MASTER

On deep reinforcement learning for data-driven traffic control

Duwaer, D.A.

Award date:
2016

Link to publication
On Deep Reinforcement Learning for Data-Driven Traffic Control

Master Thesis

Daniël A. Duwaer

Eindhoven, July 2016

Supervisors
TU/e: prof. dr. M. Pechenizkiy
LD Software: drs. U. Claassen RA RE CIA

Public version
Parts of this research are classified as confidential and excluded from this publication. The full research may be disclosed upon request.

LD software
This research was conducted at LD Software in Waalre, the Netherlands.

www.ldsoft.nl/ai
Abstract

To cope with the ever increasing traffic congestion in road networks, there is a need for methods to handle more cars with less time and space. Traffic lights and their corresponding scheduling algorithms play a key role in effectively managing traffic flows. Scheduling algorithms currently employed throughout world wide road networks are static, while research points out that with the usage of smart road sensors and dynamic machine learning techniques based on reinforcement learning, the average waiting times and throughput of a road network can significantly be improved upon. In this research, we advance the current state-of-the-art further in both intelligent transport and reinforcement learning by proposing and analyzing a technique called deep reinforcement learning in data-driven traffic light scheduling controllers. We aim to give insight into the traffic light scheduling problem itself, how machine learning techniques can be applied within this problem domain, and what the current state-of-the-art is in data-driven traffic light scheduling. Using these insights, we present several traffic light scheduling algorithms based on deep reinforcement learning and a generalized traffic simulation framework, which allows us to realistically model complex traffic flows through user-defined road networks, such that we can benchmark different types of traffic light scheduling algorithms. Ultimately, using the generalized traffic simulation framework, we show that the scheduling algorithms based on deep reinforcement learning outperform the current state-of-the-art algorithms significantly, cutting vehicle waiting times in highly congestive situations by $58 - 86\%$ ($76\%$ on average) when compared to traditional timing-based algorithms widely deployed across the real-world, and by $29 - 77\%$ ($59\%$ on average) when compared to researched traditional reinforcement learning algorithms.
# Contents

1 Introduction 5
  1.1 Motivation ................................. 5
  1.2 Research goals ............................. 7
  1.3 Problem formulation ....................... 7
  1.4 Research methodology ...................... 8
  1.5 Main results & contributions ............... 9
  1.6 Outline .................................. 10

2 Background 11
  2.1 Approaches to traffic light scheduling .......... 11
    2.1.1 Traditional approaches .................. 11
    2.1.2 Data-driven approaches ................... 12
  2.2 Incorporating machine learning in data-driven traffic light scheduling .... 13
    2.2.1 Offline versus online learning ............. 13
    2.2.2 Supervised, unsupervised, and reinforcement learning ................. 14
    2.2.3 Q-learning ................................ 16
  2.3 State-of-the-art in data-driven traffic light scheduling .......... 18
    2.3.1 Unsupervised learning .................... 18
    2.3.2 Reinforcement learning .................... 19
  2.4 Traffic light scheduling problem formalization .............. 20

3 Traffic light scheduling algorithms 23
  3.1 Research goals ................................ 23
  3.2 Traditional algorithms ........................ 24
  3.3 Data-driven algorithms ........................ 25
    3.3.1 Q-learning with reduced state- and action-space ........... 25
    3.3.2 Deep Q-learning .......................... 28
    3.3.3 Extended Deep Q-learning .................... 34
    3.3.4 Pre-emptive Extended Deep Q-learning .............. 35

4 Traffic simulation 38
  4.1 Requirements ................................ 38
  4.2 Architecture ................................ 39
  4.3 Traffic network ................................ 41
  4.4 Traffic flows ................................ 41
Chapter 1

Introduction

1.1 Motivation

Public transport and the building of new roads are primary solutions to address the problem of traffic congestion, but neither can keep up with the dramatic rise in the number of drivers on this planet. As a result, we have less space to fit more cars in. A key interest is thus to find novel methods of increasing the throughput of a road network without changing the structure of that network.

Let us now consider traffic lights and their corresponding scheduling algorithms, which play a key role in the performance of road networks in terms of delays, throughput, and many other quality attributes. Traditional approaches to traffic light scheduling are primarily timing-based. Optimization of these traffic light schedules to reduce overall waiting times is mostly carried out by either human-based or model-based agents. Both approaches raise the question of accuracy, since it is common knowledge that both humans make mistakes and models can become outdated quickly. It is also common knowledge that building models and relying on human interaction for optimization can be tedious, expensive, and once the road network becomes more complex it also becomes infeasible.

In light of the increasing popularity of the Internet of Things, cheaper computational power, and a relatively new world of computational reasoning, we may now consider other approaches to deal with real-world problems such as the traffic light scheduling problem. In the quest of finding optimality in traffic light scheduling, we can now focus on using smart road sensors, i.e. sensors that measure vehicle queue lengths at road intersections, in combination with machine learning techniques instead of the traditional timing-based approaches. The adoption of machine learning techniques in traffic management is not new, and has been studied occasionally since the early 1990s. For example, Artificial Neural Networks (ANNs) have been researched to predict traffic congestion by Gilmore et al. in 1993 [1] and Smith et al. in 1994 [2]. Even the adoption of reinforcement learning in traffic light scheduling has been studied occasionally since the year 2000, i.e. by utilizing traditional Q-learning, a form of reinforcement learning, by Abdulhai et al. in 2003 [3] and by a follow-up PhD thesis in 2012 [4]. However, as will be explained later on in Section 2, these approaches suffer from state and action space explosions. The more relevant information about the traffic state we take into account, the worse an algorithm based on traditional Q-learning is able to make its decisions,
since its state space will become too big to explore in practice. Note that scheduling decisions are made in real-time, and thus, the algorithm is bound to wait for each scheduling decision to finish until it can observe a new traffic state and “learn” from its decision. Waiting for thousands or millions of scheduling decisions to finish before it can learn from them, which is the case when our state space becomes too big, is simply not acceptable in practice. However, often it is the collective information about the traffic state that allows us to learn which scheduling decisions are good and which are bad.

In this research, we aim to summarize and compress the information contained in the otherwise huge state space, and only remember and learn from the details that are relevant based on pattern recognition. We achieve this by combining reinforcement learning with ANNs, a technique called deep reinforcement learning, in order to improve upon the current state-of-the-art in data-driven traffic light scheduling.

A form of deep reinforcement learning has been studied for the first time in 2013 by Mnih et al. to allow a computerized agent to successfully play a collection of Atari games by utilizing the Q-learning algorithm combined with a deep convolutional neural network as Q-function [5] (detailed insight into Q-learning and ANNs is given in Sections 2 and 3). A deep convolutional neural network is a more complex type of neural network, primarily specialized in and used for pattern recognition in images, as it is inspired by the visual cortex in animals. Mnih et al. trained the computerized agent to play Atari games by continuously feeding the agent with (simplified) still images of each frame of the gameplay. The computerized agent was able to recognize patterns between the sequence of images and good or bad behaviour (i.e. high and low game scores), and thus, able to learn in what way to behave to obtain a high score. However, as images consist of many pixels, the process of connecting the dots between a sequence of images and good or bad behaviour is a long one: it can take an agent days or weeks of continuous playing time for the agent to stabilize and achieve higher scores. The more complex a game is, the longer it will take before the agent stabilizes and converges. One could even argue that using images as input to a deep reinforcement learning agent is an ineffective way of transferring the effective information that we want the agent to reason about, as that information is embedded in a frame with a lot of noise. The reason why it worked well in the case of Atari games was because the images mainly consisted of simple colored shapes highlighted in contrast to a black background, in which each shape was relevant to the agent in order to perform well. However, it still remains an open research question whether the agent is able to perform equally well on modern video games, with much more complex objects embedded in images with a noise ratio that is much higher.

While the usage of deep convolutional neural networks can be useful for pattern recognition in a large corpus of images, it is not designed for pattern recognition in anything other than images. Thus, the technique proposed by Mnih et al. is not portable to other problem domains, in which reinforcement learning is not based on “visuals”. Up until this research, no deep reinforcement learning technique has been published that utilizes a different type of ANN, allowing the application of deep reinforcement learning in a vast range of other problem domains, such as traffic light scheduling.
CHAPTER 1. INTRODUCTION

1.2 Research goals

Our main research goal is to explore novel approaches in data-driven traffic light scheduling to advance the current state-of-the-art even further. Our focus lies on the investigation into the feasibility of utilizing machine learning techniques to address the traffic light scheduling problem. In particular, we are interested in the performance of scheduling algorithms governed by deep reinforcement learning, in comparison to the current state-of-the-art reinforcement learning techniques. We compare the novel approaches with a generalized traffic simulation framework, which serves as a testbed to benchmark the different traffic light scheduling algorithms.

An additional goal of this research is to deliver the traffic light scheduling algorithms in a framework that can be utilized in such a way that it can be applied in a unified manner to different kinds of environments. In this research, the environment will be provided by a traffic simulator. In future work, the environment can be a real-world traffic intersection with smart road sensors in place. For this reason, computational performance of our algorithms also plays a key role in our research, since we want to be able to utilize them in real-world environments.

The final goal of this research is to demonstrate that deep reinforcement learning can effectively be applied in non-visual problem domains, opening up a vast uncharted research landscape of deep reinforcement learning-related studies in a countless number of other problem domains.

1.3 Problem formulation

The main research question is formulated as follows: do data-driven traffic light scheduling algorithms based on deep reinforcement learning provide a significant advantage over the current state-of-the-art reinforcement learning techniques?

Questions that arise during the investigation of our main research question are the following:

1. In what way can machine learning be applied to the traffic light scheduling problem? We could consider this in terms of supervised versus unsupervised learning and online versus offline learning. All of these terms and their corresponding background will be explained later on in Section 2 of this research.

2. How can we benchmark different traffic light scheduling algorithms in a generalized framework? We need to be able to combine and benchmark different traffic light scheduling algorithms to simulate varying situations where we, for instance, have a realistic environment with both traditional timing/model-based controllers and data-driven controllers.

3. How can we learn from recurrent contexts (i.e. seasonality in traffic behaviour) in our traffic light scheduling algorithms? Examples of recurrent contexts are daily rush hours and yearly winter/summer traffic.

4. How robust are the algorithms described in this research against anomalies in traffic behaviour, i.e. a one-time large event such as a football game which causes extreme
congestion for a relatively short period of time?

5. How adaptive are the algorithms described in this research to concept drift, i.e. the changing of traffic behaviour in a road network over time? We may ask ourselves whether the performance of the algorithms today is the same as in a few years from now, when the environment has changed due to, for instance, a rise in popularity of a road network.

6. Is the computational performance of the algorithms described in this research feasible for real-world applications? Ultimately, in order for our adaptive traffic light algorithms to be usable in the real world, they should be able to process and adapt to traffic data in real-time.

1.4 Research methodology

To answer our main research question, we first give detailed insights into the background of the traffic light scheduling problem, the corresponding traditional solutions that can be found in the real-world, artificial intelligence in the context of traffic light scheduling, and the current state-of-the-art machine learning techniques such as Q-learning that can be utilized in traffic light scheduling.

Using this background knowledge, we describe three classes of traffic light scheduling algorithms, namely:

1) a (static) traditional timing-based algorithm which models traffic light scheduling algorithms deployed throughout the real world;

2) a data-driven Q-learning algorithm with improvements to counter state- and action-space explosions, representing the current state-of-the-art in the application of (non-deep) reinforcement learning in traffic light scheduling;

3) a number of data-driven deep Q-learning algorithms, each taking a different granularity of traffic state information into account.

To analyze the performance and ultimately answer our research questions, we propose and describe a generalized traffic simulation framework that allows us to realistically model traffic flows in complex road networks, in which the traffic intersections can be governed by the various types of traffic light scheduling algorithms described in this research.

The research questions are then answered by a number of experiments which are conducted within the generalized traffic simulation framework. The experiments test various important aspects of the traffic light scheduling algorithms, such as:

1) the ability to learn from recurrent contexts (seasonality);

2) the robustness against anomalies in traffic behaviour;

3) the adaptiveness to concept drift;

4) the computational performance.

Using the outcomes of these experiments, each research question can be answered accordingly.
1.5 Main results & contributions

This research advances the current state-of-the-art in deep reinforcement learning and traffic light scheduling further, by making the following contributions:

1. We demonstrate that deep reinforcement learning, and in particular deep Q-learning, can be effectively implemented using classical feed-forward ANNs.

2. As a result of the implementation of deep reinforcement learning with a classical neural network, our reinforcement learning agent can now perform pattern recognition solely on useful information, i.e. only the information (without noise) that is important for the agent to reason about. This is in contrast to the current state-of-the-art research in deep reinforcement learning, which currently only focuses on the employment of convolutional neural networks that are specialized in pattern recognition in images [5]. Thus, we can now train our reinforcement learning agent to recognize patterns in non-visual data, which opens up the application of deep reinforcement learning in a vast amount of other problem domains, such as traffic light scheduling in our case.

3. We demonstrate, using a generalized traffic simulation framework, that traffic light scheduling algorithms based on deep reinforcement learning converge faster, learn better from seasonality, are more robust to special situations, and last but not least, perform better than both traditional timing-based approaches used in the real-world, as previously studied state-of-the-art approaches in traffic light scheduling that utilize traditional reinforcement learning.

4. By utilizing deep reinforcement learning in traffic light scheduling under regular circumstances, we are able to cut average waiting times by 58 – 86% (76% on average) when compared to a real-world approach and by 29 – 77% (59% on average) when compared to a traditional reinforcement learning approach.

5. Also, under these same circumstances, we are able to increase the average road network throughput by up to 16% when compared to a real-world approach and by up to 15% when compared to a traditional reinforcement learning approach. These numbers have potential to grow, since we currently only maximize the throughput of each individual traffic intersection, while a form of (deep) co-learning could allow for maximization of the road network throughput as a whole.

6. The algorithms are designed and implemented in a generalized way such that they are easy to deploy in real-world, but also simulated contexts.
CHAPTER 1. INTRODUCTION

1.6 Outline

The outline of this research closely follows the research methodology described in Section 1.4. In Section 2, we first give detailed insights into the background of the traffic light scheduling problem, the corresponding traditional solutions that can be found in the real-world, artificial intelligence in the context of traffic light scheduling, and the current state-of-the-art machine learning techniques such as Q-learning that can be utilized in traffic light scheduling. In Section 3, we describe three types of traffic light scheduling algorithms, namely that of a static timing-based algorithm deployed throughout the real-world, a data-driven non-deep reinforcement learning algorithm, and finally a number of data-driven deep reinforcement learning algorithms. To answer our research questions, we present a generalized traffic simulation framework in Section 3, which allows us to realistically benchmark the different types of traffic light scheduling algorithms. Using the generalized traffic simulation framework, we describe, conduct, and discuss a number of experiments in our experimental evaluation of Section 5. Each of these experiments cover one or more research questions, which are ultimately answered in our conclusion of Section 6.
Chapter 2

Background

In this chapter, we present in-depth background knowledge on the world of traffic light scheduling, required to understand the traffic light scheduling algorithms and experiments later on in this research.

2.1 Approaches to traffic light scheduling

We start off by discussing the various differences and trade-offs between traditional and data-driven approaches in traffic light scheduling.

2.1.1 Traditional approaches

Traditional approaches to traffic light scheduling are approaches that are static: they always react in the same way to the same input. We encounter these approaches mostly in the real-world. We can distinguish two classes of traditional traffic light scheduling algorithms, namely fixed time control and dynamic time control [6].

Traffic light scheduling algorithms based on fixed time control represent the most simple form of traffic light scheduling. The scheduling algorithm simply cycles through a list of traffic light configurations, switching on each configuration, turning it to “green”, for a (preset) fixed amount of time.

Dynamic time control is a more advanced approach to traffic light scheduling. Algorithms governed by dynamic time control are fed by some type of input. On the one hand, this input can be based on traffic models of a traffic intersection that are built offline, by recording each occurrence of a vehicle, its direction and the time it has passed the traffic intersection. From this information, it is possible to build algorithms which use the information from these models to modify their phase times for each of the traffic light configurations according to the time-of-day, day-of-week, and/or day-of-year to maximize traffic throughput. On the other hand, this input can be based on the incorporation of inductive loops in the roadbed, which makes the traffic controller aware of the presence of traffic. Note that the inductive loops do not give information on the amount of waiting traffic (queue lengths), but simply whether vehicles are waiting at different directions or not. This information is used by the algorithms to further optimize traffic throughput and delays, by incorporating a set of pre-defined rules.
An example of such rule would be that when only one direction has waiting traffic, then the traffic light for that specific direction should be turned to “green”.

2.1.2 Data-driven approaches

Note that the traditional approaches to traffic light scheduling described in the previous section are all based on models of which the parameters are static. Even though approaches governed by dynamic time control are more intelligent in the sense that traffic data is used (i.e. traffic models and/or presence-detection of traffic), their corresponding algorithms always react in the same way to the same traffic data, using a set of parameters preset during installation, rendering them static. This is contrary to data-driven approaches, which adapt their behaviour over time when new traffic data is processed. The algorithms may react differently over time to the same traffic data, which makes these algorithms adaptive.

Using traffic models to create traffic light scheduling algorithms can be tedious, expensive and the resulting models are rarely optimal for single problem instances. This is due to the fact that traffic models are very time-consuming to create (often this is still done by counting vehicles by hand), and even more so to process them in order to build an algorithm upon them. Note that traffic models are also highly subject to concept drift, meaning that the underlying distribution that generates the target variable, which the model is trying to predict, changes over time in unforeseen ways [7]. This happens in the situation when the structure of the road network changes, or when a certain road changes in popularity due to the fact that more people live nearby, for instance. New roads may be opened or closed elsewhere, affecting the way traffic travels through the network and thus rendering the initial traffic model that was built for an intersection incorrect. When taking into consideration that the the cost of traffic light installations range from several hundred thousands to several million dollars for more complex intersections\(^1\), while most of those costs are incurred by specialists that optimize the traffic light timings [8], questions arise about the sustainability of this approach.

Thus, a key objective of this research is to explore ways of cutting out these traffic models that use static parameters and find novel model-free approaches which solely operate on adaptive model-free processing of traffic data: the “data-driven” approach. In light of promising new developments in the field of machine learning, this research focuses on adopting autonomous learning mechanisms in the data-driven traffic light scheduling algorithms. The choice for using machine learning is only logical, as algorithms in this class are purely data-driven, adaptive and model-free, which are the key properties we are looking for in a traffic light scheduling algorithm.

However, we should always keep in mind that there are certain pitfalls when dealing with model-free algorithms:

1) as the algorithms should be designed for usage in the real-world, we should avoid to use vehicle drivers as “lab mice”: simply attempting different traffic light schedules without imposing any solid boundaries on what can be attempted may have a negative impact on the reputation of the system;

\(^1\) In an interview conducted with a municipality in the Netherlands, officials indicated that they spend 150,000 euros on a traffic light installation for a small intersection.
2) the algorithms should be robust against anomalies in traffic behaviour (i.e. a football event bringing an extreme amount of traffic to the network): in rare situations, the algorithms should not behave in a “weird” fashion;

3) as the algorithms are adaptive, we want the algorithms to adapt to new behaviour in a smooth manner: we should incorporate smoothing mechanisms which only allows for gradual changes in traffic light schedules to avoid behaviour of the traffic light controller which can be considered as “extreme”.

Shared knowledge

So, where does the “data” in “data-driven” come from in case of a traffic intersection? As mentioned in Section 2.1.1, the only currently widely-deployed source of real-time traffic data at traffic intersections is that of inductive loops embedded in the roadbed. However, these sensors only determine the presence of traffic, while we are interested in real-time data with much more fine-grained information about the traffic situation. Such information can be obtained through various ways, varying from different types of sensors embedded into the roadbed to the usage of video cameras on the traffic lights in combination with video processing techniques. However, as this is not in the scope of this research, we will not discuss the technical details of these techniques.

In this research, we assume that each traffic light controller in a road network \( N(I,R) \), described in Definition 2.4.5, has as shared knowledge the queue lengths at each traffic intersection in road network \( N \). Thus, we also assume that the traffic light controllers are able to communicate the information about the queue lengths at their own traffic intersections to each other, i.e. via the Internet.

2.2 Incorporating machine learning in data-driven traffic light scheduling

In this section, we will discuss which machine learning techniques are suitable to facilitate data-driven traffic light scheduling.

2.2.1 Offline versus online learning

The first aspect of machine learning that we will be treating is the choice between offline and online learning, or a combination between the two. Online learning is the process of learning “on-the-fly”, meaning that the algorithm updates its decision-making mechanism when new data arrives (i.e. in a streaming setting): the dataset grows over time. Offline learning is the process of learning in based on a static dataset, meaning that the algorithm processes a large corpus of data once and updates its decision-making mechanism based solely on that data. Typically, online learning is used when the dataset is or will eventually become too big to store. Hence, once its decision-making mechanism is updated with the training instance, the instance is thrown away either immediately or at some point later in time.

When looking at the traffic light scheduling problem, we have a training dataset that grows uninhibited over years of time. This dataset eventually becomes too big and thus too expens-
ive to store. Also, we start out with a cold start problem: in the beginning there is no training data. The training datasets require us to have traffic data combined with corresponding optimal traffic light schedules. As the training dataset is of complex nature (traffic flows on one end of the road network affect traffic flows on the other side of the road network), finding such a dataset is infeasible. If we were able to find such datasets in a feasible amount of time, then we would not have to incorporate optimization techniques such as machine learning to find better traffic light schedules, as we would already have an exact algorithm which would do this for us.

Hence, for our traffic light controller the best fit seems to be a form of online learning where training data is generated in a streaming setting, by allowing the controller to act as an agent interacting with its environment attempting to predict and optimize some kind of reward or cost function.

The caveat of the cold start problem is that putting an untrained traffic light controller in the field may have dramatic performance on any cost function, because essentially what will happen is that the controller will take a bunch of random decisions. We do not want this, and thus, we may consider an offline learning phase where we train the traffic light controller using training data generated by simulations and models. In this way, some basic “behavioural knowledge” is pre-trained into the controller before it is set out in the field.

2.2.2 Supervised, unsupervised, and reinforcement learning

In machine learning, there are three classes of algorithms that can be distinguished, namely those of supervised, unsupervised, and reinforcement learning.

In supervised learning, there is a training dataset available which contains both the inputs and the desired outputs, which is used by the algorithm to train its decision-making mechanism. In terms of our data-driven traffic light scheduling algorithms, we already know that we need to minimize some cost function by making a decision given a traffic situation as input. Since traffic is a non-deterministic process and the decisions our algorithms make have direct influence on the subsequent traffic simulations of not only their own intersections but also of neighbouring traffic intersections, it is simply impossible to have any training data a priori. Thus, for our data-driven algorithms, using supervised learning alone is out of the question.

In unsupervised learning, a training dataset is not available a priori and thus the desired outputs are not available either. Thus, learning in unsupervised learning usually resorts to the clustering of data points into unlabeled clusters. Similarly to the case of supervised learning, the model of unsupervised learning does not fit the concepts of our data-driven algorithms either. This is because there is no notion of clustering and labels in the traffic light scheduling problem.

In reinforcement learning, algorithms train their decision-making mechanisms by acting as an agent that interacts with some environment through actions in such a way that some long-term (possibly delayed) reward is maximized (see Figure 2.1) [9]. The environment is in a certain state which can be changed by the agent by performing an action, for which the environment goes into a new state. The set of states, actions, and the rules that specify which
actions are activated in which state make up for a Markov Decision Process if and only if the probability of whether the environment enters a next state depends only on the current state and action, and not any preceding states or actions (Markov assumption). Note that most reinforcement learning algorithms rely on the underlying process to be a Markov Decision Process in order for them to work, and thus it is crucial that in case of traffic light scheduling, we describe states and actions in such a way that the resulting process adheres to the Markov assumption. The rules that specify which action to choose in which state is called the policy.

The reinforcement learning model exactly suits our needs in the traffic light scheduling problem, as the agent is the traffic light controller making changes to the traffic light schedule via actions and with those actions it ultimately changes, for better or worse, the traffic situation, which is the state of the environment. Ultimately, the traffic light controller wants to maximize some long-term reward, or minimize some cost function. Reinforcement learning is also especially suitable for online learning [10]. It should come as no surprise, then, that we conclude that the reinforcement learning class of machine learning algorithms is most suitable for our data-driven traffic light scheduling algorithms.

Reinforcement learning is governed by analyzing the rewards obtained in the Markov Decision Process by choosing actions according to a certain policy. An episode, which constitutes a single run of the system in which an agent takes actions from an initial state until it reaches a terminal state in the Markov Decision Process, is a chain of states, actions, and rewards which can be described in the following form:

$$s_0, a_0, r_1, s_1, a_2, ..., s_{n-1}, a_{n-1}, r_n, s_n,$$

(2.1)

where \(s_i\) is the state the environment is in at iteration \(i\), \(a_i\) is the action that the agent selects in iteration \(i\) and \(r_{i+1}\) is the reward the agent receives after performing action \(a_i\) in state \(s_i\) and ending up in \(s_{i+1}\). The total accumulated reward \(R\) for a single episode can trivially be expressed as the sum of all of the obtained rewards:

$$R = \sum_{0 \leq i \leq n} r_i.$$  

(2.2)

The total accumulated rewards \(R_t\) from iteration \(t\) can also be trivially expressed as the sum of all of the obtained rewards from iteration \(t\) and onwards:
Note that our environment is stochastic, as the arrival and behaviour of traffic are influenced by random variables. Thus, if we perform this episode again, we never have the guarantee that we obtain the same rewards as in previous episode, as the transitions are non-deterministic due to a stochastic environment. The further in the future, the more this uncertainty will affect our long-term outcome. Thus, a discount factor $\gamma$ is introduced such that long-term rewards are discounted using this factor, where $R_t$ is now redefined as:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{n-t} r_n = \sum_{0 \leq i \leq n-t} \gamma^i r_{t+i}, \quad (2.4)$$

which can be rewritten to the following recursive definition:

$$R_t = r_t + \gamma R_{t+1}. \quad (2.5)$$

As can be concluded from Equation 2.4 and 2.5, the further in the future, the more the reward is discounted (for $0 \leq \gamma < 1$). If we set $\gamma$ (close) to 0, our strategy will consider short-term or immediate rewards, while if we set $\gamma$ close to 1, our strategy will balance out short-term and long-term rewards. If we set $\gamma$ to 1, our strategy does not differentiate between short-term and long-term rewards. If our environment is deterministic, meaning that the same actions always lead to the same rewards, we do not have to differentiate between short-term and long-term rewards and thus we can set $\gamma$ to 1. However, in the case of traffic light scheduling, as mentioned hitherto, our environment is stochastic and thus non-deterministic. Hence, our strategy has to balance out short-term and long-term rewards. While there is no golden standard in picking a value for $\gamma$, we found that a value of 0.9 gave us good results during our experimental evaluation in Section 5.

Ultimately, to obtain the highest reward, a well-trained reinforcement learning agent should always choose, given a state $s$ and a set of enabled actions $A$ from state $s$, the action $a \in A$ such that the (discounted) future reward is maximized.

### 2.2.3 Q-learning

As stated earlier in this section, reinforcement learning is a class of algorithms. This means that there are varying implementations that come with all sorts of different advantages and disadvantages in specific problem domains. One of the most well-known and generally applicable implementations of reinforcement learning is Q-learning [11].

Q-learning is a model-free reinforcement learning algorithm. This means that reinforcement learning in Q-learning is carried out by an agent interacting solely on an environment, without requiring additional information about the environment except for an awareness of the environment states, possible (enabled) actions from its current state, and the obtained rewards after performing an action.

In Q-learning we define a Q-function, which is a function $Q : S \times A \rightarrow \mathbb{R}$ where $S$ is the set of states in the environment and $A$ is the set of actions that are possible for those states. The
Q-function $Q(s,a)$ with $s \in S$ and $a \in A$ maps state-action pairs to the maximum discounted future reward when performing action $a$ from state $s$:

$$Q(s_t, a_t) = \max R_{t+1}. \quad (2.6)$$

The letter Q is derived from the word Quality, as the Q-function represents the quality score of performing an action in a certain state. If we have the Q-function, then the ideal policy $\pi_{\text{ideal}}$ for an agent to follow to maximize the future (discounted) reward from state $s$ is to always choose the action with the highest Q-value:

$$\pi_{\text{ideal}}(s) = \arg \max_{a \in A_s} Q(s, a), \quad (2.7)$$

where $A_s$ is the set of actions that are enabled in state $s$.

Of course, we do not have the Q-function. Even more so, in the beginning, we start out without any information. So how do we get the Q-function? Well, the idea of Q-learning is that we iteratively approximate the Q-function. Consider a single transition performed by a reinforcement learning agent: $(s, a, r, s')$ where $s$ was the previous state of the agent, $a$ was the chosen action by the agent when being in state $s$, $r$ is the obtained reward for performing action $a$ in state $s$, and $s'$ is the resulting state the environment is in after the agent performed action $a$. We can express the Q-value of state-action pair $(s, a)$ in terms of the next state $s'$ using the Bellman equation in Equation 2.8.

$$Q(s, a) = r + \gamma \max_{a' \in A_s} Q(s', a') \quad (2.8)$$

Equation 2.8 states that the maximum future (discounted) reward for state-action pair $(s, a)$ is the sum between the immediate reward obtained for performing action $a$ in state $s$ and the maximum future (discounted) reward for the next state $s'$.

We can now iteratively approximate the Q-function using Equation 2.8. Assuming the Q-function is implemented as a matrix $S \times A$ with all values initially 0, we can perform the approximate the actual Q-function by executing the algorithm in Algorithm 1.

**Algorithm 1** Q-learning

1. initialize matrix $Q[S, A]$ with all values being 0
2. observe initial state $s$ from environment
3. repeat
4. $a \leftarrow \pi(s)$ with $\pi(s)$ being the policy to choose actions
5. perform action $a$
6. observe new state $s'$ and obtain reward $r$ from the environment
7. $Q[s, a] \leftarrow Q[s, a] + \alpha (r + \gamma \max_{a' \in A_s} Q[s', a'] - Q[s, a])$
8. $s \leftarrow s'$
9. until $\# A_s = 0$ (s is a terminal state)

As can be seen in Algorithm 1, from the initial state and onwards an action is chosen and executed repeatedly by some action-choosing policy $\pi(s)$. There are many different ways $\pi(s)$
can be defined, but there are pitfalls that need to be avoided when defining such a policy. For instance, a policy that only picks the action with the maximum Q-value that was observed up until now in the Q-function approximator (in this case our Q-matrix), will yield poor results when the Q-learner has not yet explored a broad range of actions: the Q-learner will find a local maximum at best. To improve upon just finding local maxima, the Q-learner should introduce some randomness when picking its actions. The $\varepsilon$-greedy action-selection policy is a policy that allows us to control the balance between choosing exploitative actions, i.e. actions with as goal to maximize the future long-term reward, and explorative actions, i.e. actions with as goal to randomly explore the state space in order to find better Q-values. The $\varepsilon$-greedy policy chooses the action with the maximum Q-value that was observed up until now with probability $1 - \varepsilon$, while it chooses a random action with probability $\varepsilon$. In this way, a balance can be found between exploring “uncharted territory” in search for even better actions and choosing the best action. While there is no golden standard in picking a value for $\varepsilon$, we found that a value of 0.1 gave good results during our experimental evaluation in Section 5.

Also, the update rule of $Q(s,a)$ in Algorithm 1 slightly differs from Equation 2.8 in the sense that a learning factor $\alpha$ is introduced to cater for smoother changes of the Q-function approximation. For $\alpha = 1$, the update rule behaves exactly as Equation 2.8, while with smaller $\alpha$’s we can control how much the difference between the previous and new Q-value is taken into account. For instance, we might want to block unique and one-off situations in which a generally bad action leads to a good reward. By using a learning rate with an $\alpha < 1$, we can ensure that traffic light schedules are only gradually changed. This increases the stability of the system. That the Q-function approximation described in Equation 2.8 converges and approximates the actual Q-function is proven in [12].

2.3 State-of-the-art in data-driven traffic light scheduling

Now we have seen which machine learning approaches to traffic light scheduling are suitable, we can now describe the current state-of-the-art in data-driven traffic light scheduling that utilizes machine learning.

From the literature, we can distinguish two flavours of adaptive traffic signal control approaches, namely approaches which are governed by unsupervised learning (e.g. learning artificial neural networks by [1] and [2]) and approaches which are governed by reinforcement learning (e.g. learning through Q-Learning by [3] and Multi-Agent Reinforcement Learning (MARL) by [13] and [4]). Since optimal traffic light schedules are unknown a priori, it is not possible to utilize a supervised learning technique, as no training data is available. In the following sections, both approaches and variations in approaches found in literature will be explained and compared.

2.3.1 Unsupervised learning

Early research by Gilmore et al. [1] in 1993 and Smith et al. [2] in 1994 covered the approach of learning optimal traffic schedules by unsupervised training of artificial neural networks. The approach by Gilmore et al. utilizes a Hopfield neural network which allows for the unsupervised learning via an energy function that adapts the neuron weights such that the function converges to a (local) minimum. The approach trains a single neural network for the
entire global road network. The energy function incorporated to converge the neuron weights takes into account the following parameters:

1) the number of traffic lights;
2) the number of incoming roads into the network;
3) the number of outgoing roads out of the network;
4) the current duration that the traffic light has been on;
5) the difference in load factor (fraction between road load and capacity) of the involved roads for the traffic lights.

Interestingly, no comparison was made by Gilmore et al. between a pre-timed scheduling algorithm and their own scheduling algorithm which utilizes a neural network. Both researches conducted by Gilmore et al. [1] and Smith et al. [2] also use neural networks to predict traffic congestion. Note that training data is available for this problem, so they can now resort to supervised learning. Predictions of traffic congestion can be useful as one of the inputs to our traffic light scheduling problem, as one of the goals of the solution to our problem is to minimize future traffic congestion. Both studies did not research this and thus it remains an open research question.

2.3.2 Reinforcement learning

More recent research from the year 2000 and onwards focuses on the approach of regarding the adaptation of traffic light schedules at traffic intersections as a (Partially Observable) Markov Decision Process (POMDP), such that the adaptation policy can be trained using reinforcement learning (as described in Section 2.2.2).

Abdulhai et al. studied the application of reinforcement learning (Q-learning) to a single traffic intersection in isolation and reported significant improvements in average delay per vehicle in comparison to pre-timed traffic light controllers [3]. For highly variable traffic flows, average vehicle delays were more than halved.

In 2000, researcher Wiering studied the application of reinforcement learning (Q-learning) to a global traffic light scheduling (considering all traffic intersections in the road network) in a multi-agent system (MAS), which means that each traffic intersection has one agent which applies a reinforcement learning algorithm that co-lears with agents in neighbouring traffic intersections in the road network [13]. Wiering adopted a model-based reinforcement learning approach, meaning that first a transition model estimating the rewards is trained. Wiering’s study reports a significant decrease of 20% in average waiting times when using co-learning in contrast to no co-learning. The study also reports a significant decrease in average waiting times when using the multi-agent reinforcement technique instead of pre-timed traffic light controllers.

In 2012, Abdulhai et al. produced a PhD thesis on the subject of multi-agent reinforcement learning with the same ingredients as the study conducted by Wiering in 2000, but instead utilizing a model-free approach [4]. The study reports a 25% and 35% decrease in average vehicle delays when comparing their system to actuated and pre-timed traffic light controllers, respectively.
2.4 Traffic light scheduling problem formalization

To obtain an understanding of the formalisms that we use during the presentation of the traffic light scheduling algorithms in Section 3 and the generalized traffic simulation framework in Section 4, we end this chapter by formalizing the traffic light scheduling problem. We formalize this problem as an optimization problem. We start out by defining some basic definitions for a single traffic intersection $i$ that is controlled by a traffic light scheduling controller $tlc_i$.

Definition 2.4.1. Intersection $i$ has a non-empty set of incoming roads $R_{i,incoming}$ and outgoing roads $R_{i,outgoing}$.

Definition 2.4.2. Intersection $i$ has a non-empty set of legal crossings $LC_i \in \mathcal{P}(R_{i,incoming} \times R_{i,outgoing})$ which defines a set of tuples $(r_1, r_2)$ with $r_1 \in R_{i,incoming}$ and $r_2 \in R_{i,outgoing}$ which allows traffic flows from $r_1$ to $r_2$.

Definition 2.4.3. Intersection $i$ also has a non-empty set of legal traffic light configurations $LTLC_i \in \mathcal{P}(LC_i)$ where each configuration $\in LTLC_i$ defines the legal crossings where traffic can flow through simultaneously when the configuration is activated by the traffic light controller that controls intersection $i$ (i.e. the simultaneous traffic lights that are “green”).

For an example of an intersection with two legal traffic light configurations, see Figure 2.2.

Figure 2.2: An example of an intersection with two legal traffic light configurations $c_1$ (colored green with two legal crossings from north to south and vice versa) and $c_2$ (colored red with two legal crossings from east to west and vice versa), as per Definitions 2.4.2 and 2.4.3.

Thus, the traffic lights in intersection $i$ is controlled by traffic light controller $tlc_i$ where $tlc_i$ assigns for each time instance $t$ in our time universe $T$ a single activated configuration from $LTLC_i$.

Definition 2.4.4. The lifetime of decisions of a traffic light controller $tlc_i$ can be described as the following total function: $T \rightarrow LTLC_i$. 

20 On Deep Reinforcement Learning for Data-Driven Traffic Control
CHAPTER 2. BACKGROUND

So far we have only regarded a single traffic intersection. However, traffic congestion is hardly a problem that has its roots that are local to a single traffic intersection. It is a problem that accumulates over an entire road network.

**Definition 2.4.5.** We can formalize a road network $N$ as a directed graph $N(I, R)$ where $I$ is the set of intersections and $R$ is the set of roads (where each road is a directed edge between a pair of intersections).

Now we have formalized the structure and rules of the road network $N(I, R)$, but we also have to account for the traffic that is on those roads.

**Definition 2.4.6.** Each road $r \in R$ has a maximum capacity $\text{capacity}(r) \in \mathbb{N}$ which represents the maximum number of vehicles that fit road $r$.

**Definition 2.4.7.** Each road $r \in R$ has at each time $t \in T$ a current load $\text{load}(t, r) \in \mathbb{N}$ which represents the number of vehicles that reside on road $r$ at time $t$.

Note that future work could benefit from including time $t$ in the road capacities, making the road capacity a function over time, allowing us to dynamically change the road capacity over time. This would allow us to test our algorithms under changing conditions, i.e. account for roads closing down due to construction work or accidents. Also, road capacities may fluctuate because the velocities of the vehicles on the road fluctuate. With higher velocities, vehicles will usually retain a larger distance to each other than with lower velocities.

Now we have formalized the basic notation behind traffic intersections, we can focus on the traffic light scheduling problem. The traffic light scheduling problem is an optimization problem.

**Definition 2.4.8.** The traffic light scheduling problem is defined as follows: for a given road network $N(I, R)$ and time $t$, we are looking for each traffic intersection $i \in I$ to make a decision $d_i = (t', \text{ltlc}'_i) \in T \times \text{LTLC}_i$, representing that at time $t' \geq t$ the traffic light configuration that was activated at time $t$ for intersection $i$ changes to $\text{ltlc}'_i$, such that some cost function $\text{cost}(D, t)$ over all of the decisions $d \in D$ made for $N(I, R)$ at time $t$ is minimized.

The cost function $\text{cost}(D, t)$ can incorporate any (combination of) quality attributes that we wish to optimize for. Those quality attributes can range from average delays to $CO_2$ emissions, depending on what goals we set for our algorithms (to attempt) to achieve. The cost function may also take traffic throughput into account.

**Definition 2.4.9.** The intersection throughput is defined as the number of vehicles that cross an intersection within some duration $\Delta t$.

**Definition 2.4.10.** The road network throughput is defined as the number of vehicles that enter and either arrive at their destination within the road network or leave the road network within some duration $\Delta t$.

The cost function may also take into account attributes that attain to improving the quality of life of the drivers within the road network.

**Definition 2.4.11.** The average waiting time is defined as the average amount of time that vehicles spent waiting during their journey through the road network within some duration $\Delta t$. 

On Deep Reinforcement Learning for Data-Driven Traffic Control 21
Note that if we only attempt to minimize the average waiting time, unfair situations may occur in which few vehicles have to wait indefinitely on a less popular direction at a traffic intersection, in favour for the large majority of the vehicles driving on a popular direction.

**Definition 2.4.12.** The *average speed* is defined as the total distance travelled by the vehicles in the road network divided by the total duration of their journeys within some duration $\Delta t$.

Again, the same unfair situation may occur by only attempting to maximize the average speed as could occur when only attempting to minimize the average waiting time. However, in highly congestive situations in which all of the incoming roads to a traffic intersection are occupied by a substantial amount of the traffic at that intersection, these situations are less likely to occur and thus, these attributes do tell us something about the quality of life within the road network.
Chapter 3

Traffic light scheduling algorithms

In this section, we describe and propose three types of traffic light scheduling algorithms, namely:

1) a *traditional* timing-based algorithm which models traffic light scheduling algorithms deployed throughout the real world (see Section 3.2);

2) a data-driven Q-learning algorithm with improvements to counter state- and action-space explosions, representing the current state-of-the-art in the application of (non-deep) reinforcement learning in traffic light scheduling (see Section 3.3.1);

3) a number of data-driven *deep* Q-learning algorithms, each taking a different granularity of traffic state information into account, representing the newly proposed approach of utilizing *deep reinforcement learning* using classical artificial neural networks (see Sections 3.3.2, 3.3.3, and 3.3.4).

For each of the algorithms discussed in the section, we describe its “raison d’être”, the underlying theory of the algorithm, the performance of the algorithm in terms of computing resources, and last but certainly not least: the algorithm itself. To understand the algorithms described in this section, it is important to recall the traffic light scheduling problem formalization described in Section 2.4.

### 3.1 Research goals

In order to cover the research questions in Section 1.3, we should cover the following aspects when describing each of the data-driven traffic light scheduling algorithms:

1) **type**: the type of machine learning that is used to solve the traffic light scheduling problem;

2) **recurrent contexts**: the ability to learn from *recurrent contexts* (seasonality) in traffic behaviour;

3) **robustness**: robustness against anomalies in traffic behaviour, for which the goal should be to minimize the negative impact on the performance once an anomaly occurs;

4) **adaptation**: adaptation to concept drift in traffic behaviour;
5) **performance**: feasibility of computational performance in real-world applications.

Which algorithm covers which research goals is displayed in Table 3.1. The fixed time-based algorithm is described in Section 3.2. The Q-learning algorithm is described in Section 3.3.1, while the Deep Q-learning algorithms I, II, and III are described in Sections 3.3.2, 3.3.3, 3.3.4, respectively.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AI type</th>
<th>Recurrent contexts</th>
<th>robustness</th>
<th>adaptation</th>
<th>performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed time-based</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q-learning</td>
<td>QL</td>
<td>{X}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deep Q-learning I</td>
<td>QL+ANN</td>
<td>{X}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deep Q-learning II</td>
<td>QL+ANN</td>
<td>{X}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deep Q-learning III</td>
<td>QL+ANN</td>
<td>{X}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: Table containing information about which research goal is covered by which algorithm (QL = Q-learning, ANN = Artificial Neural Network)

### 3.2 Traditional algorithms

Traditional algorithms, which are currently widely deployed throughout the real world, mainly consist of pre-timed traffic light schedules, which are usually preset upon installation according to a human expert in the field of traffic light scheduling. Often, these algorithms are actuated by sensors in the roadbed detecting the presence of traffic, allowing the algorithms to alter their schedules in situations where there is only traffic present on a strict subset of the roads connected to a traffic intersection. In this research, however, we are mainly interested in studying situations with high traffic congestion, because these situations (i.e. during rush hours) are known for their dramatically deteriorating effects on traveling times. When there is high traffic congestion at a traffic intersection, the sensors in the roadbed will always detect the presence of traffic, and thus, provide little insight and benefit to the actuated algorithms in traditional traffic light controllers. As such, we will not implement an algorithm that is actuated by these traffic-presence sensors, but only consider a simple algorithm which only utilizes pre-timed traffic light schedules.

**Fixed time-based**

The fixed time-based algorithm is the most simple form of traffic light scheduling algorithm: namely that of adhering to a fixed pre-defined traffic light schedule. The algorithm takes as input a traffic light schedule $S$, which contains an ordering of the legal traffic light configurations $LTLC_i$ and a mapping with “green” times of each of these configurations. The algorithm simply cycles through the legal traffic light configurations, where it waits for the “green” time of each of the configurations in $S$ to finish before cycling through to the next configuration in $S$. Once the algorithm has completely cycled through all configuration in $S$, the algorithm restarts at the first traffic light configuration in $S$. 
CHAPTER 3. TRAFFIC LIGHT SCHEDULING ALGORITHMS

We can clearly see that performance in terms of computing resources will never be an issue with this fairly trivial algorithm, as it simply adheres to a pre-defined schedule, where the number of legal traffic light configurations is only a handful in any real-world situation.

3.3 Data-driven algorithms

Data-driven algorithms are inherently more complex than their non-data-driven counterpart described in the previous section. The idea of data-driven traffic controllers is that they use smart road sensors to gain more knowledge on the traffic situation. The goal is to use this knowledge that is built up over time to make better scheduling decisions. The complexity lies in the fact that the knowledge can be built up in an unbounded fashion, while scheduling decisions need to be made with limited computing (time and memory) resources. Hence, we do not have the luxury to analyze the complete history of knowledge in order to make the best scheduling decision possible. Therefore, we should consider the machine learning techniques described in Section 2.2, where we concluded that reinforcement learning has the best fit when it comes to finding (close to) optimal traffic light scheduling decisions based on built up experience of scheduling decisions made in the past.

In this section, we present two types of traffic light scheduling algorithms, namely one algorithm based on traditional (non-deep) reinforcement learning, and a few algorithms based on deep reinforcement learning. Recall from Section 2.1.2 that we assume that our traffic light scheduling algorithms have access to all of the queue lengths on each road of the network they operate in. For details of how this can be achieved in the real-world, refer to Section 2.1.2.

3.3.1 Q-learning with reduced state- and action-space

The first algorithm we discuss is a Q-learning algorithm representing the current state-of-the-art described in Section 2.3. Recall from Section 2.2.3 that a traditional Q-learning algorithm must specify what is encoded in the state, which actions are possible, and it must be able to compute the reward between two subsequent states using some reward function.

**State definition**

This algorithm encodes the following information into the environment state:

1) the traffic light configuration that is currently active (i.e. “green”);

2) the number of waiting vehicles (queue length) for each of the traffic light configurations.

Note that a traffic light configuration is nothing other than a set of legal crossings, as defined in Definition 2.4.3. Thus, the number of waiting vehicles for a traffic light configuration $tlc_i$ is defined as follows:

$$ql_i = \sum_{(r,r')}^{tlc_i} \text{number of waiting vehicles (queue length) at road } r.$$  \hspace{1cm} (3.1)

Note that with this definition, as there are extremely many different combinations of queue lengths, the number of possible states will explode. State space explosion is a big problem for any Q-learning agent, as the agent will experience difficulties when learning from past
experience when it only rarely sees a state that it has encountered before. Suppose that we have four traffic light configurations, where the agent observes the following queue lengths for each of the four traffic light configurations: \{10, 19, 15, 10\}. At some point later in time, the agent observes the queue lengths again, resulting in the following queue lengths for each of the four traffic light configurations: \{10, 20, 15, 10\}. It is easy to notice that both variants of the queue lengths are almost identical, while the agent observes them as completely different states. As a result, the agent will not benefit from the experience learned in the former observation of the queue lengths.

To counter this problem, we reduce the state space by dividing the queue lengths into discrete steps of size $n$. For instance, instead of allowing queue lengths of \{0, 1, 2, 3, 4, 5, ...\}, we now only allow queue lengths of the following sizes: \{0, 5, 10, 15, ...\}. During the observation of the queue lengths, we round the lengths to the nearest multiple of $n$, where in this example, $n = 5$. Thus, a queue length of size 7 will be rounded down to 5 and a queue length of size 8 will be rounded up to 10. At the cost of a slight loss of accuracy, we dramatically decrease the state space and thus allow for our traffic light controller to learn much faster from its past experiences.

**Action definition**

An action is defined by the following parameters:

1) the next traffic light configuration to be made active;

2) the duration that the next traffic light configuration is active (i.e. “green time”).

Note that with this definition, as the duration can be any number greater than zero, the number of possible actions that can be performed from a certain state is infinite, leading to action space explosion. Similarly as in the case of the state space explosion problem, having an explosion in our action space will result in poor convergence of the learning process of our traffic light controller, as it continuously explores actions that it did not attempt before. However, as we are scheduling traffic lights, there are only a finite number of acceptable “green time” durations: there is no point in keeping a traffic light green for more than a few minutes, let alone a few days or even months. Similarly, the effect of having a green time of 60 seconds or 61 seconds in terms of traffic throughput is negligible, allowing us to apply the same solution to the problem as we did for the state space explosion. Hence, we reduce the action space by dividing the duration into discrete steps of $m$, while adding a maximum allowed traffic light duration $m_{max}$. Thus, the possible “green time” durations will now consist of the following set of natural numbers:

$$D_{legal} \equiv \{ i \in \mathbb{N} \mid i > 0 \land i \mod m = 0 \}. \quad (3.2)$$

**Reward function definition**

Our goal is to maximize the throughput of an intersection, meaning that we want to maximize the number of vehicles passing through the intersection. Hence, the reward between two successive states $s$ and $s'$ of intersection $i$ is defined as follows:
CHAPTER 3. TRAFFIC LIGHT SCHEDULING ALGORITHMS

Action:
Next green: $c_2$
Green time: 10s

Situation at end of green phase $c_1$

Situation at end of green phase $c_2$

Reward = # waiting vehicles in $s$ - # waiting vehicles in $s'$
= 5 – 2 = 3

Figure 3.1: An example of the interplay between states, actions, and rewards in the Q-learning traffic light scheduling algorithm.

...
action-selection policy that keeps exploring actions randomly with some constant rate. If this is not the case, i.e. by using our $\varepsilon$-greedy policy with $\varepsilon = 0$ (which means that only the actions are chosen with the best Q-value), the Q-function will not update the Q-values of actions that our controller already believes are “bad” (i.e. have a lower Q-value), as these actions are never chosen to be performed again. Due to concept drift, these “bad” actions may become better over time, while our “best” actions may become worse over time. Thus, our algorithm can be made to adapt to concept drift by choosing an $\varepsilon > 0$ in our $\varepsilon$-greedy action-selection policy such that the Q-function continuously updates all possibly existing Q-values.

**Computational performance**

The time complexity for a single decision loop of the algorithm is $O(|A|)$, where $A$ is the set of possible actions. Since we strongly limit the number of possible actions by introducing a maximum duration, and only allowing durations with a multiple of some non-zero natural number, the time complexity of the algorithm has been made acceptable for real-world applications.

The storage complexity for the algorithm is slightly more complicated. Instead of using a matrix implementation for our Q-function as proposed in the Q-learning algorithm in Algorithm 1, which has a storage complexity of $O(|S| \times |A|)$ (where $S$ is the set of possible states), we utilize a more efficient hash map implementation. We map each observed state to a list of Q-values for each action, and thus our storage complexity results in $O(|S_{observed}| \times |A|)$ (where $S_{observed}$ is the set of states that have been observed by our agent). We have that, as already explained earlier, $|A|$ is negligible, while $|S_{observed}|$ may grow in an unbounded manner. However, using the proposed state space reduction, the growth of $|S_{observed}|$ has reduced dramatically.

Let us consider a concrete real-world example: consider 4 possible traffic light configurations, where we observe at most a queue length of 30 vehicles for each of the configurations. Hence, the maximum size of the observed state space without reduction is: $4 \times 30^4 = 3240000$, while the maximum size of the observed state space with reduction (queue lengths in multiple of 5) is: $4 \times \left(\frac{30}{5}\right)^4 = 4 \times 6^4 = 5184$, a factor 625 less. These numbers are not shocking for any computer to handle, and thus, storage requirements will not be an issue for this algorithm in real-world applications.

As both the time and storage complexities of the algorithm are acceptable for real-world applications, we can conclude that the computational performance of the algorithm is also acceptable for real-world applications.

### 3.3.2 Deep Q-learning

We now move on to an entirely new type of traffic light scheduling algorithm, namely an algorithm based on deep reinforcement learning. This algorithm is also a Q-learning algorithm, with the same state, action, and reward definitions, but without the proposed state space reductions. The problem that we saw that was caused by the state space explosion, is that there may be many states which are very similar, but the agent does not benefit from these similarities as it simply regards these states as being completely different (see Figure 3.2). Thus,
In practice, Q-learning converges very slowly, or may not converge at all within a reasonable amount of time. We want to be able to take these similarities between states into account with the following assumption: the reward of performing the same action in two similar states is also similar. Hence, instead of introducing a discrete Q-function, we must consider a different form of Q-function which allows for smoothing out our rewards over our input domain (state-action pairs), i.e., a non-linear continuous function which takes as input a state-action pair, and gives as output a reward. The main issue is that the Q-function is initially not known, and we have to iteratively learn what it looks like. Thus, the function must be able to change over time as we have more information on what it looks like.

The task of approximating an unknown function with a known dataset of inputs and expected outputs can be solved by applying supervised (machine) learning (refer to Section 2.2.2 for more information). One of the structures allowing for such tasks are Artificial Neural Networks (ANNs), which are a family of models inspired by biological neural networks in the brains of humans and animals. ANNs are widely used and proven to be very effective at handling the task of unknown function approximation [14]. Because of the unique manner in which ANNs can recognize patterns in data [14], we will utilize an ANN as Q-function for this algorithm.

**Artificial Neural Networks**

In short, ANNS are networks of neurons and edges between these neurons. Often, ANNs consist of multiple layers of neurons, namely an input layer, a number of hidden layers, and an output layer (see Figure 3.3). Each of the layers have connections between them, and often it is the case that a previous layer only has a forward edge to the next layer. Such a neural network is called a feed-forward neural network.

Each neuron has a set of inputs $x_i$ with $1 \leq i \leq n$, a set of weights $w_i$ with $1 \leq i \leq n$ and an activation function $\theta$ (usually a sigmoid function) (see Figure 3.4).
The activation value of a neuron \( j \) is then computed with the following formula:

\[
\theta \left( \sum_{i=1}^{n} x_i w_{i,j} \right).
\]  

The activation value is then fed as input to the next connected neurons, until the output layer is reached. The output layer will contain the expected output according to the specified input.

The idea of training an artificial neural network is to provide a dataset with inputs and expected outputs, and use an algorithm to set the weight of each neuron input in such a way that the error over this dataset is minimal. A very well-known and widely used training algorithm for artificial neural networks is the backpropagation algorithm [15].

Now the idea is that we utilize an artificial neural network as Q-function, and thus, the neural network should take as input a state-action pair, and produce as output a corresponding Q-value. However, during the update of a Q-value during Q-learning, the algorithm has to compute Q-values for each action in the original state and in the next state (see the Q-value update rule of Algorithm 1 in Section 2.2.3). Since computing the network output in an artificial neural network is computationally expensive, we can easily save computing time by trading it in for computational space. Instead of using an artificial neural network with as input a state-action pair and as output the corresponding Q-value, we specify that the neural network takes as input a state and has an output neuron for each action containing the corresponding Q-value for that action. Thus, when computing all of the Q-values for a given state, we now only have to perform one pass through the neural network, instead of \(|A|\) passes (where \(A\) is the set of possible actions).

Figure 3.3: An Artificial Neural Network with input, hidden, and output layers
CHAPTER 3. TRAFFIC LIGHT SCHEDULING ALGORITHMS

Artificial Neural Network structure

Not only do we have to encode the input to the neural network, but we also have to determine the structure of the network: how many (hidden) layers do we use, and how many neurons are in each of these layers? With neural networks, there is no golden rule in choosing your network structure. The more hidden layers and neurons are used, the more complex functions the neural network can approximate, but also the more time it takes before it converges. Hence, a “good” neural network structure can only be found by simply trying different combinations of layers and comparing their error rates and convergence durations.

Q-learning adaptation to deep Q-learning

The Q-learning algorithm from Algorithm 1 in Section 2.2.3 must now be adapted to have a neural network as Q-function. We will refer to the neural network Q-function as the Q-network. Recall that the Q-learning algorithm first selects an action \( a \) from a current state \( s \) according to some policy \( \pi(s) \). This action is selected by setting the input neurons of the Q-network to the encoded version of the current state, described hitherto, which returns \(|A|\) output neurons for each of the possible actions \( A \), containing the (normalized) Q-values for state \( s \). These Q-values can then be used by the action-selection policy \( \pi(s) \) to determine which action to perform.

After an action \( a \) has been selected and performed, and the environment has arrived at state \( s' \) with reward \( r \), then the Q-function should be updated with the newly obtained Q-value. Note that our original Q-value learning rule from Section 2.2.3 was defined as follows:
CHAPTER 3. TRAFFIC LIGHT SCHEDULING ALGORITHMS

\[ Q(s, a) \leftarrow Q(s, a) + \alpha (r + \gamma \max_{a' \in A'} Q(s', a') - Q(s, a)), \]  
with \( \alpha \) being the learning rate and \( \gamma \) being the discount factor. However, the smoothing factor \( \alpha \) does not appear anymore in the learning rule of our Q-network, since controlling the learning rate is now inherently part of the backpropagation algorithm that trains the Q-network. When the learning rate is not controlled by the update rule, we can set \( \alpha \) to 1. Hence, the training rule can be rewritten as:

\[ Q(s, a) \leftarrow r + \gamma \max_{a' \in A'} Q(s', a'). \]  
(3.6)

As explained in 3.3.2, the Q-network is not a function of the form \( Q : S \times A \to \mathbb{R} \), where we would naturally train the neural network given \((s, a_k, r, s')\) with training data of the form:

\[ \text{Q-value for } a_k \quad (\text{input: } s, a_k; \text{output: } \left(r + \gamma \max_{a' \in A'} Q(s', a')\right)). \]  
(3.7)

But our Q-network is of the form \( Q : S \to A \times \mathbb{R} \), which requires the training data to be of the form:

\[ \text{Q-value for } a_k \quad (\text{input: } s; \text{output: } \left(Q(s, a_1), ..., \left(r + \gamma \max_{a' \in A'} Q(s', a')\right), ..., Q(s, a_n)\right)), \]  
(3.8)

such that the error generated by applying the training data to the Q-network for all of the Q-values of all actions \( a_i \neq a_k \) is 0, and thus their respective Q-values do not change, except for the Q-value of \( a_k \).

Now we know how our training data is generated according to the experience gained by the traffic light controller, we have to deal with the task of incrementally training the neural network. This is a difficult task, since inherently most neural network training methods, including the backpropagation algorithm, gradually “forgets” old knowledge when new knowledge is trained into the network. Of course, it is definitely desirable to gradually retrain the network over time to make our traffic light controller adapt to concept drift, i.e. changing circumstances in the environment. For example, in years of time, traffic flows may change because of urbanization or changes to the infrastructure. However, we want complete control over what knowledge gets lost, and which knowledge gets remembered.

A solution to this problem is to keep a memory of experiences \( M \) in our traffic light controller, which acts as a First-In-First-Out (FIFO) buffer of some fixed size \( \delta \) containing experience tuples of the form:

\[ (s, a, r, s'), \]  
(3.9)

where \( s \) is an environment state, \( a \) is an action that has been performed, \( r \) is the reward obtained because the environment ended up in state \( s' \). Every time a state-action-reward-state experience cycle is finished, the controller appends this experience to the memory \( M \). Because
$M$ is a FIFO buffer of fixed size, when $M$ is full, the oldest experience tuple is removed from $M$.

Since $M$ might be very large, because $M$ may contain months or even years of gained experience by the traffic light controller, it is infeasible to continuously retrain the Q-network using the complete set of training instances in $M$. Instead, we take from this experience memory $M$ a random subset $\tilde{M} \subset M$, with $\tilde{M}$ having some fixed size $\theta$, and train the Q-network with the training instances in $\tilde{M}$.

Using this method we control the quantity of experience our traffic light controller is allowed to use by tuning $\delta$, and we control how much of this experience is trained into our neural network by tuning $\theta$.

**Robustness against anomalies in traffic behaviour**

The Q-network now has the ability to predict similar Q-values for similar states even though the controller has not observed that specific state before, but a similar one. However, when an anomaly occurs, in other words, the controller observes a state that it has not seen before, this algorithm is now slightly more robust than the Q-learning algorithm with reduced state- and action-space in the sense that if it has seen a similar state before, it is able to react on it with more knowledge. However, chances are that in case of an anomaly, a truly unique situation, the controller has not observed a similar anomaly before. Hence, while the algorithm is more robust to anomalies, it is still not ideal in terms of handling them.

**Adaptation to concept drift in traffic behaviour**

The Q-function is based on the experience contained in memory $M$, containing the last $|M|$ experience tuples experienced by the traffic light controller. Thus, when concept drift occurs, the statistical properties reflected by the experience tuples in $M$ will change gradually, and thus, the controller will always adapt to concept drift, depending on the size of $M$. If $M$ is large enough to contain a hundred years worth of experience, then concept drift caused by seasonality (i.e. summer versus winter traffic) could be missed out on. If $M$ is too small, then convergence of the Q-network will be problematic: i.e. recurrent contexts will not be detected.

**Computational performance**

Computing the output of a simple non-recursive multi-layer feed-forward neural network of a constant number of neurons for a given input has a time complexity of $O(1)$. Thus, a single run of the algorithm *without* training the Q-network (in steps 1 – 5) has a time complexity of $O(|A_{\text{possible}}|)$ (with $A_{\text{possible}}$ being the set of possible actions), as the bottleneck of the algorithm in those steps lies in iterating through all possible actions.

However, the training of the Q-network in step 6 using the backpropagation algorithm is much more expensive than retrieving Q-values from it. Training a neural network consists of minimizing the network error with respect to some training dataset. As the backpropagation algorithm may get stuck in local minima, there is no guarantee on finding the minimal error within an acceptable running time. Thus, the backpropagation algorithm iteratively attempts to find a lower error (in our case using a method called *stochastic gradient descent*) until some
minimum error $\varepsilon$ is reached, or a number of maximum iterations $\sigma$ have been performed. In the worst case, the training process has to run $\sigma$ iterations. In each iteration, the backpropagation algorithm iterates through each training example $m \in M$ with $|M| = \theta$, where:

1) the network output on $m$ is computed by doing a forward pass in $O(1)$ time;
2) the error of the network output is computed in $O(1)$ time;
3) the differences in weights are computed in $O(1)$ time;
4) the weights are updated in $O(1)$ time.

Thus, with $\sigma$ as the worst case number of iterations, the total worst case running time of our backpropagation algorithm is $O(\sigma\theta)$.

Hence, the total running time of a complete decision-making cycle is $O(|A| + \sigma\theta)$. Since we control all of these parameters, we can tune the system in such a way that the worst case running time is acceptable for real-world applications. Note that we have to trade-off between computational performance and the ability of the algorithm to achieve its desired goals, i.e. minimizing traffic delays.

In contrast to the Q-learning algorithm with reduced state- and action-space described in Section 3.3.1, which has a storage complexity of $O(|S_{\text{observed}}| \times |A|)$ (where $S_{\text{observed}}$ is the set of states that have been observed by the agent), we now no longer store Q-values for each distinct state that we observe. Instead, we approximate the Q-function in a neural network of a constant size in which we store the neural network coefficients. Also, we store all of the experiences in an experience memory $M$ of size $\delta$. Hence, the storage complexity of this algorithm is $O(1) + O(\delta) = O(\delta)$.

Let us consider a real-world example for our storage complexity. Assuming that on average, a traffic light controller makes a scheduling decision every 10 seconds. A year has a total of 31,536,000 seconds, meaning that in a year $\frac{31,536,000}{10} = 3$ million experience tuples are generated. Given the fact that each experience tuple has a (small) constant storage complexity $O(1)$, this number is not necessarily large for any modern day computer. Hence, if we were to store experience for the last year, or even the last few years, the storage requirements are definitely feasible for real-world applications.

As both the time and storage complexities of this algorithm are acceptable for real-world applications, we can conclude that the computational performance of this algorithm is also acceptable for real-world applications.

### 3.3.3 Extended Deep Q-learning

This algorithm is an extension of the Deep Q-learning algorithm described in Section 3.3.2, which is completely the same except for its state definition. The state definition of the Extended Deep Q-learning algorithm is an extended version of the state definition of the Deep Q-learning algorithm. Where in the Deep Q-learning algorithm we only took the current active traffic light configuration and the number of waiting vehicles at each of the traffic light configurations into account, we now also take the number of waiting vehicles at neighbouring
traffic intersections for each of the traffic light configurations into account. This additional state information allows our agent to “reason” in a more informed manner about the number of vehicles that will have to wait when switching the traffic light configuration to red within a certain amount of time. Because of the flexibility that the Q-network offers us, we only have to alter the input to the neural network to take more information into account.

For each of the traffic light configurations $tlc_i$, the number of waiting vehicles at neighbouring intersections that might be or are waiting to go towards traffic light configuration $tlc_i$ after the neighbouring traffic light turns green, are taken into account. We can compute the neighbouring queue length for a given traffic light configuration $tlc_i$ as follows:

$$\sum_{n \in N} \sum_{(r, r') \in LC_n \exists (r', r'') \in tlc_i} \text{number of waiting vehicles at } r,$$

where $N$ is the set of neighbouring intersections and $LC_n$ is the set of legal crossings of intersection $n$, as per Definitions 2.4.3.

**Robustness against anomalies in traffic behaviour**

As this algorithm is based on Deep Q-learning with as only difference that it has an extended version of the state definition, the same reasoning about its robustness against anomalies of traffic behavior applies to this algorithm (see Section 3.3.2): this algorithm is also not ideal in handling anomalies.

**Adaptation to concept drift in traffic behaviour**

As this algorithm is based on Deep Q-learning with as only difference that it has an extended version of the state definition, the same reasoning about its adaptation to concept drift of traffic behavior applies to this algorithm (see Section 3.3.2): this algorithm can be made adaptive to concept drift by tuning the size of its experience memory $M$.

**Computational performance**

The computational performance of this algorithm is similar to the computational performance of the Deep Q-learning algorithm, described in Section 3.3.2. The only difference between the two algorithms lies in the structure of the Q-network, where the input layer of the neural network in this algorithm has $k$ more input neurons. Because the number of traffic light configurations $k$ in real-world intersections is very small, this does not change the storage and time complexities of the Deep Q-learning algorithm in any significant way. Thus, as the computational performance of the Deep Q-learning algorithm is acceptable for real-world applications, we can conclude that the computational performance of this algorithm is also acceptable for real-world applications.

### 3.3.4 Pre-emptive Extended Deep Q-learning

This algorithm is an extension of the Extended Deep Q-learning algorithm described in Section 3.3.3. The current form of Q-learning that takes place is as follows: we observe the current traffic and traffic light state $s$, select an action $a$ based on this information, namely which traffic light configuration to activate next and for how long. At the next traffic light cycle,
we again observe the traffic and traffic light state $s'$ and compute the reward from $s$ and $s'$. However, in some cases, there might be sudden changes in circumstances such that the previously selected action $a$ turned out to be a poor decision. However, in standard Q-learning, we have to wait until the next traffic light configuration cycle before we can make a new decision.

This algorithm addresses this problem by introducing a pre-emption mechanism that allows the algorithm to preempt poor decisions. The algorithm provides an interruption point in the middle of the duration of an action, where it can re-evaluate whether the current action $a$ is still the best course of action. If it is, then nothing changes. If there is some better action $a'$ (with a higher Q-value), then the algorithm overrides action $a$ by shortening it and executing action $a'$ subsequently. By doing this, the algorithm becomes better at adjusting its behaviour after having made poor decisions.

Note that it matters whether performing an action was the result of exploration or exploitation of the state-action space. Exploration is the process of performing an action in order to further explore the state-action space, without the goal of achieving the best possible result for that action. The idea of Q-learning is that we sometimes have to make “random” (and possibly poor) decisions to learn which actions are good and which are bad in certain states. Exploitation is the process of performing the best possible action in a state in order to achieve the best possible result. The manner in which actions are explorative or exploitative is controlled by the action-selection policy. In our case the action-selection policy is the $\varepsilon$-greedy policy described in Section 2.2.3. When performing an explorative action, our goal is to learn from it. It does not matter whether it is good or bad. Especially in case of deep Q-learning, we can learn the anatomy of good and bad state-action pairs. Hence, when performing such an explorative action, we do not want to interrupt it, because it may negatively affect our learning behaviour. It is imperative that we only allow for interruption of exploitative actions, for which the goal is to achieve the best possible result.

Robustness against anomalies in traffic behaviour

In case of an anomaly in the traffic behaviour, this algorithm is now able to react and correct itself faster after having made a possibly poor decision. While the same arguments apply as with the (Deep) Q-learning algorithms (described in Sections 3.3.1 and 3.3.2), namely that the controller can only make educated decisions when it has experienced similar situations before, and as inherently this is not the case with an anomaly, it will not be able to make an educated decision. However, because this algorithm is able to review its scheduling decisions halfway through executing them, it will be more robust to small-scale anomalies that have a duration that is shorter than the green time of the action that might be preempted.

Adaptation to concept drift in traffic behaviour

As the only difference between this algorithm and the Deep Q-learning algorithm lies in the preemption of exploitative decisions, and from the fact that because we only preempt exploitative actions, the way patterns are learned over time is not affected. Note that this would not be the case when the algorithm would preempt explorative actions, as the learning process is then structurally interrupted by the preemption mechanism, always opting for the best possible action halfway. Since the learning process is not affected, we can safely apply
the same argumentation for the adaptation to concept drift of this algorithm as we did for
the Deep Q-learning algorithm described in Section 3.3.2. Mainly because of the usage of the
experience memory $M$, we are able to make this algorithm adapt to concept drift in traffic
behaviour.

Computational performance

The computational performance of this algorithm is similar to the computational performance
of the Deep Q-learning algorithm, described in Section 3.3.2. The only difference between the
two algorithms is that this algorithm has a possible preemption point for each exploitative
decision, which has a time complexity of $O(|A|)$ where $A$ is the set of possible actions and we
consider the size of the Q-network as a constant factor. In the analysis of the computational
performance of the Deep Q-learning algorithm, we concluded that the total running time of
a single run (for making a single decision and training the Q-network according to newly
gained experience) has a time complexity of $O(|A| + \sigma \theta)$, with $\sigma$ being the maximum num-
ber of learning iterations when training the Q-network and $\theta$ being the number of training
experience tuples used for training the Q-network at each training round. The additional
preemption point will add an additional $O(|A|)$ to the run, and thus, the time complexity
of this algorithm will be $O(2 \cdot |A| + \sigma \theta) = O(|A| + \sigma \theta)$, which is the same time complexity
as the Deep Q-learning algorithm. Hence, as the time complexity of the Deep Q-learning
algorithm is acceptable for real-world applications, the time complexity of this algorithm is
also acceptable for real-world applications.

This algorithm stores the exact same information as the Deep Q-learning algorithm, which
has a space complexity of $O(\delta)$, with $\delta$ being the maximum number of experience tuples that
are stored in the experience memory. The rate in which the number of experience tuples grows
will be doubled at most, as each decision might end up in two experience tuples: namely that
of the experience gained from the first half of the original decision, and the experience gained
from the new decision. Recall that it does not matter for a modern day computer to store 3 or
6 million experience tuples, representing a year of gained experience for the Deep Q-learning
algorithm and this algorithm, respectively. Hence, as the storage complexity of the Deep
Q-learning algorithm is acceptable for real-world applications, the storage complexity of this
algorithm is also acceptable for real-world applications.

As both the time and storage complexities of this algorithm are acceptable for real-world
applications, we can conclude that the computational performance of this algorithm is also
acceptable for real-world applications.
Chapter 4

Traffic simulation

In this chapter, we present a generalized traffic simulation framework that allows us to perform experiments with various implementations of traffic light controllers in a realistic visualized fashion. The aim of the traffic simulator is to accurately model and visualize (the effects of) traffic flows that are guided by traffic lights through user-defined road networks. Note that we aim, not only to realistically model traffic, but also to realistically model complex traffic light setups. The traffic is simulated at microscopic level, meaning that each vehicle is modeled separately. Our simulator is composed of four main components, namely:

1) traffic networks, described in Section 4.3;
2) traffic flows, described in Section 4.4;
3) traffic control, described in Section 4.5;
4) traffic visualization, described in Section 4.6.

4.1 Requirements

Before going into the design considerations of the components mentioned hitherto, we first describe a number of requirements for our simulation framework:

1. The simulation framework should be generic in the sense that it should be possible to:
   (a) define any kind of road network with any kind of traffic light configurations as per
       the definitions in Section 2.4;
   (b) define any kind of sensors that serve as input to the traffic light controllers;
   (c) define any kind of traffic light controller, which should allow for any complex kind
       of traffic light scheduling algorithms;
   (d) communication between traffic light controllers should be unrestricted, i.e. it should
       be possible for traffic light controllers of different intersections to communicate with
       each other within the same road network.

2. The traffic simulation should be realistic in the sense that:
   (a) traffic should be generated according to some stochastic distribution to accurately
       model real-world traffic congestion levels;
CHAPTER 4. TRAFFIC SIMULATION

(b) individual vehicles should mimic real-world behaviour, including acceleration, deceleration, adhering to speed limits, reaction times, and attaining to keeping a minimal distance between subsequent vehicles;

(c) individual vehicles should enter the road network with a predefined destination with a \textit{logical} route through the network (i.e. the shortest path), whether it be inside or outside of the road network.

3. The simulation framework should be \textit{extensible} in the sense that most of the aspects controlling the simulation framework, such as the vehicle generation and route planning modules, can be easily interchanged with different modules that model traffic in a different way. This is necessary, because future work might require that the way traffic is modeled in our simulation framework is to be changed.

4.2 Architecture

In this section, we describe the architecture of the traffic simulation framework. See Figure 4.1 for a complete overview of the architecture.

The framework consists of the following modules:

1. The \textit{Traffic Generation Module} is responsible for the process of vehicle generation and routing, described in Section 4.4.

2. The \textit{Traffic Control Module} is responsible for the process of traffic light scheduling, described in Section 4.5.

3. The \textit{Traffic State Module} is responsible for the process of updating the traffic state information that the traffic light controllers base their decisions on. The simulation of the traffic state is described in Section 4.4. This simulated module can be interchanged by a module that observes traffic data from the real-world. Thus, this framework allows for seamless integration in real-world environments.

4. The \textit{State Observation Module} is a module which allows external applications to listen to mutations in the traffic and traffic light states. Examples of such applications are the traffic visualizer, described in Section 4.6, or the statistics collector which we use to benchmark the various traffic light scheduling algorithms in the experimental evaluation in Section 5.

The simulation control loop is described in Section 4.7.
CHAPTER 4. TRAFFIC SIMULATION

Figure 4.1: Traffic simulation framework architecture
4.3 Traffic network

As stated in Section 2.4, the traffic network can be formalized as being a directed, cyclic graph \( N(I, R) \) where each vertex \( i \in I \) is an intersection and each road \( r \in R \) is a road between two intersections. Note that a bi-directional road in the real world is modeled as two distinct roads, namely one road for each direction. One of the limitations in the simulation of the road network, is that each road is modeled as a single lane. Future work would benefit in terms of realism from modeling of traffic flows on multi-lane roads, allowing vehicles to overtake each other. Each road \( r \) has a number of properties that are taken into account by the simulator, namely:

1. \( \text{source}(r) \) which is the intersection \( i \in I \) that road \( r \) originated from.
2. \( \text{target}(r) \) which is the intersection \( i \in I \) that road \( r \) heads towards.
3. \( \text{length}(r) \) is the length of road \( r \).
4. \( \text{maxSpeed}(r) \) is the maximum speed that is allowed on road \( r \).

Each intersection also has a number of properties that are taken into account by the simulator:

1. An intersection \( i \in I \) can be a source of traffic (denoted by \( \text{isSource}(i) \)), meaning that traffic can be generated randomly at this intersection.
2. An intersection \( i \in I \) can be a sink of traffic (denoted by \( \text{isSink}(i) \)), meaning that traffic can have this intersection as its destination and disappear from the simulation when having reached this intersection.
3. An intersection \( i \in I \) has a set of legal traffic light configurations \( LTLC_i \), as per Definition 2.4.3 in Section 2.4.
4. At time \( t \) in our time universe \( T \), we have that each intersection \( i \in I \) has an active intersection state \( \text{active}(t, i) \in S(i) \), as per Definition 2.4.4. Note that \( \text{active}(t, i) \) is the common decision that needs to be made by all of the traffic light scheduling algorithms. It should be no surprise by now that we would like to choose \( \text{active}(t, i) \) as good as possible, minimizing some cost function.

4.4 Traffic flows

As mentioned earlier, the simulator models traffic flows on a microscopic level, meaning that each vehicle in the simulation is modeled separately. Vehicles are generated via a Poisson distribution with intensity \( \lambda \), which we can tune to simulate different levels of traffic congestion. The vehicle generation process is defined as follows:

1. A random intersection \( s \) is selected from the set of source intersections: \( \{ i \in I \mid \text{isSource}(i) \} \).
2. A random intersection \( t \) is selected from the set of sink intersections: \( \{ i \in I \mid \text{isSink}(i) \} \).
3. The shortest route \( R = \langle r_1, ..., r_n \rangle \) between intersections \( s \) and \( t \) is selected using Dijkstra’s shortest path algorithm (note that we assume the road network to be a connected component, otherwise such a path may not exist).
4. Vehicle $v$ is generated on road $r_1$ with route $R$.

At each simulation time step $t$, vehicle positions on the corresponding roads are updated according to relative circumstances, including:

1. Maximum speed: each vehicle $v$ uses a pre-defined acceleration $\text{acceleration}(v)$ to reach either the vehicle’s top speed ($\text{topSpeed}(v)$) or to reach the road’s speed limit ($\text{speedLimit}(r)$ where $r$ is the road the vehicle is positioned on).

2. Facing vehicles: each vehicle $v$ should maintain a minimum distance $d_{\text{min}}$ between its own vehicle and the vehicle that it faces, and thus, in the case of a possible violation of $d_{\text{min}}$, the vehicle decelerates using a pre-defined deceleration $\text{deceleration}(v)$ to adjust its speed accordingly.

3. Traffic control: each vehicle $v$ timely needs to start decelerating using $\text{deceleration}(v)$ in order to stop in time for an upcoming blocked traffic intersection (i.e. a red light for the vehicle’s current direction).

4. Reaction time: each vehicle $v$ may take a random (small) amount of time before it reacts to changing circumstances, i.e. a vehicle may wait some time before it starts driving when traffic lights change from red to green.

At this moment, we do not allow vehicles to overtake each other on roads, because we only model single-lane roads. Future work would benefit from this as it provides even more realistic simulations. We also do not allow traffic collisions, as such situations are rare and very complex to model accurately.

### 4.5 Traffic control

Each intersection requires the controlling of traffic flows by making decisions on which traffic light configuration states are activated at which point in time. As discussed in Section 2, there are vastly different approaches when it comes to traffic control. However, the common traffic control decision-making process can be formalized as follows: given a road network $N(I, R)$, a traffic intersection $i \in I$, time $t \in T$, and the combined state of all the vehicles in road network $N$: $S_{\text{traffic}}$, which includes vehicle positions and corresponding velocities, make a decision $d_i = \{ t', \text{ltlc}'_i \} \in T \times \text{LTLC}_i$, where at time $t' \geq t$ the active traffic light configuration of intersection $i$ changes to $\text{ltlc}'_i$ such that some cost function is minimized. Note that this decision-making process closely follows the definition of the traffic light scheduling problem in Definition 2.4.8.

The traffic simulator provides various interfaces to implement different traffic light scheduling algorithms. Because of this, the manner in which traffic lights are scheduled is decoupled from the scheduling algorithms themselves, allowing for a generalized approach in benchmarking various versions of the traffic light scheduling algorithms. The investigated traffic light scheduling algorithms are described in Section 3.
4.6 Traffic visualization

Often we want to visually inspect what is happening to traffic flows in order to determine validity and effectiveness of our algorithms. Hence, a visualization engine has been built for the traffic simulator, which allows us to receive a live feed of the current traffic state in the road network. The road network is visualized as a graph, where the nodes are intersections, the edges are roads between these intersections, and individual vehicles are visualized as dots on the edges. At each road connected to an intersection, the vehicles can move according to the set of legal crossings (as per Definition 2.4.2), for instance straight-on, turn left or turn right. A still image of the resulting traffic visualizer is depicted in Figure 4.2. Note that an edge with arrows on both endpoints represents a dual-lane road with the traffic going both in directions.
CHAPTER 4. TRAFFIC SIMULATION

Figure 4.2: Traffic visualizer
4.7 Traffic simulator

The traffic simulator itself takes the following parameters as input:

1) the road network (graph): \( N(I, R) \);
2) the set of legal traffic light configurations for each of the intersections in \( I \): \( LTLC \);
3) maximum simulation time: \( max_t \);
4) minimum distance between vehicles;
5) traffic arrival rate \( \lambda \) (arrival rate parameter to the Poisson distribution), which may also be parametrized as a function of simulation time \( t \): \( \lambda(t) \);
6) a generic vehicle generator, which specifies various properties of each generated vehicle, such as maximum velocity, acceleration, and deceleration rates;
7) traffic light controllers for each traffic intersection which manage the active (i.e. “green”) traffic light configuration state of their corresponding intersection;
8) a number of callbacks to allow external modules to keep track of various events in the simulator.

The traffic simulator works in an update loop, where new vehicle positions and velocities are determined, and traffic light scheduling decisions are made for each sequential \( t \in (0...max_t) \).
Chapter 5

Experimental evaluation

In this chapter, we conduct the experimental evaluation of the traffic light scheduling algorithms described in Section 3 on the generalized traffic simulation framework described in Section 4.

5.1 Motivation & goals

The idea is that we compare the various traditional and data-driven traffic light scheduling algorithms under similar circumstances which are provided by the generalized traffic simulation framework. We measure the differences between the algorithms in terms of various quality attributes described in Section 5.2.

The following research questions described in Section 1.3 are covered by our experimental evaluation:

1. Do data-driven traffic light scheduling algorithms based on deep reinforcement learning provide a significant advantage over the current state-of-the-art reinforcement learning techniques? This research question is covered by Experiments 1, 2, 3, 4, 5, and 6.

2. How can we learn from recurrent contexts (i.e. seasonality in traffic behaviour) in our traffic light scheduling algorithms? This research question is covered by Experiments 2 and 3.

3. How robust are the algorithms described in this research against anomalies in traffic behaviour, i.e. a one-time large event such as a football game which causes extreme congestion for a relatively short period of time? This research question is covered by Experiment 4.

4. How adaptive are the algorithms described in this research to concept drift, i.e. the changing of traffic behaviour in a road network over time? We may ask ourselves whether the performance of the algorithms today is the same as in a few years from now, when the environment has changed due to, for instance, a rise in popularity of a road network. This research question is covered by Experiment 5.

5. Is the computational performance of the algorithms described in this research feasible for real-world applications? Ultimately, in order for our adaptive traffic light algorithms
to be usable in the real world, they should be able to process and adapt to traffic data in real-time. This research question is covered by Experiment 6.

Research questions which are of pure theoretical nature, such as:

1. In what way can machine learning be applied to the traffic light scheduling problem?
2. How can we benchmark different traffic light scheduling algorithms in a generalized framework?

are covered by the theory in Sections 2 and 4, respectively.

5.2 Experimental setup

In this section, we describe all of the parameters for performing the experiments with the various traffic light scheduling algorithms (described in Section 3) on the generalized traffic simulation framework (described in Section 4). Note that exceptions to the parameters defined in this experimental setup are explicitly described in the individual experiments in which the exceptions occur.

Quality attributes

We start off by describing the various quality attributes used to quantify the performance of the traffic light scheduling algorithms. We utilize the following performance quantifiers in our experiments:

1) the average amount of time that vehicles spent waiting during their journey, as per Definition 2.4.11;
2) the average speed, as per Definition 2.4.12;
3) the road network throughput, i.e. the total number of vehicles that have passed through the network or have reached their destination within the network, as per Definition 2.4.10.

Machine

The machine that is used to perform the experiments on has the following specifications:

1) an Intel Core i7-2630QM processor clocked at 2.00 Ghz running on 4 physical cores;
2) 8GB of memory;
3) a 64-bit Windows 10 operating system.

Traffic simulation

The experiments are conducted in a road network and traffic simulation with the following properties:

1. The road network consists of a Manhatten grid of $4 \times 4$ intersections, where each pair of adjacent intersections has a two-way road between them.
2. Each road in the network grid between neighboring intersections is 100 metres long.

3. The maximum allowed speed on each road is 50 km/h.

4. The minimum allowed distance between trailing vehicles is 5 metres.

5. All vehicles have a maximum speed of 120 km/h.

6. All vehicles have an average acceleration of $3.0 \text{ m/s}^2$.

7. All vehicles have an average deceleration of $3.0 \text{ m/s}^2$.

8. Vehicles are generated according to a Poisson distribution with $\lambda = 1.5$, which means that we can expect that three vehicles are generated every two seconds.

9. The traffic will be simulated for a duration of 20,000 seconds. The experiments will be ran with the same generation of random numbers, using a pre-defined random seed. As a result, traffic is generated in exactly the same way in each run of an experiment. This means that not only the arrival of traffic is the same, but also the pre-defined routes of the individual vehicles are the same. Thus, each algorithm will be tested under exactly the same conditions.

Note that the parameters of the simulation are picked in such a way that it accurately models a real-world (congestive) traffic scenario.

**Algorithm: fixed time-based**

We utilize the fixed time-based algorithm with a fixed phase duration of 15 seconds (of green time) for each traffic light configuration.

**Algorithm: Q-learning with reduced state- and action-space**

We utilize the Q-learning with reduced state- and action-space algorithm with the following parameters:

1) traffic queue step size $n = 5$;

2) phase duration step size $m = 5$;

3) maximum phase duration $m_{\text{max}} = 25$;

4) learning rate $\alpha = 0.9$;

5) discount factor $\gamma = 0.9$;

6) $\varepsilon$-greedy parameter $\varepsilon = 0.1$. 

On Deep Reinforcement Learning for Data-Driven Traffic Control
CHAPTER 5. EXPERIMENTAL EVALUATION

Algorithm: Deep Q-learning
We utilize the Deep Q-learning algorithm with the following parameters:

1) phase duration step size $m = 5$;
2) maximum phase duration $m_{\text{max}} = 25$;
3) discount factor $\gamma = 0.9$;
4) $\varepsilon$-greedy parameter $\varepsilon = 0.1$;
5) experience memory size $\delta = 10,000$;
6) experience memory sampling size $\theta = 2000$;
7) the Q-network has two hidden layers both consisting of 20 neurons.

Algorithm: Extended Deep Q-learning
We utilize the Extended Deep Q-learning algorithm with the same parameters used for the Deep Q-learning algorithm, described in Section 5.2.

Algorithm: Pre-emptive Extended Deep Q-learning
We utilize the Pre-emptive Extended Deep Q-learning algorithm with the same parameters used for the Deep Q-learning algorithm, described in Section 5.2.

5.3 Experiments
In this section, we describe the experiments that have been performed to cover the research questions described in Section 5.1.

Experiment 1: algorithm performance under normal circumstances
In this experiment, we simulate all of the traffic light scheduling algorithms under normal circumstances, meaning that we use the experimental setup as is without any additions or modifications. Thus, no recurrent contexts, anomalies, or concept drift takes place in this experiment.

The resulting number of vehicles that are generated by the Poisson distribution with the fixed $\lambda$-parameter as described in the experimental setup in Section 5.2 is visualized in Figure 5.1.

Experiment 2: algorithm adaptation to short-term recurrent contexts
In this experiment, we simulate all of the traffic light scheduling algorithms in a simulation that exposes them to short-term recurrent contexts (seasonality). We simulate this using the parameters of the experimental setup. However, instead of using a fixed $\lambda$-parameter for the Poisson distribution of the vehicle generation process, which essentially controls the level of traffic congestion in the road network, we use a function $\lambda_2(t)$ which takes time $t$ from our simulation time universe $T$ and outputs a $\lambda$-parameter for the Poisson distribution for time
CHAPTER 5. EXPERIMENTAL EVALUATION

Figure 5.1: Number of vehicles generated by Poisson distribution with a fixed \( \lambda \)-parameter in Experiment 1.

\( t \). We select the function in such a way that short-term recurrent contexts occur. This can be achieved by modeling \( \lambda_2(t) \) as a sinusoid with the following parameters:

1) the baseline should be centered around a \( \lambda \)-parameter of 1.0;

2) the amplitude should be 0.5 such that the \( \lambda \)-parameter oscillates between 0.5 and 1.5 (around the baseline of 1.0);

3) the sinusoid should oscillate once every 2000 seconds, meaning that we simulate a number of high-congestion and low-congestion periods during the experiment.

Hence, the function \( \lambda_2(t) \) is defined as follows:

\[
\lambda_2(t) = 1.0 + \sin\left(\frac{\pi}{1000}t\right).
\] (5.1)

The resulting number of vehicles that are generated by the Poisson distribution with the \( \lambda \)-parameter as described in Equation 5.1 is visualized in Figure 5.2.

**Experiment 3: algorithm adaptation to long-term recurrent contexts**

In this experiment, instead of measuring the adaptation of the algorithms to short-term recurrent contexts as we do in Experiment 2, we want to determine how the algorithms adapt to recurrent contexts that occur on the longer term. Hence, we use the same sinusoid as described in Experiment 2, except for using a different oscillation period: the sinusoid should now oscillate twice as long, meaning that its oscillation time is now \( 2000 \times 2 = 4000 \) seconds. Hence, the function \( \lambda_3(t) \) is defined as follows:
Figure 5.2: Number of vehicles generated by Poisson distribution described by $\lambda$-parameter in Equation 5.1. The $\lambda$-parameter follows a sinusoid to generate recurrent contexts.

$$\lambda_3(t) = 1.0 + \sin\left(\frac{\pi}{2000}t\right).$$  \hfill (5.2)

The resulting number of vehicles that are generated by the Poisson distribution with the $\lambda$-parameter as described in Equation 5.2 is visualized in Figure 5.3.

**Experiment 4: algorithm robustness to anomalies**

In this experiment, we want to measure the robustness of the algorithms to anomalies in traffic behaviour. We want determine how the algorithms deal with unique situations, i.e. the situation of a football match that brings the traffic congestion level to a temporary new high. As we do in Experiments 2 and 3, we introduce a new $\lambda_4(t)$ function to model this temporary anomaly of 4000 seconds:

$$\lambda_4(t) = \begin{cases} 2.0 & \text{if } t \geq 8000 \land t < 12000 \\ 1.0 & \text{otherwise.} \end{cases}$$  \hfill (5.3)

The resulting number of vehicles that are generated by the Poisson distribution with the $\lambda$-parameter as described in Equation 5.3 is visualized in Figure 5.4.

**Experiment 5: algorithm adaptation to concept drift**

In this experiment, we want to measure the adaptation of the algorithms to concept drift in traffic behaviour. We want determine how the algorithms deal with long-term changes in circumstances, which change the way traffic behaves in the road network, i.e. because of structural changes to the road network which make a road more popular over time. As we do in Experiments 2, 3, and 4, we introduce a new $\lambda_5(t)$ function to model this concept drift:
Figure 5.3: Number of vehicles generated by Poisson distribution described by $\lambda$-parameter in Equation 5.2. The $\lambda$-parameter follows a sinusoid to generate recurrent contexts.

\[
\lambda_5(t) = \begin{cases} 
1.0 + \sin\left(\frac{\pi}{500}t\right) & \text{if } t \geq 0 \land t < 10000 \\
1.5 + \sin\left(\frac{\pi}{1000}t\right) & \text{otherwise.}
\end{cases}
\]  

(5.4)

Figure 5.4: Number of vehicles generated by Poisson distribution described by $\lambda$-parameter in Equation 5.3. The $\lambda$-parameter is increased during $t = 8000$ and $t = 12000$ to generate an anomaly.

\[
\lambda_5(t) = \begin{cases} 
1.0 + \sin\left(\frac{\pi}{500}t\right) & \text{if } t \geq 0 \land t < 10000 \\
1.5 + \sin\left(\frac{\pi}{1000}t\right) & \text{otherwise.}
\end{cases}
\]  

(5.4)
The resulting number of vehicles that are generated by the Poisson distribution with the \( \lambda \)-parameter as described in Equation 5.4 is visualized in Figure 5.5.

Experiment 6: algorithm computational performance

In this experiment, we want to measure the computational performance of the traffic light scheduling algorithms by measuring the running time of each of the decision-making and training cycles performed by the algorithms. We simulate all of the algorithms under normal circumstances, meaning that we use the experimental setup as is with one modification: the simulation time is extended to 40,000 seconds. We do this because we want to create the situation in which the algorithms reach their maximum number of training data points which they allow to keep in-memory. By doing this, we can test the algorithms to their respective limits.

In this experiment, we measure the average amount of time that a scheduling action takes. We do not measure memory usage, as we have already shown in Section 3 that this is negligible in real-world situations because the memory usage is explicitly upper bounded by the algorithm parameters.
5.4 Results

Experiment 1: algorithm performance under normal circumstances

The results of the average vehicle waiting times for which the algorithms are responsible are depicted in Figure 5.6. As can be concluded from these results, the algorithms based on Deep Q-learning perform best in terms of average waiting times, decreasing the average waiting times by $58 - 86\%$ ($76\%$ on average) and $29 - 77\%$ ($59\%$ on average), when compared to the fixed time-based and traditional Q-learning algorithms, respectively.

The results of the average speed of the vehicles traveling through the road network for which the algorithms are responsible are depicted in Figure A.2 in Appendix A. We can see that these results are a direct reflection of our results on the average vehicle waiting times: algorithms based on Deep Q-learning outperform algorithms which are not based on Deep Q-learning.

The results of the average road network throughput for which the algorithms are responsible are depicted in Figure A.3 in Appendix A. From these results we can safely conclude that by utilizing the algorithms based on Deep Q-learning, we obtain a significant increase in road network throughput, increasing the network throughput by up to $16\%$ in comparison to the road network throughput of the fixed time-based algorithm. However, we can also see that the algorithms based on Deep Q-learning outperform the traditional Q-learning algorithm in terms of road network throughput, increasing the network throughput by up to $15\%$.

Experiment 2: algorithm adaptation to short-term recurrent contexts

The results of the average vehicle waiting times for which the algorithms are responsible are depicted in Figure 5.7. From these results we can see that the algorithms based on (Deep) Q-learning exhibit, in fact, learning behaviour. The more they are exposed to the recurrence, the more they are able to decrease the average waiting time. Especially the algorithms based on Deep Q-learning, have a much higher performance than the other algorithms, decreasing the average waiting time after 1000 time units by $52 - 88\%$ ($76\%$ on average) and $5 - 77\%$ ($58\%$ on average), when compared to the fixed time-based and traditional Q-learning algorithms, respectively. Not only do the Deep Q-learning algorithms perform better, but they also learn at a much faster rate than the traditional Q-learning algorithm, as the decline in average waiting times is much steeper for the Deep Q-learning algorithms.

The results of the average speed of the vehicles traveling through the road network for which the algorithms are responsible are depicted in Figure B.2 in Appendix B. We can see that these results are a direct reflection of our results on the average vehicle waiting times. No new insights can be gained from these results.

The results of the average road network throughput for which the algorithms are responsible are depicted in Figure B.3 in Appendix B. From these results we can see that the algorithms based on (Deep) Q-learning outperform the fixed time-based algorithm most of the time. This is also the case for the Deep Q-learning algorithms, which outperform the traditional Q-learning algorithm most of the time. A rather interesting observation is that in some situations there is no “clear” winner in terms of road network throughput. However, it is
Figure 5.6: Average amount of time that vehicles spent waiting during Experiment 1. The λ-parameter of the Poisson distribution that controls the vehicle arrival process is fixed, as shown in Figure 5.1.
Figure 5.7: Average amount of time that vehicles spent waiting during Experiment 2. The recurrence is generated by the Poisson distribution that controls the vehicle arrival process with the $\lambda$-parameter following a sinusoid, as shown in Figure 5.2.
important to keep in mind that we are dealing with a complex system: scheduling decisions we make on one end of the road network, ultimately affect the traffic flow on the other end of the network. This effect is strengthened in highly congestive situations and large fluctuations in traffic arrival rates, which we are primarily focusing on in these experiments. Because of the large fluctuations in traffic arrival rates, the system becomes so complex that maximization of the individual intersection throughput by distinct traffic light controllers does not yield maximization of the road network throughput as a whole. Thus, without communication between traffic light controllers, it is unlikely that we achieve a global maximization of the road network throughput. Future work could benefit from researching the possibility to allow traffic light controllers to “co-learn” using Deep Q-learning technology.

Experiment 3: algorithm adaptation to long-term recurrent contexts

The results of the average vehicle waiting times for which the algorithms are responsible are depicted in Figure 5.8. Similarly to the results of Experiment 2, we see that all of the algorithms based on (Deep) Q-learning exhibit learning behaviour. Also, similarly to the results in Experiment 2, algorithms based on Deep Q-learning learn at a much faster rate than the traditional Q-learning algorithm, decreasing the average waiting time after 1000 time units by $50 - 87\%$ (72\% on average) and $11 - 76\%$ (57\% on average), when compared to the fixed time-based and traditional Q-learning algorithms, respectively.

The results of the average speed of the vehicles traveling through the road network for which the algorithms are responsible are depicted in Figure C.2 in Appendix C. Similarly to previous experiments, these results directly reflect the observations that we made on our results of the average vehicle waiting times. No new insights can be gained from these results.

The results of the average road network throughput for which the algorithms are responsible are depicted in Figure C.3 in Appendix C. The same reasoning applies to these results as the reasoning applied to the results on the road network throughput in Experiment 2.

Experiment 4: algorithm robustness to anomalies

The results of the average vehicle waiting times for which the algorithms are responsible are depicted in Figure 5.9. As we can clearly see from these results, the algorithms based on Deep Q-learning perform best in terms of average waiting times, decreasing the average waiting times during the anomaly by $45 - 65\%$ (58\% on average) and $29 - 64\%$ (48\% on average), when compared to the fixed time-based and traditional Q-learning algorithms, respectively. A reason explaining why the Deep Q-learning algorithms perform better than the traditional Q-learning algorithm is that the Deep Q-learning algorithms recognize patterns within the anomaly that they encountered before the anomaly. They are able to take advantage of the experience obtained before the anomaly, by applying that experience within the anomaly, using pattern recognition that is facilitated by the Q-network. As we can see from the results, the Deep Q-learning algorithms are able to respond immediately to the anomaly, unlike the traditional Q-learning algorithm, which has to completely retrain itself based on the new situations (traffic states) it encounters.

The results of the average speed of the vehicles traveling through the road network for which
CHAPTER 5. EXPERIMENTAL EVALUATION

Figure 5.8: Average amount of time that vehicles spent waiting during Experiment 3. The recurrence is generated by the Poisson distribution that controls the vehicle arrival process with the $\lambda$-parameter following a sinusoid, as shown in Figure 5.3.
Figure 5.9: Average amount of time that vehicles spent waiting during Experiment 4. The anomaly is generated by the Poisson distribution that controls the vehicle arrival process with a higher $\lambda$-parameter during $8000 \leq t < 12000$, as shown in Figure 5.4.
the algorithms are responsible are depicted in Figure D.2 in Appendix D. Similarly to previous experiments, these results directly reflect the observations that we made on our results of the average vehicle waiting times. No new insights can be gained from these results.

The results of the average road network throughput for which the algorithms are responsible are depicted in Figure D.3 in Appendix D. From these results we can also clearly see that, during the anomaly, the algorithms based on Deep Q-learning algorithm outperform the other algorithms in terms of road network throughput, increasing the throughput by up to 21% (14% on average) and 17% (11% on average) in comparison to the fixed time-based and traditional Q-learning algorithms, respectively.

Experiment 5: algorithm adaptation to concept drift

The results of the average vehicle waiting times for which the algorithms are responsible are depicted in Figure 5.10. From these results we can see that there are no dramatic performance shifts between the various (Deep) Q-learning algorithms in terms of average vehicle waiting times after the drift in traffic behaviour occurs. We do see that the algorithms need some time to adapt to the changed circumstances after the concept drift occurs, which is natural, as this is inherent to reinforcement learning. The relative performance is affected, but not in any dramatic sense: the slight shift in relative performance is easily explained by the effectiveness of the individual algorithms in more highly congestive situations. Before the concept drift takes place ($t < 10000$), the algorithms based on Deep Q-learning decrease the average vehicle waiting times by $40 - 80\%$ ($71\%$ on average) and $10 - 72\%$ ($53\%$ on average), when compared to the fixed time-based and traditional Q-learning algorithms, respectively. After the concept drift takes place ($t \geq 10000$), the algorithms based on Deep Q-learning decrease the average vehicle waiting times by $50 - 84\%$ ($69\%$ on average) and $13 - 77\%$ ($51\%$ on average), when compared to the fixed time-based and traditional Q-learning algorithms, respectively. As we can see, the difference in performance gain before the concept drift takes place, is very similar to the performance gain after the concept drift takes place.

The results of the average speed of the vehicles traveling through the road network for which the algorithms are responsible are depicted in Figure E.2 in Appendix E. As can be seen more clearly in these results, we immediately see that the Deep Q-learning algorithms adapt the quickest to the drift in traffic behaviour, giving us the highest average speed of all of the algorithms.

The results of the average road network throughput for which the algorithms are responsible are depicted in Figure E.3 in Appendix E. Before the concept drift takes place ($t < 10000$), the algorithms based on Deep Q-learning increase the average road network throughput by up to 7% (2% on average) and 6% (2% on average), when compared to the fixed time-based and traditional Q-learning algorithms, respectively. After the concept drift takes place ($t \geq 10000$), the algorithms based on Deep Q-learning increase the average road network throughput by up to 19% (9% on average) and 12% (6% on average), when compared to the fixed time-based and traditional Q-learning algorithms, respectively. We can see that the traditional Q-learning algorithm adapts poorly to concept drift, while the Deep Q-learning algorithms are able to quickly respond to the concept drift by applying the experience gained before the drift.
Figure 5.10: Average amount of time that vehicles spent waiting during Experiment 5. After $t = 10000$, the $\lambda$-parameter of the Poisson distribution that controls the vehicle arrival process follows a different sinusoid than during $t < 10000$, as shown in Figure 5.5.
Experiment 6: algorithm computational performance

The results of the average scheduling computing time for which the algorithms are responsible are depicted in Figure 5.11. From these results, we can clearly see that the traditional Q-learning and fixed time-based algorithms have the lowest computing times (≈ 0ms). This is expected, as their corresponding time complexities are negligible as well. From these results, we can also see that the scheduling computing time of the Deep Q-learning algorithms grow linearly with respect to the amount of training data that they accumulate. However, as the amount of training data that is used to re-train the Q-networks is upper bounded by a constant ($\theta$, see Section 5.2), which has been reached after 20,000 seconds, the computing time stabilizes to around 4 to 5 milliseconds. This is still negligible in the real-world, and thus acceptable for real-world applications.
Figure 5.11: Average scheduling computing time in milliseconds during Experiment 6 (both the fixed time-based and the traditional Q-learning algorithms have negligible computing times). The $\lambda$-parameter of the Poisson distribution that controls the vehicle arrival process is fixed, as shown in Figure 5.1.
Chapter 6

Conclusions

In this section, we first demonstrate the main contributions of this research. After which, we answer each research question with evidence gained from this research. Finally, we discuss several opportunities that are of interest for future work.

6.1 Main contributions

This research advanced the current state-of-the-art in deep reinforcement learning and traffic light scheduling further, by making the following contributions:

1. We demonstrated that deep reinforcement learning, and in particular deep Q-learning, can be effectively implemented using classical feed-forward ANNs.

2. As a result of the implementation of deep reinforcement learning with a classical neural network, our reinforcement learning agent can now perform pattern recognition solely on useful information, i.e. only the information (without noise) that is important for the agent to reason about. This is in contrast to the current state-of-the-art research in deep reinforcement learning, which currently only focuses on the employment of convolutional neural networks that are specialized in pattern recognition in images [5]. Thus, we can now train our reinforcement learning agent to recognize patterns in non-visual data, which opens up the application of deep reinforcement learning in a vast amount of other problem domains, such as traffic light scheduling in our case.

3. We demonstrated, using a generalized traffic simulation framework, that traffic light scheduling algorithms based on deep reinforcement learning converge faster, learn better from seasonality, are more robust to special situations, and last but not least, perform better than both traditional timing-based approaches used in the real-world, as previously studied state-of-the-art approaches in traffic light scheduling that utilize traditional reinforcement learning.

4. By utilizing deep reinforcement learning in traffic light scheduling under regular circumstances, we were able to cut average waiting times by 58 – 86% (76% on average) when compared to a real-world approach and by 29 – 77% (59% on average) when compared to a traditional reinforcement learning approach.

On Deep Reinforcement Learning for Data-Driven Traffic Control
5. Also, under these same circumstances, we were able to increase the average road network throughput by up to 16% when compared to a real-world approach and by up to 15% when compared to a traditional reinforcement learning approach. These numbers have potential to grow, since we currently only maximize the throughput of each individual traffic intersection, while a form of (deep) co-learning could allow for maximization of the road network throughput as a whole.

6. The algorithms are designed and implemented in a generalized way such that they are easy to deploy in real-world, but also simulated contexts.

6.2 Research questions

Our main research question is formulated as follows: do data-driven traffic light scheduling algorithms based on deep reinforcement learning provide a significant advantage over the current state-of-the-art reinforcement learning techniques?

The answer is given in terms of a number of more detailed research questions, each covering an important aspect of the main research question.

**In what way can machine learning be applied to the traffic light scheduling problem?**

We concluded in Section 2 that the best way to apply machine learning in the traffic light scheduling problem is to consider a form of reinforcement learning for which data is generated in an online manner, since we do not have data sets with optimal traffic light schedules a priori. Because traditional approaches to reinforcement learning, such as Q-learning, are hindered by their poor ability to detect patterns and similarities between states, the idea is that we combine artificial neural networks with reinforcement learning: a technique called deep reinforcement learning.

**How can we benchmark different traffic light scheduling algorithms in a generalized framework?**

In Section 4, we presented a detailed and comprehensive overview of a generalized traffic simulation framework that is:

1) *generic*, meaning that we can use it within any context that we can think of, as we did in our experimental evaluation in Section 5;

2) *realistic*, such that we can benchmark our traffic light scheduling algorithms under credible circumstances, as we did in our experimental evaluation in Section 5;

3) *extensible*, such that we can extend functionality of the simulator for various external applications, i.e. to visualize traffic flows or to input data from the real-world.

The simulation framework is:

1) *generic*, because of the decoupling and generalization of the various aspects of the simulation, depicted in Figure 4.1;
2) **realistic**, because of the detailed manner in which we simulate each vehicle separately, including acceleration, deceleration, adhering to minimal distances, etc.;

3) **extensible**, because of the possibility to tap into one of the framework’s many data streams.

**How can we learn from recurrent contexts (i.e. seasonality in traffic behaviour) in our traffic light scheduling algorithms?**

In our experimental evaluation in Section 5, Experiments 2 and 3 showed us that Deep Q-learning algorithms perform much better than traditional timing-based and Q-learning algorithms when exposed to recurrent contexts, decreasing the average vehicle waiting times in comparison to both algorithms after a small amount of learning time by $50 - 87\%$ (72\% on average) and $11 - 76\%$ (57\% on average), respectively. This shows that Deep Q-learning algorithms are much better at detecting similarities and patterns between traffic states than algorithms based on traditional Q-learning, and thus, can learn better from recurrent contexts.

**How robust are the algorithms described in this research against anomalies in traffic behaviour?**

In our experimental evaluation in Section 5, Experiment 4 showed us that Deep Q-learning algorithms are much more robust to anomalies than traditional timing-based and Q-learning algorithms. During the anomaly, the algorithms based on Deep Q-learning decrease the average vehicle waiting times by $45 - 65\%$ (58\% on average) and $29 - 64\%$ (48\% on average), when compared to traditional timing-based and Q-learning algorithms, respectively. Similarly, during the anomaly, our Deep Q-learning algorithms were able to increase the average road network throughput by up to $21\%$ (14\% on average) and $17\%$ (11\% on average) when compared to traditional timing-based and Q-learning algorithms, respectively. The results point out that the Deep Q-learning algorithms are able to take advantage of the patterns learned outside of an anomaly, by utilizing pattern recognition facilitated by the Q-network within the anomaly. This allows the Deep Q-learning algorithms to respond much quicker to an anomaly than a traditional Q-learning algorithm, as the traditional Q-learning algorithm has to completely retrain itself based on the new situations (traffic states) it encounters during the anomaly.

**How adaptive are the algorithms described in this research to concept drift, i.e. the changing of traffic behaviour in a road network over time?**

In our experimental evaluation in Section 5, Experiment 5 showed us that the traditional Q-learning algorithm adapts poorly to concept drift, while the Deep Q-learning algorithms are able to quickly respond to the concept drift by applying the experience gained before the drift in traffic behaviour. Because of this, the Deep Q-learning algorithms are able to adapt and respond much quicker to the concept drift than the traditional Q-learning algorithm.
Is the computational performance of the algorithms described in this research feasible for real-world applications?

The theoretical analysis of the memory requirements of the algorithms in Section 3 pointed out that the memory requirements are definitely acceptable for real-world applications. Similarly, the theoretical analysis of the computing time requirements of the algorithms in Section 3 and the verification of those requirements in Experiment 6 in our experimental evaluation of Section 5 pointed out that the computing time requirements of the algorithms are also acceptable for real-world applications. Thus, the computational performance as a whole for the algorithms described in this research are feasible for real-world applications.

Finally, do data-driven traffic light scheduling algorithms based on deep reinforcement learning provide a significant advantage over the current state-of-the-art reinforcement learning techniques?

Given the fact that Deep Q-learning algorithms outperform the current state-of-the-art Q-learning approach to the traffic light scheduling problem in terms of convergence, learning from recurrent contexts, robustness against anomalies, adaptation to concept drift, and from the fact that the computational performance of our Deep Q-learning algorithms is still feasible for real-world applications, we can safely conclude that traffic light scheduling algorithms based on deep reinforcement learning provide a significant advantage over the current state-of-the-art reinforcement learning techniques.

6.3 Future work

The traffic light scheduling algorithms described in this research currently only attempt to maximize the throughput of their own traffic intersection. However, as mentioned in Experiment 2 in our experimental evaluation in Section 5, traffic flows in a road network are of complex nature: scheduling decisions made on one end of the road network influence traffic flows on the other end of the network. Thus, maximizing the throughput of each individual traffic intersection may not yield a maximization of the throughput of the road network as a whole. The next step in research could be to investigate the opportunities to combine deep reinforcement with multi-agent reinforcement learning techniques, to facilitate a form of co-learning.
Bibliography


Appendix A

Results of Experiment 1

In this appendix, we present all of the results of Experiment 1. Note that some results have already appeared in Section 5, but are presented here again for completeness and accessibility of the results.

See the next pages for each of the results.
Figure A.1: Average amount of time that vehicles spent waiting during Experiment 1. The $\lambda$-parameter of the Poisson distribution that controls the vehicle arrival process is fixed, as shown in Figure 5.1.
Figure A.2: Average speed of vehicles during Experiment 1. The $\lambda$-parameter of the Poisson distribution that controls the vehicle arrival process is fixed, as shown in Figure 5.1.
APPENDIX A. RESULTS OF EXPERIMENT 1

Figure A.3: Average throughput in number of vehicles that finished their journey during Experiment 1. The $\lambda$-parameter of the Poisson distribution that controls the vehicle arrival process is fixed, as shown in Figure 5.1.
Appendix B

Results of Experiment 2

In this appendix, we present all of the results of Experiment 2. Note that some results have already appeared in Section 5, but are presented here again for completeness and accessibility of the results.

See the next pages for each of the results.
Figure B.1: Average amount of time that vehicles spent waiting during Experiment 2. The recurrence is generated by the Poisson distribution that controls the vehicle arrival process with the $\lambda$-parameter following a sinusoid, as shown in Figure 5.2.
Figure B.2: Average speed of vehicles during Experiment 2. The recurrence is generated by the Poisson distribution that controls the vehicle arrival process with the $\lambda$-parameter following a sinusoid, as shown in Figure 5.2.
Figure B.3: Average throughput in number of vehicles that finished their journey during Experiment 2. The recurrence is generated by the Poisson distribution that controls the vehicle arrival process with the $\lambda$-parameter following a sinusoid, as shown in Figure 5.2.
Appendix C

Results of Experiment 3

In this appendix, we present all of the results of Experiment 3. Note that some results have already appeared in Section 5, but are presented here again for completeness and accessibility of the results.

See the next pages for each of the results.
Figure C.1: Average amount of time that vehicles spent waiting during Experiment 3. The recurrence is generated by the Poisson distribution that controls the vehicle arrival process with the $\lambda$-parameter following a sinusoid, as shown in Figure 5.3.
Figure C.2: Average speed of vehicles during Experiment 3. The recurrence is generated by the Poisson distribution that controls the vehicle arrival process with the $\lambda$-parameter following a sinusoid, as shown in Figure 5.3.
Figure C.3: Average throughput in number of vehicles that finished their journey during Experiment 3. The recurrence is generated by the Poisson distribution that controls the vehicle arrival process with the $\lambda$-parameter following a sinusoid, as shown in Figure 5.3.
Appendix D

Results of Experiment 4

In this appendix, we present all of the results of Experiment 4. Note that some results have already appeared in Section 5, but are presented here again for completeness and accessibility of the results.

See the next pages for each of the results.
Figure D.1: Average amount of time that vehicles spent waiting during Experiment 4. The anomaly is generated by the Poisson distribution that controls the vehicle arrival process with a higher \( \lambda \)-parameter during \( 8000 \leq t < 12000 \), as shown in Figure 5.4.
Figure D.2: Average speed of vehicles during Experiment 4. The anomaly is generated by the Poisson distribution that controls the vehicle arrival process with a higher \( \lambda \)-parameter during \( 8000 \leq t < 12000 \), as shown in Figure 5.4.
Figure D.3: Average throughput in number of vehicles that finished their journey during Experiment 4. The anomaly is generated by the Poisson distribution that controls the vehicle arrival process with a higher $\lambda$-parameter during $8000 \leq t < 12000$, as shown in Figure 5.4.
Appendix E

Results of Experiment 5

In this appendix, we present all of the results of Experiment 5. Note that some results have already appeared in Section 5, but are presented here again for completeness and accessibility of the results.

See the next pages for each of the results.
Figure E.1: Average amount of time that vehicles spent waiting during Experiment 5. After $t = 10000$, the $\lambda$-parameter of the Poisson distribution that controls the vehicle arrival process follows a different sinusoid than during $t < 10000$, as shown in Figure 5.5.
Figure E.2: Average speed of vehicles during Experiment 5. After $t = 10000$, the $\lambda$-parameter of the Poisson distribution that controls the vehicle arrival process follows a different sinusoid than during $t < 10000$, as shown in Figure 5.5.
Figure E.3: Average throughput in number of vehicles that finished their journey during Experiment 5. After $t = 10000$, the $\lambda$-parameter of the Poisson distribution that controls the vehicle arrival process follows a different sinusoid than during $t < 10000$, as shown in Figure 5.5.
Appendix F

Results of Experiment 6

In this appendix, we present all of the results of Experiment 6. Note that some results have already appeared in Section 5, but are presented here again for completeness and accessibility of the results.

See the next page for the result.
Figure F.1: Average scheduling computing time in milliseconds during Experiment 6 (both the fixed time-based and the traditional Q-learning algorithms have negligible computing times). The $\lambda$-parameter of the Poisson distribution that controls the vehicle arrival process is fixed, as shown in Figure 5.1.