of waiting vehicles.** Number of vehicles under a certain speed threshold. Can be measured for the full intersection (Zeng et al., 2019) or as the absolute or relative difference between legs (Coşkun et al., 2019).
- **Number of vehicles.** Total number of vehicles in the intersection region, regardless of the vehicles' speeds. See e.g. (Choe et al., 2018).
- **Intersection throughput.** Number of vehicles that pass over the stop line. Used e.g. by Zeng et al. (2019).
- **Comparative performance with fixed time control.** Proposed by Du et al. (2019). See discussion below.
- **Fuel/energy consumption.** Amount of fuel used. Can be used as reward to create a controller that reduces environmental impact (Islam, Aziz, Wang, & Young, 2019).
- **Vehicle emissions.** See section 5.3.3. Used by Fang et al. (2019) to reduce environmental impacts of intersections.
- **Penalty for (emergency) stops.** Negative reward for every vehicle that did an (emergency) stop. See e.g. (Y. Wu, Chen, & Zhu, 2019)
- **Phase change**. Penalizes agent if the phase switches. Used to avoid constant flickering of traffic signals. See e.g. (Zeng et al., 2019).
- **Number of teleports.** Teleports are specific to SUMO and only happen in case of would-be collisions or traffic jams. Used by Pol & Oliehoek (2016).

Other rewards that were mentioned in reviews (Mannion et al., 2016; Yau et al., 2017), but which were used in non-DQN algorithms were: delay incurred during phase transitions, appropriateness of green times, achieving green waves, accident avoidance and speed restrictions.

Note that all rewards mentioned are metrics which the agent wants to reduce. As such, the rewards will have negative values, making the term punishment more suitable.

Different authors have implemented vehicle-based rewards (e.g. queue length, waiting time, delay) in various ways. Some authors use cumulative values, others use averaged value per vehicle, and others use relative values between different legs or lanes. Using averaged values is more intuitive for humans to understand and to evaluate. However, average values provide the controller with no information about the total number of vehicles in the intersection. For example, having 1 or 100 vehicles with an average queue length of 10 vehicles would give the same reward, even if the former is a much better situation than the latter. The argument for using differences between legs is to promote fairness between legs.

Additionally, many rewards can either be implemented as an absolute real value (e.g. the current absolute queue length), a discretized or binary value (e.g. 1 if there was a phase change in the last time step, 0 if not) or as a measure of change between time steps (e.g. change in queue length between time step t and t+1). Using values of change immediately shows the controller whether the action improves (positive reward value) or worsens (negative reward value) the current situation. Yet if agents use the absolute value, they have a sense of the order of magnitude of a reward.

Furthermore, some models use squared rewards. Brys, Pham, & Taylor (2014) for example used the cumulative squared vehicle delay so that fewer large delays are prioritized over many short delays. This not only encourages fairness between road users but also leads to faster learning rates.

Different arguments can be used for and against certain rewards. Like for the state representation, some authors argue that certain metrics are only obtainable in simulators, but not in the real world (Mannion et al., 2016). Yet, as mentioned, connected vehicles will be able to overcome this shortcoming.

The most commonly used reward is delay. This metric is vehicle-based and intuitively represents how much time loss was caused by the traffic signal for a specific vehicle. Since the goal of the controller is to reduce the time that vehicles spend in traffic, this seems a suitable metric. The issue is that delays can only be determined with certainty after a vehicle leaves the intersection. This will cause delayed rewards, so the controller may not be able to assign rewards accurately to actions, causing slower learning or unstable control. Furthermore, delays may not properly penalize traffic jams. For a two-lane approach for instance, a controller would not be able to distinguish between one blocked and one full speed lane vs two lanes at half speed, since they would lead to similar rewards, even though the traffic flows are different (Pol, 2016). Similar reasoning can be used for using travel time as a reward.

The second most commonly used reward is waiting time. Intuitively longer waiting time implies that there is more congestion, thus it is a suitable metric. The problem with using waiting time is that the controller converges to a policy in which the traffic phase changes every time a new action is chosen. This is because as mentioned in section 5.3.3, only vehicles under a certain speed threshold are counted as "waiting". If the controller switches often between phases, the vehicles in the new phase's queue start moving slowly and are no longer counted as "waiting". The controller can thus create policies in which vehicles drive slowly, but hardly truly halt (Pol, 2016). The same problem exists when using the number of waiting vehicles.

A policy that leads to constantly flickering lights will cause vehicles to start and stop frequently. For humans this style of driving is uncomfortable and not desirable (Coşkun et al., 2019). To avoid this, some authors add a penalty for the number of (emergency) stops. Other authors decide to instead use a penalty for phase changes. They are undesirable since they require yellow and red transition phases to allow vehicles from one road to stop, before giving a green signal to the new lane. During these transition phases few or no vehicles can pass the intersection, reducing its throughput.

Yet another reward that is frequently used is the intersection throughput. The drawback of throughput is that it does not take into account how long queues are or how long vehicles have been waiting. In over-saturated traffic conditions this could lead to the agent choosing one green only and leaving the other directions red forever. This would lead to unfair results.

As can be seen, different rewards have different advantages and disadvantages, and may be suitable for different objectives or traffic situations. Islam et al. (2019) for example tested three different reward functions: total detected control delay, total detected energy consumption and total detected energy consumption with penalty for stops. They found that reward function 2 provided undesirable results, reward 1 provided better performance in trip delay and reward 3 better energy consumption.

Due to the different strengths and weaknesses of reward functions, authors started to experiment with weighted reward functions. Mannion et al. (2016) for example tested three different reward functions for a tabular Q-learning algorithm: 1. Change in average queue length between time steps, 2. Change in cumulative waiting time between time steps, 3. Weighted reward of 1 and 2. They concluded that reward function 2 performs best under steady traffic flows (i.e. for balanced flow between legs) and reward 1 for highly variable flows. Reward function 3's performance was in between and is then suggested as the best allrounder to handle dynamically changing traffic. Like Mannion et al. (2016), many authors adopted weighted reward functions to adapt to different situations (Horsuwan & Aswakul, 2019; Nawar et al., 2019; Pol & Oliehoek, 2016; S. Wang et al., 2019; Wei et al., 2018; Y. Wu et al., 2019; Zeng et al., 2019; R. Zhang et al., 2020). However, finding suitable weights for the sub-rewards was found to be a complex and time-consuming task (Mannion et al., 2016; Pol, 2016).

In general, a problem with many rewards is that traffic is a dynamic system. The arrival rate of vehicles is constantly changing, but most controllers do not take this into account for their reward. As such, if the arrival rate increases, agents may receive a punishment even if they make the right decision (and vice versa). The rewards in these cases do not represent how good the controller's action was (Du et al., 2019). An attempt to solve this was made by Du et al. (2019). In their model, they use a weighted reward of delay and waiting time. But rather than using absolute values or the change compared to the previous phase, they compare the current delay and waiting time to the delay and waiting time caused by a fixed-time controller operating under the same traffic scenario (i.e. with the same arrival rates, vehicle positions and speeds). In this case, agents receive a reward if they perform better than fixed time controllers and a punishment otherwise.

Overall rewards are a trade-off between optimizing traffic flows, driving comfort and fairness. Some authors even add environmental considerations into the reward (Fang et al., 2019; Islam, Aziz, Wang, & Young, 2018; Kim et al., 2019), thus adding another trade-off dimension.

5.3.6 Deep Q-learning extensions and Robustness

As described in section 4.8, just using a neural network as a function approximator can lead to instability and convergence problems. To solve this, nearly all DQN-models use some type of experience replay and many use target networks. As such, DQN with experience replay and target networks can be considered the base or vanilla DQN algorithm.

Additionally, many authors have attempted to improve the stability and performance of TSC algorithms by using **rainbow agent extensions (i.e. double Q-learning, dueling networks, prioritized experience replay, noisy nets, multi-step learning, distributional RL)**. To evaluate the performance improvements, authors conducted ablation studies or compared the performance against regular DQN. Pol and Oliehoek (2016) evaluated multiple DQN algorithms: base DQN, double Q-learning DQN, prioritized replay DQN and DQN with batch normalization. They found that prioritized experience replay led to an increase of average rewards, that double DQN tends to get stuck in local optima and thus does not improve performance and that batch normalization leads to less stability. Nawar et al. (2019) evaluated the performance of a DQN algorithm with compact rainbow extensions (i.e. the 3 rainbow extensions: prioritized experience replay, multi-step learning, distributional RL) to regular DQN. In the experiments the compact rainbow agent was more stable and led to lower vehicle waiting times, lower trip times and lower fuel consumption compared to regular DQN. Fang et al. (2019) conducted extensive experiments and compared DQN with prioritized experience replay, double Q-learning and dueling networks to (1) regular DQN (2) double DQN with random experience replay (3) dueling DQN with random experience replay (4) double dueling DQN with random experience replay. The results showed that the double dueling controller with prioritized experience replay outperforms all other controllers. Prioritized experience replay was most crucial in performance improvement, leading to around 15% performance improvement compared to regular DQN. Double Q-learning and dueling networks each added around 5% additional performance improvements. Liang et al. (2019) conducted ablation studies on a DQN algorithm with double Q-learning, dueling networks and prioritized experience replay and also compared it to regular DQN. They found that all three extensions contributed to faster learning times, higher rewards and improved performance metrics. From the experiments we can conclude that using rainbow extensions[2] leads to performance improvements. Especially prioritized experience replay and dueling networks gave positive results. Double Q-learning led to improved performance in experiments by Fang et al. (2019) and Liang et al. (2019), but not for Pol and Oliehoek (2016). Overall, the results are in line with the original rainbow experiments by Hessel et al. (2018).

Wei et al. (2018) recently proposed two new extensions: **memory palace** and **phase gates**. In ablation studies, the memory palace method improves results only for unbalanced traffic scenarios, while the phase gate improves the performance in all scenarios. Zeng et al. (2019) used the idea of memory palaces and also tested the extension of **mixed Q-networks** (i.e. a network in which the softmax outputs are replaced with fuzzy classification results). They found no performance differences between regular DQN, DQN with memory palaces, DQN with fuzzy classification or DQN with both extensions. As such, the effects on performance remain unclear.

Lastly, some authors have experimented with using **recurrent Q-networks**, specifically using long short term memory (LSTM). Other studies assume that states are 100% observable and that sensors provide 100% accurate information at all times. However, sensors are not ideal, and mistakes can happen, leading to false or missing state inputs. Recurrent networks are used to combat this since they can process sequential information (i.e. they can learn to understand vehicle trajectories and integrate this historical data with current sensor inputs) (Choe et al., 2018; T. Zhao & Wang, 2019). Three authors compared recurrent DQN with base DQN. T. Zhao and Wang (2019) found that recurrent DQN slightly outperforms DQN for 100% observable states, but significantly outperforms regular DQN in terms of rewards, waiting times, robustness and stability for less than 100% observable states. Choe et al. (2018) found that recurrent DQN reduced average travel times by 23% and overall vehicle waiting times by 10% compared to regular DQN. Lastly, Zeng et al. (2019) found that recurrent DQN led to more efficient exploration. In 100% observable states, regular and recurrent DQN led to similar performance, but under less than 100% observability recurrent DQN significantly outperforms regular DQN. From the experiments, it can be concluded that recurrent DQN can significantly improve the controller's performance, especially if states are not 100% observable.

5.3.7 Traffic generation

To train and evaluate the controller, it needs to be exposed to traffic scenarios. Scenarios can be constructed in different ways. To create a scenario, researchers must specify both arrival rates and turning ratios for each leg and lane. Some relevant differences are described below:

- **Under-saturated traffic vs saturated traffic vs over-saturated traffic.** Different types of traffic saturation exist. In under-saturated conditions, the vehicle arrival rate is lower than the potential intersection throughput; in saturated traffic the arrival rate is equal to the potential throughput; and in over-saturated traffic the arrival rate is higher than the potential throughput. Under-saturated traffic can e.g. be found during the night, saturated traffic during early afternoons, and over-saturated traffic during rush-hour.

[2] This conclusion has not yet been proven for noisy nets.

- **Constant vs dynamically changing traffic.** Researchers can either model traffic such that the arrival rates remain constant over time, or as changing over time. Poisson distributions are by far the most popular traffic generation method, but other distributions are suitable as well (Genders & Razavi, 2016). In DQN literature, examples of sinusoidal functions (Du et al., 2019), Weibull distributions (Genders & Razavi, 2016; Vidali, 2018) and Burr distributions (Genders & Razavi, 2016) were found.
- **Balanced vs unbalanced traffic.** Traffic can either have similar arrival rates between lanes (i.e. balanced) or unequal (i.e. unbalanced).
- **Synthetic data vs real-world.** Arrival rates and turning probabilities can either be created synthetically, or data from real-world traffic can be used. For proof-of-concept models, synthetic data may be better suited, since all factors are controllable. For real-world implementations, historical traffic data from the intersection is more useful, as it allows the controller to optimally adapt to the specific intersection.

The exact choice of traffic scenario(s) will depend on the purpose of the model. Nevertheless, some general aspects can be discussed.

To train a model, traffic scenarios should be stochastic in nature. If training scenarios are deterministic, this would mean that the agent is trained over and over on the same scenario, which can lead to severe overfitting. To avoid this, statistical distributions can be used.

Furthermore, agents will only be able to optimally control traffic in scenarios which they have been trained on (Rodrigues & Azevedo, 2019). If the scenario during testing is too different than during training, favorable performance is not guaranteed. Due to this, it is advised to train agents on many different scenarios, as long as they are relevant for the intersection in question. A major limitation in a lot of the reviewed studies was that they only consider one specific traffic scenario, rather than a mix of different ones. This only allows evaluation of a controller on that specific scenario, but not on others. This is problematic, since it has been shown that some controllers perform well in certain scenarios, but not in others (Mannion et al., 2016).

Additionally, some studies only train their controllers on constant traffic (e.g. L. Li et al., 2016; Mousavi et al., 2017; Nawar et al., 2019; Pol & Oliehoek, 2016). However, fixed-time controllers perform especially well for constant traffic flows (Abdulhai et al., 2003), and methods such as the Webster method (Webster, 1958) allow researchers to find optimal cycle times. Fixed-time controllers have even been shown to outperform RL-algorithms for constant, over-saturated traffic (Vidali, 2018). Using RL-algorithms for constant traffic only would not be worth the computational cost. Instead, the actual benefit of adaptive DQN controllers is to control dynamic traffic situations (Abdulhai et al., 2003; Yang et al., 2017).

Some good examples of models that have been trained and tested on dynamic traffic demands can be seen in M. Guo et al. (2019), Vidali (2018) and Wei et al. (2018).

5.3.8 Performance indicators

To assess the performance of a proposed controller, key performance indicators (KPI) must be gathered. Various performance metrics were found in literature:

- Gained reward
- Queue length

- Throughput / Number of passed vehicles / Number of completed trips
- Waiting time
- Travel time
- Delay
- Vehicle speed
- Green time per leg/lane
- Number of stops
- Fuel consumption
- Emissions

The found performance metrics mostly overlap with the previously discussed state and reward representations. For the specific descriptions and discussion of the metrics, please refer to sections 5.3.3 and 5.3.5.

In general, some authors report the results in terms of cumulative values, while others report values averaged over vehicles. In some papers also differences between legs/lanes are reported. Most authors also include figures showing the change of the KPIs over time within a traffic scenario. Furthermore, since the traffic scenarios are stochastic, performance measures must be evaluated over several runs. Ideally, both the averages and standard deviations of the runs should be reported and discussed.

5.3.9 Base case(s)

To evaluate the performance of a proposed model, researchers compare their model against other models, the so-called base cases. Different base cases were found in literature:

- **Fixed-time control**
- **Other traditional TSC:** actuated control, longest queue first, time-loss based control, traditional adaptive control
- **Earlier proposed state-of-the-art RL-methods** (including non-DQN methods)
- **Other, less sophisticated RL-algorithms:** tabular Q-learning, shallow NN RL-algorithms
- **Model variations:** Ablation studies on their model (i.e. their model minus certain extensions)

The most common base case used was fixed-time control. However, fixed-time control may not be the best base case, since RL-algorithms were shown to outperform or perform equally well as fixed-time control in nearly all cases. Ideally, both ablation studies and comparisons with state-of-the-art controllers would be conducted to assess the performance in different scenarios and using different KPI. However due to time and space limitations, this is oftentimes not done.

5.4 Mixed traffic scenarios and POMDP in traffic signal control

The second part of the literature review focused on research that applied RL methods (not specifically DQN) to TSC in mixed human and connected or automated vehicle environments.

The problem of mixed traffic is presented in Figure 8. The left side of the figure shows the traffic situation for 100% connected vehicles. Since all vehicles communicate information about themselves, the full intersection state can be observed. On the right side, the CV-penetration rate is less than 100%. Now assuming that traffic is only observed via V2X communication, the agent will only be able to observe the connected vehicles, leading to incomplete state representations. The unconnected vehicles essentially become invisible to the agent. Since states are now no longer fully observable, the problem is formulated as a partially observable Markov decision process (POMDP) (Arulkumaran et al., 2017).

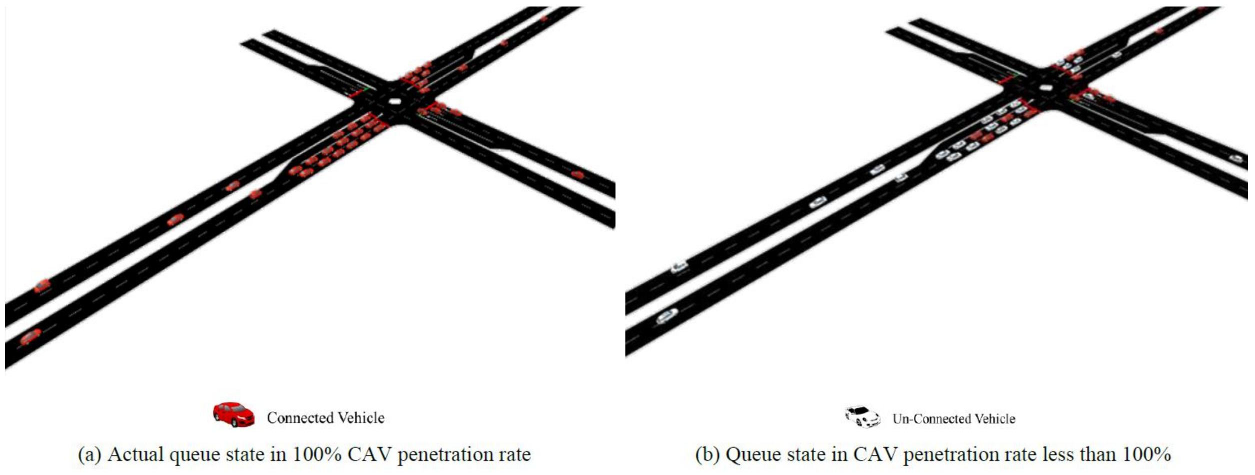

Connected Vehicle
Un-Connected Vehicle
(a) Actual queue state in 100% CAV penetration rate
(b) Queue state in CAV penetration rate less than 100%

Figure 8 Mixed connected and non-connected vehicle traffic. Adapted from Islam et al. (2019).

POMDP environments do not only happen in case of mixed traffic but can also happen due to incomplete monitoring of the environment (e.g. due to sensor breakdowns, no sensors in a specific area, obstructed camera vision), wrong information (e.g. due to sensor malfunctions) or communication delays. The real-world is full of uncertainties, so it is practically impossible to have a complete and correct state representation, making all practical RL-TSC applications POMDP (Zeng et al., 2018). The following paragraphs summarize the studies on RL under partial observability due to mixed traffic.

The earliest found experiments with different CV-penetration rates (40%, 60%, 80%, 100%) were conducted by (Yang et al., 2017). They used a DQN-algorithm with a state representation consisting of the sum of squared delays of the North/South and East/West approaches of all connected vehicles, the current phase and the elapsed phase duration. The reward representation consisted of the cumulative squared delay of all vehicles (connected and unconnected). Since they did not apply stabilizing add-ons (e.g. experience replay, target networks), their algorithm was too unstable to gain any usable results.

Islam et al. (2019) studied the impact of different CAV-penetration rates for tabular Q-learning. The performance was tested under different traffic demand rates on two real-life networks of decentralized 4-way intersections. The state representation consisted of the normalized and discretized queue length for each leg, estimated based on the last CAV to join the queue (i.e. it is assumed that all space in front of the last CAV is also occupied by queued vehicles). They found that travel times, queue times and energy consumption improved for penetration rates over 20-40%. In network 1, the controller reaches the same or better performance levels as the current real-life TSC strategy for penetration rates around 40-50%. For lower penetration rates no clear trends were found. Higher standard deviations at lower penetration rates indicate unstable controllers, which is due to the low or inexistent observability of queue lengths. They conclude that as expected, higher penetration rates lead to better performance. They also state that it is not possible to specify a critical penetration at which RL outperforms traditional controllers since the RL performance strongly depends on the design of the specific scenario.

Zeng et al. (2018) compared the performance of two deep Q-network algorithms: regular DQN and recurrent DQN. In both cases, the state representation consisted of a 2-layered DTSE of traffic density and average normalized speed observed only for connected vehicles, as well as the one-hot encoded current traffic phase. The reward consisted of the change in the number of halting vehicles between time steps (both observed and non-observed vehicles). The authors trained and tested agents under penetration rates of 10%, 25%, 50%, 75%, and 100%. The results (see Figure 9, left side) showed that both the regular and recurrent DQN agents reached stable and comparable performance for 100% penetration rates. However, regular DQN performed much worse for lower penetration rates, while recurrent DQN produced stable average waiting times under penetration rates even as low as 10%. Additionally, the authors tested the robustness of the agents when trained under a penetration rate of 50% but tested under different penetration rates. The results (see Figure 9, right side) showed that the recurrent DQN agent produced lower average waiting times than regular DQN for a wider range of penetration rates, making it more robust. The authors conclude that recurrent DQN is more robust to penetration rate differences since it allows the agent to carry historical data over to the next phase which allows it to make correct decisions even without seeing the current environmental state.

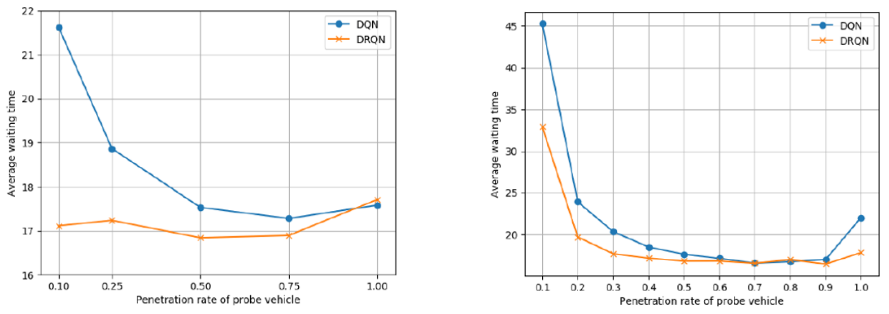


Fig. 5. Average waiting time in different penetration rates. Agent is trained and tested in the same penetration rate.

Fig. 6. Average waiting time in different penetration rates. Both of the agents are trained under the penetration rate of 50% but tested in different penetration rates.

Figure 9 Results from Zeng et al. (2018).

T. Zhao and Wang (2019) conducted a similar study. Their state representation consisted of a 2-layered DTSE using vehicle position and normalized speed. Change in cumulative delay was used as a reward. They trained and tested agents under unsaturated, near-saturated and over-saturated traffic flows. All results are in line with the results from Zeng et al. (2018) (compare Figure 9 and Figure 10). Regular and recurrent DQN performs similarly for a 100% penetration rate, but recurrent DQN significantly outperforms regular DQN for lower penetration rates (see Figure 10, middle). The higher the penetration rate, the better the agent performs. The sensitivity analysis of different penetration rates showed that a recurrent DQN agent

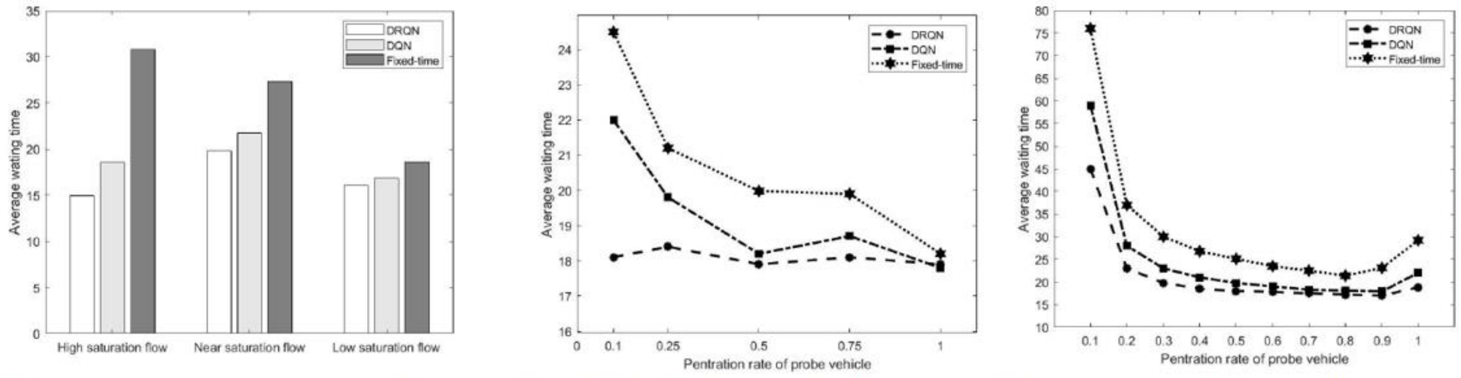


Fig. 4. The average waiting time of three traffic density network structures

Fig. 5. Average waiting time of different permeability

Fig. 6. Average waiting time of different permeability

Figure 10 Results from T. Zhao and Wang (2019).

trained under 50% penetration is more robust to penetration rate changes than regular DQN (see Figure 10, right side). Additionally, unlike Zeng et al. (2018), T. Zhao and Wang (2019) also compared the agents when trained and tested under different traffic flow rates. The results show that recurrent DQN outperforms regular DQN under all traffic flows (see Figure 10, left side).

R. Zhang et al. (2020) applied DQN for partial vehicle detection under 0%, 10%, 20%, 40%, 60%, 80% and 100% CV-penetration rates for a simplified 4-way intersection without turns. They used a weighted state representation consisting of the distance to the nearest vehicle at each approach [m], the number of detected vehicles at each approach, a yellow light phase indicator (binary), the current discretized time [h since midnight] and the current phase (encoded within the detected car count and distance to the nearest detected vehicle: negative values if red, positive values if green). The reward consisted of the average vehicle delay. Like in (Zeng et al., 2018), agents were trained and tested under a priori fixed penetration rates. Controllers were trained and tested under either sparse, medium or dense traffic flows. The authors expected the following results: for very low traffic rates, the controller will react to each vehicle individually, as if they were particles. If a connected vehicle arrives, the controller will switch to a green phase instantly. Unconnected vehicles cannot be recognized by the controller, so they will have to wait until the maximum green time has elapsed. For very high flow rates, the individual vehicle will become less important, since vehicles are part of a “liquid” rather than particles, so all vehicles will have similar speeds and behaviors. Due to this, it is expected that connected and non-connected vehicles will have similar waiting times.

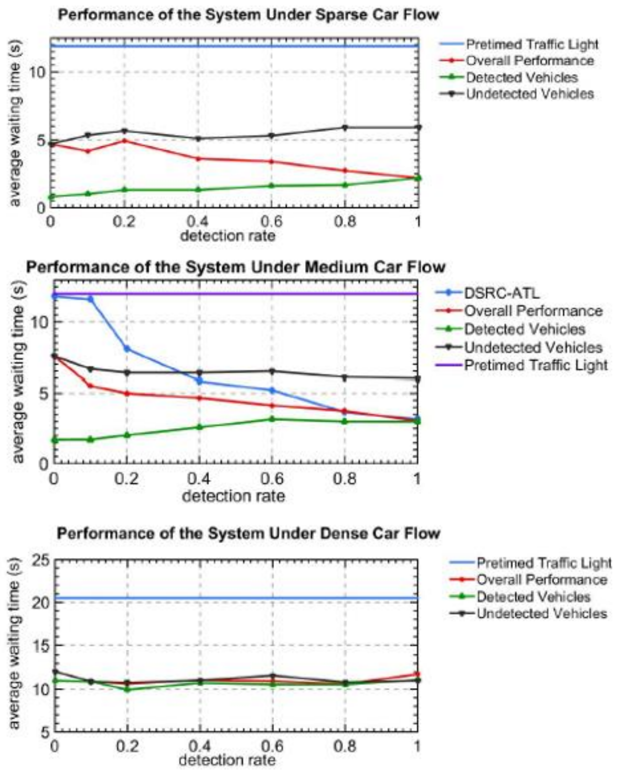


Figure 11 Waiting time under different detection rates and different traffic flows. Results from R. Zhang et al. (2020).

The experiments confirmed the authors' hypotheses. For medium traffic (see Figure 11, middle), the DQN algorithm outperforms a fixed-time controller even under low penetration rates. As expected, the overall performance increases with higher penetration rates. Approximately 80% of this performance improvement happens within the first 20% penetration rate. Additionally, as hypothesized, they found that the waiting time for detected vehicles is much lower than for undetected vehicles, even though the agent gets rewarded for reducing the waiting time of both types of vehicles. The results for sparse traffic flow (see Figure 11, top) are similar to the medium traffic flow. For dense traffic flow (see Figure 11, bottom), as expected, the performance of the controller does not improve for higher penetration times and waiting times for both detected and undetected vehicles are similar throughout. The authors conclude that the agent can perform well under different penetration rates and can seamlessly adapt to dynamic traffic flows. Furthermore, the authors conducted sensitivity analyses to different flow rates and penetration rates. They found that controllers trained under one flow rate or penetration rate but tested under another were able to adapt to different flow rates or penetration rates, as long as they were not radically different. This shows a certain level of robustness.

From the reviewed literature we can conclude that it is possible to use DQN to control partially observable traffic even under low CV-penetration rates. Nevertheless, higher CV-penetration rates lead to better performance, which can be explained by the fact that the agent has more complete information about the environment to make a decision. Agents behave differently under different flow rates. Lower traffic flow rates lead to shorter waiting times for detected vehicles, but worse waiting times for undetected vehicles. For high traffic rates, this difference disappears. Lastly, when using recurrent DQN compared to regular DQN, the agent's performance and robustness to penetration rate changes significantly improve.

5.5 Identified Literature gaps and suggested future research

Within the reviewed literature on using DQN in TSC and using RL in mixed TSC, many literature gaps were identified. This section will outline them.

Unclarity about best state representation. In literature, many different state representations were found, ranging from simple to complex. Authors argue for and against different representations, but only a few authors attempted to do cross-comparisons between different state representations. In these cross-comparisons authors compared high-dimensional state representations (e.g. DTSE) against low-dimensional state representations (e.g. queue length) (e.g. Genders & Razavi, 2018; Mousavi et al., 2017). Yet, a cross-comparison between different state representations of similar or same information density (e.g. one-hot encoded current green phases vs +/- encoded green phases) is lacking. As such, for future research it is suggested to conduct a systematic study that compares different state representations. In these studies, all model parameters should be kept constant, apart from the state representation. It should be evaluated how different state representations impact model performance and stability, and under which circumstances which state representation is most suitable.

Unclarity about the best reward representation. Like for the state representation, many different reward representations have been suggested in literature. Picking a suitable reward representation may arguably be even more difficult since the full reward needs to be aggregated into a single value. Yet, it was seen that different rewards have different advantages and disadvantages, with no reward being objectively the best. To combat this, some researchers have attempted to use weighted reward functions, yet this causes the issue of how different sub-rewards should be weighted. Overall, like for the state representation, a systematic study comparing the impact of different reward representations on model performance and stability is lacking. Research needs to investigate which rewards work under which circumstances.

Lack of systematic fine-tuning of agents. When designing agents, many hyperparameters have to be set (e.g. the number of episodes to train the model, the target network freeze interval, the memory size). These hyperparameters influence model performance and stability. Yet, not all papers report these values, and nearly none of the studies present how they determined these values. For practice this means that research from previous studies are not reproduceable, but also that it is more difficult to design new agents.

Balancing algorithm goals. When designing a traffic signal controller, several goals have to be balanced: traffic efficiency, safety, fairness to all traffic participants, driver comfort, environmental impact, etc. In current literature on RL in TSC however, most of the reviewed papers focused only on traffic efficiency, and only a few focused on energy or fuel-efficiency or minimizing emissions. Yet, the other goals are oftentimes neglected. Thus, research is needed on how to include and balance the different goals.

> **Fairness.** When designing a controller, only a few researchers explicitly considered fairness. Yet, fairness is an important aspect of TSC. A pure traffic efficiency conscious controller would likely "sacrifice" the few vehicles for the many. Some authors attempted to solve this by imposing maximum phase constraints, yet this does not solve the problem that the agent is not actively aware of fairness. Others have experimented with using squared delays to prioritize vehicles which have been waiting for a long time over vehicles which have just arrived. Yet, overall fairness does not gain enough attention. Future research could attempt to explicitly include fairness constraints within a weighted reward.
>
> **Impact on safety.** No studies were found which explicitly investigated the impact of their controller on traffic safety. Some authors included boundary conditions to ensure safety (e.g. yellow and red clearance phases), but none of the studies explicitly included it in the agent's goal.

Lack of variety in traffic scenarios and network topologies. Most literature uses uniformly distributed and balanced traffic scenarios. Yet, as discussed this is unrealistic in the real world, and it does not allow the RL agent to show its true advantages compared to fixed-time controllers. Furthermore, most controllers are only tested on artificial 4-way intersections. However, in the real world many different types of intersections exist. Overall, there is a need to train and test RL agents on a wider range of traffic scenarios (e.g. dynamic, unbalanced, different saturation levels) and network topologies (e.g. 3-way intersections, non-geometric layouts), in order to be able to evaluate for what types of intersections RL would be useful and for which intersections other controllers would be more suitable.

Lack of other traffic participants. All investigated RL agents focus only on vehicles (i.e. regular cars). Other traffic participants, e.g. buses, trucks, cyclists, pedestrians and public transport have been excluded. Excluding them makes models much simpler, but also less realistic. Future research could investigate the ability of RL agents to accommodate these other traffic participants, particularly regarding safety.

Robustness. Only few papers have investigated the robustness of algorithms (Rodrigues & Azevedo, 2019). Trained RL agents could however break in many ways: signal failures could happen, communication between traffic participants and the infrastructure could be delayed or broken down, road accidents or other unexpected traffic situations could happen, or traffic demand could radically change from the trained scenarios. Good TSC should show a certain level of robustness to the kinds of problems. First suggestions on how to achieve this have been made, but further research will be needed.

Online learning. Related to the problem of robustness is that of online learning. Generally, once an agent is trained and implemented in an intersection, it will only perform greedy actions. This however means that the agent is unable to learn while it is operating, and thus unable to adapt to slow changes in traffic scenarios. A possible solution for this would be to retrain the agent at regular intervals, but this is time-consuming and expensive. Another solution would be to train the agent while it is online, under the condition that when the agent does explorative actions, traffic is not unnecessarily hindered. Strategies on achieving this are left for future research.

Impact of mixed traffic scenarios. In the nearby future, traffic will consist of a mixture of conventional and connected and/or automated vehicles. RL TSC may exploit data from connected vehicles to gain better results. Yet, most researchers assume all vehicles are connected, and only few researchers have investigated CV-penetration rates of less than 100%. The recurrent DQN agent has led to promising results. However, other agents may lead to even better results.

Furthermore, the problem with current mixed traffic agents is that they must be trained under a fixed penetration rate. This is problematic since it is likely that traffic will consist of different mixtures of CVs and RVs (e.g. different penetration rates at different times of the day or week). Ideally, an agent would be able to accommodate different mixtures of vehicle types. How this can be achieved is currently unknown.

Problematic model validation. Due to time, cost, safety restrictions and possible public inconvenience caused by sub-optimal strategies it is not possible to test newly proposed TSC on real-world intersections. Because of this, controllers must be tested in simulators. However, to get valid results, the simulator must be able to accurately simulate real-world traffic and driving behaviors. While simulators can represent many types of traffic situations, simulators are never able to capture real-life behaviors 100% accurately. Thus, to completely validate whether proposed strategies would work in the real-world, field studies will be needed.

Lack of benchmarks and systematic comparison between controllers. Since deep RL is such a fast-developing field, new agents are being proposed continuously. Researchers are not only experimenting with the state, reward and action representations but also with the types of RL models and extensions. In recent years, many new DQN extensions and other RL algorithms have been successfully applied in other domains. Many researchers have started to implement these controllers also for TSC. Yet, up to date there is a lack of systematic comparisons between controllers. Many researchers compare their agents to traditional TSC, but only few compare them to state-of-the-art RL controllers (e.g. Fang et al., 2019).

Part of this problem may be caused by the fact that unlike in some other domains, there are no standardized test scenarios TSC. For the development of current deep RL agents for example, many researchers test their algorithms on standardized environments such as the Cartpole problem, the Doom game (Lample & Chaplot, 2017) or the Atari 2600 benchmark (Hessel et al., 2018; Mnih et al., 2015). Many of these environments have been combined into an easily accessible toolkit called OpenAI Gym (Brockman et al., 2016). Having such standardized environments with standardized scenarios enables a better comparison between model types and allows benchmarking, and thus speeds up algorithm evaluation (OpenAI, 2016). Future research may look into developing such an environment.

5.6 Summary

Overall it can be concluded that deep RL in TSC is a promising new field that is developing quickly. New controllers are continuously being proposed, and many researchers are working on increasingly better-performing algorithms.

When designing an RL agent for TSC, many design choices have to be made. In literature, many different design decisions must be taken regarding the state, action and reward representation, the RL agent, network topologies, traffic generation, KPIs and base cases. Yet, systematic comparisons and best practices for these choices are lacking. Much of the design process currently seems to be trial-and-error based.

At the end of the review, many research gaps have been suggested and future research directions have been outlined. To facilitate better systematic comparisons between proposed agents, it is highly suggested to develop standardized environments with standardized scenarios in order to benchmark controllers. This will allow for faster development and evaluation.

6. Conceptual Design

In this section the conceptual design of the reinforcement learning strategy will be discussed. First, the problem will be scoped and assumptions and simplifications will be explained. After this, the environmental setup of the simulations will be described. Lastly, design decisions about the reinforcement learning agent itself will be outlined and evaluation methods discussed.

6.1 Scope, assumptions and simplifications

As described, modeling intersections is complex. To reduce the problem, certain scoping decisions must be made. In this project, only the effects of the traffic control method on traffic performance will be investigated. Human behavior, route choice, data security, connectivity problems and other limitations of previous studies (see section 2.3.3) fall outside the scope. Furthermore, it was decided to only focus on traffic efficiency. Impacts on other factors such as safety, fairness to all traffic participants, environmental factors, passenger comfort and social perception are out of scope. Only urban signalized intersections are considered, rather than unsignalized, rural or highway intersections. Only isolated intersections rather than networks of intersections will be analyzed. Furthermore, this research will only design an agent that controls the traffic signals. An agent which controls vehicle trajectories is out of scope.

As mentioned, the focus is on the transition period between conventional vehicles to connected and/or automated vehicles. To simplify the problem, only connected and regular vehicles will be included in the system. CAVs may be included in later research.

Other assumptions for this project are based on common assumptions from other Intelligent signalized intersection management studies (L. Chen & Englund, 2016; Florin & Olariu, 2015; Q. Guo et al., 2019; Yang, Guler, & Menendez, 2016):

- All intersections and CVs are assumed to be equipped with communication devices
- Communication is ideal (no data loss, no latency, no security issues, no compatibility issues)
- Perfect information on CV states is available (e.g. on position, speed)
- No overtaking, reversing or lane changing is considered
- Vehicle characteristics of conventional vehicles of CVs are assumed to all have the same (stochastic) settings (e.g. dimensions, driving behavior)
- No traffic uncertainties (e.g. accidents, road works) are considered
- Only vehicles are modeled, other modes (e.g. cyclists, pedestrians, public transport) are excluded

6.2 Environmental setup

6.2.1 Intersection layout

In this thesis, it was chosen to use a hypothetical intersection layout, rather than to perform a case study on a real intersection layout. Using a hypothetical layout gives the modeler more control over the environmental conditions while using a case study intersection makes the model more adapted to a specific intersection. For both the hypothetical and case study intersection layout there is a risk that the controller model is only adapted to this specific network, thus it may not generalize well. To combat this risk, ideally the control strategy would be evaluated on several networks. Yet, due to time constraints this was not possible within this thesis.

In this thesis, it was decided to use a hypothetical 4-way intersection. Using a hypothetical intersection eliminates the need to find a suitable real-life intersection and allows full control over the design. Using a 4-way intersection has several advantages: (1) it is a geometric, which means that there will be no differences in controller performance between the different legs due to differences in lane or leg configuration (2) nearly all published RL algorithms are based on 4-way intersections, which makes the results of this project easier comparable to past studies (3) 4-way intersections are common types of intersections. To make the intersection more realistic compared to some other studies (see section 5.3.2), traffic is not restricted to only through traffic. Instead, vehicles can travel straight, right or left.

The finally chosen layout can be seen in Figure 12. The layout consists of a single isolated 4-way intersection. Each leg is bidirectional and has 4 incoming lanes. The most left lane is for left-turning traffic only, the two middle lanes are for through-traffic and the right lane is used for both right-turning and through-traffic. Each leg is 750m long, as measured from the edge origin to the intersection stop line. The maximum speed in the whole intersection system is 13.89m/s (50km/h).

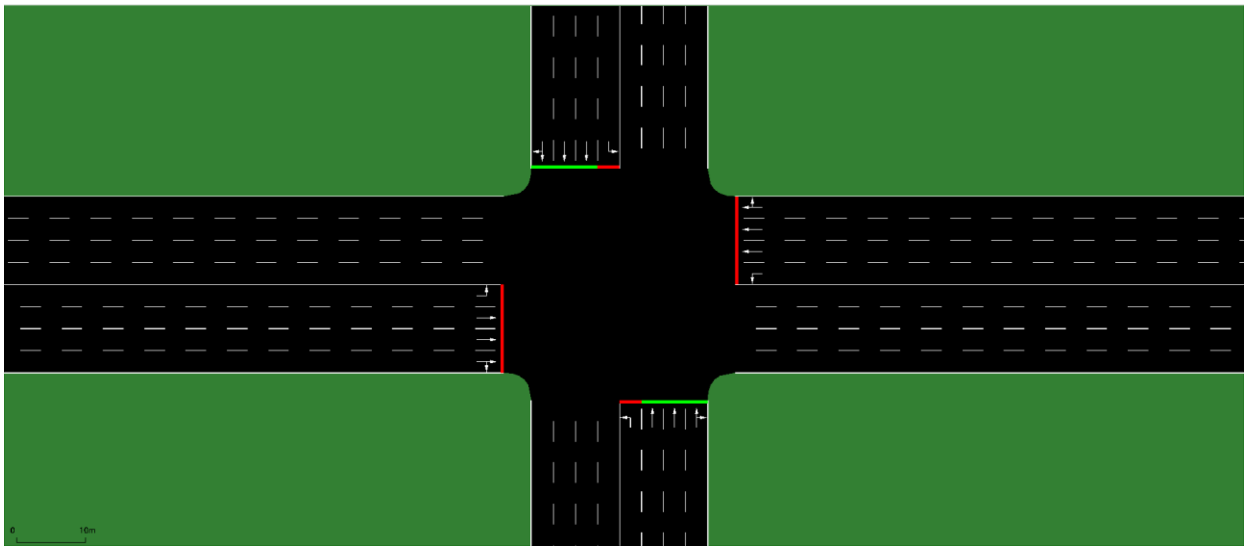


Figure 12 Intersection layout

6.2.2 Traffic signals

In the intersection, there are 8 different traffic lights, 2 for each of the four legs. One traffic light per leg controls through and right-turning traffic, thus it controls the three right lanes. Another traffic light controls left-turning traffic, so it only controls the left lane. Vehicles approaching the intersection automatically chose the appropriate lane queue for their intended destination.

Traffic movements happen in 4 different green phases, as seen in Figure 13. There are two straight-and-right going phases (with origins North/South or East/West) and two left-turning phases (also with origins North/South or East/West). We can denote the green phases as follows: {NSG, NSLG, EWG, EWLG}, where NS stands for North/South, EW for East/West, L for left-turning, and G for green phase. This choice in traffic phases and traffic movements is the most commonly chosen option in literature. This layout is also common in real-life.

Like in real-life, each traffic light has three different states: red, yellow or green. Transitions between these colors always happen in the order <red-green-yellow-red>. Some previously conducted studies have

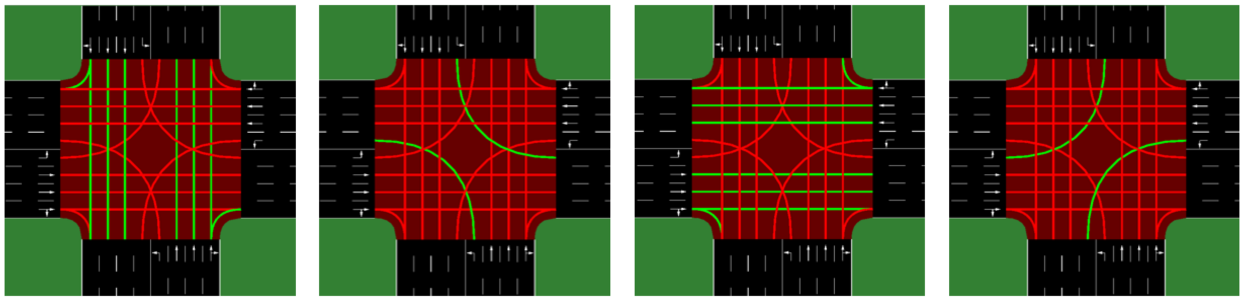


Figure 13 Four possible green phases. From left to right: (1) NSG: right-turning and through traffic with origins North/South (2) NSLG: left-turning traffic with origins North/South (3) EWG: right-turning and through traffic with origins East/West (4) EWLG: left-turning traffic with origins East/West.

excluded yellow phases, however in real-life yellow phases are needed for safety reasons and to clear the intersection. Since yellow phases will add several seconds of transition time between green phases during which vehicles cannot begin to cross the intersection, yellow phases cause lower traffic throughputs. If we exclude yellow phases from the model, this would not allow for realistic phase transitions. As such, whenever a transition between green phases takes place, there will be a yellow light phase. Yellow phases last for 4 seconds.

The model is always in exactly one of the 8 described phases (i.e. one of the 4 green phases or one of the 4 yellow phases). Not all traffic lights may be red at the same time.

6.2.3 Choice of traffic scenarios

As was concluded in the discussion on traffic generation (section 5.3.7), RL agents can only learn good traffic control strategies for scenarios that they have encountered before. As such, the agent must be exposed to all types of scenarios which it should be able to handle during real-life implementation. If an agent were to be designed for a real-life intersection, the agent would be trained on gathered real-life traffic data. In this thesis however a hypothetical thesis is being modeled, thus the traffic demand will have to be constructed manually.

For this thesis it is assumed that traffic demand from all four legs of the intersection is balanced, i.e. the arrival rates from all four sides are equal. Additionally, it is assumed that all arriving traffic has the same probability to go straight, to make a right or to make a left turn. Making these assumptions radically cuts down the number of possible traffic scenarios and thus speeds up training. A drawback is that the agent will not be able to control these excluded traffic scenarios optimally.

Since many previous studies were limited by the fact that they only studied constant arrival rates, this thesis will include both scenarios with constant and dynamic arrival rates. All scenarios will be stochastic, i.e. they will be randomly generated using probability distributions. As stated before, training agents on deterministic scenarios can lead to severe overfitting, meaning that the agent would not be able to generalize well to other similar scenarios. In this study, the scenarios with constant arrival rates are generated using a random uniform distribution. The scenarios with dynamic arrival rates are created using a Weibull distribution of shape 2, as suggested by Vidali (2018). Weibull distributions of shape 2 are well suited to generate traffic scenarios from the beginning to the end of rush hour: the traffic demand starts low, quickly increases until it reaches its peak, and then gradually decreases again until it reaches low traffic demand levels.

Figure 14 shows examples of traffic demand generation per simulation time step. The left figure shows a uniform distribution of 2000 cars over 5400 time steps. Note that while the distribution is uniform, the number of cars generated at each time step are not exactly equal, since it remains a stochastic process. The right figure shows a shape 2 Weibull distribution with the same number of time steps and generated cars. It can be seen that the number of cars over time at the peak is much higher than for the uniform distribution, while at the tails the number of cars over time is much lower.

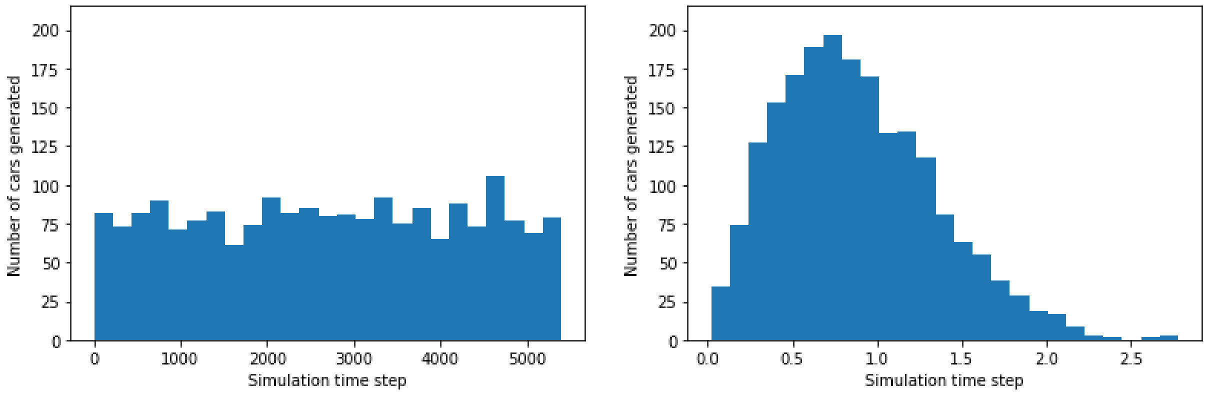


Figure 14 Examples of car generation distributions. The left figure shows a random uniform distribution. The right figure shows a Weibull distribution.

For the controller in this thesis, it was decided that it should be able to control different amounts of arriving traffic. It was decided against using only one type of scenario (e.g. only constant traffic generation with a fixed arrival rate) since this is unrealistic in the real world. In the real world intersections must accommodate rush hour traffic, regular day time traffic and night traffic. These situations generally differ in the number of cars which arrive.

To model this, it was decided to use three different types of arrival patterns: high or saturated traffic (corresponding to rush hour traffic), medium traffic (corresponding to regular day time traffic) and low or under-saturated traffic (corresponding to nigh traffic). Over-saturated traffic scenarios are excluded. The reason for this is that in over-saturated traffic scenarios the vehicle arrival rate is higher than the maximum potential intersection throughput. Thus, whatever phase the agent decides on, it will receive a negative reward. This results in agents that are unable to learn optimal strategies. Additionally, as discussed previously in section 5.3.7, fixed time controllers are the optimal TSC strategy for over-saturated traffic.

6.2.4 Vehicles

Several assumptions about the vehicles in the simulation were already described in section 6.1.

In general, vehicles will be modeled in a similar fashion. All modeled vehicles will be regular cars, i.e. there will be no trucks, buses, bikes, etc. All generated vehicles have similar behavior (e.g. vehicle following behavior) and characteristics (e.g. vehicle length, driving speed, acceleration/deceleration). Keeping vehicle parameters similar for all vehicles allows for faster training and better evaluation of the effectiveness of the controller. However, to ensure more realism in the scenarios, vehicles will differ slightly in driving speed, acceleration and following behavior. These parameters will be stochastically chosen for each vehicle.

A major difference between vehicles is the vehicle type. Vehicles can be either connected vehicles or regular vehicles. As discussed in section 2.1, connected vehicles can communicate their position, speed and other characteristics to the controller, while regular vehicles cannot. In this thesis, it is assumed that connected vehicles can only communicate information about their own states to the controller, but not about surrounding vehicles. Each controller will be trained under a certain penetration rate of connected vehicles. To evaluate how well the agent can control traffic under different penetration rates, several agents will have to be trained under different penetration rates.

6.3 Reinforcement Learning Model

6.3.1 States

From the reviewed literature it was concluded that many possible state representations exist, yet it is not clear which representation would be the best and for which scenario.

In this thesis, it is assumed that the state is observed by connected vehicles, rather than by traditional infrastructure sensors like loops or radars. As such, the available information is vehicle-based. Most researchers agree that it is better to use raw data (e.g. individual vehicle positions and speeds), rather than aggregated data (e.g. queue lengths). Many argue that the more information a controller has available, the better decisions it could potentially learn. Yet, adding more types of information also increases the state space, which increases training time.

For this project, a compromise between the amount of available information and training time was made. Since the available data is vehicle-based, DTSE is a suitable state representation. A problem with DTSE is that the chosen cell length may greatly affect the result. Most papers chose a cell length of slightly longer than the vehicle length, so that only one vehicle can fit into the cell. Yet, if such a short cell length is chosen, the number of cells to represent a full lane gets very large, and thus will take long to train. Additionally, cells that are closer to the stop line may be more valuable for the controller than for cells that are further away. Due to this, less resolution in these further away cells may be required. Thus, it was decided that the cell length for cells closer to the stop line should be short (i.e. a cell length just slightly longer than one vehicle), and cells further away should get increasingly longer. Additionally, since there are two traffic lights per leg (one traffic light controlling three lanes and one traffic light controlling one lane), it was decided to use a width of two cells per leg. One of the cells encodes vehicles on the three righter lanes, and one cell encodes vehicles on the left lane. It would be possible to use a width of four cells per leg (i.e. one cell per lane on the leg), but this would drastically increase the state space, while not

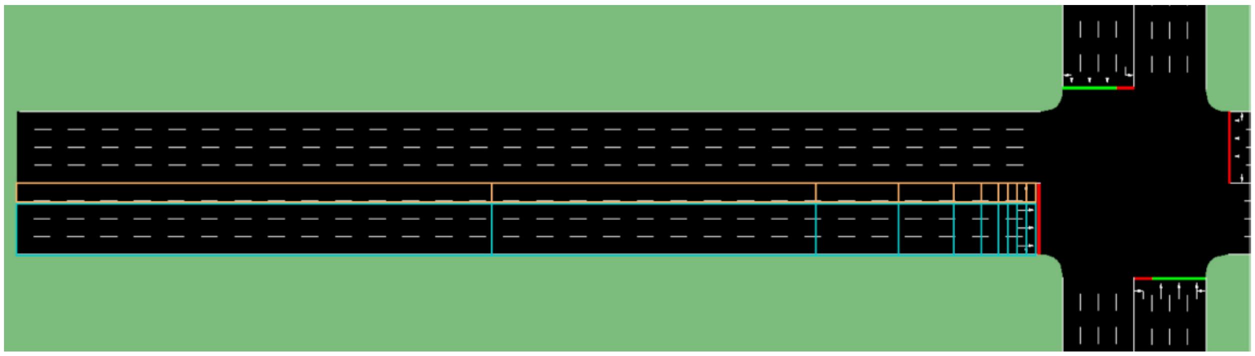


Figure 15 One leg of the intersection overlaid with DTSE cells. Cells closer to the stop line are shorter, while cells which are further away get progressively longer. Figure adapted from Vidali (2018).

adding much useful extra information, as all three lanes are controlled by the same signal. This approach was proposed by Vidali (2018), and the same layout will be used in this thesis

Figure 15 provides an overview of the chosen cells for a single leg. In total 10 cells are used to encode the full length of a lane. Since a width of 2 cells is chosen per leg, in total there are 8 times 10 cells, so 80 cells. The first 4 cells fit one vehicle each, and after this cell lengths get gradually larger (see Figure 16).

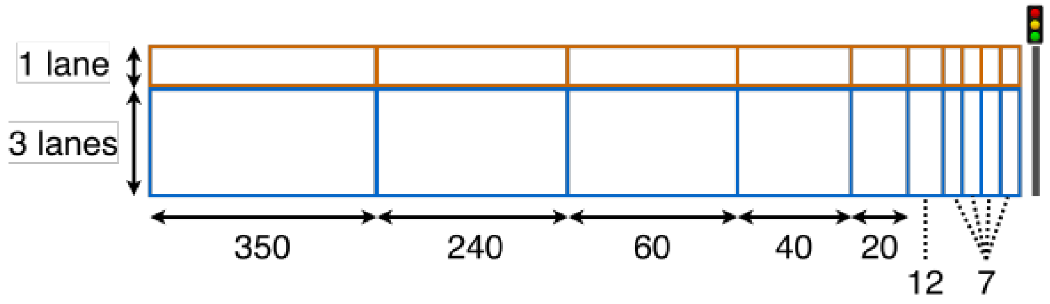
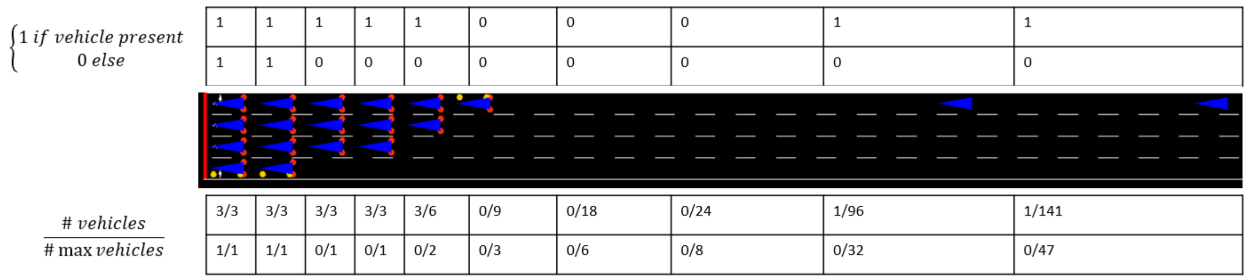


Figure 16 Cells lengths of the DTSE. The closer to the stop line, the shorter the cell length. Figure adapted from Vidali (2018).

When using DTSE, different data can be encoded in the cells, such as vehicle position, vehicle density, speed, delay, vehicle waiting time or the current phase. Most papers included a binary value for the vehicle position. The cell would include a 1 if a vehicle occupies the cell and a 0 if the cell is empty. Yet, in these papers usually a cell length of one vehicle was chosen. In our case, some cells can fit more than one vehicle, so if binary values are used some information may be lost. To overcome this, Zeng et al. (2018) used vehicle density, measured as the number of vehicles in the cell divided by the potential maximum number of vehicles in the cell. Figure 17 visualizes the two options for an example. Since it is unclear which method leads to better results, preliminary tests will be conducted with either encoding method.



$\begin{cases} 1 \; if \; vehicle \; present \\ 0 \; else \end{cases}$

1	1	1	1	1	0	0	0	1	1
1	1	0	0	0	0	0	0	0	0

$\frac{\# \; vehicles}{\# \max vehicles}$

3/3	3/3	3/3	3/3	3/6	0/9	0/18	0/24	1/96	1/141
1/1	1/1	0/1	0/1	0/2	0/3	0/6	0/8	0/32	0/47

Figure 17 DTSE of vehicle position/density. The top option shows binary position encoding. A vehicle is considered to be in a cell if its front bumper is in the cell. The lower option shows floating point encoded vehicle density measured as the relative cell occupancy. (Note: figure not to real simulation scale due to space reasons.)

Additionally, many authors had success with including a second DTSE layer which encodes the current vehicle speed as a real number. Including the speed additional to the vehicle position can be useful, since it allows the controller to distinguish between vehicles which are currently driving and about to cross the intersection and vehicles which are currently waiting. To ensure that values do not get too large, the speeds are usually normalized with the leg's speed limit. In our case, since several vehicles can occupy the cell at the same time, the average of all vehicles in the cell is used.

Lastly, including the current signal phase and the elapsed time since the beginning of the current green phase was shown to lead to well-performing agents. Thus, this will also be included in the state representation. The elapsed time will be encoded as an integer. For the current phase, two different

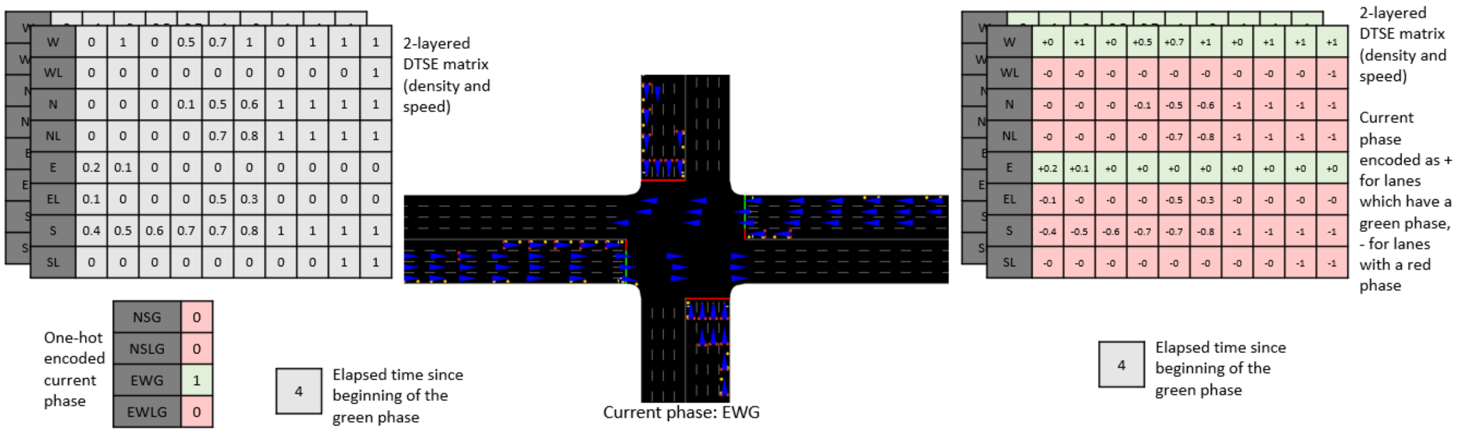

2-layered DTSE matrix (density and speed)
One-hot encoded current phase
NSG
NSLG
EWG
EWLG
4
Elapsed time since beginning of the green phase
Current phase: EWG
2-layered DTSE matrix (density and speed)
Current phase encoded as + for lanes which have a green phase, - for lanes with a red phase
Elapsed time since beginning of the green phase

Figure 18 Examples of both current phase encodings. The current green phase of the intersection in this example is EWG. The left hand side shows this phase encoded as a one-hot vector. The right hand side shows this phase encoded as a positive number in the rows corresponding to the lanes which currently have a green signal (here: lanes W and E). (Note: the values in the DTSE do not necessarily match the shown intersection).

encoding strategies will be tested. Option 1 includes encoding the phase as a one-hot vector. In option 2, the current phase will be encoded in the 2-layered DTSE. Lanes that currently have a green signal will be encoded as a positive number, lanes that currently have a red signal will be encoded as a negative number. An example of both options is presented in Figure 18. Option 1 is the more widely applied method in literature and has gained successful results, but it means that the model has an extra input dimension. Option 2 has not been widely used, but R. Zhang et al., (2020) found that it led to a more stable agent.

6.3.2 Actions

The agent's action set consists of picking one of the 4 green phases (as described in section 6.2.2) for each action time step t. As such the action set is defined as

$$A = \{NSG, NSLG, EWG, EWLG\} \in t$$

Unlike in some other previous works, the phase order is not fixed, i.e. the agent can start any green phase, leading to acyclic phase orders. This allows the controller to respond more dynamically to new traffic demand.

The next green phase will be initiated for a duration of 5 seconds. Shorter durations may provide more flexibility to the agent, but also take longer to train since the difference in reward between time steps would be very short. 5 seconds was chosen as a trade-off. Choosing a new phase for the next 5 seconds effectively imposes a minimum green time for every lane. This also prevents the controller from switching too rapidly from green to red and ensures that at least one vehicle can pass the intersection.

Note that the time interval between action time steps t and $t+1$ is not fixed. If the agent decides to remain in the same green phase, the interval between action time steps t and $t+1$ is equal to one green phase duration (i.e. 5 seconds). If the agent decides to switch to a different green phase, first the agent must initiate a yellow phase and then it can start the new green phase. In this case, the interval between action time steps t and $t+1$ is equal to one yellow phase duration plus one green phase duration (i.e. 4 + 5 = 9 seconds). To visualize this, see Figure 19. At action time step t_1, the agent is at the end of green phase 0 and decides to remain in green phase 0. Thus, the time interval between t_1 and t_2 is 5 seconds. In action time step t_2 the agent is at the end of green phase 0 and decides to switch to green phase 1. The time interval between t_2 and t_3 is 9 seconds.

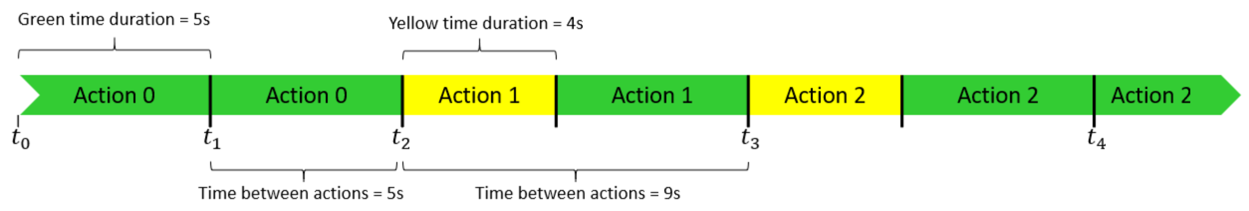


Figure 19 Action time steps and intervals

In some literature, researchers implemented maximum phase durations. This means that after the controller has spent a maximum amount of seconds in one phase, it must switch to another phase. This ensures more fairness between lanes. Yet, other authors argue against imposing these constraints, since they claim that the agent would learn such policies by itself for situations where it is necessary. In our agent, no maximum phase durations will be implemented, since fairness of the algorithm is out of the scope. If the agent turns out to result in unfair policies, these constraints may be added in future research.

6.3.3 Rewards

As discussed in section 5.3.5, many different performance indicators have been proposed as reward functions in literature. Additionally, different researchers have implemented the same performance indicator in slightly different ways. Different reward functions have different advantages and disadvantages. In an attempt to overcome this, several authors attempted to use weighted reward functions which combine several performance indicators. A major shortcoming in literature is that it has not been extensively studied how sub-rewards should be weighted, how different reward functions impact the agent's performance and under which traffic situations different reward functions work well.

Because of this, it is difficult to pick the reward function a-priori. Thus, in this thesis tests will be conducted with several different reward functions as performance metrics. The chosen reward functions consist of the three most commonly chosen singular metrics: cumulative vehicle delay, cumulative vehicle waiting time and queue length. Additionally, a 50/50 weighted reward consisting of both the cumulative delay and waiting time will be tested, since this led to good results in previous studies (Dijk, 2017; Samad, 2020). Each of the four rewards will be tested (1) as an absolute value (equation 17) and (2) as a value of change between two consecutive action time steps t (equation 18):

$$r_t = R_t \tag{17}$$

$$r_t = R_t - R_{t-1} \tag{18}$$

where r_t is the reward at action time step t and R_t is the measure of the chosen metric (i.e. cumulative delay $\sum_i D_{t,i}$, cumulative waiting time $\sum_i W_{t,i}$, queue length Q_t, 50 cumulative delay/50 cumulative waiting time $\sum_i DW_{t,i}$) at action time step t.

Delay per vehicle is defined as the difference between the actual time the vehicle has driven measured at time t and the optimal time the vehicle would have needed to cover the driven distance, assuming it had driven at the speed limit[3]. Since the cumulative delay is used, the delay of all vehicles at an action time step is summed. The equations are as follows:

[3] Driver speed is generated stochastically per vehicle. Some vehicles will be driving above the speed limit, so their actual driving time can be shorter than the optimal driving time. This results in a negative value for the delay.

$$R_t = \sum_i D_{t,i} \tag{19}$$

$$D_{t,i} = T_{t,i}^{actual} - T_{t,i}^{optimal} \tag{20}$$

$$T_{t,i}^{actual} = T_t - T_i^{departure} \tag{21}$$

$$T_t^{optimal} = \frac{d_i}{v_{speed\ limit}} \tag{22}$$

T denotes simulation time steps (one simulation step corresponds to one second) and t denotes action time steps (which happen once every 5 or 9 simulation steps, see section 6.3.2). i denotes a vehicle from the set I of all vehicles in the intersection system at that moment. $D_{t,i}$ is the delay of vehicle i at time t, $T_{t,i}^{actual}$ is the actual driving time of vehicle i at time t, $T_{t,i}^{optimal}$ is the optimal driving time of vehicle i at time t, T_t is the current simulation time at action time step t, $T_i^{departure}$ is the departure time of vehicle i, d_i is the distance vehicle i has driven until now and $v_{speed\ limit}$ is the maximum allowed speed on the leg.

Waiting time per vehicle is defined as the number of seconds that a vehicle has spent at a speed under 0.1m/s. The cumulative waiting time is calculated as:

$$R_t = \sum_i W_{t,i} \tag{23}$$

where $W_{t,i}$ denotes the waiting time of vehicle i at time t.

Queue lengths is calculated as the sum of queuing vehicles on all four legs. Vehicles are considered to be in a queue if their speed is under 0.1m/s. The queue length is defined by the last halting vehicle's position. It is calculated as:

$$R_t = Q_t \tag{24}$$

$$Q_t = Q_t^N + Q_t^W + Q_t^S + Q_t^E \tag{25}$$

where Q_t is the cumulative queue length at time t and Q_t^X denotes the queue length of the respective leg $X = \{N, W, S, E\}$.

Lastly, the **weighted reward consisting of vehicle delay and weighting time** will be calculated as:

$$R_t = 0.5 \sum_i D_{t,i} + 0.5 \sum_i W_{t,i} \tag{26}$$

Since training the agent will be conducted in a simulator, it is possible to gather vehicle-based reward data on all vehicles, including regular unconnected vehicles. This allows us to include the reward for all vehicles, even regular vehicles who are not included in the state. Some authors argue that this is unrealistic since regular vehicles cannot be directly observed. Yet, training will generally always be performed in a simulator, since it is not possible to train an agent on real-life traffic without completely disrupting traffic flows until the agent is trained.

In this research, only a 50/50 weighted reward of delay and waiting time will be tested. Past research has used a wide range of weighted rewards consisting of several sub-rewards. Yet, studies found that it was difficult to determine appropriate weights for them. Since this would likely cost too much time, more complex rewards functions were deemed out of the scope of this thesis.

6.3.4 Agent

For this thesis it was decided to use implement both a deep Q-learning agent (to be referred to as vanilla DQN agent or DQN agent) and a deep recurrent Q-learning agent (to be referred to as recurrent DQN agent or DRQN agent). DQN agents are the most commonly applied type of agent and have been used to successfully control traffic signals. Many reinforcement learning extensions have been proposed (e.g. the rainbow extensions or recurrent DQN agents) and may be applied to gain good results. For this thesis, only recurrence was implemented as an extension, since this extension has been shown to be able to perform well even when the environment state is not fully observable. Due to time restrictions, a comparison with other model extensions could not be implemented.

6.3.4.1 Deep Neural Network Architecture

When building a deep RL model, an integral part is the deep neural network that is used. In literature many different network architectures have been found, ranging from relatively small and shallow (e.g. Genders & Razavi, 2018) to large and deep (e.g. Du et al., 2019). The problem is that there are no clear standards or best practices on how to design neural network architectures for reinforcement learning algorithms yet. Previous studies may be used for inspiration but may not guarantee good results. This is because the best network architecture may depend largely on the specific problem at hand, which means that a network architecture that works for one problem, may not work for a slightly modified problem. An even more complicating factor is that deep reinforcement learning algorithms take long to train and depend on a large range of other hyperparameters as well, so it is not feasible to test a large number of network architectures.

Nevertheless, the network architecture is somewhat restricted by the specific problem at hand. In this thesis DQN and DRQN are being applied, which constrains the allowed network inputs and outputs. For the DQN agent, the neural network input must be a batch of state samples. For the DRQN agent, the input must be a batch of trajectories containing a fixed number of samples. The shape of these samples depends on the chosen state (see section 6.3.1). The output of the DQN agent must be a batch of Q-values for each of the four allowed actions (4 x 1). For the DRQN model the neural network output must be a batch of trajectories of Q-values. Apart from these boundary constraints, the choice of architecture is up to the researcher.

For both the DQN and the DRQN agent, nearly the same network architecture is used. The only difference is that the DRQN has an additional LSTM layer.

For both agents, convolutional layers will be applied to the DTSE-part of the state input. Convolutional neural networks (CNN) are commonly for image processing since it allows to extract spatial information, which would be lost in multilayer perceptrons (MLP). For an introduction to CNNs, please refer to Aghdam and Heravi (2017). Our 2-layered DTSE can be seen as a type of image with a width of 10 cells, a height of 8 cells and 2 channels (position/density and speed). In CNNs, usually ReLu activations are used. When using CNNs, the input is passed through several convolutional layers. To go from one layer to the next, a filter size, kernel and stride must be specified. Generally, the number of filters start small to collect local information and gradually increases to combine local information into complex shapes to distinguish between different features (here: between states) (Ramesh, 2018). The kernel size and the stride depends on the specifics of the image. For the design of the convolutional layers, the architecture from well-performing literature agents was used. It was decided to test two different convolutional architectures. Architecture 1 is a small network inspired by Zeng et al. (2018) and T. Zhao & Wang (2019) consisting of two convolutional layers with 8 and 4 filters, a kernel size of (2,2) and (2,2) and a stride of (2,2) and (1,1) respectively. Architecture 2 is a large network with three convolutional layers of 16, 32 and 64 filters, a kernel size of (2,2), (2,2) and (1,1) and a stride of (2,2), (2,2) and (1,1) respectively, as inspired by (Fang et al., 2019).

The output of the convolutional layers will then be flattened and concatenated with the elapsed time input and the current phase input (if one-hot encoded). In case of the DRQN agent, information will now be passed through an LSTM layer of size 96. Lastly, all information will be passed through two fully connected layers of size 32 and 16 which are ReLu activated. The output layer consists of a fully connected linearly activated layer which outputs the four predicted Q-values.

Figure 20 shows an example of a DRQN network architecture that can be used. Note that the to-be tested networks may differ from the one shown in this figure: some networks will have 3 instead of 2 convolutional layers, vanilla DQN agents will not have the LSTM layer and some architectures will not have the (4 x 1) current phase input. The finally chosen network architecture will depend on the outcome of the preliminary experiments.

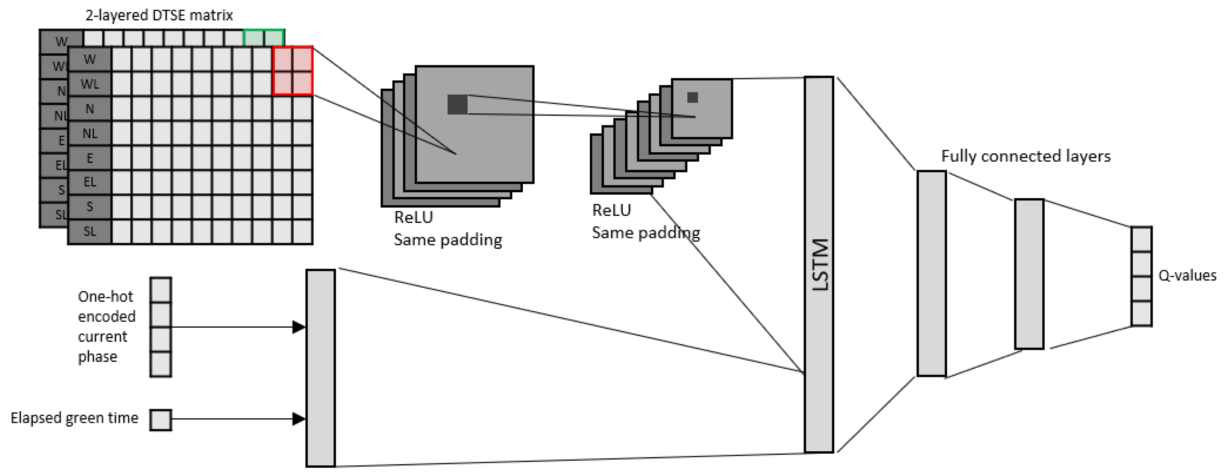


Figure 20 Example of a possible network architecture. Note that the finally chosen network may not be the same.

6.3.4.2 Algorithm stabilization

As discussed in section 4.8, DQN algorithms may not always converge. To stabilize the training process, experience replay and target networks will be applied. The algorithm stability may depend on the hyperparameters which are chosen for the memory size and the target network freeze interval. Since no best practices for choosing these exist yet, tests will be conducted with different ranges of values.

6.3.4.3 Other details

To train neural networks, an optimizer that minimizes a loss function is needed. In this case, the loss function consists of the mean squared error between the model output (here: the predicted Q-values) and the true values (here: the target Q-values). For this model, the Adam optimizer is being used, since this was shown to perform well and converge quickly (Liang et al., 2018).

To trade-off exploration and exploitation, an ε-greedy strategy will be used in which ε linearly decays from 1 to 0 over all training episodes. Each time the agent picks an action, it will pick a random action with probability ε and a non-random action (i.e. the action with the highest Q-value) with a probability $1-\varepsilon$.

6.4 Model evaluation

6.4.1 KPI

During testing, the controller's performance will be evaluated at every simulation step. Since different metrics measure different aspects of traffic controller performance, several KPIs have been chosen:

Average vehicle delay. Delay measures how many seconds a vehicle has lost up until that point compared to the optimal time needed to drive up to that point, assuming the vehicle would have driven at the speed limit. In the simulation, vehicles would only drive slower than their preferred driving speed if they stop for red signals, if they wait behind other vehicles in a queue or if they are accelerating or decelerating. It is then reasonable to assume that delay is a good measure to quantify how much time has been lost due to the traffic signal. It was chosen to evaluate the average rather than the cumulative delay, since this is easier to interpret for humans, and since ultimately the goal is to reduce the effects of signal congestion as much as possible for all vehicles, rather than for just a few vehicles. The formula for delay was presented in equations 17-20 in section 6.3.3. The average delay per simulation step t for all episodes $e \in E$ is calculated as:

$$average\ vehicle\ delay\ at\ time\ t\ = \frac{1}{E}\sum_{e}^{E}\ [\frac{1}{|I|}\sum_{i}^{I} D_{t,i}] \tag{27}$$

Average connected vehicle delay and **average regular vehicle delay.** Additional to the full vehicle delay, the vehicle delay for only connected or regular vehicles will be collected. This allows evaluating whether an agent which can see only connected vehicles also leads to strategies that work for regular vehicles.

Average waiting time. The average waiting time is correlated to the vehicle delay. It measures how many seconds a vehicle spends on average driving under 0.1m/s (see section 6.3.3). It is measured as:

$$average\ vehicle\ waiting\ time\ at\ time\ t\ = \frac{1}{E}\sum_{e}^{E}\left[\frac{1}{|I|}\sum_{i}^{I} W_{t,i}\right] \tag{28}$$

Average queue length. The average queue length measures how many vehicles are waiting on average at time step t. It is measured as:

$$average\ queue\ length\ at\ time\ t\ = \frac{1}{E}\sum_{e}^{E} Q_t \tag{29}$$

6.4.2 Base case

Ideally, the trained agent would be compared against different traditional controllers and state-of-the-art control methods. Due to time limitations, this is not possible in this thesis.

Instead, it was decided to only compare the performance against a fixed-time controller. The fixed-time control has the same green phases as the reinforcement agent. One traffic cycle goes through all four phases in a fixed order (NSG, NSLG, EWG, EWLG). Each phase has a duration of 30s.

The base case will be evaluated in the same manner as the RL agent: it will be tested on the same scenarios and the same KPI will be gathered.

6.5 Summary

In this section the conceptual design of the reinforcement learning strategy has been discussed. Decisions regarding the scope, the environmental setup and the reinforcement learning model have been made. No implementations have been made yet at this point.

Many scoping decisions have been made to simplify the problem. In this project, it will be studied how the RL control strategy affects traffic efficiency in signalized intersections. The focus will be on how different penetration rates of connected vehicles influence the controller's performance.

The environmental setup consists of a geometric 4-way intersection with 4 lanes per leg: the left lane is used for left turns, the two middle lanes for through traffic and the right lane for right turns and through traffic. In total the intersection has 8 traffic signals, or 2 per leg. One signal controls each of the left-turning lanes, and one signal controls each of the three through and right-turning lanes. In total there are four green phases: North/South right/through traffic, North/South left-turning traffic, East/West right/through traffic and East/West left-turning traffic. Between different phases, a yellow phase of 4 seconds is used.

Traffic will be generated in four different types of stochastic scenarios: (1) low constant traffic (2) medium constant traffic (3) high constant traffic (4) dynamic traffic. Generated vehicles in the simulation are either connected vehicles or regular vehicles. The amount of generated connected vehicles is dependent on the penetration rate set for the scenario.

Regarding the reinforcement learning agent, it was decided to implement both a deep Q-learning agent and a deep recurrent Q-learning agent. Both agents will use experience replay and target networks to stabilize the training process. The state representation consists of a 2-layered DTSE of vehicle position/density and speed, the current green phase and the elapsed green time. The exact state representation will be decided on after preliminary experiments have been conducted. The available actions are to decide the next green phase for 5 seconds. The reward representation is currently undecided, but experiments will be conducted on which reward leads to the best performance. Options include the difference in cumulative vehicle delay, waiting time or queue length.

To evaluate the agent's performance, several KPI will be gathered at every simulation step. The chosen KPIs are the averaged (total, connected, and regular) vehicle delay, the averaged waiting time and the averaged queue length. Furthermore, it was decided to evaluate the agent by comparing it against a fixed-time controller.

7. Implementation

In this chapter, the implementation details will be described. This includes the implementation in software, details regarding the traffic generation, an overview of training and testing agents and a discussion on hyper-parameters.

7.1 SUMO (Simulation of Urban Mobility)

In this thesis, SUMO was used as the simulator (SUMO, 2020). SUMO is an open-source microscopic traffic simulator. It is time-discrete and space-continuous (Behrisch, Bieker, Erdmann, & Krajzewicz, 2011).

For this thesis project a simulator had to meet several requirements:

- Be able to do microscopic traffic simulation
- Be able to specify networks of intersections
- Be able to model congestion
- Allow designing a traffic signal control strategy
- Be able to model V2I and V2V communication
- Be able to measure specified performance metrics
- Be able to conduct scenario experiments

Different review articles were available that compare simulator packages. Pell et al. (2017) compared 17 packages with a focus on the realized functionalities. Kotusevski and Hawick (2009) and Ejercito et al. (2017) focused on evaluating the usage and performance of the packages. Several of the packages which were open-source or accessible via the TU Delft met these requirements (SUMO, VISSIM, OpenTrafficSim, Aimsung). SUMO was the finally chosen package since this is by far the most used simulator for RL (see Appendix C). A basic reinforcement learning model that uses SUMO was found from a previous Master thesis (Vidali, 2018). This RL model will be used as the basis for my algorithm.

The implementation of the environment was done using SUMO's NetEdit. All legs and lanes, allowed movement directions, speed restrictions, the intersection itself are specified in NetEdit, along with the position and allowed phases of the traffic lights. For the description of the environment, see section 6.2. This implementation was left the same as in Vidali's (2018) work.

Traffic scenarios can be specified using .rou files. In these files it is possible to specify different vehicle types and routes. More information on traffic generation will be provided in section 7.2.

To interface a traffic control strategy with SUMO, the TraCI (Traffic Control Interface) library is used. This allows us to run the algorithm outside of SUMO and it provides the interface to get and set values in the SUMO simulation.

Lastly, the SUMO GUI can be used. If the GUI is enabled, it is possible to see the model's decisions. This is useful during debugging, e.g. to see how the agent behaves or to visually evaluate the final trained agent. Generally, however the algorithm will be run without the GUI since this is much faster.

7.2 Traffic generation

7.2.1 Traffic scenarios

To train the controller, the agent has to be exposed to episodes of different traffic scenarios. Each episode is 5400 simulation steps long. One simulation step is equal to one second, so episodes are 1,5h long.

As described in section 6.2.3, traffic will be generated with three different amounts of arriving vehicles: low (undersaturated) traffic, medium saturated traffic and high saturated traffic. These arrival rates can be either constant (i.e. uniformly distributed) or dynamic (i.e. Weibull distributed).

To determine the vehicle arrival rates for the three constant traffic scenarios, some simple tests were conducted with a fixed-time controller. The fixed time controller has a green phase duration of 30 seconds and cycles through the four green phases described in section 6.2.2. The controller was tested by using different values for the total number of generated vehicles within an episode. It is assumed that traffic is balanced equally over the legs. For each run, the queue length per simulation step was recorded. Additionally, it was visually checked in the SUMO GUI how many vehicles can pass at once through a green phase. The resulting queue lengths for some of the values which were tested can be seen in Figure 21.

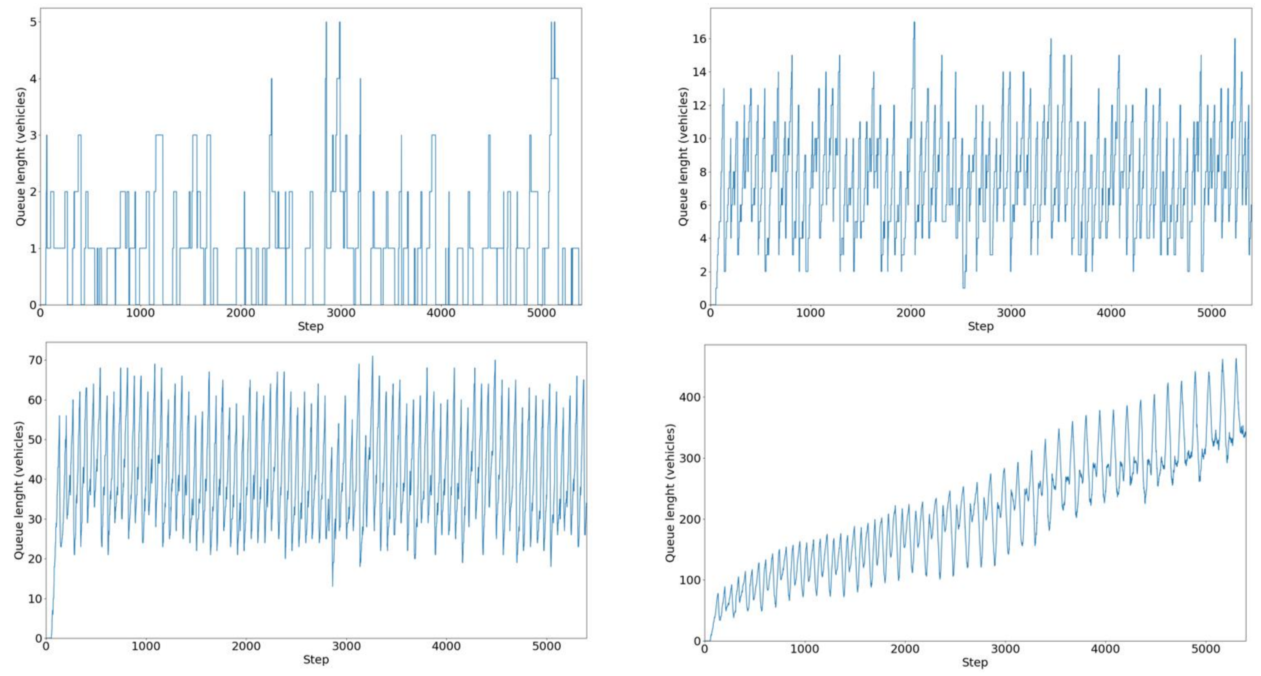


Figure 21 Queue lengths over time in the fixed time controller. Experiments are conducted under constant traffic scenarios with different numbers of total arriving vehicles in an episode. The total number of generated vehicles in the experiments are: 150 vehicles (upper left), 2000 vehicles (upper right), 5000 vehicles (lower left), 7000 vehicles (lower right).

In all figures, the queue lengths are rapidly jumping between higher and lower queue lengths. These jumps can be explained by one queue of vehicles getting a green light and driving through the intersection all at once, which rapidly decreases the queue length. Furthermore, the plots show that as intuitively expected, the more vehicles are generated, the higher the queue lengths. In the plots as well as in the GUI it could be seen that vehicle arrival rates up to around 5000 generated vehicles in an episode of 5400 seconds can be handled by the fixed time controller. When watching the simulation via the GUI, it could be seen that all vehicles in the queue could pass through the intersection at once during a green light phase. The plots also confirm this: the queue lengths follow a regularly repeating pattern, without any up- or downwards trends. The trend in the figure for 7000 generated cars shows a different pattern: the queue lengths show a continuous upwards trends, meaning that queue lengths are increasing over time. The same behavior was easily seen in the GUI: more vehicles were arriving than the fixed-time controller could handle. This resulted in not all vehicles being able to drive through the intersection during a green light phase.

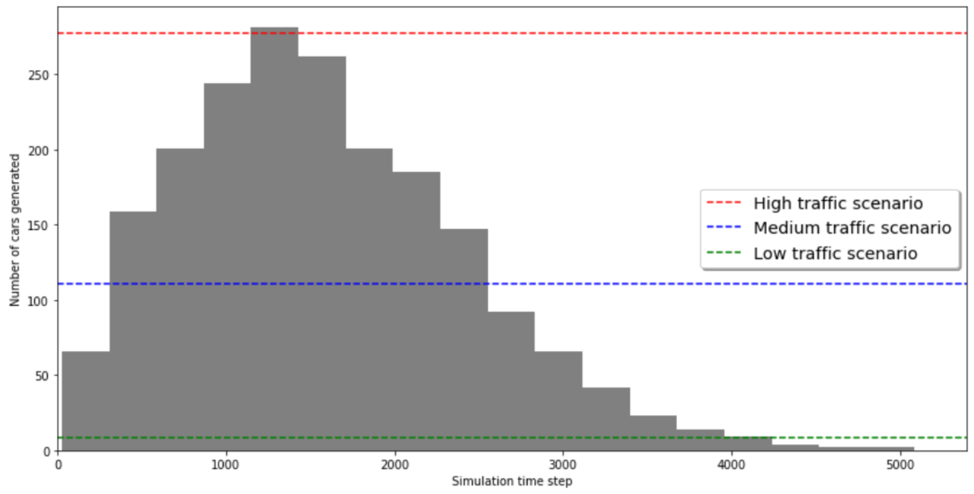


Figure 22 Traffic arrival rates following a Weibull distribution with 2000 generated cars. Overlaid with the constant arrival rates of the high, medium and low traffic scenarios.

From the experiments, it was concluded that the cut-off between saturated and over-saturated traffic is between 5000 and 7000 vehicles. Based on this, for the constant high traffic scenario 5000 cars will be generated, for the medium traffic scenario 2000 cars will be generated and for the low traffic scenario 150 cars will be generated. The low number of 150 generated cars leads to a situation in which oftentimes only a single vehicle is waiting at the intersection (see upper left Figure 21). Overall, the constant traffic scenarios cover the whole range from singe vehicle queues to near-saturated queues.

For the dynamic traffic scenario, the peak of the arrivals should correspond to the high traffic arrival rate. This is the case for a Weibull distribution with approximately 2000 generated cars. A histogram of this Weibull distribution over one full episode is shown in Figure 22. A bin size of 5 minutes was chosen, resulting in 18 bins. Additionally, the constant arrival rates of the high, medium and low traffic scenarios per 5 minutes are plotted (277.78 veh/5 mins, 111.11 veh/5 mins, 8.33 veh/5 mins resp.). As can be seen, the Weibull distribution arrival rates cover all three constant traffic scenarios. Because of this, only one type of dynamic traffic scenario is needed. A summary of the four different scenarios is shown in Table 4.

Table 4 Scenario specifications

Scenario	**Distribution**	**Number of generated vehicles**
Low constant traffic scenario	Uniform	150
Medium constant traffic scenario	Uniform	2000
High constant traffic scenario	Uniform	5000
Dynamic traffic scenario	Weibull	2000

7.2.2 Vehicle generation

Once it has been decided which scenario will be simulated in the episode, the algorithm starts with determining the arrival patterns for the full episode. The arrival rates are generated according to Table 4.

Additional to the arrival rates, each vehicle will be randomly assigned a certain origin and destination. Since it was decided to use balanced traffic, each generated vehicle has equal chances to start on either of the four legs (25% chance for each leg). Regarding the turning movements, each vehicle has a 25% chance to turn right, a 25% chance to turn left, and a 50% chance to continue straight.

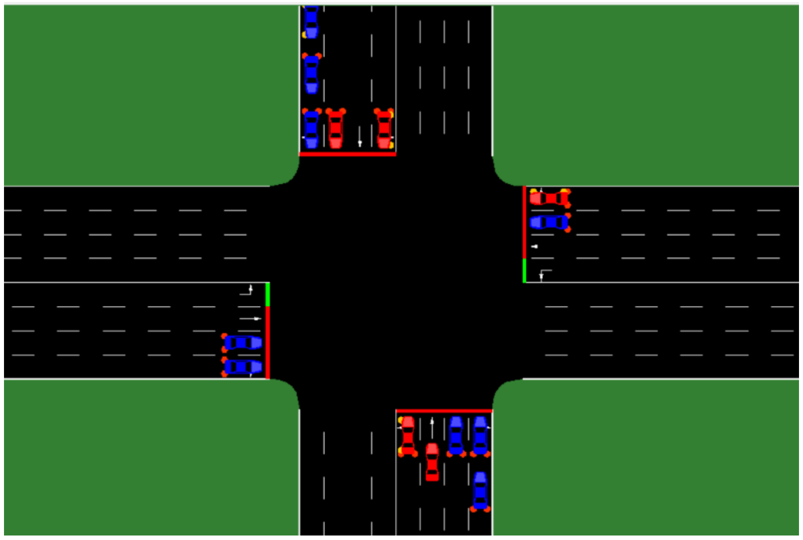


Figure 23 Screenshot from the SUMO GUI. Red vehicles are regular vehicles, blue vehicles are connected vehicles.

Since in this thesis we are interested in how different penetration rates of connected vehicles affect the controller's performance, vehicles must be either a connected vehicle or regular vehicle. This assignment is done randomly. Vehicles are assigned to be a connected vehicle with the probability of the penetration rate, else they will be a regular vehicle. Within this thesis, the agent will be trained and tested under CV-penetration rates of 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% and 100%. To visualize the vehicle type in the GUI, connected vehicles are displayed in blue, while regular vehicles are displayed in red (see Figure 23). Within the simulation, all vehicles have the same characteristics (see Table 5).

Table 5 Vehicle parameters in SUMO

Parameter	**Value**
Acceleration	1 m/s2
Deceleration	4.5 m/s2
Vehicle length	5 m
Car following model	Krauß following model (Krauß, 1998)
Minimum gap between two vehicles	2.5 m
Maximum Speed	25 m/s
Departure Speed	13.89 m/s
Departure Lane	Random
Driver imperfection[4]	0.5

[4] The driver imperfection is a parameter that causes differences between vehicles in their driving speeds and the acceleration behavior. It must be a value between 0 and 1. A value of 0 denotes that all driver's follow the set parameters perfectly. Adding driver imperfection makes simulations more realistic, since not all drivers drive exactly the same.

7.3 Training

7.3.1 Vanilla DQN Agent

Algorithm 2 describes the training process of the vanilla DQN agent. An agent is trained for a fixed amount of episodes using a fixed penetration rate.

Algorithm 2: Training process of the DQN agent

Input hyperparameters: total number of episodes *total_episodes*, maximum number of steps per episode *max_steps*, green phase duration *green_duration*, yellow phase duration *yellow_duration*, target network freeze interval β, batch size *batch_size*, learning rate α, number of epochs *training_epochs*, minimum memory size *memory_size_min*, maximum memory size *memory_size_max*, discount rate γ

```
1  Initialize online with random weights $\theta$ and target network with identical weights $\theta^{-} = \theta$
2  Initialize empty experience replay memory
3  for total_episodes episodes:
4      Generate next traffic scenario
5      # RUNNING THE SIMULATION AND GATHERING SAMPLES
6      while current simulation step < max_steps:
7          Get the current state $s$ at this time step $t$
8          Observe the reward $r$ at this time step $t$
9          Add observed sample to memory in the form $(s_{t-1}, a, r, s_t)$
10         if current memory size > memory_size_max:
11             Remove the oldest sample from the memory
12         Pick next action $a$ based on the current state using the online network:
           $a = \underset{a}{\operatorname{argmax}} Q(s, a, \theta)$ with probability $1 - \varepsilon$, else select a random action
13         if action $a_t$ == old action $a_{t-1}$:
14             Execute a yellow phase for yellow_phase seconds
15         Execute a green phase for green_phase seconds
16     # TRAINING THE NETWORK
17     for training_epochs epochs:
18         if current memory size is bigger than memory_size_min:
19             Sample a uniformly random batch with batch_size number of samples
               from the memory
20             for each sample $(s_{t-1}, a, r, s_t)$ in batch $b$:
21                 calculate the target Q-values with
                   $Q(s, a, \theta) = r + \gamma * \underset{a}{\operatorname{argmax}} Q(s, a, \theta^{-})$
22             Train online model (i.e. update $\theta$) using batch $b$ by fitting the batch state
               as network inputs and the new batch target Q-values $Q(s, a, \theta)$ as
               network outputs
23     if current training iteration % $\beta$:
24         Copy model weights $\theta$ from online network into target network (i.e. replace
           parameters $\theta^{-}$)
25 Save trained online network weights $\theta$ and plot cumulative reward for every episode
```

Every episode contains the same processes. Note that the number of episodes must be specified a priori. Each episode runs for 5400 simulation steps, but the number of action steps within an episode may vary, since the time interval between consecutive green phases varies (5s or 9s). The number of action steps ranges between 600 action steps (if always the same action is chosen) and 1080 action steps (if always a different action is chosen). On average, an episode will include 675 action steps. An episode can be roughly split into three phases: traffic generation, running the simulation and gathering samples, and training. Figure 24 shows the different processes and decisions within one episode for the vanilla DQN model.

Each episode begins with generating a new training scenario (see section 7.2.1). Scenarios are generated in the following cycle: dynamic traffic, low traffic, dynamic traffic, medium traffic, dynamic traffic, high traffic. This means that half the trained scenarios are constant traffic situations and the other half are dynamic traffic situations. Once the scenario for the current episode is picked, the algorithm randomly generates a SUMO route file with the corresponding number of generated vehicles and penetration rate.

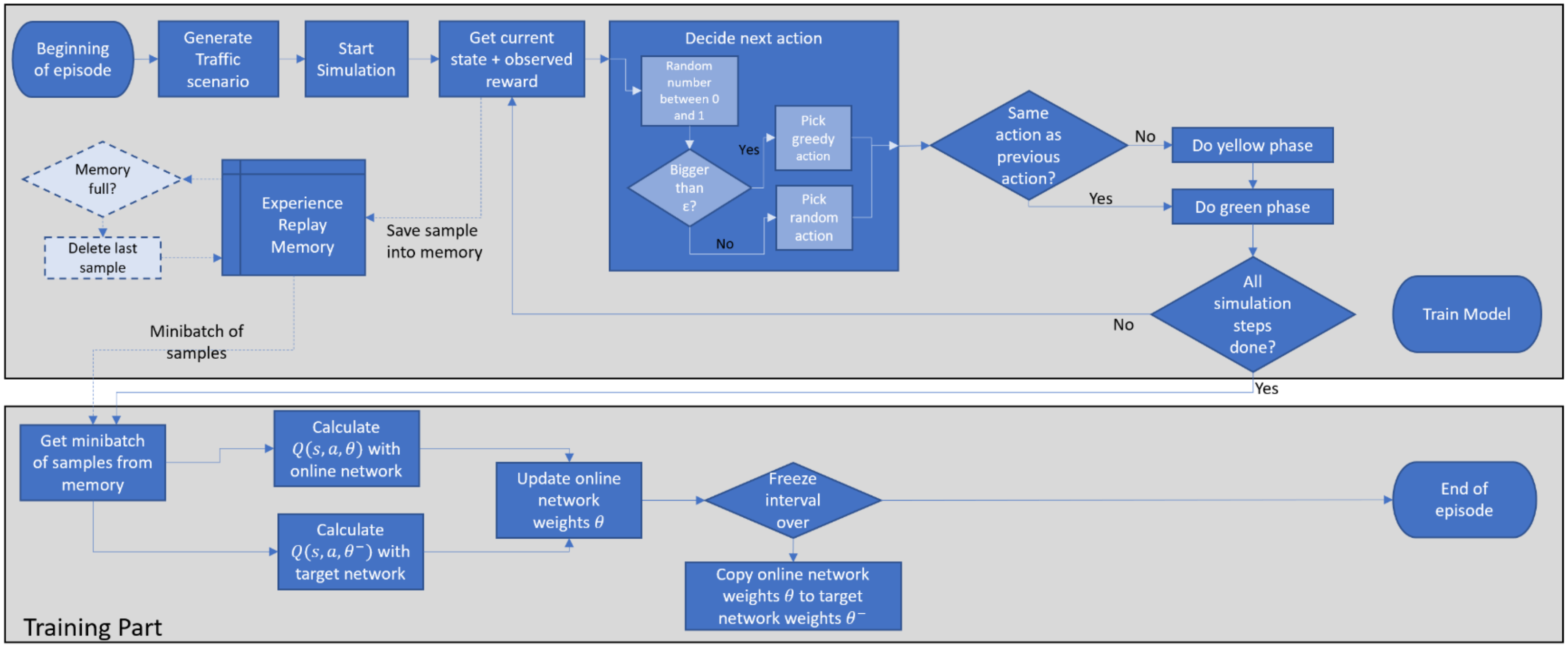
The next step is to run the simulation. The simulation is run for 5400 simulation steps. During the simulation, the agent gathers experiences (i.e. samples in the form (s_{t-1}, a, r, s_t)) and saves these samples into the experience replay memory. If the memory gets fuller than a certain threshold, the oldest sample gets deleted. During the simulation, the agent acts as follows: at the end of the green phase duration, the algorithm gets the current state and the observed reward from the simulator. Based on the current state, it decides which action to pick, i.e. it picks the green phase for the next 5 seconds. The agent picks a random action with a probability ε and a greedy action with a probability of $1 - \varepsilon$ (ε-greedy strategy). To do a greedy action, the agent inputs the current state into the online network and picks the action which returns the highest Q-value (i.e. the highest valued output). If the action is the same as the action in the previous time step, the agent remains in the same green phase. If the action is different, the agent switches to the corresponding yellow signal phase for 4 seconds and then switches to the new green phase. After the simulation has run for 5 seconds in the new green phase, the steps described are repeated until the episode is over.

After the simulation part is finished, the agent will be trained. Training follows the process described in chapter 4. The agent is trained on a fixed number of batches, one batch at a time. Batches consist of 100 randomly chosen samples from the memory. For each entry in a batch, the current Q-value $Q(s, a, \theta)$ is calculated using the online network and the target Q-value $Q(s, a, \theta^-)$ is calculated using the target network. The online network is then updated using equation 15 from section 4.7: $Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) - \alpha\big[Q(s_t, a_t) - \big(r_t + \gamma V(s_{t+1})\big)\big]$. After a fixed number of steps, the target network is unfrozen, and the weights from the online network are copied into the target network.

7.3.2 Recurrent DQN agent

For the DRQN agent, the flow of processes during one episode is mostly the same as for the vanilla DQN agent. This section will only describe the changes compared to the vanilla agent.

- **Saving to memory**. In the vanilla DQN model, the memory consists of individual experience samples for an action time step t in the form (s_{t-1}, a, r, s_t). The recurrent agent however learns from trajectories of experiences (i.e. several consecutive samples from time steps t_{start} until t_{end}). If samples are saved as individual experiences, information about how the time dimension would be lost. To solve this, the memory instead stores full episodes of samples during which the order of the experience samples is preserved.


Beginning of episode
Generate Traffic scenario
Start Simulation
Get current state + observed reward
Decide next action
Random number between 0 and 1
Bigger than ε?
Yes
No
Pick greedy action
Pick random action
Same action as previous action?
No
Yes
Do yellow phase
Do green phase
All simulation steps done?
No
Yes
Train Model
Memory full?
Delete last sample
Experience Replay Memory
Save sample into memory
Minibatch of samples
Get minibatch of samples from memory
Calculate $Q(s,a,\theta)$ with online network
Calculate $Q(s,a,\theta^-)$ with target network
Update online network weights θ
Freeze interval over
Copy online network weights θ to target network weights θ^-
End of episode
Training Part
Figure 24 Flow of one episode. An episode has three parts: traffic generation, simulation and training.

- **Get a minibatch of samples.** In the vanilla DQN model, a minibatch consists of a batch of individual samples. In the recurrent DQN model, a minibatch of consists of a batch of sample *trajectories.* Each trajectory consists of a fixed number of consecutive samples and is randomly chosen from memory. Thus, time relations are preserved within sample trajectories, while there is no time correlation between trajectories.
- **Choose next action.** When choosing greedy actions, the action with the highest corresponding Q-value is chosen. In the DQN model, these Q-values depend only on the current state. In the recurrent model however, the network includes an LSTM layer that can remember past states (called the hidden state). Q-values then not only depend on the current state, but also on the hidden state.
- **Training using trajectories.** During training, the algorithm tries to minimize the difference between trajectories of predicted Q-values and trajectories of target Q-values. Generally, there are two ways to update the DRQN's weights: bootstrapped sequential updates and bootstrapped random updates (Hausknecht & Stone, 2015). Both have been shown to lead to similar performance. In this thesis bootstrapped random updates are performed. This means that sample trajectories of a fixed length with a random starting point within an episode are selected randomly from memory. Chosen trajectories are used for one weight update step. The hidden state of the LSTM will be reset to zero for every trajectory. In Hausknecht and Stone (2015), the full trajectory is used for training. Several improvements to this implementation have been made in research which have been shown to lead to better results (Kapturowski et al., 2019; Lample & Chaplot, 2017). Nevertheless, for this thesis due to time limitations it was chosen to only implement the less complex solution of Hausknecht and Stone (2015). Extensions could be implemented in future research.

7.4 Testing

After a model has been trained, it can be tested. During testing, the model will no longer do any exploration (i.e. pick any random actions), but instead will only do exploitation (i.e. choose greedy actions). Assuming that the model has been trained sufficiently, this should lead to a good traffic signal control strategy.

During testing, the model will be tested on each scenario separately. This allows to evaluate how well the controller has learned to control a certain scenario, and whether it can recognize and switch between different scenarios. For each trained agent, the KPI described in section 6.4.1 will be gathered at every simulation step.

Since traffic scenarios are stochastically generated, it is not possible to trust a single scenario run, since results may vary a lot for different scenarios even if they are generated under the same random distribution. To solve this, a trained model must be tested under one scenario for enough runs to get a good estimate of all outcomes. Both the mean and the range between the 5% and 95% percentile will be reported. In this project, each model will be evaluated using 50 runs. The agent will only be tested on episodes that have not yet been seen during training (i.e. traffic scenarios during testing are generated with a different random seed than during training).

7.5 Model hyper-parameters

Reinforcement learning models have many hyper-parameters. Hyperparameters are parameters which are not inferred during training, but which are set by the modeler. They influence the training process of the agent, as well as the model's final performance. As such, it is important to pick suitable values. The problem is that many of these parameters have not been investigated extensively, so it is unclear which values would be optimal for the model.

In this thesis, some of the hyperparameters are chosen based on previous investigations in literature. Yet, for other parameters no clear results were found, so systematic preliminary tests will be done in which the final model will be fine-tuned. This section will discuss how the hyper-parameters influence the training process and model performance, and which values will be chosen.

Training episodes. The number of training episodes influences how many new traffic scenarios an agent will experience. The more scenarios it sees, the more data it has to train on. If too few scenarios are used, the agent may overfit these scenarios, leading to a bad testing performance on new scenarios. Yet, more scenarios lead to an increase in training time.

The number of training episodes which is needed is difficult to determine. It not only depends on the model being trained but also on the number of training epochs and the mini-batch size. Due to this, no value was chosen yet. Instead, tests will be conducted with several different values for the number of episodes (see sections 8.1 and 8.2.5).

Training epochs. The number of training epochs affects how many mini-batches are being used to train the agent after every episode. If too few batches are picked, then underfitting may occur. If too many batches are picked, overfitting may occur. In either case, the test performance will be unsatisfactory.

Similar to the number of training episodes, it is difficult to generalize how many training epochs are needed. Because of this, tests will be run with different numbers of epochs (see sections 8.1 and 8.2.5).

Mini-batch size. The mini-batch size parameter defines how many samples are included in one training batch. Small batch sizes may lead to a fast converging learning process but could lead to training noise. Large batch sizes lead to slower convergence with less noise.

In literature, different batch sizes have been used, ranging usually from 32 to 128. Vidali (2018) tested different training strategies and found that frequent training using a batch size of 100 outperformed a strategy with a batch size of 50. Due to this, it was decided to use a mini-batch size of 100.

Learning rate. Deep neural network weights are updated using stochastic gradient descent methods. The gradient descent algorithm estimates an error gradient based on the training data and updates the model weights by back-propagating the errors through the network. The learning rate determines the step size of these updates, or in other words, how fast an algorithm "overwrites" old information with new information when the weights are updated. Higher learning rates lead to faster learning but may lead to overshooting the optimum and unstable training. Lower learning rates lead to slower training and could get stuck in local optima.

Pol (2016) did experiments using high and low learning rates. She found that higher learning rates ($\alpha = 0.1$) lead to earlier oscillation, but lower learning rates ($\alpha = 0.00025$) also lead to oscillation. She hypothesized that due to the ADAM optimizer's adaptive gradients, the learning rate does not influence the final model's stability to a large extent. Based on these results, for this thesis it was chosen to use a relatively low learning rate of 0.001.

Discount factor. The discount factor is a parameter that determines how important future rewards are compared to the current reward. Low discount rates lead to prioritizing current events or events in the close future over distant events. Short-term optimal decisions may be taken at the expense of future events. High discount factors lead to future rewards being nearly as important as current rewards.

In traffic signal control problems, future rewards should be nearly as important as immediate rewards, because short-term optimal actions could have negative effects in the long term on queue lengths and waiting times. As such, in literature usually high gamma values are used. A commonly chosen discount factor is 0.99. This value will also be used here.

Target network freeze interval. The freezing interval determines how often the target network weights are copied over into the online network. Shorter freeze intervals lead to more frequent target network updates. Too short intervals can lead to instability of the training process since the target Q-values are still shifting too frequently. Too long intervals can lead to a need for long training, too strong gradient updates, or getting stuck moving in a wrong direction (Pol, 2016).

Pol (2016) tested freezing interval of 10,000 (1 episode), 30,000 (3 episodes) and 50,000 steps (5 episodes). She found that both a freeze interval of 1 and 5 episodes leads to unstable results, but a freeze interval of 3 episodes gained more stable performance. However, she trained her agent only on one type of scenario rather than four different ones, so her results may not transfer to this thesis' agent. Thus, tests with different freeze intervals will be conducted (sections 8.1 and 8.2.6).

Memory size. The memory size determines how many samples can be stored in memory, and thus how long a sample will be stored until it is deleted. Small memory sizes only let the agent see recent episode samples and could lead to catastrophic forgetting and training instability. Large memory sizes allow the agent to see experiences from more episodes, but this poses the risk that some old samples are outdated and irrelevant. Additionally, computational memory issues could occur.

Pol (2016) also investigated the effect of memory size on training stability. Her results indicate that a memory size of one episode is too short and leads to unstable results. Yet, her results for larger memory sizes (10 episodes in her case), also lead to somewhat unstable results. As such, the optimal memory size remains unclear and will be tested (see sections 8.1 and 8.2.7).

Trajectory sequence length (for the DRQN model only). The length of a sampled trajectory provides the controller with several consecutive samples of experiences to build up the hidden state. If a too short trajectory is chosen, the model has too little knowledge about past states to build up an accurate hidden state since the hidden state is reset after every trajectory. This can make it harder or impossible for the model to learn good policies. If a too long trajectory is chosen, the training time increases (Dijk, 2017). Furthermore, the i.i.d. assumption is violated more (see section 4.8.1), which can lead to instability.

Dijk (2017) experimented with different history sizes by comparing model performance and stability for models trained with trajectory sequence lengths of 2, 4, 10 and 20. The results show that a trajectory length of 2 leads to unstable results which are worse than that of a vanilla DQN model. A trajectory length of 4 leads to equally good stability and performance as the vanilla DQN model. Both the trajectory lengths of 10 and 20 perform equally well and both outperform the vanilla DQN model in stability and performance. Since the training time between the 10 and 20 trajectory length agents is nearly doubled, Dijk (2017) advises using the agent with a trajectory length of 10. However, since the agent which is trained in this thesis is not equal to the agent of Dijk (2017), it is unclear whether these results transfer. Due to this, experiments with different trajectory lengths will be done (see sections 8.1 and 8.2.8).

7.6 Implementation, Verification and Validation

7.6.1 Software Implementation and Verification

This DQN agent was implemented in python 3 using the deep learning library Keras and the machine learning library TensorFlow 2. The simulation part uses the SUMO simulator. The interface between the two parts is handled by the SUMO library Traci.

During implementation, each time a new functionality was added small tests were run to verify it. Often small training runs were done with few episodes and epochs to see if the model breaks. Plots of the gathered KPI and the deep NN architecture were used to check if parameters are implemented correctly.

For a lot of debugging it was useful to check the in- and output array shapes. For example, if the state consists of a 2-layered DTSE, a one-hot encoded phase vector and the elapsed time, the deep NN's input shape should be of the dimensions ((10, 8, 2), (4, 1), (1)). The output of the deep NN should be the four Q-values and should thus be of shape (4,1).

Apart from this, it was checked whether the algorithm behaves as it is expected. For example, when implementing the state representation, the python code's state representation was manually compared against the current state visible in the SUMO GUI. For the memory, it was for example tested whether it stores newly added samples correctly and whether it removes the oldest samples once the memory gets too full. And for the target network it was checked whether the online model's weights continuously get updated, while the target model's weights only get updated at the end of a freeze interval. If inconsistencies were found, these errors were corrected before implementing new functionalities.

Verifying the correct functioning of the network models itself is more complicated. Deep neural networks are useful to approximate complex non-linear functions, but humans are not able to easily interpret their weights. However, since the neural network and machine learning libraries Keras and TensorFlow are being used, it can be assumed that the neural network is being built as specified and that model fitting and predictions work correctly.

7.6.2 Validation

Since DQN models are model-free, it is difficult to evaluate what is happening inside the algorithm and to understand why an agent makes a certain decision. This makes it difficult to understand what effect changing certain parameters has on the model.

Additionally, it is not possible to test a traffic controller which is under development on real-world traffic due to time, cost, safety restrictions and possible public inconvenience caused by sub-optimal strategies. Because of this, the controller has to be tested in a simulator, so in this case in SUMO. However, to get valid results, the simulator must be able to accurately simulate real-world traffic and driving behaviors. Since SUMO is developed by researchers and is widely applied by researchers as well, it can be assumed that the tool has been sufficiently validated and that it realistically represents traffic flows.

7.7 Summary

This chapter described the implementation and experimental setup. For this project, both a DQN agent and a DRQN agent will be trained and tested. To stabilize training, experience replay and target networks are used. The reinforcement learning part of the algorithm is implemented in python using Keras and TensorFlow, while the simulation part is implemented in SUMO.

To evaluate the agent's training stability and performance, the cumulative reward per training episode is plotted and analyzed. After training, an agent can be tested for its ability to reduce congestion using the gathered KPI.

Since many model settings and hyper-parameters cannot be determined a priori, preliminary experiments will be conducted to fine-tune the settings. Only the best performing model will be used for the penetration rate experiments. In these experiments, an agent will be trained under CV-penetration rates between 10% and 100%. The performance of the different agents can be compared to each other. Additionally, the agents will be compared to a fixed-time controller as a base case.

8. Experiments

This section will present and analyze the experimental setup and the results. First, the experimental strategy is described. Then, the preliminary experiments will be presented during which several parameters were tested. Based on this, a fine-tuned vanilla and a fine-tuned recurrent agent have been built. This agent has been trained and tested under different penetration rates. These tests will also be analyzed in this chapter.

8.1 Methodology and expected results

8.1.1 Preliminary experiments methodology

As mentioned in sections 6.3.1 on states, 6.3.3 on rewards, 6.3.4 on the reinforcement agent and 7.5 on model hyperparameters, not all model settings and parameters could be determined a priori, since it is often unclear which settings would perform better. Due to this, several preliminary experiments will be conducted which are used to build a fine-tuned controller.

In preliminary experiments, a base model will be used with the currently best-performing settings. Within the base model, a single setting or parameter will be changed at a time (using 2 or 3 different values) to find the best performing one. Each model must be trained and then evaluated. Using the experiments, the agent's performance will hopefully iteratively improve until a good agent is found. The following preliminary experiments will be conducted:

- **State representation** – floating-point density vs binary vehicle position. See section 6.3.1.
- **State representation** – one-hot encoded phase vs +/- encoded phase. See section 6.3.1.
- **Reward representation** – cumulative delay vs cumulative waiting time vs queue length. See section 6.3.3.
- **Convolutional network architecture** – small or large network. See section 6.3.4.1.
- **Number of episodes –** 500, 600, 700 episodes.
- **Number of epochs –** 300, 400, 500 epochs.
- **Combinations of episodes and epochs –** more episodes/fewer epochs vs fewer episodes/more epochs
- **Freeze interval –** 4000, 6000, 8000, 10000, 12000, 14000 steps (4, 6, 8, 10, 12, 14 episodes resp.)
- **Memory size –** 30,000, 40,000, 50,000, 60,000, 70,000, 80,000 memory size
- **Trajectory length (for DRQN only) –** 9, 12, 15, 18, 21, 24 samples per trajectory

The evaluation takes place by investigating how the cumulative reward or the cumulative KPI per episode changes during the training process. For the first experiments, only the cumulative reward per episode was tracked. Since different scenarios lead to different magnitudes of rewards, the rewards must be split into the different scenario types and analyzed separately (see e.g. in Figure 25).

Investigating the cumulative reward plots allows us to evaluate the agents' performance and training stability. In the beginning, the agent only chooses random actions, so the performance may be unstable. With increasing ε-values, the fraction of random actions decreases, while the fraction of greedy actions increases. This means that in later episodes increasingly more actions are being chosen based on the learned policies. For a well-performing agent, the graphs are expected to show an upwards trend towards a less negative reward. This would imply that a better policy than a random policy was learned. A well-performing agent should also not show unstable behavior towards the end of the plot. Instability towards

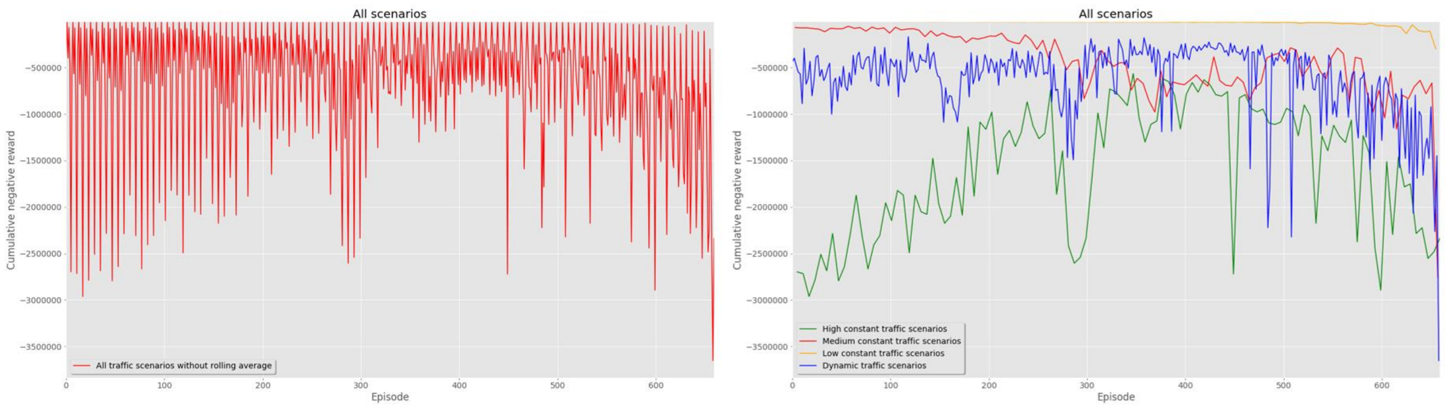


Figure 25 Example plots of the cumulative reward per episode over the full training period. The left figure plots the absolute reward values per episode. It can be seen that the reward jumps up and down a lot, due to the different number of vehicles in the episodes. The right figure plots the absolute cumulative reward per scenario type. Developments per scenario can be evaluated.

the end can point to underfitting (e.g. due to not having been trained enough) or overfitting (e.g. getting stuck in local optima), which may lead to unsatisfactory results.

The problem when only looking at the gained reward during training episodes is that a certain fraction of actions (ε) is chosen randomly rather than greedily. Especially in the first half of the episodes, a majority of the actions is chosen randomly. This makes it more difficult to analyze how a model converges and how stable it is, since both factors may be negatively influenced by the random actions. To counter this problem, in later experiments testing episodes were performed after every 10 training episodes. During these testing episodes, each of the 4 scenarios is run using only greedy actions. Greedy actions are chosen based on the most recent version of the neural network which is being trained. When looking only at the greedy testing episodes, it is easier to analyze the algorithm's stability and performance over time. The drawback is that 4 additional testing episodes have to be run after every 10 training episodes, which will significantly increase the training time. Due to this, greedy episodes have only been run for part of the experiments.

For each of the tested (hyper-)parameters, the best-performing model will be selected. Once all preliminary experiments have been evaluated, a fine-tuned model can be built by combining all the best-performing settings. Note that individual settings may be near-optimal for the agent used during the preliminary experiment, yet there is no guarantee that settings are still near-optimal after being combined into a fine-tuned agent. This issue is left for future research.

8.1.2 Penetration rate methodology

The fine-tuned agents from the preliminary experiments will be used for the penetration rate experiments. Both a DQN and a DRQN agent will be built. The agents will be trained under CV-penetration rates of 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% and 100%. Again, the cumulative reward plots from training can be used to evaluate the agents' training stability.

To evaluate the actual performance, the agents will be tested according to the procedure described in section 7.4. Based on the gathered KPI, the performance of the models can be compared for the different model types and penetration rates.

8.1.3 Hypotheses for expected results

For our experiments, we expect the following results:

(1) Increases in penetration rates lead to increases in agent stability. Since agents trained under lower CV-penetration rates can observe less of the state than agents trained under higher penetration rates, likely these agents will not be able to map states well to Q-values. Especially agents trained under low penetration rates can only see a very limited amount of cars. The agent may be able to learn good policies in some of the scenario runs, but likely not as well as the higher penetration rate agents. Thus, more divergence for the KPIs in these agents is expected.

(2) Increases in penetration rates lead to increases in agent performance. For the same reason, it is expected that on average the agents trained under higher penetration rates will achieve lower delays, waiting times and queue lengths.

(3) Connected vehicles have lower delays and waiting times than regular vehicles. When agents observe the state, they can only see connected vehicles. Because of this, it is expected that the agents will create strategies targeted at connected vehicles. Since agents cannot see regular vehicles, it is more difficult for agents to develop good strategies for them.

(4) Increases in penetration rates lead to decreasing differences in delays and waiting times between CVs and RVs. Higher CV-penetration rates mean that more CVs are seen by the agent. In low penetration rates, the agent can react to individual CVs. In higher penetration rates however, the agent will see more CVs at the same time and must decide which CVs to prioritize. Ultimately, this means that increasing CV-rates lead to higher average delays and waiting times for CVs.

(5) Increases in the number of cars in a traffic scenario lead to decreasing differences in delays and waiting times between CVs and RVs. In the low traffic scenario, very few vehicles are generated, so vehicles generally arrive one at a time. Since the agent can only observe connected vehicles, it would be expected that the agent will directly switch to a green phase for an arriving connected vehicle, thus leading to low delays. Regular vehicles however would not be observed, so they would have to wait until a connected vehicle joins them to be recognized. In high traffic scenarios on the other hand, many vehicles arrive at the same time. The agent will no longer be able to distinguish or react to individual vehicles but will respond to groups of vehicles as a whole. This makes it more likely that CVs and RVs arrive around the same time and wait in the same queue. The agent can now react to the CVs in the queue, and the RVs in the queue can pass at the same time. Ultimately, regular vehicles will have similar delays and waiting times as connected vehicles in the high traffic scenario.

(6) Good performance in one scenario is correlated with good performance in another scenario. The goal of controllers is to be able to reduce congestion in different traffic scenarios. Due to how deep Q-learning functions, the agent tries to optimize policies for all scenarios simultaneously. This makes it unlikely that the agent will learn good policies in one scenario, while sacrificing performance in the other scenarios.

(7) Reinforcement learning agents can outperform fixed-time controllers.

(8) Recurrent DQN agents are more stable than vanilla DQN agents. Since DRQN agents not only see the current state, but also have access to past states, it is expected that these agents will be more stable than DQN agents.

(9) Recurrent DQN agents perform better than vanilla DQN agents. For the same reason, it is expected that recurrent agents will lead to lower delays, waiting times and queue lengths than vanilla agents.

8.2 Preliminary parameter test results

This section describes the results of the preliminary experiments per experiment. Finally, combining all the preliminary experimental results, a fine-tuned agent can be built.

8.2.1 State representation - floating-point density vs binary vehicle position

As described in section 6.3.1, the vehicle position inside each cell in the first layer of the DTSE can either be expressed binary (i.e. 1 if a vehicle is present in the cell, 0 if no vehicle is present in the cell) or as a floating-point number in terms of vehicle density (i.e. number of vehicles in cell divided by potential maximum number of vehicles in cell). The binary representation means that there are less possible states, while the floating-point representation means that there is less information loss when translating the real

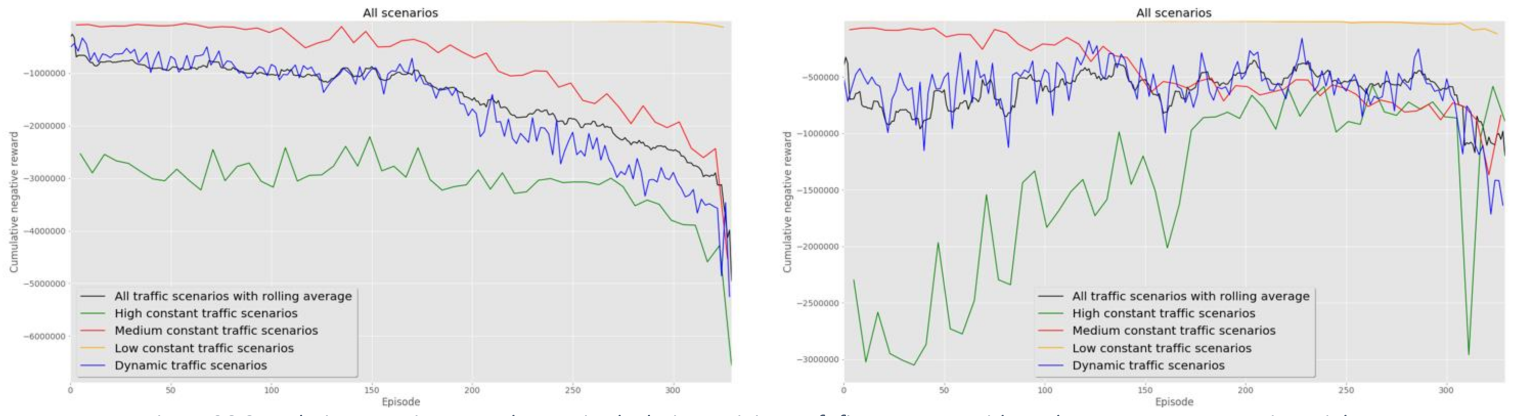


Figure 26 Cumulative negative reward per episode during training. Left figure: agent with Boolean state representation. Right figure: agent with floating point density state representation.

world into a DTSE. It is unclear which representation would be more stable and which would perform better. To investigate which representation to pick, both representations were tested on the base agent presented in Table 7 (see Appendix A.1).

The results are shown in Figure 26. The left figure shows the training reward for the Boolean state representation agent. It can be seen that in all four scenarios the performance towards the end of training (i.e. when mostly greedy actions are chosen) is worse than at the beginning of training (when mostly random actions are chosen). This means that this agent's learned strategy is worse than a random strategy. The right figure shows the training reward for the agent with a density representation. The final performance of the agent is still worse than picking random actions, except for the high traffic scenario. In the high traffic scenario, the learning curve went upwards, thus showing that the agent is learning better-than-random policies. However, the cumulative negative reward for the other three scenarios is less than half the reward for the same scenarios in the Boolean representation agent, meaning while the policy is still bad, it is much better than the Boolean agent.

From the experiments, it can be concluded that the Boolean representation is unsuitable. Likely too much information is lost when translating the real world into binary cell occupancy, making the agent unable to distinguish different traffic situations from each other. Thus, for all succeeding agents, only the density representation will be used.

Nevertheless, the density agent's performance is unsatisfactory in performance, as well as unstable (which can be seen in the jumps in the learning curve). Possible explanations for this are underfitting (e.g.

due to too low number of training episodes or epochs), an unsuitable reward function or other untuned parameters.

8.2.2 State representation - one-hot encoded phase vs +/- encoded phase

Section 6.3.1. discussed two possible ways to encode the current green phase in the model. The first option is to one-hot encode the green phase and use it as an additional (4x1) neural network input. The second option is to encode the current phase in the DTSE: cells in lanes that currently have a green phase will have a positive-valued cell entry, while cells in lanes that have a red phase will have a negative-valued cell entry. To investigate which option leads to better results, the agent was tested on the base agents presented in Table 8 and Table 9 (see Appendix A.2).

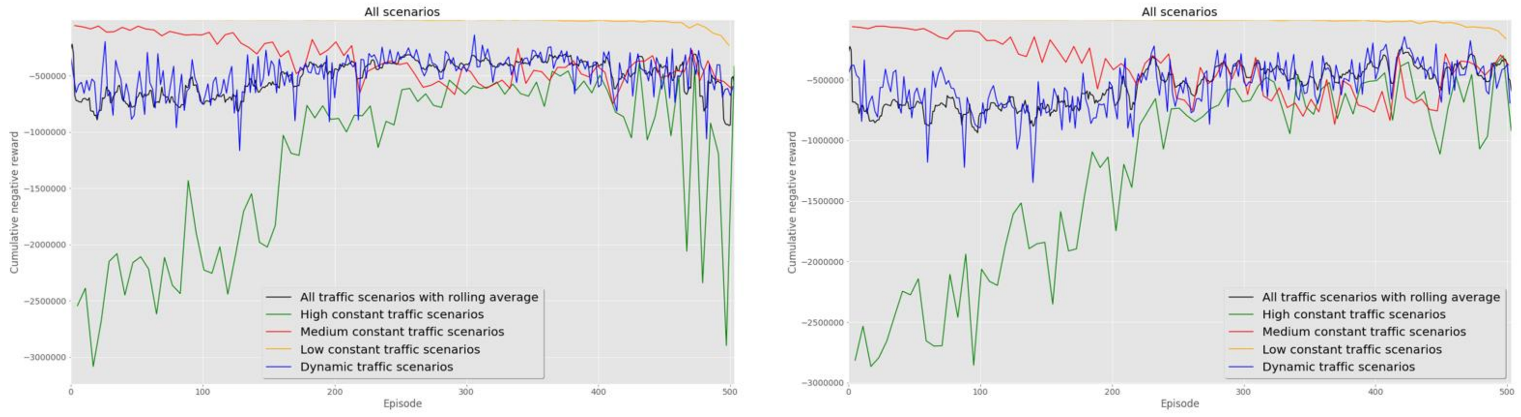


Figure 27 Cumulative negative reward per episode during training. Left figure: one-hot encoded phase. Right figure: +/- encoded phase.

The experiments with the first base agent are inconclusive. While the +/- encoded agent seems to perform slightly better (in the medium and dynamic traffic scenarios) and to be slightly more stable (especially in the high and medium scenarios), both agents are too unstable to draw any definite conclusions.

Figure 27 shows the training rewards for the experiments with the second base agent. In both cases, the agent initially learns how to control the high traffic scenario. However, in the one-hot encoded agent, control of the high traffic scenario ends up unstable, while it remains much more stable in the +/- encoded agent. The performance and stability in the low, medium and dynamic scenarios are relatively comparable, but with a slightly better reward for the +/- encoded agent.

Overall, both experiments point to the +/- encoded agent leading to slightly more stable performance and improved results compared to the one-hot encoded agent. This encoding method will be used for future experiments.

8.2.3 Convolutional network architecture

Within literature, different architectures to extract information from images (here: the DTSE) were found. In this section, two layouts will be tested (for full details, see Table 10, Appendix A.3):

(1) Small network: 2 convolutional layers (filter sizes: 8, 4; kernel sizes: 2, 2; stride: 2, 1)
(2) Large network: 3 convolutional layers (filter sizes: 16, 32, 64; kernel sizes: 2, 2, 1; stride: 2, 2, 1)

The results are shown in Figure 28. Both networks are converging after around 100 episodes, yet it can be seen that the smaller network is more stable and leads to a lower reward. Based on this result, the smaller network is preferred. Nevertheless, it may be possible that the larger network may perform better when trained with different settings, but this is left for future research.

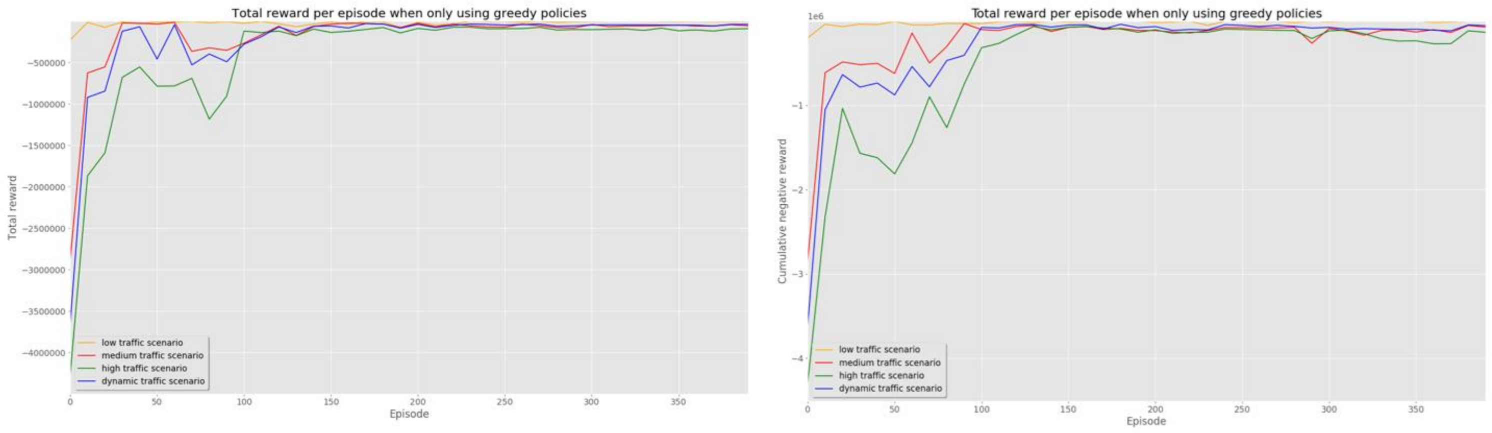


Figure 28 Cumulative negative reward per greedy episode during training. Left figure: smaller network. Right figure: larger network.

8.2.4 Reward representation

Section 6.3.3 described eight different reward functions which could be investigated: the absolute value of or the change in cumulative delay, cumulative waiting time, total queue length or 50/50 weighted cumulative delay and waiting time. In this preliminary experiment, it is investigated which reward leads to better results. Experiments are conducted with the settings described in Table 11, appendix A.4.

Figure 29 shows the cumulative delay per episode for the greedy testing episodes during training. Only the high traffic scenario is shown here. The other scenarios show similar behavior and can be seen in appendix A.4. The plots for the waiting time, queue length and reward show the same trends and have been excluded.

The results in all four scenarios show that the agents using the absolute waiting time, absolute delay, change in delay and the absolute weighted delay/waiting time are not converging, which means that no suitable policy is being learned. These agents are unstable and remain poor in performance. Four reward functions lead to converging results: change in waiting time, the change in queue length, absolute queue length and the change in weighted delay and waiting time. Agent's using these reward functions are able to learn much-better-than-random policies.

Overall, it seems that agents using the absolute-valued reward functions do not converge (except for the queue length) and that agents using a change in KPI are converging well (except for the delay). Why this is the case is unclear. A possible explanation for this may however be that using a value of change immediately shows the agent whether an action improves the current traffic situation (leading to a positive-valued reward) or worsens the situation (leading to a negative-valued reward). An agent that only uses absolute values has to learn this by itself. Nevertheless, this does not explain why this does not hold for the change in delay or the absolute queue length.

Furthermore, it can be noted that using a weighted reward function (here a 50/50 weighted reward consisting of delay and waiting time) does not result in an agent whose performance is halfway between agents using only delay or only waiting time as reward. Surprisingly, the agent using the change in delay

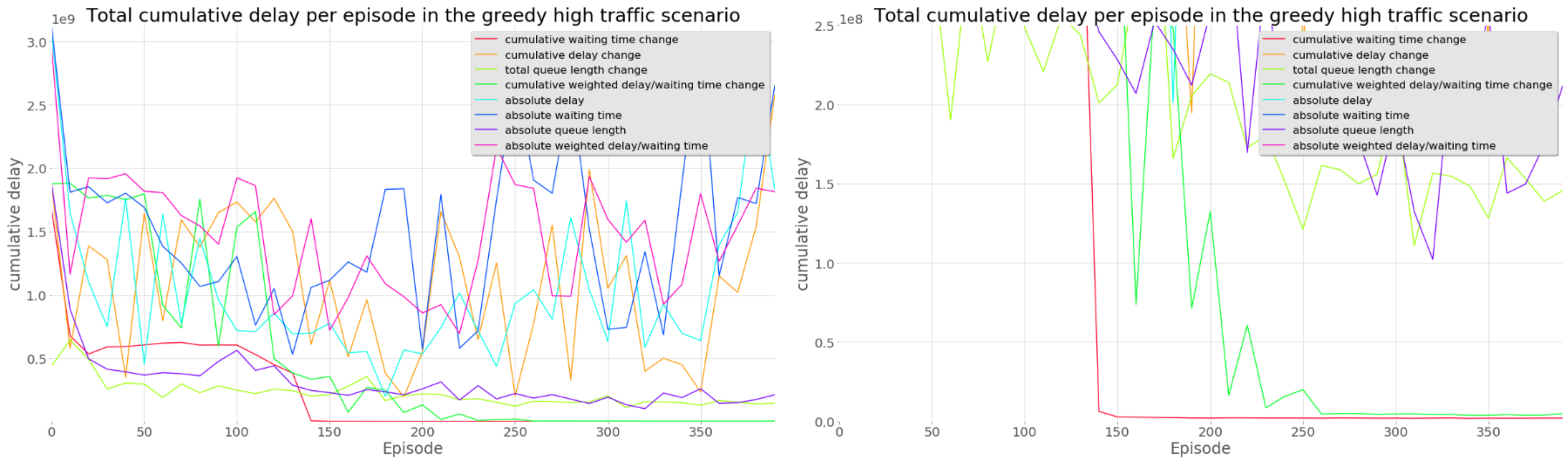


Figure 29 Impact of different reward functions: cumulative delay per episode during training using only greedy policies in the high traffic scenario. Left: overview over all reward functions. Right: zoomed in view over the best-performing reward functions

performs poorly, while the agent which uses the weighted delay and waiting time performs very well. This shows that even a reward which leads to bad results by itself can lead to acceptable performance when used in a weighted function. Yet, how the different sub-rewards interact within a weighted reward is difficult to predict. Further research that tests more weighted reward functions will be needed.

To determine which of the four converging reward functions should be chosen, the plot on the left of Figure 29 is zoomed in on only the best-performing agents (see right figure). In this zoomed-in view it can be seen that in all 4 scenarios the change in waiting time leads to the most stable agent and the lowest cumulative delay. The agent using this reward converges after approximately 140 episodes and after that remains very stable. The other three agents which lead to convergence (using the rewards change in queue length, absolute queue length and the change in weighted delay and waiting time) lead to less stable results and a higher final delay time. Based on this result, the change in cumulative waiting time is used as a reward in the fine-tuned agent.

8.2.5 Numbers of epochs and episodes

An agent must be trained on several episodes for a certain number of epochs. However, it is difficult to determine how many episodes and epochs are needed a priori, since the number of training instances will depend on other model choices, e.g. the state representation, reward, memory size, network architecture. In the experiments, different combinations of values for epochs and episodes were tested.

First, it was tested how epochs and episodes are related. Figure 30 shows agents with the same settings (see Table 12 in Appendix A.5) trained for different number of episodes and epochs: (1) 504 episodes, 1000 epochs (2) 250 episodes, 2000 epochs (3) 1000 episodes, 500 epochs. These values lead to each agent being trained for approximately the same amount of training batches[5] (≈ 500,000). The batch size in this experiment is 100, meaning the three agents being trained on ≈50,000,000 samples. The results of this experiment show that agents trained on fewer episodes, but more epochs lead to less stability and lower performance than agents trained on more episodes but fewer epochs.

[5] 504 episodes * 1000 epochs ≈ 250 episodes * 2000 epochs ≈ 1000 episodes * 500 epochs ≈ 500,000 training batches

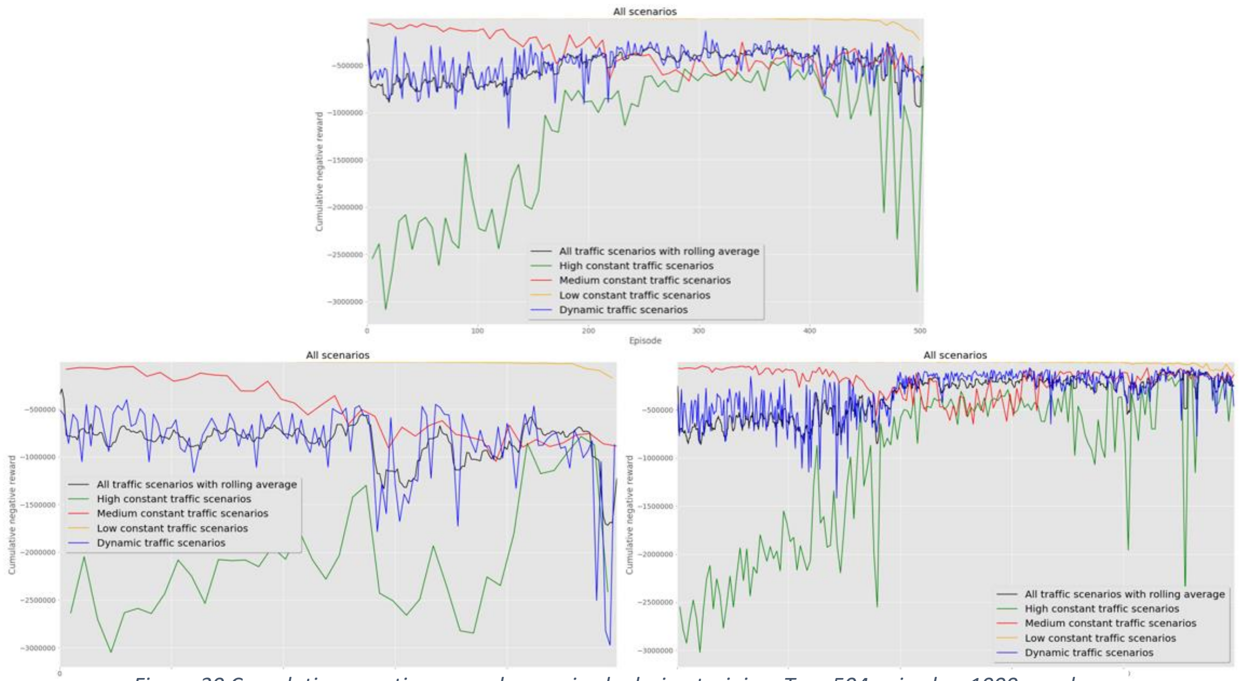


Figure 30 Cumulative negative reward per episode during training. Top: 504 episodes, 1000 epochs. Bottom left: 250 episodes, 2000 epochs. Bottom right: 1000 episodes, 500 epochs

A possible explanation for this would be that the agent with more episodes and fewer epochs sees more variety in training samples. This can be demonstrated with some example calculations. The three agents each have a batch size of 100 and a memory size of 50,000. On average, 675 samples are added to the memory each training episode (see section 7.3 for the calculation), so a sample will be deleted from memory after around 74[6] episodes. An agent trained for 1000 episodes and 500 epochs will see 500 (epochs) * 100 (batch size) = 50,000 samples per episode. In other words, each sample from the memory is seen on average once per episode (assuming a full memory), or 74 times until it is deleted from the memory. An agent trained for 250 episodes and 2000 epochs will see 2000 (epochs) * 100 (batch size) = 200,000 samples per episode, or on average each sample 4 times per episode. In total every sample will be seen around 296 times until deletion. This may be problematic since agents need data to learn new experiences. If one sample is seen for too many times, it can lead to severe overfitting in the model, as seems to be the case for the agent trained for 250 episodes. Training an agent on more episodes but less often on the same samples should be able to combat overfitting.

Nevertheless, note that the agent with the highest amount of episodes and the lowest number of epochs (1000 episodes, 500 epochs) still shows some instability. This can either be caused by untuned other settings, or it may nonetheless point to over- or underfitting. The problem with Figure 30 is that only the reward during training is tracked. This means that the agent partially does random actions, and partially greedy actions. This makes it more difficult to analyze the agent's convergence behavior and thus to determine whether the agent is overtrained or undertrained. To solve this, the next experiments ran a greedy episode for each scenario after every 10 training episodes (see section 8.1.1).

[6] 50,000 (memory size) / 675 (new samples per episode) ≈ 74 episodes

In the second test, combinations of different numbers of episodes (700, 600, 500) and epochs (500, 400, 300) have been tested using the mentioned greedy tests. The settings for this base agent are shown in Appendix A.5, Table 13. Due to space limitations, only the two extreme cases (700 episodes/500 epochs and 500 episodes/300 epochs) are shown in Figure 31. The rest of the experiments can be seen in Figure 54, appendix A.5.

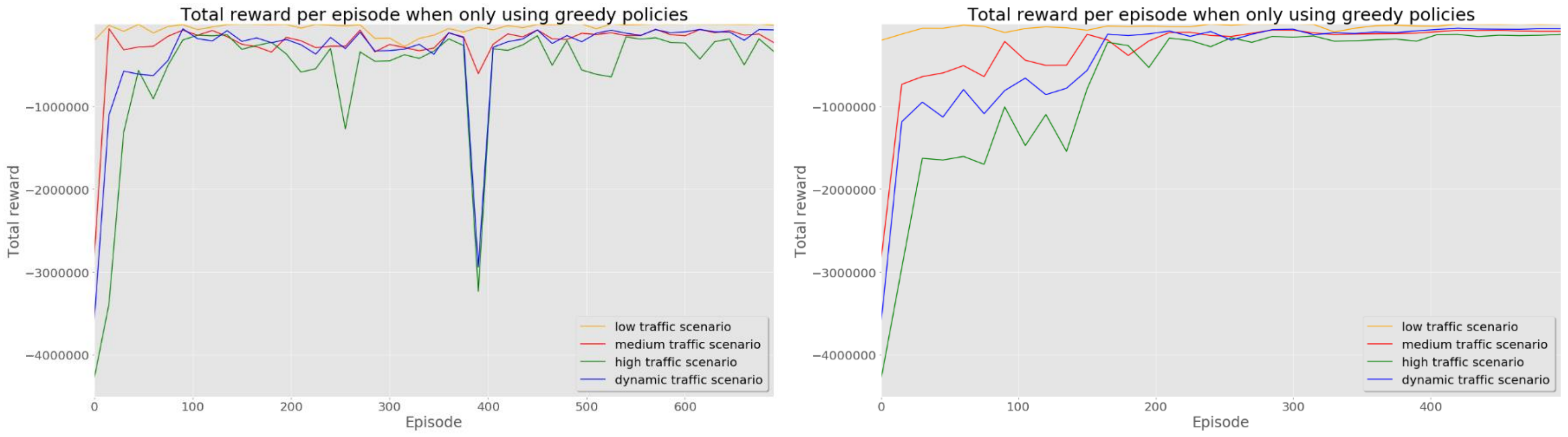


Figure 31 Cumulative negative reward per episode in the greedy episodes during training. Left: 700 episodes, 500 epochs. Right: 500 episodes, 300 epochs.

When looking at the results, agents which were trained on fewer epochs (300 epochs) were more stable than agents trained on more epochs (500 epochs). This is in line with the result from the previous experiment (see Figure 30). This indicates that training an agent too often with the same experience samples can lead to overfitting and thus instability. Based on this result, agents trained with a lower number of epochs are preferred. Indeed, in our experiments all agents trained with 300 epochs converged, so 300 epochs will be used in future experiments.

When looking at the plots of agents trained with 300 epochs, the major performance improvements happen in the first 150 to 200 episodes (see e.g. on the right side of Figure 31). After these first episodes, the agent has converged to relatively stable performance. Nevertheless, slight performance improvements are still being made. As a trade-off between performance and time needed to train the model, 400 episodes are chosen for future experiments.

8.2.6 Target network freeze interval

Section 7.5 discussed the target network freeze interval. The purpose of freezing the target network is to stabilize training, yet in order to do this, an appropriate freeze interval must be set. In this section, different freeze intervals are tested on the base agent presented in Table 14, appendix A.6. Freeze intervals here are specified in terms of training steps. Since the epoch size is 300, a freeze interval of 300 corresponds to freezing the network for 1 full episode. In this test, freeze intervals of 4,000 steps (13.3 episodes), 6,000 steps (20 episodes), 8,0000 steps (26.7 episodes), 10,0000 steps (33.3 episodes), 12,0000 steps (40 episodes), 14,0000 steps (46.7 episodes) and 16,000 steps (53.3 episodes) are being tested.

The results for the greedy episodes in the high traffic scenario are displayed in Figure 32. Results for the other scenarios again show similar patterns and can be seen in Figure 55, appendix A.6.

From the plots it can be seen that increasing the freeze interval leads to higher performance and more stability. Agents with low freeze intervals (4,000 steps/13.3 episodes and 6,000 steps/20 episodes) lead to especially unstable agents with large jumps in performance and overall lower rewards. It seems that in these agents, the target network is updated too often, so the moving target problem persists. With increasing freeze intervals, stability and overall performance increases, thus indicating that the moving target problem is avoided. Nevertheless, it can also be seen that the highest freeze interval does not lead to the best performing agent. Possible explanations can be that the target network updates are too infrequent so the agent may need more training time, or the system starts to move in the wrong direction during updates.

The best performing agent is the agent with a relatively high freeze interval of 14,000 steps. This means the target network is updated every 46.7 episodes or around 8.5 times[7] in total.

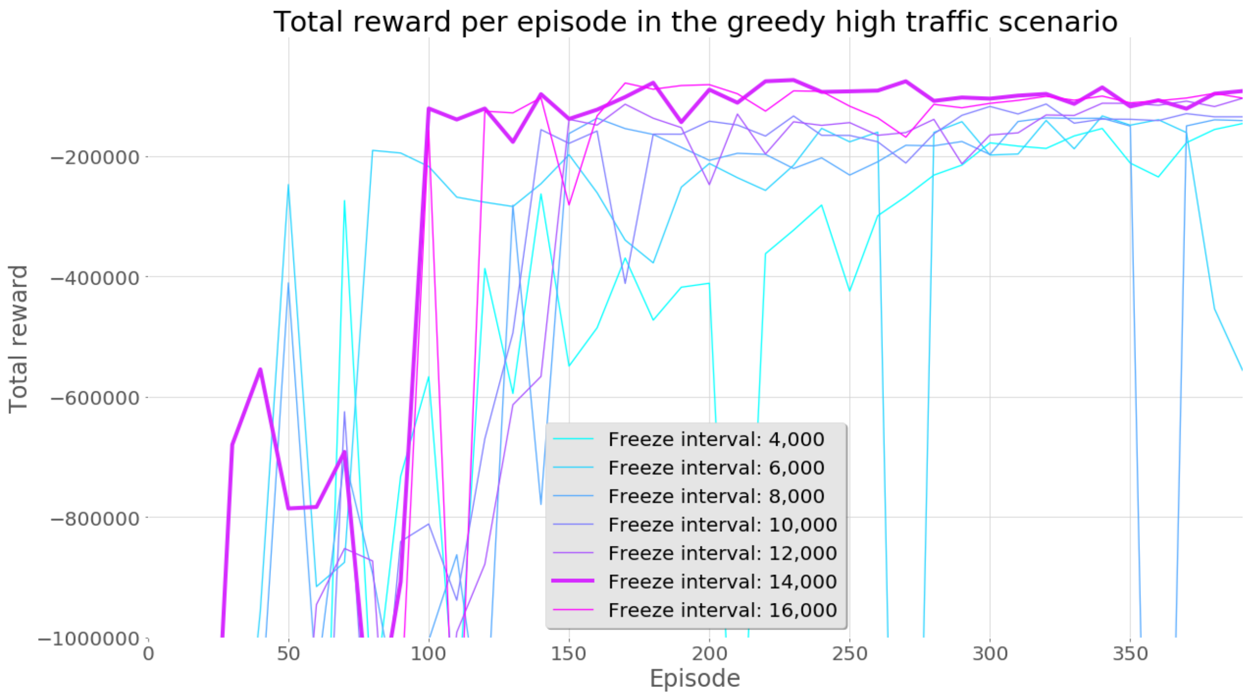


Figure 32 Cumulative reward per greedy episode for the high traffic scenario. The overall best performing agent from all scenarios is highlighted.

8.2.7 Memory size

In section 7.5 the role of the memory size was discussed. In this experiment, agents trained with memory sizes of 30,000, 40,000, 50,000, 60,000, 70,0000 and 80,000 samples are investigated (see Table 15, Appendix A.7 for details).

Figure 33 shows the resulting cumulative rewards per greedy episode for the high traffic scenario. The plots for the other scenarios show similar trends and can be found in Figure 56, appendix A.7.

Overall, the results for the different memory sizes are relatively similar. Yet, it seems that lower memory sizes (30,000 and 40,000) lead to slightly less stable agents. In Figure 33 for example it can be seen that the agent with a memory size of 30,000 finds a good policy at around 110 episodes, but then it forgets it again. For higher memory sizes, some agents reach a high performance and good level of stability (e.g.

[7] 400 (episodes) / 46.6 (episodes between updates) = 8.5 (updated in total)

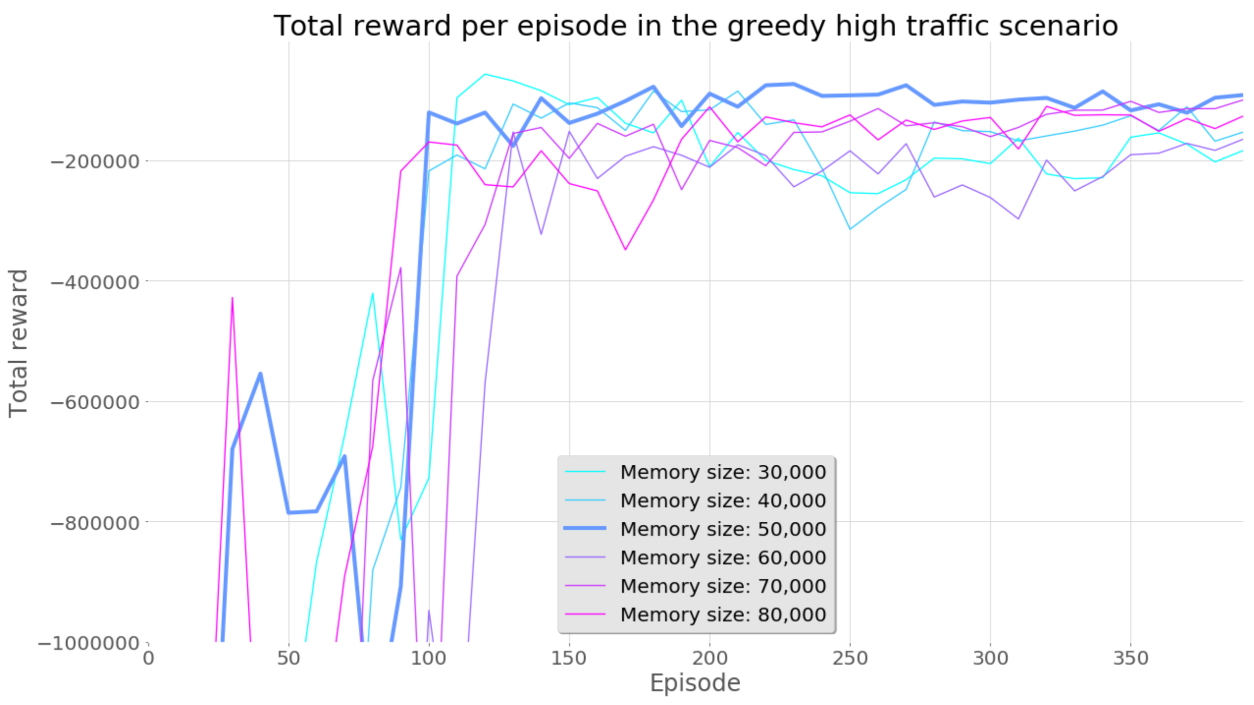


Figure 33 Cumulative reward per greedy episode for the high traffic scenario. The overall best performing agent from all scenarios is highlighted.

the 50,000 agent), while other agents remain worse in performance and stability (e.g. the 60,000 agent). Nevertheless, it was found that none of the highest memory sizes led to the best-performing agents.

From the experiments it can be concluded that a too low memory size can lead to catastrophic forgetting and instability. A possible reason for this is that in order to lead to stable results, the agent must have access to enough different experiences. If the memory gets replaced too often, it is likelier that stored experiences are too similar, which can impede learning.

For bigger memory sizes, no clear conclusions can be drawn. Nevertheless, when deciding on an appropriate memory size it is desirable to have fewer samples in memory since this reduces the memory size required by the algorithm. Additionally, if the memory size gets too large there may be a risk of training the agent on no longer relevant experiences, which may slow down training. Thus, it is advised to pick the lowest memory size which leads to acceptable performance levels and stability.

For the tested agents, a memory size of 50,000 leads to the overall best stability and performance in all scenarios. Thus, a memory size of 50,000 will be used for the fine-tuned agent.

8.2.8 Trajectory sequence length (only for DRQN)

Section 7.5 discussed the impact of the trajectory sequence length. In this preliminary experiment, difference sequence lengths are tested: 9, 12, 15, 18, 21, and 24 samples per trajectory. Since sequence lengths should work well for both 100% CV-penetration and less than 100% CV-penetration, the experiments are carried out using both 100%- and 50%-CV-penetration agents. For more details on the agents, refer to Table 16, appendix A.8.

Figure 34 shows the results for the medium traffic scenario. The plots for the other scenarios can be seen in Figure 57, appendix A.8. From the plots it can be seen that when a too short trajectory length is used, results are less stable and may take longer to converge. This is particularly the case for the agent trained on a 50% CV-penetration rate. In Figure 34 for example, the left-hand side shows that agents with trajectory lengths of 9 or 15 samples converge less quickly and perform less well than the agents trained with higher trajectory lengths. For agents trained on longer trajectories, agent stability and final performance are similar. Furthermore, the difference between agents is more pronounced for the 50% CV-penetration agent than for the 100% CV-penetration agent.

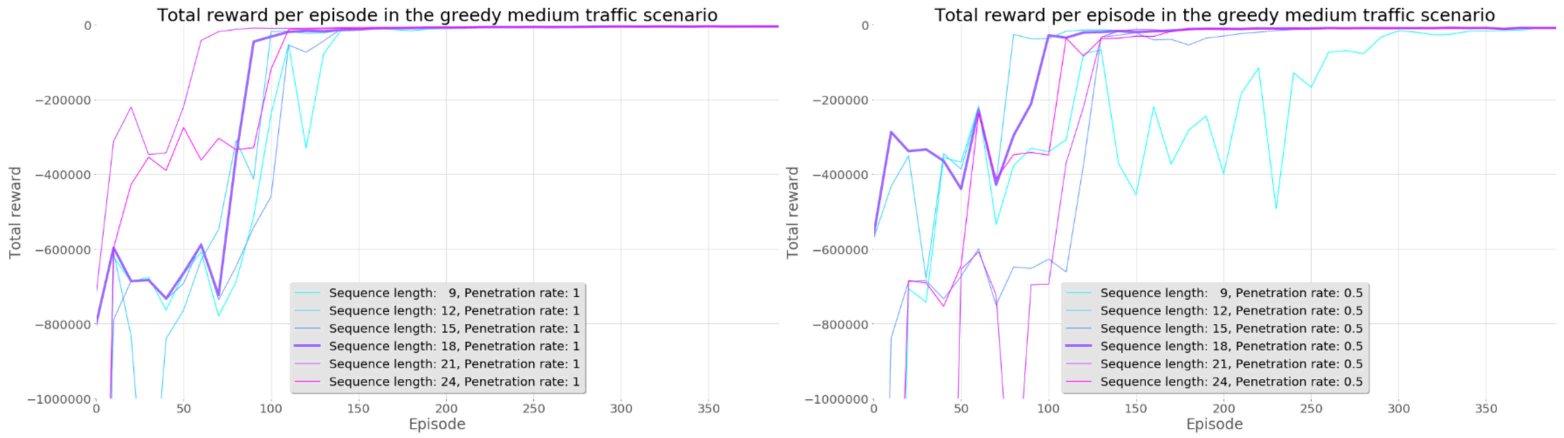


Figure 34 Cumulative reward per greedy episode for the medium traffic scenario. The agent with the best trade-off between performance and training time from all scenarios is highlighted. Left figure: CV-penetration rate of 100%. Right figure: CV-penetration rate of 50%.

From these results it can be concluded that when an agent is trained on longer trajectories, the performance and stability of the agent improves. This result is intuitive since longer trajectories mean that the agent has access to a longer history which means that the hidden state will be more accurate. Particularly for agents which are trained under a CV-penetration rate less than 100%, having an accurate hidden state is important since the agent cannot observe the full state, which means the agent has to rely more on the past observations.

Nevertheless, the plots also show that the longest trajectory length (here 24 samples) does not necessarily lead to the best performance. Dependent on the scenario and the CV-penetration rate, the best-performing agents use trajectory lengths between 18 and 24 samples. It may be possible that after a certain point adding more samples to trajectories does not make the hidden state significantly more accurate, or it may be possible that the i.i.d. assumption gets overly violated. Additionally, adding more samples to a trajectory increases the training time, which is undesirable if it does not increase in performance improvements.

Due to this, the agent with the best trade-off of training time and stability and performance is chosen. In this case, the agent with a trajectory length of 18 will be used in the fine-tuned agent.

8.2.9 Fine-tuned agent

After having conducted all the preliminary experiments, the best performing parameters are combined into a fine-tuned agent. For the final experiments, a DQN and a DRQN agent will be built. Both of them will be trained under penetration rates between 10% and 100%. The final fine-tuned agent settings are presented in Table 6.

Table 6 Model settings and parameters of the fine-tuned agent

Model setting or (Hyper-)parameter	**Value**
Training episodes	400
Training epochs	300
Minibatch size	100
Target network freeze interval [steps]	14,000
Replay memory size	50,000
Trajectory length (only for DRQN)	18
Discount factor	0.99
Learning rate	0.001
Penetration rates	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1
Network architecture	Convolution: Filters: 4, 8 Kernel: 2, 2 Stride: 2, 1 Vehicle position: Density (float) Phase: +/- encoded
Reward	Change in cumulative waiting time
Recurrence	No / Yes (sequence length: 18)

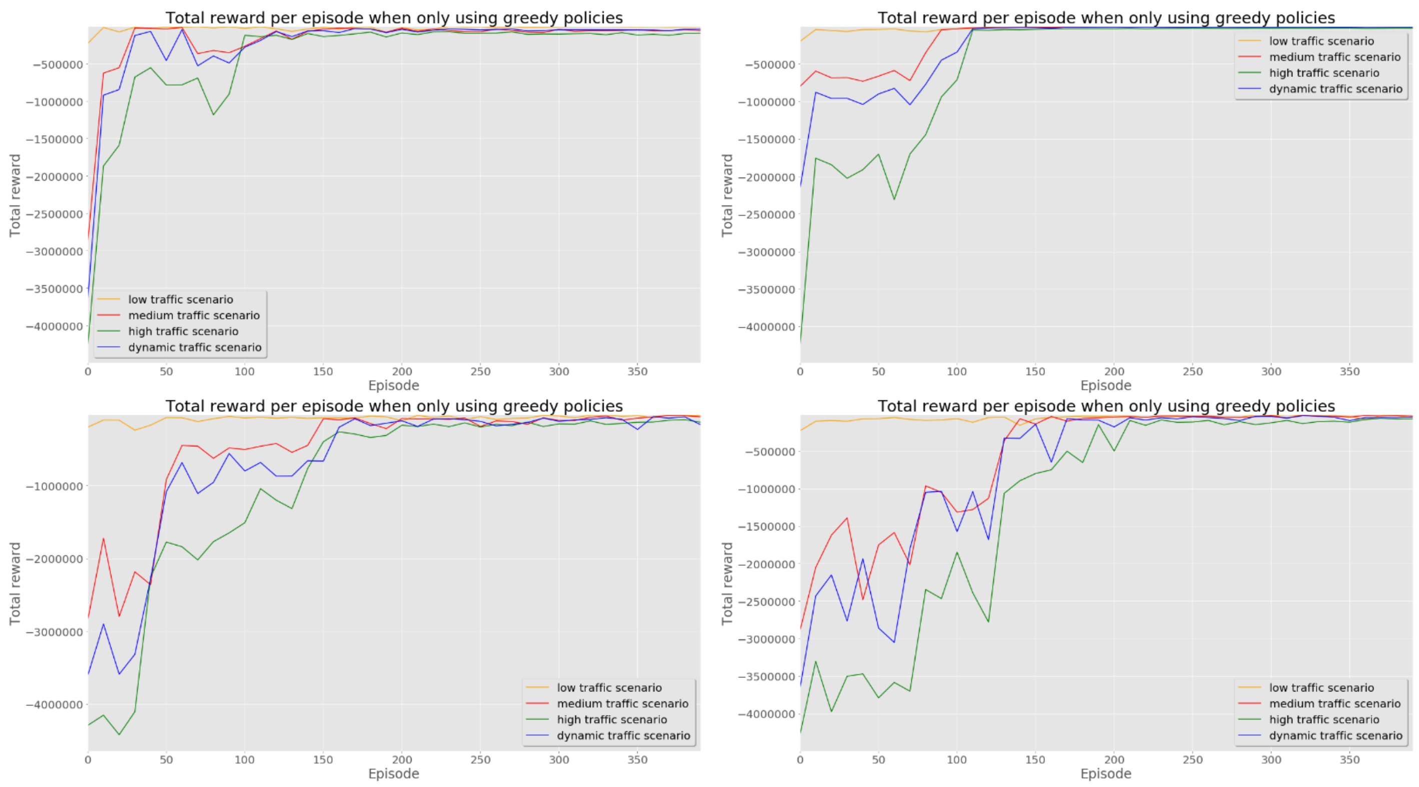


Figure 35 Cumulative negative reward per greedy episode during training of the fine-tuned agent. Top left: Vanilla DQN trained under 100% CV-penetration. Top right: Recurrent DQN trained under 100% CV-penetration. Bottom left: Vanilla DQN trained under 10% CV-penetration. Bottom right: Recurrent DQN trained under 10% CV-penetration

Figure 35 shows the cumulative negative reward per greedy episode for the fine-tuned agents for the vanilla DQN agent (left) and the recurrent DQN agent (right). Due to space limitations, only the two extreme penetration rates are shown here: the 100% CV-penetration agent is shown on the left, the 10% agent on the right. For the agents trained under different penetration rates, refer to Appendix A.9.

When looking at the plots, it can be seen that in all 20 agents better-than-random policies have been learned. Furthermore, it can be noted that the vanilla agents are less stable and gain lower rewards than the recurrent agents. This is intuitive, since recurrent agents have access to a hidden state which provides them with more information to decide on the next action. This first impression will be further evaluated in chapter 8.3.1.

8.3 Penetration rate test results

In this section, the results of testing the fine-tuned agent are presented and discussed. Each agent has been tested on 50 unique runs. First the agent stability is analyzed and then the traffic scenarios will be analyzed separately. Next, the performance of the vanilla and recurrent agents is compared. Lastly, the robustness of both agents is evaluated.

8.3.1 Analysis of agent stability

When plotting the KPIs for a single scenario and penetration rate, results like in Figure 36 were seen. Since scenarios are generated stochastically, the trained agents will perform slightly differently for each of the runs. To analyze how stable the trained agents are, the median 5-95 percentile range is chosen. It measures the difference between the 5% and the 95% percentile (i.e. the spread of the blue area in Figure

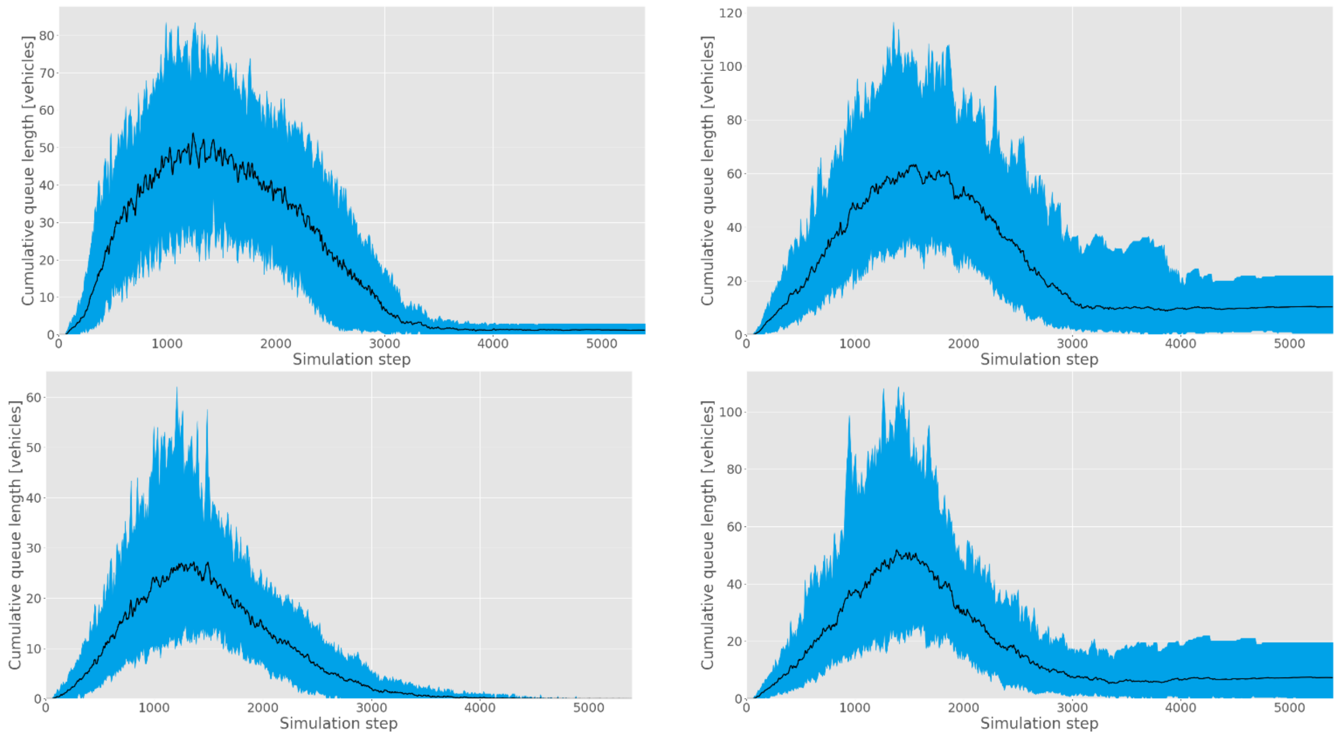
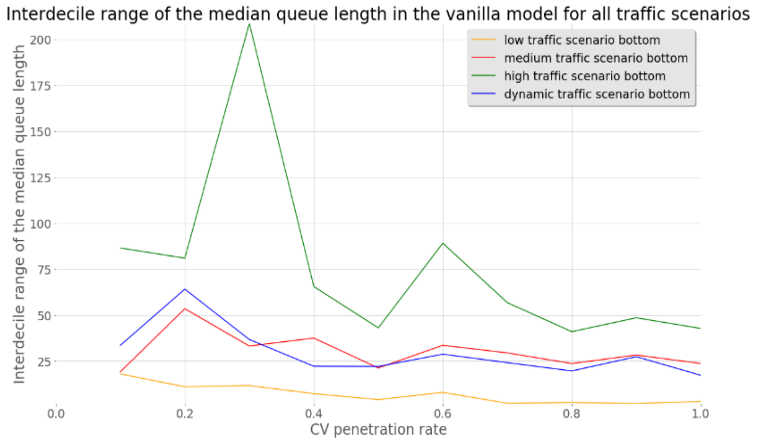


Figure 36 Examples of diverging KPIs in test runs. The black line shows the average values and the blue area shows the 5% to 95% percentiles. All example figures show the cumulative queue length in the dynamic traffic scenario. Top left: vanilla agent under 100% CV-penetration. Top right: vanilla agent under 10% CV-penetration. Bottom left: recurrent agent under 100% CV-penetration. Bottom right: recurrent agent under 10% CV-penetration.

36). Figure 37 shows the stability measured in terms of the queue length for both agent types in all 4 scenarios. For the results of the other KPI, refer to Appendix B.1.

In the figures it can be seen that the recurrent agent has a lower range for all scenarios and penetration rates than the vanilla agent. This means that the recurrent agent is more stable than the vanilla agent, thus supporting hypothesis 8 (see section 8.1.3).

Furthermore, for the recurrent agent it can be observed that the stability increases with higher penetration rates (except for the 70% agent), which supports hypothesis 1. This result is expected, since agents with higher penetration rates know more about the state than agents trained on lower penetration rates, thus they can make better decisions. For the vanilla agent however, higher penetration rates do not necessarily lead to higher stability. See for example the 10% in the medium traffic scenario. This agent has the lowest range, despite being trained under the lowest penetration rate. Why the vanilla agent is stable for some penetration rates but unstable for others is unclear. It may be possible that in the less stable cases the algorithm got stuck in local optima.



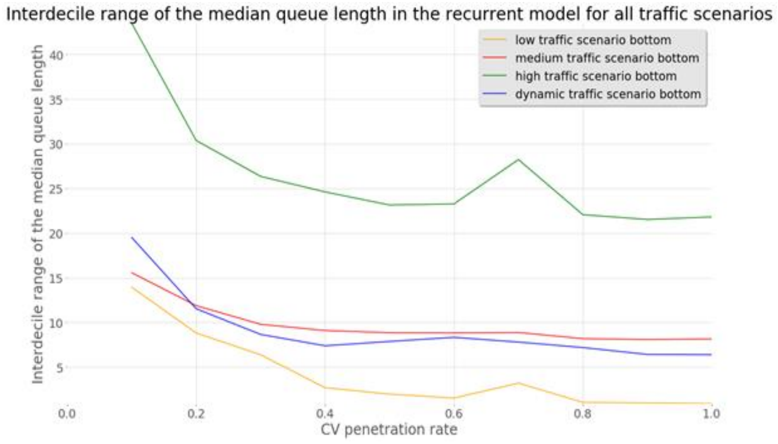


Figure 37 Stability of the trained agents. Both figures plot the Interdecile range of the median queue length. Left: Vanilla agent. Right: Recurrent agent.

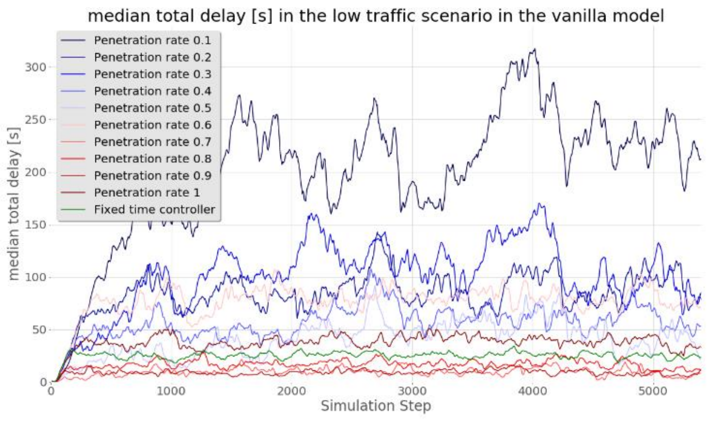
8.3.2 Analysis of results per traffic scenario

Analyzing scenario runs separately per KPI, model type and scenario is difficult due to the high number of generated plots. Instead of looking at single plots, it is more useful to combine several runs into a single plot, since this facilitates easier comparisons. In these comparisons, the distribution of the KPI values (i.e. the blue areas in plots such as Figure 36) is excluded, since the values diverge too much to be able to plot them or to be able to provide useful results. Instead, every single run will be quantified using the median over the 50 runs. It was chosen to use medians rather than averages since a few single runs had large unexplained outliers which unfairly skewed the average.

The next section will compare the results per scenario.

8.3.2.1 Low traffic scenario

Figure 38 shows the median cumulative delays per simulation step for agents trained under all penetration rates for the low traffic scenario. Figure 39 shows the average vehicle delay per episode broken down into



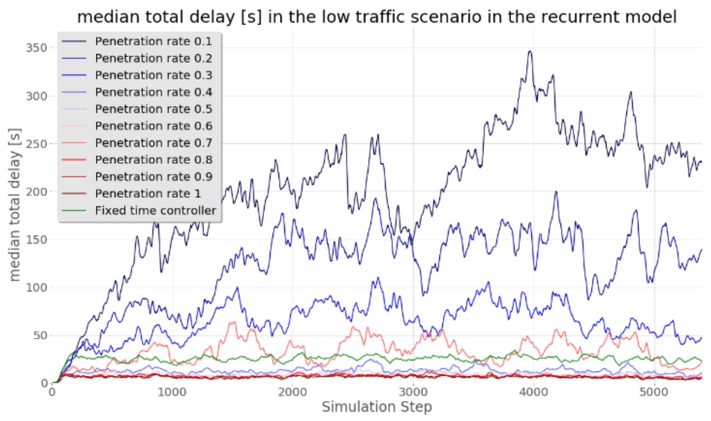
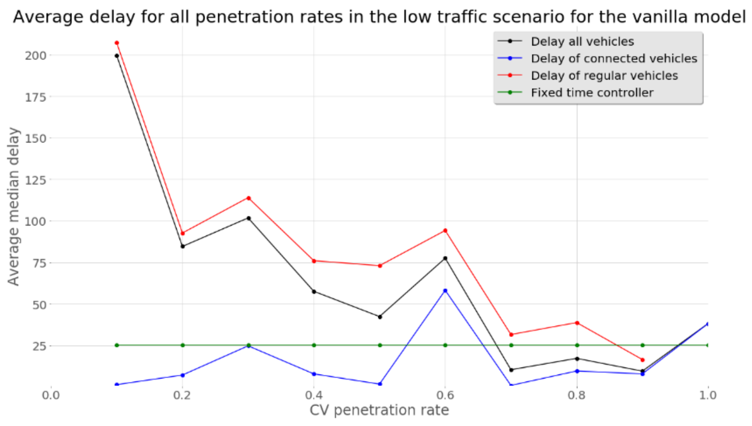


Figure 38 Median cumulative delay in the low traffic scenario for all penetration rates. Left figure: Vanilla DQN agent. Right figure: Recurrent DQN agent.

the delay for connected vehicles and delay for regular vehicles[8]. In both figures, the left side shows the results for the vanilla agent, and the right side the results for the recurrent agent. Figure 40 compares the average vehicle delay between the vanilla and the recurrent model. Since in all cases the plots for the delays, waiting times and queue lengths showed similar trends, only the plots of the delays have been included in the main text (refer to Appendix B.1 for the plots of the other KPI).

In Figure 38 it can be seen that for both the vanilla and recurrent agents, the delays, waiting times and queue lengths are relatively constant over the full episode (when ignoring the noise of the quick up and down jumps). When looking at the queue lengths, queues for agents with higher penetration rates remain around or under 1 queuing vehicle, while for the lowest penetration rate the queues remain generally between 4 to 7 vehicles long. For all agents this means that all vehicles can pass the intersection at some point, and that no vehicles have to wait infinitely long.

Both the vanilla and the recurrent agent show that the higher the penetration rate is, the lower the delays, waiting times and queue lengths are. This trend is especially visible in the recurrent agent: for all increases



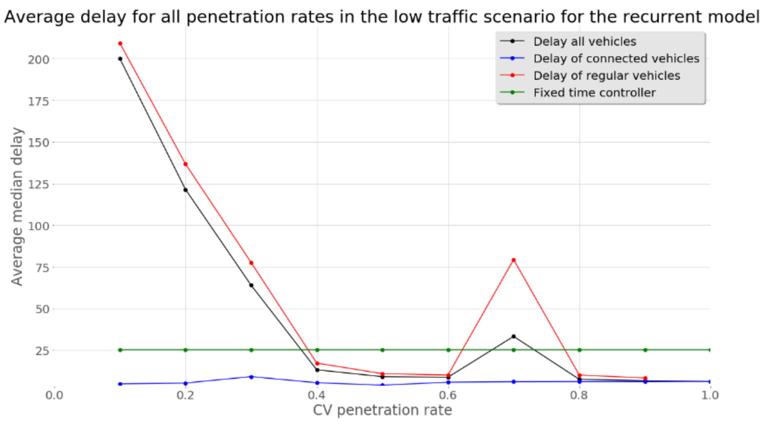


Figure 39 Average median cumulative delay for all vehicles, for connected vehicles and for regular vehicles in the low traffic scenario Left figure: Vanilla DQN agent. Right figure: Recurrent DQN agent.

[8] For plotting purposes, only the averages of the run medians have been used.

in penetration rates, the performance of the model improves (the only exception being the 70%-agent). For the vanilla agent, the trend can also be seen, albeit not as clearly as for the recurrent agent since agents trained under penetration rates of 30%, 60%, 80% and 100% do not improve the agent's performance. Nevertheless, in both models an overall downwards trend can be seen, which supports hypothesis 2 (see section 8.1.3).

Section 8.1.3 further hypothesized that for the low traffic scenario, CVs will have much lower delays than RVs (hypothesis 3), but it is also expected that this difference will get smaller with increasing CV rates (hypothesis 4). When looking at Figure 39, it can be seen that as expected in hypothesis 3, the delay for connected vehicles is lower than for regular vehicles. For both model types it can be seen that when the penetration rate is increased, the difference between connected and regular vehicles decreases. For agents trained under a penetration rate of 10%, the delay for connected vehicles is around 0 seconds for both model types, while the delay for unconnected vehicles is around 205 seconds. When looking at the agents trained under 90% CV-penetration, the difference between connected and unconnected vehicles is very small. These results can be explained as follows: in the low traffic scenario, cars are arriving one-by-one, which means that the controller has to react to every arriving car individually. The controller can however only see the connected vehicles. In high penetration rates this means the controller can switch the traffic signal to a green phase for the majority of the arriving vehicles. But for a low penetration rate, if an unconnected vehicle arrives, the controller cannot observe and react to it. Instead, the vehicle will have to wait until either a connected vehicle arrives on the same leg (which may take long, since there are only very few connected vehicles) or it will have to be lucky that the controller switches to a green phase despite not seeing the vehicle.

Lastly, Figure 40 shows that for most penetration rates the recurrent model performs better than the vanilla agent. The vanilla agent's performance improvements with higher penetration rates are much slower and less stable than those of the recurrent agent. For the vanilla agent, a penetration rate of 70% or higher is required to reach a better performance than the fixed time controller (with 100% being an outlier). For the recurrent agent, a penetration rate as low as 40% is sufficient to perform better than the fixed time controller (with 70% being an outlier). From these results we can conclude that reinforcement learning models can outperform fixed-time controllers in the low traffic scenario for sufficiently high CV-penetration rates, supporting hypothesis 7. Furthermore, it is concluded that recurrent agents generally perform better and reach these performance improvements faster, which supports hypothesis 9.

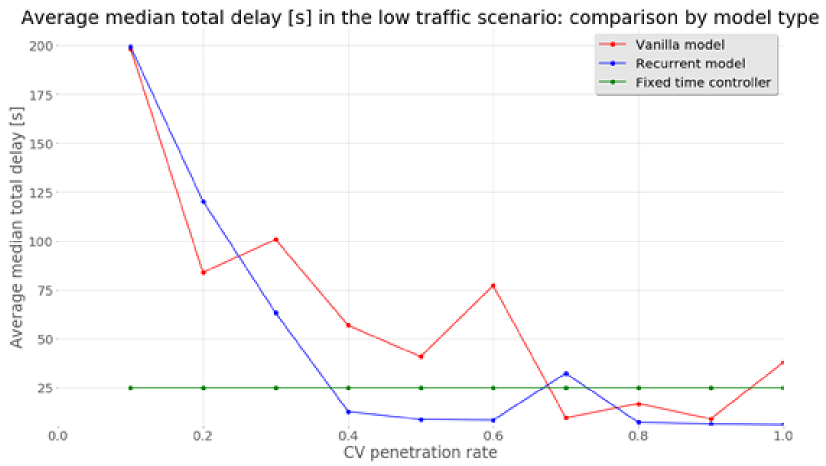
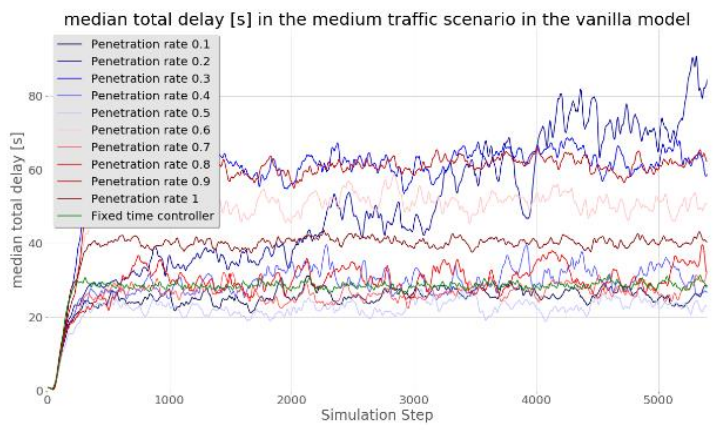


Figure 40 Comparison between the vanilla and recurrent agents: average median cumulative delay in the low traffic scenario.



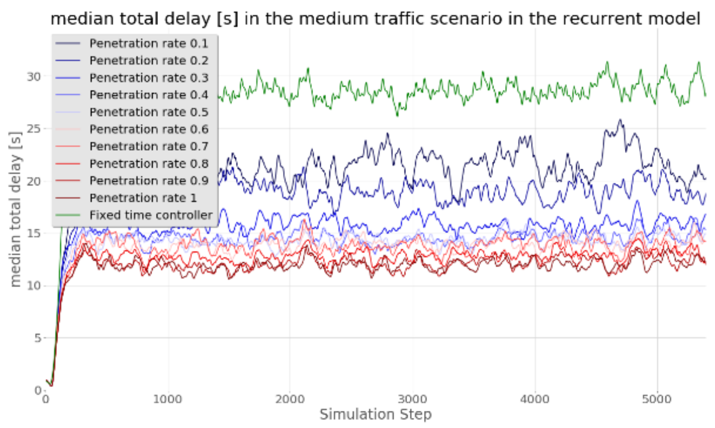


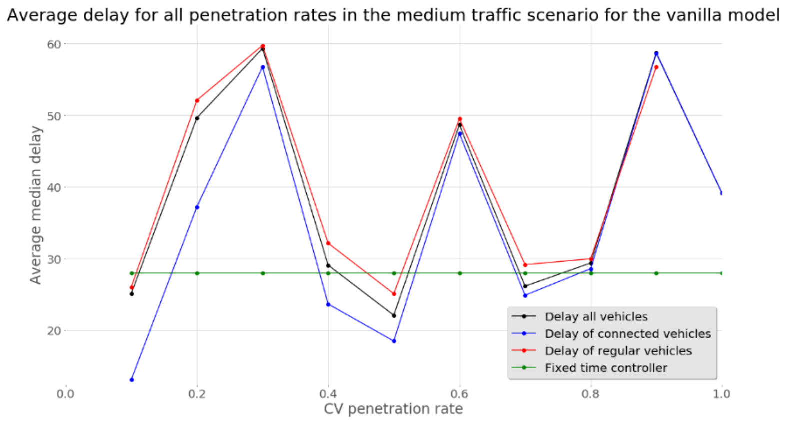
Figure 41 Median cumulative delay in the medium traffic scenario for all penetration rates. Left figure: Vanilla DQN agent. Right figure: Recurrent DQN agent.

8.3.2.2 Medium traffic scenario

The results for the medium traffic scenario are shown in Figure 41, Figure 42 and Figure 43. For full details, refer to Appendix B.2.

Like for the low traffic scenario, the delays, waiting times and queue lengths in the medium traffic scenario remain relatively stable over the full episode (when ignoring the jumps), which means that the agent has learned a strategy that allows all vehicles to pass the intersection. Only for the vanilla agent trained under a 10% penetration this is not the case: for this agent, all three KPI are continuously decreasing, which points towards a few vehicles being "stuck" at the intersection. This can happen if one of the lanes never receives a green signal, so vehicles can never pass.

Looking at figures Figure 41 and Figure 43, it can be seen that for the recurrent agent, the median delay is decreasing for every increase in CV-penetration. The major performance improvements for the recurrent agent are made between 20%-30% CV-penetration; after this the performance improvement rate is lower but constant. For the vanilla agent however, no such trend can be observed. Agents trained under penetration rates of 10%, 40%, 50%, 70% and 80% perform relatively well (although still worse than any of the recurrent agents), while agent trained under penetration rates of 20%, 30%, 60%, 90% and 100% perform very bad. This is unexpected since the two best performing agents are trained under low



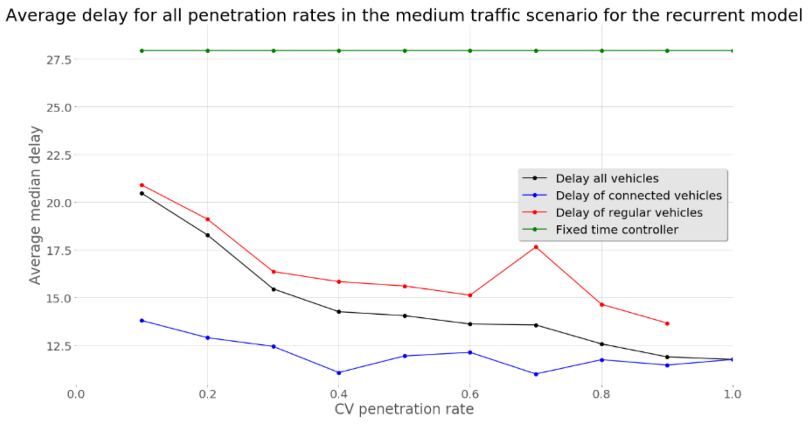


Figure 42 Average median cumulative delay for all vehicles, for connected vehicles and for regular vehicles in the medium traffic scenario Left figure: Vanilla DQN agent. Right figure: Recurrent DQN agent.

penetration rates (10%, 40%), while two of the bad performing agents are trained under high penetration rates (90%, 100%). One possible reason for this may be that some agents “sacrificed” performance in the medium traffic scenario for better performance in other scenarios. Based on these results it can be concluded that hypothesis 2 is confirmed for the recurrent agent but rejected for the vanilla agent.

When comparing the differences in delays between connected and regular vehicles (see Figure 42), similar conclusions like for the low traffic scenario can be drawn: CVs have lower delays than RVs, and this difference generally gets smaller with increasing penetration rates. Hypotheses 3 and 4 are thus confirmed for the medium traffic scenario.

Furthermore, both models can be compared to the fixed-time controller. It can be seen that the recurrent agent significantly outperforms the fixed-time controller, even for penetration rates as low as 10%. This means that even under super low penetration rates DRQN agents can find traffic signal control policies which significantly reduce congestion compared to fixed-time controllers. This however is not the case for the vanilla agent. For this model type, only the agents trained under 10%, 50% and 70% CV-penetration can outperform the fixed-time controller. For the other agents, no suitable policies were found.

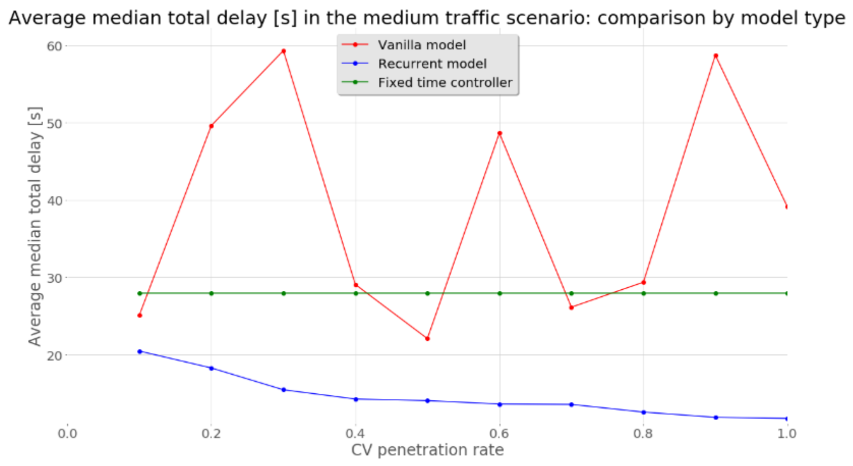


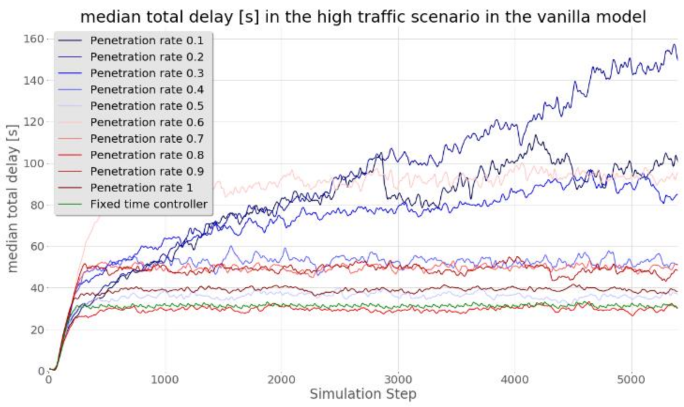
Figure 43 Comparison between the vanilla and recurrent agents: average median cumulative delay in the medium traffic scenario.

8.3.2.3 High traffic scenario

The results for the medium traffic scenario are shown in Figure 44, Figure 45 and Figure 46. Again, the trends for the queue lengths and waiting times are similar to the trends for the median delays, and as such are excluded here (see Appendix B.3 for full results).

Looking at Figure 44, it can be seen that for the vanilla agents trained under the lowest penetration rates (10%-30%), the delays, waiting times and queue lengths increase continuously. For these three agents, this likely means that not all vehicles can pass the intersection due to some lanes having an infinite red signal. The vanilla agents trained under higher penetration rates and all recurrent agents have learned policies in which all agents can pass the intersection.

When looking at Figure 44 and Figure 46 similar trends as for the medium traffic scenario can be seen. For the recurrent agents, the delay is continuously decreasing with higher penetration rates (with the exception of the agent trained at 70% CV-penetration). This supports the second hypothesis. The biggest



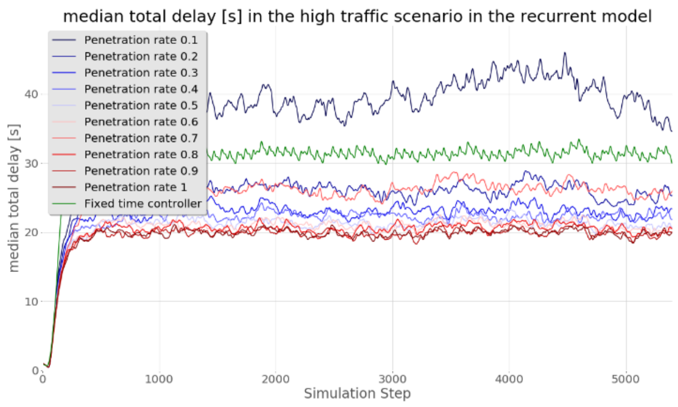
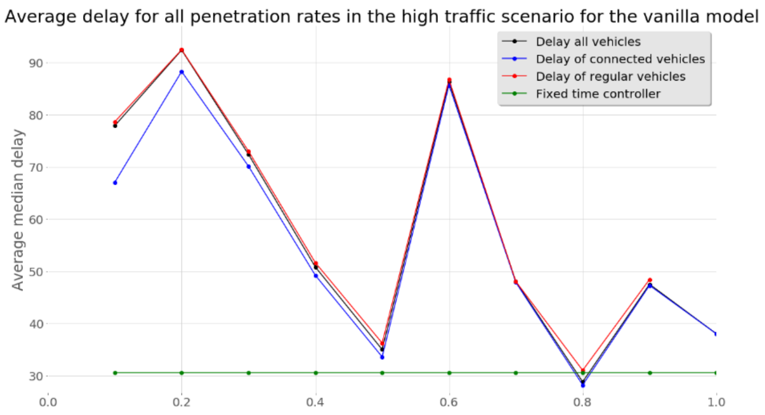


Figure 44 Median cumulative delay in the high traffic scenario for all penetration rates. Left figure: Vanilla DQN agent. Right figure: Recurrent DQN agent.

performance improvements are made within the first 30% CV-penetration. For the vanilla agents, hypothesis 2 is only true for some of the agents (30%, 40%, 50%, 80%), so it cannot be confirmed.

Regarding the differences between connected vehicles and regular vehicles, Figure 45 shows that for both model types the RV delay is higher than the CV delay, which is in accordance with hypothesis 3. It can also be seen that the difference between CV and RV delay gets smaller for higher penetration rates, thus supporting hypothesis 4. Furthermore, it can be noted that the difference between CVs and RVs is smaller for the high traffic scenario than for the low traffic scenario. This is in particular the case for the low penetration rates. Compare for example and Figure 45. In the low traffic scenario under 10% CV-penetration, the difference in delay between CVs and RVs is around 205 seconds for both model types. In the high traffic scenario, the difference for the 10%-agent is only around 7-12 seconds. This supports hypothesis 5: more cars in a scenario lead to a lower difference in delay times between CVs and RVs. It seems that indeed the agent no longer reacts to vehicles individually, but rather reacts to groups of vehicles. In the high traffic scenario likely enough vehicles are generated that the regular vehicles are "surrounded" by connected vehicles, so then they can cross the intersection when the controller reacts to the surrounding CVs.

Lastly, the performance of the vanilla and recurrent agents can be compared. Figure 46 shows that all but the 10%-agent of the recurrent agents outperform the fixed-time controller. For the vanilla agent, only



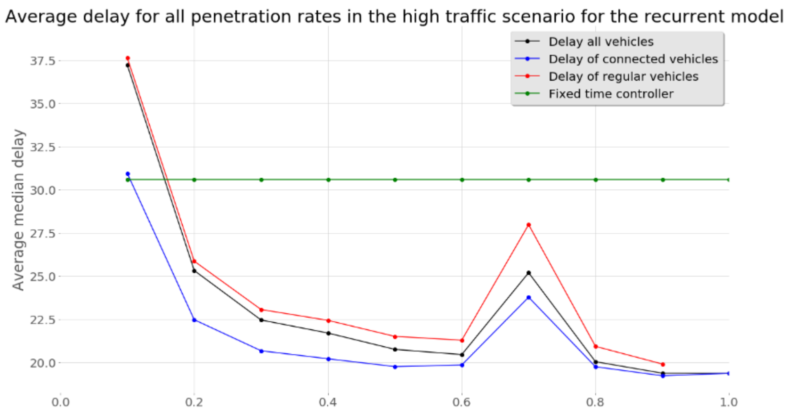


Figure 45 Average median cumulative delay for all vehicles, for connected vehicles and for regular vehicles in the high traffic scenario Left figure: Vanilla DQN agent. Right figure: Recurrent DQN agent.

the agent trained under 80% CV-penetration can (barely) outperform the fixed-time controller. It can be concluded that the recurrent DQN agent is able to reduce congestion much better than both the fixed-time controller and vanilla DQN controller.

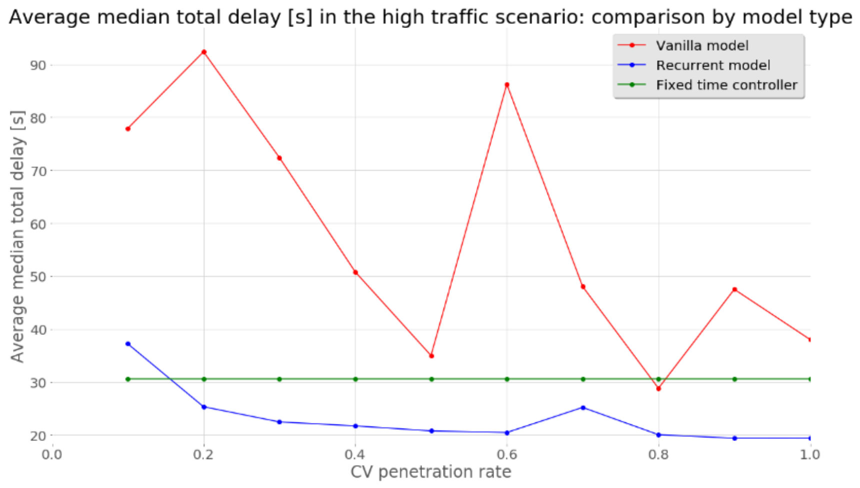


Figure 46 Comparison between the vanilla and recurrent agents: average median cumulative delay in the high traffic scenario.

8.3.2.4 Dynamic traffic scenario

The results for the dynamic scenario are shown in Figure 47, Figure 48 and Figure 49. For this agent, the trends in the plots of the delays and waiting times are different from the trends in the plots of the queue lengths. Due to this, both the delay and queue length plots are included here. For full details, refer to Appendix B.4.

Figure 47 shows different behavior than the plots for the other scenarios. When looking at the plots of the delays, it can be seen that for many of the agents the delays increase drastically towards the end of the scenario. This can be explained when examining the plots of the queue lengths: at the end of the scenario, the queue lengths for those agents remain higher than 0, even though no new cars are arriving. This means that some vehicles cannot pass the intersection because they have a red light forever. Since they cannot pass, the delay and waiting times of these vehicles keep accumulating infinitely, resulting in the steep rise in the plots.

Nevertheless, this behavior does not occur for all agents. It seems that when agents are trained under higher penetration rates, this phenomenon is less likely to occur. For the vanilla model, this behavior does not occur for agents trained under penetration rates of 70%, 80% and 90%, and for the recurrent model this does not occur for agents trained under penetration rates of 50%, 60%, 80%, 90% and 100%. From this it can be concluded that DQN and DRQN agents can learn to control dynamic scenarios such that all cars can cross the intersection, as long as the penetration rate is sufficiently high.

When comparing the performance to the fixed-time controller, it can be seen that the recurrent model is able to greatly outperform the fixed-time controller in queue lengths, delays and waiting times for all penetration rates up until simulation steps 3000-4000. After these time steps, the agents trained under penetration rates of 10%, 20%, 30%, 40% and 70% no longer outperform the fixed-time controller, but agents trained under 50%, 60%, 80%, 90% and 100% do. This is related to the above-described problem that certain vehicles cannot cross the intersection, which leads to ever-increasing delays and never-shortening queues. Based on this it can be concluded that recurrent agents trained under higher

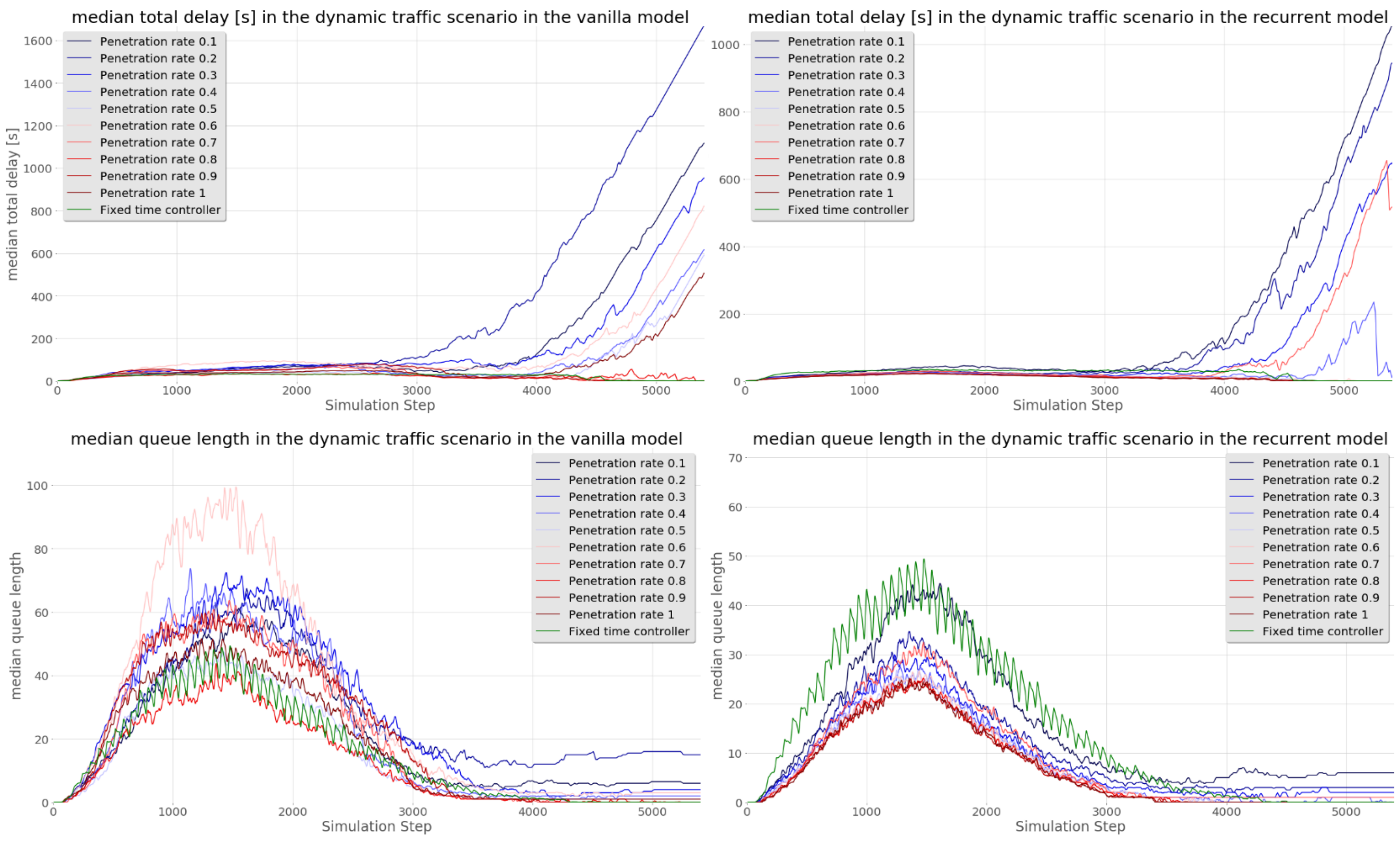


Figure 47 Median cumulative delay and queue length in the dynamic traffic scenario for all penetration rates. Left figure: Vanilla DQN agent. Right figure: Recurrent DQN agent.

penetration rates are able to control dynamic traffic situations efficiently. For lower penetration rates however, recurrent agents can only handle dynamic traffic situations at moments when there are many vehicles in the simulation (e.g. at the rush-hour peak, during the decline of rush-hour traffic), but not at moments when there are only a few vehicles (e.g. during the off-peak at the end of the scenario). Hypothesis 7 is thus supported by the higher penetration agents, but not for the lower penetration agents.

Figure 49 shows the performance of the agents in terms of the averages of the median cumulative delays and the median queue lengths. Representing the performance of the full episode as a single aggregated value can be misleading since some of the agents perform well in most of the episode, but bad in the last part of the episode. Due to this, care must be taken when interpreting the averaged values, particularly when comparing the performance to the fixed-time controller. Nevertheless, Figure 49 can provide useful information. It shows that for the recurrent agent, increasing the penetration rate leads to lower delays and queue lengths (with the 70% agent being an outlier), thus supporting hypothesis 2.

For the vanilla agents, the results are less clear. An interesting thing to note for the vanilla agent is that the plots for the average median delay and the average median queue lengths (Figure 49) do not correlate as well as in the other scenarios. See for example the vanilla agent trained under 70% CV-penetration. This agent achieves a low average median delay (comparable to the delay of the 80% agent), but only a

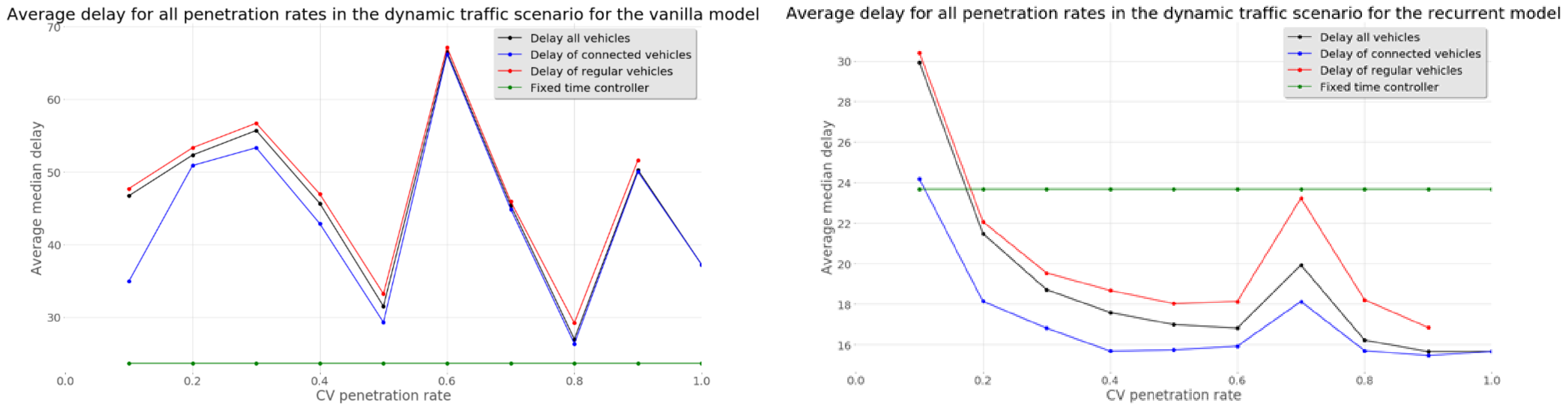


Figure 48 Average median cumulative delay for all vehicles, for connected vehicles and for regular vehicles in the dynamic traffic scenario (note: averages only include simulation steps 0 – 2800). Left figure: Vanilla DQN agent. Right figure: Recurrent DQN agent.

mediocre average median queue length (much worse than the 80% agent). Overall, the two plots show too different trends with too many outliers to be able to support hypothesis 2 for the vanilla agent.

When comparing the vanilla agents' performance to the fixed-time controller, Figure 47 shows that only the agent trained under 80% CV-penetration can outperform the fixed-time controller in terms of queue lengths. With respect to delays, none of the vanilla agents can outperform the fixed-time controller during the full episode, although the 80%-agent is able to reach similarly good performance levels on average. It performs equally well during the first 2800 simulation steps, much better during simulation steps 2800-4400 and much worse during simulation steps 4400-5400. Overall, however, the vanilla DQN agents are unable to control the dynamic traffic scenario satisfactorily, thus rejecting hypothesis 7.

Lastly, the difference between connected and regular vehicles can be investigated (see Figure 48). Like for Figure 49, looking only at the averaged values can be misleading, since the last third of the simulation steps greatly increases the average delays. Due to this, in Figure 48 only averages over the first 2800 simulation steps. The plots show that indeed CVs have lower delays than RVs, supporting hypothesis 3.

Furthermore, hypothesis 4 (higher penetration rates lead to a decreasing gap between CV and RV delays) can be supported for both model types. For the vanilla model, the difference between CVs and RVs is

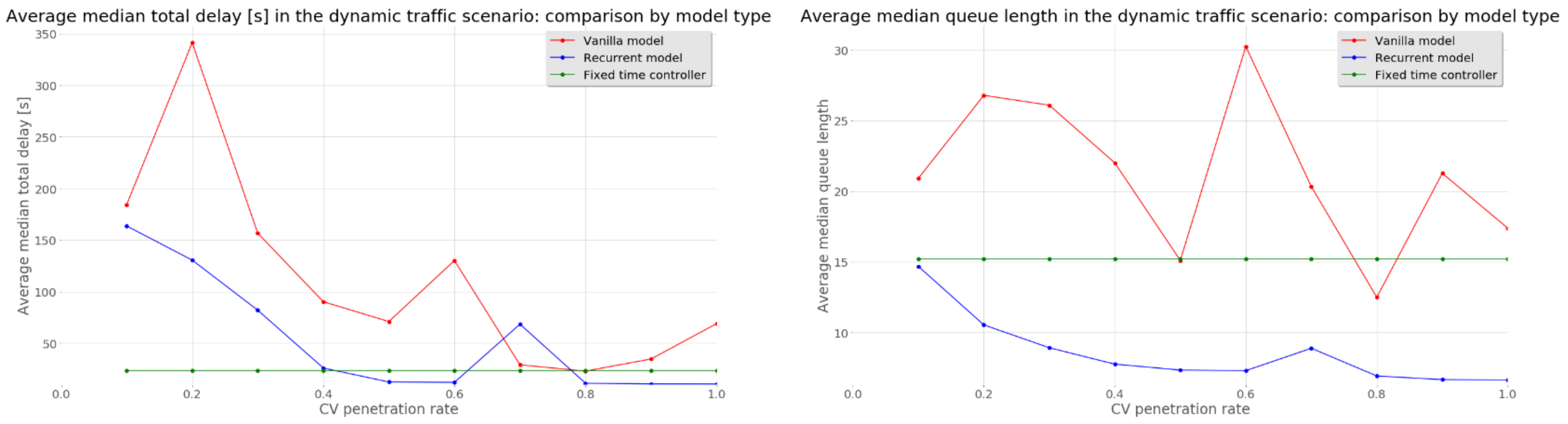


Figure 49 Comparison between the vanilla and recurrent agents: average median cumulative delay and average median queue length in the dynamic traffic scenario.

initially relatively high, but this difference rapidly gets smaller with increasing penetration rates. For penetration rates of 60% and higher, the difference between CVs and RVs has nearly disappeared, meaning CVs no longer have any advantages over RVs. For the recurrent agent, the difference between CVs and RVs for lower penetration rates is less high than for the vanilla agent. Also, for this agent, at higher penetration rates the differences between CVs and RVs are minimal.

8.3.3 Analysis of results per KPI

Lastly, we can discuss the results of all agents in every scenario per KPI. Figure 50 shows the results for the waiting times and queue lengths.

For both the vanilla and the recurrent agent, the curves of the four scenarios follow the same shapes. This means that good (or bad) performance in one scenario is correlated to good (or bad) performance in another scenario. This supports hypothesis 6.

Furthermore, we can compare the performance across scenarios. For the recurrent agent, in all four scenarios increasing penetration rates lead to better performance (with the 70% penetration agent being the outlier). For the medium and high traffic scenarios, even CV-penetration rates as low as 10% leads to nearly equally good delays as higher penetration rates. For the low and dynamic traffic scenarios, agents trained under low penetration rates (10%-30%) performed badly, but agents trained under high penetration rates performed very well. From this it can be concluded that recurrent DQN agents are able to control traffic signals such that congestion is reduced, even under relatively low CV-penetration rates.

The same conclusion can however not be drawn for the vanilla agent. Higher penetration rates do not necessarily lead to better performance. See for example the medium traffic scenario. The agent trained under the lowest penetration rate is one of the best-performing agents, while the agent trained under

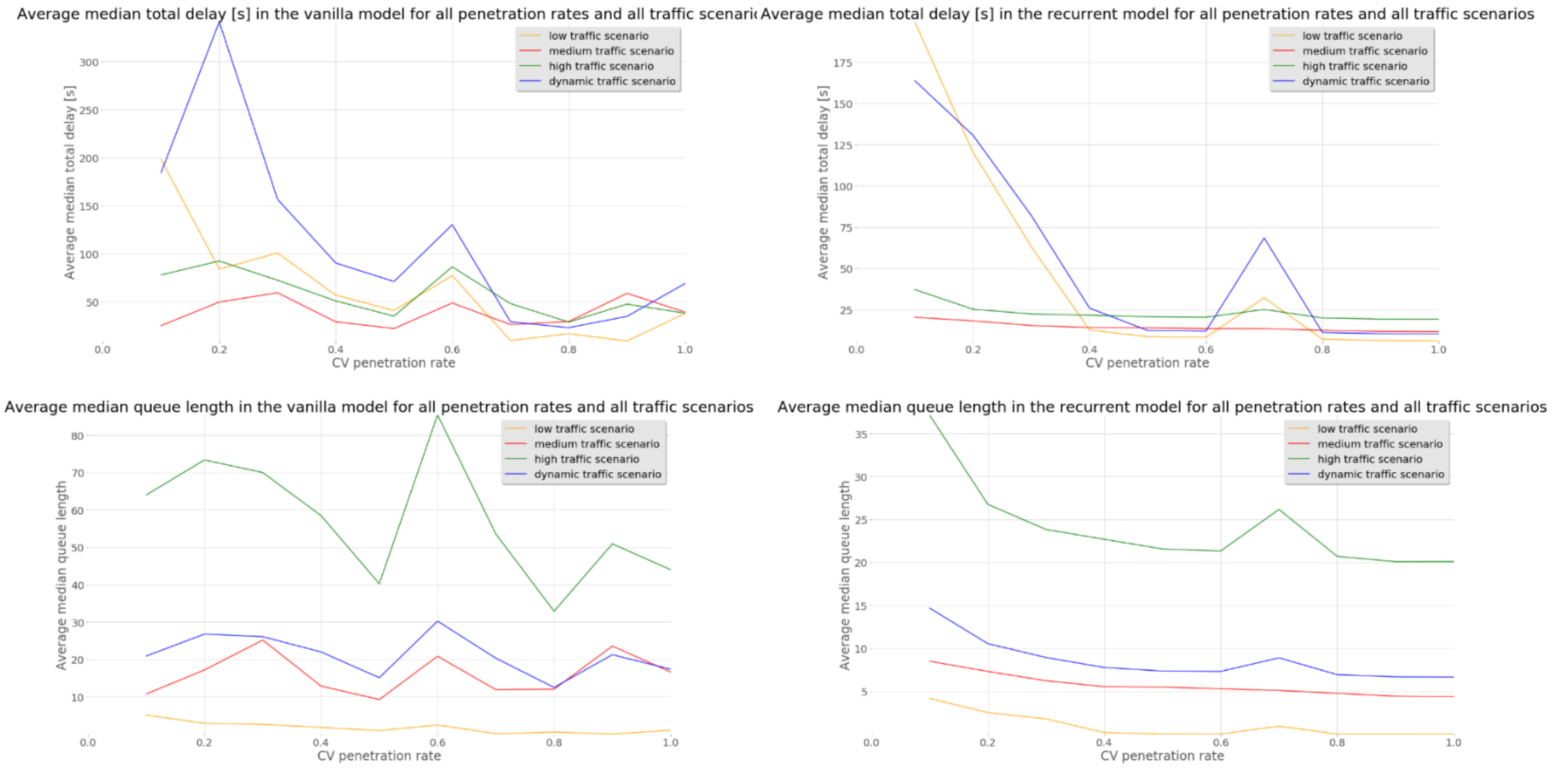


Figure 50 Average median queue length for all penetration rates and scenarios per model types. Left figure: Vanilla agent. Right figure: Recurrent agent.

the 90% penetration rate is one of the worst-performing agents. Why this is the case is unclear. A possible explanation may be that vanilla agents get stuck in local optima, and thus cannot learn optimal policies. Overall, the vanilla DQN agent is unable to control traffic signals efficiently.

8.3.4 Analysis of agent robustness

In the previous experiments, each of the agents was tested on the same CV-penetration rates as they were trained on. However, this requires that the CV-penetration rate is known beforehand and that this penetration rate does not change over time. This assumption is unrealistic for real-world applications, since at different times of the day different CV-penetration rates may be present.

To evaluate how robust agents are to changes in penetration rates, it was investigated how agents that are trained under a certain penetration rate will perform when tested under a different penetration rate. Since testing the robustness of each agent would be too time-consuming, only the agents trained under a penetration rate of 50% were tested. This allows us to evaluate how the agents perform in lower as well as in higher penetration rates. In the experiments, both agents were tested on all four scenarios and under penetration rates between 10% and 100%. The results for the median cumulative delays are presented in Figure 51. For the results for the waiting times and queue lengths, refer to Appendix B.3.

The figure shows that in all four scenarios, the recurrent performs equally well or better when tested on penetration rates higher than 50%. For the vanilla agent this is also the case for the low, high and dynamic traffic scenarios. This indicates that both agents trained under 50% CV-penetration are robust to increases in CV-penetration rates. The only exception to this conclusion is the vanilla agent in the medium traffic scenario. Why this is the case is unclear.

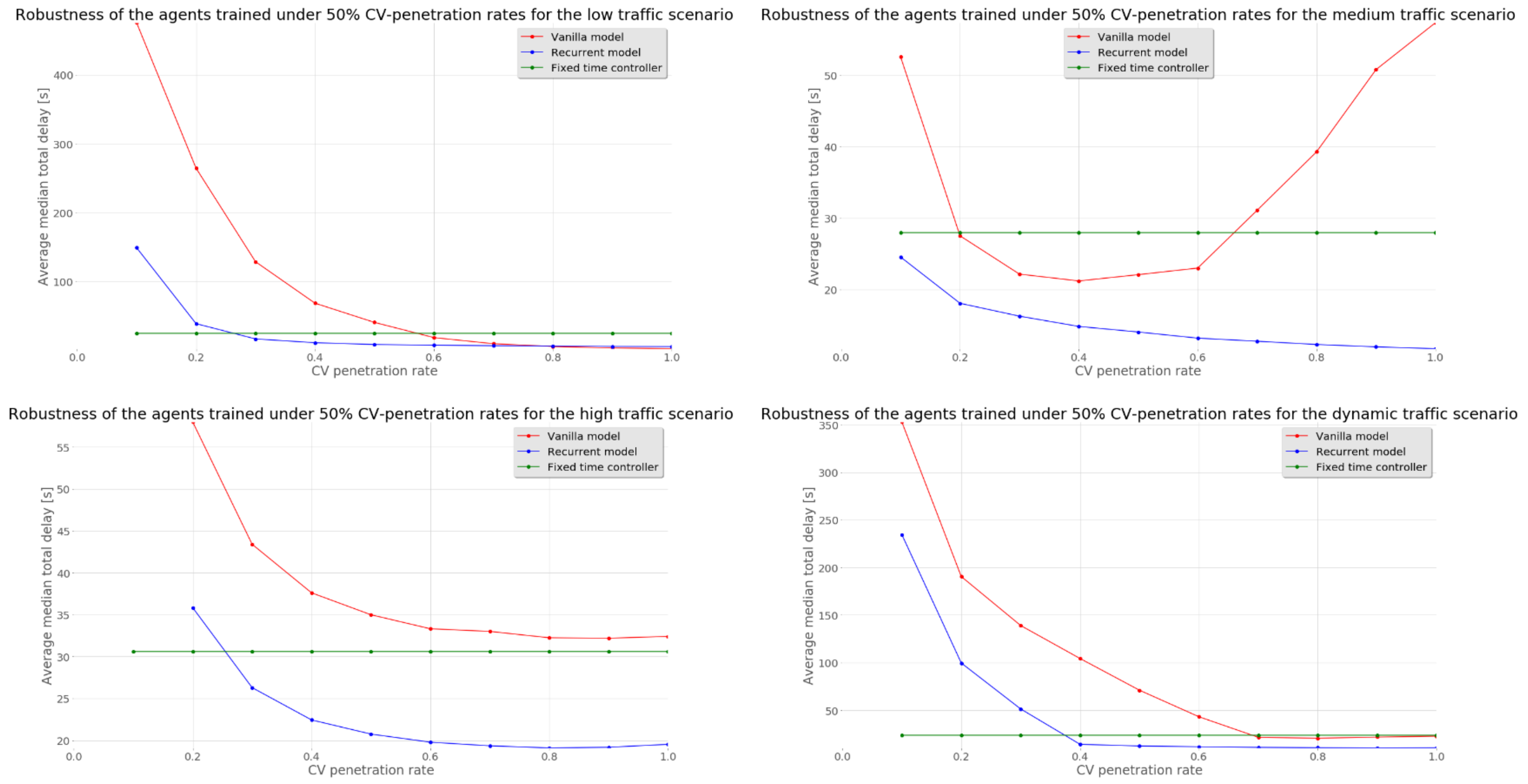


Figure 51 Results of the robustness experiments for all four scenarios. Within the experiments, the vanilla and recurrent agents trained under a penetration rate of 50% were tested on penetration rates between 10%-100%. Top left figure: low traffic scenario. Top right figure: medium traffic scenario. Bottom left figure: high traffic scenario. Bottom right figure: dynamic traffic scenario.

Looking at the results for penetration rates lower than 50%, it can be seen that for both model types performance decreases with lower penetration rates. This is expected since the agents can observe fewer vehicles and thus have less information to decide on the next action. However, it can also be seen that the recurrent agent is more robust to lower penetration rates than the vanilla agent. See the low traffic scenario for example: the recurrent agent's performance remains relatively constant up to a penetration rate between 20%-30%. For the vanilla agent however, the performance already significantly worsens for a penetration rate of 40%. Similar observations can be made for the medium and dynamic scenarios.

Based on these results it can be concluded that both the vanilla and the recurrent agent are robust to increases in penetration rates during testing, but that only the recurrent agent is robust to decreases in penetration rates. For the real-world, this implies that even if agents are trained under a different penetration rate than actually encountered in the traffic situation, the recurrent agent can still be expected to perform similarly well, as long as the real penetration rate is not too drastically lower.

8.4 Summary

This chapter described the experimental setup and experimental results. Experiments were separated into two parts: the preliminary experiments to fine-tune the agent's settings and the actual experiments to evaluate the agent's performance under different CV-penetration rates.

In the preliminary experiments, systematic tests were conducted to determine how different state representations, rewards, convolutional network architectures, combinations of numbers of episodes and epochs, target network freeze intervals, memory sizes, and trajectory sequence lengths impact agents' stability and performance. It was found that using a 2-layered DTSE with floating-point encoded vehicle density and +/- encoded green phase leads to the most stable performance. Furthermore, it was found that the most suitable reward function is the change in cumulative delay.

The preliminary experiments showed that it is difficult to design a well-performing agent. This is partly because it is difficult to judge if instability of the agent is caused by under- or overfitting, and partly because there are too many different settings and parameters that could influence the final performance to exhaustively test them all. More research on how settings influence the agent's performance is needed.

For the final experiments, the best performing parameters were combined into two agents: one vanilla DQN agent and one DRQN agent. Both agents are identical, apart from the addition of an LSTM layer to the recurrent agent. The two fine-tuned agents were trained on CV-penetration rates between 10% and 100% and tested on 50 runs under the same penetration rate. During the evaluation, first the stability of the agents was analyzed, then the performance was analyzed per traffic scenario and finally the performance was compared between traffic scenarios and model types.

The results show that recurrent agents lead to more stable and better-performing agents than vanilla agents. Vanilla agents were generally not able to learn policies that can outperform fixed-time controllers, while recurrent agents were shown to be able to outperform fixed-time controllers even under penetration rates as low as 10%-40%. Furthermore, it was found that for recurrent agents, increasing the CV-penetration rate leads to increases in agent stability and performance. Overall, the recurrent agent was able to learn policies that efficiently control traffic signals such that congestion is reduced for all four traffic scenarios.

9. Discussion

In this chapter, the results of the thesis will be discussed. First, the hypotheses from chapter 8.1.3 are evaluated. Then, the limitations of the model will be discussed, and possible model improvements suggested. Lastly, the results will be embedded in the existing literature.

9.1 Evaluation of hypotheses

To wrap up the analysis of our agents, the evaluation of our hypotheses described in chapter 8.1.3 is summarized here.

(1) **Increases in penetration rates lead to increases in agent stability.**
   a. ***Vanilla agent:*** While the agent stability plots (see Figure 37 and Figure 59 in Appendix B.1) do seem to show a downwards trend towards more stability for higher penetration rates, the plots have too many jumps to draw clear conclusions. Thus, this hypothesis is not supported for the vanilla agent.
   b. ***Recurrent agent:*** For the recurrent agent, this hypothesis is supported. The stability plots show clearly that for all but one agent, higher penetration rates lead to higher stability. The biggest improvements in stability are made in the first 40% of CV-penetration, but stability continues to improve even for higher penetration rates.

(2) **Increases in penetration rates lead to increases in agent performance.**
   a. **Vanilla agent:** For the vanilla agent, this hypothesis is unsupported. While downward trends in the low (Figure 40) and dynamic scenarios (Figure 49) seem present, the plots have too many jumps to conclusively confirm the hypothesis. For the medium and high traffic scenarios (see Figure 43 Figure 46), this hypothesis is rejected. In the medium scenario for example, the agent trained under 10% performs better than any of the agents trained under high penetration rates (70% and more).
   b. **Recurrent agent:** For the recurrent agent however, the hypothesis is supported. Looking at the plots (e.g. Figure 40, Figure 43, Figure 46, Figure 49) shows that in all scenarios agent performance improves with increasing penetration rates (apart from the outlier at 70% CV-penetration, which can be explained by the instability of this agent). Dependent on the scenario, the biggest performance improvements are made between the first 10% to 40% of CV-penetration.

(3) **Connected vehicles have lower delays and waiting times than regular vehicles.** This hypothesis is supported for both the vanilla and the recurrent agents. CV delays are at all times lower than RV delays, which shows that DQN and DRQN agents can reduce delays and waiting times more efficiently for CVs.

(4) **Increases in penetration rates lead to decreasing differences in delays and waiting times between CVs and RVs.** The results of both the vanilla and recurrent agent support this hypothesis. In all scenarios, the differences between CVs and RVs are higher for lower penetration rates than for higher penetration rates. The biggest improvements in RV delays can be seen in the low traffic scenarios. For both agents, the RV delay under 10% CV-penetration is very large (200 + seconds), but drastically decreases with even slightly higher penetration rates (e.g. 15 seconds delay for the 40% recurrent agent).

(5) **Increases in the number of cars in a traffic scenario lead to decreasing differences in delays and waiting times between CVs and RVs.** Both the vanilla and recurrent agent results support this. In

both cases the difference in delays between vehicle types is very large (up to 205 seconds difference) for the low traffic scenario, while for the high traffic scenario this difference is much smaller (maximum of 12 seconds difference). The medium traffic scenario has differences in delays slightly higher than the high traffic scenario. Since the dynamic traffic scenario is a special case which includes low, medium and high traffic situations, this hypothesis cannot be evaluated for this scenario. From these results it can be concluded that particularly agents trained under low penetration rates in the low traffic scenario CVs have a large advantage compared to RVs. For agents trained under higher penetration rates or in scenarios with more vehicles, this advantage becomes much smaller.

(6) **Good performance in one scenario is correlated with good performance in another scenario.** This hypothesis is supported for both agents. For both agents, agents that perform well in one scenario also perform well in the other scenarios. Vice versa, agents that perform badly in a scenario also perform badly in other scenarios (see e.g. the recurrent agent trained under 70% CV-penetration). It does not seem to be the case that agents solely focus on performing well in one scenario and "sacrifice" performance in other scenarios. Instead, agents seem to learn policies which work equally well/poorly in all of the scenarios.

(7) **Reinforcement learning agents can outperform fixed-time controllers.**

   a. **Vanilla agent:** For the vanilla agent, the results for this hypothesis are mixed. For each of the scenarios, some of the vanilla agents can outperform the fixed-time controller in at least one of the KPI. For the low traffic scenario for example, three agents trained under high penetration rates (70%, 80%, 90%) can outperform the fixed-time controller in terms of delay, waiting times and queue lengths. For the medium traffic scenario, the 10%, 50%, 70% and 80% agents can outperform the fixed-time controller in terms of waiting times and delays. For the high and dynamic scenarios, only the 80% agent can reach a better performance than the fixed-time controller. However, by far the majority of the agents is unable to learn policies that can control traffic signals more efficiently than the fixed-time controller. Due to this, the hypothesis is rejected for the vanilla agent. It may be possible that vanilla agents with different settings may lead to better results which could outperform the fixed-time controller, however for this experiment the vanilla DQN agent is not a suitable method to reduce congestion.

   b. **Recurrent agent:** The recurrent agent on the other hand performs much better than the vanilla agent and is able to outperform the fixed-time controller in all scenarios even under partial observability. For the low traffic scenario, agents trained under penetration rates of 40% or higher[9] manage to learn policies that result in better delays, waiting times and queue lengths than for the fixed-time controller. A similar result can be seen for the dynamic scenario agent: agents trained under penetration rates of 40% and higher outperform the fixed-time controller in terms of delays and waiting times, and all agents outperform the fixed-time controller in terms of queue lengths. The results for the medium and high traffic scenarios are even better: in the medium traffic scenario, agents trained under any penetration rate outperform the fixed-time controller, and in the high traffic scenario agents trained under CV-penetrations of 20% and higher outperform the fixed-time controller for all KPI. Based on this it can be concluded that the recurrent DQN

[9] The 70% agent is excluded in this conclusion. The bad results of this agent are likely due to the lower stability of this agent. Why this agent is less stable is unclear.

agent can control traffic signals more efficiently than a fixed-time controller for medium and high traffic scenarios for penetration rates as low as 10-20%, and for the low and dynamic traffic scenarios for penetration rates as low as 40%. This makes recurrent DQN is a suitable method to reduce congestion due to traffic signals, even under low penetration rates.

(8) **Recurrent DQN agents are more stable than vanilla DQN agents.** This hypothesis is supported. Figure 37 and Figure 59 show that the recurrent agent leads to higher queue length, waiting times and delay stability than the vanilla agent. Based on this it can be concluded that when an agent has a memory of past states added to the information about the current state, agent stability improves.

(9) **Recurrent DQN agents perform better than vanilla DQN agents.** This hypothesis is supported. Except for the unstable recurrent agent trained under 70% CV-penetration, all the recurrent agents perform better or equally good as the vanilla agents. In most of the cases, the performance of the two model types is vastly different. Overall, the recurrent agents are thus able to learn much better traffic signal control policies than vanilla agents.

From these results it can be concluded that many of the hypotheses hold for the recurrent agent, but not for the vanilla agent. The recurrent agent shows that indeed, higher CV-penetration rates lead to more stable and better-performing agents. Recurrent agents trained even under low penetration rates can outperform the fixed-time controller, which makes DRQN a suitable method for traffic signal control.

For vanilla agents however, many of the hypotheses cannot be supported. Vanilla agents tend to be less stable and perform much less good than recurrent agents. Even under high penetration rates, vanilla agents are unable to outperform fixed-time controllers. This may be caused by the fact that despite conducting the preliminary experiments, many of the agents were slightly unstable. Potential reasons for this may be overfitting (e.g. due to too long training), underfitting (e.g. due to too short training), learning bad policies (due to too high target network freeze intervals), catastrophic forgetting (e.g. due to too high learning rates or too small memory sizes), unsuitable state or reward representations, an unsuitable neural network architecture, too complex traffic scenarios, etc. It can thus clearly be concluded that designing a well-performing and stable vanilla agent is a difficult task. This is partly due to the difficulty of judging if instability of the agent is caused by under- or overfitting, and partly due to too many different settings and parameters that could influence the final performance to exhaustively test all of them.

Designing a well-performing recurrent agent however is much easier. Even within the limited amount of time of an MSc project, the designed agents were able to reach a better performance than fixed-time controllers, even under low penetration rates. This makes recurrent DQN agents a promising method for future research projects on mixed traffic signal control.

9.2 Limitations

The conducted research has several limitations that have to be taken into account when evaluating the results. This project is subject to many of the same limitations as previous research, both regarding traffic signal control research in general (see section 2.3.3) and RL-specific traffic signal control research (see section 5.5). Since modeling intersections is complex, many simplifications had to be made in order to scope down the problem. A comprehensive list of these scoping decisions can be found in chapter 6.1.

For example, simplifications were made which reduce the realism of the scenario. For instance, it was decided to only include regular vehicles and to exclude all other types of traffic participants. The model assumes that human behavior is uniform, that there are no unexpected traffic situations (e.g. no accidents, no traffic law violations) and that sensor inputs and communication are ideal. While these assumptions do reduce the realism of the scenario, these types of situations are less relevant in proof-of-concept research. Since reinforcement learning is a new field, a lot of early experimentation is happening, for which simpler models suffice. Once proposed agents become more mature, the above-mentioned issues will have to be addressed.

Another limitation of this project which is common in a lot of other research is that the model considers the intersection in isolation. If we would connect several such intersections into a network, each intersection would make its own locally optimal decisions, but would not cooperate with its neighbors. These "selfish", local decisions could negatively impact other agents (e.g. block another vehicle, oversaturate another controller's capacity) and lead to negative emergent patterns for the system as a whole (e.g. decreased intersection/network throughput) (Martínez-Díaz et al., 2019). Some researchers have started to investigate this problem (e.g. Chu, Wang, Codeca, & Li, 2020; Gong, Abdel-Aty, Cai, & Rahman, 2019; Hussain, Wang, & Jiahua, 2020; Klöckner & Klose, 2020; Lee, Chung, & Sohn, 2019; Xu et al., 2020; Yin, Wang, & Li, 2020), but none of the studies have jointly studied mixed traffic and network control. Since both network and mixed traffic control are complex problems that have not been extensively researched on their own, for now the concepts should first be studied separately.

This project is also subject to the problem of generalizability. The proposed model has been trained and tested on a specific intersection layout and for a specific set of traffic scenarios. Different network topologies or traffic scenarios may lead to different results, and other traffic signal control methods may be more suitable for the situations. Due to this, it is suggested that when designing a traffic signal control method for a real-life intersection, several control methods should be compared for the specific needs of that intersection. In this project, it was chosen to use a simple 4-way intersection since this is the most researched topology, which makes the gained results easier to compare to past research.

Lastly, there are limitations related to the chosen model type. Deep Q-learning models are suitable for complex problems such as traffic signal control because they are model-free. Yet, this same property is also their biggest disadvantage. Since the model only requires the modeler to specify a state, action and reward representation and specify the model hyper-parameters, it is difficult to understand what is happening inside the model and why an agent learns a certain policy. Model-free models do not allow the modeler to change inner model-workings, except by imposing artificial restrictions (e.g. maximum green times). This makes it not only difficult to validate a reinforcement learning model, but it also makes it difficult to determine why the model is not working optimally.

9.3 Possible model improvements

Since the recurrent model leads to much more promising results than the vanilla agent, it is suggested to only continue working on the recurrent agent. All further suggestions thus only concern the DRQN agent.

In this thesis, the implementation of the first DRQN agent by Hausknecht and Stone (2015) was followed. Yet, other researchers have improved this implementation. In Hausknecht and Stone (2015), the full trajectory is used in training to update the model weights. Both Lample and Chaplot (2017) and Kapturowski et al. (2019) raise the issue that for the first few samples within a trajectory of samples, Q-values are estimated based on an almost non-existing or inaccurate history of states, since the hidden state is reset at the beginning of every trajectory. Using these first few samples to update the network weights may lead to inaccurate results. Instead the authors suggest using the first few samples in a trajectory to build up a more accurate hidden state value and to only use the later samples in a trajectory to update the network weights. Additionally, Kapturowski et al. (2019) also proposed to store the hidden state when saving a sample as part of a trajectory. When initializing the hidden state of a trajectory during training, this stored hidden state can be used rather than setting it to 0. Kapturowski et al. (2019) conclude that combining both strategies leads to significant performance improvements for the Atari-57 and the DMLab-30 benchmarks. Thus, it may be possible that implementing the improvements may also lead to improved results for the DRQN traffic signal controller.

Furthermore, one or more extensions could be added to the model. Many of the Rainbow extensions for example have been shown to lead to more stable and better-performing agents. Particularly prioritized experience replay and dueling networks seem to lead to good results for traffic signal controllers (Fang et al., 2019; Liang et al., 2019; Nawar et al., 2019; Pol, 2016). Up-to-date, no studies were found which built a traffic signal controller that combined both recurrence and one of the rainbow extensions. When looking at the wider deep reinforcement learning community however, several studies can be found which combined LSTMs/recurrent Q-learning with one of the rainbow extensions (e.g. Kapturowski et al., 2019; Schulze & Schulze, 2018). Similar implementations could also be made for traffic signal controllers.

Additionally, even though a well-performing and relatively stable agent has been designed, further fine-tuning could further improve the agent's stability and performance. In this project, several preliminary experiments to fine-tune the agent have been conducted. Yet, fine-tuning one parameter could have influenced the optimal settings for other parameters. Ideally, fine-tuning agents should be an iterative process. Due to time limitations, in this project only a few iterations could be implemented, so, more iterations could potentially further improve the results.

In future preliminary experiments, it is suggested to test different state and the reward representations. Particularly the reward function may hugely impact performance, since this is the metric by which the agent assesses whether an action is good or bad. In this thesis, eight different reward functions were tested, yet none combined more than two sub-rewards. Other research however has had success with more complex reward functions; therefore it is suggested to further experiment with this.

Lastly, in this project only connected vehicles and regular vehicles were compared. Yet, as described in section 2.1.3, it is likely that connected and automated vehicles will merge. Thus, the model could be extended to include CAVs rather than CVs. CVs only communicate their own state (position and speed) with the controller. CAVs however would also be able to inform the controller about neighboring vehicles within their line of sight. As such, they could communicate information on some of the regular vehicles' positions and speeds as well. This would lead to a more observable traffic situation for the controller, and likely to improved performance, particularly for low penetration rates.

9.4 Cross-comparison with literature results

9.4.1 Agent design

This thesis presented several preliminary experiments to fine-tune the agent. These results are difficult to compare with literature, since most studies do not report how they designed their agent. Yet, excluding this information makes it difficult for researchers which are new to the field to develop agents.

Nevertheless, some similar types of studies have been presented by Dijk (2017), Pol (2016) and Vidali (2018). Vidali (2018) tested different values for the discount factor γ, as well as different types of sampling strategies. Pol (2016) tested variations in the network architecture, state representation, learning rate, optimization algorithms, the target network freeze interval, the memory size and compared performance with or without batch normalization, prioritized experience replay and double Q-learning. Dijk (2017) investigated the impact of the trajectory length, the freeze interval and different update methods. In this section, only the experiments which tested the same model variation are discussed.

Pol (2016) found that networks with fewer filters and fewer layers performed slightly better and were slightly more stable than deeper networks with more layers. Nevertheless, she acknowledges that these differences may be caused by the deeper network needing slightly more training time. These results are in line with the results from the preliminary experiment described in section 8.2.3.

Pol (2016) also studied the impact of the target network freeze interval. She found that a freeze interval of 1 or 5 episodes leads to unstable results, while a freeze interval of 3 episodes leads to stable results. Similar to Pol (2016), the experiment in section 8.2.6 found that a too low freeze interval (here: under 20 episodes) leads to unstable agents. Both results thus show that if a too low freeze interval is chosen, instability may occur. A major difference between the two experimental results however are the differences in the acceptable freeze intervals: for Pol (2016), a freeze interval of 3 episodes was chosen, while in this thesis a freeze interval of 46.7 episodes worked best. A possible explanation for this are the differences between the agent's designs and training processes: Pol's agents were trained on longer episodes (10,000 steps instead of 5,400) and only one type of scenario (instead of 4 different ones) and agents could choose fewer actions (2 instead of 4). It may be possible that the higher complexity of the agent from this thesis (bigger action space, more scenarios) required a higher freeze interval to stabilize the agent. Clearly, the choice of the freeze interval depends largely on the other agent design choices. Nevertheless, it remains unknown how a suitable freeze interval can be easily determined.

Furthermore, Pol (2016) also investigated variations in memory size. She found that agents with a low memory size (10,000 samples/1 episode) are unstable, while agents with larger memory sizes (100,000 samples/10 episodes) are most stable. She attributes the low stability in lower memory agents to the problem of catastrophic forgetting. In the experiment in this study (see section 8.2.7), agents with small memory sizes (30,000 samples/44 episodes; 40,000 samples/59 episodes) also result in unstable performance. Like in Pol (2016), the instability in the low memory agent is likely due to instability. In this thesis' experiment, no instability was found in agents with higher memory sizes (up to 80,000 samples/119 episodes), although they did gain slightly lower rewards than the agent with a medium memory size. It may be possible that the agent would become unstable if the memory size would have been increased even more. Nonetheless, it can be concluded that both too high and too low memory sizes result in sub-optimally performing agents. But, like for the freeze interval, it remains unknown how to pick a good memory size a priori.

Dijk (2017) focused specifically on design choices for recurrent agents. The author found that trajectory lengths of 2 and 4 samples are too short to improve agent stability and performance compared to vanilla DQN agents. Trajectory lengths of 10 and 20 samples however led to stable agents that significantly outperformed vanilla agents. Both the 10 and the 20 sample agents resulted in a similar performance. The experiment in this thesis (see section 8.2.8) found that like for Dijk (2017), too short trajectory lengths (9 - 15 samples) result in unstable and under-performing agents. Furthermore, as in the experiment by Dijk

(2017), agents trained on longer trajectory lengths (18+ samples) were stable and led to high rewards. After reaching a certain trajectory length, neither agent stability nor performance improved significantly. In this thesis, this saturation point was reached after a trajectory length of 18 samples, for Dijk (2017) this point was already reached after 10 samples. This difference may be caused by the fact that in Dijk (2017) all vehicles were observable (equivalent to 100% CV-penetration), while in some of the experiments in section 8.2.8 only half of the vehicles were observable (equivalent to 50% CV-penetration). Likely, lower CV-penetration requires longer trajectory lengths in order for the agent to build up an accurate hidden state.

Lastly, the preliminary experiment on state representations can be compared to R. Zhang et al. (2020). They found that using +/- encoded current green phases leads to more stable agents than using one-hot encoded green phases. This result is in line with the experiment results in section 8.2.1.

Nevertheless, despite conducting the preliminary experiments, satisfactory stability and performance were not achieved for all agents. Particularly the vanilla agents remained somewhat unstable and were mostly unable to gain better results than the fixed-time controller. It may be possible that combinations of different parameters perform well on their own, but not in combination with other settings. Similar problems with instability were reported by other Master thesis students (Pol, 2016; Samad, 2020; Vidali, 2018), showing that designing good reinforcement learning agents is not as straight-forward as it seems.

9.4.2 Performance in mixed traffic scenarios

As was seen in section 5.4, only a few researchers have used reinforcement learning algorithms for mixed traffic situations. Of the found studies, one paper used tabular Q-learning (Islam et al., 2019), one used tabular Q-learning and shallow neural network Q-learning (Yang et al., 2017), one used deep Q-learning (R. Zhang et al., 2020) and two used deep Q-learning and deep recurrent Q-earning with an LSTM layer (Zeng et al., 2018; T. Zhao & Wang, 2019). Since the study by Yang et al. (2017) resulted in too unstable controllers, no conclusions regarding the impact of penetration rates were drawn, and as such their study will be excluded from the further discussion.

All four of the other papers found that in non-recurrent agents, higher penetration rates led to higher performing agents. In the study by Islam et al. (2019), the major performance improvements happened between 20% and 50%. When looking at the vanilla DQN agents, both Zeng et al. (2018) and T. Zhao and Wang (2019) find that performance majorly improves with increasing penetration rates between 10% and 50% and then plateaus. R. Zhang et al. (2020) analyzed performance under different CV-penetration rates in different traffic scenarios and found that with increasing penetration rates, performance does not improve in the high traffic scenario, but that it does continuously improve between 0% and 100% penetration rate in the medium and low traffic scenarios (with 80% of the performance improvements happening in the first 20%).

The results in this thesis are not in line with these previous findings. In this thesis' experiments, higher CV-penetration rates did not necessarily lead to better performance when using the vanilla DQN agent (see Figure 49 and Figure 64 in Appendix B.3). A possible explanation for this is that vanilla agents were unstable to a certain degree which could have caused lower performance. This instability may be due to agents not being fully fine-tuned, despite conducting the preliminary experiments.

Furthermore, the results of the recurrent agent experiments can be compared to literature. Both Zeng et al. (2018) and T. Zhao and Wang (2019) trained recurrent DQN agents and found that the critical penetration rate can be as low as 10%. For higher penetration rates, no major performance improvements were observed. They also find that using recurrent DQN outperforms vanilla DQN agents for all penetration rates and all traffic scenarios, and that recurrent agents are less sensitive in differences between the trained and tested penetration rates.

These results are partially comparable to the results in this thesis. Like in Zeng et al. (2018) and T. Zhao and Wang (2019), recurrent DQN was shown to be able to outperform fixed-time traffic signal controllers in all traffic scenarios. However, unlike in Zeng et al. (2018) and T. Zhao and Wang (2019), agent performance in our experiments did not plateau after 10% CV-penetration, but instead plateaued after around 40% CV-penetration. This result indicates that it is not possible to specify a single critical penetration rate up until which the RL agent keeps improving. Instead, like Islam et al. (2019) concluded, this critical penetration rate depends largely on the design of the specific agent and the traffic scenarios.

Another aspect that can be compared to literature are the differences between traffic scenarios. In literature, only T. Zhao and Wang (2019) and R. Zhang et al. (2020) tested their controllers under different penetration rates. Since the study by T. Zhao and Wang (2019) does not report how agents' performance changes for different penetration rates in the different traffic scenarios, this study cannot be taken further into account. R. Zhang et al. (2020) however extensively discuss these results. They found that for the low and medium traffic scenarios, agent performance improved significantly for increasing penetration rates, but that for high traffic scenarios, agent performance did not improve for increasing penetration rates. Their result implies that agents in high traffic scenarios are able to perform equally well under low and high penetration rates. In this thesis however, this conclusion is unsupported. For the recurrent agent, performance in all traffic scenarios increased with higher penetration rates. This suggests that even high traffic scenarios can profit from increasing CV-penetration rates.

Lastly, the differences between CV and RVs can be evaluated, which was only studied by R. Zhang et al. (2020). They found that CVs have lower waiting times than RVs, but that this advantage of CVs over RVs decreases the more cars there are in the traffic scenario. This result is also supported by this thesis.

To conclude, some results of this thesis are similar to previous studies' results, while others are contradicting. Unlike in literature, vanilla agents were unable to outperform fixed-time agents and higher penetration rates did not lead to increased performance. For recurrent agents however, these results are clearly supported.

Yet, each of the studies has shortcomings that may limit their results. Islam et al. (2019) only implemented a tabular Q-learning controller, which is limited in its state representation and led to relatively unstable performance. Zeng et al. (2018) implemented a cyclic controller which is only tested only on a single constant traffic scenario. T. Zhao and Wang (2019) also train their controller on constant traffic scenarios, but they improve on Zeng et al. (2018) by including three different levels of traffic demand (higher saturated, near-saturated and low-saturated traffic). However, they do not explicitly discuss how agent performance is affected for the different levels of traffic demand. Lastly, R. Zhang et al. (2020) present the most comprehensive research. They included sparse, medium-dense and dense traffic scenarios and evaluated agent performance separately for each of these scenarios, and they also discussed how this affects CV and RV waiting times. However, unlike this thesis they did not build a recurrent agent. A shortcoming of all previous studies was that none trained or tested their controller under dynamic traffic scenarios, as this thesis has attempted to do.

10. CONCLUSION AND RECOMMENDATIONS

This chapter concludes the thesis. First, the results will be summarized, and the research questions will be answered. Next, the scientific contributions are highlighted. Lastly, recommendations for policy-makers and future research are made.

10.1 Summary

In this thesis, the research goal was to investigate whether reinforcement learning-based algorithms would be a suitable method to control signalized intersections in mixed traffic scenarios such that traffic congestion is minimized for both connected and regular vehicles. To scope down the topic, only the most commonly used reinforcement learning method was under investigation: deep Q-learning.

To answer the research questions, first a literature review on deep Q-learning in TSC has been conducted. Based on this literature, two deep Q-learning agents were designed: one vanilla DQN agent and one recurrent DQN agent. To calibrate the two agents, systematic fine-tuning experiments were conducted in which several design choices were compared. The two finalized agents were then trained and tested on different traffic scenarios and under different CV-penetration rates.

The main research question for this thesis was:

> ***Can deep Q-learning models be used to control signalized intersections in mixed traffic scenarios such that traffic congestion is reduced?***

The main research question was subdivided into six sub-questions, which were defined in chapter 3.2. This section will summarize the findings by answering each of the sub-research questions, before concluding by answering the main research question.

RQ1 What is the current state-of-the-art in intelligent traffic signal control using deep Q-learning for both homogeneous and mixed traffic scenarios?

Reinforcement learning is a fast-developing field in which new algorithms are being proposed continuously. When looking at reinforcement learning in traffic signal control, the most popular reinforcement learning method is deep Q-learning (DQN). Within DQN models, authors have been experimenting with different model extensions, agent representations (including state, action and reward representations) and traffic environments (including network topologies and traffic scenarios). Many of the newly proposed controllers have been shown to outperform traditional control methods.

The literature review in this thesis has identified, analyzed and discussed how previous researchers designed their agents. The review discussed each of the most important design choices in agent design. Included topics were state, action and reward representations, network topologies, traffic generation, DQN extensions, performance indicators and base cases.

Furthermore, part of the review specifically focused on reinforcement learning in mixed traffic scenarios. Much less research has been done on this topic. For mixed traffic situations, two types of agents were found in literature: vanilla DQN agents and recurrent DQN agents which use an LSTM layer. The results show that even for low CV-penetration rates, both model types can successfully reduce traffic congestion compared to traditional traffic signal controllers.

Yet research gaps remain. Currently, no clear best practices and guidelines exist that could guide researchers on how to design new agents. Many studies do not or only partially describe why they made certain design choices and how they calibrated their agents, which makes it difficult for future researchers to learn from their experiences.

Another research gap that contributes to this problem is the lack of systematic studies that compare different agents. This makes it more difficult to evaluate which design choices lead to better results. This lack of comparative studies may partly be caused by the lack of standardized testing scenarios. Without standardized scenarios, it is difficult to establish best practices and benchmarks.

RQ2 What types of deep Q-learning models would be suitable to design an intelligent traffic signal controller for mixed traffic scenarios?

In this thesis, it was decided to design a vanilla DQN agent and a recurrent DQN agent, because in previous studies both agents have been successfully used to control mixed traffic situations.

Within this thesis, the two agents have the same settings, the only difference being the recurrent agent having an LSTM layer while the vanilla agent does not. The agents' state representation consists of a 2-layered DTSE matrix of the vehicle density and average normalized speed, the current green phase, and the elapsed green time. The set of actions consists of the four green phases which the agent can choose acyclically. The finally chosen reward function was the change in cumulative waiting time. To stabilize training, experience replay and target network freezing have been used. Since for many parameters no clear guidelines existed, agents were calibrated by means of several fine-tuning experiments (see RQ 3).

RQ3 How can the reinforcement learning-based traffic signal controllers be calibrated and trained in a systematic manner?

One of the identified research gaps was that it was unclear what the best design choices and parameters for DQN agents were. Many previous studies either did not report their fine-tuning process, or they applied a trial-and-error process. In this thesis, one contribution was to present a systematic and comprehensive calibration process, in which the impacts of different design choices on the agents' performance and stability have been analyzed.

The calibration process in this thesis attempted to apply a more structured approach. First, a base agent was designed which combined design choices from previous studies in novel ways. Yet, since the agents would be tested on different types of scenarios than in those previous studies, it would not be guaranteed that these choices would be optimal.

To solve this, each of the design choices and parameters which we were unsure about were investigated in experiments. Each parameter was investigated by using a new base model. In this base model, all settings were identical, apart from the parameter under investigation. This allowed to assess how the different alternatives for every parameter impacted both agent stability and performance. To compare the agents tested under different parameter settings, the cumulative reward per episode during training was used. Stable agents show no jumps in rewards and agents which perform well lead to a very low negative reward. Finally, the parameter setting which resulted in the best stability and highest performance was chosen. Based on this result, a new base agent could be designed to experiment with the next parameter.

In the end, the best-performing settings for every design choice were combined into a fine-tuned vanilla and a fine-tuned recurrent agent. Ideally, each of the parameters should have been investigated several times, since the parameters which are chosen at later times may influence whether the parameters chosen at earlier times are still acceptable. The goal would thus be to iteratively fine-tune the agents. However, in this thesis this was not possible due to time reasons.

Overall, agents in this thesis were calibrated in a systematic manner. Yet, the results show that the optimal design choices and parameter values vary a lot between different agents. The optimal settings not only depend on the other design choices which were made for the agent, but also on the traffic scenarios on which agents will be tested and trained. This makes it difficult to derive more general best-performing settings. In the end, fine-tuning the agents remained a time-consuming trial-and-error process.

RQ4 How can the designed deep Q-learning-based traffic signal controllers be evaluated in order to determine which controller performs better under which circumstances?

In this thesis, vanilla and recurrent agents were trained under a penetration rate between 10%-100% and tested under this same penetration rate. Testing took place under the same types of scenarios as agents had been trained on (low, medium and high constant traffic scenarios and dynamic traffic scenarios). Traffic scenarios were chosen to reflect different levels of traffic demand (low/undersaturated, medium/medium-saturated, high/oversaturated) and traffic distributions (constant, dynamic). Ideally, traffic scenarios should reflect real-life scenarios as closely as possible, since controllers can only perform well for situations on which they have been trained. Since the controller in this thesis was designed as a proof-of-concept, no real-life traffic data has been used. Instead, scenarios were designed such that agent stability and performance could be evaluated under as different situations as possible.

To train and test DQN agents, a microscopic traffic simulator was used. Because scenarios are stochastically generated, controllers have been tested on every scenario for 50 runs. At every simulation step the following KPI were gathered: the average cumulative delay per vehicle and vehicle type (i.e. per CV and RV), the average cumulative waiting time per vehicle and the average queue length. Furthermore, to compare the agents' performance, a traditional fixed-time controller was used as a benchmark.

To evaluate the gathered data, different analyses have been done. In the first analysis, the stability of the agents was evaluated based on the interdecile range. In the next step, the performance of the agents was evaluated per traffic scenario. Lastly, the robustness of the agents was evaluated. For this, the vanilla and recurrent agents trained under CV-penetration rates of 50% were tested on all penetration rates between 10%-100%. This allowed investigating how well the agents can perform when the penetration rate in real-life does not match the penetration rate seen during training.

RQ5 To what extent are the designed deep Q-learning-based traffic signal controllers able to reduce traffic congestion for mixed traffic situations in a robust manner?

In the evaluation of the experiments, it was found that the vanilla agent which was designed in this thesis was unable to reduce traffic congestion in mixed traffic scenarios. Many of the hypotheses which were described in section 8.1.3 were unsupported: neither agent stability nor agent performance increased with higher penetration rates and vanilla agents were unable to outperform fixed-time controllers.

Recurrent agents however performed very well, with all hypotheses from section 8.1.3 being supported. They were quite stable and performed well even under low penetration rates. Yet, the experiments also showed that higher penetration rates led to more stability and higher performance across all traffic scenarios. Additionally, the recurrent agent was found to be quite robust to changes in the penetration rate during testing: the recurrent agent trained under 50% CV-penetration was able to perform well for CV-penetration rates between 30%-100%. Furthermore, the recurrent agent was able to outperform the fixed-time controller for most of the penetration rates and scenarios. The required CV-penetration rate to outperform the fixed-time controller was between 0% (medium traffic scenario) and 40% (low and dynamic scenarios). This indicates that while the required CV-penetration rate is relatively low across scenarios, no critical transition penetration rate can be determined, since this largely depends on the type of scenario. When comparing the scenarios, it was found that for scenarios with less cars, the recurrent agent required a higher CV-penetration rate to reach a good level of performance than for scenarios with many cars. Moreover, the recurrent agent was able to perform better in constant traffic scenarios than in dynamic traffic scenarios.

Based on the results it can be concluded that the designed vanilla DQN agent is unsuitable for mixed traffic control. The recurrent DQN agent however, performed better than the vanilla agent in terms of stability, performance and robustness, and only the recurrent agent was able to outperform the fixed-time controller for all but the lowest penetration rates. This makes recurrent DQN a promising method for future research on reinforcement learning-based traffic signal control.

RQ6 How do the results compare to other literature results?

Section 9.4 compared the both the results of the calibration experiments and of the final mixed traffic experiments to results in literature.

When comparing the results of the calibration experiments, it was concluded that the optimal design choices and parameters used in literature are oftentimes different than the choices which were found to work best for our agents. This indicates that optimal parameters for an agent depend to a large extent on not only the overall agent design but also on the training and testing environment (e.g. traffic scenarios, intersection layout). Nevertheless, despite the finally chosen values being different, the assessments of how certain parameters affect model stability and performance are mostly similar. See the results for the target network freeze interval for example. Despite the optimal values being different, both the results of this thesis and the study by Pol (2016) show that too low freeze intervals can lead to unstable agents.

When comparing the results of the final mixed traffic experiments to literature, not all findings from past research were supported. A major difference between the results was that in past research, vanilla DQN agents have been shown to be able to successfully control traffic signals even for mixed traffic situations, while the agent in our study was unable to do so. Possibly, the vanilla agent is this thesis was not fine-tuned enough. For the recurrent agent however, both this thesis and past research concluded that recurrent agents are able to significantly outperform fixed-time controllers, even under low penetration rates. Further similarities were that higher penetration rates lead to higher performance, and that recurrent agents are up to a certain degree robust to changes in penetration rates. Unlike past research however, this thesis found that for all traffic scenarios agent performance improves with increasing penetration rates. Previous study however found that this was only the case for low and medium traffic scenarios, but not for high traffic scenarios.

MAIN: Can deep Q-learning algorithms be used to control signalized intersections in mixed traffic scenarios such that traffic congestion is reduced?

After having answered each of the sub-research questions, the main research question can be answered. Overall, the results of this thesis show that deep Q-learning algorithms could potentially be used to control signalized intersections in mixed traffic scenarios such that traffic congestion is reduced.

While vanilla DQN was unable to control traffic signals efficiently, recurrent DQN was able to outperform fixed-time controllers even under low CV-penetration rates. This makes recurrent DQN a promising method for future research.

Nevertheless, this thesis only resulted in a proof-of-concept. Experiments were conducted under many assumptions and simplified conditions (see section 6.1) and results are subject to multiple limitations (see section 9.2). All of these may compromise the results. In order to answer the main research question with certainty, recurrent DQN controllers (and reinforcement learning controllers in general), will need to be further developed, calibrated, and tested under more realistic circumstances. Several suggestions for future research steps will be suggested in section 10.5.

10.2 Scientific contributions

The scientific contribution of this thesis can be summarized as follows:

In depth-review on using reinforcement learning in traffic signal control, with a focus on deep Q-learning as well as mixed traffic situations. The only other review which was found on traffic signal control using reinforcement learning was published in 2017, yet most of the papers on TSC using deep reinforcement learning were published after this date. Additionally, no review was dedicated specifically to mixed traffic situations. The review in this thesis (see chapter 5) updated the outdated review and closed the research gap.

Systematic experiments on fine-tuning the agents. In literature, nearly no authors describe why they made certain design choices and how they fine-tuned their models. Additionally, many researchers do not report all the relevant parameters in their models. This not only makes these studies difficult to reproduce, but it also means that it remained known how certain design choices and parameters affect the final agent's stability and performance. For future researchers this means that they cannot use the previous researchers' experiences to calibrate their own models. Instead, they will have to conduct their own fine-tuning process which can be time-consuming. This thesis contributed to closing this research gap by presenting the full fine-tuning process. This calibration process was conducted by systematically experimenting with different design choices (e.g. state representation, reward representation) and parameters (e.g. memory size, target network freeze interval). The results of these experiments can hopefully help future researchers to better understand the effects certain design choices will have, and to cut down on the time to fine-tune their own agents.

Experimental comparison of two model types (DQN and DRQN) regarding their stability and performance under different CV-penetration rates and traffic scenarios. Only a few previous researchers investigated how well reinforcement learning models are able to control traffic signals under mixed traffic situations, and even fewer researchers investigated how two different model types would perform in these situations. This thesis added new experimental insights to this by designing, training and testing two types of deep Q-learning controllers (vanilla DQN and recurrent DQN). Within the experiments, the stability and performance of both agents were analyzed both separately and comparatively for different traffic scenarios (low constant, medium constant, high constant, dynamic) and CV-penetration rates (10%-100%). Compared to previous research, the agents in this thesis were not only evaluated on constant traffic scenarios, but also on dynamic traffic scenarios. Additionally, results were compared to a fixed-time controller. Results of the experiments show that recurrent DQN agents are able to outperform fixed-time agents even under low penetration rates (10%-40%), but also that both agent stability and performance improve for higher penetration rates. Vanilla DQN agents resulted in unsatisfactory performance. Based on this result and the results by Zeng et al. (2018) and T. Zhao and Wang (2019), it can be concluded that recurrent DQN agents are superior to vanilla DQN agents, and that recurrent DQN agents are a promising method for future traffic signal control even under low CV-penetration rates.

10.3 Link to CoSEM

This thesis was carried out within the MSc program Complex Systems Engineering and Management (CoSEM). The research done within this project fits well to CoSEM for the following reasons: CoSEM is about designing solutions within complex socio-technical systems that solve problems that concern both public and private interests. Each of these aspects is included in this thesis.

Intelligent intersections are great examples of complex socio-technical systems. Within intersections, many traffic participants (regular vehicles, connected and/or automated vehicles, cyclists, pedestrians) interact with each other. Every traffic participant acts according to their own behavioral rules such as traffic laws and their own destinations. Altogether, these individual behaviors create emergent traffic patterns that evolve over time and can be potentially unpredictable. This makes designing traffic signal controllers that regulate traffic such that safety is ensured, and such that congestion is minimized a challenging task.

This thesis has a clear design component: within the project, two traffic signal controllers were designed. The designs were based on previously published state-of-the-art DQN agents, but also based on the systematic calibration experiments described in section 8.2. Each of the design choices has been described and justified. Within the design, values from both the private and the public domain have been taken into account: the overall goal was to increase traffic efficiency in order to reduce congestion, yet also safety had to be considered. Additionally, within the design phase it was aimed to use state-of-the-art methods from the field of reinforcement learning. At the same time however, also technical limitations needed to be taken into account (e.g. connected/automated technology limitations, computational limits).

The project itself was multidisciplinary in nature, and multiple engineering components were present. It combined different aspects related to transportation science, civil engineering and computer science/ machine learning.

Within this project CoSEM-related tools and knowledge were used. In the first part of the project, a systematic literature review as taught in the course SEN131A was conducted. Furthermore, this project required domain-specific knowledge from my track transport and logistics, as well as knowledge on simulation and modeling in order to design and evaluate the agents. Lastly, within the project the impacts of the designed controllers on traffic congestion were evaluated quantitatively and the implications of this were discussed.

10.4 Recommendations for practice

The research in this thesis was explorative in nature and resulted in evidence that recurrent DQN agents may be able to efficiently control signalized intersections. Nevertheless, currently it is too early to implement the designed agent in practice. The agent designed in this thesis should be seen as a successful proof-of-concept, rather than as a method that is ready for real-world implementation. This is because the designed agent is subject to many simplifications and limitations (as discussed in section 9.2), but also because agent performance is not yet stable across all traffic scenarios and penetration rates.

However, the results of this thesis show that recurrent DQN (and deep RL in general) may be a promising method that could be applied in the future. Thus, it is recommended that more research will be conducted in this field. Several research recommendations have been made in sections 9.3 and 10.5.

To conclude, it is currently too early for policy-makers to take action, apart from providing sufficient funding. First, controllers need to become more mature and must be validated on more types of scenarios under more realistic circumstances. Only then can exact technical, social and legal implications be known and more specific policy recommendations be made.

10.5 Recommendations for future research

Several recommendations to improve the model built in this thesis have already been discussed in section 9.3. This section will provide further recommendations independently from the agent in this project. Suggestions made relate to the literature gaps in applying reinforcement learning in traffic signal control, as identified in chapter 5.5.

A major limitation in reinforcement learning-based traffic signal control research is the lack of standardized testing scenarios and environments. Currently, each research team implemented its agents in slightly different ways (e.g. different network topologies, different traffic demand generation), which reduces the ability to do cross-comparisons between different studies. If a set of standardized environments and scenarios would exist, researchers could test their proposed algorithms on these. Having unified testing scenarios would facilitate benchmarking and objective comparison of agents, and thus more systematic evaluations. When developing a set of environments, care must be taken that traffic scenarios cover the relevant different types of situations (e.g. under-saturated, saturated, over-saturated traffic; constant and dynamic traffic). The set of scenarios should neither be too small (which would limit researchers), nor be too large (which would hamper cross-comparisons). Regarding the implementation, similar approaches can be taken as for the implementation of the Atari or Doom environments in the OpenAI Gym (OpenAI, 2016).

Additionally, more research is needed on how agents can be fine-tuned. At the moment, no best practices regarding choices in states, rewards, network topologies and hyperparameters exist, making it time-consuming to design new agents. Having clearer guidelines would significantly reduce the time to develop agents, since researchers would not have to conduct as many fine-tuning experiments by themselves. Thus, it is suggested to systematically conduct and document fine-tuning experiments, similarly as has been done in the preliminary experiments in section 8.2.

Another future research suggestion is to further develop existing traffic signal controllers' models. Deep reinforcement learning is a fast-developing field in which authors continuously develop new model types or extensions to existing models. Combining several extensions could lead to significantly increased performance compared to models without extensions or only single extensions (as for example shown in the highly-cited Rainbow paper by Hessel et al. (2018)). But it may also be possible that extensions do not combine well. Additionally, it may be possible that other model types than the most commonly used deep Q-learning model perform better, such as the hybrid actor-critic model. For traffic signal control specifically, only a limited amount of papers systematically compared model types and/or extensions in ablation studies (Fang et al., 2019; Wei et al., 2018; Zeng et al., 2019). Yet, many other newly developed methods exist which may improve performance further, but which have not yet been investigated for traffic signal control. For example, Kapturowski, Ostrovski, Quan, Munos, & Dabney (2019) have created an agent that combined recurrent deep Q-learning with prioritized experience replay and distributed Q-learning, which outperforms existing deep RL agents on the Atari-57 benchmark. Similar studies could be conducted in TSC.

More research is needed on multi-goal agents. Currently, many authors focus solely on traffic efficiency, at the expense of other goals. Yet, safety, fairness, environmental impacts, driver comfort and social acceptance are all goals which also have to be taken into account when designing a new traffic signal controller. For example, if controllers are purely performance-based, they could for example “sacrifice” a single vehicle for the better performance of the system as a whole by making the single vehicle wait very long. Clearly, this is undesirable, which means other goals should also be included. However, in existing controllers, it is difficult to include multiple goals. Partly this is caused by the fact that the algorithms are model-free which makes it not possible to change the inner workings. And partly it is caused due to it being difficult to weigh several goals and combine them into a balanced reward function. This leads to the next research suggestion, namely, to investigate how multiple rewards can be balanced to gain good performance.

Lastly, future research should investigate whether agents will be able to perform well in the real-world under less ideal circumstances than the simulator provides. Most studies currently assume ideal conditions, yet the real-world is more chaotic. For example, unexpected or previously unencountered traffic situations may happen (e.g. accidents, radical demand changes), lane switches and traffic law violations could take place or signal failures, communication delays or errors could happen. These types of issues may compromise the robustness of the agent, and thus adversely affect its ability to reduce traffic congestion.

References

Abdulhai, B., Pringle, R., & Karakoulas, G. J. (2003). Reinforcement learning for true adaptive traffic signal control. *Journal of Transportation Engineering*. https://doi.org/10.1061/(ASCE)0733-947X(2003)129:3(278)

Aghdam, H. H., & Heravi, E. J. (2017). *Guide to convolutional neural networks*. https://doi.org/10.1007/978-3-319-57550-6

Araghi, S., Khosravi, A., & Creighton, D. (2015). A review on computational intelligence methods for controlling traffic signal timing. *Expert Systems with Applications*, *42*(3), 1538–1550. https://doi.org/10.1016/j.eswa.2014.09.003

Artimy, M. (2007). Local density estimation and dynamic transmission-range assignment in vehicular ad hoc networks. *IEEE Transactions on Intelligent Transportation Systems*. https://doi.org/10.1109/TITS.2007.895290

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep Reinforcement Learning: A brief survey. *IEEE Signal Processing Magazine*, *34*(6), 26–38. https://doi.org/10.1007/978-981-13-8285-7

Bagloee, S. A., Tavana, M., Asadi, M., & Oliver, T. (2016). Autonomous vehicles: challenges, opportunities, and future implications for transportation policies. *Journal of Modern Transportation*, *24*(4), 284–303. https://doi.org/10.1007/s40534-016-0117-3

Behrisch, M., Bieker, L., Erdmann, J., & Krajzewicz, D. (2011). SUMO--simulation of urban mobility: an overview. *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*.

Bellemare, M. G., Dabney, W., & Munos, R. (2017). A distributional perspective on reinforcement learning. *34th International Conference on Machine Learning, ICML 2017*.

Bishop, C. M. (1996). Neural networks: a pattern recognition perspective. *Neural Networks*. https://doi.org/10.1.1.46.8742

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai Gym. *ArXiv Preprint ArXiv:1606.01540*.

Brys, T., Pham, T. T., & Taylor, M. E. (2014). Distributed learning and multi-objectivity in traffic light control. *Connection Science*. https://doi.org/10.1080/09540091.2014.885282

Catapult Transport Systems. (2017). *Market Forecast for Connected and Autonomous Vehicles*. Retrieved from https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/642813/15780_TSC_Market_Forecast_for_CAV_Report_FINAL.pdf

Chang, G. L., & Xiang, H. (2003). The relationship between congestion levels and accidents. In *State Highway Adiministration*.

Chang, H. J., & Park, G. T. (2013). A study on traffic signal control at signalized intersections in vehicular ad hoc networks. *Ad Hoc Networks*, *11*(7), 2115–2124. https://doi.org/10.1016/j.adhoc.2012.02.013

Chen, L., & Englund, C. (2016). Cooperative Intersection Management: A Survey. *IEEE Transactions on*

Intelligent Transportation Systems, *17*(2), 570–586. https://doi.org/10.1109/TITS.2015.2471812

Chen, P., Zhu, Z., & Lu, G. (2019). An Adaptive Control Method for Arterial Signal Coordination Based on Deep Reinforcement Learning. *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*, *100191*(37), 3553–3558. https://doi.org/10.1109/ITSC.2019.8917051

Choe, C., Baek, S., Woon, B., Kong, S., & Member, S. (2018). Deep Q Learning with LSTM for Traffic Light Control. *2018 24th Asia-Pacific Conference on Communications (APCC)*, 331–336.

Chu, T., Wang, J., Codeca, L., & Li, Z. (2020). Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, *21*(3), 1086–1095. https://doi.org/10.1109/TITS.2019.2901791

Coşkun, M., Baggag, A., & Chawla, S. (2019). Deep reinforcement learning for traffic light optimization. *IEEE International Conference on Data Mining Workshops, ICDMW*, 564–571. https://doi.org/10.1109/ICDMW.2018.00088

Denney, R. W., Curtis, E., & Olson, P. (2012). The national traffic signal report card. *ITE Journal (Institute of Transportation Engineers)*.

Dijk, J. van. (2017). *Recurrent Neural Networks for Reinforcement Learning: an Investigation of Relevant Design Choices*. University of Amsterdam.

Dresner, K., & Stone, P. (2004). Multiagent traffic management: A reservation-based intersection control mechanism. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2004*.

Du, Y., Shangguan, W., Rong, D., & Chai, L. (2019). RA-TSC: Learning Adaptive Traffic Signal Control Strategy via Deep Reinforcement Learning. *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*, 3275–3280. https://doi.org/10.1109/ITSC.2019.8916967

Ejercito, P. M., Nebrija, K. G. E., Feria, R. P., & Lara-Figueroa, L. L. (2017). Traffic simulation software review. *2017 8th International Conference on Information, Intelligence, Systems and Applications, IISA 2017*, 1–4. https://doi.org/10.1109/IISA.2017.8316415

El-Tantawy, S., Abdulhai, B., & Abdelgawad, H. (2014). Design of reinforcement learning parameters for seamless application of adaptive traffic signal control. *Journal of Intelligent Transportation Systems: Technology, Planning, and Operations*. https://doi.org/10.1080/15472450.2013.810991

Elliott, D., Keen, W., & Miao, L. (2019). Recent advances in connected and automated vehicles. *Journal of Traffic and Transportation Engineering (English Edition)*, *6*(2), 109–131. https://doi.org/10.1016/j.jtte.2018.09.005

Fang, S., Chen, F., & Liu, H. (2019). Dueling Double Deep Q-Network for Adaptive Traffic Signal Control with Low Exhaust Emissions in A Single Intersection. *IOP Conference Series: Materials Science and Engineering*, *612*(5). https://doi.org/10.1088/1757-899X/612/5/052039

Fellendorf, M., & Vortisch, P. (2010). Microscopic traffic flow simulator VISSIM. In *International Series in Operations Research and Management Science*. https://doi.org/10.1007/978-1-4419-6142-6_2

Florin, R., & Olariu, S. (2015). A survey of vehicular communications for traffic signal optimization. *Vehicular Communications*, *2*(2), 70–79. https://doi.org/10.1016/j.vehcom.2015.03.002

Fortunato, M., Azar, M., Piot, B., Menick, J., Hessel, M., Osband, I., … Legg, S. (2017). Noisy networks for exploration. *ArXiv Preprint ArXiv:1706.10295.*

Gao, J., Shen, Y., Liu, J., Ito, M., & Shiratori, N. (2017). *Adaptive Traffic Signal Control : Deep Reinforcement Learning Algorithm with Experience Replay and Target Network* (pp. 1–10). pp. 1–10. arXiv preprint arXiv:1705.02755.

Genders, W., & Razavi, S. (2016). Using a Deep Reinforcement Learning Agent for Traffic Signal Control. *ArXiv Preprint ArXiv:1611.01142*, pp. 1–9.

Genders, W., & Razavi, S. (2018). Evaluating reinforcement learning state representations for adaptive traffic signal control. *Procedia Computer Science*, *130*, 26–33. https://doi.org/10.1016/j.procs.2018.04.008

Gong, Y., Abdel-Aty, M., Cai, Q., & Rahman, M. S. (2019). Decentralized network level adaptive signal control by multi-agent deep reinforcement learning. *Transportation Research Interdisciplinary Perspectives*, *1*(100020). https://doi.org/10.1016/j.trip.2019.100020

Guanetti, J., Kim, Y., & Borrelli, F. (2018). Control of connected and automated vehicles: State of the art and future challenges. *Annual Reviews in Control*, *45*(March), 18–40. https://doi.org/10.1016/j.arcontrol.2018.04.011

Guo, M., Wang, P., Chan, C. Y., & Askary, S. (2019). A Reinforcement Learning Approach for Intelligent Traffic Signal Control at Urban Intersections. *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*, 4242–4247. https://doi.org/10.1109/ITSC.2019.8917268

Guo, Q., Li, L., & Xuegang, B. (2019). Urban traffic signal control with connected and automated vehicles: A survey. *Transportation Research Part C: Emerging Technologies*, *101*(January), 313–334. https://doi.org/10.1016/j.trc.2019.01.026

Hausknecht, M., & Stone, P. (2015). Deep Recurrent Q-Learning for Partially Observable MDPs. *ArXiv:1507.06527v4*, pp. 1–7.

He, Z., Zheng, L., Chen, P., & Guan, W. (2017). Mapping to Cells: A Simple Method to Extract Traffic Dynamics from Probe Vehicle Data. *Computer-Aided Civil and Infrastructure Engineering*. https://doi.org/10.1111/mice.12251

Heberer, R. (2019). Why Going from Implementing Q-learning to Deep Q-learning Can Be Difficult. Retrieved May 13, 2020, from towardsdatascience.com website: https://towardsdatascience.com/why-going-from-implementing-q-learning-to-deep-q-learning-can-be-difficult-36e7ea1648af

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., … Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 3215–3222.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*. https://doi.org/10.1162/neco.1997.9.8.1735

Horsuwan, T., & Aswakul, C. (2019). Reinforcement learning agent under partial observability for traffic light control in presence of gridlocks. *EPiC Series in Computing*, *62*, 29–47. https://doi.org/10.29007/bdgn

Huang, Q., & Miller, R. (2004). Reliable wireless traffic signal protocols for smart intersections. *ITS*

America Annual Meeting. https://doi.org/10.1.1.219.4813

Hussain, A., Wang, T., & Jiahua, C. (2020). *Optimizing Traffic Lights with Multi-agent Deep Reinforcement Learning and V2X communication* (pp. 1–6). pp. 1–6. Retrieved from http://arxiv.org/abs/2002.09853

Ilgin Guler, S., Menendez, M., & Meier, L. (2014). Using connected vehicle technology to improve the efficiency of intersections. *Transportation Research Part C: Emerging Technologies*, *46*, 121–131. https://doi.org/10.1016/j.trc.2014.05.008

Islam, S. M. A. B. Al, Aziz, H. M. A., Wang, H., & Young, S. (2019). *Investigating the Impact of Connected Vehicle Market Share on the Performance of Reinforcement-Learning Based Traffic Signal Control (No. ORNL/TM-2019/1233)*. Oak Ridge, TN (United States).

Islam, S. M. A. B. Al, Aziz, H. M. A., Wang, H., & Young, S. E. (2018). Minimizing energy consumption from connected signalized intersections by reinforcement learning. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, *2018-Novem*, 1870–1875. https://doi.org/10.1109/ITSC.2018.8569891

Jing, P., Huang, H., & Chen, L. (2017). An adaptive traffic signal control in a connected vehicle environment: A systematic review. *Information (Switzerland)*, *8*(3). https://doi.org/10.3390/info8030101

Kamal, M. A. S., Hayakawa, T., & Imura, J. I. (2020). Development and Evaluation of an Adaptive Traffic Signal Control Scheme under a Mixed-Automated Traffic Scenario. *IEEE Transactions on Intelligent Transportation Systems*, *21*(2), 590–602. https://doi.org/10.1109/TITS.2019.2896943

Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., & Dabney, W. (2019). Recurrent experience replay in distributed reinforcement learning. *7th International Conference on Learning Representations, ICLR 2019*, 1–19.

Kim, J., Jung, S., Kim, K., & Lee, S. (2019). The Real-Time Traffic Signal Control System for the Minimum Emission using Reinforcement Learning in Vehicle-to-Everything (V2X) Environment. *Chemical Engineering Transactions*, *72*(March 2018), 91–96. https://doi.org/10.3303/CET1972016

Klöckner, R., & Klose, P. (2020). Deep-MARLIN: Using Deep Multi-Agent Reinforcement Learning for Adaptive Traffic Light Control. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3378184.3378194

Kocabaş, M. (2017). Human-level Control Through Deep Reinforcement Learning (Presentation). Retrieved May 13, 2020, from https://www.slideshare.net/MuhammedKocaba/humanlevel-control-through-deep-reinforcement-learning-presentation

Kotusevski, G., & Hawick, K. A. (2009). A review of traffic simulation software. *Research Letters in the Information and Mathematical Sciences*, *13*, 35–54.

Krauß, S. (1998). *Microscopic Modeling of Traffic Flow:Investigation of Collision Free Vehicle Dynamics*. Universität Köln.

Lample, G., & Chaplot, D. S. (2017). Playing FPS Games with Deep Reinforcement Learning. *Thirty-First AAAI Conference on Artificial Intelligence*.

Lee, J., Chung, J., & Sohn, K. (2019). Reinforcement learning for joint control of traffic signals in a transportation network. *IEEE Transactions on Vehicular Technology*, 1–12.

Li, L., Lv, Y., & Wang, F.-Y. (2016). Traffic Signal Timing via Deep Reinforcement Learning. *IEEE/CAA Journal of Automatica Sinica*, *3*(3), 247–254. https://doi.org/10.1007/978-981-13-8683-1_12

Li, L., Wen, D., & Yao, D. (2014). A survey of traffic control with vehicular communications. *IEEE Transactions on Intelligent Transportation Systems*, *15*(1), 425–432. https://doi.org/10.1109/TITS.2013.2277737

Li, Y. (2018a). Deep Reinforcement Learning: An Overview. *ArXiv Preprint ArXiv:1701.07274*. https://doi.org/10.1007/978-3-319-56991-8_32

Li, Y. (2018b). Deep reinforcement learning. *ArXiv:1810.06339v1*. https://doi.org/10.18653/v1/p18-5007

Liang, X., Du, X., Member, S., & Wang, G. (2019). A Deep Reinforcement Learning Network for Traffic Light Cycle Control. *IEEE Transactions on Vehicular Technology*, *68*(2), 1243–1253. https://doi.org/10.1109/TVT.2018.2890726

Liang, X., Du, X., Wang, G., & Han, Z. (2018). Deep Reinforcement Learning for Traffic Light Control in Vehicular Networks. *IEEE Transactions on Vehicular Technology*, *68*(2), 1–11. https://doi.org/10.1109/TVT.2018.2890726

Lin, L. J. (1992). Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. *Machine Learning*, *8*(3–4), 293–321. https://doi.org/10.1023/A:1022628806385

Mannion, P., Duggan, J., & Howley, E. (2016). An Experimental Review of Reinforcement Learning Algorithms for Adaptive Traffic Signal Control. *Autonomic Road Transport Support Systems*, 47–66. https://doi.org/10.1007/978-3-319-25808-9

Marchesini, P., & Weijermars, W. (2010). The relationship between road safety and congestion on motorways. *SWOV Institute for Road Safety Research*.

Martínez-Díaz, M., Soriguera, F., & Pérez, I. (2019). Autonomous driving: A bird's eye view. *IET Intelligent Transport Systems*, *13*(4), 563–579. https://doi.org/10.1049/iet-its.2018.5061

Mikami, S., & Kakazu, Y. (1994). Genetic reinforcement learning for cooperative traffic signal control. *IEEE Conference on Evolutionary Computation - Proceedings*. https://doi.org/10.1109/icec.1994.350012

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., … Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Mousavi, S. S., Schukat, M., & Howley, E. (2016). Deep Reinforcement Learning: An Overview. *Proceedings of SAI Intelligent Systems Conference*, 426–440. https://doi.org/10.1007/978-3-319-56991-8_32

Mousavi, S. S., Schukat, M., & Howley, E. (2017). Traffic light control using deep policy- gradient and value-function-based reinforcement learning. *IET Intelligent Transport Systems*, *11*, 417–423. https://doi.org/10.1049/iet-its.2017.0153

Namazi, E., Li, J., & Lu, C. (2019). Intelligent Intersection Management Systems Considering Autonomous Vehicles: A Systematic Literature Review. *IEEE Access*, *7*, 91946–91965. https://doi.org/10.1109/ACCESS.2019.2927412

Nawar, M., Fares, A., & Al-Sammak, A. (2019). Rainbow Deep Reinforcement Learning Agent for Improved Solution of the Traffic Congestion. *Proceedings of the International Japan-Africa Conference on Electronics, Communications and Computations, JAC-ECC 2019*, 80–83. https://doi.org/10.1109/JAC-ECC48896.2019.9051262

Nicholson, C. (n.d.). A Beginner's Guide to Deep Reinforcement Learning. Retrieved May 12, 2020, from pathmind.com website: https://pathmind.com/wiki/deep-reinforcement-learning

OpenAI. (2016). Getting Started with Gym. Retrieved July 1, 2020, from https://gym.openai.com/docs/

Osiński, B., & Budek, K. (2018). What is reinforcement learning? The complete guide. Retrieved May 12, 2020, from deepsense.ai website: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/

Overtoom, I. (2018). *Assessing the impact of shared and autonomous vehicles on urban traffic. A simmulation approach*. TU Delft.

Park, S. J., & Shires, D. (2019). Learning optimal actions with imperfect images. *Proc. SPIE 10996, Real-Time Image Processing and Deep Learning 2019, 109960F*, (May), 15. https://doi.org/10.1117/12.2518921

Pell, A., Meingast, A., & Schauer, O. (2017). Trends in Real-time Traffic Simulation. *Transportation Research Procedia*, *25*, 1477–1484. https://doi.org/10.1016/j.trpro.2017.05.175

Pol, E. van der. (2016). *Deep Reinforcement Learning for Coordination in Traffic Light Control*. University of Amsterdam.

Pol, E. van der, & Oliehoek, F. A. (2016). Coordinated Deep Reinforcement Learners for Traffic Light Control. *Proceedings of Learning, Inference and Control of Multi-Agent Systems (at NIPS 2016).*

Rafter, C. B., Anvari, B., & Box, S. (2018). Traffic responsive intersection control algorithm using GPS data. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*. https://doi.org/10.1109/ITSC.2017.8317795

Rahmati, Y., & Talebpour, A. (2017). Towards a collaborative connected, automated driving environment: A game theory based decision framework for unprotected left turn maneuvers. *IEEE Intelligent Vehicles Symposium, Proceedings*. https://doi.org/10.1109/IVS.2017.7995894

Ramesh, S. (2018). A guide to an efficient way to build neural network architectures- Part II: Hyper-parameter selection and tuning for Convolutional Neural Networks using Hyperas on Fashion-MNIST. Retrieved June 23, 2020, from towardsdatascience.com website: https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7

Rios-Torres, J., & Malikopoulos, A. A. (2017). A Survey on the Coordination of Connected and Automated Vehicles at Intersections and Merging at Highway On-Ramps. *IEEE Transactions on Intelligent Transportation Systems*, *18*(5), 1066–1077. https://doi.org/10.1109/TITS.2016.2600504

Rios-Torres, J., & Malikopoulos, A. A. (2018). Impact of Partial Penetrations of Connected and Automated Vehicles on Fuel Consumption and Traffic Flow. *IEEE Transactions on Intelligent Vehicles*. https://doi.org/10.1109/tiv.2018.2873899

Rodrigues, F., & Azevedo, C. L. (2019). Towards Robust Deep Reinforcement Learning for Traffic Signal

Control : Demand Surges , Incidents and Sensor Failures. *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 3559–3566. IEEE.

SAE International. (2019). SAE Standards News: J3016 automated-driving graphic update. Retrieved March 23, 2020, from https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic

Samad, A. M. (2020). *Multi Agent Deep Recurrent Q-Learning for Different Traffic Demands*. TU Delft.

Sarker, A., Shen, H., Rahman, M., Chowdhury, M., Dey, K., Li, F., … Narman, H. S. (2020). A Review of Sensing and Communication, Human Factors, and Controller Aspects for Information-Aware Connected and Automated Vehicles. *IEEE Transactions on Intelligent Transportation Systems*, *21*(1), 7–29. https://doi.org/10.1109/TITS.2019.2892399

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized experience replay. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*.

Schrank, D., Eisele, B., & Lomax, T. (2019). *Urban Mobility Report*. Retrieved from https://static.tti.tamu.edu/tti.tamu.edu/documents/mobility-report-2019.pdf

Schulze, C., & Schulze, M. (2018). ViZDoom: DRQN with prioritized experience replay, Double-Q learning and snapshot ensembling. *Proceedings of SAI Intelligent Systems Conference*, 1–17. Springer, Cham.

Shabestary, S. M. A., & Abdulhai, B. (2019). Deep reinforcement learning for adaptive traffic signal control. *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. https://doi.org/10.1115/DSCC2019-9076

Sharma, S. (2019). The Ultimate Beginner's Guide to Reinforcement Learning. Retrieved May 12, 2020, from towardsdatascience.com website: https://towardsdatascience.com/the-ultimate-beginners-guide-to-reinforcement-learning-588c071af1ec

Shi, M. K., Jiang, H., & Li, S. H. (2016). An intelligent traffic-flow-based real-time vehicles scheduling algorithm at intersection. *2016 14th International Conference on Control, Automation, Robotics and Vision, ICARCV 2016*, *2016*(November), 1–5. https://doi.org/10.1109/ICARCV.2016.7838779

Silver, D. (2015). *UCL reinforcement learning course*. Retrieved from http://www0.cs.ucl.ac.uk/staff/d.%0Asilver/web/Teaching.html

Sousa, N., Almeida, A., Coutinho-Rodrigues, J., & Natividade-Jesus, E. (2018). Dawn of autonomous vehicles: Review and challenges ahead. *Proceedings of the Institution of Civil Engineers: Municipal Engineer*, *171*(1), 3–14. https://doi.org/10.1680/jmuen.16.00063

SUMO. (2020). Simulation of Urban MObility. Retrieved June 19, 2020, from https://www.eclipse.org/sumo/

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, *3*(1), 9–44. https://doi.org/10.1023/A:1022633531479

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd Editio). Cambridge, MA; London, England: MIT Press.

Vagia, M., & Rødseth, E. J. (2019). A taxonomy for autonomous vehicles for different transportation modes. *Journal of Physics: Conference Series*, *1357*(1). https://doi.org/10.1088/1742-

6596/1357/1/012022

Validi, A., Ludwig, T., Hussein, A., & Olaverri-Monreal, C. (2018). Examining the Impact on Road Safety of Different Penetration Rates of Vehicle-to-Vehicle Communication and Adaptive Cruise Control. *IEEE Intelligent Transportation Systems Magazine*. https://doi.org/10.1109/MITS.2018.2867534

Van Hasselt, H., Guez, A., & Silver, D. (2015). Deep reinforcement learning with double Q-Learning. *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, 2094–2100.

Vidali, A. (2018). *Simulation of a traffic light scenario controlled by a Deep Reinforcement Learning agent*. Università degli Studi di Milano Bicocca.

Wang, S., Xie, X., Huang, K., Zeng, J., & Cai, Z. (2019). Deep reinforcement learning-based traffic signal control using high-resolution event-based data. *Entropy*, *21*(8), 1–16. https://doi.org/10.3390/e21080744

Wang, Y., Yang, X., Liang, H., & Liu, Y. (2018). A review of the self-adaptive traffic signal control system based on future traffic environment. *Journal of Advanced Transportation*, *2018*. https://doi.org/10.1155/2018/1096123

Wang, Z., Schaul, T., Hessel, M., Lanctot, M., Parisotto, E., Ba, J. L., … Botvinick, M. (2016). Dueling Network Architectures for Deep Reinforcement Learning Hado van Hasselt. *Advances in Neural Information Processing Systems*. https://doi.org/10.1039/c004615a

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. University of Cambridge.

Webster, F. V. (1958). Traffic signal settings. *Road Research Technical Paper, No.39, Road Research Laboratory, London*.

Wei, H., Zheng, G., Yao, H., & Li, Z. (2018). IntelliLight : A Reinforcement Learning Approach for Intelligent Traffic Light Control. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2496–2505.

Wu, T., Kong, F., & Fan, Z. (2019). Road Model Design Based on Reward Function in Traffic Light Control. *2019 5th International Conference on Control, Automation and Robotics, ICCAR 2019*, 407–412. https://doi.org/10.1109/ICCAR.2019.8813381

Wu, Y., Chen, H., & Zhu, F. (2019). DCL-AIM: Decentralized coordination learning of autonomous intersection management for connected and automated vehicles. *Transportation Research Part C: Emerging Technologies*, *103*(November 2018), 246–260. https://doi.org/10.1016/j.trc.2019.04.012

Wuthishuwong, C., & Traechtler, A. (2017). Consensus-based local information coordination for the networked control of the autonomous intersection management. *Complex & Intelligent Systems*, *3*(1), 17–32. https://doi.org/10.1007/s40747-016-0032-6

Xu, M., Wu, J., Huang, L., Zhou, R., Wang, T., & Hu, D. (2020). Network-wide traffic signal control based on the discovery of critical nodes and deep reinforcement learning. *Journal of Intelligent Transportation Systems: Technology, Planning, and Operations*, *24*(1), 1–10. https://doi.org/10.1080/15472450.2018.1527694

Yang, K., Guler, S. I., & Menendez, M. (2016). Isolated intersection control for various levels of vehicle technology: Conventional, connected, and automated vehicles. *Transportation Research Part C: Emerging Technologies*, *72*, 109–129. https://doi.org/10.1016/j.trc.2016.08.009

Yang, K., Tan, I., & Menendez, M. (2017). A reinforcement learning based traffic signal control algorithm in a connected vehicle environment. *17th Swiss Transport Research Conference (STRC 2017)*. https://doi.org/10.3929/ethz-a-010782581

Yau, K.-L. A., Qadir, J., Khoo, H. L., Ling, M. H., & Komisarczuk, P. (2017). A survey on reinforcement learning models and algorithms for traffic signal control. *ACM Computing Surveys*, *50*(3), 38. https://doi.org/10.1145/3068287

Yin, M., Wang, Y., & Li, Z. (2020). Optimization of Multi-Intersection Traffic Signal Timing Model Based on Improved Q-Learning Optimization of Multi-Intersection Traffic Signal Timing Model Based on Improved Q-Learning. *IOP Conf. Ser.: Mater. Sci. Eng.* https://doi.org/10.1088/1757-899X/768/7/072100

Zeadally, J., Guerrero, J., & Contreras, S. (2019). A tutorial survey on vehicle - to - vehicle communications Line of Sight Non Line of Sight. *Telecommunication Systems*, (0123456789), 469–489. https://doi.org/10.1007/s11235-019-00639-8

Zeng, J., Hu, J., & Zhang, Y. (2018). Adaptive Traffic Signal Control with Deep Recurrent Q-learning. *2018 IEEE Intelligent Vehicles Symposium (IV)*, 1215–1220. https://doi.org/10.1109/IVS.2018.8500414

Zeng, J., Hu, J., & Zhang, Y. (2019). Training Reinforcement Learning Agent for Traffic Signal Control under Different Traffic Conditions. *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*, 4248–4254. https://doi.org/10.1109/ITSC.2019.8917342

Zhang, H., Tam, G. M., & Shi, J. J. (2002). Simulation-based methodology for project scheduling. *Construction Management and Economics*. https://doi.org/10.1080/0144619022000014088

Zhang, R., Ishikawa, A., Wang, W., Striner, B., & Tonguz, O. (2020). Using Reinforcement Learning with Partial Vehicle Detection for Intelligent Traffic Signal Control. *IEEE Transactions on Intelligent Transportation Systems*, 1–12. https://doi.org/10.1109/TITS.2019.2958859

Zhao, D., Dai, Y., & Zhang, Z. (2012). Computational intelligence in urban traffic signal control: A survey. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, *42*(4), 485–494. https://doi.org/10.1109/TSMCC.2011.2161577

Zhao, T., & Wang, P. (2019). Traffic Signal Control with Deep Reinforcement learning. *2019 International Conference on Intelligent Computing, Automation and Systems (ICICAS)*, *2*(c), 763–767. https://doi.org/10.1109/ICICAS48597.2019.00164

Zheng, J., & Liu, H. X. (2017). Estimating traffic volumes for signalized intersections using connected vehicle data. *Transportation Research Part C: Emerging Technologies*. https://doi.org/10.1016/j.trc.2017.03.007

APPENDIX A: PRELIMINARY TESTING SETTINGS AND RESULTS

This appendix includes all base agents mentioned in chapter 8.2, including all details on their model settings. Additionally, experiments which have been referred to in chapter 8.2, but which have not been included there due to space reasons are shown here. This appendix follows the same order as chapter 8.2.

A.1 State representation experiments – vehicle position vs density

Table 7 Model settings for the preliminary experiment on the state representation (vehicle position vs density)

Model setting or (Hyper-)parameter	**Value**
Training episodes	330
Training epochs	1000
Minibatch size	100
Target network freeze interval [steps]	6000
Replay memory size	50,000
Discount factor	0.99
Learning rate	0.001
Penetration rate	0.9
Network architecture	Convolution: Filters: 32, 64, 32 Kernel: 2, 2, 2 Stride: 2, 2, 1 Phase: One-hot encoded
Reward	Change in cumulative delay
Recurrence	No

A.2 State representation experiments – one-hot encoded phase vs +/- encoded phase

Table 8 Model settings for the first preliminary experiment on the state representation (one-hot encoding vs +/- encoding)

Model setting or (Hyper-)parameter	**Value**
Training episodes	330
Training epochs	1000
Minibatch size	100
Target network freeze interval [steps]	6000
Replay memory size	50,000
Discount factor	0.99
Learning rate	0.001
Penetration rate	0.9
Network architecture	Convolution: Filters: 32, 64, 32 Kernel: 2, 2, 2 Stride: 2, 2, 1 Phase: (under investigation)
Reward	Change in cumulative delay
Recurrence	No

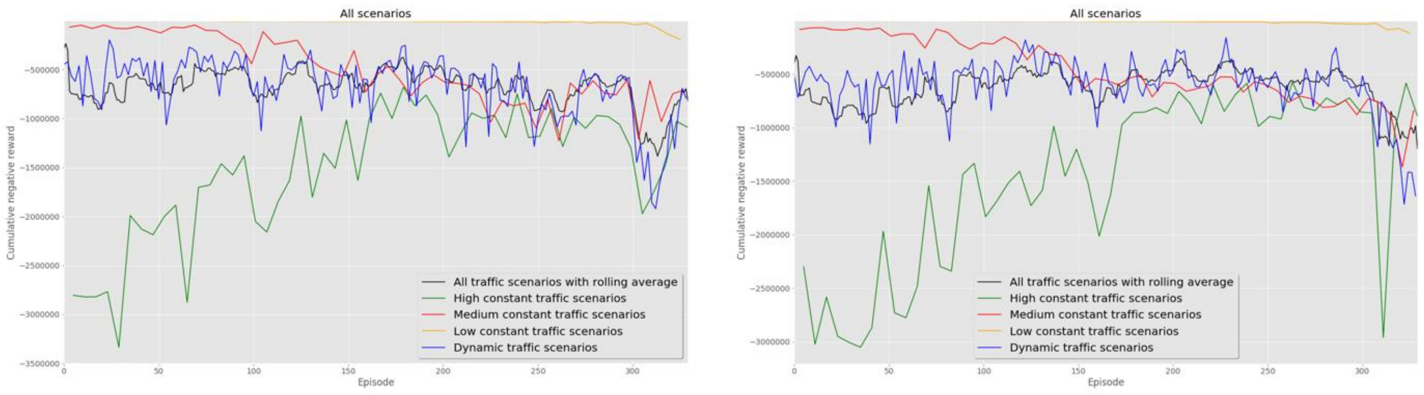


Figure 52 Cumulative negative reward per episode during training. Experiments with base agent 2. Left figure: +/- phase encoding. Right figure: one-hot phase encoding.

Table 9 Model settings for the second preliminary experiment on the state representation (one-hot encoding vs +/- encoding)

Model setting or (Hyper-)parameter	**Value**
Training episodes	504
Training epochs	1000
Minibatch size	100
Target network freeze interval [steps]	6000
Replay memory size	50,000
Discount factor	0.99
Learning rate	0.001
Penetration rate	0.9
Network architecture	Convolution: Filters: 4, 8 Kernel: 2, 2 Stride: 2, 1 Vehicle position: Density (float) Phase: (under investigation)
Reward	Change in cumulative delay
Recurrence	No

A.3 Convolutional network shape experiments

Table 10 Model settings for the preliminary experiment on the shape of the convolutional network

Model setting or (Hyper-)parameter	**Value**
Training episodes	400
Training epochs	300
Minibatch size	100
Target network freeze interval [steps]	14,000
Replay memory size	50,000
Discount factor	0.99
Learning rate	0.001
Penetration rate	1
Network architecture	Convolution: (under investigation) Phase: +/- encoded
Reward	Change in cumulative waiting time
Recurrence	No

A.4 Reward representation experiments

Table 11 Model settings for the preliminary experiment on the reward function

Model setting or (Hyper-)parameter	**Value**
Training episodes	400
Training epochs	300
Minibatch size	100
Target network freeze interval [steps]	14,000
Replay memory size	50,000
Discount factor	0.99
Learning rate	0.001
Penetration rate	1
Network architecture	Convolution: Filters: 4, 8 Kernel: 2, 2 Stride: 2, 1 Vehicle position: Density (float) Phase: one-hot encoded
Reward	(under investigation)
Recurrence	Yes Sequence length: 18

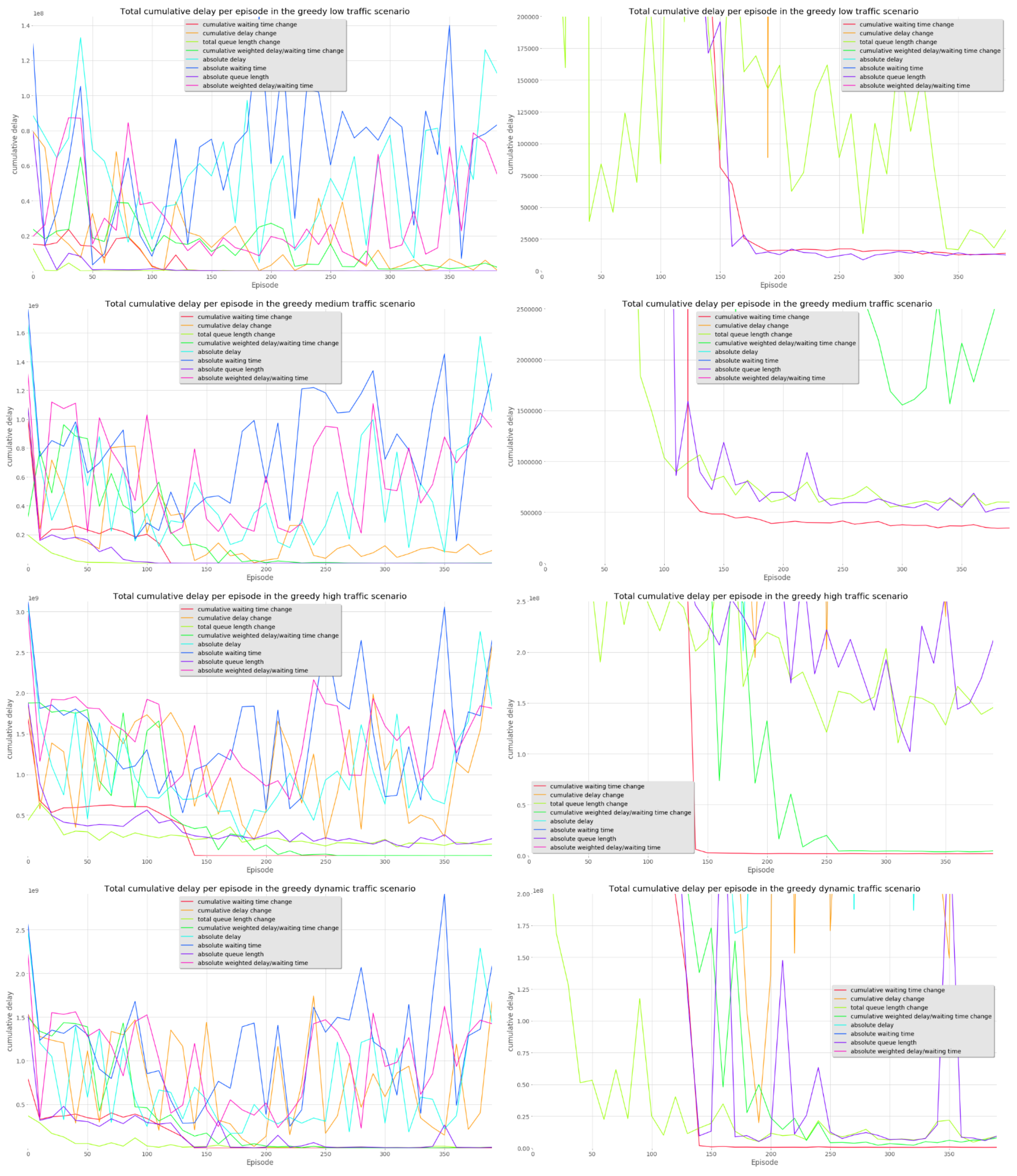


Figure 53 Impact of different reward representations of the cumulative delays in different greedy scenarios. The left figures show the cumulative delay per greedy episode of all reward representations. Since the details of the best-performing agents are hidden, the right figure zoom in only on these best-performing agents. From top to bottom, the figures show the results for the low, medium, high and dynamic traffic scenario.

A.5 Numbers of epochs and episodes

Table 12 Model settings for the first preliminary experiment on the number of episodes and epochs

Model setting or (Hyper-)parameter	**Value**
Training episodes	(under investigation)
Training epochs	(under investigation)
Minibatch size	100
Target network freeze interval [steps]	6000
Replay memory size	50,000
Discount factor	0.99
Learning rate	0.001
Penetration rate	0.9
Network architecture	Convolution: Filters: 4, 8 Kernel: 2, 2 Stride: 2, 1 Vehicle position: Density (float) Phase: one-hot encoded
Reward	Change in cumulative delay
Recurrence	No

Table 13 Model settings for the second preliminary experiment on the number of episodes and epochs

Model setting or (Hyper-)parameter	**Value**
Training episodes	(under investigation)
Training epochs	(under investigation)
Minibatch size	100
Target network freeze interval [steps]	12,000
Replay memory size	50,000
Discount factor	0.99
Learning rate	0.001
Penetration rate	1
Network architecture	Convolution: Filters: 4, 8 Kernel: 2, 2 Stride: 2, 1 Vehicle position: Density (float) Phase: +/- encoded
Reward	Change in cumulative waiting time
Recurrence	No

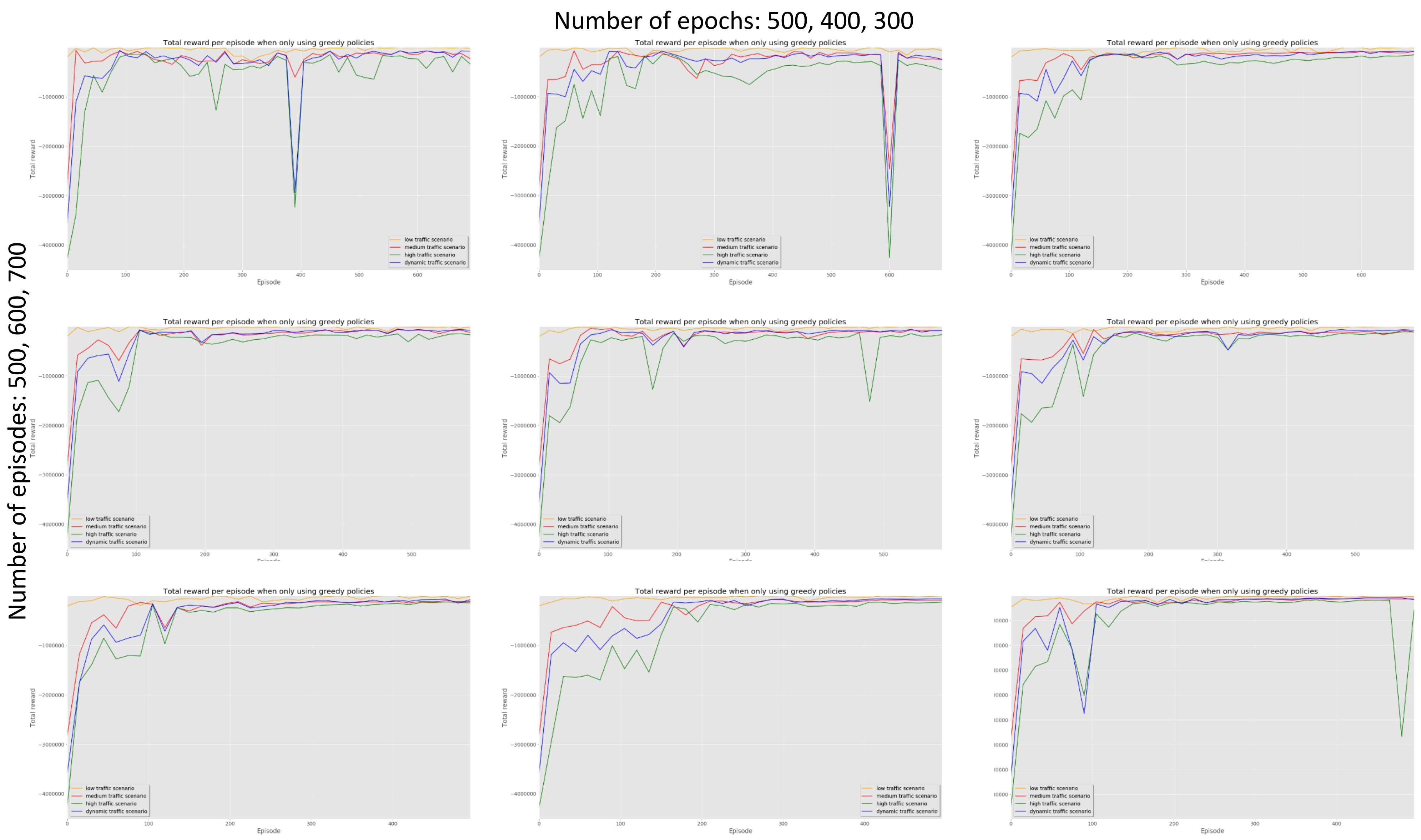


Figure 54 Cumulative rewards per greedy episode during training. The figures show different combinations of episodes (from top to bottom: 700, 600, 500) and epochs (from left to right: 500, 400, 300)

A.6 Target Network Freeze Interval

Table 14 Model settings for the preliminary experiment on the target network freeze interval

Model setting or (Hyper-)parameter	**Value**
Training episodes	400
Training epochs	300
Minibatch size	100
Target network freeze interval [steps]	(under investigation)
Replay memory size	50,000
Discount factor	0.99
Learning rate	0.001
Penetration rate	1
Network architecture	<u>Convolution</u>: Filters: 4, 8 Kernel: 2, 2 Stride: 2, 1 <u>Vehicle position</u>: Density (float) <u>Phase</u>: +/- encoded
Reward	Change in cumulative waiting time
Recurrence	No

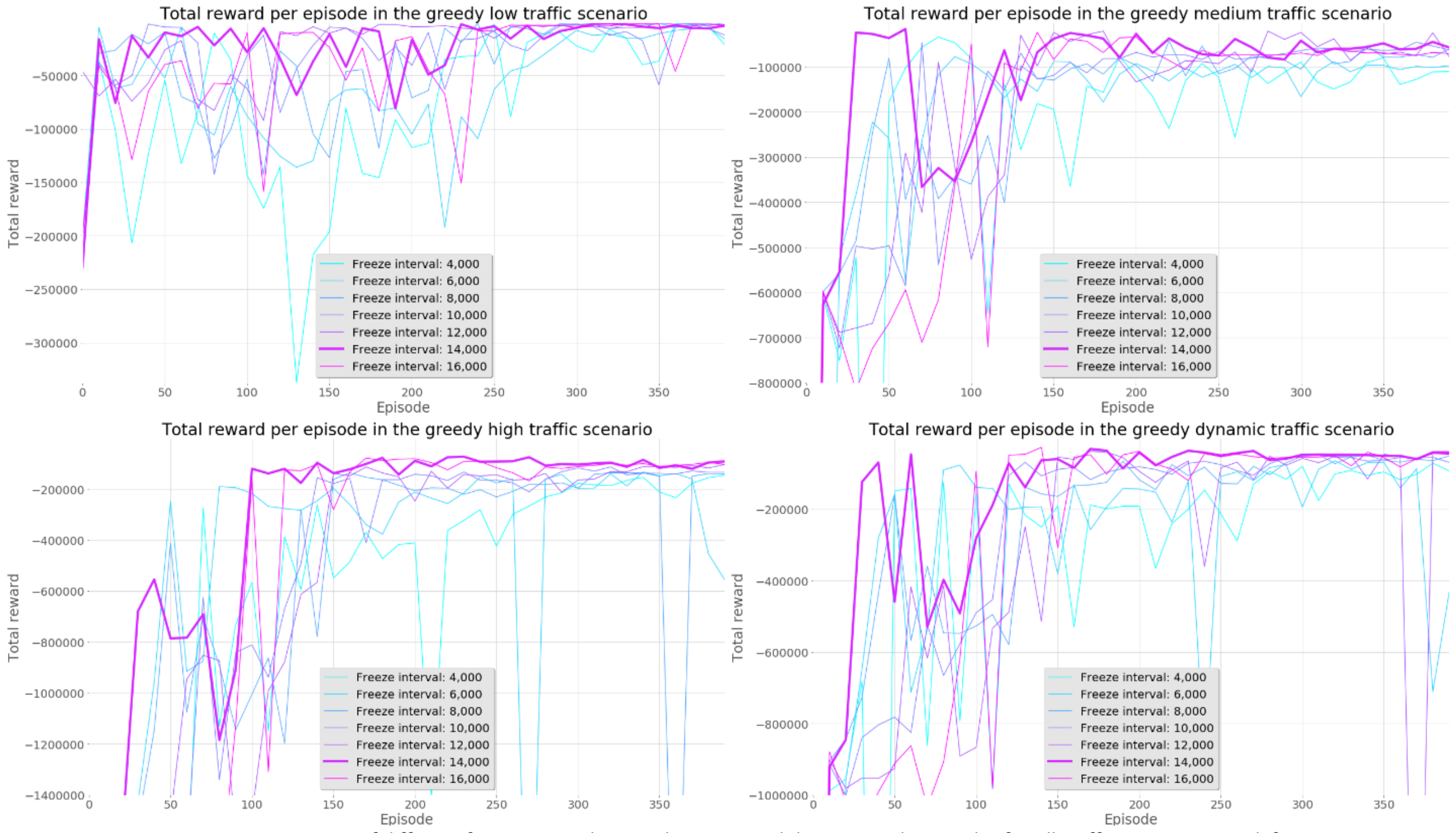


Figure 55 Comparison of different freeze intervals: cumulative reward during greedy episodes for all traffic scenarios. Top left: low traffic scenario. Top right: medium traffic scenario. Bottom left: high traffic scenario. Bottom right: dynamic traffic scenario.

A.7 Memory size

Table 15 Model settings for the preliminary experiment on the memory size

Model setting or (Hyper-)parameter	**Value**
Training episodes	400
Training epochs	300
Minibatch size	100
Target network freeze interval [steps]	14,000
Replay memory size	(under investigation)
Discount factor	0.99
Learning rate	0.001
Penetration rate	1
Network architecture	Convolution: Filters: 4, 8 Kernel: 2, 2 Stride: 2, 1 Vehicle position: Density (float) Phase: +/- encoded
Reward	Change in cumulative waiting time
Recurrence	No

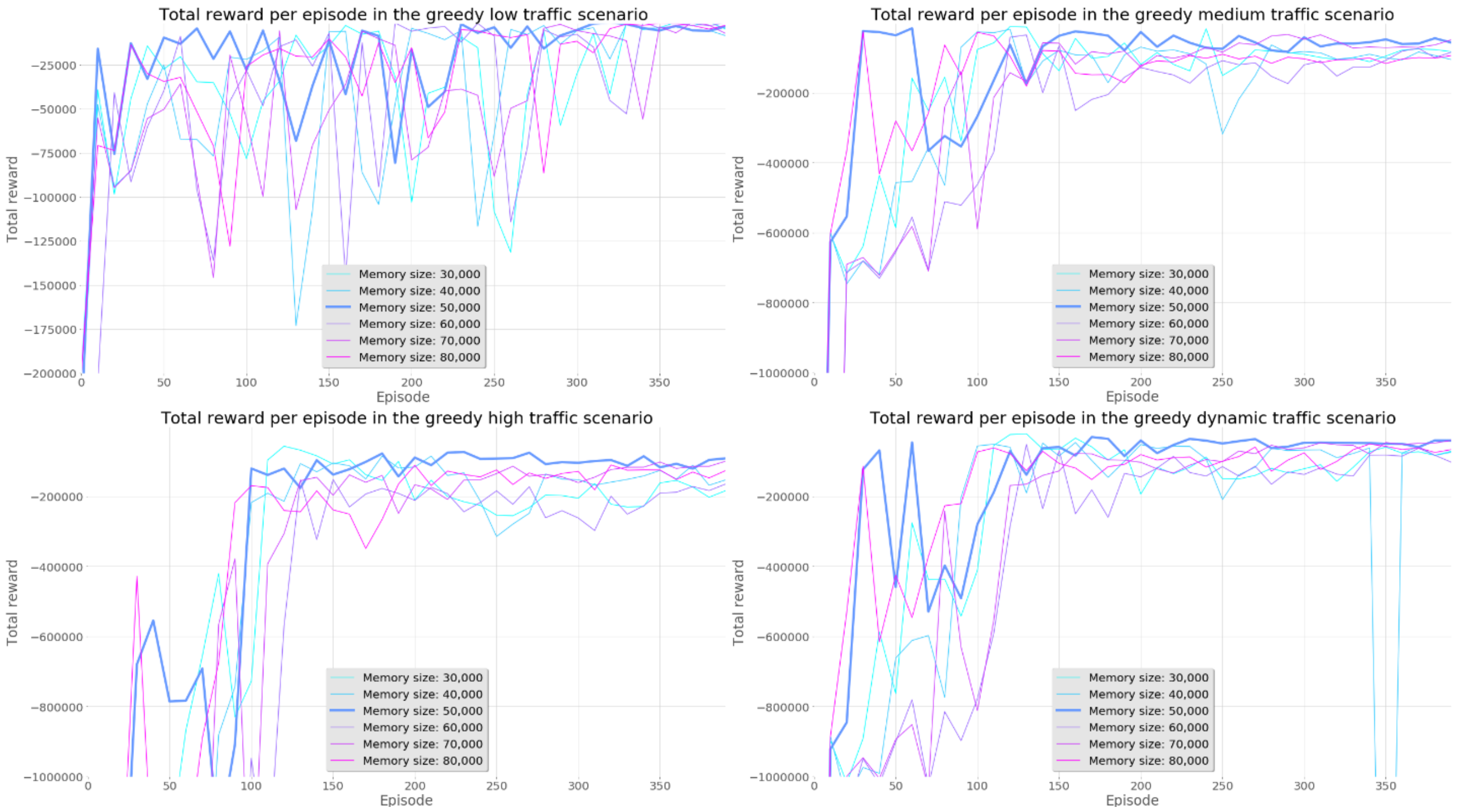


Figure 56 Comparison of different memory sizes: cumulative reward during greedy episodes for all traffic scenarios. Top left: low traffic scenario. Top right: medium traffic scenario. Bottom left: high traffic scenario. Bottom right: dynamic traffic scenario.

A.8 Trajectory sequence length experiments

Table 16 Model settings for the preliminary experiment on the sequence length

Model setting or (Hyper-)parameter	**Value**
Training episodes	400
Training epochs	300
Minibatch size	100
Target network freeze interval [steps]	14,000
Replay memory size	50,000
Discount factor	0.99
Learning rate	0.001
Penetration rate	[1, 0.5]
Network architecture	Convolution: Filters: 4, 8 Kernel: 2, 2 Stride: 2, 1 Vehicle position: Density (float) Phase: +/- encoded
Reward	Change in cumulative waiting time
Recurrence	Yes Sequence length: (under investigation)

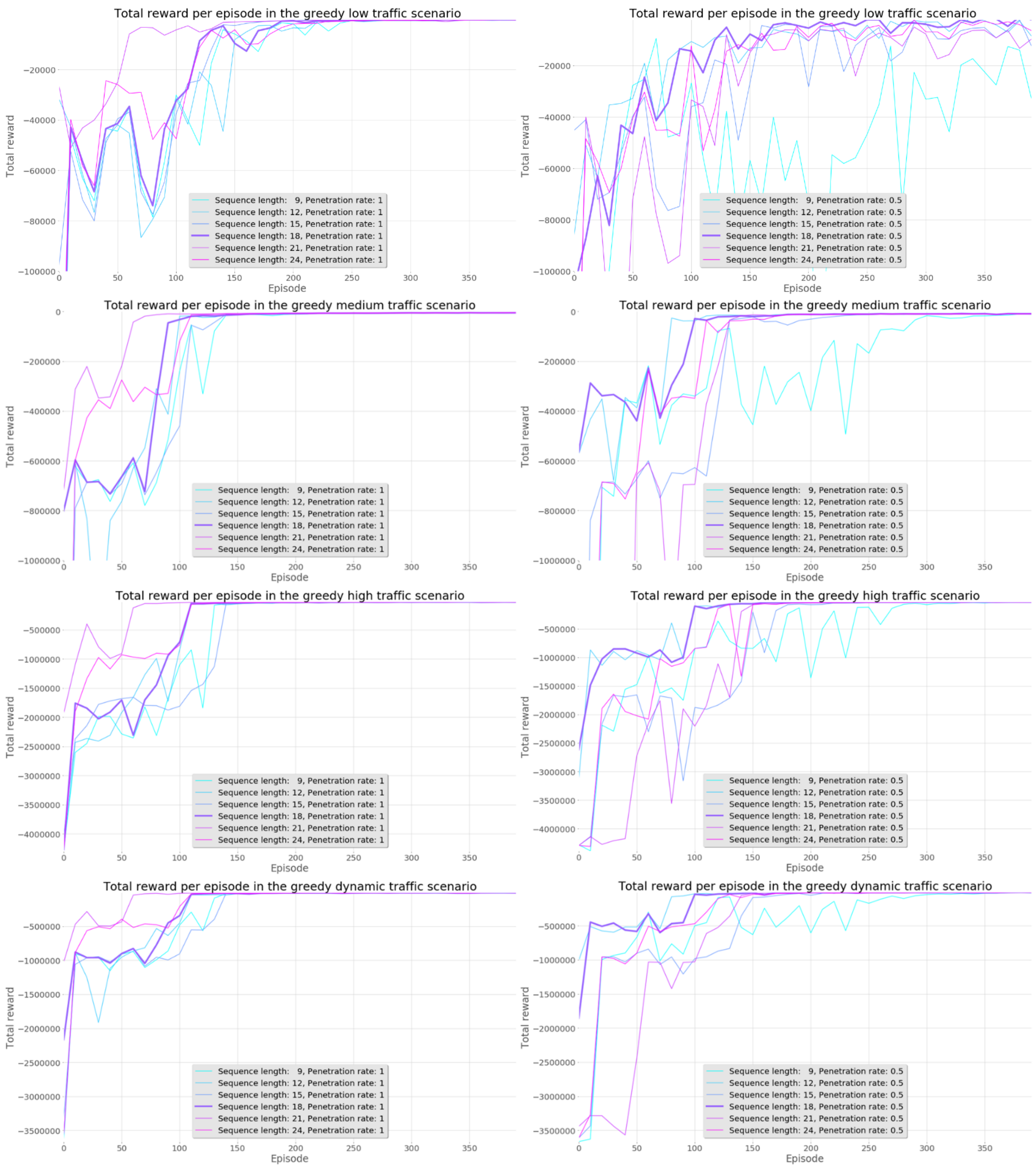
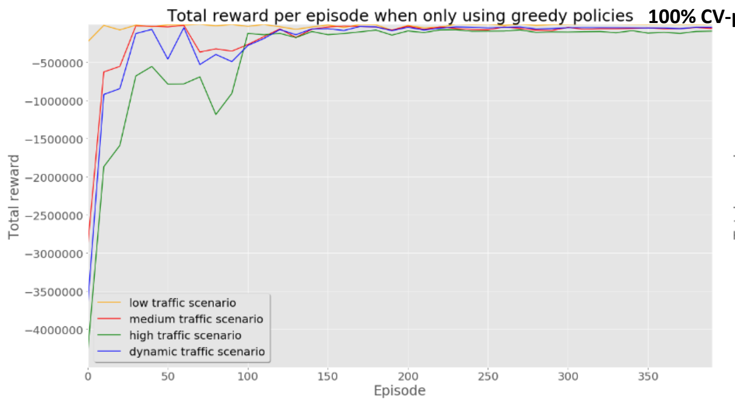
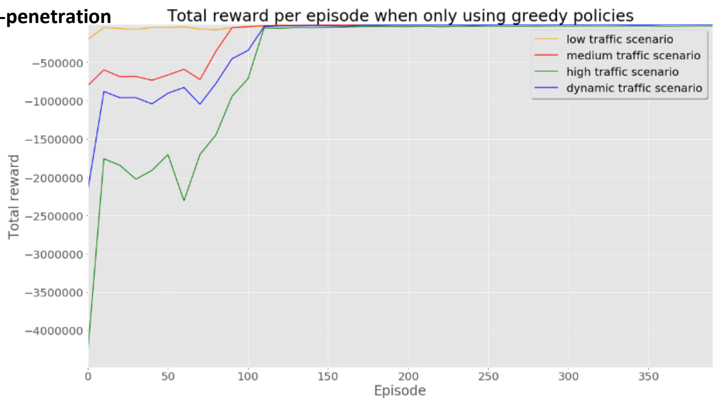


Figure 57 Comparison of different trajectory lengths: cumulative reward during greedy episodes for all traffic scenarios. The left figures show the results for agents trained on 100% CV-penetration; the right figures show the results for agents trained under 50% CV-penetration. From top to bottom the figures show the results for the low, medium, high and dynamic traffic scenarios.

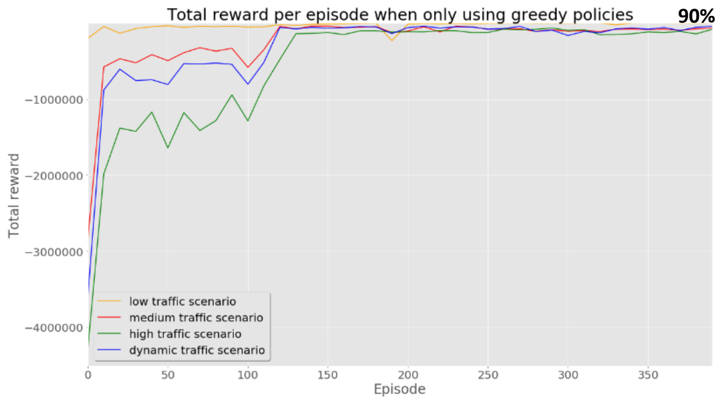
A.9 Fine-tune agents

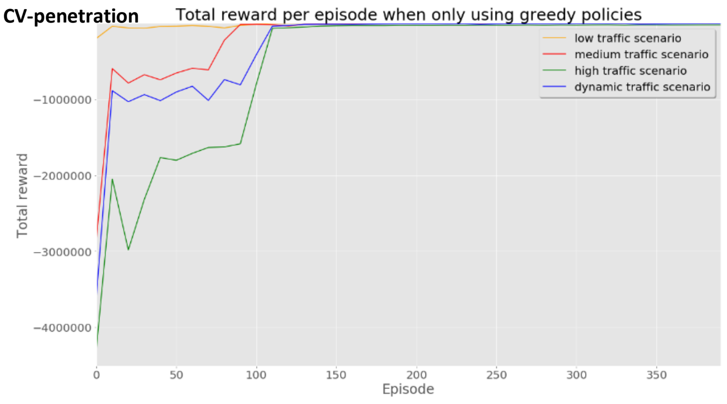
100% CV-penetration



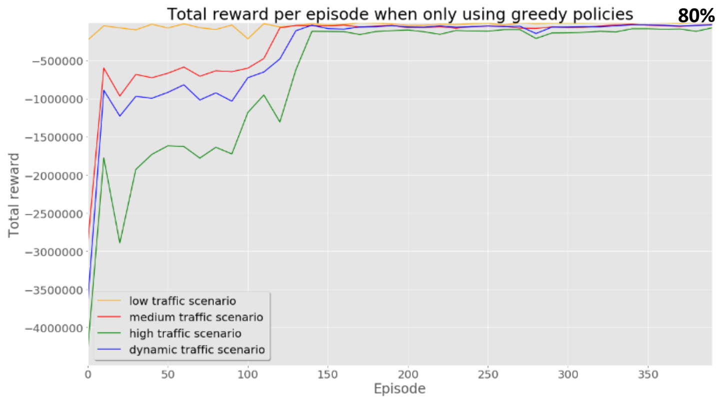


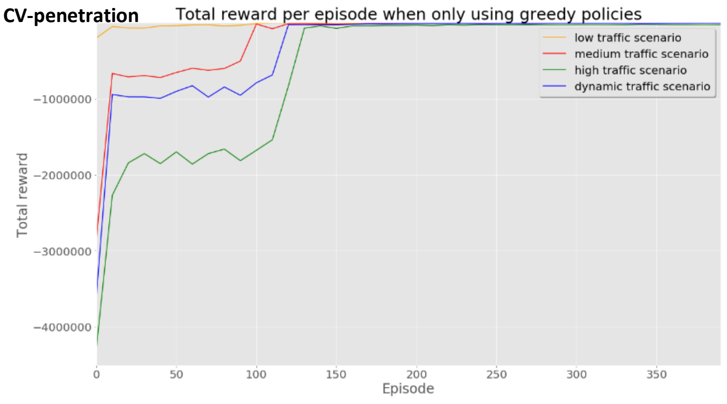
90% CV-penetration



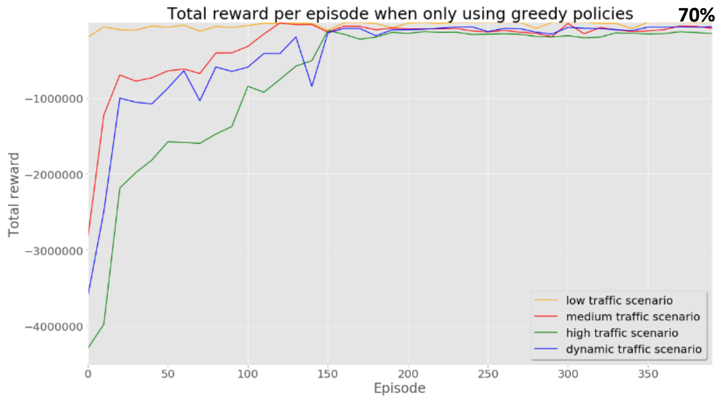


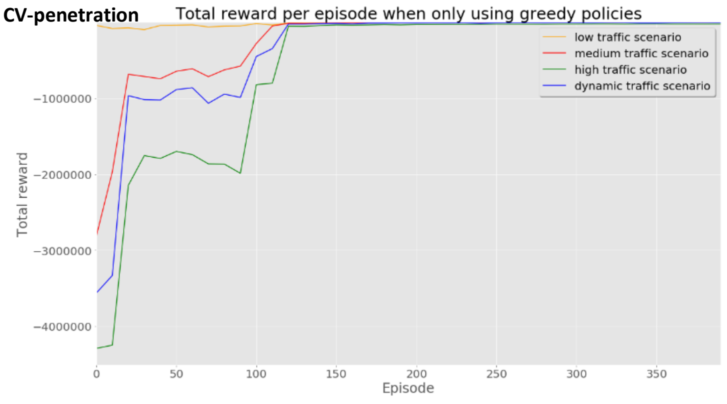
80% CV-penetration



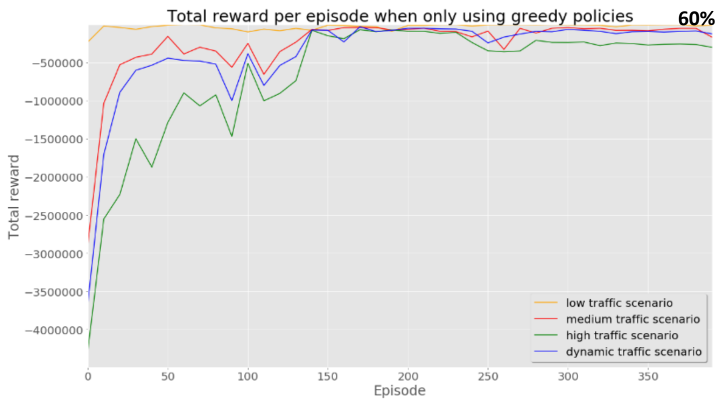


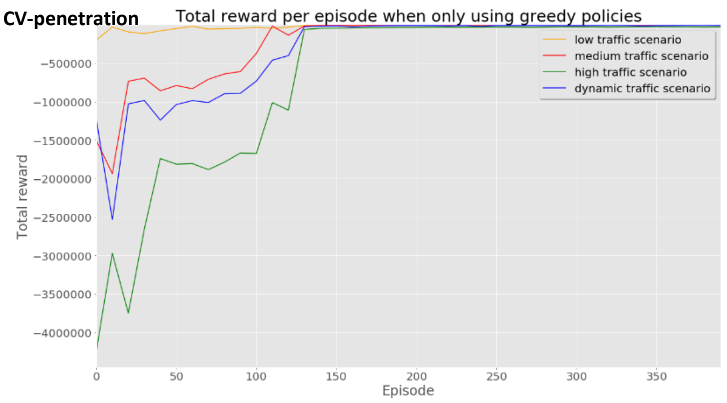
70% CV-penetration



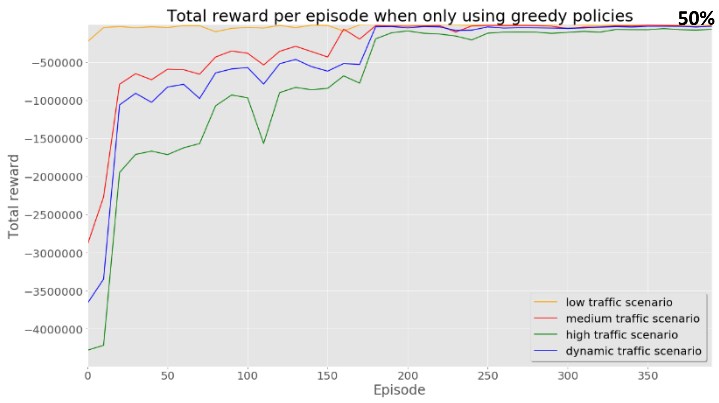


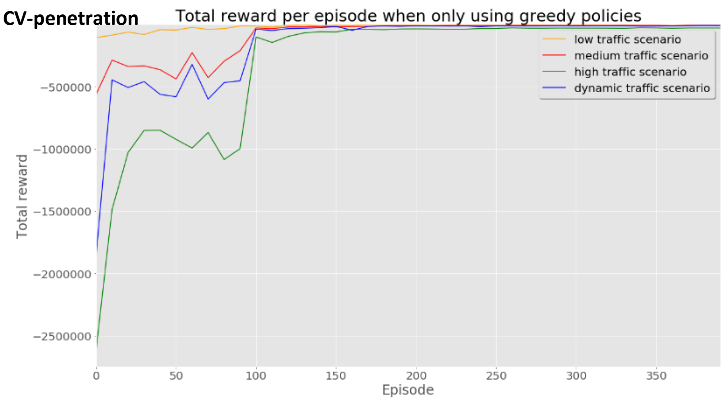
60% CV-penetration



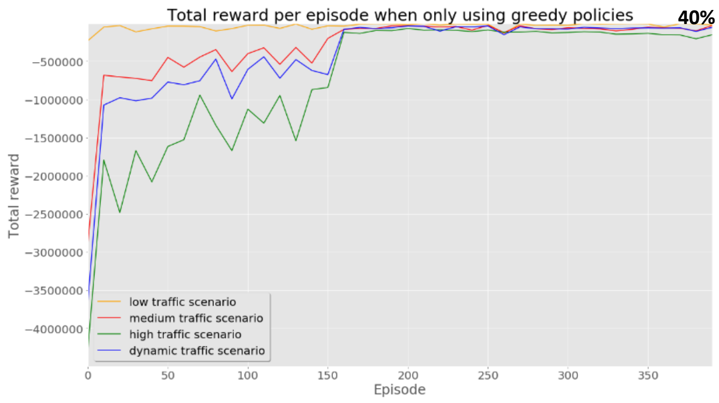


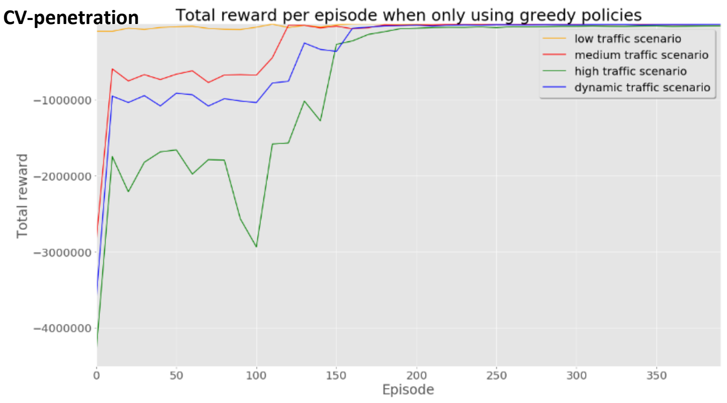
50% CV-penetration



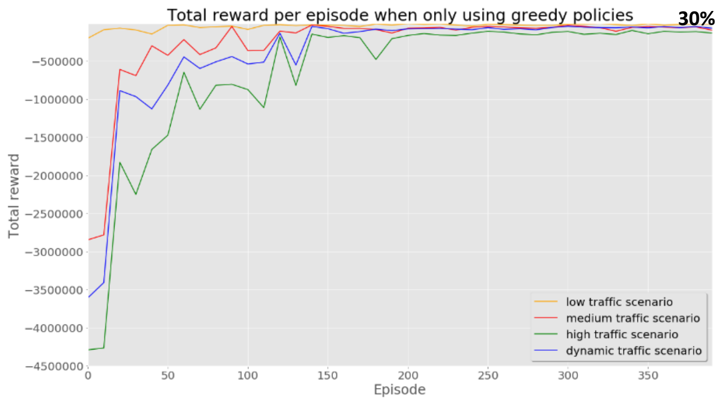


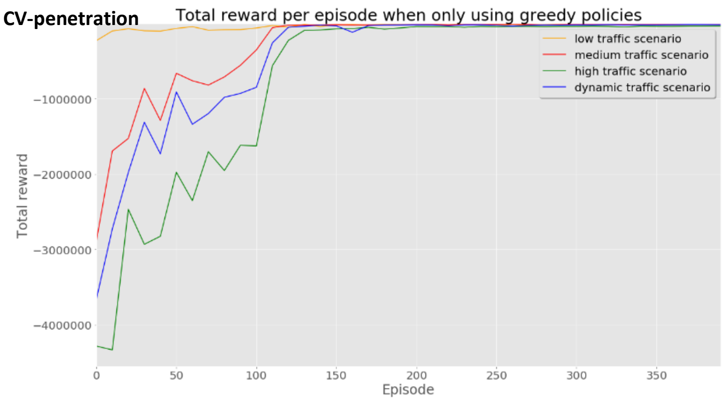
40% CV-penetration



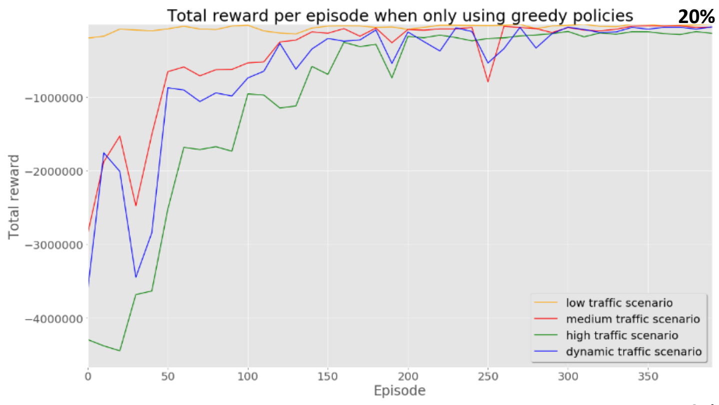


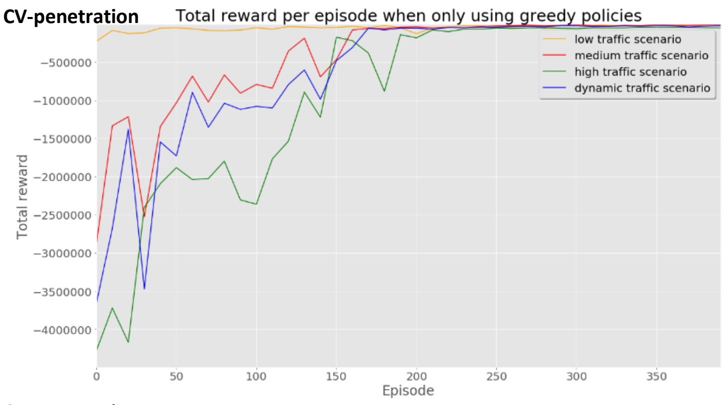
30% CV-penetration



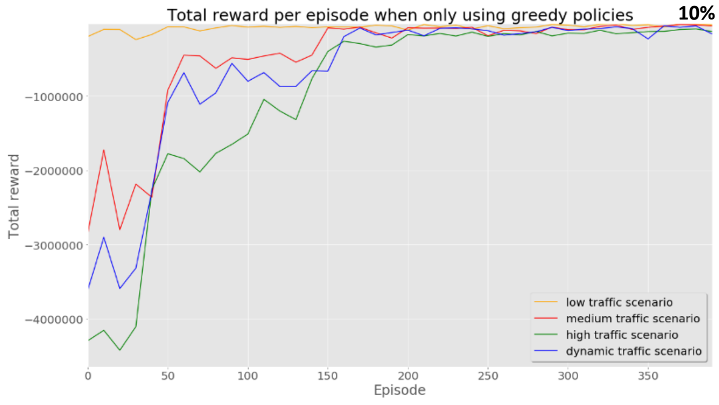


20% CV-penetration





10% CV-penetration



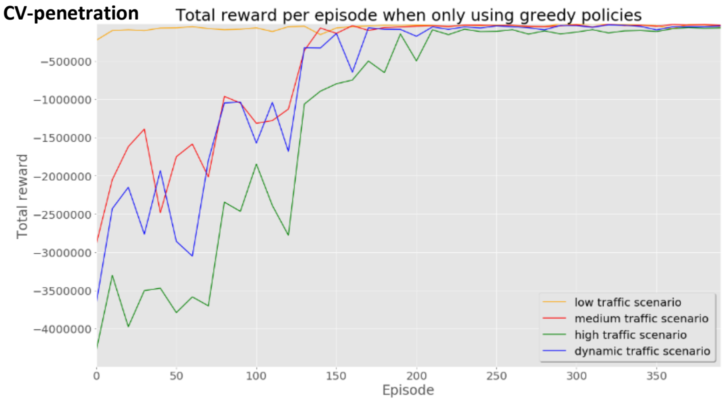


Figure 58 Comparison of the final fine-tuned agents trained under different penetration rates: cumulative rewards per greedy episode. The left figures show the results for vanilla agents, the right figures show the results for recurrent agents. From top to bottom, the figures show the results for agents trained under 100%, 90%, 80%, 70%, 60%, 50%, 40%, 30%, 20%, 10% CV-penetration

APPENDIX B: MIXED TRAFFIC EXPERIMENTS RESULTS

This appendix includes the results of the mixed traffic experiments (section 8.3) which have not been included in the main text due to space limitations. In this appendix, first the results for the agent stability are shown, then the results per traffic scenario (low, medium, high, dynamic) are presented, and finally the plots comparing vanilla, recurrent and fixed-time controllers are shown.

B.1 Analysis of agent stability

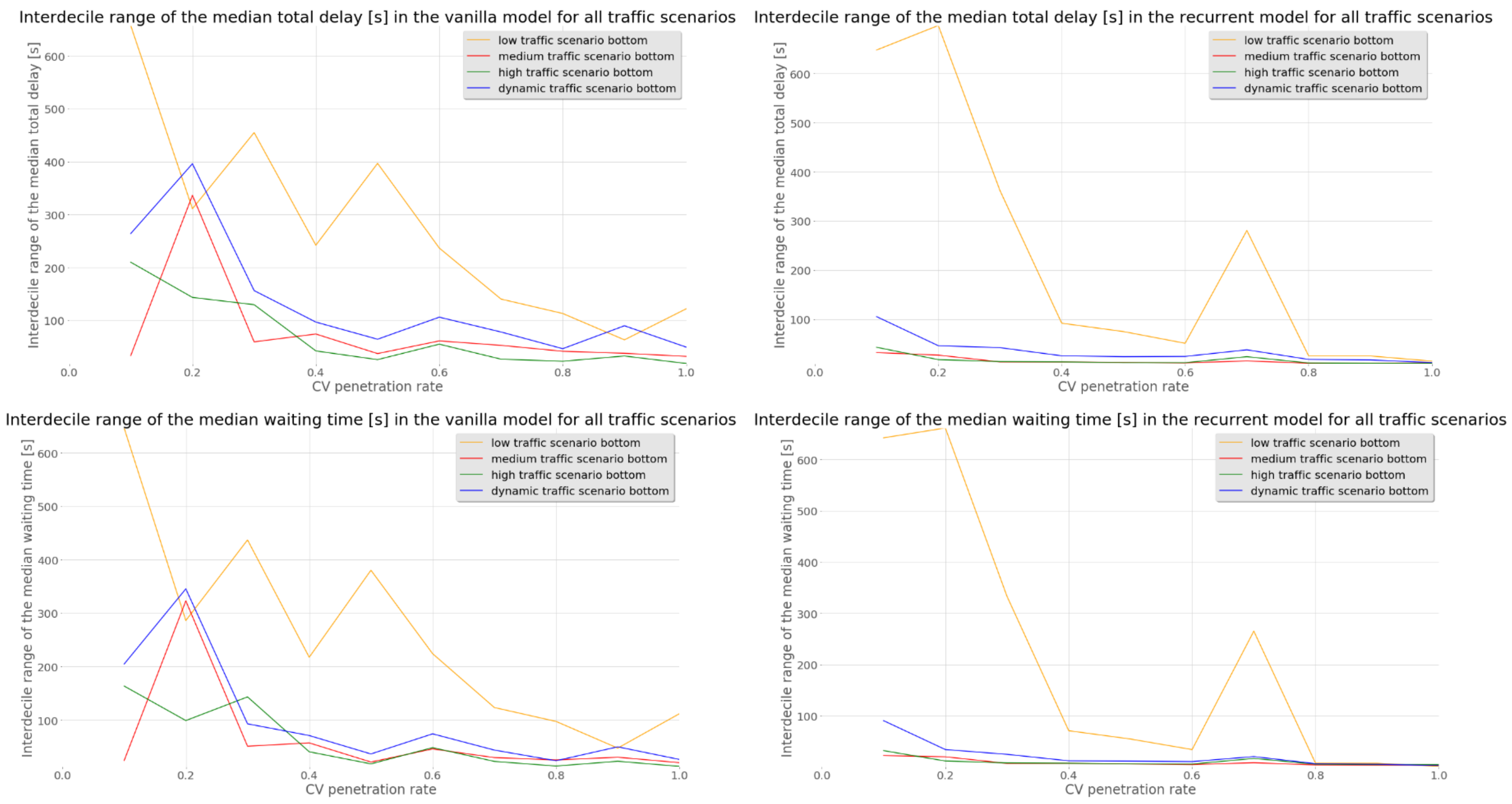


Figure 59 Stability of the trained vanilla and recurrent agents. The top figures show the interdecile range of the median delay and the bottom figures show the interdecile range of the median waiting time. The left figures show the stability of the vanilla agents and the right figures show the stability of the recurrent agents.

B.2 Analysis of agents per traffic scenario

B.2.1 Low traffic scenario

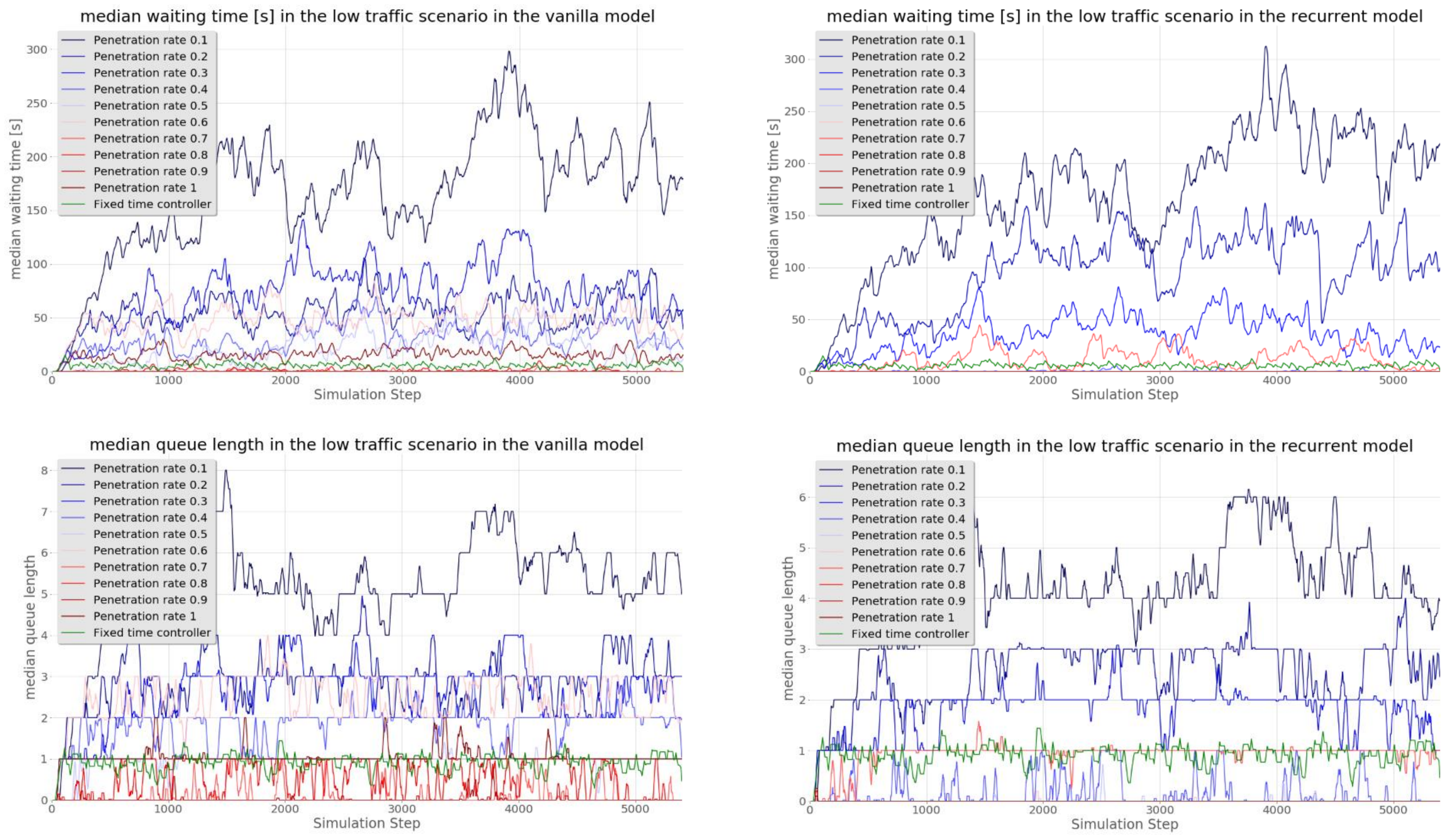


Figure 60 Median cumulative waiting times and delays in the low traffic scenario for all penetration rates. Top left figure: median waiting times for the vanilla agent. Top right figure: median waiting times for the recurrent agent. Bottom left figure: median queue lengths for the vanilla agent. Bottom right figure: median queue lengths for the recurrent agent.

B.2.2 Medium traffic scenario

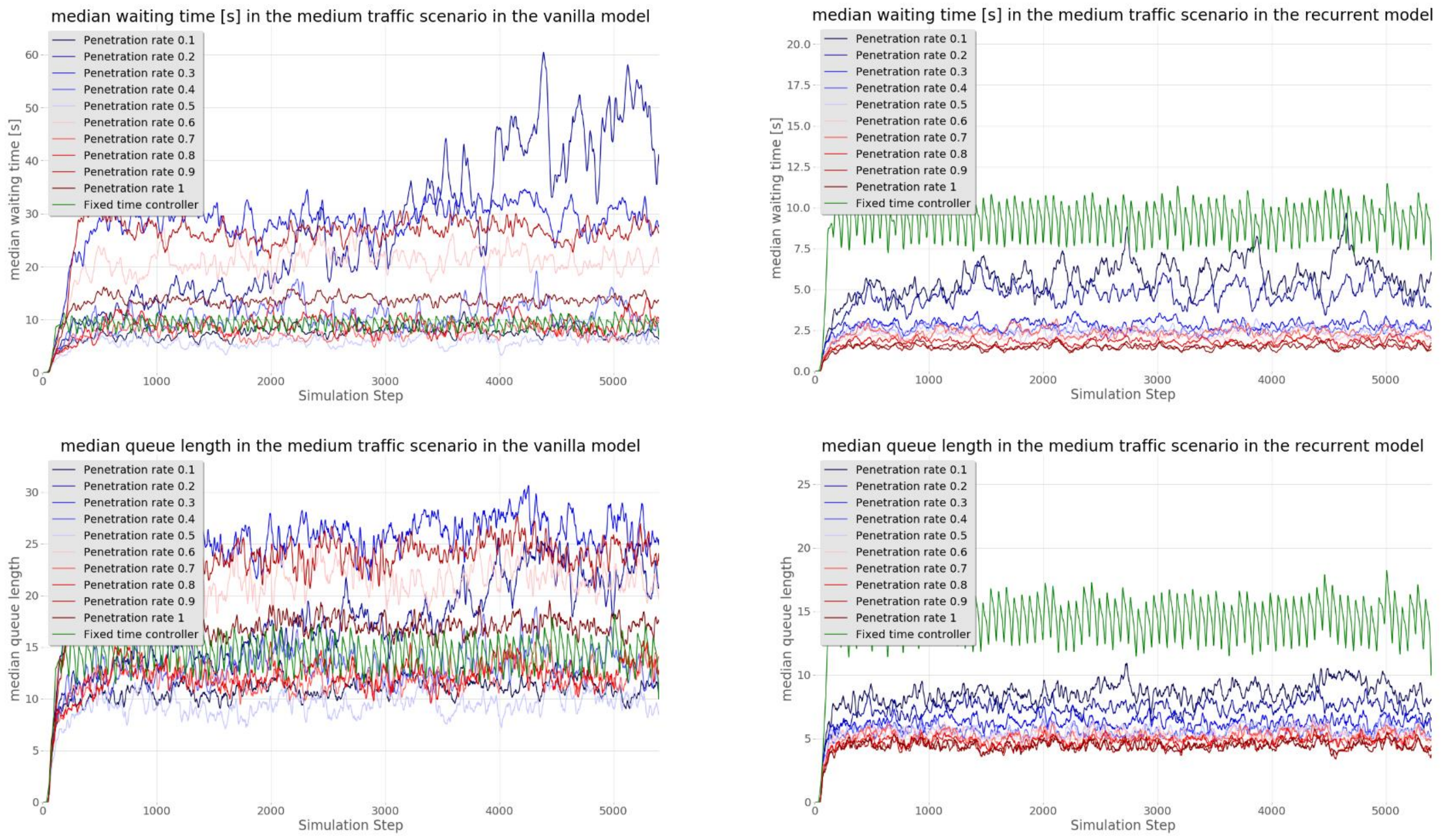


Figure 61 Median cumulative waiting times and delays in the medium traffic scenario for all penetration rates. Top left figure: median waiting times for the vanilla agent. Top right figure: median waiting times for the recurrent agent. Bottom left figure: median queue lengths for the vanilla agent. Bottom right figure: median queue lengths for the recurrent agent.

B.2.3 High traffic scenario

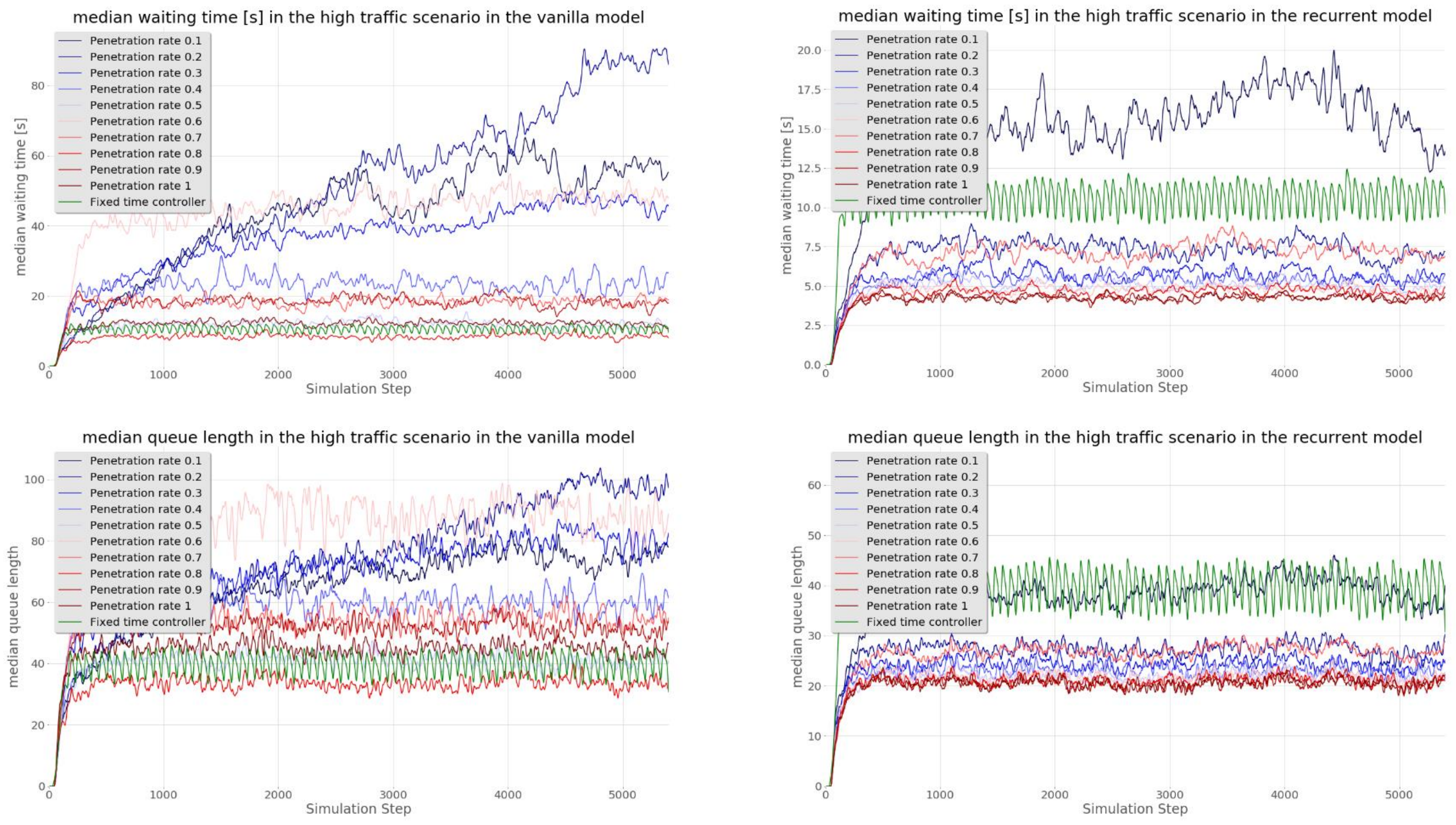


Figure 62 Median cumulative waiting times and delays in the high traffic scenario for all penetration rates. Top left figure: median waiting times for the vanilla agent. Top right figure: median waiting times for the recurrent agent. Bottom left figure: median queue lengths for the vanilla agent. Bottom right figure: median queue lengths for the recurrent agent.

B.2.4 Dynamic traffic scenario

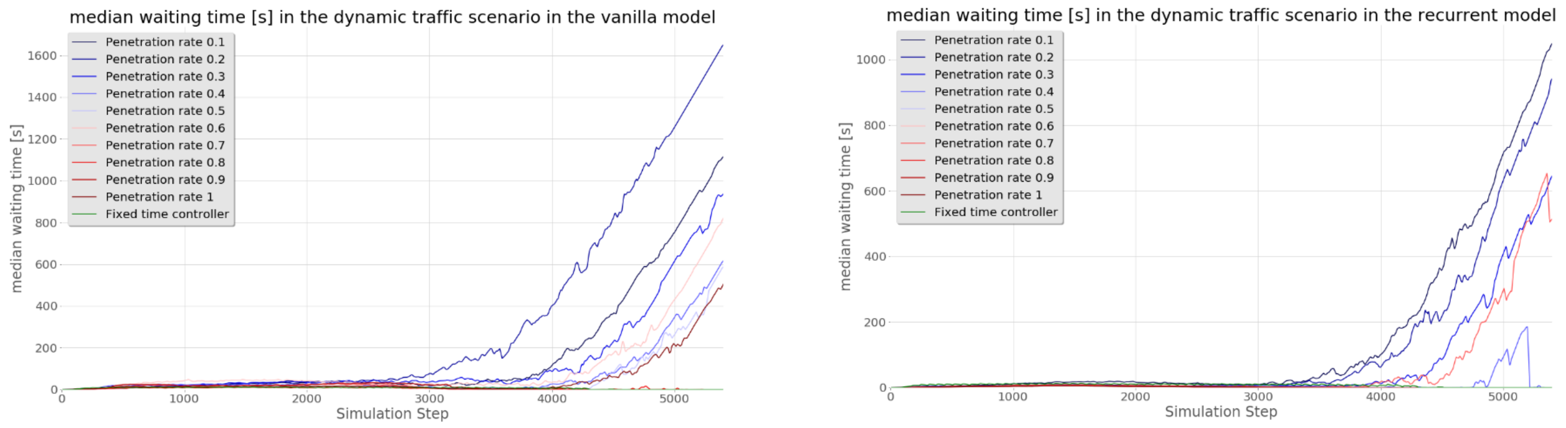


Figure 63 Median cumulative waiting times and delays in the dynamic traffic scenario for all penetration rates. Top left figure: median waiting times for the vanilla agent. Top right figure: median waiting times for the recurrent agent. Bottom left figure: median queue lengths for the vanilla agent. Bottom right figure: median queue lengths for the recurrent agent.

B.3 Comparison of vanilla and recurrent agents

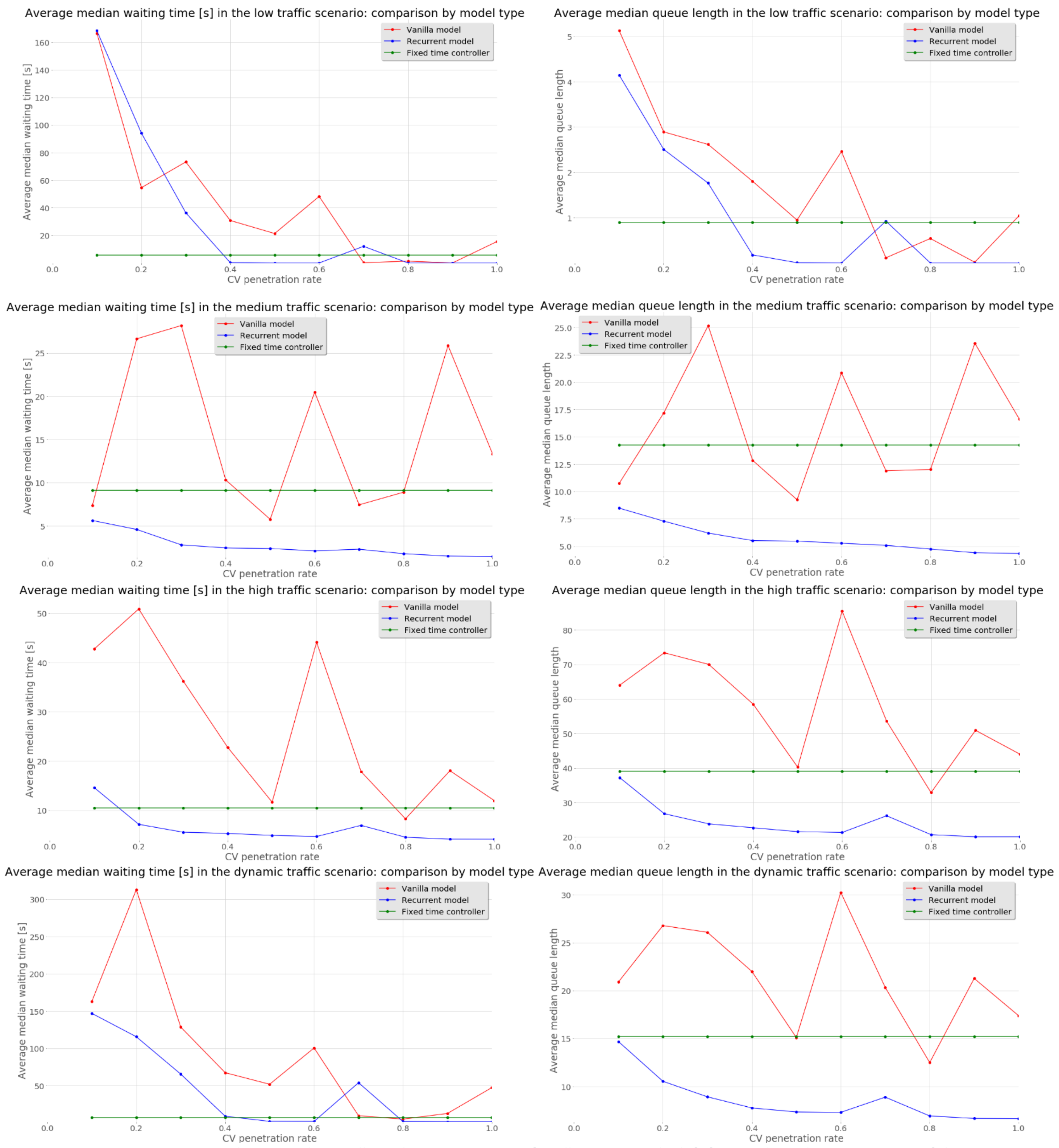


Figure 64 Comparison between the vanilla and recurrent agents for all scenarios. The left figures show the comparisons of the average median waiting times and the right figures show the comparisons for the average median queue lengths. From top to bottom, the figures show the results for the low, medium, high and dynamic traffic scenarios.

APPENDIX C: OVERVIEW OVER THE FOUND LITERATURE

Due to size limitations, the table could not be attached in this thesis. Instead it has been attached in a separate Excel sheet.