Traffic Flow Optimization using Reinforcement Learning

Master's Thesis

Erwin Walraven

Traffic Flow Optimization using Reinforcement Learning

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Erwin Walraven
born in Lisse, the Netherlands

Algorithmics Group
Department of Software and Computer Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.alg.ewi.tudelft.nl

Traffic Flow Optimization using Reinforcement Learning

Author:	Erwin Walraven
Student id:	4013166
E-mail:	`e.m.p.walraven@tudelft.nl`

Abstract

Traffic congestion causes unnecessary delay, pollution and increased fuel consumption. In this thesis we address this problem by proposing new algorithmic techniques to reduce traffic congestion and we contribute to the development of a new Intelligent Transportation System. We present a method to determine speed limits, in which we combine a traffic flow model with reinforcement learning techniques. A traffic flow optimization problem is formulated as a Markov Decision Process, and subsequently solved using Q-learning enhanced with value function approximation. This results in a single-agent and multi-agent approach to assign speed limits to highway sections. A difference between our work and existing approaches is that we also take traffic predictions into account. The performance of our method is evaluated in macroscopic simulations, in which we show that it is able to significantly reduce congestion under high traffic demands. A case study has been performed to evaluate the effectiveness of our method in microscopic simulations. The case study serves as a proof of concept and shows that our method performs well on a real scenario.

Preface

During the last year of my master studies, I had the opportunity to contribute to a research project within the Algorithmics group. The thesis that you are currently reading describes my contributions to this research project and to the development of Smoover, a new Intelligent Transportation System. In the project we applied reinforcement learning techniques to solve traffic related problems.

I would like to thank my fellow students in the student room on the 7th floor, with whom I spent a lot of time. Our numerous coffee breaks and the (real) Christmas tree we bought definitely improved the quality of our work. Secondly, I would like to thank Matthijs Spaan for being my supervisor and suggesting new ideas. Moreover, I would like to thank all the members of the Algorithmics group for the lunch breaks together, cake invitations and for making it a nice academic environment.

Erwin Walraven
Delft, the Netherlands
July 8, 2014

Contents

Preface **iii**

Contents **v**

List of Figures **vii**

1 Introduction **1**

1.1 Regulating traffic flow . . . 1
1.2 Intelligent Transportation Systems . . . 3
1.3 Smoover . . . 4
1.4 Objective and research questions . . . 6
1.5 Main contributions . . . 8
1.6 Outline . . . 8

2 Traffic Flow **9**

2.1 Traffic flow models . . . 9
2.2 Macroscopic simulation with METANET . . . 9
2.3 Microscopic simulation with SUMO . . . 13
2.4 Examples of control algorithms . . . 14
2.5 Summary . . . 16

3 Reinforcement Learning **17**

3.1 Framework . . . 17
3.2 Model-Free Reinforcement Learning . . . 20
3.3 Function approximation . . . 22
3.4 Summary . . . 26

4 Speed Limit Policies **27**

4.1 Problem description . . . 27
4.2 Formulation as Markov Decision Process . . . 28
4.3 Learning policies using Q-learning . . . 31

4.4 Experiments 32
4.5 Discussion 40
4.6 Summary 40

5 Improved Policy Learning 43
5.1 Function approximation with neural networks 43
5.2 Modified formulation as Markov Decision Process 46
5.3 Preserving the Markov property 49
5.4 Experiments 50
5.5 Discussion 58
5.6 Summary 58

6 Multi-Agent Traffic Control 61
6.1 Problem description 61
6.2 Formulation as Markov Decision Process 62
6.3 Experiments 64
6.4 Discussion 68
6.5 Summary 70

7 Probabilistic Policy Reuse 71
7.1 Motivation 71
7.2 Biased action selection 72
7.3 Reusing policies in Q-learning 73
7.4 Experiments 73
7.5 Discussion 75
7.6 Summary 76

8 Case study: Simulation A67 77
8.1 A67 simulation in SUMO 77
8.2 Automatic Incident Detection 78
8.3 Smoover control 79
8.4 Experiments 79
8.5 Discussion 82
8.6 Summary 82

9 Conclusions and Future Work 85
9.1 Main contributions 85
9.2 Future work 86

Bibliography 89

List of Figures

1.1 Regulating traffic flow. 2

2.1 Highway containing N sections. 10
2.2 Fundamental flow-density diagram. 13
2.3 SUMO: Simulation of Urban MObility. 14

3.1 Agent executing action a when environment is in state s. 18
3.2 Tile coding. 24

4.1 Example highway stretch. 27
4.2 Schematic representation of speed limit policy algorithm. 32
4.3 Traffic demands of the scenarios. 34
4.4 Policies for each individual scenario: comparison of policy quality. 36
4.5 One policy for multiple scenarios: comparison of policy quality. 38
4.6 Policy quality for different numbers of episodes. 39

5.1 Artificial neural network structure. 44
5.2 Neuron of an artificial neural network. 44
5.3 Artificial neural network with multiple layers. 45
5.4 Neural networks as function approximator. 46
5.5 Q-value as a function of stored states. 50
5.6 Policies for individual scenarios: policy quality (5000 episodes). 51
5.7 Policies for individual scenarios: policy quality (20000 episodes). 53
5.8 Running time comparison tile coding and neural networks. 54
5.9 Robustness for increasing noise rate on scenario 1, without predictions. 56
5.10 Robustness for increasing noise rate on scenario 1, with 5 minute predictions. . 57
5.11 Speed progress without control and with assigned speed limits. 59

6.1 Highway section i and its neighbors. 61
6.2 One policy for multiple scenarios: vehicle hours. 65
6.3 Learned policies with extended state description: vehicle hours. 66

6.4 Speed progress without control and with assigned speed limits. 67
6.5 Robustness for increasing noise rate on scenario 1. 69

7.1 Received rewards when learning a policy for scenario 1 by reusing existing policies that have been learned for other scenarios. 75
7.2 Probability distributions during first 2000 episodes. 76

8.1 Knooppunt De Hogt in OSM and SUMO. 78
8.2 Assignment of speed limits to matrix signs. 79
8.3 Smoover: vehicle hours for several participant percentages. 80
8.4 AID: vehicle hours. 81

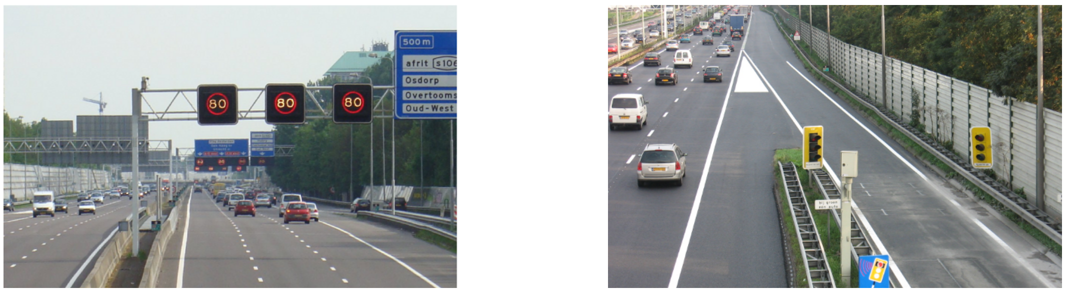
Chapter 1

Introduction

In modern society many people are faced with it every day: traffic congestion. When driving home after work, people encounter congested highways during their trip, which causes an increased travel time, but also more fuel consumption and pollution. A straightforward solution would be expanding the infrastructure to increase its traffic capacity, but due to space and budget limitations this is not always feasible in practice. Intelligent Transportation Systems (ITS) have emerged as a potential solution to improve highway efficiency. Designers of such intelligent systems aim to make traffic flow safer and more coordinated by using advanced technologies like floating car data, in-car information systems and mobile applications. These systems make it possible for Artificial Intelligence researchers to apply their techniques to the domain of traffic flow control and transportation in general. In this thesis project we contribute to Smoover, a new Intelligent Transportation System that is currently under development in the Netherlands. In this chapter we will give examples of Intelligent Transportation Systems, and we give more details about Smoover, which is part of a larger research project. At the same time, we identify and motivate the research questions that we attempt to answer. Eventually, this leads to our main contributions that will be presented in the remainder of this thesis.

1.1 Regulating traffic flow

In the last few decades, several techniques have been applied to regulate traffic flow on road networks. A technique that is well known in many countries consists of variable speed limits displayed by variable message signs or so-called matrix signs. Variable speed limits are used to reduce the speed on specific highway sections. In the seventies, the first matrix signs have been installed in the Netherlands, which are able to show speed limitations, as well as other symbols to indicate that it is not allowed to drive on a specific highway lane. Figure 1.1a shows matrix signs above the A10 highway in the Netherlands, displaying a speed limit of 80 kilometer per hour. These speed limits will be activated if the detected speed in downstream sections is below a predefined boundary value. Even though these speed limits may prevent the occurrence of traffic jams, the main purpose is accident prevention at the tail of traffic jams. Approaching cars are warned and decrease their speed in an earlier stage,



(a) Variable speed limits.

(b) Ramp metering.

Figure 1.1: Regulating traffic flow.

which reduces the possibility of accidents. Another type of message signs is the variable message sign. This is a traffic sign that is able to give car drivers warnings about events, congestion, alternative routes, expected travel times and other types of messages.

A second strategy to regulate traffic flow is ramp metering. In contrast to variable speed limits, ramp meters reduce the traffic demand to increase the speed and flow near on-ramps. Typically, a ramp meter consists of one or more traffic lights together with a controller. The controller decides how many cars are allowed to enter the highway, depending on the conditions of the road network. Figure 1.1b shows a ramp metering device near Delft in the Netherlands.

Applying control actions, such as activating ramp metering devices and assigning speed limits, does not always occur automatically. A decision support system can be used to support a traffic operator when making decisions given the current traffic situation and measurements. A traffic operator selects the control actions, but it is not obvious which control action should be applied and where it should be applied. In general this is a difficult task that requires expert knowledge and experience. A decision support system does not replace the traffic operator, but it is a complementary tool for analysis and supports the decision making procedure. A straightforward solution to determine the best possible control action is running simulations of all possible future control scenarios, but this is not tractable for online control due to the running time and the number of possible combinations.

An example of a decision support system is the multi-agent case based control system discussed in [24]. This system presents a ranked list of control actions to the traffic operator, based on the current traffic situation and an optimization criterion (e.g., minimizing travel time). The system does not run simulations to determine the best possible control actions, but uses a case base. Given the current traffic state, similar cases are selected from the case base, and the effects of a control action can be determined by taking the similarity between the selected cases and the current state into account. Eventually, the operator is able to select appropriate control actions from the presented alternatives based on experience and further assessment of the expected effects. New algorithms and control techniques may be integrated in such support systems and can be presented as one of the alternatives the operator can choose from.

1.2 Intelligent Transportation Systems

Intelligent Transportation Systems are systems that apply advanced communication, information and electronics technology in order to solve transportation problems [9]. They have the intention to solve problems related to traffic congestion, safety, transport efficiency and environmental conservation. A common property of ITS is that they try to embed several forms of intelligence in roads, vehicles and the information presented to car drivers.

ITS can be categorized into six categories [9]. Advanced Traffic Management Systems (ATMS) regulate traffic flow by responding to the currently measured traffic conditions. This means that an AMTS is able to interact with the road network and intervenes in real time, taking communication delays into account. Examples of technologies being used are video detectors, induction loop detectors, variable message signs and ramp metering devices. The main components of an AMTS are monitoring services, supported by devices such as cameras and other types of sensors. Information gathered by these systems is eventually used to regulate the traffic flow using message signs and ramp meters. The variable speed limits and ramp meters discussed in the previous section can be categorized as an ATMS.

A second category of ITS consists of Advanced Travelers Information Systems (ATIS). These systems aim to present information about the road network and traffic conditions to the car drivers, such that this information influences the car drivers when making decisions about their trip. For example, people may decide to choose an alternative route if their current route is affected by congestion. Intelligent systems are able to present several alternative routes to reach the destination. The information can be presented using message signs, as well as navigation systems and in-vehicle displays.

Commercial Vehicles Operation (CVO) systems try to make commercial vehicles and fleets safer and more efficient. For example, transportation companies and mail services may want to see locations of all their vehicles in real time to optimize their transport or delivery process. Such systems rely on GPS positioning and communication through modern communication networks like 4G. Typically, CVO systems are used by commercial companies to control their fleet and, at the same time, making their business process more cost efficient.

Advanced Public Transportation Systems (APTS) can be used to improve the efficiency of public transportation. An APTS uses technologies from ATMS and ATIS to, for example, present route information and real time information regarding changes. An example of a APTS system in the Netherlands is the central monitoring system for bus services in public transportation (e.g., Arriva, Connexxion). These systems can be used to locate vehicles, communicate with bus drivers and provide information about their schedule and delays. Information displays located at bus stops can be connected to the underlying database to provide real time information about expected departure times and disruptions.

Increasing driving safety by assisting car drivers can be important to react faster and more effectively in dangerous situations. Advanced Vehicles Control Systems (AVCS) use in-car sensors to gather information about the traffic situation around the car and vehicle conditions. Car manufacturers have already implemented technologies to keep the car in the right lane and to activate the brakes in case of quickly decreasing speeds of other vehicles.

In the near future people may be able to drive in autonomous cars. Google recently made significant progress in autonomous car driving, by means of a futuristic Google driverless car project. Their cars already have successfully shown to be able to drive autonomous in the urban area, but currently there should be at least one passenger to intervene in case of dangerous situations in which the car does not react appropriately. Early developments of guide systems for autonomous cars already started in the eighties [23].

The last category consists of Advanced Rural Transports Systems (ARTS), which can be used in rural zones to assist car drivers. This is relevant since such areas have a few alternative routes and typically navigational signs are missing.

The examples of ITS discussed above show that there are many applications of intelligent systems in transportation and road networks. The increasing bandwidth of communication networks and the ability to gather and process large amounts of information (i.e., big data) make it possible to create new innovative systems to make transportation more efficient. Researchers in the area of transportation have been investigating traffic phenomenons for years, resulting in mathematical models and methods to improve road network efficiency and transportation in general. Techniques from the domain of Artificial Intelligence have a huge potential to further accelerate this process and to build intelligent systems. In this thesis, we contribute to an innovative traffic management system that combines techniques from ATMS and ATIS with methods from AI and Computer Science in general. This will allow us to apply AI methods that have proven to be successful to a new application domain and, in the process, we attempt to reduce the gap between traffic flow theory and Artificial Intelligence. In the next section we introduce the newly proposed system and we identify its relationships with the ITS categories we just described.

1.3 Smoover

Smoover is a new Intelligent Transportation System that is currently under active development in the Netherlands. A consortium of five companies is working on this project until the end of 2014, together with the Algorithmics group at Delft University of Technology. The companies involved in this project are Locatienet-PTV BV, Adapticon BV, Cygnify BV, Tom van de Ven ITS Consultancy and Tessa Bouw Communicatie.

The project contributes to the Brabant in-car III (BIC III) program, which has the intention to solve concrete traffic flow problems using in-car technologies. These technologies should optimize the interactions between cars and trucks to further optimize traffic flow. The pilot area of BIC III is the A67 highway, located in the provinces of Noord-Brabant and Limburg in the Netherlands. This highway connects Belgium (A21) near Hapert with Eindhoven and the German border (A40) near Venlo.

Traffic congestion on the A67

High traffic demands often lead to an increased traffic flow and, as a consequence, traffic congestion arises. This is also the case on the A67 in the Netherlands. Transportation in a congested area makes it less efficient in terms of costs and time. This is a potential problem, because trucks from the Port of Antwerp with destination Germany mostly use the A67, resulting in cargo transport on this highway.

Traffic congestion arises if the flow exceeds the so-called critical flow. The critical flow depends on the characteristics of the road network, such as the maximum speed, the number of lanes and the behavior of individual car drivers. External factors that may affect this are weather conditions and accidents. On the A67 the on-ramps are relatively short. This may also trigger disruptions and irregular merging behavior. If the flow is below the critical flow, then it is called free flow.

Traffic demands can be balanced and regulated by giving speed advices to car drivers, with the intention to reduce or resolve traffic congestion. Another factor that may prevent the flow from exceeding the critical flow is lane changing guidance, which gives an advice regarding lane changing. A harmonized lane changing behavior may also decrease congestion and the probability of car accidents.

Shortcomings of existing solutions

Induction loops and matrix signs can be used to reduce congestion and prevent the occurrence of accidents. However, such systems have several shortcomings and disadvantages in practice. Induction loops and matrix signs are expensive to install and maintain, and in the Netherlands there are several highways without such systems. Localized disruptions and speed drops cannot be measured accurately by induction loops. This is caused by the fact that they measure average speed and flow, which is only representative for larger highway stretches. To resolve problems around on-ramps and off-ramps, induction loops and matrix signs are therefore not appropriate. Induction loops are also affected by failures, which in turn causes the data to be unreliable. Existing systems based on induction loops and matrix signs issue speed limitations based on speed measurements. This means that these systems are reactive, in the sense that they will only be activated if congestion is measured. However, it may be possible to reduce traffic congestion more effectively by using traffic predictions. Another shortcoming is that speed limitations and advices given by existing systems are intended to be applicable to all the vehicles, and they do not depend on vehicle type and their route. Personalized advices may be helpful to further reduce congestion.

We can conclude that a new ITS requires more information than the quantities measured by induction loops and speed limitations using matrix signs. Other types of information, as well as innovative technologies can be used to design a system that does not fully depend on matrix signs and data that comes from induction loops. Below we discuss opportunities and challenges for in-car solutions.

Opportunities and challenges for in-car technologies

Instead of focusing on centralized traffic flow control, in-car technologies can focus on the individual car driver. In the last decade, innovative technologies became more accessible to the general public. Most of the people have a smartphone with a GPS device to measure location. The coverage of modern communication networks (e.g., GPRS, 3G and 4G) is high and these networks are generally reliable enough to send and receive information in real time. This means that in-car technologies can be connected to a centralized system to transmit data. There is potentially a huge amount of data from which more personalized speed advices can be extracted, where the individual aspects and preferences of each car driver are taken into account. From this perspective, big data technology and machine learning techniques are relevant to come up with real time personalized speed advices that can be presented to the car driver by using in-car technologies.

Features of Smoover

Smoover aims to use cooperative in-car technologies to optimize traffic flow. The most important aspect of this system consists of personalized advices with respect to driving speed, lane changing and distance to preceding vehicles. Advices can be presented to the car driver on his or her smartphone or another display, which is attached to the dashboard. This display can also be used to show warnings in case of accidents, expected congestion or dangerous on-ramps.

The innovative aspect of Smoover is the integration of predictions of road conditions into speed advices. As a consequence, the advices can be used to reduce congestion, but may also prevent congestion. Data is gathered using in-car devices and detectors in the infrastructure, and subsequently processed using big data technology. Predictions of traffic conditions are based on techniques from the machine learning domain. Using this information, speed advices can be computed and communicated to the car drivers. All the features of Smoover are implemented in a smartphone application that will be publicly available for Android devices.

1.4 Objective and research questions

The objective of this thesis is to propose new algorithmic techniques to optimize traffic flow and reduce congestion, based on techniques from the Artificial Intelligence domain. These algorithms will not directly be implemented in the resulting ITS, but serve as a theoretical reference that can be demonstrated using simulations. The knowledge and concepts that follow from this research can be used in the development of Smoover, in which a generalized version of the presented algorithms will be integrated.

The research presented in this thesis is carried out in close cooperation with Cygnify BV, one of the companies that is part of the consortium. They have been working on traffic flow prediction for several years. This is the reason that the integration of traffic flow predictions is one of the goals of the project. They also provide other traffic related services, such as processing road maps and gathering historical traffic data. This data can be used to set up simulations. Based on the objectives of the Smoover project presented in the previous section, we aim to answer the main research question below.

Main Research Question. How can traffic congestion be reduced by algorithms to compute dynamic speed limits combined with traffic predictions?

To answer the main research question, we identified three research questions. They will be presented below and we discuss why they are relevant and how these questions can be answered.

Research Question 1. How can we devise an algorithm to determine speed limitations to reduce congestion?

To devise algorithms optimizing traffic flow, it is required to study which optimization methods exist and which mathematical objective functions can be optimized, in such a way that traffic congestion is reduced. Traffic predictions may be useful to control traffic flow more efficiently, so we need to study how predictive data can be combined with algorithms to determine speed limits. Before we can study optimization algorithms, we need a model of traffic flow. This leads to our second research question.

Research Question 2. Which computational models can be used to simulate traffic flow, taking speed limits into account, at low computational cost?

To create algorithms related to traffic flow, it is required to have models that can be implemented to represent traffic characteristics. Therefore, it is important to know which categories of simulation models exist, and which aspects determine their running time. Since we are interested in speed limits, it should be possible to integrate them in a traffic flow model. From a practical point of view, it is interesting to know how accurate simulations of traffic flow can be created and how such simulations correspond to real highways. This is reflected by the third research question.

Research Question 3. To which extent are the proposed methods applicable in practice?

Given an algorithm that is able to compute speed limits, it is interesting to investigate how speed limits would behave in practice and how they influence congestion. Therefore, it is important to know which data sources are required to set up a realistic highway simulation, and how this data can be combined with an existing simulation framework. To compare our work with existing methods, we need to study existing algorithms and control rules to assign speed limits to highways. From the perspective of Smoover, it is important to investigate how many participants are required to obtain the desired effects.

There are several things that make the topic relevant to study. In the traffic flow domain, little attention has been paid to the application of Artificial Intelligence methods to solve traffic related problems. Including predictive data into such methods is relatively new and may enable us to show how predictions are able to enhance these methods. This is covered by the research questions we described above.

1.5 Main contributions

Our main contributions can be summarized as follows. We present a method to reduce congestion, by generating policies using reinforcement learning algorithms. Reinforcement learning is an area of machine learning focusing on agents and how they interact with an environment. We show that our method can be enhanced with predictive data during the learning stage of the algorithms. We consider the single-agent case, in which one agent regulates the speed limits associated with a highway. We also present a generalization to the multi-agent case, in which multiple agents are able to assign speed limits to highways. We show that learned policies can be reused to enhance learning efficiency. The last part of the thesis involves a simulation using real traffic data, in which we present a proof of concept to show how policies can be applied in practical scenarios.

1.6 Outline

In this chapter we discussed the project this thesis contributes to, as well as the relevance of combining methods from Artificial Intelligence with traffic flow concepts. In this section we describe the topics of the remaining chapters, and their relationship with the research questions we identified in Section 1.4.

Traffic flow theory is the topic of Chapter 2. We give a brief comparison of microscopic and macroscopic traffic simulation models, and we present METANET. This is the macroscopic model we use in our speed limit algorithms. This answers research question 2.

In Chapter 3 we discuss background information regarding reinforcement learning algorithms. These algorithms will be used in Chapter 4, in which we present a reinforcement learning algorithm to calculate speed limitations. We formulate the problem as an optimization problem in terms of a Markov Decision Process. Furthermore, we show that we can use Q-learning to find policies to resolve or reduce congestion. A further improvement that we propose is the integration of model-based traffic predictions. Experiments show that the algorithm enhanced with traffic predictions is able to produce better policies. This answers research question 1 partially. Based on our initial experiments, we are able to improve the speed limit algorithm significantly. An improved version of the algorithm is presented and evaluated in Chapters 5 and 6.

In Chapter 7 we show that previously learned policies can be reused as a probabilistic bias in the exploration phase of the reinforcement learning algorithm from Chapter 5. This improves the learning capabilities of the algorithm and enables us to generate policies for new traffic scenarios more efficiently. This also relates to the first research question.

Chapter 8 presents a case study, in which we demonstrate that the proposed algorithms may reduce congestion in practical scenarios as well. This serves as a proof of concept and answers research question 3. The policies generated by the algorithms from Chapter 6 are applied to a real road network in a microscopic simulation. For this purpose, a partial map of the A67 is translated to the macroscopic model from Chapter 2. Then we use the algorithms from Chapter 6 to learn policies, which can subsequently be tested in more realistic scenarios.

Chapter 2

Traffic Flow

In this chapter we provide the theoretical background of traffic flow modeling and related approaches for traffic simulation. We briefly introduce traffic models in general and we give a description of METANET, the macroscopic model we use in this thesis. We also present microscopic simulation environments. In particular, we introduce SUMO, an open source simulation package for traffic flow. This chapter answers research question 2 and is important, since our algorithms will interact with a traffic flow model, and eventually our algorithms can be evaluated through simulation. The background knowledge we present is important to understand the basic idea of the algorithms we present in the remaining chapters.

2.1 Traffic flow models

Two categories of traffic flow models can be distinguished: microscopic and macroscopic. Microscopic models define the behavior of traffic flow for individual vehicles, in terms of their speed, position and the characteristics of the vehicle itself (e.g., maximum speed, acceleration). Such models make it possible to set up an accurate simulation of traffic flow. Unfortunately, they have high computational cost, which makes it infeasible to use them for several algorithmic applications. Macroscopic models, on the other hand, model traffic flow using average speed, flow and density of highway sections. They are relatively easy to implement and have a low time complexity, because the number of calculations involved is fixed and does not depend on the number of vehicles on the highway. Moreover, the analytical aspects of such models make them suitable for the design of traffic control systems [16]. The most important differences between microscopic and macroscopic traffic models are shown in Table 2.1.

2.2 Macroscopic simulation with METANET

In this thesis we use the METANET traffic flow model, proposed by Messner and Papageorgiou [18, 16]. This model defines highway traffic in terms of flow, speed and density. Flow is defined as the number of vehicles that pass per time unit. Flow can be measured with loop

	Microscopic	Macroscopic
Quantities	Position, speed and vehicle characteristics (e.g., acceleration, emission).	Flow, density and speed of highway sections.
Computation time	High, since several quantities are calculated for individual vehicles	Low. For instance, polynomial in the number of simulation time steps and number of sections.
Accuracy	At the level of individual vehicles.	At the level of highway sections.

Table 2.1: Comparison of microscopic and macroscopic models.

detectors that count the number of vehicles in a time period of several minutes. The speed is an average of the speed of the vehicles that pass a specific point of the highway. The density is the number of vehicles per unit length. This value can be approximated using the flow, the average speed and the number of lanes of a highway section. An alternative model that could be used is the Cell Transmission Model [7]. This model is based on kinematic wave equations, and is widely used as a numerical method for traffic simulations. However, it assumes a fixed section length and has no support for multiple driving lanes. This was one of the most important reasons to choose METANET, which can also be adapted to take speed limitations into account.

Now we describe the METANET model in more detail. The formulas we present are based on an extension, proposed by Karaaslan et al. [15]. Additional variables were introduced in [32] to support multiple lanes. We consider the uni-directional highway of *N* sections in Figure 2.1 below. The arrows indicate the direction of vehicle flow. Table 2.2 shows the variables of the model, where *n* denotes the number of the current simulation time step. The meaning of the variables is important, since we will use them in the description of our algorithms in the next chapters.

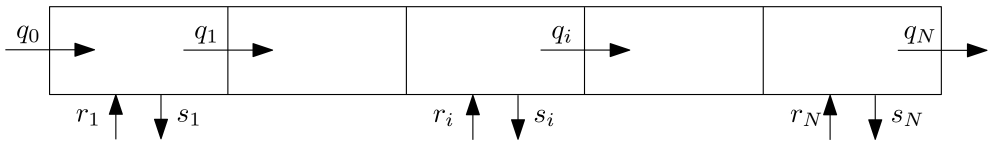


Figure 2.1: Highway containing *N* sections.

Variable	**Description**
T	time step size (hour)
M_i	number of lanes in section i
L_i	length of section i (km)
$k_i(n)$	density in section i at time nT (veh/km/lane)
$v_i(n)$	mean speed of vehicles in section i at time nT (km/hour)
$q_i(n)$	traffic volume leaving section i and entering $i+1$ at time nT (veh/km/lane)
$r_i(n)$	on-ramp traffic volume of section i at time nT (veh/hour)
$s_i(n)$	off-ramp traffic volume of section i at time nT (veh/hour)

Table 2.2: Variables of the METANET model.

The equations below describe the macroscopic traffic model for a uni-directional highway ($i = 1, \ldots, N$).

$$q_i(n) = \alpha k_i(n)v_i(n) + (1-\alpha)k_{i+1}(n)v_{i+1}(n), \tag{2.1}$$

$$k_i(n+1) = k_i(n) + \frac{T}{M_i L_i}\left[M_{i-1}q_{i-1} - M_i q_i(n) + r_i(n) - s_i(n)\right], \tag{2.2}$$

$$\begin{aligned} v_i(n+1) = v_i(n) + \frac{T}{\tau}\left\{V_e\left[k_i(n)\right] - v_i(n)\right\} + \frac{T}{L_i}\cdot\frac{k_{i-1}(n)}{k_i(n+1)+\kappa'} \\ \cdot v_{i-1}(n)\left[\sqrt{v_{i-1}(n)v_i(n)} - v_i(n)\right] - \frac{\mu(n)T}{\tau L_i}\cdot\frac{k_{i_1}(n) - k_i(n)}{k_i(n)+\kappa}, \end{aligned} \tag{2.3}$$

$$\mu(n) = \begin{cases} \mu_1 \frac{\rho}{k_{jam} - k_{i+1}(n) + \sigma} & \text{if } k_{i+1}(n) > k_i(n) \\ \mu_2 & \text{otherwise} \end{cases}.$$

In these equations, $\alpha, \rho, \sigma, \kappa, \kappa', \tau, \mu_1$ and μ_2 are positive parameters and k_{jam} is the jam traffic density. For this density, the average speed approaches zero. The function V_e describes the fundamental relationship between speed and density and is defined as:

$$V_e(k) = v_f\left[1 - (k/k_{jam})^l\right]^m \tag{2.4}$$

where $l > 0$, $m > 1$ are a constant and v_f is the free flow speed. There will be two virtual sections 0 and $N+1$ which are used to define the boundary conditions of the model:

$$\begin{aligned} v_0(n) &= v_1(n) \\ k_0(n) &= k_1(n) \\ v_{N+1}(n) &= v_N(n) \\ k_{N+1}(n) &= k_N(n) \end{aligned} \tag{2.5}$$

Section 0 represents a queue of vehicles that can be filled if vehicles are unable to enter the highway. If there is no congestion, then the queue will be empty. Otherwise, the queue will be filled with vehicles, depending on the demanding traffic volume and the remaining capacity of section 1. The queue length $w_0(n+1)$ is defined as:

$$w_0(n+1) = w_0(n) + M_1 T\,[q_d(n) - q_0(n)] \tag{2.6}$$

where q_d is the demanding traffic flow and q_0 is the flow from section 0 to section 1, which is defined below.

$$q_0(n) = \min\left\{ q_d(n) + \frac{w_0(n)}{M_i T}, \alpha k_0(n) v_0(n) + (1-\alpha) k_1(n) v_1(n) \right\} \tag{2.7}$$

The on-ramp of section i is also modeled as a queue. This queue can be used as a buffer in case of congestion, or if a ramp metering strategy is applied to regulate the incoming flow. The definition of the on-ramp queue and the traffic volumes of the on-ramp and off-ramp are given below.

$$\begin{aligned} w_i(n+1) &= w_i(n) + T\lfloor r_{d,i}(n) - r_i(n) \rfloor && (2.8)\\ r_i(n) &= r_{d,i}(n) + \frac{w_i(n)}{T} \\ s_i(n) &= \beta_i q_{i-1}(n) \end{aligned}$$

In these equations, β_i is a non-negative constant, $r_{d,i}$ and r_i are the demand flow and realized flow of section i. The values w_i, r_i, $r_{d,i}$ are zero if section i does not contain an on-ramp. Similarly, if section i has no off-ramp, then β_i will be equal to zero.

There are a few additional boundary conditions that need to be imposed to ensure that the quantities flow, density and speed will be calculated correctly. In most of the literature they are not mentioned explicitly, but they are relevant for implementing the model. Firstly, the condition $L_i > v_f \cdot T$ should hold for all sections i ($i = 1, \ldots, N$), because otherwise vehicles can pass through a complete road section within one simulation time step, which makes the macroscopic simulation inaccurate. The second problem we need to take care of is limitation of incoming flow, to prevent the density from exceeding k_{jam}. A simple solution to this problem is calculating the flow values backwards (i.e., from section N to 1). For each section i the outgoing flow can be determined, based on the number of vehicles that is allowed to enter section $i+1$.

Note that the characteristics of traffic flow depend on several parameters. They can be calibrated to fit real observations from a highway stretch. Unless stated otherwise, we will use the parameters shown in Table 2.3. Most of the parameters have been derived from [32] and correspond to a Californian highway. The only exception is the free flow speed v_f, which is increased to ensure that higher speed values are possible within the macroscopic simulation.

The relationship between the quantities flow, speed and density is illustrated in Figure 2.2. If the density is below the critical density, then small perturbations will resolve

v_f	k_{jam}	l	m	α	κ'	κ	μ_1	μ_2	τ	σ	ρ
130	110	1.86	4.05	0.95	4	40	12	6	20.4	35	120

Table 2.3: Parameters of the METANET model.

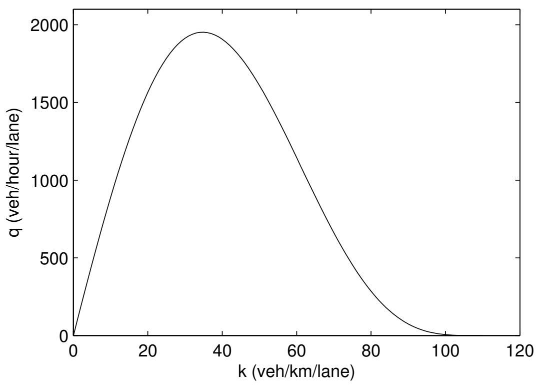


Figure 2.2: Fundamental flow-density diagram.

quickly. In this example, the critical density is slightly below 40 vehicles per kilometer per lane. If the density is close to this critical density, then the highway reaches its maximum flow capacity. A further increase of the density will cause congestion and the average flow and average speed of the vehicles drop immediately. From Figure 2.2 we can conclude that if we regulate the density of a highway segment, we can ensure that the highway efficiency is preserved. For example, a control strategy may regulate the traffic such that the density stays close to the critical density, in order to maximize the traffic flow.

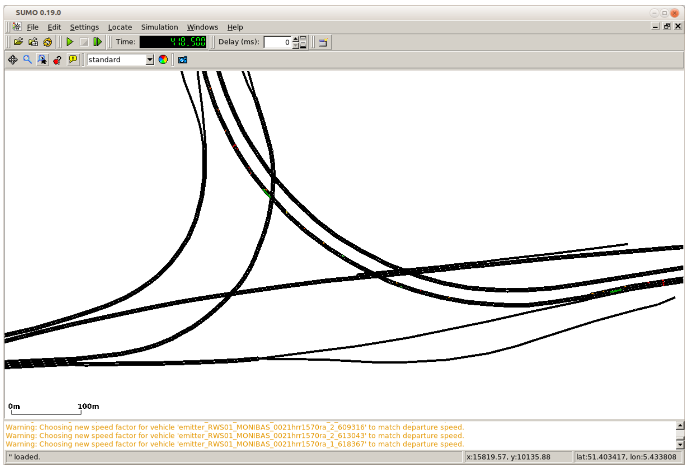
2.3 Microscopic simulation with SUMO

SUMO is an open source software package for traffic simulation, originally developed by the German Aerospace Center (DLR) [2]. The first open source version was released in 2002. The main reason to create an open source simulation environment is that it allows everyone to implement their algorithms in an existing simulation environment. We use SUMO in our work because other people working on the Smoover project use the same simulation environment. For more details regarding SUMO and its relation to the Smoover project we refer to [17].

The SUMO simulation suite consists of *sumo*, which actually performs the traffic simulation. It requires a road network and traffic demand pattern as the input, in a specific format. These files can be generated using the applications *netgen* and *netconvert*, which are also part of SUMO. It is also possible to import road networks from other simulators (e.g., Vissim) and OpenStreetMap. Using these tools, real road networks can easily be converted to the right format to be used in simulations.

SUMO is a microscopic simulation environment and therefore it maintains a representation of each individual vehicle. Vehicles have their own route through the network and also have their own departure time, position and speed. Loop detectors and traffic lights can be placed at specific places to mimic the equipment of real road networks. Figure 2.3 shows a screenshot of road interchange De Hogt near Eindhoven, modeled in SUMO.

In Chapter 8 we will use SUMO to perform a microscopic simulation to evaluate the algorithms presented in Chapter 4 to 6. Although SUMO is assumed to be a realistic simula-



```
Warning: Choosing new speed factor for vehicle 'emitter_RWS01_MONIBAS_0021hrr1570ra_2_609316' to match departure speed.
Warning: Choosing new speed factor for vehicle 'emitter_RWS01_MONIBAS_0021hrr1570ra_2_613043' to match departure speed.
Warning: Choosing new speed factor for vehicle 'emitter_RWS01_MONIBAS_0021hrr1570ra_1_618367' to match departure speed.
" loaded.    x:15819.57, y:10135.88    lat:51.403417, lon:5.433808
```

Figure 2.3: SUMO: Simulation of Urban MObility.

tion environment for traffic flow, it is worth noting that it cannot capture all the dynamics of traffic flow and the behavior of car drivers exactly. Several car following models have been implemented in SUMO, trying to mimic real car following behavior. For more information regarding SUMO and its features we refer to the overview in [2].

2.4 Examples of control algorithms

There exist several control approaches and optimization algorithms based on the METANET model. The objective of this thesis is to propose new techniques combining concepts of traffic flow and Artificial Intelligence, but for the sake of completeness we briefly discuss a few other control strategies in this section.

Model Predictive Control

Model Predictive Control (MPC) is a method to control complex dynamic processes and has a wide range of applications. MPC algorithms use a model of the system under consideration to find optimal control signals, taking the future behavior of the system into account. MPC methods are suitable to control systems in which prediction is an essential aspect [11]. According to [11], there are two main advantages over non-predictive control. Firstly, MPC can ensure that the output of a system will follow a trajectory to a certain goal. This is useful for systems in which the goal is not the only important aspect, but also the trajectory to reach the goal is relevant. The second advantage is that MPC methods inherently make a trade-off between immediate performance and future performance. For some systems it may be necessary to take decisions that decrease the current performance, but give a better performance in the future. For example, when optimizing traffic flow, it is relevant to optimize with respect to a future traffic scenario or expected traffic scenario. We do not give the formal details behind Model Predictive Control. Below we give a high level description of the MPC scheme [11].

1. **Prediction.** The current state of the system, expected external influences and a planned control signal are used to predict the behavior of the considered system in the future. For traffic flow, this involves the evaluation of a model to predict the future road conditions.

2. **Performance evaluation.** An objective function is used to evaluate the performance of the system under the planned control signal from the prediction phase. The number of vehicle hours can be used as a performance metric when optimizing traffic flow.

3. **Optimization.** In an optimization step, the optimal control signal is found. This control signal optimizes the objective function for the chosen time horizon of the prediction phase.

4. **Control action.** Given the optimal control signal from the previous step, the next control action is taken from the optimal control signal and subsequently applied to the system. This means that the optimal control signal is calculated every control time step, but only the first action is taken. Recalculating optimal control signals proceeds using a rolling horizon scheme. Typically, the control time step size is smaller than the time horizon of the prediction phase.

MPC can be used for optimal coordination of variable speed limits. Hegyi et al. present an algorithm based on MPC to determine optimal speed limits for highway sections [12]. The prediction model they use is METANET and the objective function is defined in terms of the total number of vehicle hours people spend on the highway. The reason behind this decision is that the optimal control signal (i.e., the speed limitations issued) needs to minimize the additional time people spend as a consequence of congestion. The objective function also includes a penalty term for large variations of the speed limits. Optimizing the objective function is performed through sequential quadratic programming, an algorithm for constrained nonlinear optimization. Experiments have shown that the proposed MPC approach achieves an improvement of about 20 percent for simple scenarios.

Ramp metering using ALINEA

ALINEA is a policy to locally control ramp metering devices [21]. It tries to maximize the total throughput by controlling the density at a downstream section of the on-ramp. In order to be applied to ramp metering control, ALINEA requires a loop detector at a downstream section of the highway to measure the density. A ramp metering rate $r(t)$ is determined for the time interval from t to $t+\Delta t$, according to the following update rule:

$$r(t) = \bar{r}(t-\Delta t) + K_R \cdot (O^* - O(t))$$

where $O(t)$ is the measured density, O^* is the desired density and $\bar{r}(t-\Delta t)$ is the measured ramp metering rate and K_R is a regulation parameter. The desired density should be less than the critical density. In the example of Figure 2.2, the desired density would be slightly below 40. Real world experiments have shown that 70 is a good choice for K_R and a loop detector should be located downstream between 40 meters and 500 meters from the on-ramp [5]. ALINEA can be combined with METANET to evaluate the performance of ramp metering devices and its influence on the total throughput.

Rules for variable speed limits

Instead of explicitly optimizing an objective function using advanced algorithmic techniques, there are also more simple intuitive rule based systems to determine speed limits. Zhang et al. propose a control system for highway traffic flow, in which the speed limit of a section is used as a virtual ramp meter for the next section [32]. The system is based on the fundamental flow-density relationship, as shown in Figure 2.2, and implemented using METANET. Experiments have shown that this system is able to reduce the number of vehicle hours by 28 percent for simple scenarios. An advantage of this method over MPC, is that it requires less computation time and there is a limited number of parameters to be adjusted.

2.5 Summary

In this chapter we discussed the difference between microscopic and macroscopic traffic flow models. The main difference is that microscopic models simulate traffic flow at the level of individual cars, while macroscopic models define traffic flow using average speeds, flows and densities of highway sections. An example of a microscopic simulation environment is SUMO, which is assumed to be able to perform realistic traffic simulations. For macroscopic simulation, we presented the METANET formulas to calculate speeds, flows and densities in discrete time steps. From the perspective of running time, microscopic simulation is much more expensive. To optimize traffic flow there exist several control strategies. We briefly discussed Model Predictive Control, ALINEA and a rule based algorithm for speed limitations. The macroscopic model we discussed will be integrated in the algorithms we present in the remaining chapters, and the microscopic simulation environment can be used to run simulations of a more realistic highway. The concepts and models discussed in this chapter form an answer to research question 2, since we can use METANET to run a traffic simulation at low computational cost. Speed limits can also be integrated in this traffic flow model.

Chapter 3

Reinforcement Learning

In this chapter we introduce reinforcement learning. This is a branch of machine learning which studies systems that are able to learn from data. Arthur Samuel, a pioneer in AI, defined machine learning as a "field of study that gives computers the ability to learn without being explicitly programmed" [25]. A more formal definition of machine learning was given by Tom M. Mitchell. He stated that "a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [19].

Reinforcement learning deals with agents interacting with their environment to maximize a cumulative reward. In reinforcement learning, the agent *learns* how to execute actions in the environment, when there is only limited feedback in terms of a single reward that an agent receives after executing an action.

Typically, the environment is modeled as a Markov Decision Process (MDP), which defines a set of states and the actions that can be executed to control the environment. Practical problems, such as automated robot control and traffic light assignment problems, have been successfully formulated as an MDP. Currently it is a popular framework for modeling sequential decision making problems.

This chapter covers the background of modeling decision making problems as a Markov Decision Process and shows how an agent is able to learn to act in an unknown environment. Section 3.1 provides the theoretical background of the MDP framework and in Section 3.2 we discuss model-free learning techniques that can be applied if the model of the MDP is unknown. The first part of this chapter follows roughly the same structure and notation as the introduction to reinforcement learning in [31].

3.1 Framework

In this section we formalize the notion of a Markov Decision Process (MDP) [22]. We only consider MDPs with a finite number of states and actions. These finite MDPs are extensively used in reinforcement learning and therefore this section presents the required background knowledge to understand the theory behind reinforcement learning algorithms. We will discuss the formal definition of an MDP, policies, the optimality of policies and we introduce several value functions. These value functions will be used to represent policies and define the optimality of policies.

Markov Decision Processes

As mentioned in the introduction of this chapter, a Markov Decision Process can be used to describe the environment of an agent. The environment is represented by a state $s \in S$, where S is a finite set containing n states. Actions can be executed to change the state of the environment. We define A as a finite set containing m actions and $A(s)$ denotes the set of actions that can be executed in state $s \in S$.

If the system is in a state $s \in S$, then an action can be executed to make a transition to another state $s' \in S$. There is a probability distribution that defines the probability to make a transition from s to another state s'. This is represented by a transition function $T : S \times A \times S \rightarrow [0,1]$. For this transition function it should hold that $T(s,a,s') \geq 0$ and $T(s,a,s') \leq 1$, for all $s,s' \in S$ and $a \in A$. Furthermore, for all states s and actions a, the probability distribution defining the transitions to the next states should be proper, which means that $\sum_{s' \in S} T(s,a,s') = 1$. The system is said to be *Markovian* if the outcome of an action only depends on the previous state and action: $P(s_{t+1}|s_t,a_t,s_{t-1},a_{t-1},\ldots) = P(s_{t+1}|s_t,a_t) = T(s_t,a_t,s_{t+1})$. This is also called the *Markov property*.

An agent executing actions in an environment receives rewards for transitioning to a state after executing an action. This reward is specified by the reward function $R : S \times A \times S \rightarrow \mathbb{R}$. It is important to note that reward may be a bit misleading in this context, since a negative reward can also be interpreted as a punishment received by the agent. In general, the reward is a single scalar value that the agent receives as feedback from the environment. In Section 3.2 we will see that the reward function can be used to guide the agent in the direction of a predefined goal when solving an MDP. The concepts introduced in this section are summarized in Definition 1 below.

Definition 1 (Markov Decision Process)**.** *A Markov Decision Process (MDP) is a tuple $\langle S,A,T,R \rangle$, where $S = \{s_1,\ldots,s_n\}$ is a finite set of states and $A = \{a_1,\ldots,a_m\}$ a finite set of actions. The function $T : S \times A \times S \rightarrow [0,1]$ defines the transition function and the reward is represented by the reward function $R : S \times A \times S \rightarrow \mathbb{R}$.*

A schematic representation of an agent interacting with an environment modeled using the MDP formalism is shown in Figure 3.1. The agent executes action a in state s and receives reward $R(s,a,s')$ after arriving in state s'.

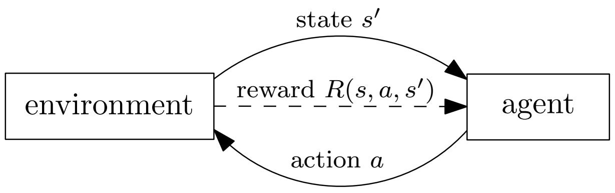


Figure 3.1: Agent executing action a when environment is in state s.

Policies and optimality

Agents use policies to determine which action is executed. Such a policy π is a mapping from states to actions: $\pi : S \to A$. Executing actions proceeds in an iterative fashion. Given an initial state s_0, the first action $a_0 = \pi(s_0)$ is executed and the transition from state s_0 to state s_1 is made. The agent receives a reward $r_0 = R(s_0, a_0, s_1)$. This process may continue infinitely, or stops if a goal state has been reached. A natural question that arises is how we can define the optimality of policies and which criterion should be optimized when finding policies. In this thesis we consider infinite-horizon optimality, which means that all rewards received by the agent are taken into account. Rewards received will be discounted by a discount factor γ (with $0 \leq \gamma < 1$), such that rewards received in the near future are more important than rewards received later. Therefore, optimal policies should maximize the expected sum of discounted rewards, as shown below. In this equation, r_i denotes the reward received by the agent, and it does not relate to the METANET traffic volumes.

$$E\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots\right] = E\left[\sum_{t=0}^{\infty} \gamma^t r^t\right] \tag{3.1}$$

Alternative optimality criteria are the finite horizon model without discounting and the average reward model. More information regarding these models can be found in [31]. In the section below we discuss the correspondence between policies and the discounted infinite-horizon optimality model.

Value functions

Now we introduce two types of value functions: a state value function and a state-action value function. They represent *how good* it is for an agent to be in a state, or to execute an action. This is expressed in terms of the discounted infinite-horizon optimality model we described above. The value of a state s under policy π is denoted by $V^\pi(s)$ and represents the expected return when the agent starts in state s and subsequently follows policy π. The definition is given below, where E_π denotes the expected value under policy π.

$$\begin{aligned} V^\pi(s) &= E_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s\right] \\ &= E_\pi\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s\right] \\ &= E_\pi\left[r_t + \gamma V^\pi(s_{t+1}) | s_t = s\right] \\ &= \sum_{s' \in S} T(s, \pi(s), s')\left(R(s, a, s') + \gamma V^\pi(s')\right) \end{aligned} \tag{3.2}$$

Equation 3.2 is known as the Bellman equation for $V^\pi(s)$ and describes the relationship between the value of a state and the value of the successor states. An optimal policy π^* is a policy that satisfies the equation below.

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s')\left(R(s, a, s') + \gamma V^*(s')\right) \tag{3.3}$$

For an optimal policy π^* it holds that $V^{\pi^*}(s) \geq V^{\pi}(s)$ for all $s \in S$ and all policies π. Now we can define the optimal policy π^* in terms of V^* as follows:

$$\pi^*(s) = \operatorname*{argmax}_{a \in A} \sum_{s' \in S} T(s,a,s') \left(R(s,a,s') + \gamma V^*(s')\right) \tag{3.4}$$

The algorithms we discuss in Section 3.2 use another representation of the state value function: the state-action value function $Q^{\pi}(s,a)$. It is the expected reward when the agent is in state s, executes action a and follows policy π afterwards. Similar to the derivations of the state value function, the optimal state-action value function is presented below.

$$Q^*(s,a) = \sum_{s' \in S} T(s,a,s') \left(R(s,a,s') + \gamma \max_{a' \in A} Q^*(s',a') \right) \tag{3.5}$$

The optimal value function and policy can also be expressed in terms of the Q-function: $V^*(s) = \max_{a \in A} Q^*(s,a)$ and $\pi^*(s) = \operatorname{argmax}_{a \in A} Q^*(s,a)$. An important property is that if we define the optimal policy in this way, we do not need to use the transition function T and reward function R explicitly. The Q-function is sufficient to determine the next action to execute. In the next section we discuss algorithms that learn a policy represented by a Q-function, instead of learning the state value function V. These techniques do not need the model of the MDP (i.e., the transition function and reward function) to learn a policy.

3.2 Model-Free Reinforcement Learning

In this section we give an overview of the most important model-free learning techniques for Markov Decision Processes. They learn a policy based on the experiences, by repeatedly observing rewards after applying actions. We make the distinction between two different algorithms. We present the Q-learning algorithm, which is an off-policy algorithm to learn the Q-function. Then we discuss SARSA, which is an on-policy learning algorithm.

Q-learning

The first algorithm we discuss is the Q-learning algorithm [28, 29]. This algorithm is able to learn the Q-values, which represent a policy, without using the model of the MDP. It only uses the rewards received from the environment to update the estimates of the Q-function. Therefore, it is called *model-free* learning. The update rule used by Q-learning is shown in Equation 3.6. Recall from Section 3.1 that the Q-function is sufficient to determine the next action to execute. This means that learning Q-values also results in a policy.

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_t + \gamma \max_{a \in A} Q(s_{t+1},a) - Q(s_t,a_t)] \tag{3.6}$$

The Q-learning algorithm estimates the Q-function by executing actions and observing rewards. The algorithmic description is given in Algorithm 1 below.

The Q-learning algorithm is an off-policy algorithm and actions may be selected according to a different policy than the policy being learned. If each state-action pair is encountered an infinite number of times, then Q-learning converges to an optimal policy [29].

Algorithm 1: *Q*-learning

```
input  : initial state $s$, learning rate $\alpha$, discount factor $\gamma$
output: $Q$-function, representing a policy
$Q(s,a) = 0 \quad \forall s \in S, \forall a \in A$
foreach episode do
    foreach step of the episode do
        choose action $a$ based on policy derived from $Q$ using an exploration strategy
        execute $a$, observe reward $r$ and new state $s'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$
```

In reinforcement learning, it is a common problem to balance exploration and exploitation. Exploration is useful to discover parts of the state space that have not been encountered before, but existing knowledge can also be exploited to receive more reward. It is important to balance exploration and exploitation, which we will illustrate with a simple example. Consider a path finding problem in the direction of a certain goal. After finding a good path in the direction of the goal, there may be better paths leading to the same goal but giving more reward. The agent will never find these paths if it does not explore the environment. On the other hand, the agent does not learn to exploit learned knowledge when there is no exploitation involved. Balancing exploration and exploitation is an extensively studied problem in the literature. In this thesis, we use the ε-greedy exploration strategy. This strategy chooses the action with maximum Q-value (i.e., greedy) with probability ε, and selects a random action otherwise. The algorithm executes several episodes, starting from an initial state, and the probability to select random actions is decreased over time, such that it starts to follow the policy that is currently learned. An episode ends if a final state has been reached, or if a predefined number of actions has been executed.

SARSA

The SARSA algorithm is, in contrast to Q-learning, an on-policy algorithm and follows the same policy as the policy that is used by the agent. The structure of the algorithm is the same as Q-learning, but SARSA uses a different update rule:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$$

The action a_{t+1} is selected using the policy followed, starting from state s_{t+1}. If all state-action pairs are visited an infinite number of times, then the policy will converge to an optimal policy and eventually this policy will be the greedy policy [27, 26]. After that, exploration does not occur anymore.

3.3 Function approximation

In Section 3.1 we introduced the notion of a value function and its role in the temporal difference learning algorithms. An intuitive representation of a value function is a table with a value for each state, or for each state-action pair. This representation causes problems once the state and action spaces become larger. Storing the data in memory is not the major problem, but it takes much more learning effort to find all the values to fill the table. When using the table representation, experiences of states, or state-action pairs, do not generalize to similar states and state-action pairs. Generally, this is problematic if the state space is large or continuous. In such cases it is unlikely that exactly the same states will be encountered several times. To solve this problem, previous experiences need to be generalized to states and state-action pairs that have never been encountered before.

Function approximators can be used to make an approximation of a function. In this case, it can be the state value function, or the state-action value function. By giving examples of the desired input and output behavior, a function approximator generalizes this mapping to create an approximation of the function. Function approximators can be effectively combined with temporal difference learning algorithms, such as Q-learning. For example, once the Q-value for state-action pair (s,a) has been learned, the function approximator generalizes this experience to state-action pairs similar to (s,a), even though they have not been encountered explicitly during learning. If an appropriate approximation technique is chosen, it is likely that Q-learning is able to learn more efficiently, because an update of the Q-function (i.e., the approximator), also results in updated Q-values for similar state-action pairs. In the remaining part of this section we discuss linear function approximation techniques. They will be integrated in the algorithm we present in Chapter 4. A more sophisticated function approximation technique is discussed in Chapter 5.

Gradient descent methods

Gradient descent methods are widely used in linear function approximators, and are suitable to be combined with reinforcement learning [27]. Gradient descent is an optimization algorithm to find the local minimum of a function. To find this minimum, the algorithm takes steps proportional to the negative gradient. Consider an arbitrary function f that depends on one variable x. If the algorithm is in point $f(x)$, the gradient $\nabla f(x)$ is determined. Now the function will decrease the fastest when moving in the direction of the negative gradient. When proceeding in this way, a local minimum can be found.

The method outlined above can be used to approximate value functions. A value function $V(s)$ can be expressed as a linear combination of features derived from s, where each feature is a binary value that represents whether a feature is present in state s or not. Features can also be real valued, but for convenience we assume that features are either 0 or 1. Equation 3.7 shows a value function V formulated as a linear combination of features, where ϕ_s is an n-dimensional feature vector of state s, and θ_t is the n-dimensional parameter vector of the approximate function. The subscript t is used to distinguish approximators and parameters at consecutive time steps t and $t+1$.

$$V_t(s) = \theta_t^\top \phi_s = \sum_{i=1}^{n} \theta_t(i)\phi_s(i) \tag{3.7}$$

If $V_t(s)$ is a smooth differentiable function of the parameter vector θ_t for all states s, then the gradient descent method can be applied to adjust the parameters such that V approximates the desired function V^π more closely. This will be formalized in Equation 3.8, which is derived from [27]. The parameter α denotes the learning rate.

$$\begin{aligned} \theta_{t+1} &= \theta_t - \frac{1}{2}\alpha\nabla_{\theta_t}\left[V^\pi(s_t) - V_t(s_t)\right]^2 \\ &= \theta_t + \alpha\left[V^\pi(s_t) - V_t(s_t)\right]\nabla_{\theta_t}V_t(s_t) \end{aligned} \tag{3.8}$$

The parameter values are adjusted in the direction proportional to the negative gradient of the mean squared error of the true value function V^π and the approximation V_t. The motivation behind this is that is that an approximation should minimize the difference between the desired function and the current approximation that depends on θ_t. Hence, the parameters can be adjusted in the direction of the negative gradient of this error. The derivation above yields a quite natural update rule for the parameters θ_t, because it holds that $\nabla_{\theta_t}V_t(s_t) = \phi_s$. Therefore, the update rule from Equation 3.8 reduces to:

$$\theta_{t+1} = \theta_t + \alpha\left[V^\pi(s_t) - V_t(s_t)\right]\phi_s \tag{3.9}$$

In Example 1 below we will argue that $\nabla_{\theta_t}V_t(s_t)$ is equal to ϕ_s, and we illustrate how a function can be approximated using this recipe.

Example 1 (Linear function approximation)**.** *Consider a value function* $V_t(q) = \theta_t(1)\cdot\phi_q(1) + \theta_t(2)\cdot\phi_q(2)$*, where q is a state for which* $\phi_q = (1,0)^\top$*. The desired function V^π is equal to* 20 *for state q:* $V^\pi(q) = 20$*. We assume that all the parameters in the vector θ_t are initially zero, and that the learning rate α is equal to* 0.8*. This means that* $V_t(q) = 0\cdot 1 + 0\cdot 0 = 0$*. Now it holds that:*

$$\nabla_{\theta_t}V_t(q) = \left(\frac{\partial V_t(q)}{\partial\theta(1)}, \frac{\partial V_t(q)}{\partial\theta(2)}\right)^\top = (1,0)^\top = \phi_q$$

Now the update rule from Equation 3.9 can be applied to update the parameter vector:

$$\theta_{t+1} = \theta_t + \alpha\left[V^\pi(q) - V_t(q)\right]\phi_q = \begin{bmatrix}0\\0\end{bmatrix} + 0.8\cdot[20-0]\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix} + \begin{bmatrix}16\\0\end{bmatrix} = \begin{bmatrix}16\\0\end{bmatrix}$$

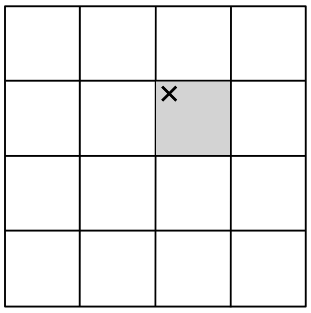
The newly obtained approximator V_{t+1} evaluates to 16 *for state q:* $V_{t+1}(q) = 16\cdot 1 + 0\cdot 0 = 16$*, which is closer to the desired value* 20*. This procedure can be repeated several times to obtain a better approximation of the desired function.*

The approximation technique described above can be used to approximate value functions, as shown in Example 1. Applying linear function approximation directly to approximate value functions would be a naive approach, since it is unlikely that all value functions can be expressed as a linear combination of features. Typically, for large state spaces and non-linear reward functions, a simple linear approximation is not sufficient. Next we describe a more sophisticated technique to deal with this issue.

Tile coding

As mentioned previously, generalization is required to be able to learn effectively if the state space is large and continuous. Gradient descent linear function approximators can be used to achieve this, but the accuracy of the approximation depends on the features that can be extracted from the state description. In this section we present tile coding, a method to discretize the state space and identify features [27]. The number of features can be controlled explicitly and does not depend on the state description.

Consider a two dimensional state space, in which the state variables are real numbers in the domain $[0,1]$. The state space can be visualized as a so-called tiling, as shown in Figure 3.2a. The cells in this tiling partition the state space and are called tiles. The cross symbol represents a state in the two dimensional space and lies in the shaded tile. A weight is associated with each tile, and each tile corresponds to a feature. This means that for every state in the state space, there is exactly one feature present in a tiling. Features are binary values, indicating whether a state belongs to a tile or not. This interpretation of the state space ensures generalization, because multiple states will be mapped onto the same tile. The shape of the tiles can be changed to create other forms of generalization.


(a) Tiling.

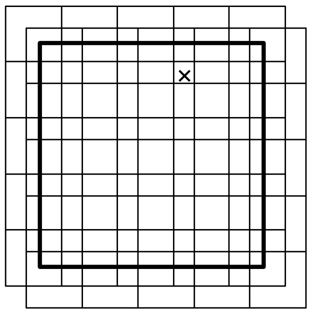

(b) Overlapping tilings.

Figure 3.2: Tile coding.

If there is one tiling to partition the state space, there will be only one feature present in each state. To obtain a higher accuracy with linear function approximation techniques, it is required to have multiple features for each state. Otherwise, the approximate function value of a state would be exactly the weight of the corresponding tile. Figure 3.2b shows two overlapping tilings, where the tilings have a different offset in each dimension. The bold square represents the two dimensional state space. The states have two corresponding features and the approximate function value equals the sum of the two weights.

The number of features present in each state can be controlled by changing the number of tilings m. If there are m tilings, the approximation can be computed by taking the sum of m weights, where the weights correspond to the feature tiles associated with the state. The number of tilings and their respective offset also controls the resolution of the approximation. This scheme can also be generalized to state spaces with a higher dimensionality.

For gradient descent linear function approximation, the update rule in Equation 3.9 remains the same. Note that it is useful to define the learning rate in terms of the number of tilings m. For example, if the learning rate α is set to $\frac{1}{m}$, a weight update will immediately change the approximate function value to the desired value. This fraction can be adapted to obtain the desired learning behavior.

Q-learning with function approximation

Function approximation techniques can be combined with reinforcement learning algorithms, such as Q-learning. In Section 3.3 we have shown how a value function can be approximated, but this recipe can easily be generalized to make approximations of state-action value functions. Algorithm 2 below shows how Q-learning can be combined with tile coding function approximation. It assumes that all state variables are real values in the domain $[0,1]$. Earlier in this chapter we mentioned that Q-learning converges to an optimal policy if each state-action pair is encountered an infinite number of times, but convergence proofs are no longer valid when applying function approximation.

Algorithm 2: Q-learning with tile coding function approximation

```
input  : initial state $s$, discount factor $\gamma$, tile width $w$, number of tilings $m$
output: approximate $Q$-function, representing a policy

$d \leftarrow$number of dimensions of the state space
foreach action $a$ do
    let $\theta_a$ be a vector containing $m \cdot \left(\frac{1}{w}+1\right)^d$ zeros
foreach tiling do
    foreach state space dimension do
        assign an offset $o \sim U(0,w)$
foreach episode do
    foreach step of the episode do
        determine $\phi_s$ using tile coding scheme, taking offsets into account
        calculate $\theta_a^\top \phi_s$ for each action $a \in A$
        choose action $a$ based on $Q$-values, using an exploration strategy
        execute $a$, observe reward $r$ and new state $s'$
        determine $\phi_{s'}$ using tile coding scheme, taking offsets into account
        $\theta_a \leftarrow \theta_a + \frac{1}{m}\left(r + \gamma \max_{a' \in A}\left(\theta_{a'}^\top \phi_{s'}\right) - \theta_a^\top \phi_s\right)\phi_s$
        $s \leftarrow s'$
```

Note that the algorithm maintains a tile coding function approximator for each action $a \in A$, represented by a weight vector θ_a. If the agent is in state s, the next action can be determined by calculating the Q approximation for each action a. First it should be determined which features are present in state s (i.e., determine ϕ_s using tile coding). Moreover, there are m additions and multiplications for each action a to calculate the approximation.

3.4 Summary

In this chapter we discussed the most important concepts of reinforcement learning, which deals with agents that act in an environment to optimize a notion of cumulative reward. Typically, the environment is modeled as a Markov Decision Process (MDP), which involves states, actions and rewards received after performing actions. A solution to an MDP is a mapping from states to actions, such that the expected sum of rewards received by the agent is optimized. If the transition probabilities of the MDP are unknown, model-free reinforcement learning algorithms can be used. The most important ones are Q-learning and SARSA, which can be enhanced with function approximation. This is a technique to store an approximation of the expected rewards in memory, rather than maintaining a complete table containing all these values explicitly. One of the benefits is that learning becomes more efficient, because the experiences of the agent generalize and thus it improves the learning capabilities of the algorithm.

Chapter 4

Speed Limit Policies

In this chapter we present our first algorithm to compute speed limitations that aim to reduce congestion. The algorithm we propose is based on the reinforcement learning concepts presented in Chapter 3 and uses the macroscopic traffic flow model from Chapter 2. In Section 4.1 we give an informal introduction to the problem at hand, including the requirements we impose on the solution. This problem is formulated as a Markov Decision Process (MDP) in Section 4.2. This involves a translation of the problem to states, actions and a reward function. It is possible to find solutions to the problem, represented by policies, which is the topic of Section 4.3. An experimental evaluation of the algorithm is presented in Section 4.4. In the discussion in Section 4.5 we describe shortcomings and possible improvements of the algorithm. This allows us to present an improved version of the algorithm in the next chapter. The results presented in this chapter are related to research question 1.

4.1 Problem description

As mentioned in the introductory chapter, the objective of this thesis is to propose new algorithmic techniques to compute speed limitations, which aim to reduce congestion. One solution would be optimizing the traffic flow of the entire road network. However, it is likely that increasing traffic demands cause congestion nearby, and do not affect the traffic conditions of the complete road network. Another aspect is the scalability of the solution, which tends to be problematic in case of very large road networks. Based on these considerations we will propose techniques for localized control, at the level of individual highways.

Consider the example highway in Figure 4.1. The highway has been divided into eight sections. Section 6 and 7 have an on-ramp, allowing car drivers to enter the highway. In

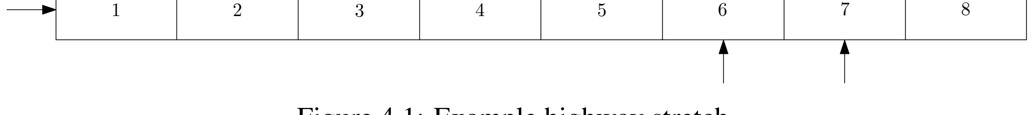


Figure 4.1: Example highway stretch.

case of high traffic demands for both the origin and the on-ramps, congestion will arise around section 6 and 7 and will propagate upstream. This means that after a while, sections 4 and 5 may become congested as well, as a consequence of the increased traffic demands. Intuitively, this problem can be circumvented by assigning speed limitations to the upstream sections, with the intention to harmonize traffic flow. However, initially it is unclear when to assign speed limits, and to what extent the speed upstream should be decreased. Moreover, for algorithms computing these speed limitations, the following requirements can be imposed:

1. Speed limits should not cause unnecessary delay. This means that intervention should only take place if congestion does not resolve without speed limits. If there is no reason to assign speed limitations, the algorithm should not trigger unnecessary changes.

2. Speed limits should be decreased and increased smoothly, at predefined control steps. Large differences between consecutive speed limits and frequently changing speed limits are inconvenient for car drivers and may eventually cause dangerous situations. Additionally, it is desirable that the speed limitations are round numbers (e.g., multiples of 10 km/h).

3. Speed limits should optimize the traffic conditions of all the highway sections, and should not focus on the travel times of individual car drivers. A global optimization strategy should optimize the number of vehicle hours, which represents the number of hours people are spending on the highway within a time interval. In case of congestion, the number of vehicle hours will be unnecessary high as a consequence of the reduced speed and increased travel times. Hence, the number of vehicle hours can be used as an objective function.

Returning to our example from Figure 4.1, it is reasonable to assign speed limitations to the sections 2 to 6 in case of high origin and on-ramp demands. The idea behind this observation is that if the speed is lower, it is easier to merge near the on-ramps. Secondly, lower speeds cause the distances between vehicles to decrease, such that there can be more vehicles on the highway. The informal description to the optimization problem will be formalized in the next section.

4.2 Formulation as Markov Decision Process

In this section the traffic flow optimization problem will be formulated as a Markov Decision Process (MDP), as introduced in Chapter 3. We will use the notational conventions introduced in Chapter 2 and Chapter 3. Before we define the MDP, it is worth mentioning that the simulation time step size T of METANET may be smaller than the control time step size of the desired highway controller. For example, the simulation step size may be 15 seconds, while speed limits are changed every 5 minutes. We define the control time step size T_c as an integer multiple of the simulation time step size: $T_c = c \cdot T$, where $c \in \mathbb{N}$. In the example, c would be equal to 20, such that $T_c = 20 \cdot \left(\frac{0.25}{60}\right) = \frac{5}{60}$ hour. Below we describe the actions, the state description and the reward function of the MDP.

Actions

The actions of the MDP correspond to the speed limits that can be assigned to section 2 to 6. The action space consists of four elements: $A = \{1,2,3,4\}$. An action $a \in A$ can be translated to a speed limitation using the following function:

$$lim(a) = \begin{cases} 120 & a = 1 \\ 100 & a = 2 \\ 80 & a = 3 \\ 60 & a = 4 \end{cases} \tag{4.1}$$

The speed limitations given by the function *lim* can be assigned directly to highway sections. The function also relates to requirement 2 from the previous section, which states that speed limits should be round numbers. We mentioned that speed limits are assigned to section 2 to 6, but in general speed limits can be assigned to any highway section.

State description

We define a six-dimensional state space, in terms of the average speeds of the highway and previous speed limits. The initial state and a recursive formulation of the other states is shown below.

$$s_0 = \left(\frac{120}{v_f}, \frac{120}{v_f}, \frac{v_4(0)}{v_f}, \frac{v_5(0)}{v_f}, \frac{v_6(0)}{v_f}, \frac{v_7(0)}{v_f} \right) \tag{4.2}$$

$$s_n = \left(\frac{lim(a_{n-1})}{v_f}, s_{n-1}(1), \frac{v_4(cn)}{v_f}, \frac{v_5(cn)}{v_f}, \frac{v_6(cn)}{v_f}, \frac{v_7(cn)}{v_f} \right) \tag{4.3}$$

The first entry of the state space represents the chosen speed limit from time $(n-1)c$ to nc. This speed limit corresponds to the previously chosen action. The second entry is the speed limit that was issued from time $(n-2)c$ to $(n-1)c$. In the reward function this value will be used to prevent the speed limits from alternating frequently. The remaining entries of the state space are the average speed of section 4 to 7, as we have discussed in the previous section. The value v_f is a METANET parameter, representing the free flow speed. The state description can be further extended with information regarding speeds and densities of other sections, but in this chapter we decided to keep the dimensionality of the state space limited. The reason is that learning is more difficult if the state space is large and function approximation methods may not scale well enough.

In the state description above, we use the current speed values, represented by the last four state variables. Instead of defining the state in terms of the current speed, model-based predictions of future speeds can also be used in the state description. A model-based prediction can be generated using METANET, starting from the current traffic state. Running a traffic simulation for 5 minutes results in a perfect prediction of the traffic conditions after 5 minutes, which can be included in the states. This approach assumes that future traffic demand profiles are known. In the experiments at the end of this chapter we will also evaluate this variant of the algorithm.

Reward function

The reward function of the MDP can be used to implicitly define the objective function of the traffic flow optimization problem. First we will present the reward function, and thereafter we motivate how the distinguished cases relate to the requirements from the previous section. The rewards will be negative numbers, which can be interpreted as a punishment when the agent has selected a *bad* action. Maximizing the received rewards yields a solution that optimizes the objective function. The reward r_n received by the agent after arriving in state s_{n+1} is as follows:

$$r_n = \begin{cases} -\varphi & s_n(2)v_f = lim(a_n) \wedge s_n(1)v_f \neq lim(a_n) \\ -\varphi & |s_n(1)v_f - lim(a_n)| > 20 \\ 0 & \min\{v_i((n+1)c) \mid i = 1,\dots,N\} > 101 \\ -VH(n) & \text{otherwise} \end{cases} \tag{4.4}$$

where φ is a large constant and $VH(n)$ is defined as:

$$VH(n) = T \sum_{p=nc}^{(n+1)c} \left(\sum_{i=1}^{N} [M_i L_i k_i(p)] + \sum_{i=0}^{N} w_i(p) \right)$$

The variables M_i, L_i and k_i represent the number of lanes, length and density of section i, respectively. The variable w_i represents the length of the queue associated with section i. The cases are evaluated in this order and the reward corresponding to the first valid condition is taken. The first case ensures that the agent receives a punishment if the selected actions result in alternating speed limits. This happens if the selected speed limit in state s_n is the same as the speed limit selected in state s_{n+2}, but differs from the speed limit selected in state s_{n+1}. This is reflected by the first condition, and it explains why we included the second entry of the state description. This case addresses requirement 2 from the previous section.

The second case results in a punishment for the agent if the difference between two consecutive speed limits is too large. In our definition we decided to use the value 20, resulting in smooth changes, but this value can be changed depending on the desired behavior. This case also relates to requirement 2.

If the minimum speed of the highway sections is above 101, it can be assumed that there is currently no congestion. Therefore, the agent does not receive a punishment in such states. This is represented by the third case. Additionally, it is used to ensure that the agent selects actions in such a way that congestion is prevented or eventually resolves. The value 101 is chosen such that the condition can only be true if the current speed limit is equal to 120.

In all other cases, the agent receives a punishment that is proportional to the number of vehicle hours people spent between the previous control time step and the current control time step. In the calculation of this value there are two components: the number of vehicle hours for vehicles that are actually on the highway, as well as the vehicle hours for vehicles that are unable to enter the highway due to congestion (e.g., in front of on-ramps). The punishment represented by this value will be large in case of congestion, and also depends on the speed limits that have been assigned to the sections. If the speed limits decrease too much, the number of vehicle hours will be unnecessary high, resulting in a lower reward value. This will ensure that speed limits are only decreased if strictly necessary. This case relates to requirement 1 and 3. Note that the parameter φ should be chosen in such a way that it is large compared to the values $VH(n)$ can take. This means that it should be at least as large as the maximum number of vehicle hours that may occur between two control time steps.

Solution

A solution to the MDP is a policy $\pi : S \rightarrow A$, which maps traffic conditions to speed limitations. These speed limitations can be assigned to section 2 to 6. The macroscopic METANET model needs to be adapted to include speed limitations. Equation 2.4 from Chapter 2 can be redefined as follows:

$$V_e(k,a) = \min\left\{lim(a), v_f\left[1-(k/k_{jam})^l\right]^m\right\} \tag{4.5}$$

where a is the selected action. This only applies to sections for which speed limits have been activated. A similar adaptation can be found in [32]. Policies can be learned using the Q-learning algorithm that runs interleaved with a METANET traffic simulation, as we will discuss in the next section.

4.3 Learning policies using Q-learning

The Q-learning algorithm from the previous chapter can be applied to learn speed limit policies. Algorithm 2 can be adapted in such a way that it simulates traffic flow using METANET, and performs a Q-learning step at every control time step of the highway controller. A high level description of the adapted algorithm is shown in Algorithm 3. Within each episode, a regular METANET simulation is executed. This involves a calculation of speed, flow, density and queue length values for each simulation time step. The time complexity of this simulation is $O(NK)$, where N is the number of sections and K is the number of simulation time steps. Every c steps, a regular Q-learning step is performed to update the Q-values and select a new action. The algorithm also includes tile coding function approximation to represent the Q-function. For clarity, the indices of the consecutive states have been discarded in the description of the algorithm, but this should be clear from the context.

Figure 4.2 shows a schematic representation of the speed limit policy algorithm. Using the initial traffic conditions, the initial state of the MDP is defined and the first action is selected. Then 5 minutes of traffic flow is simulated, which results in a new state. The reward is calculated based on the simulation values of the past five minutes, which makes it possible to update a Q-value. An optional step is to run a separate METANET simulation to generate a traffic prediction. Basically, this is an extrapolation of the current traffic conditions and gives information about the traffic conditions after five minutes. This information can be included in the state description. If the simulation has not been completed yet, the algorithm

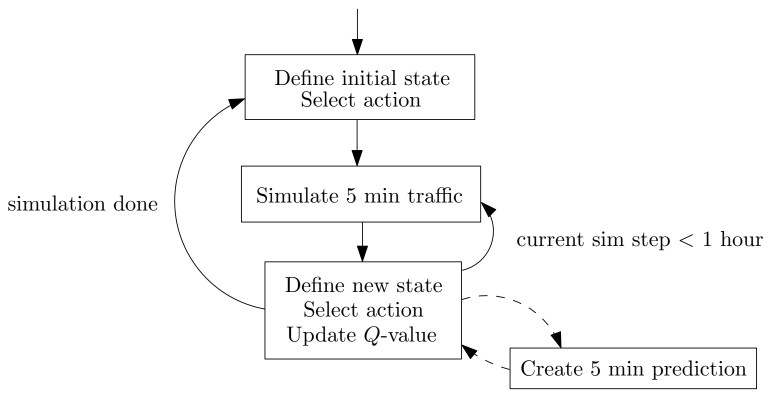


Figure 4.2: Schematic representation of speed limit policy algorithm.

goes back to the simulation phase. In all other cases it starts a new learning episode, starting from the initial traffic conditions. In this figure, it is assumed that the traffic simulation involves 60 minutes, and that control actions are applied every five minutes.

4.4 Experiments

In this section we describe the experiments we performed to evaluate the proposed algorithm and the policies it generates. Using these experiments we want to show that the speed limit policies generated by Algorithm 3 can reduce the number of vehicle hours significantly. The policy quality can be compared to the optimal number of vehicle hours and the number of vehicle hours without speed control. We also want to show that Algorithm 3 generates better policies if the states are defined in terms of model-based traffic predictions, instead of the currently measured speeds. The policy quality can be compared to the quality of the policies generated without predictions. In the remainder of this section we discuss the experimental setup, the results of our experiments and we conclude with a discussion of the capabilities and shortcomings of the proposed algorithm.

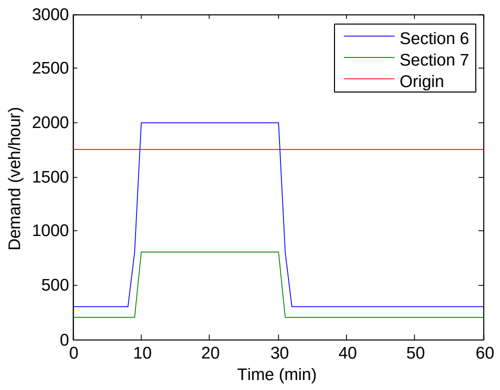
Experimental setup

For our experiments we have defined a unidirectional highway of 24 km, divided into 8 sections of 3 km each. All highway sections have two lanes. Sections 6 and 7 have an on-ramp, allowing vehicles to enter the highway. Initially, the average speed of each section is 120 km/hr and the average density of each section is 17 veh/km/lane. A schematic representation of the highway we consider is shown in Figure 4.1. We define four traffic scenarios, in which we simulate 60 minutes traffic flow. The demand profiles of the origin and the on-ramps of section 7 and 8 are shown in Figure 4.3.

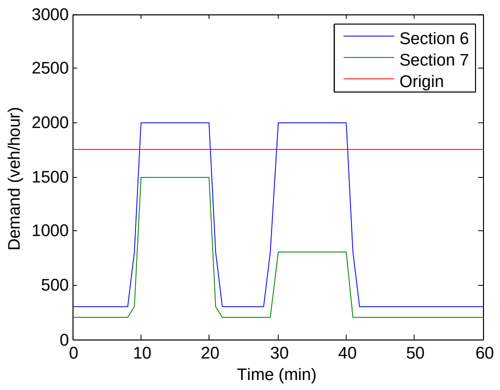
The algorithm we use is the speed limit policy learning algorithm shown in Algorithm 3. Unless stated otherwise, we run 5000 episodes, in which the last 100 episodes are fully greedy. This means that in the last 100 steps, the currently learned policy is followed without any random exploration. The probability to select random actions is decreased over time

Algorithm 3: Speed limit policy learning using Q-learning

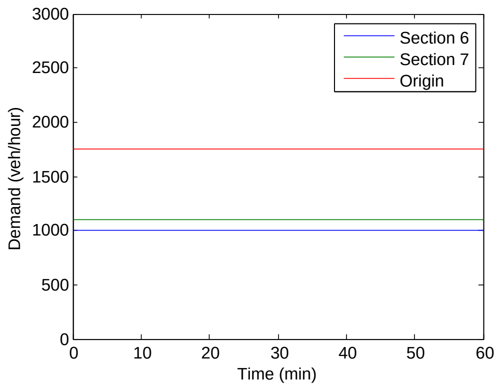
```
input : initial speeds and densities of each section, METANET model parameters,
        number of simulation time steps $K$, origin demands $r_d$, on-ramp demands $r$,
        off-ramp percentages $\beta$, control time step multiplier $c$, discount factor $\gamma$, tile
        width $w$, number of tilings $m$
output: approximate $Q$-function, representing a policy

$d \leftarrow$number of dimensions of the state space
foreach action $a$ do
    let $\theta_a$ be a vector containing $m \cdot \left(\frac{1}{w}+1\right)^d$ zeros
foreach tiling do
    foreach state space dimension do
        assign an offset $o \sim U(0,w)$
foreach episode do
    calculate initial flows and origin queue length using Equation 2.1 and 2.6
    define $s$ as the initial state using Equation 4.2
    determine $\phi_s$ using tile coding scheme, taking offsets into account
    calculate $\theta_a^\top \phi_s$ for each action $a \in A$
    choose action $a$ based on $Q$-values, using an exploration strategy
    assign limit $lim(a)$ to section 2 to 6
    foreach time step $i \in \{2,\dots,K\}$ of the traffic simulation do
        calculate speed, density and flow using Equation 2.1, 2.2 and 2.3
        calculate queue lengths using Equation 2.8
        if $i$ mod $c = 0$ then
            define new state $s'$ using Equation 4.3
            determine $\phi_{s'}$ using tile coding scheme, taking offsets into account
            calculate the reward $r$ using Equation 4.4
            $\theta_a \leftarrow \theta_a + \frac{1}{m}\left(r + \gamma \max_{a' \in A}\left(\theta_{a'}^\top \phi_{s'}\right) - \theta_a^\top \phi_s\right)\phi_s$
            calculate $\theta_a^\top \phi_s$ for each action $a \in A$
            choose action $a$ based on $Q$-values, using an exploration strategy
            assign limit $lim(a)$ to section 2 to 6
            $s \leftarrow s'$
```



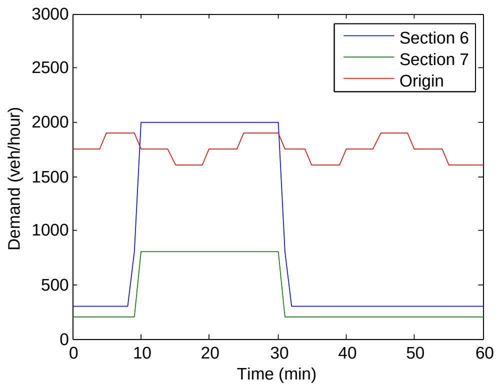
(a) Scenario 1.



(b) Scenario 2.



(c) Scenario 3.



(d) Scenario 4.

Figure 4.3: Traffic demands of the scenarios.

and follows this formula: $1-((i-1)/4900)$, where i is the episode number. The tile coding function approximator consists of 60 tilings and the width and height of each tiling is equal to 1.32. In each dimension there are four tiles. The offsets assigned are drawn from the $U(0,0.3)$ distribution when initializing the tilings. Offsets are added to the state variables when determining which tile of a tiling is activated. State variables are real valued between 0 and 1, so this probability distribution ensures that the state variables are less than 1.32 after adding the offset values. It is important that offsets associated to tilings are different, to ensure that the tilings have a different offset relative to other overlapping tilings.

We distinguish two versions of the algorithm. The first version defines the states using the current speed measures and the second version measures the current speed limits and uses METANET to make a prediction of the future speeds. For this prediction, the time horizon is set to 5 minutes. This prediction step is also visualized in Figure 4.2. Control actions are applied every 5 minutes, so c is equal to 20. The constant φ in the reward function is equal to 500 and the discount rate γ of the Q-learning algorithm is 0.8.

The METANET model is initialized using the model parameters from Chapter 2. The simulation step size is 25 seconds, so to simulate 60 minutes we need to calculate 241 values for each section (including the initial traffic conditions at time zero).

The quality of a policy can be evaluated by applying the learned policy to a scenario, following a fully greedy exploration strategy. We use the number of vehicle hours realized during one hour of traffic flow as a performance metric. This is defined as follows:

$$T \sum_{p=1}^{241} \left(\sum_{i=1}^{N} [M_i L_i k_i(p)] + \sum_{i=0}^{N} w_i(p) \right)$$

In the literature this value is known as the Total Time Spent (TTS) in a traffic network and it is used to evaluate road network controllers [32].

Bounds on the policy quality

To place the quality of the resulting policies into perspective, we can impose weak bounds on the number of vehicle hours realized by the generated policies. The situation in which there is no speed control (i.e., the speed limits are always 120) is considered as the baseline In practice it may be possible to exceed the baseline by setting speed limitations to very low values, but in any case we want our policies to realize less vehicle hours than the number of vehicle hours without control.

A lower bound can be computed by enumerating all sequences of actions, running a METANET simulation and calculating the number of vehicle hours. However, not all possible combinations of actions represent sequences of actions that can directly be applied. Recall from requirement 2 in Section 4.1 that speed limitations should be decreased smoothly. Therefore, we enumerated all sequences of actions $a_1, a_2, \ldots, a_{13}$, such that $|lim(a_i) - lim(a_{i+1})| \leq 20$ for all $i \in \{1, 2, \ldots, 12\}$. There may be combinations that give a lower number of vehicle hours, but these are not convenient to be used in practice. Note that it is not always feasible to compute optimal sequences of actions. For large scenarios and a small control time step size, the number of possible sequences to evaluate grows rapidly.

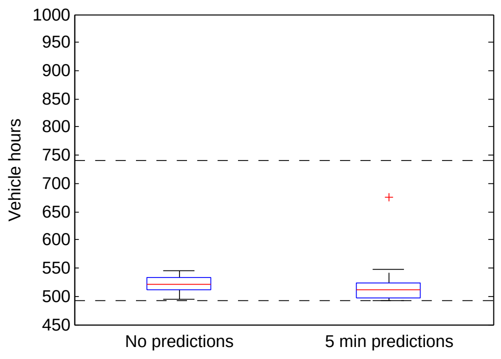
The computed baselines and lower bounds can be found in Table 4.1. Opt denotes the lower bound which is assumed to be optimal, and no control represents the number of vehicle hours without speed control.

Scenario	Opt	No control
1	491.6	740.8
2	511.2	954.5
3	543.5	933.8
4	514.5	768.6

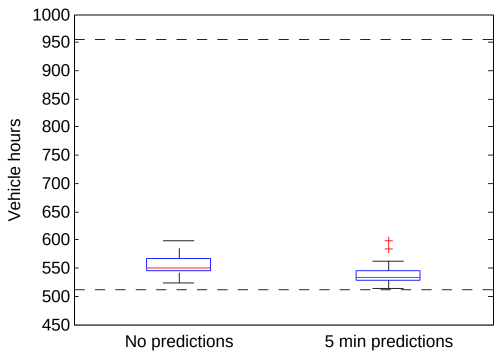
Table 4.1: Baselines and lower bounds for each scenario.

Policies for individual scenarios

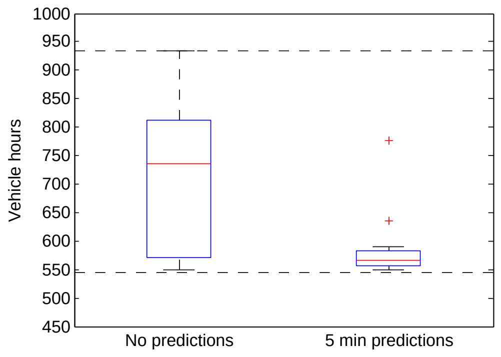
In the first experiment we investigate the policy quality for individual scenarios. There is randomness involved in the exploration strategy of the learning algorithm, so we generate 20 policies for each scenario, giving us a distribution of the vehicle hours realized by



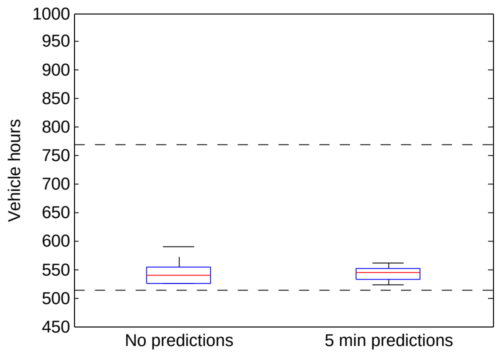
(a) Scenario 1.



(b) Scenario 2.



(c) Scenario 3.



(d) Scenario 4.

Figure 4.4: Policies for each individual scenario: comparison of policy quality.

the generated policies. This is repeated for the variant in which states are defined using model-based traffic predictions. The results of the experiment are shown in Figure 4.4 and Table 4.2. The horizontal lines represent the lower bound on the policy quality and the baseline.

Based on the results we conclude that the speed limit policy algorithm is able to generate useful policies to regulate traffic flow. We also observe that the predictive variant of the algorithm produces slightly better policies, which shows that it makes sense to include traffic predictions when determining speed limits.

The algorithm performs poorly on Scenario 3. An intuitive reason is that the traffic demands in this scenario are too high, and that speed limitations have limited effects on resolving congestion. As a consequence, the reward signal received by the learning agent is not helpful enough to converge in the direction of good policies. An improved version of our algorithm, presented in Chapter 5, generates better quality solutions for this scenario.

Policies for multiple scenarios

In practice it is desirable to have one single policy that can be used for multiple traffic scenarios. Therefore, we performed an experiment in which a policy was trained sequentially

	No control	Opt	Min	Max	Median	Mean	Std
Scenario 1	740.8	491.6	495.6	546.0	520.3	520.8	13.8
Scenario 2	954.5	511.2	523.0	598.5	550.4	553.9	17.8
Scenario 3	933.8	543.5	549.1	933.6	735.3	716.1	142.6
Scenario 4	768.6	514.5	525.1	589.0	538.5	542.2	17.2

(a) Vehicle hours without predictions.

	No control	Opt	Min	Max	Median	Mean	Std
Scenario 1	740.8	491.6	491.6	676.4	512.2	519.2	40.1
Scenario 2	954.5	511.2	513.7	598.9	534.2	540.5	20.8
Scenario 3	933.8	543.5	548.6	775.3	565.5	580.4	50.3
Scenario 4	768.6	514.5	522.5	560.1	545.6	542.9	12.4

(b) Vehicle hours with 5 minute predictions.

Table 4.2: Policies for each individual scenario: vehicle hours.

on multiple scenarios, from scenario 1 to scenario 4. This was repeated 20 times, resulting in 20 policies without a predictive state description, and 20 policies with predictions included.

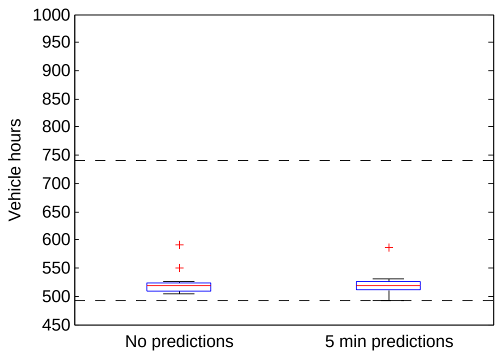
The 20 policies were applied to each scenario, and the policy quality was measured in terms of the number of vehicle hours. The results are shown in Figure 4.5 and Table 4.3. We conclude that policies that have been generated for multiple scenarios are able to reduce the number of vehicle hours significantly. An advantage is that this process gives policies that can be applied to regulate traffic in multiple scenarios, instead of one single scenario. An important result of this experiment is that the policies generated using the predictive state description are consistently better, on the average.

Compared to the results in Table 4.2, the average policy quality decreased slightly. A possible explanation for this observation is that the policies in this experiment should be able to solve multiple scenarios, and represent a more general policy than the policies in the previous experiment. Again we observe that the algorithm is unable to produce consistently high quality policies for the third scenario. This can be an indication that the traffic flow in this scenario cannot be efficiently regulated by assigning speed limitations.

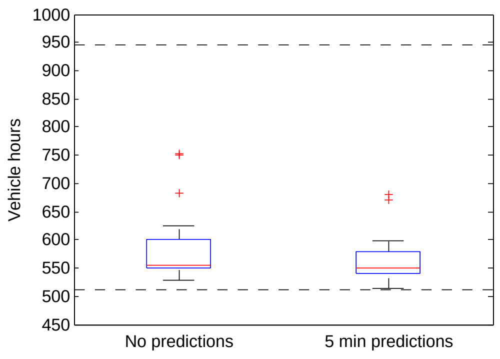
Convergence of the algorithm

The Q-learning algorithm is proven to converge to an optimal solution if every state action pair is encountered infinitely often. Convergence is not guaranteed if function approximation is used, though. The reason is that convergence proofs are no longer valid if the states are discretized, and our tile coding scheme is a form of discretization. As can be seen in Figure 4.4 and Figure 4.5, the algorithm does not always converge to the same solution, but for Scenario 1, 2 and 4 the resulting policies are within a relatively small range and close to the optimal policies we identified.

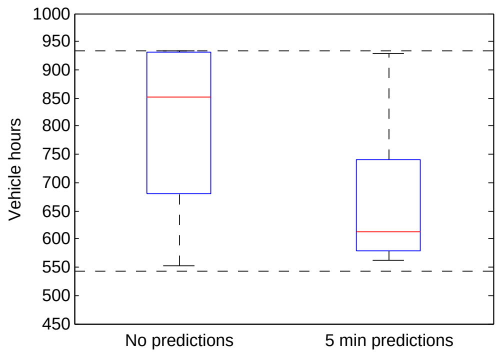
To understand which aspects influence the quality of the resulting policies, we per-



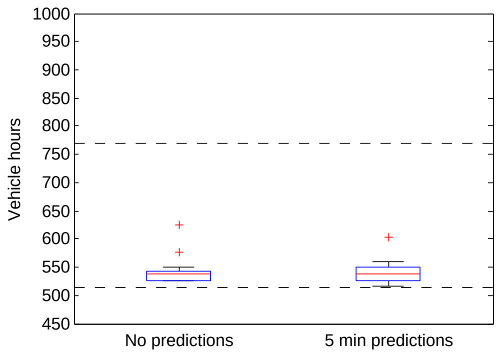
(a) Scenario 1.



(b) Scenario 2.



(c) Scenario 3.



(d) Scenario 4.

Figure 4.5: One policy for multiple scenarios: comparison of policy quality.

	No control	**Opt**	**Min**	**Max**	**Median**	**Mean**	**Std**
Scenario 1	740.8	491.6	503.9	590.9	518.2	521.0	19.4
Scenario 2	954.5	511.2	529.2	752.4	555.2	591.5	77.1
Scenario 3	933.8	543.5	551.5	933.8	850.6	805.0	140.2
Scenario 4	768.6	514.5	525.1	625.0	537.1	541.4	22.9

(a) Vehicle hours without predictions.

	No control	**Opt**	**Min**	**Max**	**Median**	**Mean**	**Std**
Scenario 1	740.8	491.6	491.6	587.1	518.9	518.8	19.8
Scenario 2	954.5	511.2	514.2	679.8	550.7	568.1	50.6
Scenario 3	933.8	543.5	561.3	928.7	612.2	679.0	131.6
Scenario 4	768.6	514.5	516.4	603.8	539.2	540.2	19.5

(b) Vehicle hours with 5 minute predictions.

Table 4.3: One policy for multiple scenarios: vehicle hours.

formed an experiment. If we take a look at the Q-learning algorithm itself, the number of episodes may be an important parameter that can be varied to obtain better policies. A larger number of episodes ensures that the algorithm performs more learning steps and encounters more, possibly unique, state action pairs. If this intuitive idea is true, we would observe a better policy quality if we increase the number of episodes.

We performed an experiment on the first scenario, and we executed 5000, 10000, 15000 and 20000 episodes without predictions. For each configuration we generated 20 policies. The resulting policy quality is visualized in Figure 4.6 and Table 4.4. We observe, as expected, that the distribution of the policy qualities improves if we increase the number of episodes. However, the distribution for 15000 and 20000 episodes is similar if we do not take the outliers into account. Interestingly, the minimum found is identical for 10000, 15000 and 20000 episodes, but when running 5000 for each run, the algorithm found a better policy. This may be caused by overfitting, which occurs when a learning algorithm runs too long and performance on unseen training examples decreases, while there is still accurate performance on training samples encountered before. Based on our observations we conclude that the number of episodes has influence on the resulting policy quality, but probably it cannot be further increased to let the algorithm converge to the same policy every run. If we take the baseline from Table 4.1 into account, we conclude that the resulting policies are within a relatively small range, which is acceptable.

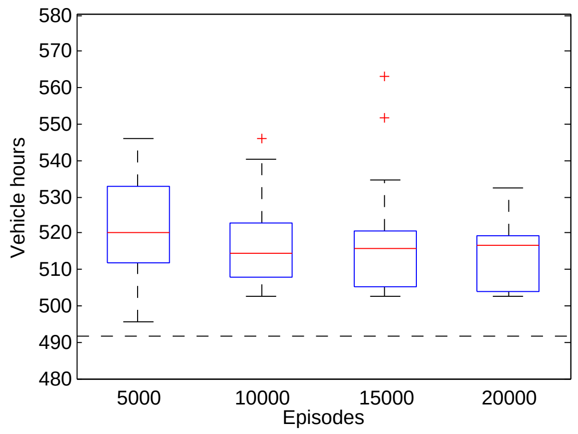


Figure 4.6: Policy quality for different numbers of episodes.

Episodes	Min	Max	Median	Mean	Std
5000	495.6	546.0	520.3	520.8	13.8
10000	502.7	546.1	514.5	517.8	13.9
15000	502.7	563.0	515.9	518.6	16.2
20000	502.7	532.6	516.7	513.6	8.6

Table 4.4: Policy quality for different numbers of episodes.

4.5 Discussion

Based on the exploratory experiments in this chapter, we can conclude that the proposed formulation of the optimization problem as an MDP solves the problem at hand, and generates policies of reasonable quality. Including speed predictions in the state description shows that there is a consistent improvement in the policy quality. However, there are a few remarks to be made.

The problem formulation we introduced in Section 4.2 is not Markovian. This is inherently the case when applying function approximation and discretization techniques, but it may have influence on the quality of the resulting policies. Another reason that can be easily observed is that the reward function we presented is not exclusively expressed in terms of the state variables. The term responsible for the penalty proportional to the number of vehicle hours also uses information about the traffic conditions that occurred between two consecutive states. An alternative approach would be determining the transition probabilities of the MDP and the reward function, such that standard dynamic programming techniques can be applied to find policies (e.g., value iteration) [31]. Given the size and dimensionality of the state space, however, it is not practical to determine the transition probabilities of the MDP. Moreover, the reward function $R(s,a,s')$ is unknown beforehand, which makes it difficult to apply dynamic programming to find policies.

Another limitation of the proposed algorithm is the approximation capability of tile coding. Even though the accuracy can be increased by changing the resolution of the tilings (e.g., adding more tiles), the approximation method itself is still a linear approximation technique and more powerful methods may exist to approximate the value function. Another disadvantage of the current approximation method is the fact that it does not scale well. Increasing the dimensionality of the state space leads to a larger running time, and the generalization capabilities are questionable. The scalability of tile coding will be further examined in the next chapter. There we show that another method scales better, but we cannot present the results of this comparison before introducing a more sophisticated approximation technique, presented in Section 5.1.

We conclude that the proposed algorithm generates policies of acceptable quality, and it eventually gave us useful insights, but there is some room for improvement. In particular, other function approximation techniques and methods to formulate an MDP that preserves the Markov property better than the current one may be helpful to devise an algorithm that performs better. An improved policy learning algorithm will be discussed in the next chapter.

4.6 Summary

In this chapter we defined a traffic flow optimization problem as a Markov Decision Process (MDP). By combining a macroscopic model with Q-learning, we have been able to construct an algorithm to find so-called speed limit policies. To learn a policy that optimizes a traffic scenario, we ran the macroscopic traffic simulation interleaved with Q-learning, such that the actions performed by the learning algorithm influence the traffic simulation. Experiments have shown that the resulting policies are able to reduce congestion significantly under high demands. We also experimented with traffic predictions included in the state description of the MDP and it turns out that policy quality improves. We identified shortcomings of the proposed method, that will be addressed in the remaining parts of this thesis.

Chapter 5

Improved Policy Learning

In this chapter we introduce and evaluate an improved speed limit policy learning algorithm, in which we build upon the techniques and insights from Chapter 4. In the previous chapter we discussed some shortcomings of the algorithm and after our exploratory experiments we observed that there is room for improvement. Based on this understanding, we will propose possible improvements to address the shortcomings of the algorithm. Then we modify the problem formulation as a Markov Decision Process and we show that the improved algorithm performs well and generates better policies.

5.1 Function approximation with neural networks

As mentioned in the previous chapter, the scalability and linearity of the tile coding function approximator is a bottleneck of the proposed algorithm, since it prevents us from extending the state space while preserving performance in terms of both running time and policy quality. Non-linear function approximation techniques have shown to be successful in combination with reinforcement learning [6]. For example, artificial neural networks can be used to learn to approximate a value function. A disadvantage is that convergence guarantees of the reinforcement learning algorithms no longer hold, but interesting empirical results exist in the literature [31, 14, 1]. In this section we give an introduction to neural networks and we explain how a Q-function can be approximated with such a network.

Neural networks

A neural network is a model based on the behavior of human brains and consists of a large number of processing units, called neurons [10]. These neurons form an interconnected network and propagate signals to other neurons by generating electronical reactions. Human brains have nearly 10 billion neurons and 60 trillion connections between them that can be used simultaneously, which makes them faster than any existing computer [20].

Artificial neural networks are a generalization of the neural network in the human brain. It uses experiences to improve its performance and it can be used to generalize these experiences to other experiences that have not been encountered before. An artificial neural

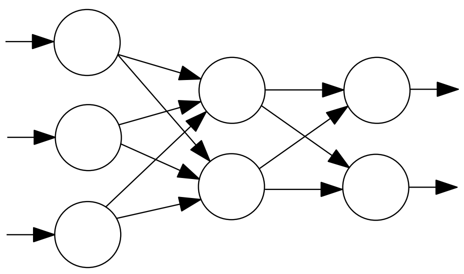


Figure 5.1: Artificial neural network structure.

network consists of neurons connected by weighted links, resembling a directed weighted graph. An example structure of an artificial neural network is shown in Figure 5.1.

The leftmost neurons (i.e., the input layer) receive a value as their input and propagate it to the neurons in the middle layer (also known as hidden layer). Based on the input signals received, the neurons in the middle layer compute a single output signal, which is propagated to the neurons in the rightmost layer (i.e., the output layer). The weights associated with the links are used to learn an input-output pattern. They represent the importance of an input signal of a neuron and these weights can be changed in order to learn.

To explain the workings of an artificial neural network further, we consider the behavior of one individual neuron shown in Figure 5.2.

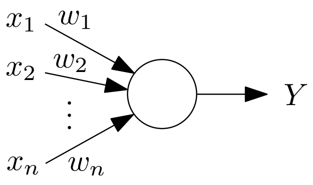


Figure 5.2: Neuron of an artificial neural network.

A neuron receives input signals $x_1, x_2, \ldots, x_n$ through its weighted inputs. The weights associated with these inputs are denoted by $w_1, w_2, \ldots, w_n$, respectively. The inputs and the weights are used to calculate the weighted sum of the neuron inputs: $X = \sum_{i=1}^{n} x_i w_i$. The weighted input X is translated to an output signal Y using an activation function. A commonly used activation function is the sigmoid function, which accepts any numerical input and returns a value between 0 and 1. The sigmoid function $S(t)$ and the formula for the neuron output Y are shown below.

$$S(t) = \frac{1}{1+e^{-t}}$$
$$Y = S\left(\sum_{i=1}^{n} x_i w_i\right)$$

Other activation functions are the sign activation function, which is 1 if $X \geq 0$ and -1 otherwise, and the linear activation function $Y = X$. The artificial neural network that we will use later in this chapter consists of neurons with a sigmoid activation function in the middle layer and a linear activation function in the output layer.

An artificial neural network with one middle layer can learn any continuous function. For discontinuous functions, more hidden layers are required [20]. The neurons in the middle layers identify the features that are present in the input and determine the expressiveness of the network. It is common to use one input that is always equal to one. This input is called a bias and is useful because the corresponding weights are always present in the calculation of the weighted sum and the activation in the middle layer. It shifts the activation function by introducing this bias.

Learning in multilayer neural networks

To learn a function using a neural network, we consider the artificial neural network in Figure 5.3. There are n input neurons, m neurons in the middle layer and the output is the function value. The neurons in the hidden layer use the sigmoid activation function and the activation function of the output neuron is the linear activation function.

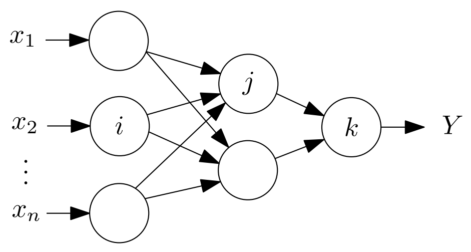


Figure 5.3: Artificial neural network with multiple layers.

Suppose that we have the input vector $(x_1, x_2, \ldots, x_n)$ and a desired network output Y_d. In the first phase of a learning iteration, the output of the network is computed by propagating the input values from left to right through the network, using the neuron activation scheme discussed above. This gives an output value y_p for each neuron p in the network.

In the second phase of a learning iteration, errors are propagated backwards from right to left. Consider a neuron j in the middle layer and output neuron k, the weight correction Δw_{jk} of the link from j to k is computed as follows:

$$\Delta w_{jk} = \alpha \cdot y_j \cdot (Y_d - y_k)$$

where α is the learning rate. Now we can calculate the error gradient δ_j of neuron j:

$$\delta_j = y_j \cdot (1 - y_j) \cdot (Y_d - y_k) \cdot w_{jk}$$

This is repeated m times, resulting in m weight corrections and error gradients. The error gradients are used to adjust the weights of the incoming links of the neurons in the middle layer. The weight correction for a link from input neuron i to neuron j in the middle layer is computed as follows:

$$\Delta w_{ij} = \alpha \cdot x_i \cdot \delta_j$$

This step involves $n \cdot m$ weight corrections, since all the input neurons are connected to all the neurons in the middle layer. Note that the weight corrections are applied after computing all the weight corrections. Otherwise, the weight corrections of the incoming links of the output neuron would influence the corrections of the other links. The recipe above is called back-propagation [4], and is a commonly used technique to train neural networks. Another remark that has to be made is that the back-propagation rules we presented are only applicable to the special case we considered. If there are multiple output neurons and if their activation function is not linear, the back-propagation rules are different. For a more elaborate introduction to back-propagation and artificial neural networks in general, we refer to [20]. The complete derivations of the back-propagation rules can also be found in [20].

Q-learning combined with neural networks

Neural networks can be used as a function approximator combined with Q-learning [1]. For each action, a neural network can be created, where the number of inputs is equal to the dimensionality of the state space. The output of the network is the Q-value. This means that the neural network corresponding to action a approximates $Q(s,a)$ for all possible states s. An example function approximator for a policy π is shown in Figure 5.4 below. In this case the action space is defined as $A = \{1,2,3\}$, so there are three neural networks to approximate the Q-function. Given a state s, the policy can be evaluated by calculating $Q^{\pi}(s,1)$, $Q^{\pi}(s,2)$ and $Q^{\pi}(s,3)$, and taking the action corresponding to the maximum.

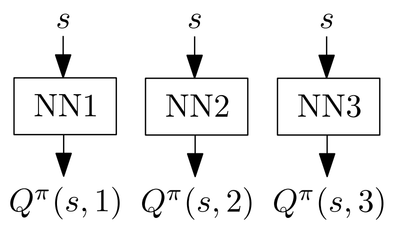


Figure 5.4: Neural networks as function approximator.

If the Q-learning algorithm changes a Q-value, then one iteration of the back-propagation algorithm is performed, such that the neural network gradually learns an approximation of the real Q-function. Generalization is ensured because a neural network is able to generalize training samples to inputs it has not seen before. Note that we do not retrain the neural network on a complete training set containing the desired input-output mapping. After each step of an episode, just one Q-value error is propagated backwards through the network.

5.2 Modified formulation as Markov Decision Process

Building upon the problem formulation from the previous chapter, and the considerations in the previous section, we present a modified MDP in this section. We start with a new state description, then we discuss the modified actions and finally we describe the consequences for the reward function.

State description

We adopt the state description from the previous chapter, but we include four additional variables corresponding to the density of section 4 to 7. The modified state description is shown below.

$$s_0 = \left(\frac{120}{v_f}, \frac{120}{v_f}, \frac{v_4(0)}{v_f}, \frac{v_5(0)}{v_f}, \frac{v_6(0)}{v_f}, \frac{v_7(0)}{v_f}, \frac{k_4(0)}{k_{jam}}, \frac{k_5(0)}{k_{jam}}, \frac{k_6(0)}{k_{jam}}, \frac{k_7(0)}{k_{jam}}\right) \tag{5.1}$$

$$s_n = \left(\frac{lim(a_{n-1})}{v_f}, s_{n-1}(1), \frac{v_4(cn)}{v_f}, \frac{v_5(cn)}{v_f}, \frac{v_6(cn)}{v_f}, \frac{v_7(cn)}{v_f}, \frac{k_4(cn)}{k_{jam}}, \frac{k_5(cn)}{k_{jam}}, \frac{k_6(cn)}{k_{jam}}, \frac{k_7(cn)}{k_{jam}}\right) \tag{5.2}$$

In order to guide the neural network during the process of learning an approximation of the Q-function, it can be helpful to include redundant information in the state description. Binary indicator variables are useful to indicate whether certain features are present in the state or not. This will ensure that the neural network weights corresponding to the indicator input nodes are only taken into account if the features are present in the state. We add twelve indicator variables $I_{n,j}$ to a state s_n ($j = 1,\dots,12$), and their definition is given below. Note that $s_n(1)$ denotes the first state variable, so we count from 1.

$$I_{n,1} = \begin{cases} 1 & \min\{v_i(cn) \mid i = 4,\dots,7\} < 100 \\ 0 & \text{otherwise} \end{cases}$$

$$I_{n,2} = \begin{cases} 1 & \max\{k_i(cn) \mid i = 4,\dots,7\} > 50 \\ 0 & \text{otherwise} \end{cases}$$

$$I_{n,3} = \begin{cases} 1 & s_n(1)v_f < 120 \\ 0 & \text{otherwise} \end{cases}$$

$$I_{n,4} = \begin{cases} 1 & \min\{v_i(cn) \mid i = 4,\dots,7\} > 110 \\ 0 & \text{otherwise} \end{cases}$$

$$I_{n,5} = \begin{cases} 1 & s_n(1)v_f = 60 \\ 0 & \text{otherwise} \end{cases}$$

$$I_{n,6} = \begin{cases} 1 & s_n(1)v_f = 80 \\ 0 & \text{otherwise} \end{cases}$$

$$I_{n,7} = \begin{cases} 1 & s_n(1)v_f = 100 \\ 0 & \text{otherwise} \end{cases}$$

$$I_{n,8} = \begin{cases} 1 & s_n(1)v_f = 120 \\ 0 & \text{otherwise} \end{cases}$$

$$I_{n,9} = \begin{cases} 1 & s_n(2)v_f = 60 \\ 0 & \text{otherwise} \end{cases}$$

$$I_{n,10} = \begin{cases} 1 & s_n(2)v_f = 80 \\ 0 & \text{otherwise} \end{cases}$$

$$I_{n,11} = \begin{cases} 1 & s_n(2)v_f = 100 \\ 0 & \text{otherwise} \end{cases}$$

$$I_{n,12} = \begin{cases} 1 & s_n(2)v_f = 120 \\ 0 & \text{otherwise} \end{cases}$$

The ten-dimensional state description, extended with the twelve indicator variables, gives a state space consisting of 22 dimensions. The additional indicator variables do not provide additional information, in the sense that the features they represent are already present in the other state variables. However, it adds additional contextual information that is useful to enrich the state space. The first two indicators represent whether there is lower speed (i.e., congestion) and an increased density. Variable three and four represent whether speed limits are active and whether there is free flow. The remaining eight variables encode which speed limits were activated in the current and previous state.

Actions

The original action space $A = \{1,2,3,4\}$ consisted of four actions, representing the speed limits 60, 80, 100 and 120, respectively. The function $lim(a)$ remains unchanged. The reward function was responsible for preventing the speed limits from alternating and ensured that speed limits were increased and decreased smoothly. In the new formulation we decided to define this explicitly in the action space, by defining feasible actions for each state. The conditions that need to be true to apply an action in state s_n are shown in Table 5.1.

Reward function

The intuitive reason to change the actions is that the reward function becomes easier to learn for a function approximator, since two penalty terms can be discarded. Thus, the remaining cases of the reward function in which the agent receives a punishment will only be related to the number of vehicle hours. The reward r_n received by the agent after arriving in state s_{n+1} is as follows:

$$r_n = \begin{cases} 0 & \min\{v_i((n+1)c) \mid i = 1,\ldots,N\} > 101 \\ -VH(n) & \text{otherwise} \end{cases} \tag{5.3}$$

Action	Conditions
1	$(s_n(1)v_f = 100 \vee s_n(1)v_f = 120) \wedge \neg(s_n(2)v_f = 120 \wedge s_n(1)v_f \neq 120)$
2	$(s_n(1)v_f = 80 \vee s_n(1)v_f = 100 \vee s_n(1)v_f = 120)$ $\neg(s_n(2)v_f = 100 \wedge s_n(1)v_f \neq 100)$
3	$(s_n(1)v_f = 60 \vee s_n(1)v_f = 80 \vee s_n(1)v_f = 100)$ $\neg(s_n(2)v_f = 80 \wedge s_n(1)v_f \neq 80)$
4	$(s_n(1)v_f = 60 \vee s_n(1)v_f = 80) \wedge \neg(s_n(2)v_f = 60 \wedge s_n(1)v_f \neq 80)$

Table 5.1: Conditions that should to be true to apply an action.

where $VH(n)$ is defined as:

$$VH(n) = T \sum_{p=nc}^{(n+1)c} \left(\sum_{i=1}^{N} [M_i L_i k_i(p)] + \sum_{i=0}^{N} w_i(p) \right)$$

Again, the first case represents that no punishment will be given if there is currently no significant congestion. In all other cases, the agent receives a punishment proportional to the number of vehicle hours between the previous and the current control time steps.

5.3 Preserving the Markov property

We observed in the previous chapter that the original formulation as a Markov Decision Process is not Markovian. There is hidden information that the agent cannot derive from the state variables of the current state. Describing the problem in such a way that the state is Markov is not obvious in this case, since we would need to include information regarding flow and future traffic demand profiles, for example. It is unrealistic to assume that all this information is available in practice to define the state of a highway or road network. Therefore, we aim for a different solution to make the state representation ‘more Markov’. Dealing with non-Markov decision tasks has been studied in the literature. We will use and apply a method that is discussed in [30].

The method we will use allows the agent to keep previous states in memory. This history of states can be useful to identify hidden information, since the combination of the current state and n previous states may provide more information about the current state than the current state itself. Instead of using the state variables as the input to a function approximator, a history of states can be used as well. A schematic representation of this architecture is shown below in Figure 5.5.

The parameter n we use here represents a variable number and does not relate to the parameter n we used previously. The choice of the parameter n can be difficult, since it is unknown how many previous states are required to create an extended state space that is Markovian, and in general it is unknown whether this is possible to do. Additionally, if a neural network is used as a function approximator, then a large number of input nodes requires more training to learn the Q-function and thus more Q-learning episodes are required.

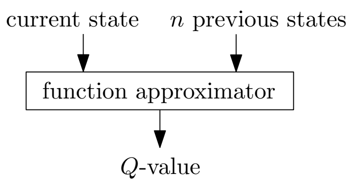


Figure 5.5: Q-value as a function of stored states.

5.4 Experiments

In this section we present the results of our experiments. The purpose of these experiments is to show that the improved speed limit policy generates better policies than the algorithm from Chapter 4. Our experimental setup is the same, except that we use artificial neural networks as a function approximator, and we solve the problem using the modified formulation as a Markov Decision Process. The artificial neural network is a multi layer neural network with one hidden layer. The learning rate is set to 0.01 and there is one bias input node. We use the stored state architecture shown in Figure 5.5, and we keep one previous state in memory. This means that the neural network has 44 state input and one bias input that is always equal to one. The number of hidden nodes is equal to 45.

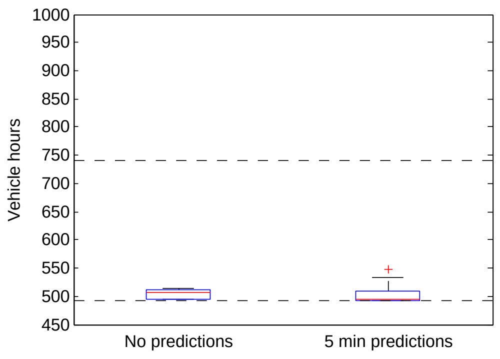
Policies for individual scenarios

In our first experiment we generate policies for individual scenarios, in the same way as we did in the previous chapter. We used the same scenarios and the same number of learning episodes. The only difference is that we use the modified state description and neural networks as a function approximator. The results of the experiment are shown in Figure 5.6 and Table 5.2.

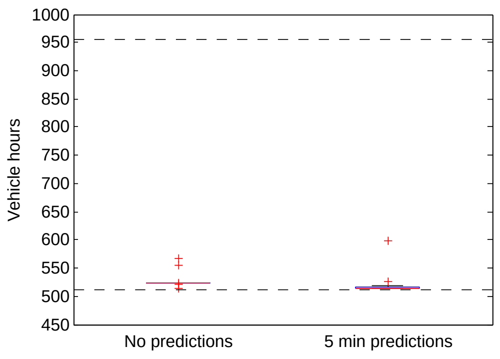
We observe that the improved algorithm performs consistently better and generates better policies. Compared to the results in Table 4.2, we can see that the standard deviation, mean and median have decreased in all the cases we considered. If we take predictions into account, the best policies found (denoted by 'min') are optimal or very close to the optimal policy. It is also interesting to observe that the policy quality of the third scenario has improved and the variation of the solution quality is much smaller. This means that the improved algorithm is also able to learn policies for scenarios with high traffic demands.

To investigate the influence of the number of episodes on the policy quality, we repeated the experiment, but now we ran 20000 episodes. The results of the experiment are shown in Figure 5.7 and Table 5.4. We observe that the standard deviation of the solution quality has decreased, compared to the results discussed above. The only exception is scenario 2, for which the standard deviation with traffic predictions has become slightly worse. However, this may be caused by the outlier that is visualized in Figure 5.7. The median and best solution found have improved after increasing the number of episodes, so we argue that this is not problematic.

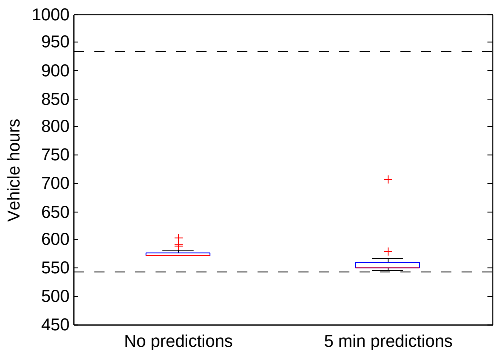
Based on the experiments we presented until now, we can conclude that the improved policy algorithm performs better than the algorithm presented in Chapter 4. It does not leave



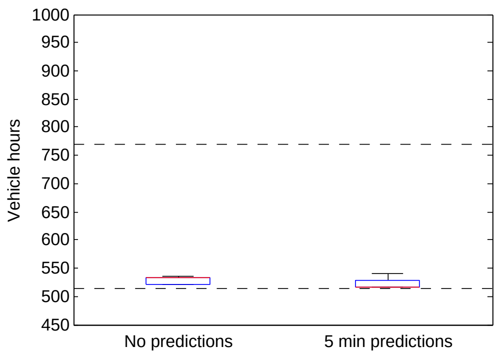
(a) Scenario 1.



(b) Scenario 2.



(c) Scenario 3.



(d) Scenario 4.

Figure 5.6: Policies for individual scenarios: policy quality (5000 episodes).

	No control	Opt	Min	Max	Median	Mean	Std
Scenario 1	740.8	491.6	495.6	513.1	505.8	504.1	7.8
Scenario 2	954.5	511.2	514.1	567.1	523.2	525.6	12.9
Scenario 3	933.8	543.5	572.5	602.4	572.5	576.9	8.6
Scenario 4	768.6	514.5	521.0	534.6	532.2	528.0	5.9

(a) Vehicle hours without predictions.

	No control	Opt	Min	Max	Median	Mean	Std
Scenario 1	740.8	491.6	491.6	548.2	494.7	501.9	15.2
Scenario 2	954.5	511.2	514.1	598.9	514.1	519.7	18.9
Scenario 3	933.8	543.5	544.1	707.8	551.1	569.3	48.1
Scenario 4	768.6	514.5	516.4	539.8	516.4	521.1	7.8

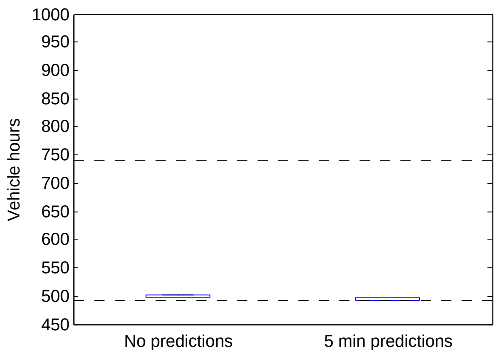
(b) Vehicle hours with 5 minute predictions.

Table 5.2: Policies for individual scenarios: vehicle hours (5000 episodes).

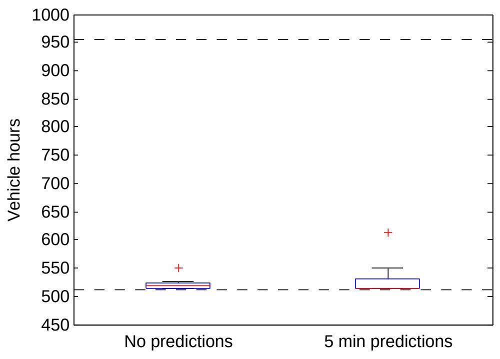
	Scenario 1	Scenario 2	Scenario 3	Scenario 4
No predictions				
Min SL	495.6	523.0	549.1	525.1
Min SLI	495.6	514.1	572.5	521.0
	−0.0%	−1.7%	+4.3%	−0.8%
Max SL	546.0	598.5	933.6	589.0
Max SLI	513.1	567.1	602.4	534.6
	−6.0%	−5.2%	−35.5%	−9.2%
Median SL	520.3	550.4	735.3	538.5
Median SLI	505.8	523.2	572.5	532.2
	−2.8%	−4.9%	−22.1%	−1.2%
Mean SL	520.8	553.9	716.1	542.2
Mean SLI	504.1	525.6	576.9	528.0
	−3.2%	−5.1%	−19.4%	−2.6%
Std SL	13.8	17.8	142.6	17.2
Std SLI	7.8	12.9	8.6	5.9
	−43.5%	−27.5%	−94.0%	−65.7%
5 minute predictions				
Min SL	491.6	513.7	548.6	522.5
Min SLI	491.6	514.1	544.1	516.4
	−0.0%	+0.1%	−0.8%	−1.2%
Max SL	676.4	598.9	775.3	560.1
Max SLI	548.2	598.9	707.8	539.8
	−19.0%	−0.0%	−8.7%	−3.6%
Median SL	512.2	534.2	565.5	545.6
Median SLI	494.7	514.1	551.1	516.4
	−3.4%	−3.8%	−2.5%	−5.4%
Mean SL	519.2	540.5	580.4	542.9
Mean SLI	501.9	519.7	569.3	521.1
	−3.3%	−3.8%	−1.9%	−4.0%
Std SL	40.1	20.8	50.3	12.4
Std SLI	15.2	18.9	48.1	7.8
	−62.1%	−9.1%	−4.4%	−37.1%

Table 5.3: Comparison results Chapter 4 and Chapter 5 (5000 episodes).

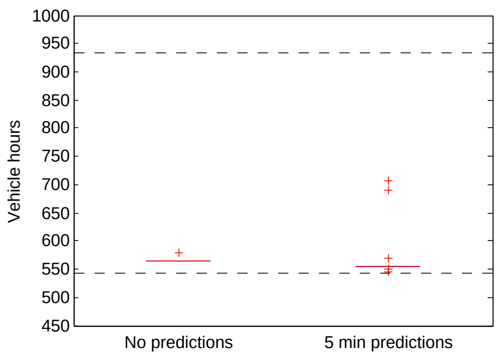
much room for further improvement, though, because the resulting policies are very close to the lower bounds we computed. To summarize, we will present a detailed comparison of the results from Chapter 4 and the experiments in this section. The comparison is shown in Table 5.3. In this table, SL denotes the speed limit policy algorithm from the previous chapter, and SLI denotes the improved version we described in this chapter. We only compare the experiments in which we executed 5000 episodes.



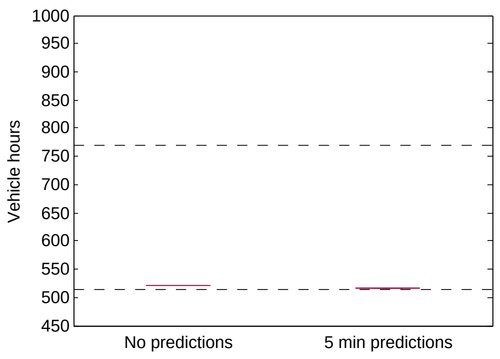
(a) Scenario 1.



(b) Scenario 2.



(c) Scenario 3.



(d) Scenario 4.

Figure 5.7: Policies for individual scenarios: policy quality (20000 episodes).

	No control	**Opt**	**Min**	**Max**	**Median**	**Mean**	**Std**
Scenario 1	740.8	491.6	496.9	501.5	496.9	498.5	2.2
Scenario 2	954.5	511.2	514.1	549.4	519.2	520.6	8.6
Scenario 3	933.8	543.5	564.6	578.1	564.6	565.3	3.0
Scenario 4	768.6	514.5	521.0	521.0	521.0	521.0	0.0

(a) Vehicle hours without predictions.

	No control	**Opt**	**Min**	**Max**	**Median**	**Mean**	**Std**
Scenario 1	740.8	491.6	491.6	496.8	496.8	494.9	2.5
Scenario 2	954.5	511.2	513.7	612.9	514.1	525.2	23.2
Scenario 3	933.8	543.5	544.1	707.8	553.8	569.1	44.7
Scenario 4	768.6	514.5	516.4	516.4	516.4	516.4	0.0

(b) Vehicle hours with 5 minute predictions.

Table 5.4: Policies for individual scenarios: vehicle hours (20000 episodes).

Comparison with initial algorithm

To investigate whether function approximation with neural networks scales better than tile coding, we measured the running time of the policy learning algorithm with tile coding and neural networks. The reward function was exactly the same, and we repeated the experiment for an increasing state space dimensionality. Figure 5.8 shows how the running time grows if the dimensionality is increased. Note that a log scale is used for the *y*-axis of the graph. To generate this graph, we took the average of five runs for each number of state variables. Due to the large running time of tile coding for states with more than six dimensions, we were unable to obtain more results for this approximation technique. However, it is clear that neural networks scale better than tile coding, allowing us to extend the state space with additional information.

In our experiments, we observed that the improved algorithm performs better than the algorithm from the previous chapter. There may be multiple reasons for this, since we changed the approximation technique and state representation. To find out whether the approximation technique is responsible for this improvement, or the extended state description, we did an experiment in which we use the MDP formulation from the previous chapter, combined with neural networks as a function approximator. We applied this algorithm to each scenario and we set the number of episodes equal to 5000. The results are shown in Table 5.5. We can conclude that the policies become worse if we only replace the function approximator with neural networks, and thus the approximation technique itself is not the main reason that the policy quality improved. We extended the state space with additional information, and this resulted in better policies. Using the larger state space would not be possible without neural networks as approximation technique, so both the approximation technique and improved MDP formulation contributed to the improvements.

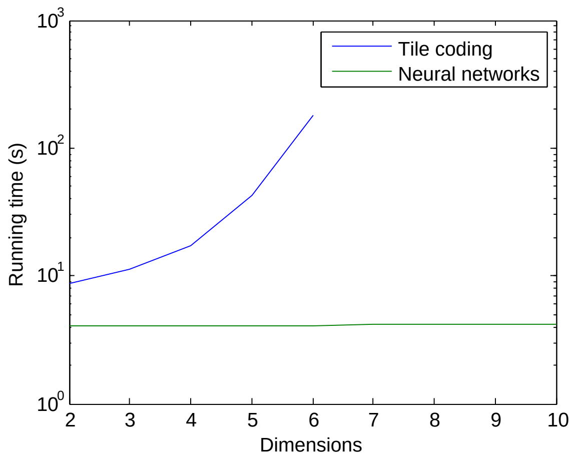


Figure 5.8: Running time comparison tile coding and neural networks.

	No control	Opt	Min	Max	Median	Mean	Std
Scenario 1	740.8	491.6	515.9	768.8	725.8	698.6	79.8
Scenario 2	954.5	511.2	751.5	976.2	751.5	799.0	85.0
Scenario 3	933.8	543.5	777.9	777.9	777.9	777.9	0.0
Scenario 4	768.6	514.5	541.0	798.8	760.6	731.5	82.9

(a) Vehicle hours without predictions.

	No control	Opt	Min	Max	Median	Mean	Std
Scenario 1	740.8	491.6	768.8	768.8	768.8	768.8	0.0
Scenario 2	954.5	511.2	598.9	976.2	976.2	908.4	142.4
Scenario 3	933.8	543.5	777.9	777.9	777.9	777.9	0.0
Scenario 4	768.6	514.5	798.8	798.8	798.8	798.8	0.0

(b) Vehicle hours with 5 minute predictions.

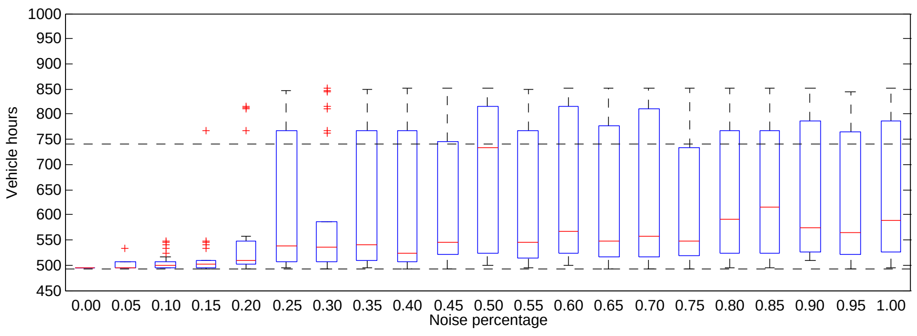
Table 5.5: Improved algorithm with initial MDP formulation: vehicle hours (5000 episodes).

Robustness study

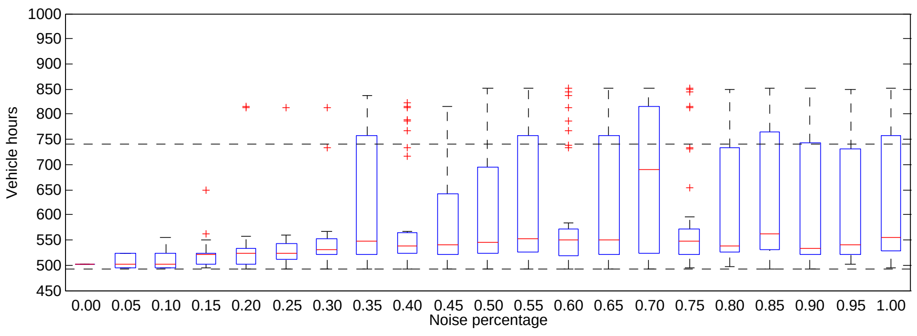
In practice, it is unrealistic to assume that speed and density can be measured with 100 percent accuracy. Therefore, we performed a robustness study in which we introduced noise. We generated four policies to solve the first traffic scenario, using the improved algorithm that executed 5000 episodes. In the learning algorithm we did not introduce noise, because generally these policies will be learned offline and the algorithm will never be using real time data. To investigate the robustness of the policies, we ran 50 simulations of scenario 1, for different noise percentages. The noise is drawn from a uniform distribution and added to the speed and density values in the state description during simulation. The parameters of this distribution can be derived from the noise percentage, reported in Figure 5.9 and 5.10.

The results for the simulations without predictions are shown in Figure 5.9. We observe that for an error percentage up to 10 percent, the policies perform well. We can also conclude that if the measurement data is erroneous, the actions suggested by the policies are not useful and make it worse, compared to the situation in which there is no control at all. This can be concluded from Figure 5.9, because there are some cases in which the number of vehicle hours exceeded the baseline representing the no control case.

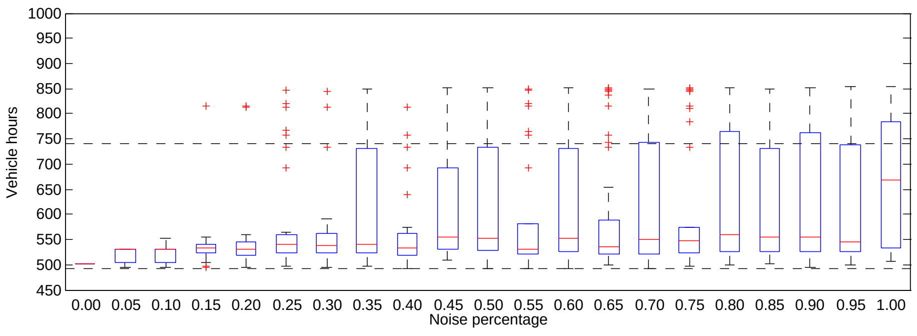
Figure 5.10 shows the results for the same experiment with 5 minute predictions. Interestingly, the policies with a predictive state description are more robust, in the sense that they still have reasonable performance for error percentages higher than 10 percent. For example, considering policy 1 with error percentage 0.35, the mean and standard deviation are 613.3 and 132.1 without predictions. If we take predictions into account then the mean and standard deviation are 513.6 and 13.6, which is clearly better. The differences between the distributions of vehicle hours can also be compared in Figure 5.9 and 5.10.



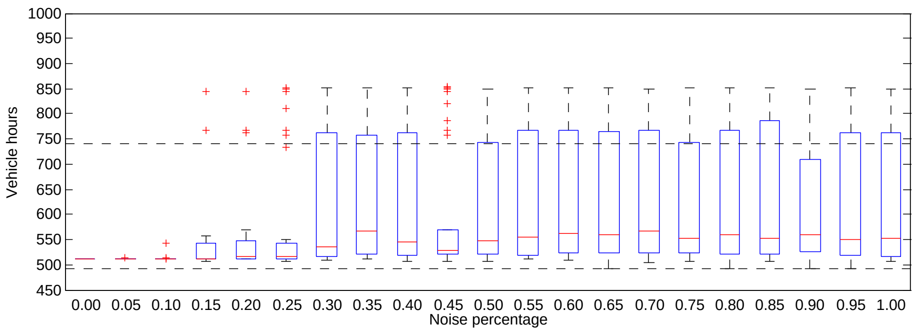
(a) Policy 1.



(b) Policy 2.

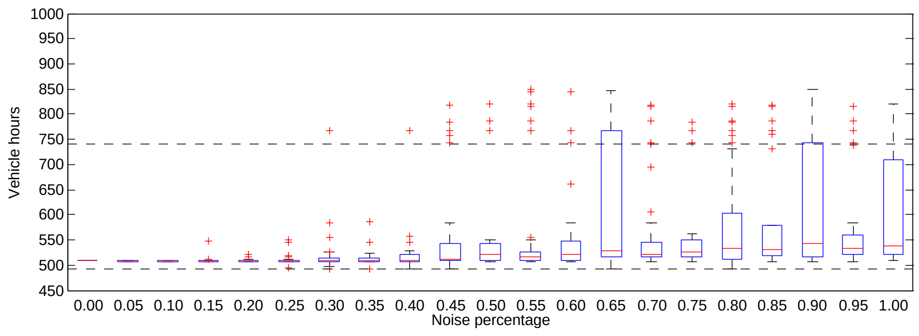


(c) Policy 3.

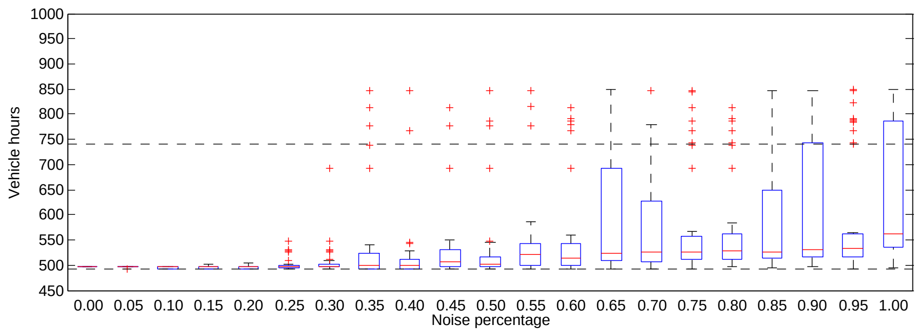


(d) Policy 4.

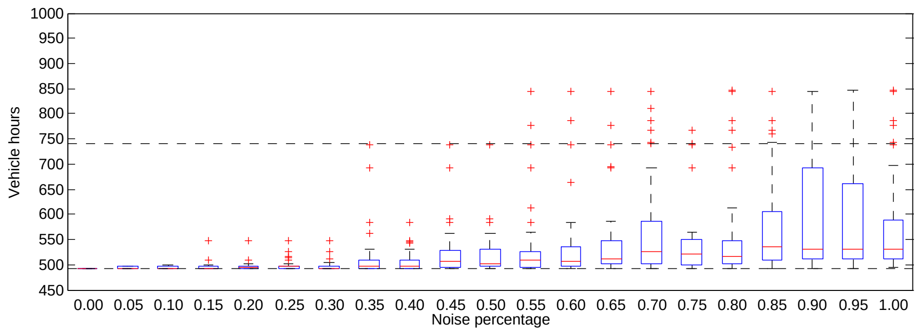
Figure 5.9: Robustness for increasing noise rate on scenario 1, without predictions.



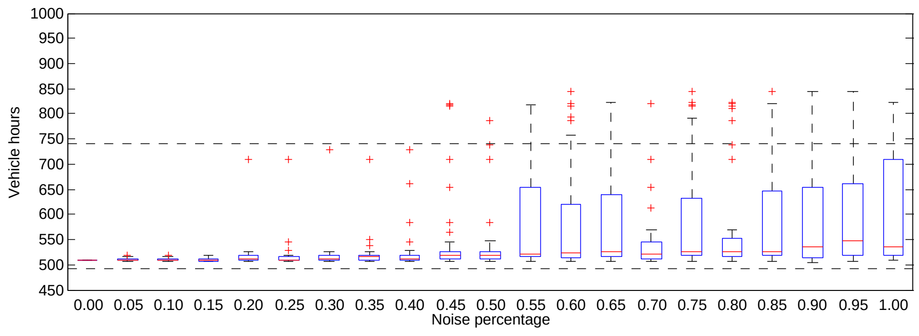
(a) Policy 1.



(b) Policy 2.



(c) Policy 3.



(d) Policy 4.

Figure 5.10: Robustness for increasing noise rate on scenario 1, with 5 minute predictions.

Example policy execution

Until now we only discussed policy quality in terms of vehicle hours, but we did not investigate how the traffic flow changed over time, as a consequence of policy intervention. Therefore, we generated a policy for each scenario and visualized the results. Figure 5.11 depicts how the average speed of the sections changes over time. The leftmost color map shows the situation without speed control. Clearly, in all scenarios traffic slows down completely. If we use the improved policy learning algorithm and apply the resulting policy to the same scenario, we obtain the second color map.

The rightmost color map shows the corresponding speed limits that were assigned to section 2 to 6. Here we can observe that the assigned speed limits to the sections are constant at every time step, as expected. We can also observe that there is a correspondence between the middle color maps and the rightmost color maps, since the speed of section 2 to 6 decreases to the speed limit value if speed limits are activated.

In the third scenario, the traffic demand is too high to let congestion resolve completely within one hour, but we conclude that the policy cannot do much better in this case, since it is impossible to assign speed limits lower than 60.

Another remark that has to be made is that the policies we used to generate Figure 5.11 are the result of one run of the policy learning algorithm. A second run of the algorithm may give us slightly different policies, which would also give us different results in the color maps. Figure 5.11 illustrates the concept, though, and shows that it is able to control the traffic flow such that it is better than the baseline scenario without control.

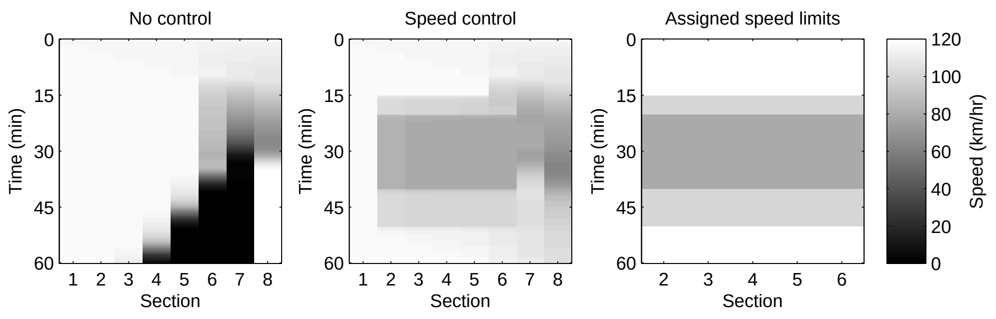
5.5 Discussion

The improved algorithm has shown that it is able to perform consistently better than the algorithm presented in the previous chapter. The combination of neural networks, storing states in memory and including more information in states has shown that it was possible to obtain better results than in the previous chapter.

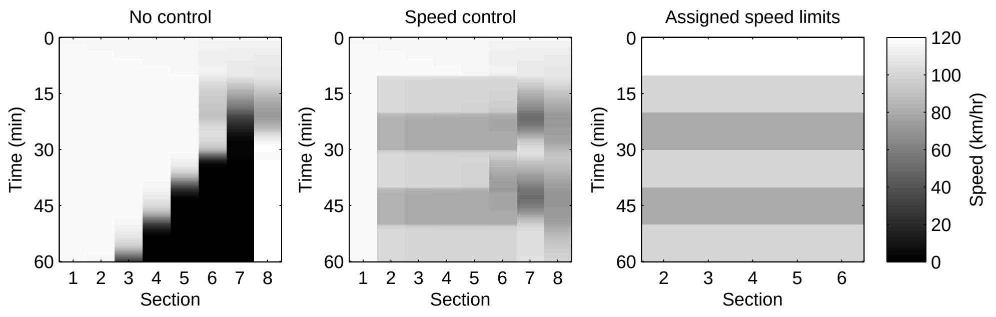
The robustness experiment has shown that policies can also be useful if the measurements of the state variables is not accurate. In the experiment we did not study other distributions, such as a Gaussian distribution or normal distribution. This can be studied in future work. Currently, it is unclear to which extent the policies would be useful in real life scenarios. The reason is that the macroscopic model should be calibrated using real traffic data to accurately simulate the flow of the highway. In our case, it would be interesting to know the accuracy of the model with respect to the A67 highway in the Netherlands. The results presented in this chapter can be considered as theoretical results, proving the concept, but cannot directly be applied in practice.

5.6 Summary

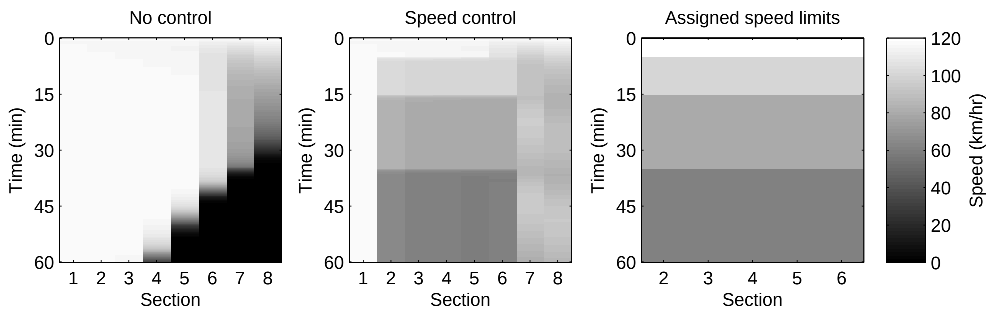
In this chapter we presented an improved version of the speed limit policy algorithm. The main difference between the improved algorithm and the basic algorithm from the previous chapter is that we included a different type of function approximation and we keep states in



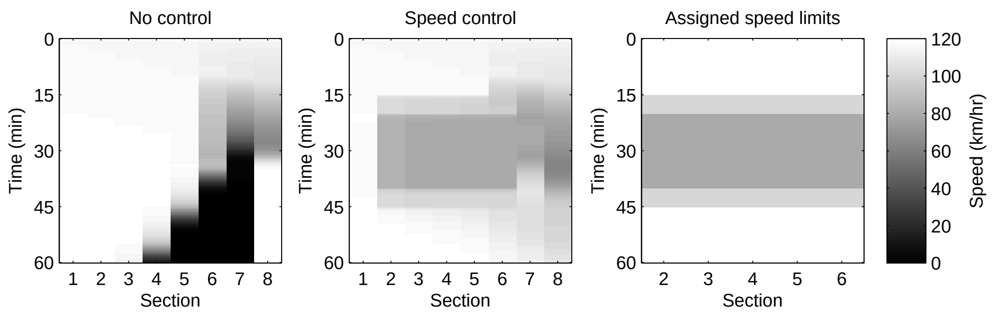
(a) Scenario 1.



(b) Scenario 2.



(c) Scenario 3.



(d) Scenario 4.

Figure 5.11: Speed progress without control and with assigned speed limits.

the memory of the agent. Experiments have shown that the performance is better and that the quality of the resulting policies is consistently better than the policies from the same experiments in the previous chapter. A robustness study has shown that the policies without predictions in the state description perform well if there is at most 10 percent noise in the speed and density measurements. Policies including traffic predictions have also shown to be robust, and may also perform well if the noise percentage is higher than 10 percent.

Chapter 6

Multi-Agent Traffic Control

In the previous chapters we considered the traffic flow optimization problem from the perspective of the entire highway. An advantage of such an approach is that only one Markov Decision Process (MDP) and associated policy is sufficient to regulate flow, but the proposed approach did not include speed limit patterns that can be assigned to sections. In this chapter we present a similar formulation of the traffic flow optimization problem as an MDP, but now from the perspective of individual highway sections. This gives a multi-agent traffic optimization approach. The aim of this formulation is to create a more simple state representation that can be implemented in real scenarios as well. In the first two sections of this chapter we present the new problem description and the MDP formulation, followed by experiments and a discussion of the results.

6.1 Problem description

In this section we describe the problem we attempt to solve using multi-agent policy learning. We consider the optimization problem from the perspective of an individual highway section *i*, as shown in Figure 6.1. We assume that information about highway sections $i-1$, $i+1$ and $i+2$ is also present. Now the problem becomes how we should set the speed limit of section *i*, and when it should be activated.

The additional constraints we impose are the same as in Chapter 4, but we also need to ensure that consecutive speed limits of section *i* and $i+1$ do not lead to dangerous situations. This means that we perform multi-agent optimization, but after determining the speed limits

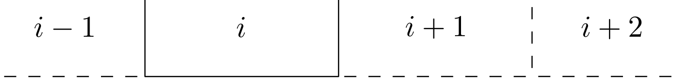


Figure 6.1: Highway section *i* and its neighbors.

$V_i(n)$ of the individual sections i at control time step n, the following constraints should hold:

$$V_i(n) \geq V_i(n-1) - 20$$
$$V_i(n) \leq V_{i+1}(n) + 20$$

These constraints have been derived from [32]. The first one ensures that the speed limit of a section is not decreased too much, while the second constraint ensures that speed limits are smoothly reduced along the highway. The value 20 can be changed to obtain the desired behavior.

The most important reason to create a multi-agent formulation is that it is easier to map onto highway sections in a microscopic simulation environment. Therefore, the aim of this approach is to learn policies that can be tested in a microscopic simulation. We are unable to generate traffic predictions in such a simulation environment, and our current predictions are relatively simple. We are unable to obtain more predictions than the simple predictions we generate using METANET, so this prevents us from studying other interesting directions, such as including probabilistic predictions. This is the reason that we will not study predictions in this chapter.

6.2 Formulation as Markov Decision Process

The formulation as a Markov Decision Process for the multi-agent approach at section level consists of a five dimensional state space. Below we discuss the new state description and the reward function. The action space is not discussed in this section, because it did not change compared to the previous chapters. The aim of the modified definition is that we enable individual sections to change their maximum speed after observing the traffic conditions of neighboring sections.

State description

The state description is shown below. The speed values used in the state description correspond to the average speeds of section i itself, and its neighbors $i-1$ and $i+1$. The subscript i refers to the numbers of the sections. Thus, state $s_{i,n}$ is the state of section i defined at simulation time step cn and $a_{i,n-1}$ is the action that was selected for section i at simulation time step $c(n-1)$.

$$s_{i,0} = \left(\frac{120}{v_f}, \frac{120}{v_f}, \frac{v_{i-1}(0)}{v_f}, \frac{v_i(0)}{v_f}, \frac{v_{i+1}(0)}{v_f} \right)$$
$$s_{i,n} = \left(\frac{lim(a_{i,n-1})}{v_f}, s_{i,n-1}(1), \frac{v_{i-1}(cn)}{v_f}, \frac{v_i(cn)}{v_f}, \frac{v_{i+1}(cn)}{v_f} \right)$$

Note that we make the assumption that the agent associated with a section has access to the speed values of adjacent sections and that decisions are made at the same time steps.

Reward function

The reward function is able to give a penalty that is proportional to the number of vehicle hours that has been realized on highway section i, $i+1$ and $i+2$. We also take section $i+2$ into account, because it is desirable that speed limits are activated if congestion occurs downstream, and the information about section $i+1$ may not be sufficient. The reward $r_{i,n}$ received by the agent i after arriving in state $s_{i,n+1}$ is as follows:

$$r_n = \begin{cases} 0 & \min\left\{v_j((n+1)c) \mid j \in \{i, i+1, i+2\}\right\} > 101 \\ -VH(n) & \text{otherwise} \end{cases}$$

where $VH(n)$ is defined as:

$$VH(n) = T \sum_{p=nc}^{(n+1)c} \left(\sum_{j=i}^{i+2} [M_j L_j k_j(p)] \right)$$

This approach gives a policy that can be generally applied to highway sections, and changes the speed of a section, without considering information other than the speed values of the adjacent sections. In the term $VH(n)$ we ignore the queue lengths w, because we only optimize the vehicle hours on the main lanes. The queue lengths can be included in the reward function, but in our implementation the policy quality became worse, which we did not study further.

As we observed previously, the states do not contain all the information that is required to calculate the rewards. In our implementation this is not an issue since all the variables of the METANET model can be used, but in general this is considered inconvenient. The reason to use the current state description is that we want to create policies that can be applied in a microscopic simulation, where is it more difficult to obtain accurate information regarding densities. To demonstrate that it is also possible to define an MDP in which all the required information is included in the states, we will perform an experiment later in this chapter.

Safety rules

As we mentioned previously, we need to ensure that the speed limits that have been selected for individual sections give a speed limit pattern that can be safely used in practice. This is not really relevant during policy learning, but in practice this matters. Note that the safety rules are not required in the single-agent case, because a speed limit consisting of one single value is inherently safe to apply.

Suppose we are given a sequence of speed limits $V_1(n), V_2(n), \ldots V_k(n)$ at simulation time step cn, determined by the agents associated with highway section $1, \ldots, k$. We can apply the following function for $i = 1, \ldots, k-1$ to make sure that the two constraints from Section 6.1 hold [32]:

$$\overline{V_i}(n) = \begin{cases} \overline{V_i}(n-1) - 20 & V_i(n) \leq \overline{V_i}(n-1) - 20 \\ \overline{V_{i+1}}(n) + 20 & V_i(n) \geq \overline{V_{i+1}}(n) + 20 \\ V_i(n) & \text{otherwise} \end{cases}$$

The resulting speed limits $\overline{V_1}(n), \overline{V_2}(n), \ldots, \overline{V_k}(n)$ can be safely applied to regulate traffic flow. The safety rule does not strictly relate to the definition as MDP, but is required during the execution of the resulting policies.

6.3 Experiments

To evaluate the performance of multi-agent traffic flow control, we performed experiments similar to the experiments from the previous chapters. We use the scenarios presented in Chapter 4, but we allow the agents to change their speed limit every minute. This means that the control time step multiplier c is equal to 4 instead of 20.

As a Q-function approximator we use artificial neural networks, and the agents do not keep previous states in memory. For learning we use the regular Q-learning algorithm for every agent, running 20000 episodes, but they update a joint Q-function. The reason behind this choice is that we want to learn a general policy that characterizes the speed control behavior of the sections, rather than learning policies for specialized control of individual sections. However, each section is using this policy to select his own speed limit value.

During learning we do not apply the safety rule, but it is used when applying a policy to calculate the number of vehicle hours. The lower bounds we obtained in the previous chapters are no longer valid, since they were based on the assumption that the speed limits are the same for each section. Due to the number of combinations that form a valid speed limit sequence, we have not been able to enumerate all solutions to compute lower bounds for multi-agent traffic control.

Policies for multiple scenarios

In the first experiment we learned 20 policies, and each policy was sequentially trained on the four scenarios. This results in 20 policies that should be capable of solving every scenario, instead of being a specialized policy that solves just one scenario. The reason to learn joint policies rather than policies for individual scenarios is that it is desirable in practice to have policies that can be used under several different circumstances. By training the policies on multiple scenarios, the learning algorithm encounters more state-action pairs, and it is expected that the resulting policies can be applied more generally. The quality of the policies is shown in Figure 6.2 and Table 6.1 below. The horizontal lines represent the baselines corresponding to the scenarios. As mentioned before, we did not compute optimal solutions for the multi-agent case. Recall that the boxplots visualize the distribution of policy quality for the same policies, applied to different scenarios.

From the results we can conclude that the multi-agent policy learning algorithm is able to generate policies that can be applied to multiple scenarios, and the deviation of their quality is quite small. The best policies found are still close to the lower bounds we used in the previous chapters, so based on that we can argue that multi-agent control performs well, compared to single-agent speed control.

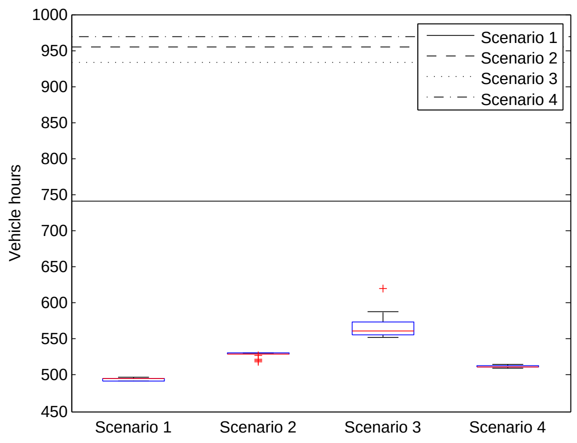


Figure 6.2: One policy for multiple scenarios: vehicle hours.

	No control	Min	Max	Median	Mean	Std
Scenario 1	740.8	492.2	496.7	494.6	494.4	1.5
Scenario 2	954.5	517.9	530.9	529.8	528.5	3.7
Scenario 3	933.8	551.9	620.8	561.5	569.4	20.5
Scenario 4	968.6	509.8	514.9	512.2	512.2	1.7

Table 6.1: One policy for multiple scenarios: vehicle hours.

Enhanced state description

In the previous chapters we observed that the state description does not contain all the information that is required to calculate the reward values. We learn policies by interacting with the METANET model, so the agent has access to all the calculated variables, but in general this is considered to be inconvenient. The reward function in our MDP formulation is defined in such a way that it uses density values that occur in between control steps. To demonstrate that we can also learn policies when the state contains this information, we did an experiment. Instead of defining a state $s_{i,n}$ in terms of the speed values at time cn only, we included the speed and density values of section i to $i+2$ for time steps $c(n-1)$ to cn in the state associated with agent i. In this case, a state does not represent the traffic conditions at one specific time step, but it also includes information about previous simulation time steps that have led to the new state. This means that the reward value can be expressed as a function of the state variables.

We used Q-learning combined with the neural network function approximation technique to learn 20 policies for each scenario. The results are shown in Figure 6.3 and Table 6.3. Compared to the previous results, the policy quality decreased. A possible explanation for this observation is that the dimensionality of the states increased, since we included more state variables, and that it becomes more difficult for a neural network to identify the Q-function that it should approximate. However, compared to the baselines we calculated earlier, there is still a significant improvement.

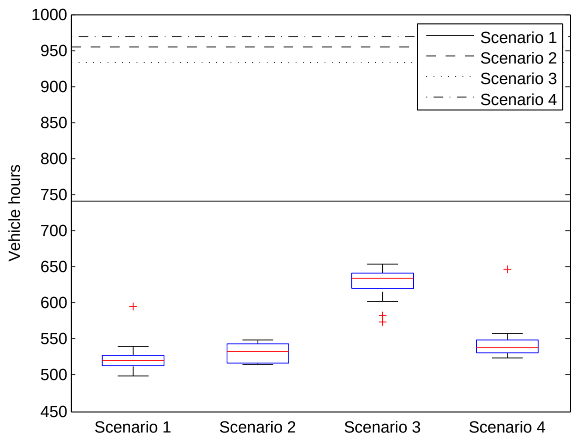


Figure 6.3: Learned policies with extended state description: vehicle hours.

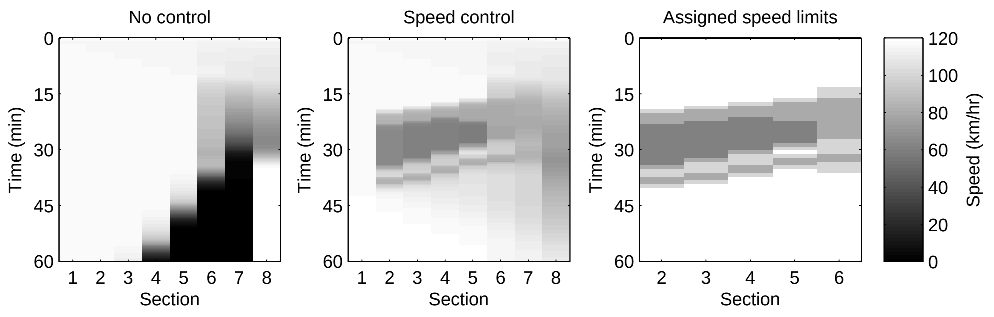
	No control	Min	Max	Median	Mean	Std
Scenario 1	740.8	499.2	594.2	519.9	522.1	20.2
Scenario 2	954.5	514.1	548.1	533.4	531.1	13.6
Scenario 3	933.8	574.6	654.1	634.6	627.6	21.3
Scenario 4	968.6	524.0	646.2	538.5	544.0	26.0

Table 6.2: Learned policies with extended state description: vehicle hours.

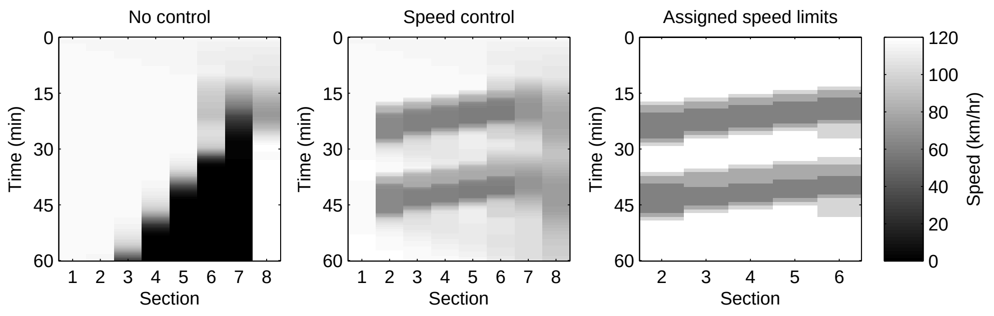
The fact that our state description does not provide sufficient information to calculate the rewards was identified as one of the shortcomings of our method, but in this experiment we addressed this issue by adding more information to the states. In general, we aim to learn policies that can be tested in a microscopic simulation environment by mapping it onto highway sections. Due to running time issues, it is problematic to sample the density and speed values at every simulation time step, so we did not study this in more detail.

Example policy execution

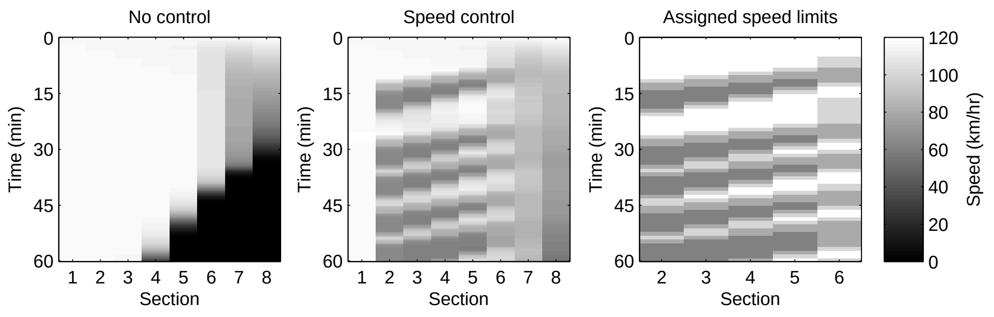
For each scenario we selected the best policy from the previous experiment, and evaluated the policies through macroscopic simulation. The results are shown in Figure 6.4. It is clear that the assigned speed limits at a control time step may differ, in the sense that adjacent sections may have different speed limits. The speed limits close to the congested area are activated first, and they propagate backwards. Compared to the results in Figure 5.11, we observe that the speed limits are deactivated earlier. In the second scenario there are two demand peaks, and after the first peak congestion resolves almost completely, but in the single-agent case it does not. In between the two demand peaks, the speed limits are deactivated when using multi-agent control and congestion resolved.



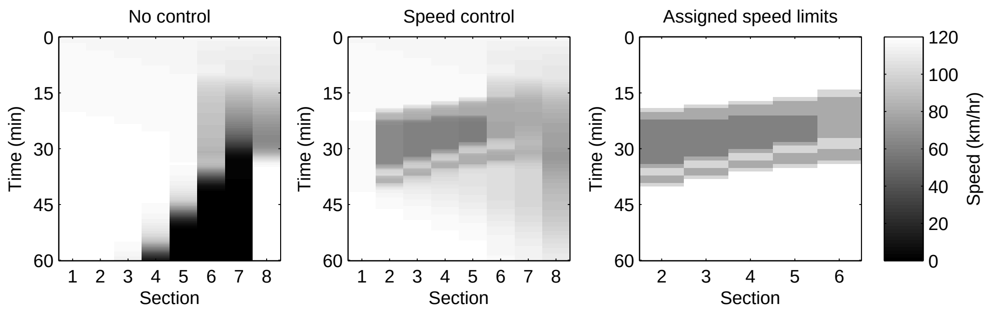
(a) Scenario 1.



(b) Scenario 2.



(c) Scenario 3.



(d) Scenario 4.

Figure 6.4: Speed progress without control and with assigned speed limits.

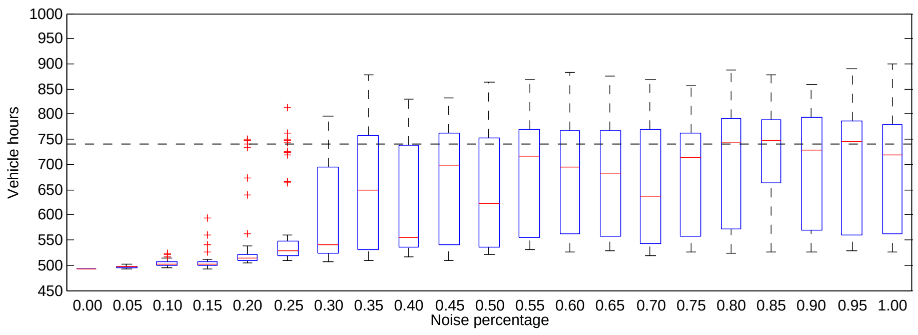
Robustness study

We performed a robustness study similar to the experiment from the previous chapter. We generated four policies that solve scenario 1, using 20000 learning episodes. To investigate the robustness of the policies, we ran 50 simulations of scenario 1, for different noise percentages. The noise is drawn from a uniform distribution and added to the speed values in the state description during simulation. The results are shown in Figure 6.5. We can conclude that the generated policies in the multi-agent case also perform well. For noise percentages up to 20 percent the deviation of policy quality is relatively small. If there is more noise, the policies perform worse, and sometimes they may regulate traffic in such a way that it becomes worse compared to the baseline.

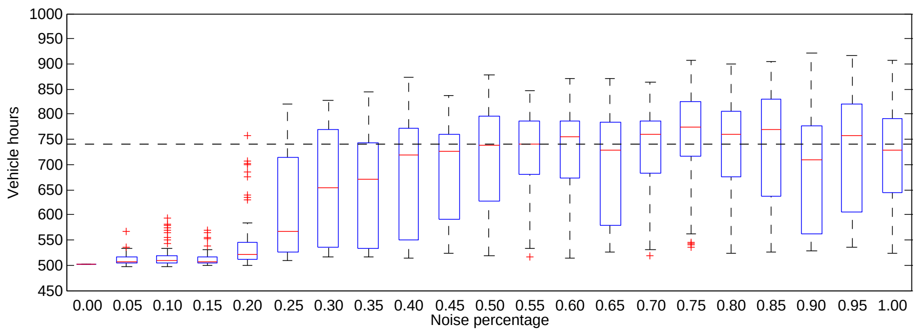
6.4 Discussion

The policies that were generated in the experimental section of this chapter have shown to be capable of solving multiple traffic scenarios. Another advantage is that they operate at the level of individual sections. As can be seen in Figure 6.4, this may result in smoother speed patterns, rather than assigning the same value to all upstream sections. We did not show that the policies are close to the optimal solution. However, there is strong evidence that multi-agent policy learning delivers policies of sufficient quality, since the number of vehicle hours is close to the optimal solutions we found in the previous chapters, and sometimes it is even less. Based on the color maps that were created, one may argue that the assigned speed limits are less stable, in the sense that they change more frequently. An improved version of the safety rule may ensure more safety in practice, if desired. The most important improvement that we presented in this chapter is that the generated policies are more general, since they describe the behavior of individual sections, and do not depend on the layout of the highway (e.g., number of sections).

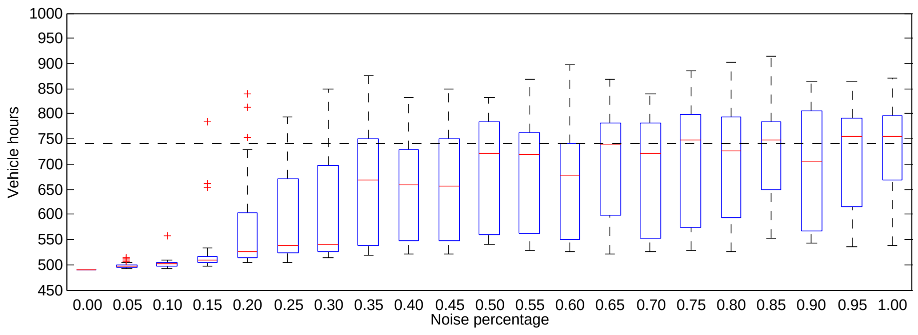
One of the disadvantages of the current approach is that it does not learn different individual policies for sections. During the learning episodes, the same Q-function approximator is updated for every section. This means that the outcome of the algorithm is one single policy, even if there are multiple highway sections involved. Alternative solutions may define an MDP in which the state is composed by joining the individual states together, and the action space may be defined as the Cartesian product of the individual action spaces. Another issue may be that the actions performed at section i do not only influence the traffic behavior of section $i-1$ and $i+1$, but also influence the conditions of other sections. Furthermore, learning for section i may be influenced by the action selection behavior of its neighbors. One might argue that the reward should be defined exclusively in terms of state variables, which is not the case in our solution. During learning there is full access to all variables associated with the sections, and since the learning methods will never interact will real traffic systems, we do not consider this as a problem. An experiment has shown that policies can still be learned if all information to calculate the reward signal is available in the state description.



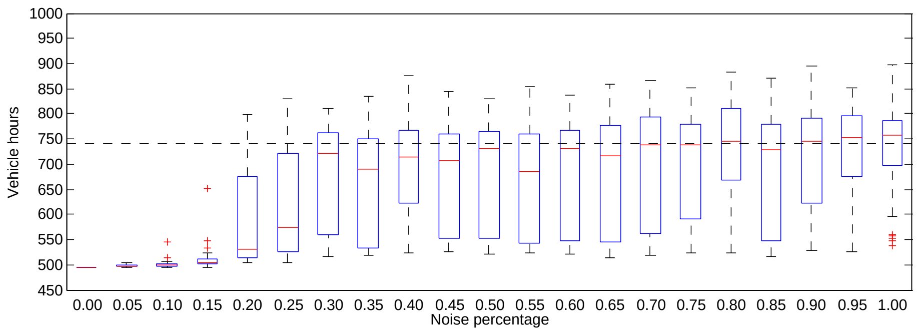
(a) Policy 1.



(b) Policy 2.



(c) Policy 3.



(d) Policy 4.

Figure 6.5: Robustness for increasing noise rate on scenario 1.

6.5 Summary

In this chapter we presented a different traffic regulation method, based on multi-agent control. We have shown that we can define an MDP for each individual section of the highway, to learn how it should select its speed limits, based on the information about neighboring sections. Experiments have shown that the multi-agent control method is able to reduce congestion, and in some cases it performs even better than the learning algorithms we presented in previous chapters. There is still room for improvement, though, because we now learn one single policy and maintain one function approximator representing the Q-function. Instead, we may be able to devise a method that learns specialized policies for each section.

Chapter 7

Probabilistic Policy Reuse

The algorithms we discussed previously are able to learn speed limit policies from scratch. This may not always be the most efficient way of learning, since previously learned policies implicitly contain valuable information about how to solve traffic congestion in different road networks and different scenarios. In this chapter we present a probabilistic policy reuse strategy, in which previously learned policies can be efficiently reused to bias the action selection procedure in the early stages of learning a new policy. In the first section of this chapter we argue why it can be useful to reuse existing policies. Then we present a modified action selection procedure based on existing literature, which we will evaluate in the remaining parts of the chapter.

7.1 Motivation

As demonstrated in the previous chapters, a speed limit policy algorithm can be used to learn policies from scratch. In practice, however, there may be situations in which this is not the most efficient approach to learn policies, and in such cases it may be desirable to reuse knowledge present in existing policies. For example, there can be road controllers applied to specific highways. In such cases, the layout of this specific highway is taken into account by the learning algorithm. The policies are likely to work for this highway, but may be useless to accurately reduce congestion on other highways. The desired policy for another highway may be similar, though, so then it is desirable to transfer knowledge from a previously learned policy to a new policy. There can also be multiple policies to handle different traffic scenarios. For example, specific policies may have been learned to handle a peak hour during typical Monday mornings. A policy for typical peak hours on Thursday can be learned from scratch, but probably the desired policy is similar to the policy for Monday morning. In such cases, transferring knowledge may be useful.

In Chapter 4, we have shown that general policies can be learned to handle multiple scenarios, but as explained above, there can still be situations in which it is useful to learn separate policies. To learn new policies more efficiently in such situations, we show in the remainder of this chapter that policies can be reused to accelerate the learning techniques from Chapter 5.

7.2 Biased action selection

In this section we will describe an action selection strategy to reuse existing policies. It is an adapted version of a general policy reuse technique proposed by Fernández et al. [8]. There are two important differences. The authors argue that policies can be reused to solve new tasks in a domain. In this thesis we are always dealing with the same task: resolving congestion. Our domain, the road network or highway we attempt to optimize, is not always the same, though. Therefore, we will show that knowledge can be transferred across different traffic networks and scenarios. The second difference is that the strategy presented in [8] depends on a parameter that turns out to be hard to determine. For our specific problem, we will propose a modified version that does not depend on this parameter.

The idea of reusing policies is as follows. A set $L = \{\pi_1, \dots, \pi_n\}$ of policies is given, generated by the algorithm from Chapter 5. If the traffic flow is in a certain state, actions proposed by these policies can be used to explore the state space, rather than exploring the state space randomly. This means that the exploration action selection can be biased by the policies in L. Now the question becomes how the algorithm knows which policy proposes useful actions to guide the exploration. This can be done by defining a probability distribution over the policies in L, depending on the average reward received when using the policies in L. We will formalize this conceptual idea below. First we define a reuse exploration strategy that can be used by the Q-learning algorithm.

Definition 2 (Reuse exploration strategy). *The reuse exploration strategy starts with a given policy π_k and a value $0 \leq \psi \leq 1$. In each step of the Q-learning algorithm, with probability ψ, π_k is used to select an action. In all other cases, a random action is taken with probability ψ, otherwise the policy that is currently learned will be followed. After each step, ψ will be decreased by factor ψ'.*

If there are multiple policies to be reused, it is not immediately clear which one introduces the best bias in the exploration phase of the learning algorithm. Suppose that we have n policies $\pi_1, \dots, \pi_n$, and suppose that W_j denotes the average reward obtained when reusing π_j according to the exploration strategy above. The probability of reusing π_j can be defined as follows:

$$P(\pi_j) = \frac{e^{W_j}}{\sum_{p=1}^{n} e^{W_p}} \tag{7.1}$$

The probability distribution is known as the Boltzmann distribution. The version used in [8] includes a temperature parameter τ, but exploratory tests have shown that it is difficult to find a value τ that works to learn in multiple scenarios. Therefore, we propose to scale the values W_i to values W_i' ($i = 1, \dots, n$) as follows:

$$W_i' = \frac{\theta \cdot W_i}{-1 \cdot \min\{W_i \mid i = 1, \dots, n\}} \tag{7.2}$$

where the parameter θ can be varied. The scaled values W_i' can be used to define the probability distribution in exactly the same way as described in Equation 7.1. The probability distribution is defined in such a way that policies that give a large gain, in terms of the total reward, are more likely to be reused than policies that do not propose useful actions to solve a new scenario. It is important that the probability of choosing the most useful existing policies increases, but not too fast. Otherwise, it would not be possible to recover from changes made to the probability distribution in the first steps. The fraction $(W_i / -1 \cdot \min\{W_i \mid i = 1, \ldots, n\})$ is a value between -1 and 0, so a good choice for the parameter θ is a value in the interval $(0, 2]$, since the exponential function e^x quickly approaches zero for values x lower than -2. Other values may cause the probability distribution in Equation 7.1 to converge too fast, which would prevent some policies from being reused.

7.3 Reusing policies in Q-learning

Algorithm 4 below shows how the probability distribution and the reuse exploration strategy can be combined with Q-learning. This algorithm is similar to the algorithm presented in [8], but we propose a two phase algorithm. In the first R episodes, either existing policies are reused in an episode, or a fully greedy episode is performed using the new policy. In this process, the existing policies bias the action selection during learning to create a policy to start with in the remaining episodes. In the remaining episodes, the standard learning techniques are performed. The motivation behind this two phase approach is that in the first few episodes, learning is accelerated by exploiting existing knowledge, instead of starting from scratch. Once a good starting policy has been learned in the first phase, the regular algorithm can be used to learn a policy for a new scenario or road network.

7.4 Experiments

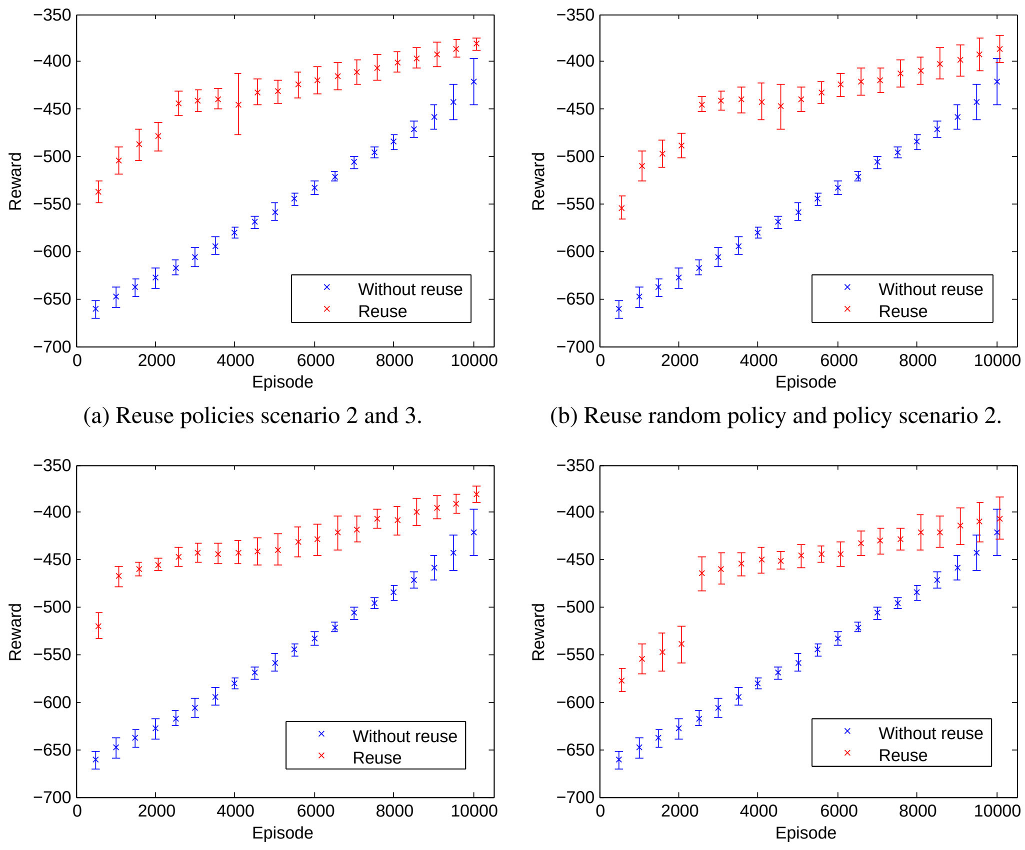
To evaluate whether the policy reuse algorithm is able to propose useful actions, based on existing policies, we conducted an experiment. We generated regular policies for scenario 2, 3 and 4, and we created a random policy by initializing the internal weights of the neural network function approximator randomly. Then we applied the reuse algorithm to generate a policy for scenario 1, by reusing two existing policies in the first 2000 learning episodes. The parameters we used are $R = 2000$, $\theta = 0.1$ and $\psi' = 0.95$. The results are shown in Figure 7.1. It depicts the rewards received by the agent during learning, and the vertical intervals indicate the standard deviation of the received rewards in 20 runs. This is important since there is randomness involved, and multiple runs of the algorithm may produce different results. The figures also include the received rewards for 20 runs of the regular learning algorithm without reuse.

Based on the graphs we can conclude that the biased action selection heuristic suggests useful actions to accelerate the learning process. The reuse rule was applied in the first 2000 episodes, but rewards received during the remaining episodes are also consistently higher.

In Figure 7.2 we visualize how the probability distribution over policies changes during the first 2000 episodes of the learning algorithm. When reusing a random policy, we can see that the probability of reusing the random policy decreases over time, and after a while the new policy has the highest probability of being chosen. When reusing policies for

Algorithm 4: Q-learning with policy reuse

```
input  : initial state $s$, learning rate $\alpha$, discount factor $\gamma$, tile width $w$, number of
         tilings $m$, set $L$ containing $n$ policies, number of reuse episodes $R$, reuse
         strategy parameter $\psi'$
output: approximate $Q$-function, representing a policy

d ←number of dimensions of the state space
initialize $W_i$ and $U_i$ to 0 for $i = 1, \ldots, |L| + 1$
foreach action a do
    create a neural network $n_a$ with $d$ inputs, $d + 1$ hidden nodes and 1 output
for first R episodes do
    rescale the values $W_i$ according to Equation 7.2
    define the probability $P(\pi_i)$ according to Equation 7.1 for $i = 1, \ldots, |L| + 1$
    select a policy $\pi_k$ using the probability distribution
    $\psi \leftarrow 1$
    foreach step of the episode do
        if $k < |L| + 1$ then
            use reuse strategy with policy $\pi_k$ and probability $\psi$ to select action $a$
            $\psi \leftarrow \psi' \cdot \psi$
        else
            choose action $a$ according the currently learned policy $\pi_k$
        execute $a$, observe reward $r$ and new state $s'$
        $e \leftarrow n_a(s) + \alpha(r + \gamma \max_{a' \in A} n_{a'}(s') - n_a(s))$
        propagate error $e$ backwards through $n_a$
        $s \leftarrow s'$
    let $rw$ be the reward received in the last episode
    $W_k = \frac{W_k U_k + rw}{U_k + 1}$
    $U_k = U_k + 1$
for all other episodes do
    foreach step of the episode do
        calculate $n_a(s)$ for each action $a \in A$
        choose action $a$ based on $Q$-values, using an exploration strategy
        execute $a$, observe reward $r$ and new state $s'$
        $e \leftarrow n_a(s) + \alpha(r + \gamma \max_{a' \in A} n_{a'}(s') - n_a(s))$
        propagate error $e$ backwards through $n_a$
        $s \leftarrow s'$
```



(a) Reuse policies scenario 2 and 3.

(b) Reuse random policy and policy scenario 2.

(c) Reuse policies scenario 3 and 4.

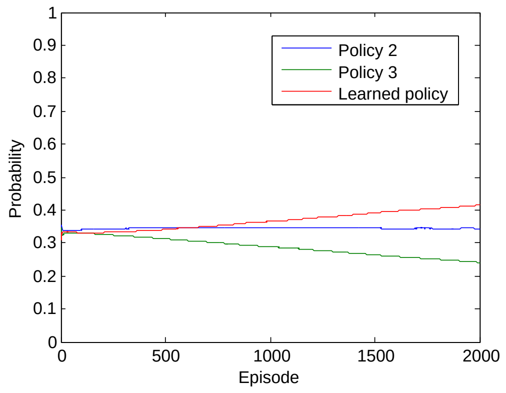
(d) Reuse random policy and policy scenario 3.

Figure 7.1: Received rewards when learning a policy for scenario 1 by reusing existing policies that have been learned for other scenarios.

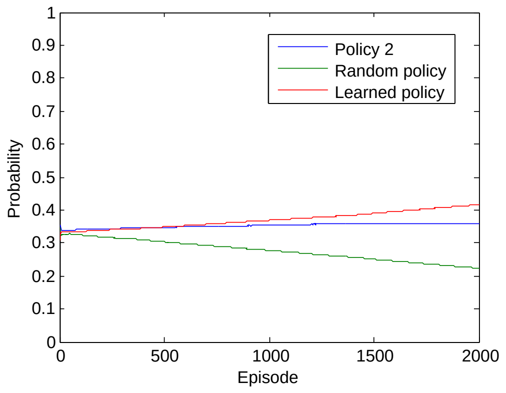
scenarios 2 and 3, the probability of selecting the policy corresponding to scenario 3 is also decreasing, which could be an indication that the actions proposed by this policy are less useful than the actions derived from the policy for scenario 2. The probabilities change relatively slowly, because otherwise a large decrease in the first few episodes may cause that one of the policies cannot be selected anymore, and then its probability can never be increased again. The parameter θ can be changed to obtain the desired behavior.

7.5 Discussion

The policy reuse scheme has shown that it can be used as a better action selection heuristic than random action selection in ε-greedy. Given a set of existing policies, it identifies the policies that turn out to be useful, and the probability distribution is changed in such a way that useful policies have a higher probability to be chosen. For our speed limit policy learning techniques this may not be useful in the first place, since the solutions we find with



(a) Reuse policies scenario 2 and 3.



(b) Reuse random policy and policy scenario 2.

Figure 7.2: Probability distributions during first 2000 episodes.

Q-learning are already close to optimal in terms of vehicle hours. However, in the context of multi-agent policy learning there may be some opportunities to generalize the reuse idea. If separate policies are learned for individual highway sections, an agent may reuse knowledge learned by agents associated with neighboring sections to improve the learning efficiency. The basic idea could be that neighboring agents share knowledge by maintaining a probability distribution over their policies. The results presented in this chapter can be considered as a first step in a future work direction.

7.6 Summary

In this chapter we presented a strategy to reuse existing policies as a bias in the exploration process of Q-learning. We presented a probability distribution over existing policies, which is used to select a policy, instead of selecting exploration actions completely random. Originally, this idea was proposed in [8], but we made some small modifications and we have shown that their reuse strategy also works well when combining it with our speed limit policy algorithm.

Chapter 8

Case study: Simulation A67

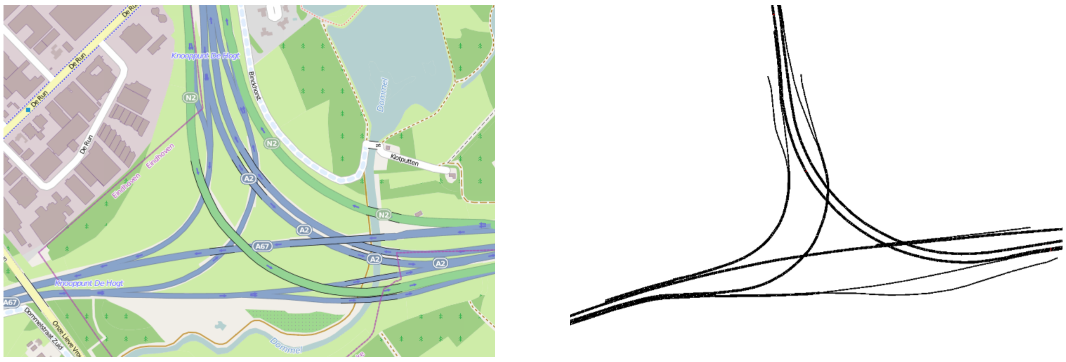
In this chapter we present the results of a case study we performed to evaluate the proposed algorithms in real life scenarios, related to the third research question. We set up a simulation of the A67 highway in SUMO, and applied our generated policies to communicate speed limits to participants of Smoover. We also implemented the Automatic Incident Detection (AID) system that controls the speed limits shown on matrix signs in the Netherlands, and we compare this system to our new control approach. In the first part of this chapter we discuss the translation of flow data and the A67 map to SUMO, and how we can obtain realistic traffic flows. This part of the work has been done in collaboration with Pieter Loof [17]. In the remaining sections we describe AID and the Smoover control approach, and we present the results of the experiments we performed.

8.1 A67 simulation in SUMO

We use SUMO as a microscopic simulation environment. More details about this simulation software package can be found in Section 2.3 of this thesis. SUMO contains a feature that makes it possible to import a road network from OpenStreetMap (OSM)[1]. This enables us to make a simulation using a realistic A67 road network. An example is given in Figure 8.1 below. It shows an overview of Knooppunt De Hogt, derived from OpenStreetMap, and the corresponding road network after importing the map in SUMO. Note that not all the roads of Figure 8.1a are present in Figure 8.1b, since we only import a highway map.

To obtain realistic vehicle flows, we use data from the Nationale Databank Wegverkeersgegevens (NDW), a Dutch national database containing traffic data. This database contains data collected from detectors and induction loops that have been installed in the pavement. Based on this data simulations can be performed. For instance, after retrieving the data that was collected during a typical morning rush hour in the past, we can run a simulation of a real traffic scenario that occurred in the past. Furthermore, a probability distribution over different types of cars can be defined to make the simulation more realistic. If a vehicle enters the road network in the simulation, its type can be selected according to this probability distribution. More information regarding the simulation setup can be found in [17].

[1]For more information, consult `http://www.openstreetmap.org`.



(a) OpenStreetMap. (b) SUMO.

Figure 8.1: Knooppunt De Hogt in OSM and SUMO.

8.2 Automatic Incident Detection

Matrix signs have been installed in the Netherlands to communicate speed limits to car drivers. The most important reason to use these matrix signs is safety, because it warns car drivers to reduce their speed gradually if downstream congestion is detected. Traffic flow regulation and reducing congestion is just one of the side effects of such a system, but it is still interesting to investigate how traffic flow behaves in a simulation when using matrix signs in a similar way as in practice.

The algorithm that determines the speed limits shown on matrix signs is called Automatic Incident Detection (AID) [3]. It assumes that a detector is associated with each matrix sign above the highway, and that there is a detector that measures speed for each lane. If the speed drops below a threshold value (typically, 35 km/hr), the AID system is activated. The system is deactivated if congestion has resolved and the speed exceeds another threshold value (typically, 50 km/hr).

A detector maintains a so-called smoothed driving time value P_j for each lane j, which is updated according to the following formula upon detection of a vehicle:

$$P_j = \begin{cases} \alpha_{acc} \cdot P_{measured} + (1 - \alpha_{acc}) \cdot P_j & P_{measured} \leq P_j \\ \alpha_{dec} \cdot P_{measured} + (1 - \alpha_{dec}) \cdot P_j & \text{otherwise} \end{cases}$$

where $\alpha_{acc} = 0.15$, $\alpha_{dec} = 0.40$ and $P_{measured}$ is the measured driving time at lane j. Speed can be translated to driving time by calculating the amount of time that is required to cover a distance of 2.5 meters, and vice versa. The reason to use driving time instead of speed in this formula is that it is more sensitive for speed changes, because driving time is a relatively large number compared to speed if the measured speed is low [3].

If the minimum speed measured at detector i drops below 35 km/hr, the AID system is activated by assigning the speed limits 50 and 70 to the matrix sign at detector $i-1$ and $i-2$, respectively. An example is shown in Figure 8.2, where congestion is measured by detector i, and speed limits have been assigned to upstream matrix signs. Note that two consecutive matrix signs may display the same speed limit if multiple consecutive detectors

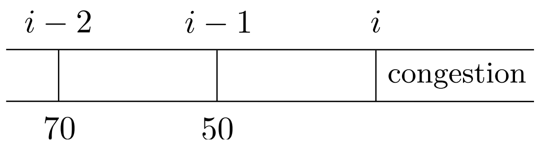


Figure 8.2: Assignment of speed limits to matrix signs.

measure congestion. Speed limits are deactivated if the minimum measured speed at a detector exceeds 50 km/hr. AID can be implemented in SUMO by measuring the speeds at the locations of real detectors, and assigning the resulting speed limits to specific parts of the road network.

8.3 Smoover control

As mentioned in the introductory chapter of this thesis, Smoover aims to regulate traffic flow by using in-car technologies. For example, speed limits can be shown on a display integrated in the dashboard of a car, and such a control approach is not dependent on the infrastructure itself. Where the regulation capabilities of AID depend on the locations of matrix signs, Smoover is able to assign speed limits depending on the current positions of cars, measured with GPS. This means that a virtual partitioning of the road network in sections can be created, such that the speed limit policies from Chapter 6 can be applied to determine the speed limits that should be communicated to cars participating in Smoover.

The Smoover concept can be simulated in SUMO by virtually partitioning the road network into sections, and applying the multi-agent control method to each virtual section. The resulting speed limits can be assigned to vehicles participating in Smoover by changing their individual maximum speed. Note that the number of participants can be varied, such that only a limited amount of vehicles complies to the speed limits that have been issued.

8.4 Experiments

We performed several experiments in the microscopic simulation environment SUMO, with the aim to investigate how effective our methods can be, and to make a comparison with AID. In these experiments, we varied the number of participants of Smoover, and the compliance rate of the vehicles in case of AID.

We focus on the last eight kilometers of the A2 near Eindhoven, in the direction of Eindhoven, before it merges with the A67 at the road interchange De Hogt. We derived real traffic patterns from the NDW database to simulate a realistic morning traffic scenario that occurred on October 21 in 2013 from 7am to 8am. This defines the traffic flow on the A2 within a time window of several hours. The flow has been increased by a factor of 1.7 to generate more vehicles, but the flow pattern remained the same. The vehicles in the simulation have different types (e.g., truck, car, fast car) with different dimensions, acceleration parameters and maximum speed values. We used the NDW data to find the locations of detectors and matrix signs on the A2, and defined them in SUMO accordingly.

This enabled us to run a traffic simulation in which we determine speed limits using the same algorithm as in practice. Furthermore, it allows us to run a simulation of the Smoover concept, where a limited number of car drivers receives a speed limit that is shown on the dashboard.

The purpose of the experiments is not to argue that the same congestion reductions would be possible in practice, since we did not calibrate the macroscopic model and the microscopic simulation may not be accurate enough. However, the experiments should give evidence that policies can be applied in real systems and that they propose useful actions if congestion arises.

Smoover participants

For the development of Smoover, it is relevant to know how many cars should comply with the speed limits to reduce congestion. It is likely that congestion can be reduced significantly when the number of participants is high, but initially this will be very low, so through simulation we need to investigate how many participants are required. In the simulation, speed limits are determined using a policy that has been generated with the algorithm from Chapter 6, and we consider the last eight kilometers of the A2 as two sections of about equal length. We simulate one hour traffic flow, and after about eight minutes we mimic an accident at the road interchange De Hogt, which causes the traffic to slow down for fifteen minutes. Performance is measured using the vehicle hours metric, similar to the macroscopic simulations in the previous chapters. The experiment is repeated for several participation rates, where every vehicle that enters the highway becomes Smoover participant with a predefined probability. For each participation rate, we repeated the simulation 20 times. The results are shown in Figure 8.3.

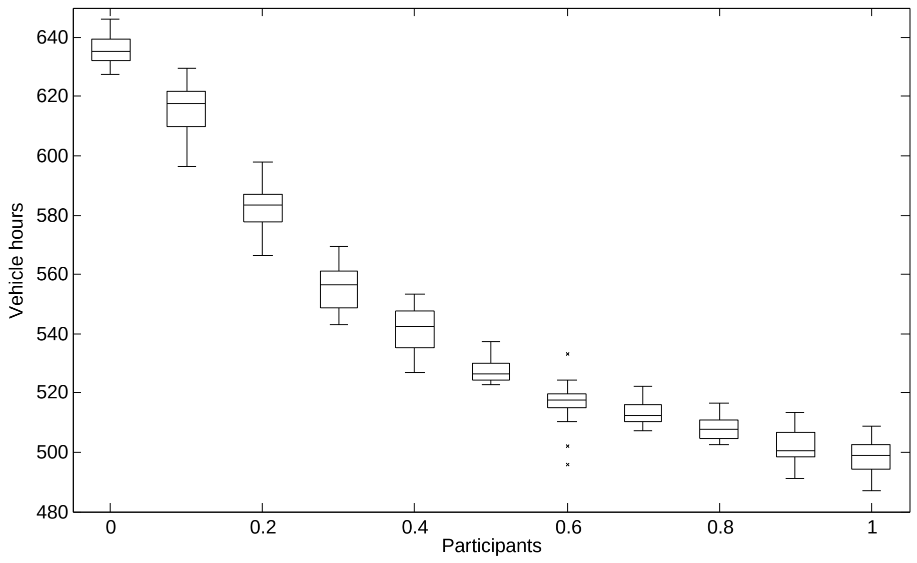


Figure 8.3: Smoover: vehicle hours for several participant percentages.

From the graph we conclude that small participation rates may have significant influence on traffic flow. The case in which there are no Smoover participants at all can be considered as a baseline, since there are no cars that comply to the speed limits that have been assigned. For low participation rates, the gain is relatively large if the number of participants is further increased, but for a higher number of participants this improvement is much smaller. This seems plausible, because if the number of participants is high, cars that do not participate in Smoover also need to comply to the speed limits, as a consequence of the chosen speed by nearby cars.

AID

We applied the AID algorithm to run a simulation with matrix sign speed limitations. In Section 8.2 we explained that the speed limits 50 and 70 are assigned to section $i-1$ and $i-2$, respectively. In our experiment we assigned speed limitations to p preceding matrix signs, and only the first one displayed 50. The reason is that we want to investigate the influence of the length of the speed limited area on vehicle hours. We assumed a compliance rate of 90 percent. Figure 8.4 shows the results for different values of p, from which we can conclude that a larger speed limited area reduces vehicle hours more. We expect that it should be possible to find a value of p that minimizes the number of vehicle hours for this scenario, because p cannot be increased infinitely to let the number of vehicle hours decrease further. Due to the limited size of the road network we used, we are unable to go further than 12 preceding matrix signs. Compared to the results from the Smoover experiment, we observe that AID performs better for $p = 11$ and $p = 12$. In such cases, almost all parts of the highway have a speed limit assigned if speed limits are activated. Smoover, however, used a partitioning into two sections, so it may initially assign speed limits to a smaller part of the highway. Although Smoover performed slightly worse in

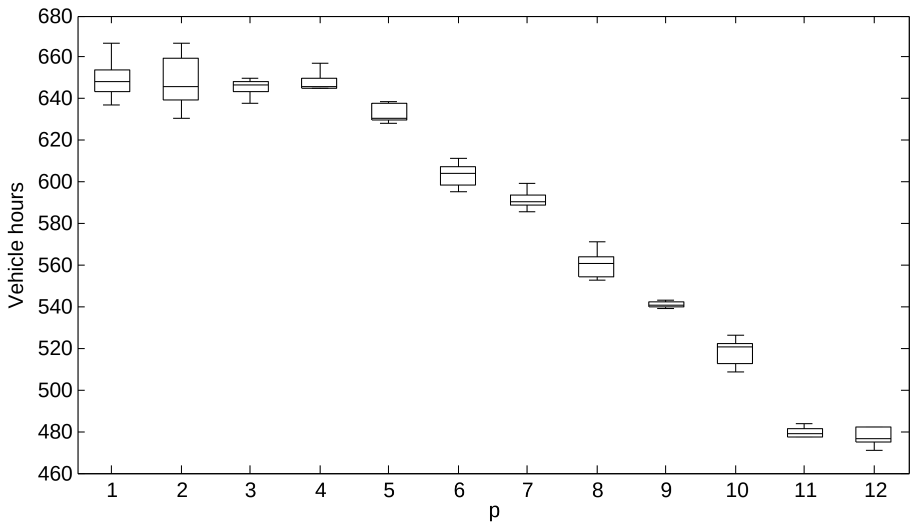


Figure 8.4: AID: vehicle hours.

our experiment, it is still a significant improvement compared to the baseline, and another partitioning of the highway into sections may reduce the number of vehicle hours more. From this point of view, it is relevant to study how the highway can be partitioned into sections, and which partitionings perform well. Another important issue is how far an agent should look ahead in its state description. Now we only considered downstream section $i+1$ for agent i, but it may be useful to include other downstream sections as well. This is not studied in this thesis.

8.5 Discussion

Although the results in this chapter look promising, there are a few remarks that have to be made. Since we did not calibrate the macroscopic model using real data, we were unable to learn policies using parameters that represent the characteristics of the highway that is modeled in SUMO. Furthermore, we do not know how accurate the underlying models of SUMO are and how it corresponds to the characteristics of the A2 and A67 that we considered. This means that we cannot claim that it would achieve the same congestion reduction in practice as well. The results presented in this chapter can be considered as a proof of concept, showing that learned policies propose useful speed limits if congestion occurs, and that congestion can be reduced by assigning speed limits. When deploying the proposed techniques in a real system, the macroscopic model should be calibrated to ensure that policies are learned specifically for the highway that is considered.

Another shortcoming of our analysis is that we do not know how multiple congested areas influence each other, and how the applied policies behave under these circumstances. In particular it is interesting to know to which extent the Smoover idea generalizes to larger road networks. Due to the limited scenario data and time restrictions, we have not been able to investigate this further.

The AID experiment shows that determining speed limitations using AID may give a similar improvement as the Smoover concept, but it relies on the locations of matrix signs. Since not all the highways have matrix signs installed, this method cannot always be applied in practice. Smoover relies on floating car data and detector measurements, and does not need matrix signs to communicate speed limits to vehicles. From a practical point of view, it is unlikely that such systems operate completely autonomously without a human traffic operator. This aspect was ignored in the simulation experiments.

To make a better comparison with existing techniques, we may want to implement more algorithms. It is particularly interesting to compare our work with techniques like SPECIALIST [13], which is an algorithm to suppress shock waves by assigning speed limits, but this is beyond the scope of this thesis.

8.6 Summary

In this chapter we presented a case study involving microscopic simulations of the A2 and A67 in the Netherlands. We used a real traffic demand pattern to set up a more realistic simulation of this highway, and we applied one of the policies that was learned using the

methods from the previous chapters. We ran simulations of the proposed Smoover concept, in which only the participating vehicles comply to the speed limits that have been issued. We also implemented Automatic Incident Detection (AID), which is an existing algorithm to activate speed limits shown on matrix signs if congestion occurs. Using simulations we have shown that the Smoover system is able to reduce congestion, even if there is only a small number of participants. It may achieve a similar congestion reduction as AID, but one of the benefits is that it does not require matrix signs or other enhancements of the infrastructure. The results presented in this chapter can be considered as a proof of concept, to demonstrate how policies can be applied in practical scenarios.

Chapter 9

Conclusions and Future Work

The objective of this thesis is the development of new algorithmic techniques to optimize traffic flow, closing the gap between traffic flow theoretical concepts and artificial intelligence. In this final chapter we list our most important contributions, related to the research questions we identified in Chapter 1. Moreover, we discuss potential directions to extend this research in future work.

9.1 Main contributions

In this section we list the main contributions of this thesis. We discuss how this contributes to the traffic flow optimization field and how this may be used in practical applications.

Reinforcement learning algorithm to optimize traffic flow

Previous work in the field of traffic flow optimization mostly relied on computationally expensive calculations or rule-based control techniques, but we approached this problem from a rather different perspective. We have shown that the problem of finding speed limits can be solved using reinforcement learning. Existing traffic flow models can be combined with reinforcement learning techniques to devise a new traffic flow optimization algorithm. As far as we know, it has not been shown before that reinforcement learning and planning techniques can be applied to determine speed limits combined with METANET.

Matrix signs have been used for years to communicate speed limits to car drivers, but given the huge potential of in-car technologies, it is expected that a transition will be made to more decentralized control. Moreover, the emergence of big data analysis techniques may help to use new data sources that potentially contain useful information that has not been used before. Our work can play an important role in this context. We have shown that predictions of traffic flow can be combined with our techniques to regulate flow more efficiently. Given the scalability of the reinforcement learning algorithms, it is also possible to include other information about traffic conditions or road characteristics in the future.

Our algorithm is using the METANET traffic flow model and performs Q-learning to find single-agent as well as multi-agent policies that optimize traffic flow. The traffic flow model we selected can be used as a representation of the environment, and the actions selected by the agent can be instantly taken into account in the traffic simulation. The same model can be used to evaluate the performance of the learned policies. This relates to research question 1 and 2 we posed in the first chapter. We have also shown that solutions to an MDP, representing the speed limits to be assigned, can be reused to improve learning efficiency. In our work we only considered a few similar demand profiles, so there may be more difficult scenarios that our algorithm cannot handle. We did not study scenario difficulty, so it may be required to enhance our approach to be able to solve more scenarios and learn better policies.

Traffic predictions

Our second contribution relates to traffic flow prediction. We have shown that traffic predictions can be used to regulate traffic flow more efficiently, and that the resulting policies are more robust in case of inaccurate measurements of speed, density and flow. In our experiments, predictions are just an extrapolation of the current traffic conditions, but in practice this can be replaced by predictions that follow from traffic data analysis techniques. However, in this case it is important to use a calibrated traffic flow model, so predictions cannot be directly integrated in practice. Additionally, other prediction horizons may be taken into account. This part of our work relates to research question 1 from Chapter 1. We only studied traffic predictions in the first method we presented, using a 5 minute control step size, so there is still room for improvement and further research.

Traffic simulation environment

We implemented a microscopic simulation environment on top of SUMO, to evaluate the Smoover concept in more realistic scenarios. Additionally, we implemented Automatic Incident Detection (AID) that is used by Rijkswaterstaat to make a comparison with Smoover. The simulation environment is implemented in such a way that different regulation strategies can be tested, and multiple traffic scenarios derived from NDW can be executed. We have shown how generated policies can be used in the Smoover concept, and we have shown that generated policies may be able to reduce congestion in practice. Depending on the partitioning of the highway into sections, it can be a competitive alternative to AID. This part of our work answers research question 3, but we have not been able to prove how it will work in practice. The reason is that we do not know how realistic our simulation environment is, and we do not know its correspondence to practical scenarios. We did not study in which scenarios our method can be useful, since we tested our approach on a small number of scenarios of limited length. Our results are still valuable and can be considered as a proof of concept that can be extended in the future.

9.2 Future work

An extension of our work that we can propose as future work is the integration of stochastic traffic predictions. We made the assumption that traffic predictions are a single value, but such a prediction may involve a probability distribution over possible future traffic conditions. In particular, it is interesting to figure out how traffic predictions with a bimodal or multimodal probability distribution can be exploited to find speed limit policies. Such a probability distribution may occur if traffic either slows down completely or is not affected by congestion at all.

Another extension of our work can be found in the area of multi-objective optimization. In our work we used the total travel time as the objective function to optimize, but other objective functions can be taken into account as well, such as total throughput of the road network. Objectives related to vehicle characteristics (e.g., emission) cannot be expressed using the macroscopic model. Another aspect that can be studied further is how multiple objectives can be balanced in this context.

The multi-agent coordination approach for variable speed limits can be further improved. Currently we define an MDP for every road section, and we let them independently train the same policy. However, action selection behavior of adjacent sections may influence the rewards received by the agents as well. A more sophisticated learning technique can be tested, taking into account the dependencies between the actions selected by the agents. Another thing that may be tested is a different formulation of the multi-agent approach, in which the state space is the concatenation of the individual states, and the global action space is defined as the Cartesian product of the individual action sets. As mentioned previously, it can also be studied which information about neighboring sections should be included in states, and which scenarios can be considered difficult to solve. The policy learning algorithm may be improved after studying this.

The policy reuse idea from [8] has been tested, but perhaps it is possible to generalize this idea to the multi-agent case, where agents can reuse knowledge learned by other agents. In case of multi-agent coordination of speed limits, a section may reuse knowledge from neighbors because it is likely that their resulting policies are similar. This requires an algorithm that learns different policies for different sections.

Bibliography

[1] I. Arel, C. Liu, T. Urbanik, and A. Kohls. “Reinforcement learning-based multi-agent system for network traffic signal control”. In: *Intelligent Transport Systems, IET* 4.2 (2010), pp. 128–135.

[2] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz. “SUMO - Simulation of Urban MObility - An Overview”. In: *SIMUL 2011, The Third International Conference on Advances in System Simulation*. 2011, pp. 55–60.

[3] N. van den Bosch. “Blik op het Wegennet”. MSc thesis. Delft University of Technology, 2007.

[4] E. Bryson and Y. C. Ho. “Applied Optimal Control: Optimization, Estimation, and Control”. In: Blaisdell Publishing Company, 1969.

[5] L. Chu and X. Yang. “Optimization of the ALINEA Ramp-metering Control Using Genetic Algorithm with Micro-simulation”. In: *Transportation Research Board 82nd Annual Meeting*. Washington, DC, 2003.

[6] R. Coulom. “Reinforcement Learning Using Neural Networks, with Applications to Motor Control”. PhD thesis. Institut National Polytechnique de Grenoble, 2002.

[7] C. F. Daganzo. “The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory”. In: *Transportation Research Part B: Methodological* 28.4 (1994), pp. 269–287.

[8] F. Fernández and M. M. Veloso. “Probabilistic policy reuse in a reinforcement learning agent”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*. 2006, pp. 720–727.

[9] L. Figueiredo, I. Jesus, J. Machado, J. Ferreira, and J. Martins de Carvalho. “Towards the Development of Intelligent Transportation Systems”. In: *Intelligent Transportation Systems*. Vol. 88. 2001, pp. 1206–1211.

[10] S. Haykin. *Neural Networks: A Comprehensive Foundation*. 2nd. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

[11] A. Hegyi. “Model Predictive Control for Integrating Traffic Control Measures”. PhD thesis. Delft University of Technology, 2004.

[12] A. Hegyi, B. De Schutter, and J. Hellendoorn. “MPC-based optimal coordination of variable speed limits to suppress shock waves in freeway traffic”. In: *Proceedings of the 2003 American Control Conference*. Denver, Colorado, 2003, pp. 4083–4088.

[13] A. Hegyi, S. Hoogendoorn, M. Schreuder, H. Stoelhorst, and F. Viti. “SPECIALIST: A dynamic speed limit control algorithm based on shock wave theory”. In: *11th International IEEE Conference on Intelligent Transportation Systems*. 2008, pp. 827–832.

[14] B. Huang, G. Cao, and M. Guo. “Reinforcement Learning Neural Network to the Problem of Autonomous Mobile Robot Obstacle Avoidance”. In: *Proceedings of the International Conference on Machine Learning and Cybernetics*. Vol. 1. 2005, pp. 85–89.

[15] U. Karaaslan, P. Varaiya, and J. Walrand. *Two Proposals To Improve Freeway Traffic Flow*. Tech. rep. Institute of Transportation Studies, UC Berkeley, 1990.

[16] A. Kotsialos, M. Papageorgiou, C. Diakaki, Y. Pavlis, and F. Middelham. “Traffic Flow Modeling of Large-Scale Motorway Networks Using the Macroscopic Modeling Tool METANET”. In: *IEEE Transactions on Intelligent Transportation Systems* 3.4 (2002), pp. 282–292.

[17] P. Loof. MSc thesis. Delft University of Technology, 2014, to appear.

[18] A. Messner and M. Papageorgiou. “METANET: A macroscopic simulation program for motorway networks”. In: *Traffic Engineering & Control* 31.8-9 (1990), pp. 466–470.

[19] T. M. Mitchell. *Machine Learning*. 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

[20] M. Negnevitsky. *Artificial Intelligence: A Guide to Intelligent Systems*. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[21] M. Papageorgiou, H. Hadj-Salem, and J.-M. Blosseville. “ALINEA: a local feedback control law for on-ramp metering”. In: *Transportation Research Record* 1320 (1991), pp. 58–64.

[22] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st. New York, NY, USA: John Wiley & Sons, Inc., 1994.

[23] Y. Saitoh, M. Kondoh, and Y. Komatsu. *Driverless car travelling guide system*. US Patent 4,855,656. 1989.

[24] B. de Schutter, S. Hoogendoorn, H. Schuurman, and S. Stramigioli. “A Multi-Agent Case-Based Traffic Control Scenario Evaluation System”. In: *Proceedings of the IEEE 6th International Conference on Intelligent Transportation Systems (ITSC’03)*. Shanghai, China, 2003, pp. 678–683.

[25] P. Simon. *Too Big to Ignore: The Business Case for Big Data*. Wiley & SAS Business Series. New Delhi: Wiley, 2013.

[26] S. P. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvri. “Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms”. In: *Machine Learning* 38.3 (2000), pp. 287–308.

[27] R. S. Sutton. “Learning to Predict by the Methods of Temporal Differences”. In: *Machine Learning* 3.1 (1988), pp. 9–44.

[28] C. J. C. H. Watkins. “Learning from Delayed Rewards”. PhD thesis. Cambridge, UK: King’s College, 1989.

[29] C. J. C. H. Watkins and P. Dayan. “Q-Learning”. In: *Machine Learning* 8.3-4 (1992), pp. 279–292.

[30] S. D. Whitehead and L. J. Lin. “Reinforcement learning of non-Markov decision processes”. In: *Artificial Intelligence* 73.1-2 (1995), pp. 271–306.

[31] M. Wiering and M. van Otterlo. *Reinforcement Learning: State-of-the-Art*. Adaptation, Learning, and Optimization. Springer, 2012.

[32] J. Zhang, H. Chang, and P. Ioannou. “A Simple Roadway Control System for Freeway Traffic”. In: *American Control Conference*. 2006, pp. 4900–4905.