Otto-von-Guericke University Magdeburg

Department of Computer Science
Institute for Intelligent Cooperating Systems

Master Thesis

Domain Randomization of Deep Reinforcement Learning Environments for Zero-Shot Traffic Signal Control

Author:
Frank Dreyer

December 19, 2023

Advisers:

Supervisor
Prof. Dr.-Ing. Sebastian Stober

Department of Computer Science
Otto-von-Guericke University
Universitätsplatz 2
39106 Magdeburg, Germany

Supervisor
Dr.-Ing. Tom Assmann

Department of Mechanical Engineering
Otto-von-Guericke University
Universitätsplatz 2
39106 Magdeburg, Germany

Dreyer, Frank:
Domain Randomization of Deep Reinforcement Learning Environments for Zero-Shot Traffic Signal Control
Master Thesis, Otto-von-Guericke University
Magdeburg, 2023.

Abstract

The escalating traffic demand in urban areas necessitates intelligent transportation engineering solutions that can regulate traffic more efficiently. The optimization of traffic signal control is considered a crucial factor in this regard. In recent years, deep reinforcement learning has gained traction as a potential solution concept to learn an adaptive signal controller, with promising initial results demonstrating its superiority over conventional solutions. However, the practical utility of such solutions is often constrained by a rigid model representation for the signal controller and a limited representation of traffic scenarios during the training phase. As a result, these methods severely overfit to the idiosyncrasies of the traffic environment used during training and fail to adapt to changing traffic conditions. To enhance adaptability, this thesis introduces TransferLight, a model architecture that employs neural message passing on a graph-structured state representation of an intersection to predict its next phase. This approach facilitates adaptability to arbitrarily structured intersections and is implemented for both a Q-learning and an advantage actor-critic algorithm. Additionally, a domain randomization mechanism is proposed, perturbing specific parameters associated with the road network and traffic within the environment during each training episode. This deliberate randomization aims to enhance the variability of traffic conditions encountered during the training phase. The efficacy of the proposed techniques is assessed by comparing them to established baseline solutions derived from conventional transportation engineering theories and reinforcement learning on both synthetically generated and real-world test scenarios. The findings reveal that TransferLight exhibits pronounced overfitting when trained on identical scenarios in each episode. Conversely, when the proposed domain randomization mechanism is leveraged to train TransferLight, generalization markedly improves and enables a zero-shot transfer to the test scenarios with a more consistent and robust performance, often being superior to the benchmark solutions, with respect to average travel time and throughput. However, upon closer examination within the context of an arterial road network scenario, the behavior of the trained model does not consistently align well with solutions specifically calibrated for this scenario. The latter exhibit improved signal progression, necessitating fewer stops for vehicles to traverse the arterial roadway. An estimation of model uncertainty using Monte Carlo dropout further indicates that despite the application of domain randomization during training, TransferLight's confidence in its predictions is not consistently high across traffic conditions, as the uncertainty distribution for some scenarios show a noticeable shift or spread towards higher uncertainties. These observations imply that although the proposed domain randomization mechanism enhances the transfer performance of TransferLight to a wide range of traffic conditions, it does not implicitly equip the model with the necessary knowledge to act optimally and with a high degree of certainty on a consistent basis.

Acknowledgements

I extend my gratitude to my advisor, Johann Schmidt, whose guidance and support have been instrumental throughout this research journey. His openness to my ideas and suggestions have played a crucial role in shaping this work.

I am deeply thankful to my family and friends for being a constant source of encouragement and inspiration. Their unwavering support has fueled my determination and ambition. A special acknowledgment goes to my parents for their support, both emotionally and financially.

I would also like to express my gratitude to the team of the Artificial Intelligence Lab of the Otto-von-Guericke University for providing me with the resources necessary to train all models in this work and for the exceptional education they provide, making subjects like deep learning not only informative but also enjoyable, easily understandable, highly practical, and encouraging to dig deeper.

This work would not have been possible without the collective support and encouragement of these individuals. Thank you for being a part of this journey.

Contents

1 Introduction **1**
1.1 Motivation 1
1.2 Aim of Thesis 3
1.3 Outline 4

2 Background **5**
2.1 Traffic Signal Control 5
2.1.1 Operation Principles of Signalized Intersections 5
2.1.2 Objectives and Performance Metrics 9
2.1.3 Webster Method 12
2.1.4 GreenWave 13
2.1.5 Max Pressure 14
2.2 Deep Learning 15
2.2.1 Artificial Neural Networks 15
2.2.2 Fully Connected Layers 17
2.2.3 Neural Message Passing Layers 18
2.2.4 Training Artificial Neural Networks 22
2.2.5 Monte Carlo Dropout 24
2.3 Reinforcement Learning 25
2.3.1 Markov Decision Process 26
2.3.2 Value-based Methods 29
2.3.3 Policy Gradient Methods 33
2.3.4 Cooperative Markov Game 36
2.3.5 Independent Learning 37
2.3.6 Domain Randomization 37

3 Related Work **39**
3.1 Transportation Engineering Solutions 39
3.2 Reinforcement Learning Solutions 40

4 Method **45**
4.1 Reinforcement Learning Problem Formulation 45
4.1.1 Reward Function 46
4.1.2 State Representation 46

4.1.3 Action Definition 49
4.2 Neural Network Architecture: TransferLight 50
4.2.1 Graph Embedding Module 50
4.2.2 Deep Q-Network 52
4.2.3 Actor-Critic Network 53
4.3 Training 53
4.3.1 Independent Q-Learning Algorithm 53
4.3.2 Independent Advantage Actor-Critic Algorithm 56
4.4 Domain Randomization Mechanism 57
4.4.1 Road Network 58
4.4.2 Traffic 59
4.5 Implementation Details 61

5 Evaluation **63**
5.1 Research Questions 63
5.2 Evaluation Framework 64
5.2.1 Test Scenarios 65
5.2.2 Compared Methods 68
5.2.3 Evaluation Metrics 69
5.3 Evaluation Results 70
5.3.1 Performance Evolution during Training 71
5.3.2 Zero-Shot Transfer Performance 72
5.3.3 Signal Progression 76
5.3.4 Model Uncertainty 78

6 Conclusion **83**
6.1 Summary 83
6.2 Limitations 84
6.3 Future Work 85
6.4 Implications 87

A Appendix **89**
A.1 Traffic Congestion in the `random-heavy` Scenario 89

B Abbreviations **93**

C List of Figures **95**

D List of Tables **97**

E List of Algorithms **99**

F Bibliography **101**

1

Introduction

1.1 Motivation

In the face of population growth and urbanization, urban transport infrastructures are increasingly reaching their limits when it comes to coping with rising traffic demand. As a result, traffic congestion in cities around the world is becoming an ever-increasing problem that is detrimental in many ways. For example, commuters are losing more and more of their valuable time to traffic congestion. In 2022, the average driver in the United Kingdom, the United States, and Germany wasted 80, 51, and 40 hours in traffic jams, respectively [1]. These extended travel times could not only be better used for other, more important things in life, but also cause stress and annoyance for drivers [2]. In addition, congested roads have been shown to significantly increase emissions of harmful pollutants like carbon monoxide, particulate matter, and nitrogen oxides [3]. Such traffic-related pollutants have been linked to various health outcomes such as cardiovascular and respiratory diseases, as well as cancer [4], with an estimated public health cost of $17 billion in 2030 from traffic congestion alone [5]. Similarly, it is evident that carbon dioxide emissions increase significantly as a result of traffic congestion, primarily due to frequent and heavy acceleration in stop-and-go traffic, which contributes substantially to global warming [6]. Lastly, the impact on the economy should not be forgotten. In 2022, delayed movement of goods and services due to traffic congestion cost the United States $81 billion, the United Kingdom £9.5 billion, and Germany €3.9 billion [1]. All of these impacts on society, health, the environment, and the economy underscore the need to improve transportation conditions in inner cities to enable healthier, more sustainable, and more prosperous lives.

To alleviate congestion on urban streets, one approach is to enhance *Traffic Signal Control (TSC)*. Properly optimized signalized intersections can substantially mitigate congestion. However, many conventional solutions implemented today fall short of reaching optimality. They rely on simplified traffic models, whose assumptions do not accurately capture the dynamics of real traffic. Recently, *Reinforcement Learning (RL)* has gained prominence in the TSC research community as it does not require simplifying assumptions about the traffic environment, but provides a way to learn a signal control policy from the feedback provided by the real traffic environment while interacting with it. Several independent papers have already shown that a signal control policy learned in this way often performs significantly better than a conventional solution in certain simulated scenarios, especially when *Deep Reinforcement Learning (DRL)* is applied, where the control policy is represented by an *Artificial Neural Network (ANN)* allowing control policies to be learned over rich but high-dimensional sensory inputs [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17].

Despite the potential of RL in TSC, most of the existing solutions fail when the traffic conditions under which they were trained change. This is partly due to an inappropriate choice of the model for the signal control policy. Most works use a model whose applicability is limited to a specific intersection type [18, 19, 20, 21, 7, 22, 10, 9, 23, 14, 24, 25, 17]. Consequently, a new model must be trained for each intersection type, which is very inefficient and costly. Apart from that, signal control policies are often trained too onesidedly. In most cases, researchers train with a simulated but fixed traffic environment, naively assuming that traffic conditions do not change [20, 26, 27, 28, 21, 8, 9, 23, 10, 14, 24, 25, 16, 17]. In reality, however, traffic is inherently stochastic and influenced by a variety of random factors such as individual driving behavior, time of day (e.g., peak time vs. off-peak time), day of week (week day vs. weekend traffic), weather conditions, and even the occurrence of sudden events (e.g., accidents, road closures, concerts, etc.). All of these factors can cause traffic to behave in unpredictable ways. If this randomness is not adequately accounted for during training, it has been shown that the learned policy will overfit the training environment and experience a severe deterioration in performance when it encounters a previously unseen traffic situation [11, 29]. As a result of all these aspects, such solutions have to be retrained again and again as soon as conditions change, which makes their use cumbersome in practice.

Ideally, a TSC policy, once trained, should enable *zero-shot transfer*, allowing it to seamlessly adapt to any traffic situation without the necessity for additional training to achieve the desired performance. A popular approach in DRL to achieve zero-shot transferability is *Domain Randomization (DR)* [30]. The idea is to randomly change certain domain parameters of the environment within predefined ranges in each training episode, in the hope that the increased variability will make the policy more robust and encourage it to generalize across different environments, including realistic conditions. Since its inception, this training paradigm has found application in various fields, including robotics [30, 31] and autonomous driving [32, 33], and has consistently shown its efficacy, enabling learned policies to be directly applied in the physical world without experiencing severe performance degradation, despite being exclusively trained in a simulator. These successes raise the question of whether DR could also be beneficial for training a TSC policy, thereby better preparing it for the stochastic nature of real-world traffic.

1.2 Aim of Thesis

The aim of this thesis is to investigate the effectiveness of DR within the realm of TSC. The primary focus is on assessing the extent to which a signal control policy, trained using this method, can successfully transfer its learned knowledge to traffic scenarios that were not present during the training phase. To achieve this goal, a DR mechanism is developed, designed to introduce variability into the traffic conditions of the environment in each training episode. Additionally, a model for the signal control policy of an intersection, named TransferLight, is proposed, aimed at enhancing applicability and generalization across a wide range of intersection types and traffic conditions. To evaluate the effectiveness of the proposed methods, an evaluation framework is established, designed to address the following research questions:

Q1: *Does TransferLight exhibit overfitting when trained exclusively on a fixed scenario?*

Q2: *To what extent does the zero-shot transfer performance of TransferLight experience improvement when trained in conjunction with the suggested DR mechanism, and what are the potential limitations associated with the effectiveness of this approach?*

Q3: *To what extent does the integration of the proposed DR mechanism into the training procedure diminish the uncertainty in the predictions made by TransferLight when confronted with novel traffic scenarios?*

By providing answers to these inquiries, this research intends to contribute to the understanding of the implications and opportunities presented by DR in the field of TSC. The insights gleaned from this work could potentially pave the way for the development of more sophisticated training algorithms utilizing DR. This could yield more robust and adaptable signal control policies that do not require additional training to respond effectively to changing traffic conditions. These policies, in turn, may eventually be integrated into real-world signal control systems, marking a significant advancement in the field.

1.3 Outline

The remainder of this thesis is organized as follows.

Chapter 2 familiarizes the reader with the necessary background used in this work. This includes an introduction to the operating principles of TSC as well as frequently used evaluation metrics and popular baseline solutions in this area. In addition, relevant techniques related to ANNs and RL are presented in this chapter.

A discussion in Chapter 3 then examines how well existing TSC solutions from the literature can be transferred to changed traffic conditions. For this purpose, both solutions derived from transportation engineering theories and those using RL are considered.

Based on these findings, several improvements are presented in Chapter 4, including the model architecture of TransferLight and the proposed DR mechanism.

Chapter 5 begins with the reiterated formulation of the research questions, thereby situating them within a broader context. Subsequently, an evaluation framework is delineated, comprising test scenarios, compared methods, and assessment metrics. This framework then serves as the methodological apparatus employed to systematically investigate and respond to the stated research questions.

Finally, Chapter 6 concludes the thesis by summarizing the most important findings, highlighting the limitations of this work, suggesting future research directions that could build upon the results of this study and discussing its implications.

2

Background

2.1 Traffic Signal Control

Coordinating traffic at intersections is a major challenge for urban planning. Due to the high and ever-increasing volume of traffic in city centers, intersections can quickly become a bottleneck if traffic is not properly coordinated, which can lead to severe traffic congestion. To avoid congested roads, traffic engineers have worked on various solutions in the past to allow road users to pass through intersections safely and efficiently. One of these solutions is the *signalized intersection*, which coordinates traffic at the intersection through electronically adjustable light signals. This section introduces the discipline of *Traffic Signal Control (TSC)*, which is concerned with adjusting the light signals of one or more signalized intersections with the goal of optimizing the efficiency of the road network based on a performance metric [34].

2.1.1 Operation Principles of Signalized Intersections

Signalized intersections come in various forms, primarily differing in the number of roads meeting at the intersection, the corresponding number of lanes, and the restrictions on road users passing through the intersection, which are coordinated by light signals. To gain a comprehensive understanding of the components constituting a signalized intersection, Figure 2.1 illustrates a generic four-way signalized intersection, serving as a visual guide to explain these concepts.

The four roads that meet the intersection from the north, south, east, and west are called *approaches* [35]. More specifically, there are four *incoming approaches* from which road users can enter the intersection and four *outgoing approaches* from which road users can exit the intersection. Each approach consists of one or more *lanes*. In the example, the east- and westbound approaches each have only one lane, while the north- and southbound approaches each have two lanes.

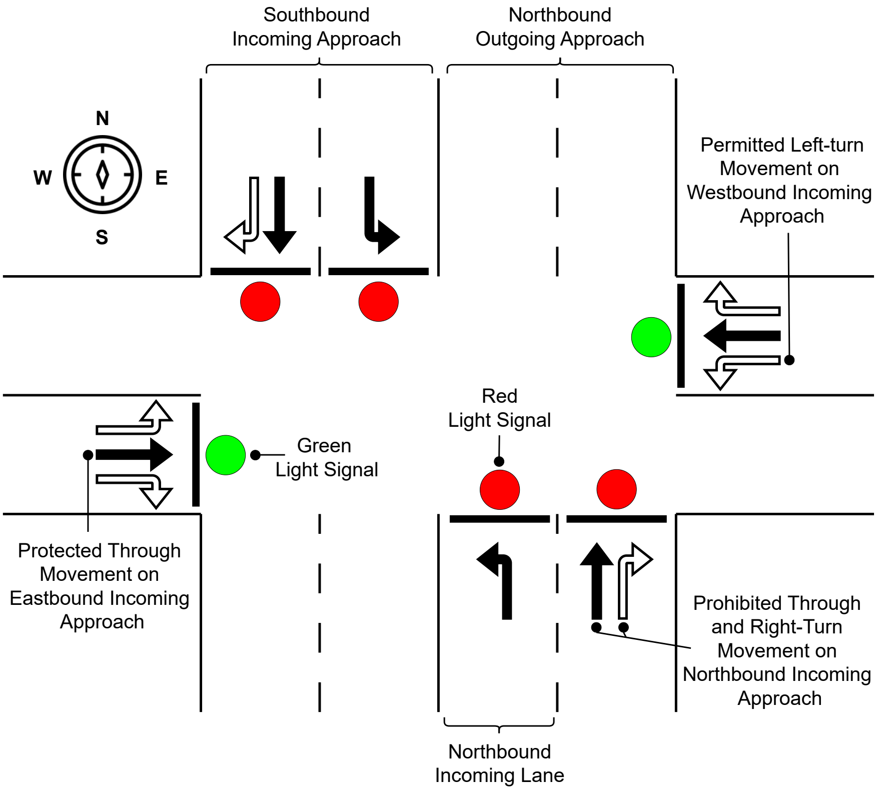


Figure 2.1: Signalized four-way intersection

Incoming lanes are associated with one or more *traffic movements* (also simply referred to as *movements*) [35, 36], each of which describes a valid path for crossing the intersection from a corresponding incoming lane to one of the outgoing lanes. As in real life, these are indicated in Figure 2.1 by markings on the ground in front of the stop line of each entry lane. Thus, in our example both the east- and westbound entry lanes enable traffic movements to all other three approaches, while the leftmost lane of the north- and southbound approaches only allow left-turn movements.

Whether a movement is allowed (i.e., has the right-of-way) or not at a given time is indicated by a *light signal* [35] (also known as *display* [36]). Light signals can either be green to indicate that the movement is allowed, yellow to warn that there is soon going to be a change of the right-of-way assignment or red if the movement is prohibited. In order to increase the efficiency of the intersection, it is common to give the green light to several movements at the same time. Considering that two simultaneously served traffic movements may conflict with each other if their paths cross, further distinction is made as to whether the movements are protected or permitted when given the green light [36]. If the movement is *protected*, the associated road users have priority and do not have to give way to other movements. If, on the other hand, a movement is only *permitted* when the light signal is green, the associated road users must yield the right-of-way to colliding traffic before they are allowed to cross the intersection. The latter is often associated with turning traffic, such as left-turners who must yield the right-of-way to oncoming through traffic, or right-turners who may not turn until pedestrians have crossed the street. In our example of a signalized four-way intersection, protected movements are indicated by filled arrows, while permitted movements are indicated by unfilled arrows. Accordingly, all movements from the westbound and eastbound approaches have the green light (indicated by the green circle), with through traffic protected, while left and right turns are only permitted. In contrast, all movements from the northbound and southbound approaches are all prohibited as indicated by the red circle.

A *phase* describes a timing procedure associated with the simultaneous operation of one or more traffic movements [36]. A single phase can be divided into a *green interval*, a *yellow change interval* and an optional *red clearance interval*. During the green interval, one or more traffic movements are granted the right-of-way, while the others are prohibited. The yellow change interval warns the associated road users given right-of-way that the phase will soon end. Finally, a red clearance interval is often inserted to clear the intersection before the next phase is initiated. The number of phases with which a signalized intersection operates is usually fixed, and each phase has a fixed assignment of protected, permitted, and prohibited traffic movements.

The order at which phases are executed and the amount of time given to each phase in the sequence define the *signal timing plan* [35]. These schedules may be obliged to satisfy several constraints [37]. One of these conditions concerns the sequence of phases. In a *cyclic-sequence phasing*, the order of phases in the plan must repeat cyclically. This is usually the standard, as it guarantees that all possible movements are served within one cycle. If the controller is allowed to use a *variable-sequence phasing* (or *skip phasing* [34]), the phases can be executed in any order. This usually results in better performance compared to a cyclic ordering as the sequence can be better matched to traffic conditions. However, this advantage may come at the cost of unhappy and confused road users whose phase may not be selected by the signal controller for a long time. Another restriction relates to the duration of the individual green phases. These can be limited to a minimum or maximum time period. A minimum time restriction is often used for safety reasons, to give road users and especially pedestrians enough time to cross the intersection, while a maximum time restriction can be applied to ensure that a particular phase does not last indefinitely.

Responsible for the execution and possibly also the adjustment of a signal timing plan is a computer program known as the *signal controller* which resides in a cabinet at the intersection. In general, a distinction can be made between pretimed and adaptive signal controllers [36]. In a *pretimed signal controller* (also known as *fixed-time controller* [34]), the signal timing plan is created offline based on historical traffic data, and the signal controller is only responsible for executing the specified plan. This type of controller has the advantage of being easy to install, since no traffic acquisition equipment is required. However, the signal controller is not able to adapt the signal control schedule to variations that deviate from the traffic pattern for which the signal timing plan was optimized. *Adaptive signal controllers* can instead adjust a signal timing plan online based on real-time traffic data. This requires detectors installed at the intersection that can sense current traffic. One of the most commonly used detector technologies is the inductive loop, which is installed under the pavement and, depending on its configuration, can measure the presence, number, queue length, and average speed of vehicles on a given lane [38]. Other, more sophisticated detectors such as surveillance cameras (in combination with computer vision technology) can measure traffic in a limited reception area rather than at a single point [38] and are therefore also capable of detecting microscopic traffic information from individual vehicles (e.g., the speed or position of a car) [34].

In addition to the adaptability of signal controls, another distinguishing feature is whether they operate in *isolation* (i.e., independently of other intersections) or in *coordination* with other intersections [36]. Isolated signal controllers are usually adequate when the nearest signalized intersections are far away or traffic demand is comparatively low. However, since the signal timing plan at one intersection can greatly affect the arrival pattern of vehicles at a downstream intersection [35], the closer the signalized intersections are to each other and the higher the traffic demand, the more reasonable it is to coordinate the signal timing plans of multiple intersections to facilitate smooth traffic flow. For this purpose, coordinated signalized intersections are treated as one system rather than as independent entities. A special type of coordination is *signal progression*, which attempts to coordinate the onset of green times of successive intersections along an arterial street in order to move road users through the major roadway as efficiently as possible [34]. Intuitively, the hope here is to create a *green wave* in which green times are cascaded so that a large group of vehicles (also called a *platoon*) can pass through the arterial street without stopping.

Throughout this thesis the set of signalized intersections in a road network is denoted by $\mathcal{I}$. The set of incoming and outgoing lanes of an intersection $i \in \mathcal{I}$ are denoted by $\mathcal{L}_{in}^{i}$ and $\mathcal{L}_{out}^{i}$, respectively. The set of possible traffic movements an intersection $i \in \mathcal{I}$ provides is represented by $\mathcal{M}^i \subset \mathcal{L}_{in}^{i} \times \mathcal{L}_{out}^{i}$, where each movement $m = (l, o) \in \mathcal{M}^i$ is associated with a corresponding outgoing lane $o \in \mathcal{L}_{out}^{i}$ that can be reached from a corresponding incoming lane $l \in \mathcal{L}_{in}^{i}$. The set of phases of an intersection $i \in \mathcal{I}$ will be denoted by Φ^i. The subset of movements being assigned the right-of-way (i.e., protected or permitted movements) if a phase $\phi \in \Phi^i$ is active will be denoted by $\mathcal{M}^{\phi}$.

2.1.2 Objectives and Performance Metrics

Signal timing plans can be optimized according to various objectives. These include mobility (e.g., minimizing travel time), safety (e.g., avoiding accidents), sustainability (e.g., minimizing emissions of pollutants and greenhouse gases), and user satisfaction (e.g., minimizing waiting times) [36]. Although all objectives are worth considering, this work will focus on mobility, as it usually also has positive impacts on other objectives such as sustainability and user satisfaction (e.g., a smooth traffic flow leads to fewer stops, which in turn reduces emissions and unhappy road users who have to wait a long time to cross the intersection), and the fact that safety concerns can be addressed through an appropriate phasing scheme design and timing constraints.

To quantify how well a signal timing configuration improves mobility, different performance metrics can be used. These can be generally categorized into intersection-level and network-level criteria [34].

Network-Level Criteria:

Network-level criteria measure the efficiency of the entire road network with a single metric. Calculating this metric and optimizing signal timing plans according to it requires network-wide sensing capabilities that become impractical as the number of intersections increases. For this reason, they are typically measured only in simulated environments to evaluate the effectiveness of signal timing plans [34].

A commonly used network-level metric is *average travel time*, which is the average time it takes a road user to travel from origin to destination. A more or less related metric is *average delay*, which measures the time a trip takes longer on average compared to the expected travel time. The expected travel time is usually based on the assumption of free-flowing traffic, i.e. traffic that is not impeded by other road users or red signal indi-

Metric	Definition	Objective
Average travel time:	$\frac{1}{\lvert\mathcal{C}\rvert}\sum_{c\in\mathcal{C}} T_{trip}(c)$	Minimize
Average delay:	$\frac{1}{\lvert\mathcal{C}\rvert}\sum_{c\in\mathcal{C}} (T_{trip}(c) - T_{free}(c))$	Minimize
Throughput:	$\lvert\mathcal{C}\rvert$	Maximize

Notation: $\mathcal{C}$ corresponds to the set of vehicles that have completed their trips within a given time period, while $T_{trip}(c)$ and $T_{free}(c)$ are the respective actual and free-flow (unimpeded) travel times of vehicle $c \in \mathcal{C}$.

Table 2.1: Network-level TSC performance metrics

cations. A third common metric is the network's *throughput*. This metric measures for a given time period the number of completed trips. A formal mathematical definition of average travel time, average delay, and throughput can be found in Table 2.1.

It should be noted that minimizing average travel time or delay is not the same as maximizing throughput, and the two can lead to different results. While the former aims at minimizing the waiting time of vehicles due to a red signal, the latter aims at increasing the capacity of the network, i.e., the number of vehicles that can efficiently traverse the network at the same time [37]. Thus, a signal setting that maximizes throughput may cause vehicles to wait a long time on a lightly trafficked route if doing so maximizes the capacity of heavily trafficked routes, while minimizing delay does so less.

Intersection-Level Criteria:

Intersection-level criteria only consider traffic variables that can be measured locally at a given signalized intersection (i.e., on the incoming or outgoing lanes). Although such criteria are well suited to quantify the performance of individual intersections, they are not sufficient by themselves to evaluate the coordination ability of multiple signalized intersections in a roadway network. To overcome this limitation and obtain a network-level metric, a common strategy is to aggregate (usually sum) the measurements of all participating signalized intersections [34].

Two of the most commonly used intersection-level metrics are the total *queue length* and *waiting time* of vehicles queuing in the intersection's entry lanes. In both cases, it is common to assume that vehicles are queuing or waiting when they are traveling below a certain reference speed [35]. A commonly used reference speed in practice is

Metric	Definition	Objective
Queue length:	$\sum_{l \in \mathcal{L}^i_{in}} N_{wait}(l)$	Minimize
Waiting time:	$\sum_{l \in \mathcal{L}^i_{in}} T_{wait}(l)$	Minimize
Pressure:	$\left\lvert \sum_{(l,o) \in \mathcal{M}^i} N(l) - N(o) \right\rvert$	Minimize

Notation: As mentioned before, each intersection $i \in \mathcal{I}$ has a set of incoming and outgoing lanes, denoted by $\mathcal{L}^i_{in}$ and $\mathcal{L}^i_{out}$, respectively, as well as a set of possible traffic movements $\mathcal{M}^i \subset \mathcal{L}^i_{in} \times \mathcal{L}^i_{out}$ connecting incoming and outgoing lanes. The number of vehicles currently in a lane l is denoted by $N(l)$, while $N_{wait}(l)$ denotes the number of vehicles currently queuing in lane l (i.e., the number of vehicles traveling below a certain speed threshold). $T_{wait}(l)$ is the current total waiting time of the vehicles queued in lane l (i.e., the sum of the waiting times of the individual vehicles in the queue). Here, the waiting time of a single vehicle corresponds to the time since its last movement.

Table 2.2: Intersection-level TSC performance metrics

0.1 meters per second [7, 21, 17]. The formal mathematical definitions of queue length, waiting time, and pressure at an intersection are summarized in Table 2.2. It should be noted that this list is not exhaustive. The interested reader can find a more complete list in [34].

Similar to throughput and travel time at the network level, it is worth mentioning that minimizing queue length may lead to a different outcome than minimizing waiting time. More specifically, a plan that minimizes queue length may result in priority being given to road users located on high-volume approaches in order to keep the overall queue length low. This, in turn, could result in road users waiting indefinitely on the low volume approaches, which is contrary to the goal of minimizing the waiting time. On the other hand, if the arriving traffic flow is evenly distributed among all approaches to the intersection, it can be shown that a plan that minimizes the queue length also minimizes the waiting time [10]. Another commonly used measure at the intersection level is the *pressure* [39] of an intersection, which can be defined as the absolute sum of differences between the number of vehicles on the incoming and outgoing lanes of every traffic movement [9], although definitions vary [13, 25]. From an intuitive standpoint, pressure is essentially a measure to describe the degree of imbalance between vehicles approaching the intersection and vehicles leaving it. By minimizing the sum of pressures of multiple signalized intersections in a road network, it can be shown that this is equivalent to maximizing the throughput of the entire road network and results in a coordinative signal timing plan that attempts to distribute vehicles evenly throughout the road network [9].

2.1.3 Webster Method

A legitimate question for a pretimed signal controller with a cyclic signal schedule is how to optimally set the *cycle length* (i.e., the duration of one cycle of all phases) and the *phase split* (i.e., the proportion of the cycle length assigned to each phase) based on historical traffic data. The *Webster method* [40] offers a possible solution to this question.

For this method to be applied, some traffic-related variables must be known in advance. This includes the normal and saturation flow rate of each entry lane to the intersection i as well as its total lost time. The *normal flow rate* $d(l)$ of an entry lane $l \in \mathcal{L}_{in}^{i}$ refers to the hourly rate at which vehicles would pass the lane's stop line to the intersection if the light signal were permanently green. In contrast, the *saturation flow rate* $c(l)$ of an entry lane $l \in \mathcal{L}_{in}^{i}$ describes the maximum possible hourly vehicle rate that the lane can handle, again assuming a permanent green signal indication. This rate is typically observed shortly after the lane's light signal switches from red to green and the vehicles in the queue begin to move (usually after the fourth or fifth vehicle in the queue has passed the lane's stop line [41]). Although saturation flow can be measured based on field observations, Webster provides a simple heuristic for determining the rate based on the width of a lane that gives good results in practice [40]. Finally, the *total lost time* $T_{lost}(i)$ of the intersection refers to the total time within a cycle in which the intersection is not effectively used by any vehicle. It can be calculated using the formula

$$T_{lost}(i) = \sum_{\phi \in \Phi^{i}} \big(T_{startup}(\phi) + T_{clearance}(\phi)\big), \tag{2.1}$$

where $T_{lost}(\phi)$ is the lost startup time, i.e., the time lost due to road user reaction time when phase $\phi \in \Phi^{i}$ is initiated (usually assumed to be 2 seconds [41]), and $T_{clearance}(\phi)$ is the lost clearance time of phase $\phi \in \Phi^{i}$, which Webster assumes to be the total length of its yellow change and all-red clearance interval.

Once these variables are determined, for each entry lane $l \in \mathcal{L}_{in}^{i}$, the ratio of normal to saturation flow rate $y(l) = d(l)/c(l)$ is calculated. Then, for each phase $\phi \in \Phi^{i}$, the critical ratio $y(\phi)$ is computed, which corresponds to the maximum ratio of all entry lanes that receive a green light when the phase is executed, i.e.,

$$y(\phi) = \max_{(l,\cdot)\in\mathcal{M}^{\phi}} y(l). \tag{2.2}$$

In a final step, the cycle length $T_{cycle}(i)$ and the green time duration $T_{green}(\phi)$ of each phase $\phi \in \Phi^i$ (i.e., the phase split) are determined using the following formulas:

$$T_{cycle}(i) = \frac{1.5T_{lost}(i)+5}{1-\sum_{\phi\in\Phi^i} y(\phi)}, \qquad T_{green}(\phi) = \frac{y\phi}{\sum_{\phi\in\Phi^i} y(\phi)}\big(T_{cycle}(i) - T_{lost}(i)\big). \tag{2.3}$$

Assuming that traffic arrives uniformly at the intersection, the cycle length and phase split determined by Webster's method is guaranteed to yield a signal timing plan that minimizes travel time [35].

2.1.4 GreenWave

Although the Webster method provides a simple way to determine a cyclic signal timing plan for an isolated intersection, it is not sufficient on its own for scenarios that require coordination between multiple signalized intersections. To address this, one can use the method described in [41], also referred to by some authors as *GreenWave* [35, 9]. This method allows the determination of the offsets between the cyclic signal timing plans (e.g., determined via the Webster method) of adjacent intersections along an arterial street to achieve signal progression for traffic coming from one direction on the major roadway.

To determine the offsets, GreenWave requires that the signal schedules of the intersections involved share the same cycle length. In addition, the distance $D(i,j)$ between each neighboring intersections $i \in \mathcal{I}$ and $j \in \mathcal{I}$ must be known, as well as the expected speed $v_{arterial}$ of vehicles on the main road in the direction in question. Then, the signal timing offset $\Delta t(i,j)$ that an intersection i must have with respect to the preceding intersection j can be determined using the following formula:

$$\Delta t(i,j) = \frac{D(i,j)}{v_{arterial}}. \tag{2.4}$$

If the assumptions hold that traffic flows uniformly and unidirectional along the major and minor streets, this method guarantees a cascading of green times that minimizes delay [35].

2.1.5 Max Pressure

So far, using the Webster method and GreenWave, only solutions for pretimed signal controllers with a cyclic signal timing plan have been considered. As discussed in Section 2.1.1, such methods are fair and do not require real-time traffic detection, but they do not provide the necessary flexibility as traffic conditions change. *Max Pressure* control [39], a fully adaptive, variable-sequence phasing method, attempts to remedy this. Its main idea is to represent the evolution of vehicle queues[1] in a road network as a Markov chain, called a *store-and-forward model.* Based on this model, a simple algorithm can be derived for each independent intersection that greedily selects a suitable phase for each time step in the chain.

An important precondition of Max Pressure is that each signalized intersection $i \in \mathcal{I}$ is capable of simultaneously detecting the number of vehicles in each of its entry and exit lanes at time t, which can be summarized in a state representation $s_t(i) = \{N(l) | l \in \mathcal{L}^i_{in} \cup \mathcal{L}^i_{out}\}$ (where $N(l)$ denotes the number of vehicles on lane l). If this is satisfied, Max Pressure lets each intersection $i \in \mathcal{I}$ select phases $\phi \in \Phi^i$ according to the following control policy [9]:

$$\begin{aligned} \pi(s_t(i)) &= \underset{\phi \in \phi^i}{\arg\max}\, \gamma(\phi), \\ \text{where } \gamma(\phi) &= \sum_{(l,o) \in \mathcal{M}^\phi} \big(N(l) - N(o)\big). \end{aligned} \tag{2.5}$$

Here, $\gamma(\phi)$ is often referred to as the *pressure* of phase ϕ and summarizes in a single number the imbalance between the queue length in the incoming and outgoing lanes of each traffic movement $(l, o) \in \mathcal{M}^\phi$. Accordingly, by choosing the phase with the highest pressure at each time step t, Max Pressure is essentially trying to balance the length of the queues at the intersection.

One of the most significant advantages of Max Pressure is its theoretical guarantees. Despite relying only on local traffic information at each intersection, controlling each intersection with this policy provable leads to a stabilizing of queue length in the entire road network and a maximization of network throughput, as long as the assumptions of the model hold (see proof in [39]). These assumptions include that lanes have unlimited vehicle storage capacity and are assigned to only one traffic movement.

---

[1] In Max Pressure, vehicles are considered queued as soon as they enter the incoming lane to an intersection

2.2 Deep Learning

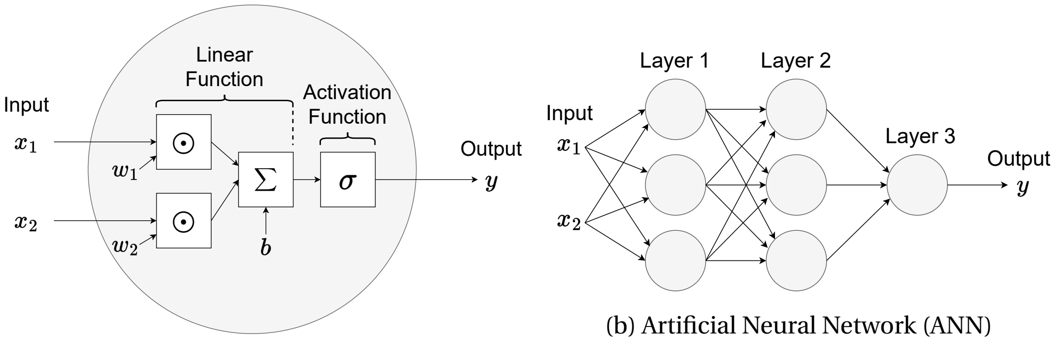
As already indicated in Section 2.1, an adaptive signal controller employing variable-sequence phasing can be considered a computer program that receives as input a representation of the current traffic state and outputs the next phase to be executed. Suppose for a moment that we had an oracle that provided us with a dataset $D = \{(\boldsymbol{x}, \boldsymbol{y})\}^m$, consisting of m samples, where each traffic situation is described by a vector $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^n$ and is mapped to the best possible next phase $\boldsymbol{y} \in \mathcal{Y}$. Then, assuming that there is a true functional relationship $f^* : \mathcal{X} \to \mathcal{Y}$, which is unknown to us, the task could simply boil down to approximate f^* using the available dataset D. This section introduces the reader to *Deep Learning (DL)*, a branch of *Machine Learning (ML)*, that provides a way to approximate f^* with arbitrary accuracy using a parameterized functional mapping $\boldsymbol{y} = f(\boldsymbol{x};\boldsymbol{\theta})$ whose parameters $\boldsymbol{\theta}$ are learned in an end-to-end fashion. Unless otherwise noted, the information contained in this section refers to the DL textbook by Goodfellow et al. [42].

2.2.1 Artificial Neural Networks

One of the main reasons for the success of DL is probably the functional representation of $f(\boldsymbol{x},\boldsymbol{\theta})$, which is strongly inspired by the structure and functioning of a biological brain. As a basic processing unit, this functional representation relies on so-called *artificial neurons* (also known as *Perceptrons* [43]). Artificial neurons are essentially feature detectors that receive a series of scalar input signals $x_1, x_2, \ldots \in \mathbb{R}$ and produce a scalar output signal $y \in \mathbb{R}$ that describes whether and to what extent a particular pattern is present in the input signals. To do this, they calculate a *linear function* of their input by multiplying each input signal x_i by an associated *weight* $w_i \in \mathbb{R}$ and sum these intermediate results, including a *bias* $b \in \mathbb{R}$. The result of this linear function is then passed through a nonlinear *activation function* $\sigma : \mathbb{R} \to \mathbb{R}$ that decides whether the feature that the neuron can detect is present in the input, and if so, with what intensity. These two operations, the linear function followed by the nonlinear activation function, are shown in a computation graph in Figure 2.2(a) and can be summarized mathematically by the following expression:

$$y = \sigma(\sum_i w_i x_i + b). \tag{2.6}$$

An interesting property of artificial neurons is that the weights $w_1, w_2, \ldots$ as well as the bias b, which are primarily responsible for which feature a neuron can recognize in



(a) Artificial Neuron

(b) Artificial Neural Network (ANN)

Figure 2.2: Computation graph of a single artificial neuron and a network of artificial neurons

its input, are part of the parameters $\boldsymbol{\theta}$ and thus do not have to be set manually but are learned. Accordingly, artificial neurons are learnable feature detectors.

As far as the choice of a suitable activation function is concerned, there are many possibilities. In the early days of DL, the *Sigmoid* or *Hyperbolic Tangent (Tanh)* functions were mostly used. These are monotonically increasing functions that transform their input into a range between 0 and 1 (Sigmoid) or -1 and 1 (Tanh). However, since these two functions saturate over most of their domain, making learning difficult (see Section 2.2.4), their use today is mostly limited to situations where neurons needs to output a value in the desired range (e.g., in a binary classification task). Instead, today it is more common to use a *Rectified Linear Unit (ReLU)* [44] function that is linear in half of its domain and outputs 0 only for negative inputs. If the output of multiple neurons needs to be normalized (e.g., to represent a probability vector in a multiclass classification task), one can also use the *softmax* activation function. In cases where the neuron is forced to output a potentially unconstrained value (e.g., in a regression task), the activation function can be omitted altogether. The formal definitions of these activation functions is presented in Table 2.3.

One limitation of artificial neurons is that they cannot approximate nonlinear functions on their own. To solve this problem, multiple neurons can be combined into a network by using the output of neurons as input to other neurons. Here, it is common to arrange neurons into *layers*, with neurons in the same layer performing computations based on the outputs of neurons from the previous layer(s) and passing the results to the subsequent layer(s) as input. To ensure that the network is capable of learning nonlinear functions, the only requirement is that neurons of *hidden layers* (all layers except the final *output layer*) must use nonlinear activation functions. The resulting

Activation Function	Definition	Range
Sigmoid:	$\sigma(z) = \frac{1}{1+\exp(-z)}$	$(0,1)$
Tanh:	$\sigma(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$	$(-1,1)$
ReLU:	$\sigma(z) = \max\{0,z\}$	$[0,\infty)$
Softmax:	$\sigma(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$	$(0,1)$
Linear (no activation):	$\sigma(z) = z$	$(-\infty,\infty)$

Table 2.3: Activation functions frequently used in DL

architectural structure is generally referred to as an *Artificial Neural Network (ANN)*. An exemplary ANN with two hidden layers, each with three neurons, and a final output layer with one neuron is shown in Figure 2.2(b). By representing the computations of all neurons of each layer $l = 1,\dots,L$ as a vector-valued function $f^{(l)}$, one can also represent an ANN with L layers as a composition of functions:

$$\boldsymbol{y} = f(\boldsymbol{x},\boldsymbol{\theta}) = f^{(L)} \circ f^{(L-1)} \circ \dots \circ f^{(1)}(\boldsymbol{x}). \tag{2.7}$$

Over the years, DL researchers have developed various types of layers with different data structures in mind. This includes layers for vector-valued data [43], grid-structured data (e.g., images) [45, 46], sequential data [47] as well as graph-structured data [48]. The proposed layers differ mainly in how neurons in one layer are connected to (i.e., depend on) neurons in the previous layer(s) and whether neurons share parameters within and between layers.

2.2.2 Fully Connected Layers

Perhaps the most common building block in an ANN is a *fully connected layer* [43], in which each neuron in a layer l is connected to all neurons in the previous layer $l-1$. Because of this dense connectivity, the weights and biases of the individual neurons of layer l can be combined into a matrix $\boldsymbol{W}^{(l)} \in \mathbb{R}^{u\times v}$ and a vector $\boldsymbol{b}^{(l)} \in \mathbb{R}^{u}$, respectively, where u corresponds to the number of neurons in layer l and v to the number of neurons in layer $l-1$. The function computed by layer l can then be expressed mathematically as follows[2]:

$$f^{(l)}(\boldsymbol{x}^{(l-1)}) = \sigma^{(l)}(\boldsymbol{W}^{(l)}\boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)}). \tag{2.8}$$

---

[2] Note that the activation function $\sigma^{(l)}$ is applied element-wise.

A composition of fully connected layers with at least one hidden layer is often referred to as a *Multilayer Perceptron (MLP)* [43] (see Figure 2.2(b)). Since each neuron in a MLP is able to learn a different function (no sharing of parameters between neurons) based on all neuron outputs from the previous layer, a MLP has maximum flexibility when it comes to approximating functions. This can also be confirmed by the *universal approximation theorem* [49, 50], which shows that a MLP with a linear output layer and at least one hidden layer can theoretically represent any continuous function on a closed and bounded subset of the real numbers with arbitrary accuracy, provided its hidden layers comprise enough neurons.

Despite their flexibility in function approximation, fully connected layers are inflexible when the dimensionality of their input changes. Their use is therefore limited to input data with fixed dimensionality. For grid-structured data with varying resolutions or graph-structured data with a varying number of nodes and edges, the exclusive use of fully connected layers is thus unsuitable.

2.2.3 Neural Message Passing Layers

Neural message passing [48] is a framework that attempts to solve the inflexibility issue of fully connected layers for graph-structured data. Before explaining its general operation, it is necessary to introduce some notation which will be reused throughout this thesis. A *graph* will be denoted by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the set of *nodes* (or vertices) while $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the set of directed *edges* (or connections) between nodes. The *neighbors* of a node $v \in \mathcal{V}$ (i.e., the set of nodes that have a directed edge to v) are denoted by $\mathcal{N}(v) = \{u | (u, v) \in \mathcal{E}\}$. If a node v is associated with some features, the corresponding vector is denoted by $\boldsymbol{x}(v)$. Similarly the feature vector of an edge $(u, v) \in \mathcal{E}$ is denoted by $\boldsymbol{x}(u, v)$. Moreover, $\boldsymbol{x}_{||}(u, v)$ represents a shorthand notation for the concatenation of the node feature vectors $\boldsymbol{x}(u)$ and $\boldsymbol{x}(v)$ as well as the edge feature vector $\boldsymbol{x}(u, v)$.

Using this notation, the main idea of neural messaging is to update the vector representation of each node $v \in \mathcal{V}$ in the graph, by enriching it with features from its neighborhood $\mathcal{N}(v)$. To this end, a single layer l of neural messaging applies the same parameterized function $f^{(l)}(v)$ to each node $v \in \mathcal{V}$. This function can be decomposed into a chain of two functions. The first is an *aggregation function* $f_{agg}^{(l)}$, which receives as input the concatenated feature representation $\boldsymbol{x}_{||}^{(l-1)}(u, v)$ of each neighbor $u \in \mathcal{N}(v)$ of a node $v \in \mathcal{V}$ and is responsible for aggregating this information into a single message vector $\boldsymbol{m}_{\mathcal{N}(v)}^{(l)}$. The second function in the chain, the *update function* $f_{upd}^{(l)}$, then

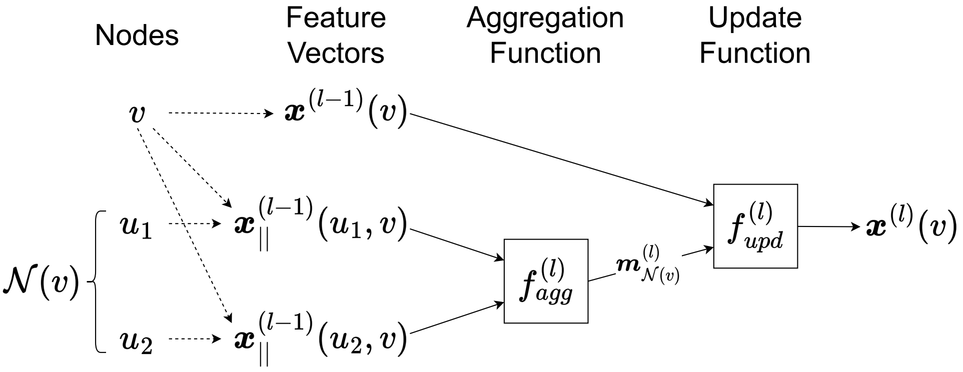

Nodes
Feature Vectors
Aggregation Function
Update Function
v
u_1
u_2
$\mathcal{N}(v)$
$\boldsymbol{x}^{(l-1)}(v)$
$\boldsymbol{x}_{||}^{(l-1)}(u_1, v)$
$\boldsymbol{x}_{||}^{(l-1)}(u_2, v)$
$f_{agg}^{(l)}$
$\boldsymbol{m}_{\mathcal{N}(v)}^{(l)}$
$f_{upd}^{(l)}$
$\boldsymbol{x}^{(l)}(v)$

Figure 2.3: Computation graph of a neural message passing layer

uses the message vector $\boldsymbol{m}_{\mathcal{N}(v)}^{(l)}$ of v along with its old vector representation $\boldsymbol{x}^{(l-1)}(v)$, to generate an updated vector representation $\boldsymbol{x}^{(l)}(v)$ from it. This general computation scheme is shown for a simple example in a computation graph in Figure 2.3 and can be summarized mathematically by the following expression:

$$\begin{aligned} \boldsymbol{x}^{(l)}(v) &= f^{(l)}(v) \\ &= f_{upd}^{(l)}\left(\boldsymbol{x}^{(l-1)}(v), f_{agg}^{(l)}\left(\{\boldsymbol{x}_{||}^{(l-1)}(u,v), \forall u \in \mathcal{N}(v)\}\right)\right) \\ &= f_{upd}^{(l)}\left(\boldsymbol{x}^{(l-1)}(v), \boldsymbol{m}_{\mathcal{N}(v)}^{(l)}\right). \end{aligned} \tag{2.9}$$

A common implementation for the update function $f_{upd}^{(l)}$ is a MLP applied to the concatenated input vectors. However, the choice of the aggregation function $f_{agg}^{(l)}$ is less trivial, since it should be applicable to any number of adjacent nodes. Also, since the input to the function is a set of vectors with no particular order, the function should be *permutation invariant*, i.e., it should return the same message vector $\boldsymbol{m}_{\mathcal{N}(v)}^{(l)}$ regardless of the order in which the vectors are presented to the function. A popular approach that satisfies both conditions is to transform the concatenated vector representation $\boldsymbol{x}_{||}^{(l-1)}(u,v)$ of each neighbor $u \in \mathcal{N}(v)$ by the same parameterized *message function* $f_{msg}^{(l)}$ (e.g., a MLP), which is then followed by a summation [48]:

$$\boldsymbol{m}_{\mathcal{N}(v)}^{(l)} = \sum_{u \in \mathcal{N}(v)} f_{msg}^{(l)}\left(\boldsymbol{x}_{||}^{(l-1)}(u,v)\right). \tag{2.10}$$

This approach has the advantage of capturing structural graph features (e.g., the degree of nodes) well in the updated vector representations [51]. However, due to the summation it can also lead to numerical instabilities and difficulties in optimization during training (see Section 2.2.4) if the magnitude of the aggregated message vectors varies widely as a result of highly fluctuating node degrees [51]. A solution to this problem, which is especially applicable when the function to be approximated depends less on the structure of the graph and relies more on the information about the node features, is to compute a normalized sum instead [51]:

$$\boldsymbol{m}^{(l)}_{\mathcal{N}(v)} = \sum_{u \in \mathcal{N}(v)} \alpha^{(l)}_{u,v} \cdot f^{(l)}_{msg}\left(\boldsymbol{x}^{(l-1)}_{||}(u,v)\right). \tag{2.11}$$

The normalization weights $\alpha^{(l)}_{u,v}$ can be determined in various ways. A common strategy known as the *attention mechanism* [52] lets the aggregation function simply learn how to set the weights based on the vector representations involved:

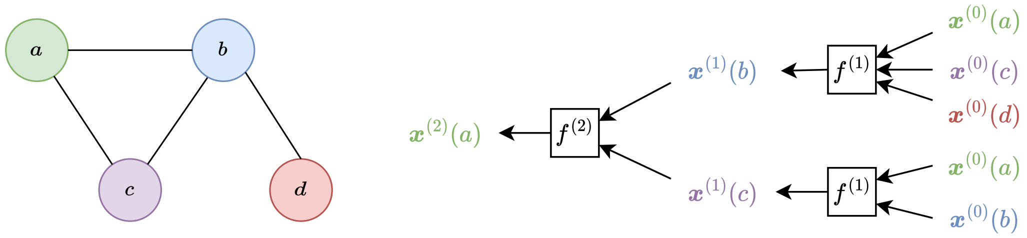
$$\alpha^{(l)}_{u,v} = \frac{\exp\left(f^{(l)}_{score}\left(\boldsymbol{x}^{(l-1)}_{||}(u,v)\right)\right)}{\sum_{u' \in \mathcal{N}(v)} \exp\left(f^{(l)}_{score}\left(\boldsymbol{x}^{(l-1)}_{||}(u',v)\right)\right)}. \tag{2.12}$$

Here, the scoring function $f^{(l)}_{score}$ represents a parameterized function (e.g., a MLP with a linear output layer) that is applied to each concatenated feature vector to produce a scalar for each neighbor. To obtain normalized weights that sum to one, the scalars are subsequently transformed by the softmax function. A popular extension is the use of *multi-head attention* [53]. Here, a fixed number H of so-called *attention heads* are used in parallel. Each attention head $h = 1, \ldots, H$ learns its own scoring function $f^{(l,h)}_{score}$ and message function $f^{(l,h)}_{msg}$ with its own parametrization. The results of each normalized summation can then be simply combined by concatenation (denoted here by $\|$) to form the final message vector $\boldsymbol{m}^{(l)}_{\mathcal{N}(v)}$:

$$\boldsymbol{m}^{(l)}_{\mathcal{N}(v)} = \overset{H}{\underset{h=1}{\Big\|}} \left[\sum_{u \in \mathcal{N}(v)} \alpha^{(l,h)}_{u,v} \cdot f^{(l,h)}_{msg}\left(\boldsymbol{x}^{(l-1)}_{||}(u,v)\right)\right]. \tag{2.13}$$

The advantage of using multiple heads is that each head can attend to and capture a different aspect in the neighborhood, which can greatly increase the representational power of the aggregation function.

An ANN that uses multiple message passing layers is commonly referred to as a *Graph Neural Network (GNN)*. An interesting property of GNNs is that the receptive field of a node increases by one hop with each additional message passing layer. Accordingly, after L layers of neural message passing, each node $v \in \mathcal{V}$ in the graph contains information about nodes L hops away in its vector representation $\boldsymbol{x}^{(L)}(v)$. This can be easily demonstrated with a bidirectional example graph with four nodes, shown in Figure 2.4(a). As can be seen, although node a is only connected to nodes b and c and not to


a
b
c
d
$\boldsymbol{x}^{(2)}(a)$
$f^{(2)}$
$\boldsymbol{x}^{(1)}(b)$
$f^{(1)}$
$\boldsymbol{x}^{(0)}(a)$
$\boldsymbol{x}^{(0)}(c)$
$\boldsymbol{x}^{(0)}(d)$
$\boldsymbol{x}^{(1)}(c)$
$f^{(1)}$
$\boldsymbol{x}^{(0)}(a)$
$\boldsymbol{x}^{(0)}(b)$

(a) Example graph

(b) Computation graph

Figure 2.4: Two layers of neural message passing applied on a bidirectional example graph

node d, the computational graph in Figure 2.4(b) reveals that after two layers of neural message passing, information stored in node d reaches node a.

Despite the advantage of neural message passing in representation learning over arbitrarily structured graphs, one challenge is that node identity is lost. Each neighboring node is treated equally by the same message function before those messages are aggregated. This is conceptually very different from fully connected layers. Here, the identity of neurons is preserved as they have a fixed order in a vector and are multiplied by distinct weights before being aggregated. Therefore, identity-related features that can be implicitly learned in fully connected layers are lost in neural message passing. To account for this information loss, a common approach is to explicitly encode such features in the node and edge feature vectors. In this way identity-related features of nodes can be preserved even if processed by the same message function. For example, to preserve node order information, one can add a *positional encoding* to node feature vectors. A popular method here is a *sinusoidal positional encoding* [53], which generates a d-dimensional vector based on $d/2$ pairs of sine and cosine signals whose wavelengths geometrically progress from 2π to $10000 \cdot 2\pi$. Specifically, at position t the i^{th} dimension of the corresponding positional encoding vector is computed as follows:

$$PE(t,i) = \begin{cases} \sin(t/10000^{i \cdot d}), & \text{if } i \bmod 2 = 0 \\ \cos(t/10000^{i \cdot d}), & \text{otherwise.} \end{cases} \tag{2.14}$$

In contrast to directly encoding the position as a natural number, a sinusoidal positional encoding has several advantages. It keeps the values in a normalized range, a property generally desired in ANN training to avoid numerical instabilities. Moreover, according to the authors, it allows extrapolation to positions larger than those observed during training, since any offset to t can be represented by a linear function of t [53].

Range $\boldsymbol{y}$	**Model** $p(\boldsymbol{y}\vert\boldsymbol{x};\boldsymbol{\theta})$	**Cost** $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{(\boldsymbol{x},\boldsymbol{y})\in D} L(\boldsymbol{\theta})$
$y \in (-\infty, \infty)$	**Gaussian** $\mathcal{N}(y; \mu = f(\boldsymbol{x};\boldsymbol{\theta}), \sigma = 1)$	**Mean Squared Error (MSE)** $L(\boldsymbol{\theta}) = (y - f(\boldsymbol{x},\boldsymbol{\theta}))^2$
$y \in \{0,1\}$	**Bernoulli** $\mathcal{B}(y; \phi = f(\boldsymbol{x};\boldsymbol{\theta}))$	**Binary Cross-Entropy** $L(\boldsymbol{\theta}) = -[y \log(f(\boldsymbol{x};\boldsymbol{\theta}))$ $+(1-y)\log(1 - f(\boldsymbol{x};\boldsymbol{\theta}))]$
For k categories $\boldsymbol{y} \in \{0,1\}^k$ s.t. $\sum_{j=1}^{k} \boldsymbol{y_j} = 1$	**Multinoulli / Categorical** $\mathcal{C}(\boldsymbol{y}; \boldsymbol{\phi} = f(\boldsymbol{x};\boldsymbol{\theta}))$	**Cross-Entropy** $L(\boldsymbol{\theta}) = -\sum_{j=1}^{k} \boldsymbol{y_j} \log(f(\boldsymbol{x};\boldsymbol{\theta}))$

Table 2.4: Cost functions frequently used in DL

2.2.4 Training Artificial Neural Networks

Having described the general structure of ANNs, the question remains how their parameters $\boldsymbol{\theta}$ are learned to approximate the target function f^*. In DL the most common way is to treat $\boldsymbol{y} = f(\boldsymbol{x};\boldsymbol{\theta})$ as a parametric model $\boldsymbol{y} \sim p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$ and perform *Maximum Likelihood Estimation (MLE)*, whereby it is assumed that the samples $(\boldsymbol{x},\boldsymbol{y}) \in D$ are *independent and identically distributed (i.i.d.)*. One can show that this is equivalent to the following optimization objective:

$$\boldsymbol{\theta_{MLE}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim D} -\log p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}) \tag{2.15}$$

The expression shows that we are trying to find the parameters $\boldsymbol{\theta}$ of our model that minimize a *cost function* $J(\boldsymbol{\theta})$ corresponding to the expected *negative log-likelihood* $L(\boldsymbol{\theta}) = -\log p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$ (often referred to as the *loss function*) with respect to the empirical data distribution defined by our data set D. Depending on the distribution chosen for the parametric model $p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta})$, which should be chosen according to the range of the output $\boldsymbol{y}$, different cost functions can be derived. Some common ones are listed in Table 2.4.

The cost function $J(\boldsymbol{\theta})$ is typically minimized using an iterative algorithm known as *gradient descent*. This algorithm starts with an initial parameter setting, and iteratively updates the parameters by moving them in the opposite direction of the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, scaled by a learning rate η, until a convergence criterion is satisfied. The update rule is given as follows:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}). \tag{2.16}$$

To calculate the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ the *back-propagation algorithm* [47] is usually used, which exploits the chain rule of calculus to provide an efficient method for computing the gradient of chained-like functions such as ANNs. Moreover, instead of computing the gradient over the entire dataset D, it is common to estimate it using *minibatch gradient descent*, a variant in which each gradient update is performed on a random subset of samples in dataset D whereby the size of the subset, called the *batch size* B, remains fixed. As the calculation of the gradient requires a sum over the entire dataset, this adaptation of the algorithm results in a lower computational cost for performing one step of gradient descent.

The goal of this training procedure is *generalization*, which describes the ability of the model to perform well on examples of the data distribution, which were not observed during training. To estimate generalizability of a model, the dataset D is typically split into a *training dataset* and a *test dataset*. While the training dataset is used for training, the test dataset is used solely to evaluate the generalization of the model against a performance metric. If the final model performance on the test dataset is as desired and approximately equal to the performance on the training dataset, the model is said to generalize well. If, on the other hand, the final model performs poorly on both the training and test datasets, the model is said to *underfit* the training dataset. In such a case, the capacity of the model, defined in an ANN roughly as the number of its neurons and layers, is usually too low and should be increased. When performance is good on the training dataset but poor on the test data set, the model is said to *overfit* the training dataset. This problem usually occurs when the capacity of the model is too high, so that the model unnecessarily adapts to the noise in the training dataset without capturing the general trend. To reduce the capacity of the model in such a scenario, in addition to an elaborate adaptation of the architecture of the ANN, one can apply *regularization* methods, which are modifications to the learning algorithm aimed at improving generalization. Since regularization techniques are beyond the scope of this thesis, they are not discussed in detail here. The interested reader will find a detailed description of common regularization techniques in the DL textbook by Goodfellow et al. [42].

Besides generalization difficulties, gradient-based optimization poses many challenges. Due to the nonlinearities in the ANN architecture, the cost function can become highly non-convex, making convergence of gradient descent cumbersome. This usually manifests itself in the algorithm getting stuck at unfavorable local minima (i.e., singularities), saddle points or plateaus due to *vanishing gradients* (especially in initial layers) or the algorithm behaving unstable as a result of large or even *exploding gradients*. In addition, neural network training suffers from a phenomenon known as *internal covariate shift* [54], which describes the problem that the distribution of inputs to a layer is not constant during training, but changes with each update applied to the parameters of the previous layers.

To tackle these challenges, DL researchers have come up with several techniques. Among them are *parameter initialization strategies*, that attempt to start gradient descent in a region of parameter space that aids optimization, particularly at the beginning of training. For example, in the *Xavier* [55] or *He initialization* [56], the weights of each layer are randomly sampled and scaled accordingly so that the activation and gradient variance is initially constant across all layers, whereas the *Saxe initialization* [57] uses random orthogonal matrices to initialize the weights. *Architectural design decisions* are another way to accelerate training. For example, *skip/residual connections* [58, 59], where the input of a sequence of layers is combined with its output (e.g., by addition or concatenation), have been shown to drastically reduce the number of singularities in the loss landscape [60]. Besides initialization and architectural modifications, one can also use more *advanced optimization algorithms* that attempt to improve gradient descent. For example, the *Adam optimizer* [61], one of the most widely used gradient descent optimization algorithms, individually adjusts the learning rate for each parameter during training by estimating the first and second moments of the gradients. Finally, to mitigate the effects of internal covariate shift, *input normalization* can be used in which the input signals to a layer are standard normalized and then rescaled and shifted based on learnable parameters. The statistics used for normalization can be determined in several ways. While *batch normalization* [54] estimates the scale and shift parameters for each individual input signal over all examples in the batch, which is sensitive to the size of the batch, *layer normalization* [62] determines a single scale and shift parameter over all input signals of an example based on which the individual input signals of the example are normalized. Such normalization techniques have been shown to significantly stabilize the training process and lead to faster convergence.

2.2.5 Monte Carlo Dropout

To assess how trustworthy a model is after training, in addition to quantifying the final performance of the model in its predictions, it may also be desirable to quantify the uncertainty (i.e., confidence) of the model (also called *epistemic uncertainty* [63]). A simple technique that can be used to estimate model uncertainty in DL is *Monte*

Carlo (MC) dropout [64]. This method relies on *dropout* [65], a technique originally developed for regularization. Here, the output signals $\boldsymbol{x}^{(l)} \in \mathbb{R}^n$ of a layer l, before being passed to the next layer $l+1$, are element-wise multiplied at each training step by a random binary mask $\boldsymbol{z}^{(l)} \in \{0,1\}^n$ whose entries $z_1^{(l)}, \ldots, z_n^{(l)} \overset{iid}{\sim} \mathcal{B}(1-\phi)$ are 0 with probability ϕ (called the *dropout rate*). This process causes neurons in layer l to drop out randomly with a probability of ϕ, and accordingly, neurons are induced to learn to become useful feature extractors with respect to many possible subnetworks of the original ANN, effectively reducing overfitting. By applying dropout at inference time (i.e., after training), it can be shown that each sampled *dropout mask* (which combines all binary masks of each layer where dropout is applied) results in an ANN, whose parameters $\boldsymbol{\theta}_t \sim q(\boldsymbol{\theta}|D)$ correspond to a sample of the *parametric approximate posterior distribution* $q(\boldsymbol{\theta}|D)$. MC dropout takes advantage of this property by using an ensemble of T ANNs to approximate the mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$ of the *predictive distribution* $p(\boldsymbol{y}|\boldsymbol{x}, D)$ for an arbitrary input x:

$$\boldsymbol{\mu} = \frac{1}{T}\sum_{t=1}^{T} f(\boldsymbol{x};\boldsymbol{\theta}_t), \qquad \boldsymbol{\sigma}^2 = \frac{1}{T}\sum_{t=1}^{T} (f(\boldsymbol{x};\boldsymbol{\theta}_t) - \boldsymbol{\mu})^2, \qquad \text{s.t. } \boldsymbol{\theta}_t \sim q(\boldsymbol{\theta}|D). \tag{2.17}$$

While the mean $\boldsymbol{\mu}$ is typically used as the model's prediction, the variance $\boldsymbol{\sigma}^2$ gives an estimate for it's uncertainty.

2.3 Reinforcement Learning

In Section 2.2, we assumed that we can learn the optimal signal timing plan using a dataset in which each traffic situation is labeled with the next best phase. Unfortunately, we usually do not have such a dataset, and the creation of one by human experts is extremely difficult, time-consuming, and error-prone due to the complexity of the problem. This section introduces *Reinforcement Learning (RL)*, a ML paradigm that attempts to circumvent this problem by allowing the signal controller to learn optimal behavior solely from the feedback it receives from interacting with its environment in a sequential decision process. In particular, this section focuses on *Deep Reinforcement Learning (DRL)*, which equips RL methods with ANNs to benefit from its function approximation capabilities. Unless otherwise noted, the information contained in this section is drawn from the RL textbook by Sutton and Barto [66].

2.3.1 Markov Decision Process

In RL sequential decision-making problems involving a single decision maker are typically modeled as a *Markov Decision Process (MDP)* [67], that operates in discrete time steps. This process involves an *agent*, i.e., a decision maker, and an *environment*. At each consecutive *time step t* of this process, the agent is presented with a representation of the current *state* $s_t \in \mathcal{S}$ of the environment from a set of possible states $\mathcal{S}$ (called *state space*). Given this state s_t, the agent must decide how to proceed by selecting an *action* $\phi_t \in \Phi$ from a set of available actions Φ (called *action space*). The agent's behavior in choosing actions is controlled by its *policy* $\pi(\phi_t|s_t) = p(\phi_t|s_t) : \Phi \times \mathcal{S} \to \mathbb{R}^+$, which describes a *Probability Density Function (PDF)* indicating how likely it is that the agent will perform action $\phi_t \in \Phi$ in a given state $s_t \in \mathcal{S}$. The agent's chosen action in turn changes the state of the environment. As a consequence, at the next time step $t+1$, the environment responds with a representation of its new state $s_{t+1} \in \mathcal{S}$ along with a scalar *reward* signal $r_{t+1} \in \mathbb{R}$, quantifying how good the transition (s_t, ϕ_t, s_{t+1}) was with respect to an overall goal. The *model* of the environment (also called the *dynamics* of the environment) is typically stochastic and unknown to the agent. It is defined by two functions: the *transition function* $T(s_{t+1}|\phi_t, s_t) = p(s_{t+1}|s_t, \phi_t) : \mathcal{S} \times \Phi \times \mathcal{S} \to \mathbb{R}^+$, a PDF defining how likely it is that the environment transitions to state s_{t+1} after the agent performs action ϕ_t in state s_t, and the *reward function* $R(s_t, \phi_t, s_{t+1}) : \mathcal{S} \times \Phi \times \mathcal{S} \to \mathbb{R}$, which outputs the corresponding reward signal after the transition. The transition function also shows that states in a MDP satisfy the *Markov property*, meaning that next states depend only on the previous state and the agent's action chosen in it, i.e., $p(s_{t+1}|s_t, \phi_t) = p(s_{t+1}|s_t, \phi_t, \ldots, s_0, a_0)$. Accordingly, states capture all relevant information about past transitions needed to predict the future. Based on the representation of the new state s_{t+1}, the agent must again choose an action ϕ_{t+1}, and the process continues. The number of time steps the MDP is potentially allowed to run within an *episode* (i.e., within one simulation of this process) define its *planning horizon*. In this work, we will consider only MDPs with infinite horizons (i.e., without a well-defined end), which also corresponds to the description of signaled intersections, that can potentially run infinitely long. An illustration summarizing the mechanics of a MDP is depicted in Figure 2.5.

An important point to consider in a MDP is that actions may have long-term consequences and often do not reveal their goodness immediately, but only after some time steps. Accordingly, the agent should not greedily direct its actions toward the immediate reward, but should also include the rewards of later time steps in his ac-

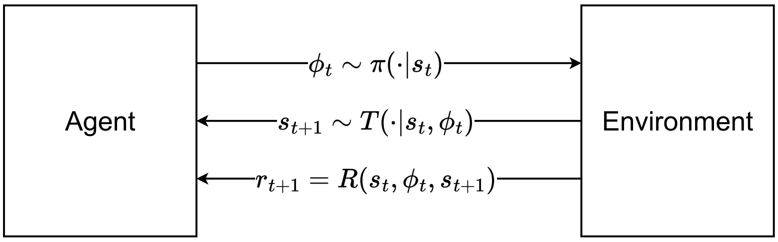

Agent
Environment
$\phi_t \sim \pi(\cdot|s_t)$
$s_{t+1} \sim T(\cdot|s_t, \phi_t)$
$r_{t+1} = R(s_t, \phi_t, s_{t+1})$

Figure 2.5: Markov Decision Process

tions. A quantity that summarizes future rewards starting at time step t is the *return* $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, which is defined as the discounted sum of future rewards starting at t, where $\gamma \in [0, 1)$ is a *discount rate* that geometrically discounts future rewards. Discounting rewards has two advantages. First, it guarantees that the sum of rewards is bounded in a MDP with an infinite time horizon. Second, it allows to express a bias towards recent rewards, where the degree of importance between immediate and future rewards can be adjusted by the discount rate γ.

Value functions take advantage of the return definition in order to quantify with a single number how good it is for an agent to be in a particular state or perform an action in a state, assuming the agent follows a certain policy. More specifically, while the *state-value function* $v_\pi(s_t) = \mathbb{E}_\pi[G_t|s_t]$ is defined as the expected return the agent receives when it acts according to policy π from state $s_t \in \mathcal{S}$, the *action-value function* $q_\pi(s_t, \phi_t) = \mathbb{E}_\pi[G_t|s_t, \phi_t]$ is defined as the expected return the agent receives if it starts in state $s_t \in \mathcal{S}$, performs action $\phi_t \in \Phi$, and then acts according to policy π. A fundamental property of value functions often exploited in RL is that they both satisfy self-consistency conditions, which means that they can be defined recursively. The corresponding equations, often called *Bellman equations* [67], are defined as follows:

$$v_\pi(s_t) = \mathop{\mathbb{E}}_{\substack{\phi_t \sim \pi \\ s_{t+1} \sim T}} \left[R(s_t, \phi_t, s_{t+1}) + \gamma v_\pi(s_{t+1})\right], \tag{2.18}$$

$$\begin{aligned} q_\pi(s_t, \phi_t) &= \mathop{\mathbb{E}}_{s_{t+1} \sim T} \left[R(s_t, \phi_t, s_{t+1}) + \gamma v_\pi(s_{t+1})\right] \\ &= \mathop{\mathbb{E}}_{s_{t+1} \sim T} \left[R(s_t, \phi_t, s_{t+1}) + \gamma \mathop{\mathbb{E}}_{\phi_{t+1} \sim \pi} \left[q_\pi(s_{t+1}, \phi_{t+1})\right]\right]. \end{aligned} \tag{2.19}$$

Both value functions are also related by means of the *action-advantage function*

$$a_\pi(s_t, \phi_t) = q_\pi(s_t, \phi_t) - v_\pi(s_t), \tag{2.20}$$

which quantifies how much better it is (i.e., how much more or less return to expect) when the agent takes action ϕ_t in state s_t rather than following policy π from s_t.

By letting $\mu(s_t|\pi)$ represent the underlying *stationary distribution* of the Markov chain when the agent follows policy π, one can define the objective of an MDP, which is to find the optimal policy π_* that maximizes the expected return in all possible states:

$$\pi_* = \operatorname*{argmax}_{\pi} J(\pi) = \operatorname*{argmax}_{\pi} \mathop{\mathbb{E}}_{s_t \sim \mu} \left[v_\pi(s_t) \right]. \tag{2.21}$$

The self-consistency conditions for the optimal state-value and action-value functions, denoted by $v_*(s)$ and $q_*(s,a)$, respectively, can each be expressed in a specialized form, facilitating the substitution of expectations over actions with the maximum over actions:

$$v_*(s_t) = \max_{\phi_t \in \Phi} q_*(s_t, \phi_t) = \max_{\phi_t \in \Phi} \mathop{\mathbb{E}}_{s_{t+1} \sim T} \left[R(s_t, \phi_t, s_{t+1}) + \gamma v_*(s_{t+1}) \right], \tag{2.22}$$

$$q_*(s_t, \phi_t) = \mathop{\mathbb{E}}_{s_{t+1} \sim T} \left[R(s_t, \phi_t, s_{t+1}) + \gamma \max_{\phi_{t+1} \in \Phi} q_*(s_{t+1}, \phi_{t+1}) \right]. \tag{2.23}$$

These equations, known as the *Bellman optimality equations* [67], reveal, that the optimal policy π_* is equivalent to always taking the action $\phi_t \in \Phi$ that maximizes the optimal action-value function $q_*(s_t, \phi_t)$ in any given state $s \in \mathcal{S}$. Consequently, for any MDP there must exist a deterministic policy $\pi(s_t) : \mathcal{S} \to \Phi$ that is optimal. This policy corresponds to a greedy policy with respect to the optimal action-value function:

$$\pi_*(s_t) = \operatorname*{argmax}_{\phi_t \in \Phi} q_*(s_t, \phi_t). \tag{2.24}$$

It should be noted that optimal deterministic policies may not be unique. For example, if in a given state $s_t \in \mathcal{S}$ two actions $\phi_t^{(1)} \in \Phi$ and $\phi_t^{(2)} \in \Phi$ have the same optimal action-value, i.e., $q_*(s_t, \phi_t^{(1)}) = q_*(s_t, \phi_t^{(2)})$, then both deterministic policies, either following action $\phi_t^{(1)}$ or action $\phi_t^{(2)}$, will be optimal. However any stochastic policy that follows action $\phi_t^{(1)}$ with probability $\beta \in [0,1]$ and action $\phi_t^{(2)}$ with probability $1-\beta$ will also be optimal.

Formulating TSC as a MDP

In the context of TSC, formulating the problem of adapting light signals as a MDP usually boils down to specifying a reward function, a representation of the state, and defining actions for an isolated signal controller that takes the role of the agent. The dynamics of the environment (i.e., its transition function) is usually taken over by a traffic simulator.

As far as the choice of reward function is concerned, a direct minimization of the average travel time would probably be the most desirable. However, it is problematic to convert the average travel time directly into a reward function, as it depends on the duration of individual vehicle trips, which can only be determined after the vehicles arrive, making an immediate estimate difficult [35]. Due to this reason it is more common to convert intersection-level metrics (see Section 2.1.2) into reward functions, as they can be easily measured locally at any point in time. Popular choices include the negative queue length [8, 22, 10, 14, 15, 68, 69], the negative waiting time [21] or the negative pressure [9, 24, 25, 13, 12, 70]. Some works also consider a weighted combination of multiple criteria [19, 26, 7, 17], although it has been found that the individual weights are difficult to tune [10, 9].

The choice of state representation is typically made in accordance with the type of information that can be captured by the detectors installed on site. Accordingly, the most common state representation includes traffic information (number, queue length, density, presence or average speed of vehicles) related to individual lanes [18, 19, 20, 21, 7, 8, 10, 11, 14, 15, 24, 25, 17, 70] or lane segments [71, 27, 22, 9, 12]. Others assume that more sophisticated detector devices are available and design a state representation that encode information about the position or speed of individual vehicles [26, 28, 7, 68, 69]. In addition to traffic-related information, it is also common to include information about the status of the signal configuration, e.g., by indicating which phase is currently active [71, 26, 27, 28, 7, 8, 22, 10, 11, 9, 13, 14, 15, 24, 25, 70] or the elapsed time of the current phase [18, 20, 68].

There are also several ways to define actions for a signal controller in the existing literature. Perhaps the most common variant is to have the action space correspond to the set of available phases [20, 19, 71, 26, 27, 8, 9, 11, 13, 15, 24, 25, 17, 12, 70, 69]. In this way, the signal controller can select any phase in each time step and thus adapt the sequence of phases to the current traffic conditions as required. Another option, which better meets the expectations of road users, is to run the phases in a cycle and have the signal controller make a binary decision as to whether extend the current phase or move to the next phase [18, 7, 22, 10, 68, 69]. In both cases, it is common to set a fixed duration for the execution of an action before the next action can be selected.

2.3.2 Value-based Methods

Motivated by the idea that the optimal policy π_* can be fully represented by the optimal action-value function q_* (see Equation 2.24), *value-based* RL algorithms learn a policy in an indirect way by learning a value function instead and then deriving a deterministic policy from it by acting greedily with respect to the learned value function. To learn the value function, an algorithmic principle known as *Generalized Policy Iteration (GPI)* is applied. The idea behind GPI is to gradually move a (randomly) initialized Q-function as a proxy for the action-value function q_π towards the optimal action-value function q_* by alternating between two steps, *policy evaluation* and *policy improvement*. While policy evaluation attempts to make the Q-function consistent with the current policy π, policy improvement makes the policy greedy with respect to the Q-function. If the policy evaluation step is exact, i.e., $Q = q_\pi$, then the succeeding policy improvement step yields a policy π' that is guaranteed to be better than or equivalent to π (in which case $\pi = \pi' = \pi_*$). This can be easily proven by the *policy improvement theorem*:

$$q\big(s_t, \pi'(s_t)\big) = q_\pi\big(s, \operatorname*{arg\,max}_{\phi_t \in \Phi} q_\pi(s_t, \phi_t)\big) = \max_{\phi_t \in \Phi} q_\pi\big(s_t, \phi_t\big) \geq q\big(s_t, \pi(s_t)\big), \forall s_t \in \mathcal{S}. \quad (2.25)$$

One popular instance of a value-based algorithm is *Q-learning* [72]. Here GPI is applied without explicit knowledge of the environment's model (i.e., its transition and reward function) and instead solely works with sampled transitions (i.e., sampled states, actions, and rewards) generated from the interaction between agent and environment. Accordingly, a policy improvement step in Q-learning corresponds to sampling a transition according to a policy π that is greedy with respect to the current Q-function, leaving some room for exploration. A policy evaluation step then uses the Bellman optimality equation of the action-value function (see Equation 2.23) to compute a *bootstrap estimate* of the sampled transition's action-value $\hat{q}_\pi$ and moves Q closer to it. Specifically, for each transition in which the agent started in state $s_t \in \mathcal{S}$, (greedily) chose action $\phi_t \sim \pi(\cdot|s_t)$, ended up in state $s_{t+1} \sim T(\cdot|\phi_t, s_t)$ and received reward $r_{t+1} = R(s_t, \phi_t, s_{t+1})$, which we will refer to as an *experience tuple* $(s_t, \phi_t, r_{t+1}, s_{t+1})$, the following update rule is applied in the policy evaluation step:

$$\begin{aligned} Q(s_t, \phi_t) &:= Q(s_t, \phi_t) + \eta\big[\hat{q}_\pi(s_t, \phi_t) - Q(s_t, \phi_t)\big], \\ \text{where } \hat{q}_\pi(s_t, \phi_t) &= r_{t+1} + \gamma \max_{\phi_{t+1} \in \Phi} Q(s_{t+1}, \phi_{t+1}). \end{aligned} \quad (2.26)$$

Here, the bootstrap estimate $\hat{q}_\pi(s_t, \phi_t)$ corresponds to a target and the term $\hat{q}_\pi(s_t, \phi_t) - Q(s_t, \phi_t)$ expresses how far $Q(s_t, \phi_t)$ is above or below this target. Thus, by shifting $Q(s_t, \phi_t)$ in the direction of this error term, scaled by a learning rate η, the update step is essentially an attempt to correct $Q(s_t, \phi_t)$. Although one step may not be exact, especially since the target $\hat{q}_\pi(s_t, \phi_t)$ is based in part on the learned Q-function itself, which may be inconsistent with q_π, it can still be shown that the Q-function converges to the optimal action-value function q_*, provided all state-action pairs are sufficiently explored.

A limitation of the update rule presented in Equation 2.26 is that the Q-function must be represented as a table whose rows correspond to states, columns to actions, and entries to the corresponding state-action values. As a consequence, the update rule is only applicable to MDPs with finite and discrete state and action spaces. To apply Q-learning in contexts with continuous state spaces, one can instead parameterize the Q-function by representing it as an ANN, which is commonly referred to as a *Deep Q-Network (DQN)* [73]. The parameters $\boldsymbol{\theta}$ of the DQN can then be trained based on a similar update rule that applies gradient descent to minimize the squared error between target $\hat{q}_\pi(s_t, \phi_t)$ and prediction $Q(s_t, \phi_t; \boldsymbol{\theta})$:

$$\begin{gathered}\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \big(\hat{q}_\pi(s_t, \phi_t) - Q(s_t, \phi_t; \boldsymbol{\theta})\big)^2, \\ \text{where } \hat{q}_\pi(s_t, \phi_t) = r_{t+1} + \gamma \max_{\phi_{t+1} \in \Phi} Q(s_{t+1}, \phi_{t+1}; \boldsymbol{\theta}).\end{gathered} \tag{2.27}$$

Although the use of a DQN potentially allows for generalization beyond the states observed during training, Q-learning and in particular the update rule in Equation 2.27 presents several challenges. Since the experience tuples between successive transitions are highly correlated and thus not i.i.d., and the fact that targets are constantly overestimated due to the $\arg\max$ operator and not stationary as they depend in part on the changing Q-function itself, the assumptions on which supervised neural network training is based (see Section 2.2.4) are severely violated, degrading the stability of gradient-based optimization. Additionally, the agent is constantly in an *exploration-exploitation dilemma*. That is, at each time step, the agent is confronted with the question of whether to exploit its current knowledge of the Q-function to obtain a high return (i.e., take the action that maximizes the output of the Q-function), or explore actions that deviate from this greedy policy and potentially lead to higher returns and thus a better policy.

To overcome these challenges and enable more stable and faster convergence, researchers made several improvements to Q-learning. One of these extensions is *experience replay* [73], an attempt to make the data better satisfy the i.i.d. assumption by storing experience tuples after transitions in a data set D, often called the *replay buffer*, from which experience tuples are sampled uniformly at random for update steps:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{(s_t, \phi_t, r_{t+1}, s_{t+1}) \sim D} \big(\hat{q}_\pi(s_t, \phi_t) - Q(s_t, \phi_t; \boldsymbol{\theta})\big)^2. \tag{2.28}$$

Another improvement is to use *fixed Q-targets* [74]. Here, two DQNs with identical architecture are used, an *online DQN* and a *target DQN*. The targets $\hat{q}_\pi(s_t, \phi_t)$ for updating the parameters $\boldsymbol{\theta}$ of the online DQN are then computed based on the target DQN, whose parameters $\boldsymbol{\theta}^-$ in turn lag behind the parameters of the online DQN:

$$\hat{q}_\pi(s_t, \phi_t) = r_{t+1} + \gamma \max_{\phi_{t+1} \in \Phi} Q(s_{t+1}, \phi_{t+1}; \boldsymbol{\theta}^-). \tag{2.29}$$

In this way, the Q-targets remain temporarily stationary between update steps, which improves the stability of the training process. The parameters of the target DQN can be updated in several ways. In the original work, they are set to the parameters of the online DQN every k gradient update steps, where k is a hyperparameter. Another method, first utilized by Lillicrap et al. [75] in the context of Q-learning, is *Polyak averaging* [76]. Here the target DQN is updated at each step based on a weighted average between the parameters of the online DQN and the old parameters of the target DQN:

$$\boldsymbol{\theta}^- := \tau \boldsymbol{\theta} + (1 - \tau) \boldsymbol{\theta}^-. \tag{2.30}$$

Here, $\tau \in [0, 1]$ is a hyperparameter that scales how much the parameters of the online DQN are mixed with the parameters of the target DQN.

To address the overestimation of Q-targets, the idea of two DQNs can be further extended by an approach known as *double Q-learning* [77]. Here, the Q-targets are computed by first determining the action corresponding to the maximum value with respect to the online DQN, and then evaluating the value of that action with respect to the target DQN:

$$\hat{q}_\pi(s_t, \phi_t) = r_{t+1} + \gamma Q\big(s_{t+1}, \operatorname*{arg\,max}_{\phi_{t+1} \in \Phi} Q(s_{t+1}, \phi_{t+1}, \boldsymbol{\theta}); \boldsymbol{\theta}^-\big). \tag{2.31}$$

Another improvement is an architectural adaptation known as *dueling DQN* [78]. This method takes advantage of the fact that the action-value function can be decomposed into a sum of the state-value function and the action-advantage function (see Section 2.3.1). To exploit this property, after a series of shared layers in the ANN, the dueling architecture branches out into two streams: a V-stream for the state-value and an A-stream for the action-advantage. The final Q-function is then recovered in the following way:

$$Q(s_t, \phi_t; \boldsymbol{\theta}) = V(s_t; \boldsymbol{\theta}) + A(s_t, \phi_t; \boldsymbol{\theta}) - \frac{1}{|\Phi|} \sum_{\phi_{t'} \in \Phi} A(s_t, \phi_{t'}; \boldsymbol{\theta}). \tag{2.32}$$

The primary advantage of decomposing the Q-function in this manner lies in the shared utilization of the V-stream for all actions. This not only enhances generalization capabilities but also contributes to the stability and efficiency of the training process, as empirically observed by the authors.

The exploration-exploitation dilemma can be approached in several ways. One common technique is to use an *ϵ-greedy policy*, where the agent chooses a random action with probability ϵ (exploration) and the greedy action (that maximizes the Q-function) with probability $1 - \epsilon$ (exploitation). Although for tabular Q-learning it is proven that this method converges to the optimal policy as long as epsilon decreases over time and eventually approaches 0, in practice it is often difficult to adjust ϵ during training to achieve optimal performance. Another idea is to relate the need for exploration to the uncertainty of the DQN's predictions. An approach that follows this idea is MC dropout [64] (see Section 2.2.5). Here, the authors propose to use *Thompson sampling* [79] to select actions by always choosing the greedy action from the Q-values that result from a single stochastic forward pass through the dropout DQN. Consequently, the lower the confidence in the Q-value predictions, the more likely the agent will explore, and the higher the confidence, the more likely it will exploit. This method can therefore be viewed as an automatic approach to balance exploration and exploitation. The authors also demonstrate that this strategy can lead to faster convergence compared to a well-tuned ϵ-greedy policy.

2.3.3 Policy Gradient Methods

Thus far, this work only considered learning a policy in an indirect way via a corresponding value function. Although this is a valid strategy to find an optimal policy, value-based methods have several limitations. For example, they can only learn deterministic policies, which makes exploration difficult and can be problematic in situations where the state of the environment is only partially observable to the agent, in which case the optimal policy may in fact be stochastic. Moreover, to be able to find optimal actions with respect to the action-value function, which requires the computation of an arg max over all actions, value-based methods are only suitable for MDPs with discrete and finite action spaces.

Policy gradient methods [80] attempt to tackle these limitations by learning a policy directly. This is done by representing the policy as a parameterized function $\pi(\phi_t|s_t;\boldsymbol{\theta})$ and finding its parameters $\boldsymbol{\theta}$ that maximize the expected return over all states:

$$\boldsymbol{\theta}_* = \arg\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{s_t \sim \mu}\left[v_\pi(s_t)\right], \tag{2.33}$$

In this way, the learned policy can be stochastic, making explicit exploration strategies like ϵ-greedy obsolete (as actions can be randomly sampled from the parameterized policy), and during training the stochastic policy can gradually converge to a deterministic policy if necessary. Moreover, the policy can take any parametric form that also allows for continuous actions.

To find the parameters $\boldsymbol{\theta}$ of the policy that maximizes the expected return, one can simply apply *gradient ascent*[3]:

$$\boldsymbol{\theta} := \boldsymbol{\theta} + \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}). \tag{2.34}$$

One issue with the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ however, is that its analytical computation is not straightforward as the stationary distribution μ depends not only on the agent's policy π, but also on the environment's model which we assume to be unknown. Fortunately, one can make use of the *policy gradient theorem* [80]:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{s_t \sim \mu} \left[v_\pi(s_t) \right] \propto \mathop{\mathbb{E}}_{\substack{s_t \sim \mu \\ \phi_t \sim \pi}} \left[q_\pi(s_t, \phi_t) \nabla_{\boldsymbol{\theta}} \log \pi(\phi_t | s_t; \boldsymbol{\theta}) \right]. \tag{2.35}$$

This theorem shows that the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ can be conveniently reformulated into an expression that does not involve a derivative of the distribution μ. Specifically, it can be seen that the gradient is proportional to the expected log policy gradient $\nabla_{\boldsymbol{\theta}} \pi(\phi_t | s_t; \boldsymbol{\theta})$, scaled by the corresponding action-value $q_\pi(s_t, \phi_t)$. This new expression can also be leveraged to explain intuitively what happens during gradient ascent. Since the policy is a PDF, the log policy gradient essentially defines a *score function* that outputs a vector pointing in the direction in parameter space that most increases the density for action ϕ_t in state s_t. Accordingly, the higher the value $q_\pi(s_t, \phi_t)$ of an action ϕ_t in a state s_t, the more the gradient ascent step will increase the density in the corresponding region of the policy, and the more the policy will favor action ϕ_t in state s_t.

An important consequence of the policy gradient theorem is that the expected value in the reformulated expression can now be estimated from sampled experience. One possibility to do this is proposed in the *REINFORCE* algorithm [81]. Here entire episode trajectories $(s_0, a_0, s_1, s_2, \ldots)$ are sampled and the experienced returns G_t of each state-action pair (s_t, ϕ_t) are used to scale the log policy gradient $\nabla_{\boldsymbol{\theta}} \log \pi(\phi_t | s_t, \boldsymbol{\theta})$, yielding the following gradient ascent update rule:

$$\boldsymbol{\theta} := \boldsymbol{\theta} + \eta G_t \nabla_{\boldsymbol{\theta}} \log \pi(\phi_t | s_t; \boldsymbol{\theta}). \tag{2.36}$$

---

[3] Applying gradient ascent to $J(\boldsymbol{\theta})$ is equivalent to applying gradient descent to $-J(\boldsymbol{\theta})$

This approach guarantees to give an unbiased estimate of the gradient. However, due to the accumulation of multiple random events along the trajectory, the estimate suffers from high variance. To reduce the variance, a common technique is to subtract a bias $b(s_t) : \mathcal{S} \rightarrow \mathbb{R}$ from the sampled return G_t:

$$\boldsymbol{\theta} := \boldsymbol{\theta} + \eta(G_t - b(s_t))\nabla_{\boldsymbol{\theta}} \log \pi(\phi_t | s_t; \boldsymbol{\theta}). \tag{2.37}$$

As long as this bias does only depends on a state $s_t \in \mathcal{S}$, it is guaranteed that the estimator is still unbiased. A natural choice for the bias is an estimate of the state-value $\hat{v}_\pi(s_t)$ which leads to an update rule in which the log policy gradient is instead scaled by an estimate of the action-advantage $\hat{a}_\pi(s_t, \phi_t) = G_t - \hat{v}_\pi(s_t)$:

$$\boldsymbol{\theta} := \boldsymbol{\theta} + \eta \hat{a}_\pi(s_t, \phi_t) \nabla_{\boldsymbol{\theta}} \log \pi(\phi_t | s_t; \boldsymbol{\theta}). \tag{2.38}$$

The variance can be further reduced by allowing for some bias and computing a bootstrap estimate of the action-value (or action-advantage) using a simultaneously learned value function. This approach, known as *actor-critic* [82], lies at the intersection of value-based and policy-based methods. While the actor, i.e., the parameterized policy, is responsible for performing actions (policy-based), the critic, i.e., the parameterized value function, evaluates these actions and suggests a direction to update the actor's parameters (value-based). As a consequence of bootstrapping, actor-critic approaches are also suitable for continuous updates, i.e. they allow one update per transition, which also enables their application to tasks with infinite horizons (e.g. TSC). A popular choice is to use *Advantage Actor-Critic (A2C)*. In this variant, a parameterized state-value function $V(s_t, \boldsymbol{\theta}_c)$ is used as a critic to compute a bootstrap estimate of the action-advantage, i.e., $\hat{a}_\pi(s_t, \phi_t) = r_{t+1} + \gamma V(s_{t+1}; \boldsymbol{\theta}_c) - V(s_t, ; \boldsymbol{\theta}_c)$ after each transition. This estimate is then used to update the parameters $\boldsymbol{\theta}$ of the actor following the update rule defined in Equation 2.38. The parameters $\boldsymbol{\theta}_c$ of the critic, can in turn be updated using gradient descent to minimize the squared error between the bootstrap estimate of the state-value target $\hat{v}_\pi(s_t) = r_{t+1} + \gamma V(s_{t+1}; \boldsymbol{\theta}_c)$ and the prediction $V(s_t; \boldsymbol{\theta}_c)$:

$$\boldsymbol{\theta}_c := \boldsymbol{\theta}_c - \eta \nabla_c \big(\hat{v}_\pi(s_t) - V(s_t; \boldsymbol{\theta}_c)\big)^2. \tag{2.39}$$

To further stabilize training, one can take advantage of parallelization by distributing the work among multiple workers, each of whom runs an independent episode. Updates can then be done either *asynchronously*, by updating the actor and critic parameters as soon as a worker collected an experience tuple [83], or *synchronously*, by waiting until all workers have collected an experience tuple, combining the experience tuples into a batch, and then performing a single update using the batch [84]. Using experience tuples from multiple independently running environments for update steps effectively decorrelates the data for update steps, which has a similar effect to a replay buffer in Q-learning (see Section 2.3.2).

2.3.4 Cooperative Markov Game

Until now, the sole focus of this work was on single-agent sequential decision problems. Yet, real-life problems often feature multiple agents that must coordinate their actions for a shared goal. This is also true for multiple signalized intersections in a road network, coordinating their signal timing plans to facilitate efficient traffic flow.

In such multi-agent settings, the MDP framework can be extended to a *Markov Game (MG)* [85] (also known as a *stochastic game* [86]), involving a set of agents $\mathcal{I}$ acting simultaneously in a shared environment. Every time step t, each agent $i \in \mathcal{I}$ observes a representation of the environment's current state $s_t \in \mathcal{S}$ and decides for an action $\phi_t^i \in \Phi^i$ from its own action space Φ^i using its policy $\pi^i(\phi_t^i|s_t) : \Phi^i \times \mathcal{S} \rightarrow \mathbb{R}^+$. The *joint action* $\boldsymbol{\phi}_t \in \Phi$ of all agents, where $\Phi = \times_{i\in\mathcal{I}}\Phi^i$ denotes the *joint action space*, in turn induces the environment to transition to a new state s_{t+1} according to its transition function $T(s_{t+1}|\boldsymbol{\phi}_t, s_t) : \mathcal{S} \times \Phi \times \mathcal{S} \rightarrow \mathbb{R}^+$. This transition also results in a reward signal r_{t+1}^i that is individually determined for each agent $i \in \mathcal{I}$ based on its corresponding reward function $R^i(s_t, \boldsymbol{\phi}_t, s_{t+1}) : \mathcal{S} \times \Phi \times \mathcal{S} \rightarrow \mathbb{R}$.

In this work, the focus is on *cooperative* MGs [87] where agents have an incentive to work together to achieve a team goal which can be expressed by a *global reward function* $R(s_t, \boldsymbol{\phi}_t, s_{t+1})$. In a *fully cooperative* MG, the global reward is equivalent to the individual reward functions of the agents, i.e., $R(s_t, \boldsymbol{\phi}_t, s_{t+1}) = R^1(s_t, \boldsymbol{\phi}_t, s_{t+1}) = \ldots = R^{|\mathcal{I}|}(s_t, \boldsymbol{\phi}_t, s_{t+1})$. Alternatively, it can be formulated as a *team-average reward* from the agents' individual reward functions, i.e., $R(s_t, \boldsymbol{\phi}_t, s_{t+1}) = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} R^i(s_t, \boldsymbol{\phi}_t, s_{t+1})$. While the former entails aligned goals for individual agents, the latter allows agents to pursue distinct objectives that contribute to the overall team benefit.

Since individual and global rewards are functions of joint actions, value functions also depend on the *joint policy* $\boldsymbol{\pi} = \{\pi^i | i \in \mathcal{I}\}$ of all agents. Therefore, the *individual state-value function* $v_{\boldsymbol{\pi}}^i(s_t) = \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_{k=0}^{\infty}\gamma^k R^i(s_{t+k}, \boldsymbol{\phi}_{t+k}, s_{t+k+1})|s_t\right]$ of an agent $i \in \mathcal{I}$ outputs the expected discounted sum of its individual rewards from state $s_t \in \mathcal{S}$ assuming the team acts according to joint policy $\boldsymbol{\pi}$. Similarly, the *global state-value function* $v_{\boldsymbol{\pi}}(s_t) = \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_{k=0}^{\infty}\gamma^k R(s_{t+k}, \boldsymbol{\phi}_{t+k}, s_{t+k+1})|s_t\right]$ defines the expected discounted sum of global rewards the agents receive as a team when they are currently in state $s_t \in \mathcal{S}$ and act according to joint policy $\boldsymbol{\pi}$, thereafter.

The goal of a cooperative MG can be formulated as finding an optimal joint policy $\boldsymbol{\pi}_*$ that maximizes the expected discounted sum of global rewards in all possible states:

$$\boldsymbol{\pi}_* = \underset{\boldsymbol{\pi}}{\arg\max} \underset{s_t \sim \mu}{\mathbb{E}} v_{\boldsymbol{\pi}}(s_t). \tag{2.40}$$

Here, $\mu(s_t|\boldsymbol{\pi})$ denotes the stationary distribution of the Markov chain, when the agents follow joint policy $\boldsymbol{\pi}$.

2.3.5 Independent Learning

A popular strategy for learning a joint policy in a cooperative MG is *independent learning* [88]. Here, the task of learning the joint policy $\boldsymbol{\pi}$ is reduced to $|\mathcal{I}|$ single-agent learning problems, with each agent $i \in \mathcal{I}$ learning its own policy π^i that maximizes its individual long-term reward r^i. From the perspective of a single agent, the other agents are treated simply as part of the environment. In contrast to a centralized agent that learns the joint policy directly, the independent learning approach offers several advantages. For example, the approach is more scalable because each agent only needs to learn a policy for its individual action space and can ignore the joint action space, which grows exponentially with the number of agents. In addition, each agent has full autonomy over its actions during and after learning and does not need to negotiate its decisions with other agents, thus eliminating the need for inter-agent communication. Despite its advantages, a key challenge in independent learning is that the environment does not appear stationary from the perspective of a single agent, as each agent adapts its policy during training. Accordingly, the convergence guarantees that normally apply to single-agent RL algorithms no longer hold. Although this problem exists, it has been observed several times in the context of TSC that it does not seriously affect training and can still lead to competitive signal control policies [22, 9, 13, 11]. For this reason, methods that address the non-stationarity problem are not discussed further here. Interested readers can find a detailed elaboration of this problem and solution concepts in [89].

2.3.6 Domain Randomization

Using the RL techniques discussed in this section, it would theoretically be possible to train agents directly in the real world. However, since agents are usually not intelligent at the beginning of training, they can make poor decisions that can lead to problems in the real world. For this reason, agents are usually trained in a simulator before being deployed in the real world. One problem with such simulators is that they can deviate from reality, making the transfer from the *source domain* (i.e., the simulator) to the *target domain* (i.e., the real world) difficult.

One strategy for bridging this reality gap is *Domain Randomization (DR)* [30]. The basic idea of this approach is to ensure sufficient simulated variability in the source domain during training, so that after deployment, the target domain simply appears to the agent as another variant to which its policy generalizes. In a sense, the discrepancy between the source and target domains is modeled as variability in the source domain [31]. To model this degree of variability in the source domain, DR configures the behavior of the environment based on a configuration ξ of *domain parameters* that can be easily drawn from a random space Ξ. For example, in a robotics task where a robot is responsible for grasping certain objects, these domain parameters could correspond to the color, position, shape or material texture of those objects, and the lightning conditions, which in turn could be sampled uniformly from a bounded range [30]. In each training episode, a new configuration $\xi \sim \Xi$ is then sampled and the agent is trained in the resulting environment. By letting $\mu(s_t|\pi,\xi)$ denote the stationary distribution of the Markov chain when the environment is configured based on $\xi \in \Xi$ and the agent follows policy π, the agent's goal is then to find the optimal policy π_* that maximizes the expected return in all states $s_t \in \mathcal{S}$ of all possible environmental configurations $\xi \in \Xi$:

$$\pi_* = \arg\max_{\pi} J(\pi) = \arg\max_{\pi} \mathop{\mathbb{E}}_{\xi\sim\Xi}\left[\mathop{\mathbb{E}}_{s_t\sim\mu}\left[v_\pi(s)\right]\right]. \tag{2.41}$$

Accordingly, the agent must learn a policy that generalizes across different environments and is robust to the variations caused by DR.

3

Related Work

The body of research on TSC is extensive, dating back nearly half a century [40]. This chapter reviews some of the solutions that have been developed over the years, including methods based solely on Transportation Engineering (TE) theories and solutions derived from RL. Since the focus of this thesis is on the transferability of signal controllers to new traffic conditions, the extent to which such solutions meet this goal is examined.

3.1 Transportation Engineering Solutions

A common solution for TSC is to view it as an optimization problem by designing a mathematical model of the traffic environment using conventional TE theories and finding a closed-form solution based on that model. In the past, numerous researchers have developed TSC solutions in this way. Perhaps the oldest was developed by Webster [40], who demonstrated a way to determine a fixed cycle length and phase split of an isolated cycle-based signal controller to minimize delay. Building on Webster's findings, several methods have been developed to determine the offset of cyclic signal schedules between adjacent intersections, each sharing the same cycle length, to enable signal progression along arterial roadways. These include the GreenWave method [41], which determines the offsets for one direction of the arterial based on average speed and distance between intersections, which has been shown to minimize delay along the considered direction. MAXBAND [90], another progression-based solution, additionally considers traffic coming from the opposite direction and uses a mixed-integer linear programming model to derive the offsets that maximize the green time bandwidth of the arterial, i.e., the portion of a signal cycle in which a platoon of vehicles can pass through the arterial in both directions without stopping. To make cycle-based signal controllers in a road network more responsive to real-time traffic conditions, researchers have also worked on adaptive TSC systems that can adjust signal timings online. Among them are systems like SCATS [91] or SCOOT [92] that enable the adjustment of the cycle length, phase split as well as the offsets based on traffic measurements from the recent past. More recently, Max Pressure control [39] has gained traction among researchers. Here, each signalized intersection is independently controlled based on a greedy algorithm that simply selects the phase with the highest pressure in each period. Using a store-and-forward queuing model, it can be shown that this type of control maximizes the throughput of the entire road network and leads to an even distribution of vehicles in the system [39].

Although TE solutions produce good results in theory, one problem is that their theoretical guarantees hold only if the assumptions of the traffic models on which they are based are satisfied. However, assumptions such as uniform traffic [40, 41, 90] or unlimited vehicle storage capacity of lanes [39] are difficult or even impossible to observe in reality, which is why such solutions are not optimal in practice, especially when traffic demand is high and fluctuates significantly.

3.2 Reinforcement Learning Solutions

Compared to classical TE solutions, RL algorithms do not need to make simplifying assumptions about the traffic environment and instead allow learning a signal control policy from feedback received during interaction with the real traffic environment. This promising idea has motivated many researchers to explore the potential of RL in TSC. There is already a large body of work showing that a learned signal controller often performs significantly better on simulated scenarios than solutions derived from TE theories [20, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17].

Despite this potential, however, many of these solutions fail if the conditions under which they were trained change at test time. There are two reasons for this. The first is a modeling problem. Many approaches represent the policy through a model that only allows fixed size sate representations and action definitions [18, 19, 20, 21, 7, 22, 10, 9, 23, 14, 24, 25, 17]. However, since the way intersections are constructed (i.e., the number of approaches, lanes, traffic movements, phases, etc.) may vary from intersection to intersection, requiring the state representation and the action definition to vary as well, the reusability of such solutions is limited to homogeneous intersections. Consequently, individual training is required for each specific intersection, which is very inefficient and counterproductive to the goal of a universal TSC policy. The second problem concerns training. Many of the existing solutions are trained and tested for the same scenario usually involving a fixed road network structure and a fixed spatio-temporal traffic demand pattern [20, 26, 27, 28, 21, 8, 9, 23, 10, 14, 24, 25, 16, 17]. Between training episodes, this scenario is either not changed at all or is changed only slightly by randomly selecting the origin/destination or departure time of the vehicles. Strong deviations from the road network structure or the general traffic demand pattern during training remain unconsidered in these approaches. Accordingly, TSC policies are only partially trained as only a limited region of the state space is considered during training. Hence, while these methods exhibit notable efficacy in the scenarios for which they were trained, they frequently undergo substantial performance degradation when faced with conditions divergent from their training environment, as evidenced by experimental observations [11, 29]. In real-world scenarios, traffic typically follows predictable time-of-day patterns, and the road network usually stays consistent. Nevertheless, exceptions abound, like sudden and concentrated increases in demand during events such as concerts or road closures due to maintenance. An effective TSC policy should be capable of handling such exceptions without requiring expensive retraining, which would be impractical in real-world applications.

In recent years, researchers have proposed several new model architectures in order to improve the reusability and transferability of TSC policies [11, 12, 15, 29, 68, 69]. The basic idea of all these approaches is to represent the state of an intersection as a heterogeneous graph, whose nodes represent different components of the intersection (e.g. lanes, traffic movements, phases, etc.), while edges are used to describe the relationships between components. The policy is then implemented as a sequence of neural message passing layers to predict the next phase based on the graph's feature representations and structure[1]. In this way the policy can flexibly adapt to any intersection type, regardless of how many lanes, traffic movements or phases the intersection has. Moreover, the permutation invariance property of the neural message passing layers allows the results of such policies to be invariant with respect to symmetric cases such as mirrored or rotated traffic at intersections [11]. In contrast to previous solutions, the policy therefore does not need to learn all possible variants how traffic might approach the intersection. Instead, it is sufficient to train the policy on specific cases, and it automatically generalizes to possible symmetries of these cases (e.g., a policy trained only on traffic coming from one direction works just as well if the traffic is coming from the opposite direction at test time). As a result, the exploration

---

[1] It should be noted that [11, 12, 22] do not explicitly mention neural message passing, but the operations performed by these works can be implemented as neural message passing layers.

effort is greatly reduced, which in turn leads to faster convergence during training and ultimately makes it easier for the learned policy to transfer its knowledge to new traffic conditions. Despite these advantages, none of the mentioned methods adequately address the loss of node identity introduced by neural message passing. For instance, [11, 12, 15, 29] represent a lane node using the number of vehicles without considering variations in lane lengths. This oversight can hinder the policy's ability to implicitly learn crucial properties like lane capacity when lane lengths differ. Explicitly incorporating lane length as an additional node feature, as suggested by [68, 69], may help, but questions arise regarding the policy's extrapolation to lane lengths beyond the training set. Additionally, this approach can introduce stability issues during optimization and slow down training if the scale of lane length is not appropriately normalized in relation to other lane features. Apart from that, the node and edge features stored in the graph-structured state representation seem to be selected without clear rationale. Ideally, the state representation should only include information necessary for maximizing long-term rewards. Not meeting this condition may slow down training and lead to a drop in the final policy's performance, particularly when using features that do not contribute to optimization, as observed by some researchers [10, 9]. In recent years, researchers have undertaken numerous endeavors to enhance the alignment of reward functions and state representations for intersections, aiming to establish additional theoretical guarantees for both isolated [10] and coordinated [22, 9] TSC settings. One notable approach is PressLight [9], which applies Max Pressure control theory [39] to formulate a reward function that, when maximized, provably enhances network throughput. PressLight also employs a vector-valued state representation, ensuring the provision of sufficient information for optimization. However, existing studies, to the best found knowledge, have not delved into the adaptation of these theoretically motivated problem formulations for a graph-structured state representation. Such an adaptation would facilitate generalization to diverse intersection types while leveraging the associated theoretical guarantees.

Compared to adapting the model architecture of the policy, the exploration of more sophisticated training algorithms to enhance transferability has received less attention in the existing literature. Some studies propose pre-training methods that enable efficient policy transfer to new traffic scenarios in a few-shot learning manner. For instance, Wei et al. [22], apply a form of curriculum learning [93] by gradually increasing the complexity of the traffic scenario for which the policy is trained. Xiong et al. [94] use imitation learning by first training the policy using demonstrations generated from a TE-based TSC solution, which is then followed by a fine-tuning phase in which the policy learns from its own actions. Zang et al. [15] learn a useful parameter initialization for the policy by applying model-agnostic meta-RL [95] with different isolated intersection structures and phase settings as tasks. Extending this approach, Zhang et al. [70] increase the diversity of tasks using random traffic flows generated by a trained Wasserstein Generative Adversarial Network (WGAN) [96]. While these methods significantly reduce the training required for transferring a signal control policy, they do not entirely eliminate the need for additional training. An alternative training approach to obviate the need for fine-tuning is proposed by Devailly et al. [68, 69]. To enable zero-shot transfer, they advocate training the policy on randomly resampled road networks in each episode, representing a preliminary attempt to apply DR in the context of TSC. Experimentally, they demonstrate that this training methodology enables the learned policy's transfer to traffic scenarios not encountered during training, exhibiting competitive performance compared to policies specifically trained on these instances. However, in contrast to the road network, they largely avoid randomizing traffic within these networks during training, consistently employing the same vehicle departure rate within and across episodes. This approach introduces a risk of policy memorization, wherein the model overly relies on patterns associated with a constant departure rate, potentially leading to overfitting. Moreover, maintaining a constant departure rate limits the variety of traffic conditions the agents may encounter and learn from during training, thereby constraining the exploration capacity of the agents. These concerns may ultimately result in a suboptimal policy lacking the intelligence and robustness needed to effectively adapt to certain unknown traffic scenarios.

4

Method

The discussion in Chapter 3 has highlighted that while current RL-based TSC solutions demonstrate effectiveness in specific scenarios, there remains room for improvement to enhance the policy's ability to transfer to traffic scenarios not encountered during training. To address these limitations, this thesis proposes several improvements, elaborated upon in more detail in this chapter. First, a new formulation of the TSC problem is proposed that is more theoretically sound but still supports RL agents with maximum flexibility in adapting and generalizing to arbitrary intersection structures and traffic conditions. Based on the resulting state and action definition of this formulation, a new ANN architecture for the policy, called TransferLight, is introduced that fulfills this flexibility and can be implemented with minor adaptations for both a value-based Q-learning and a policy-based A2C algorithm. Lastly, to increase the variability of the traffic conditions used for training the agents, a DR mechanism is proposed that randomizes not only the road network but also the traffic.

4.1 Reinforcement Learning Problem Formulation

The formulation of the TSC problem as a MDP in an isolated setting or as a cooperative MG in a coordinated setting plays a crucial role in the success of RL algorithms in finding effective signal control policies. In this context, PressLight [9] introduces a definition for rewards, states, and actions for individual intersections, provably facilitating the learning of a policy that maximizes network throughput. However, this method is specifically applicable to homogeneous intersections, necessitating the training of individually parameterized policies for each intersection type. To extend the applicability of this method to heterogeneous intersections, such that training a single parameterized policy suffices, this thesis proposes several adjustments to the existing approach, ensuring a unified training framework for all policies in this study.

4.1.1 Reward Function

Identical to PressLight, in this work the reward function r^i of an intersection $i \in \mathcal{I}$ is defined as its negative pressure:

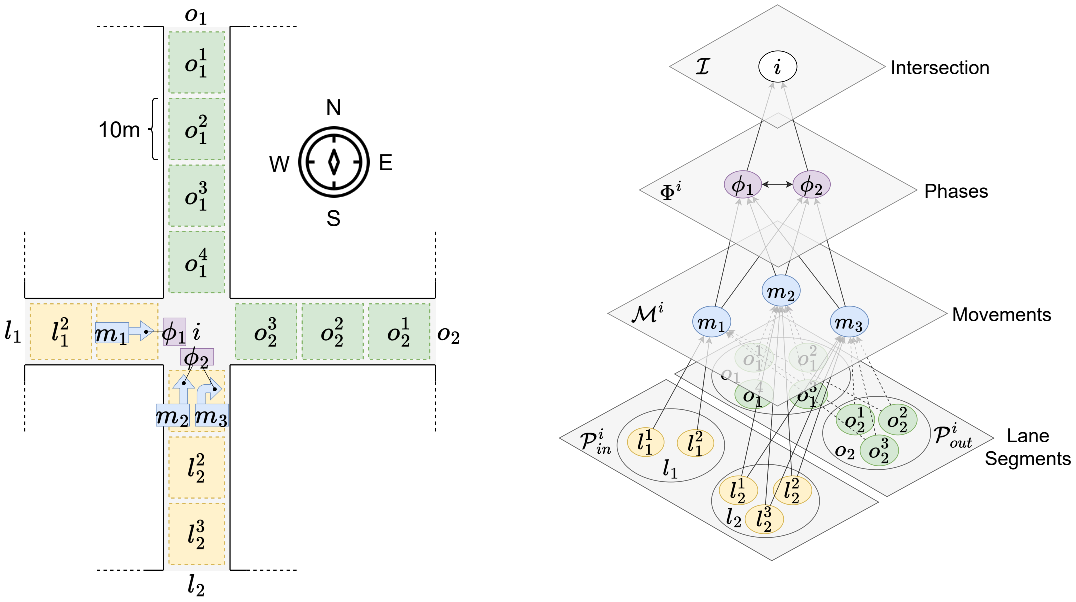
$$r^i = -\Big| \sum_{(l,o)\in\mathcal{M}^i} \frac{N(l)}{N_{max}(l)} - \frac{N(o)}{N_{max}(o)} \Big|. \tag{4.1}$$

It should be noted that, compared to the definition of pressure presented in Section 2.1.2, PressLight divides the number of vehicles $N(l)$ in each entering or exiting lane l by the number of vehicles $N_{max}(l)$ the lane can accommodate. Accordingly, the pressure of an intersection is defined as the absolute sum of the differences between the entering and exiting vehicle densities of each traffic movement the intersection provides. From an intuitive point of view, this adjustment ensures that the limited vehicle capacity of lanes is incorporated into the objective.

Employing this reward definition offers several advantages. Firstly, it ensures the maximization network throughput, even when each intersection independently maximizes its local reward, as demonstrated by the authors. Secondly, under mild assumptions, this approach also results in the minimization of travel time. Thirdly, the authors provide experimental evidence demonstrating that this reward definition outperforms other commonly used reward functions, such as total queue length, and markedly reduces the average travel time in comparison.

4.1.2 State Representation

To facilitate optimization, PressLight introduces a state representation s_t^i for each intersection $i \in \mathcal{I}$, corresponding to the environment's state $s_t \in \mathcal{S}$. This representation comprises the current phase $\phi \in \Phi^i$, the number of vehicles $N(o)$ on each outgoing lane $o \in \mathcal{L}_{out}^i$, and the vehicle count $N(l^k)$ on each segment l^k of every incoming lane $l \in \mathcal{L}_{in}^i$. Here, each incoming lane $l \in \mathcal{L}_{in}^i$ is discretized into K equally sized segments $l^1, \ldots, l^K$, where l^1 corresponds to the segment closest to the intersection. The authors provide a formal proof justifying the information contained in this representation as sufficient for describing the dynamics of the environment to maximize the long-term reward for each intersection $i \in \mathcal{I}$. However, to predict action-values, the authors combine this information in a fixed-size vector, which is then processed by a DQN consisting exclusively of fully connected layers. As argued in Section 3.2, this approach



(a) Example intersection

(b) Corresponding state representation

Figure 4.1: An example intersection and its corresponding graph-based state representation

limits reusability to homogeneous intersections. If the intersection structure changes, the DQN has to be trained from scratch, which prevents transferability.

To remedy this, this work proposes to store the same information in a heterogeneous graph, akin to previous works [11, 12, 15, 29, 68, 69]. To understand the structure of this graph, consider the signalized intersection $i \in \mathcal{I}$ displayed in Figure 4.1(a). It can be seen that the intersection comprises two incoming lanes $l_1, l_2 \in \mathcal{L}_{in}^i$ and two outgoing lanes $o_1, o_2 \in \mathcal{L}_{out}^i$. The intersection generally allows three traffic movement $m_1, m_2, m_3 \in \mathcal{M}^i$: the eastbound through movement $m_1 = (l_1, o_2)$ from lane l_1 to lane o_2, the northbound through movement $m_2 = (l_2, o_1)$ from lane l_2 to lane o_1 and the northbound right-turn movement $m_3 = (l_2, o_2)$ from lane l_2 to lane o_2. The right-of-way is controlled by two phases $\phi_1, \phi_2 \in \Phi^i$. While phase ϕ_1 permits the eastbound movement m_1 and prohibits the northbound movements m_2 and m_3, phase ϕ_2 permits the northbound movements m_2 and m_3 and prohibits the eastbound movement m_1. Similar to PressLight, lanes are segmented. However, instead of dividing each lane into a fixed number of segments K (as proposed in PressLight), here lanes are divided into segments of a fixed length of 10 meters (except the final segment which may be smaller). The advantage of this adjustment is that the length of each segment does not need to be encoded explicitly into the feature of the respective node. Instead, it can be implicitly learned by the policy along with other features related to the length, such as segment capacity. In addition, rather than solely segmenting incoming lanes, the same approach is applied to segment outgoing lanes, thereby facilitating a finer level of detail regarding vehicle distributions on the outgoing lanes. To ease the notation for later reuse, the set of segments associated with lanes incoming to an intersection $i \in \mathcal{I}$ will be denoted by $\mathcal{P}_{in}^{i}$, while the set of segments associated with outgoing lanes from i will be denoted by $\mathcal{P}_{out}^{i}$ ($\mathcal{P}$ for part). Similarly, the set of segments of a lane $l \in \mathcal{L}^{i}$ is denoted by $\mathcal{P}^{l}$.

The corresponding structure of the graph $G = (\mathcal{V}, \mathcal{E})$ describing the state of intersection $i \in \mathcal{I}$ is displayed in Figure 4.1(b) next to it. As illustrated, the set of nodes $\mathcal{V}$ encompasses five distinct types of nodes, presented here in hierarchical order. At the lowest level are the sets of incoming and outgoing lane segment nodes, denoted as $\mathcal{P}_{in}^{i}$ and $\mathcal{P}_{out}^{i}$, respectively. One level above comprises the set of movement nodes $\mathcal{M}^{i}$, followed by the set of phase nodes Φ^{i}. Finally, the top level features the intersection node $i \in \mathcal{I}$ itself. Regarding the set of edges $\mathcal{E}$, nodes on one level are typically connected by directed edges to (a subset of) nodes in the level below. While the intersection node i and phase nodes Φ^{i} connect to all nodes on the next lower level, each movement node $m = (l, o) \in \mathcal{M}^{i}$ is exclusively connected to the segment nodes $\mathcal{P}^{l}$ and $\mathcal{P}^{o}$ of the corresponding incoming lane l and outgoing lane o. Additionally, all phase nodes Φ^{i} are interconnected with each other. For later reuse, the set of neighbors of a node $v \in \mathcal{V}$ belonging to a particular type of node $\mathcal{U}$ will be denoted by $\mathcal{N}_{\mathcal{U}}(v) = \{u | (u, v) \in \mathcal{E} \cap u \in \mathcal{U}\}$. Accordingly, to give an example, for a movement $m \in \mathcal{M}^{i}$, the set of incoming lane segment nodes $\mathcal{P}_{in}^{i}$ connected to m is described by $\mathcal{N}_{\mathcal{P}_{in}^{i}}(m)$.

The feature vectors associated with individual nodes and edges and their individual components are summarized in Table 4.1. From the first two lines, it can be observed that the feature vectors generally encode the same information that is considered necessary in PressLight. Accordingly, the number of vehicles on each entering or exiting lane segment $l^{k} \in \mathcal{P}_{in}^{i} \cup \mathcal{P}_{out}^{i}$ is stored in the respective node feature vector $\boldsymbol{x}(l^{k})$, while the current phase $\phi \in \Phi^{i}$ is encoded as a flag in the corresponding node feature vector $\boldsymbol{x}(\phi)$ indicating whether the phase is currently active or not. To account for loss of node identity when encoding such information in a graph (see Section 2.2.3), some additional features are added to nodes as well as edges. In particular, to preserve the ordering of segments along a lane, an 8-dimensional sinusoidal positional encoding (see Section 2.2.3) is added to the node feature vector $\boldsymbol{x}(l^{k})$ of each lane segment $l^{k} \in \mathcal{P}_{in}^{i} \cup \mathcal{P}_{out}^{i}$. In addition, to each edge feature vector $\boldsymbol{x}(m, \phi)$ from a movement $m \in \mathcal{M}^{i}$ to a phase $\phi \in \Phi^{i}$, three flags are added indicating whether the

Feature Vectors	Components
$\boldsymbol{x}(l^k), \forall l^k \in \mathcal{P}^i_{in} \cup \mathcal{P}^i_{out}$	• Current number of vehicles • 8-dimensional sinusoidal positional encoding
$\boldsymbol{x}(\phi), \forall \phi \in \Phi^i$	• Flag whether phase is currently active
$\boldsymbol{x}(m, \phi), \forall \phi \in \Phi^i, \forall m \in \mathcal{M}^i$	• Flag whether movement is prohibited if phase is active • Flag whether movement is permitted if phase is active • Flag whether movement is protected if phase is active
$\boldsymbol{x}(\phi_j, \phi_k), \forall \phi_j, \phi_k \in \Phi^i$	• Jaccard coefficient of green signals

Table 4.1: Node / edge features of the graph-based state representation

movement is prohibited, permitted or protected when the phase is initiated. Finally, each edge feature vector $\boldsymbol{x}(\phi_j, \phi_k)$ from a phase $\phi_j \in \Phi^i$ to a phase $\phi_k \in \Phi^i$ contains the Jaccard coefficient (i.e., the intersection over the union) of the green signals between the two phases.

4.1.3 Action Definition

To facilitate a high degree of adaptability and enhance comparability with Max Pressure and PressLight, variable-sequence phasing is employed to define actions. At each time step t, an intersection $i \in \mathcal{I}$ can choose any of its $|\Phi^i|$ phases to either extend the current phase or switch to an arbitrary next. Consequently, the action space is equivalent to the intersection's set of phases Φ^i. In contrast to PressLight, intersections are not constrained to a fixed number of phases (four in the paper) and can instead have any number of phases, thereby facilitating adaptability. Each action persists for a duration of 10 seconds before the next action can be chosen. To ensure safety, every transition from one phase to another involves a 3-second yellow-change interval followed by 2-second all-red interval to clear the intersection. These configurations align with those of PressLight.

Layer l	**Updated nodes $\mathcal{V}^{(l)}$**	**Neighborhoods $\Omega^{(l)}(v)$ used for update of node $v \in \mathcal{V}^{(l)}$**
1	$\mathcal{M}^i$	$\{\mathcal{N}_{\mathcal{P}^i_{in}}(v), \mathcal{N}_{\mathcal{P}^i_{out}}(v)\}$
2	Φ^i	$\{\mathcal{N}_{\mathcal{M}^i}(v)\}$
3	Φ^i	$\{\mathcal{N}_{\Phi^i}(v)\}$
$L = 4$	$\{i\}$	$\{\mathcal{N}_{\Phi^i}(v)\}$

Table 4.2: Layers of the graph embedding module

4.2 Neural Network Architecture: TransferLight

To effectively map states to actions, a well-suited model for the agent's policy is of great importance. This model should possess adaptability to varying structures of the graph-structured state representation and accommodate any number of actions (i.e., phases), as outlined in the preceding section. To fulfill these requirements, this work introduces a novel neural network architecture named *TransferLight*. TransferLight is presented in two variants: the first, denoted as *TransferLight-DQN*, is compatible with a value-based Q-learning algorithm, while the second, named *TransferLight-A2C*, is designed for use with a policy-based A2C algorithm. Both variants share the same architecture of a graph embedding module[1] that generates vector embeddings for the nodes in the graph-structured state representation. The variants differ in how the node embeddings are used to predict action-values in the case of TransferLight-DQN or action probabilities and state-values in the case of TransferLight-A2C.

4.2.1 Graph Embedding Module

To create node vector embeddings, the graph embedding module propagates information from the bottom up through the graph-structured state representation s_t^i, starting at the lane segment nodes at the lowest level and ending at the intersection node i itself at the highest level. For this purpose, the concept of neural message passing (see Section 2.2.3) is used. Each layer l updates a subset of nodes $\mathcal{V}^{(l)} \subset \mathcal{V}$ all belonging to a particular type. The update of each of those nodes $v \in \mathcal{V}^{(l)}$ is based on a set of neighborhoods $\Omega^{(l)}(v)$ connected to v, each of which includes only nodes of a certain type themselves. Table 4.2 summarizes for each layer l in which order the nodes are updated. It should be noted that all nodes that are not updated by a layer l retain their

---

[1] Note that parameters are not shared between the two variants.

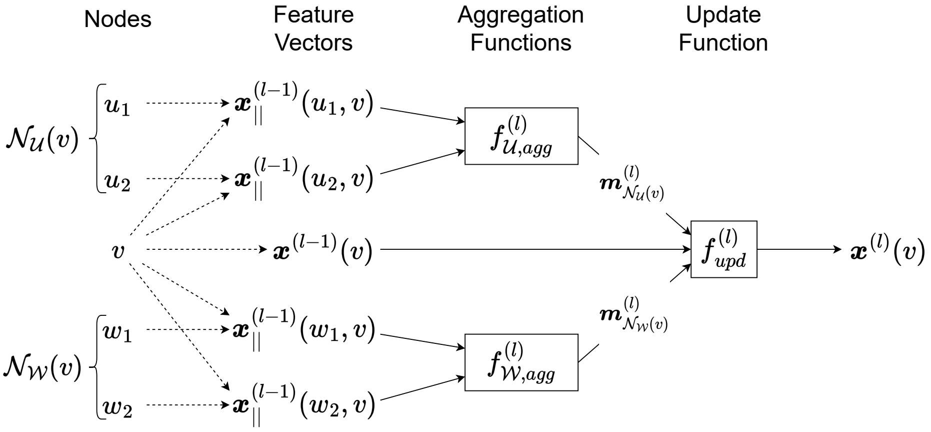

Nodes
Feature Vectors
Aggregation Functions
Update Function
$\mathcal{N}_{\mathcal{U}}(v)$
u_1
u_2
v
$\mathcal{N}_{\mathcal{W}}(v)$
w_1
w_2
$\boldsymbol{x}_{||}^{(l-1)}(u_1, v)$
$\boldsymbol{x}_{||}^{(l-1)}(u_2, v)$
$\boldsymbol{x}^{(l-1)}(v)$
$\boldsymbol{x}_{||}^{(l-1)}(w_1, v)$
$\boldsymbol{x}_{||}^{(l-1)}(w_2, v)$
$f_{\mathcal{U},agg}^{(l)}$
$f_{\mathcal{W},agg}^{(l)}$
$\boldsymbol{m}_{\mathcal{N}_{\mathcal{U}}(v)}^{(l)}$
$\boldsymbol{m}_{\mathcal{N}_{\mathcal{W}}(v)}^{(l)}$
$f_{upd}^{(l)}$
$\boldsymbol{x}^{(l)}(v)$

Figure 4.2: Computation graph of a neural message passing layer in the graph embedding module

old vector representation, i.e., $x^{(l)}(v) := x^{(l-1)}(v), \forall v \in \mathcal{V} \setminus \mathcal{V}^{(l)}$.

The sequence of operations performed by a single layer l of neural message passing in the module is very similar to the one described in Section 2.2.3. However, a key difference is that a node can be updated based on more than one type of neighborhood. This is the case in the first layer, where all movement nodes $\mathcal{M}^i$ are updated based on their associated neighboring incoming and outgoing lane segments, i.e., $\Omega^{(l)}(v) = \{\mathcal{N}_{\mathcal{P}^i_{in}}(v), \mathcal{N}_{\mathcal{P}^i_{out}}(v)\}$ (see Table 4.2). Therefore, to ensure that neural message passing treats neighborhoods differently depending on their type, each neighborhood $\mathcal{N}_{\mathcal{U}}(v) \in \Omega^{(l)}(v)$ of a node $v \in \mathcal{V}^{(l)}$ is aggregated into a message vector $\boldsymbol{m}^{(l)}_{\mathcal{N}_{\mathcal{U}}(v)}$ by an aggregation function $f^{(l)}_{\mathcal{U},agg}$ parameterized separately for each type of nodes $\mathcal{U}$. As before, these message vectors are then used by the update function $f^{(l)}_{upd}$ along with the old vector representation $\boldsymbol{x}^{(l-1)}(v)$ to create a new vector representation $\boldsymbol{x}^{(l)}(v)$ for v. This computational scheme is demonstrated on a simple example in the computation graph of Figure 4.2.

To ensure that the neighborhood aggregation runs in a numerically stable manner during training while still allowing for a high degree of representational strength, the individual aggregation functions are implemented as weighted sums with multi-head attention, exactly as described in Section 2.2.3. Each message function, scoring function, and update function is implemented as a MLP with a residual connection, incorporating additional intermediate layers for layer normalization and dropout (see Section 2.2.4). This design aims to enhance training stability and convergence. The inclusion of dropout layers is particularly advantageous for Q-learning, as it promotes

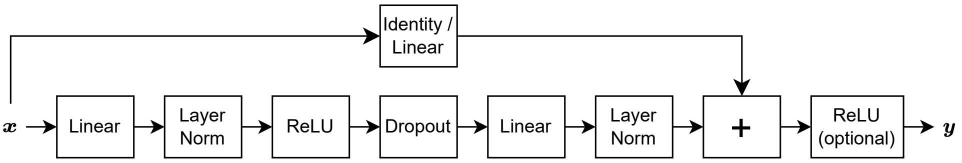


Figure 4.3: Computation graph of the MLP used in the message, scoring and update functions

exploration when confidence in the Q-value estimates is low, making other exploration strategies, like ϵ-greedy, obsolete (see Section 2.3.2). The computation graph of this MLP is represented in Figure 4.3. Each linear layer in the MLP uses a fixed number of 128 neurons, except for the scoring functions, where the last linear layer includes only one neuron without the subsequent ReLU activation (since the scoring functions are used to generate attention weights). Furthermore, following previous works [53, 97], the number of attention heads used in each aggregation function is set to $H = 8$. The dropout rate is set to 0.1. This is in agreement with the authors of MC-dropout [64].

4.2.2 Deep Q-Network

To improve the efficiency and stability of Q-learning, TransferLight-DQN uses a dueling architecture (see Section 2.3.2) on top of the final vector embeddings generated by the graph embedding module. While the V-Stream $V(s^i;\boldsymbol{\theta})$ uses the final embedding $\boldsymbol{x}^{(L)}(i)$ of an intersection i to predict its state-value, the A-Stream $A(s^i,\phi;\boldsymbol{\theta})$ uses the final embedding $\boldsymbol{x}^{(L)}(\phi)$ of a phase $\phi \in \Phi^i$ to predict its action-advantage. Both the V-Stream and the A-Stream are implemented as a 2-layer MLP with 128 neurons using ReLU activation in the first layer and a single neuron using linear activation in the second layer:

$$V(s^i;\boldsymbol{\theta}) = \boldsymbol{w}_V^T \sigma(\boldsymbol{W}_V \boldsymbol{x}^{(L)}(i)), \tag{4.2}$$

$$A(s^i,\phi;\boldsymbol{\theta}) = \boldsymbol{w}_A^T \sigma(\boldsymbol{W}_A \boldsymbol{x}^{(L)}(\phi)). \tag{4.3}$$

Here, $\boldsymbol{w}_V$, $\boldsymbol{W}_V$, $\boldsymbol{w}_A$ and $\boldsymbol{W}_A$ correspond to the trainable weights of the streams. The action-values are then recovered from the outputs of the two streams using the standard aggregation rule of the dueling architecture, yielding the final DQN:

$$Q(s^i,\phi;\boldsymbol{\theta}) = V(s^i;\boldsymbol{\theta}) + A(s^i,\phi;\boldsymbol{\theta}) - \frac{1}{|\Phi^i|} \sum_{\phi' \in \Phi^i} A(s^i,\phi';\boldsymbol{\theta}). \tag{4.4}$$

4.2.3 Actor-Critic Network

In contrast to using two separate models for the actor and the critic, TransferLight-A2C implements both in a joint model. In particular, the actor and the critic share the same graph embedding module and then branch into two streams, akin to the DQN variant. The value-stream $V(s^i;\boldsymbol{\theta})$ (i.e., the critic) is exactly implemented as defined in Equation 4.3. The policy-stream $\pi(\phi|s^i;\boldsymbol{\theta})$ (i.e., the actor) uses the same 2-layer MLP structure as the other streams but is followed by the softmax activation function to obtain a valid categorical distribution over available phases Φ^i:

$$\pi(\phi|s^i;\boldsymbol{\theta}) = \frac{\exp\left(\boldsymbol{w}_\pi^T \sigma(\boldsymbol{W}_\pi \boldsymbol{x}^{(L)}(\phi))\right)}{\sum_{\phi' \in \Phi^i} \exp\left(\boldsymbol{w}_\pi^T \sigma(\boldsymbol{W}_\pi \boldsymbol{x}^{(L)}(\phi'))\right)}. \tag{4.5}$$

The implementation of the actor and the critic as a joint model has two advantages. First, it greatly reduces the computational cost of training, as only one model needs to be queried and updated. Second, since the high-level features required for action prediction and value estimation are quite similar, learning a joint representation can be beneficial for both tasks.

4.3 Training

Besides the model, the training algorithm should also be implemented in a way that supports generalization. To this end, the training algorithms in this work use an independent learning procedure (see Section 2.3.5) where the agents share the same policy represented by the corresponding TransferLight variant (i.e., instead of learning a separate TransferLight model for each agent, all agents use the same one). In this way, the algorithms can freely adapt to any number of agents during their execution. Moreover, by sharing parameters among agents, the algorithms are essentially encouraged to converge to a region in parameter space that works well for arbitrary intersections and traffic conditions, thereby promoting generalization.

4.3.1 Independent Q-Learning Algorithm

The independent Q-learning algorithm used to train the TransferLight-DQN mainly implements the techniques discussed in Section 2.3.2, which aim to make Q-learning appear more like a supervised learning task. For example, a separate target DQN is used along with double Q-learning to estimate Q-targets in a more stationary and

```
Input: Initial online parameters $\boldsymbol{\theta}$, initial target parameters $\boldsymbol{\theta}^-$, learning rate $\eta$,
       discount factor $\gamma$, training steps $T$, set of workers $\mathcal{W}$, mixing weight $\tau$, batch
       size $B$, domain randomization space $\Xi$
Output: Trained parameters $\boldsymbol{\theta}$
$D := \{\}$;
for $w \in \mathcal{W}$ do $\xi^w \sim \Xi$; $s_0^w \sim T(\cdot|\xi^w)$;
for $t := 1 : T$ do
    for $w \in \mathcal{W}$ do
        for $i \in \mathcal{I}^w$ do $\phi_t^{w,i} := \arg\max_{\phi \in \Phi^{w,i}} Q(s_t^{w,i}, \phi; \boldsymbol{\theta})$;
        $s_{t+1}^w \sim T(\cdot|\boldsymbol{\phi}_t^w, s_t^w, \xi^w)$;
        for $i \in \mathcal{I}^w$ do $r_{t+1}^{w,i} := R^{w,i}(s_t^w, \boldsymbol{\phi}_t^w, s_{t+1}^w)$;
        for $i \in \mathcal{I}^w$ do $D := D \cup \{(s_t^{w,i}, \phi_t^{w,i}, r_{t+1}^{w,i}, s_{t+1}^{w,i})\}$;
    $J(\boldsymbol{\theta}) := 0$;
    for $b = 1 : B$ do
        $(s_{t'}^{w,i}, \phi_{t'}^{w,i}, r_{t'+1}^{w,i}, s_{t'+1}^{w,i}) \sim D$;
        $\hat{q}_\pi(s_{t'}^{w,i}, \phi_{t'}^{w,i}) := r_{t'+1}^{w,i} + \gamma Q\big(s_{t'+1}^{w,i}, \arg\max_{\phi_{t'+1}^{w,i} \in \Phi^{w,i}} Q(s_{t'+1}^{w,i}, \phi_{t'+1}^{w,i}; \boldsymbol{\theta}); \boldsymbol{\theta}^-\big)$;
        $L(\boldsymbol{\theta}) := \big(\hat{q}_\pi(s_{t'}^{w,i}, \phi_{t'}^{w,i}) - Q(s_{t'}^{w,i}, \phi_{t'}^{w,i}; \boldsymbol{\theta})\big)^2$;
        $J(\boldsymbol{\theta}) := J(\boldsymbol{\theta}) + \frac{1}{b}\Big(L(\boldsymbol{\theta}) - J(\boldsymbol{\theta})\Big)$;
    $\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$;
    $\boldsymbol{\theta}^- := \tau \boldsymbol{\theta} + (1 - \tau)\boldsymbol{\theta}^-$;
    for $w \in \mathcal{W}$ do
        if $is\_terminal(s_{t+1}^w)$ then $\xi^w \sim \Xi$; $s_{t+1}^w \sim T(\cdot|\xi^w)$;
```

Algorithm 4.1: Independent Q-learning algorithm

less biased manner. Also, a replay buffer is introduced to decorrelate the experience tuples used for updating the parameters of the online DQN. In addition to these techniques, the algorithm exploits multiprocessing capabilities by executing a fixed number of worker processes $\mathcal{W}$ simultaneously, each of which runs an independent episode. This extension is not only beneficial in terms of resource utilization, but also increases the diversity of experience tuples stored in the replay buffer, further strengthening the assumption of i.i.d. training examples. In addition to the Q-learning-specific techniques mentioned, parameters are initialized using the Saxe initialization method, and the Adam optimizer is employed to enhance convergence (see Section 2.2.4).

The procedure of the algorithm is summarized in Algorithm 4.1. To better distinguish the different episodes running in parallel, the affected elements are marked with a su-

Hyperparameter	Value
Learning rate η	0.001
Discount factor γ	0.9
Number of training steps T	2,000
Number of workers $\|\mathcal{W}\|$	64
Mixing weight τ	0.01
Batch size B	64
Replay buffer size	10,000

Table 4.3: Hyperparameters of the independent Q-learning algorithm

perscript of the corresponding worker in the algorithm. For example, $\mathcal{I}^w$ corresponds to the set of agents belonging to the environment currently being executed by worker $w \in \mathcal{W}$. The algorithm first initializes an empty replay buffer and the episode of each worker based on an environmental configuration sampled from the given DR space Ξ (line 1-2). Then, the algorithm is executed for T training steps (line 3). In each step, all agents perform an action by following the current policy to move to the next state and receive their corresponding reward (line 4-7). For this purpose, each agent chooses the action that corresponds to the maximum Q-value with respect to the online DQN (i.e., the greedy action). Since the architecture of the DQN implements dropout layers (see Section 4.2.1), this type of action selection essentially promotes exploration when the uncertainty in the Q-value prediction is high (see Section 2.3.2). The experience tuples generated by the transition are stored in the replay buffer (line 8). It should be noted that the replay buffer is implemented as a fixed-size queue. Accordingly, each time a new experience tuple is to be stored and the replay buffer is full, the oldest entry is first removed before the new entry is inserted. After the transition, the algorithm updates the parameters $\boldsymbol{\theta}$ of the online DQN. To this end, the MSE cost $J(\boldsymbol{\theta})$ between the Q-targets and the Q-predictions are estimated from a batch of B experience tuples sampled uniformly at random from the replay buffer (line 10-14). The cost is then minimized using one step of gradient descent, resulting in the updated parameters $\boldsymbol{\theta}$ (line 15). Then, the parameters $\boldsymbol{\theta}^-$ of the target DQN are updated (line 16) using Polyak averaging (see Section 2.3.2). Finally, before executing the next training step, the algorithm checks whether any of the episodes has reached a terminal state, and if so, the corresponding worker process starts a new episode (line 17-18). A terminal condition can be reached in one of two ways. Either when the last vehicle reaches its destination or when one of the vehicles exceeds a waiting time of 600 seconds. The

```
Input: Initial parameters $\boldsymbol{\theta}$, learning rate $\eta$, discount factor $\gamma$, training steps $T$, set of
       workers $\mathcal{W}$, domain randomization space $\Xi$
Output: Trained parameters $\boldsymbol{\theta}$
for $w \in \mathcal{W}$ do $\xi^w \sim \Xi$; $s_0^w \sim T(\cdot|\xi^w)$;
for $t := 1 : T$ do
    for $w \in \mathcal{W}$ do
        for $i \in \mathcal{I}^w$ do $\phi_t^{w,i} \sim \pi(\cdot|s_t^{w,i};\boldsymbol{\theta})$;
        $s_{t+1}^w \sim T(\cdot|\boldsymbol{\phi}_t^w, s_t^w, \xi^w)$;
        for $i \in \mathcal{I}^w$ do $r_{t+1}^{w,i} := R^{w,i}(s_t^w, \boldsymbol{\phi}_t^w, s_{t+1}^w)$;
    $b, J(\boldsymbol{\theta}) := 0$;
    for $w \in \mathcal{W}$ do
        for $i \in \mathcal{I}^w$ do
            $b := b + 1$;
            $\hat{q}_\pi(s_t^{w,i}, \phi_t^{w,i}),\ \hat{v}_\pi(s_t^{w,i}) := r_{t+1}^{w,i} + \gamma V(s_{t+1}^{w,i};\boldsymbol{\theta})$;
            $\hat{a}_\pi(s_t^{w,i}, \phi_t^{w,i}) := \hat{q}_\pi(s_t^{w,i}, \phi_t^{w,i}) - V^{w,i}(s_t^{w,i};\boldsymbol{\theta})$;
            $L_{actor}(\boldsymbol{\theta}) := -\hat{a}_\pi(s_t^{w,i}, \phi_t^{w,i}) \log \pi(\phi_t^{w,i}|s_t^{w,i};\boldsymbol{\theta})$;
            $L_{critic}(\boldsymbol{\theta}) := \frac{1}{2}\big(\hat{v}_\pi(s_t^{w,i}) - V(s_t^{w,i};\boldsymbol{\theta})\big)^2$;
            $J(\boldsymbol{\theta}) := J(\boldsymbol{\theta}) + \frac{1}{b}\Big(\big(L_{actor}(\boldsymbol{\theta}) + L_{critic}(\boldsymbol{\theta})\big) - J(\boldsymbol{\theta})\Big)$;
    $\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$;
    for $w \in \mathcal{W}$ do
        if $is\_terminal(s_{t+1}^w)$ then $\xi^w \sim \Xi$; $s_{t+1}^w \sim T(\cdot|\xi^w)$;
```

Algorithm 4.2: Independent Advantage Actor-Critic (A2C) Algorithm

latter is used to recover from failures where an unfavorable action selection by the agents leads to a severe traffic jam from which recovery becomes unlikely.

To attain optimal performance, an extensive hyperparameter search would be desirable. Regrettably, due to resource constraints within the scope of this work, the application of hyperparameter tuning strategies was not feasible within a reasonable time frame. Consequently, hyperparameters were selected to the best of the author's knowledge. The configuration that yielded satisfactory results across all trained models in this study is detailed in Table 4.3.

4.3.2 Independent Advantage Actor-Critic Algorithm

The independent A2C algorithm, tasked with training the parameters of TransferLight-A2C, closely adheres to the principles discussed in Section 2.3.3. Similar to the inde-

Hyperparameter	**Value**
Learning rate η	0.001
Discount factor γ	0.9
Number of training steps T	2,000
Number of workers $\lvert\mathcal{W}\rvert$	64

Table 4.4: Hyperparameters of the independent Advantage Actor-Critic (A2C) algorithm

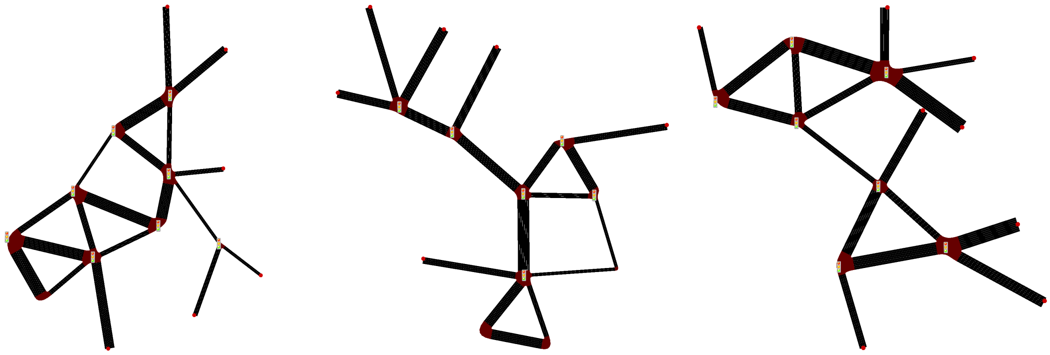
pendent Q-learning algorithm, multiprocessing is employed to decorrelate experience tuples during training. However, rather than storing these tuples in a replay buffer for subsequent reuse, they are promptly utilized to perform immediate updates. This immediacy is crucial, as estimating the policy gradient necessitates the use of experience tuples generated from the current policy. Consequently, workers adopt synchronous updates in the algorithm to mitigate the risk of one worker producing a biased policy gradient estimate due to an outdated policy. Given that both the actor and the critic share network parameters, they undergo joint updates based on a collective cost function that incorporates their individual losses. Again, the Saxe initialization method and the Adam optimizer are employed to facilitate convergence (see Section 2.2.4).

The flow of the algorithm is shown in Algorithm 4.2. As before, the individual workers are first initialized by each starting an independent episode according to an environmental configuration sampled from the DR space Ξ (line 1). Thereafter, the algorithm runs for T training steps. At each step, agents first transition to a new state by selecting an action from their current policy and collecting their reward (line 3-6). After the transitions are complete, for each experience tuple generated, the loss for the actor and the critic are combined by summation, and the intermediate results are averaged to arrive at the final cost $J(\boldsymbol{\theta})$ (line 7-15). The parameters $\boldsymbol{\theta}$ of the actor-critic network are then updated by minimizing the cost with one step of gradient descent (line 16). Finally, each worker checks to see if its episode has ended. If so, the worker samples a new environment configuration and starts a new episode (line 17-18).

The hyperparameters used for training all actor-critic models in this work are listed in Table 4.4 and were again determined to the best of the author's knowledge.

4.4 Domain Randomization Mechanism

As explained in Section 2.3.6, the main idea behind DR is to randomize certain parameters of the environment in each episode during training. In the context of TSC, a



(a) <SEED>=42 (b) <SEED>=67 (c) <SEED>=103

Figure 4.4: Randomly generated road networks for different seeds

suitable DR mechanism should expose intersections to a wide range of traffic scenarios, varying parameters that affect both the road network and traffic. Previous research [68, 69], as discussed in Section 3.2, has primarily focused on randomizing the road network, while the randomization of traffic has not been adequately addressed. To address this gap in the literature, this thesis proposes a DR mechanism that randomizes not only the road network but also the traffic in each episode, thereby generating environments with more diverse traffic conditions.

4.4.1 Road Network

To create different road networks, this work follows the same idea as Devailly et al. [68, 69]. Each road network to be generated comprises between 3-10 intersections. These intersections in turn consist of 3-5 incoming or outgoing approaches, each of which comprises between 1-4 lanes and is between 100-200 meters long. Depending on the structure of the intersection, an intersection can have between 2-5 phases. Lanes have a speed restriction of 50 kilometers per hour. The `netgenerate`[2] script of the SUMO [98] package is used to randomly generate road networks that fulfill these conditions. The corresponding command is as follows:

```
$ netgenerate -r --rand.iterations 15 --rand.max-distance 200
    ↪ --rand.min-distance 100 -j traffic_light --tls.discard-
    ↪ simple --rand.min-angle 45 --rand.num-tries 100 --rand.
    ↪ neighbor-dist5 10 --rand.neighbor-dist6 0 -L 4 --random-
    ↪ lanenumber --no-turnarounds -o /path/to/output/file.net.
    ↪ xml --seed <SEED>
```

---

[2] `https://sumo.dlr.de/docs/netgenerate.html` (accessed 11.11.2023)

When the same `<SEED>`, corresponding to an integer, is used each time this command is called, it produces the same network. However, changing the seed each time the command is invoked (e.g., by incrementing) results in diverse road networks. Figure 4.4 displays various road networks generated using this command for different seeds.

4.4.2 Traffic

To generate random traffic for a given road network, in each episode a set $\mathcal{F}$ of 25 traffic flows is randomly generated, each of which generates vehicle trips of passenger cars within a fixed time span of $T = 900$ seconds (15 minutes). Each flow $f \in \mathcal{F}$ is randomized with respect to three parameters: the route $\mathcal{R}^f = (l^f_{src}, \dots, l^f_{dst})$, i.e. the sequence of lanes that all vehicles in the flow take to get from a source lane $l^f_{src} \in \mathcal{L}$ to a destination lane $l^f_{dst} \in \mathcal{L}$, the number of vehicles n^f that follow this flow, and the exact departure times $t^f_1, \dots, t^f_{n^f}$ of these vehicles.

To generate a random route $\mathcal{R}^f$ for a flow $f \in \mathcal{F}$, a source lane $l^f_{src} \in \mathcal{L}$ and a destination lane $l^f_{dst} \in \mathcal{L}$ are first uniformly sampled from the set of lanes $\mathcal{L}$ of the road network:

$$l^f_{src}, l^f_{dst} \sim U(\mathcal{L}). \tag{4.6}$$

The final route $\mathcal{R}^f$ is determined using the *Dijkstra algorithm* [99] by searching for the shortest path of intermediate lanes connecting the source and destination lanes.

To determine the number of vehicles in a traffic flow, a pool of $N = 300$ vehicles is considered in each episode and randomly distributed among the traffic flows. For this purpose, a value u^f is sampled from the standard uniform distribution for each traffic flow $f \in \mathcal{F}$:

$$u^f \sim U(0,1). \tag{4.7}$$

These values are then normalized by division, resulting in a normalized weight w^f for each traffic flow $f \in \mathcal{F}$ that indicates the proportion of vehicles assigned to the flow:

$$w^f = \frac{u^f}{\sum_{f' \in \mathcal{F}} u^{f'}}. \tag{4.8}$$

The number of vehicles assigned to a flow $f \in \mathcal{F}$ is finally obtained by multiplying the total number of vehicles N by the corresponding weight w^f and rounding the result to the nearest integer:

$$n^f = \lfloor w^f \cdot N + 0.5 \rfloor. \tag{4.9}$$

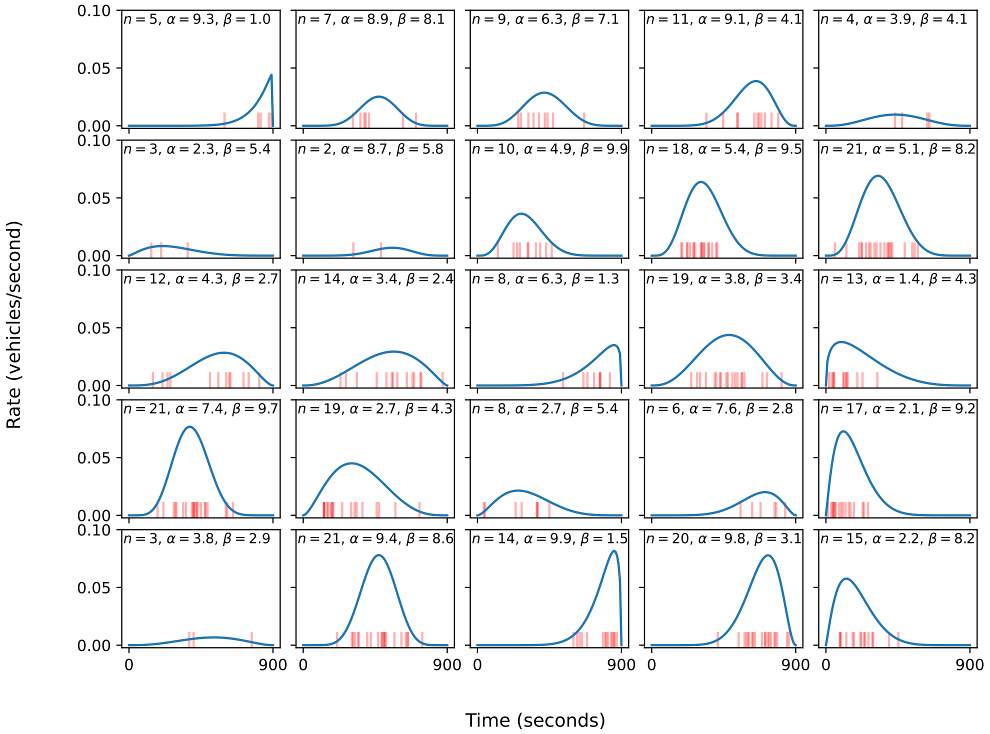


Figure 4.5: Random traffic generated for different traffic flows

A common approach to randomly determine the departure times of the n^f vehicles assigned to a traffic flow $f \in \mathcal{F}$ is to model the departures as a Poisson process with a constant rate of n^f/T vehicles per second. In this way, the n^f departure times can simply be sampled from a uniform distribution in the range from 0 to T. One problem with this approach, however, is that a constant departure rate is often not realistic in practice. Rather, it is common to observe a departure rate that fluctuates over time and often has peaks (e.g. during rush hours). To better account for such changes and ensure greater variability in the departure rate of a traffic flow, the departure times are therefore sampled from a beta distribution whose structure varies between traffic flows. Specifically, for a flow $f \in \mathcal{F}$, the α^f and β^f parameters of its beta distribution are sampled uniformly in a range between 1 and 10:

$$\alpha^f, \beta^f \sim U(1, 10). \tag{4.10}$$

Then for each of the n^f assigned vehicles of the flow, a sample is drawn independently from the resulting beta distribution:

$$b_1^f, \dots, b_{n^f}^f \sim Beta(\alpha^f, \beta^f). \tag{4.11}$$

Multiplying these samples by T yields the vehicle departure times of the flow:

$$\begin{aligned} t_1^f &= b_1^f \cdot T, \\ &\dots \\ t_{n^f}^f &= b_{n^f}^f \cdot T. \end{aligned} \tag{4.12}$$

In this way, a variety of departure time patterns can be generated for each flow $f \in \mathcal{F}$, usually with a peak in the departure rate somewhere in the time span T, whereby the special case of a constant departure rate (i.e., $\alpha^f = \beta^f = 1$) is not excluded. Figure 4.5 shows a typical random sample drawn from this process. Each of the adjacent diagrams corresponds to one of the $|\mathcal{F}| = 25$ randomly sampled traffic flows running in parallel. While the blue curves represent the temporal variation of the departure rate of these flows (i.e., the PDF of their corresponding beta distributions scaled according to the time span and the number of assigned vehicles), the red markers correspond to the departure times of the vehicles assigned to the flows.

4.5 Implementation Details

All models, training algorithms and simulations in this thesis are implemented in the *Python* [100] programming language. For the sake of simplicity, some additional packages are used that provide some integrated functionality. These include the automatic differentiation framework *PyTorch* [101], which is used to implement the models and to calculate the gradients for gradient descent. Building on this, the *PyTorch Geometric* [102] library is additionally used to implement the neural message passing layers in the models and to model the graph-structured state representation. The traffic-related simulations in this thesis are performed with the microscopic traffic simulator SUMO [98] in its default configuration, which uses the *Krauss car-following model* [103] with some minor adaptations[3]. To extract relevant traffic-related and road network-related features from a running simulation, the packages *Libsumo*[4] and

---

[3] See details in `https://sumo.dlr.de/docs/Definition_of_Vehicles,_Vehicle_Types,_and_Routes.html` (accessed (13.11.2023))

[4] `https://sumo.dlr.de/docs/Libsumo.html` (accessed 13.11.2023)

Sumolib[5] are used. For numerical and statistical operations, the packages *NumPy* [104] and *SciPy* [105] are utilized. Plots are created with the *Matplotlib* [106] library. The entire code, checkpoints of the trained models and evaluation results are available on *GitHub*[6].

---

[5] `https://sumo.dlr.de/docs/Tools/Sumolib.html` (accessed 13.11.2023)

[6] `https://github.com/franky3er/TransferLight`

5

Evaluation

To promote generalization of TSC policies to a broad spectrum of traffic scenarios, several techniques were devised in the previous chapter. This chapter examines the effectiveness of these methods in light of the overall goal of this thesis. In order to guide this evaluation, the research questions initially introduced in Chapter 1 are revisited and contextualized. Following this, an evaluation framework is delineated, detailing the test scenarios, comparison methods, and evaluation metrics. This framework is subsequently employed to conduct the analysis and provide answers to the posed research questions.

5.1 Research Questions

As discussed in the literature review in Section 3.2, several studies utilize a graph-structured state representation in conjunction with a model for the policy that incorporates neural message passing, aiming to enhance both reusability and generalization across diverse intersection structures. Some of these scholars have deemed the utilization of such a model architecture as adequate to enable the policy's transfer to novel traffic conditions [11, 12, 29]. They attribute this to the extensive use of parameter sharing between intersection components in neural message passing, allowing the policy to generalize toward symmetric cases of traffic approaching the intersection [11]. However, while some experiments on simplified isolated intersection scenarios in these works suggest that transferability is enhanced with the aid of neural message passing, the question of whether training on a fixed traffic scenario may still lead to an overfitting problem remains unexplored. Given that TransferLight follows a similar modeling approach, the following first research question addresses this research gap:

Q1: *Does TransferLight exhibit overfitting when trained exclusively on a fixed scenario?*

In addition to augmenting the policy's generalization through its model representation, this thesis proposed a DR mechanism. This mechanism introduces stochastic variations to certain parameters related to both the road network and traffic configuration in each training episode. The principal objective of incorporating such variability is to enhance the adaptability of the policy to diverse traffic scenarios that may differ from those encountered during the training phase. To evaluate the effectiveness of this introduced mechanism in achieving its intended objective, the following second research question will be explored:

> Q2: *To what extent does the zero-shot transfer performance of TransferLight experience improvement when trained in conjunction with the suggested DR mechanism, and what are the potential limitations associated with the effectiveness of this approach?*

By incorporating the proposed domain randomization mechanism into the training process, the range of conceivable traffic conditions that the agents can explore during training is substantially expanded. This allows TransferLight to learn from a more diverse set of experience tuples, fostering a more nuanced comprehension of the repercussions stemming from specific phasing decisions across a wide array of traffic scenarios. It is highly probable that this increased variability in the training data will also make the learned policy more robust, potentially diminishing uncertainties in its predictions when confronted with novel traffic conditions. To investigate whether this is the case, the following third and final research question will be addressed:

> Q3: *To what extent does the integration of the proposed DR mechanism into the training procedure diminish the uncertainty in the predictions made by TransferLight when confronted with novel traffic scenarios?*

5.2 Evaluation Framework

This section presents an evaluation framework designed to answer the research question outlined in the previous section. First, a series of test scenarios are presented that deliberately differ from the conditions used in the training phase, with the explicit aim of increasing the transfer difficulty for the learned policies. Secondly, different TransferLight variants trained under different conditions are presented alongside other baselines to provide a more nuanced basis for comparative analysis. Finally, the metrics used for the evaluation are explained.

5.2.1 Test Scenarios

Evaluating the transferability of a TSC policy demands a deliberate selection of traffic scenarios that are intentionally different from the conditions used in the training phase of the policy. Simply sampling a new traffic scenario from the proposed randomization space for evaluation may not be sufficient to meet this criterion. The bounds imposed on the randomization parameters may limit the representation of traffic dynamics encountered by the policy in practical scenarios. Due to this reason, transferability is assessed using a suite of synthetically generated and real-world test scenarios that were deliberately chosen to fall outside the prescribed randomization space. All these scenarios have a fixed duration of exactly 3,600 seconds (1 hour). Considering that the test scenarios run longer than the training scenarios, the difficulty of TSC is already greatly increased, as vehicles continue to be added to the simulation after 15 minutes (duration used to simulate vehicles during training), making it more challenging to recover from congested roads while demand remains the same. In addition to the higher complexity due to the longer duration, the test scenarios differ from the training scenarios both in the structure of the road network and in the traffic pattern. In the following, the considered test scenarios are described in more detail. The road network structures of these scenarios are presented in Figure 5.1[1].

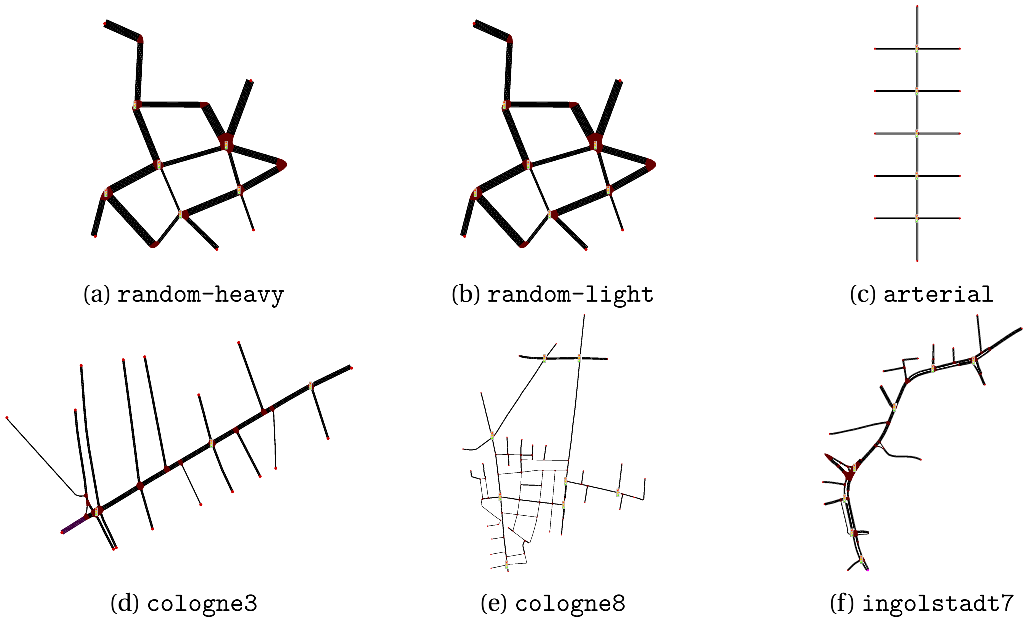
Synthetic Scenarios

The following synthetically generated scenarios are considered for testing:

- `random-heavy`: This scenario is sampled from the DR mechanism described in Section 4.4, with some minor adjustments made to increase the traffic volume and see how the policy can handle higher demand. In particular, the number of simulated vehicles is increased by 1/3 within the same time span. Given that $N = 300$ vehicles are simulated within each training scenario, and considering that the test scenario runs 4 times longer than a training scenario, a total of $\frac{4}{3} \cdot 4 \cdot 300 = 1600$ vehicles are simulated in the corresponding test scenario. In order to better distribute the vehicles over the simulated time span, the number of traffic flows is additionally increased by 4, resulting in a total of $4 \cdot 25 = 100$ traffic flows. All other aspects concerning randomization remain the same. The seed used to generate the road network corresponds to 6063. It should be noted that this seed is not used during training. Accordingly, the policy is tested on a

---

[1] Note that the road networks shown in the illustrations are not necessarily comparable in scale.



(a) `random-heavy` (b) `random-light` (c) `arterial`

(d) `cologne3` (e) `cologne8` (f) `ingolstadt7`

Figure 5.1: Road networks of the test scenarios

previously unknown road network. The structure of this road network is shown in Figure 5.1(a).

- **`random-light`**: This scenario is generated similarly to the `random-heavy` scenario, but with a 1/3 reduction in traffic volume. Thus, the total number of simulated vehicles corresponds to $\frac{2}{3} \cdot 4 \cdot 300 = 800$. The primary objective of this test scenario is to assess whether the policy can efficiently handle low demand, minimizing unnecessary waiting times for vehicles at intersections.

- **`arterial`**: The last synthetic scenario is an arterial road network with five homogeneous four-way intersections, each 200 meters apart, and with side streets that are also 200 meters long (see Figure 5.1(c)). All roads have only one lane. There are six traffic flows, each of which generates passenger car traffic in one direction (i.e., no turning traffic) at a constant rate, with vehicles uniformly distributed over the 3,600 second time period. One of these flows generates a traffic volume of 700 vehicles per hour on the arterial street, starting with the southernmost lane and ending with the northernmost lane. Each of the other five flows is assigned to one of the side streets and generates a traffic volume of 420 vehicles per hour from west to east. Each intersection has exactly two phases: one for assigning right-of-way to the main street movement and one for assigning right-of-way to the side street movement. Given the unidirectional and uniform traffic in the road network, the utilization of the Webster method [40] (see Section 2.1.3) in conjunction with the GreenWave method [41] (see Section 2.1.4) facilitates the determination of pretimed signalization parameters for each intersection that presumably minimizes vehicle delay. Other studies [22, 9] (including PressLight) have shown that it is possible to learn a cascading of green times that is competitive to the solution of Webster/GreenWave if the policy is explicitly trained on such a scenario. In the context of this work, this setting offers the opportunity to investigate whether this is also true when the policy is not specifically trained on this scenario and instead has to adapt to it from the learned knowledge of scenarios sampled from the proposed DR space.

Real-World Scenarios

To assess the adaptability of signal control policies to real-world traffic conditions, the RESCO benchmark [16] is employed. This benchmark provides TSC tasks for two major cities in Germany, Cologne and Ingolstadt, implemented in SUMO [98]. The traffic distributions in these scenarios were derived from actual traffic measurements and demographic data. Unlike synthetic scenarios, these real-world scenarios also incorporate longer and shorter lanes than those considered in the DR space. This work considers the following scenarios from the benchmark:

- `cologne3`: This scenario simulates traffic on a section of the arterial road "Dürener Straße" in Cologne and includes 3 signalized intersections. The structure of this road is presented in Figure 5.1(d).

- `cologne8`: This scenario simulates traffic in a road network in a part of the district “Nippes” in Cologne. It involves 8 signalized intersections. The structure of the road network is shown in Figure 5.1(e).

- `ingolstadt7`: The final scenario simulates traffic in a section of the “Westliche Ringstraße” in Ingolstadt and involves 7 signalized intersections (see Figure 5.1(f)). In contrast to the two other scenarios from Cologne, the traffic in this scenario also includes vehicle types other than personal cars, such as buses.

5.2.2 Compared Methods

Different TSC methods are compared with each other. These include TransferLight variants trained under different settings as well as other TE- and RL-based baselines.

TransferLight Variants

Addressing the aforementioned research questions necessitates training various instances of TransferLight using distinct configurations. In one configuration, TransferLight is trained utilizing DR, wherein a new scenario is drawn from the randomization space described in Section 4.4 for each training episode. In an alternative configuration, TransferLight is trained with a single, unchanging scenario that is reused in all training episodes. This specific scenario is drawn once from the randomization space, with vehicles evenly distributed across traffic flows, and each flow maintaining a constant departure rate. The corresponding road network for this scenario is generated with a seed of 42 and is illustrated in Figure 4.4(a).

Both TransferLight-DQN and TransferLight-A2C are examined in the context of these two training configurations to determine the generalizability of the results across different algorithmic formulations (i.e., Q-learning or A2C). To delineate the training conditions of the individual instances, the suffix "`random`" is used for instances trained with DR and "`fixed`" for those trained with the static scenario. In accordance with this naming convention, the following TransferLight models are included in the evaluation framework:

- **`TransferLight-DQN-random`**
- **`TransferLight-A2C-random`**
- **`TransferLight-DQN-fixed`**
- **`TransferLight-A2C-fixed`**

Baselines

To establish a benchmark, the TransferLight variants are compared with the following baselines:

- `FixedTime`: This method uses a cyclic signal timing plan with a fixed cycle length and phase split for each signalized intersection. The `random-heavy` and `random-light` test scenarios use the default timing parameters provided by SUMO when creating the road network. Similarly, the real-world test scenarios `cologne3`, `cologne8`, and `ingolstadt7` adopt the default settings provided by the RESCO [16] benchmark. In the `arterial` scenario, however, the cycle length and the phase split of each intersection are instead determined using the Webster method [40] and the signal timing offsets between successive intersections are calculated using GreenWave [41]. This guarantees that the timing parameters in this scenario are set optimally so that vehicle delay is minimized. For this purpose, a saturation flow rate of 1,700 vehicles per hour is used, based on Webster's findings [40]. The other parameters are determined according to the procedure described in Section 2.1.3 and Section 2.1.4. This results in a cycle length of ≈ 76.2 seconds with a green time of ≈ 40.9 seconds for the arterial movements and a green time of ≈ 25.3 seconds for the side street movements of each intersection, while successive intersections have a signal timing offset of ≈ 14.4 seconds.

- **`MaxPressure`** [39]: This method is a greedy TSC approach which always selects the phase with the maximum pressure at each intersection, resulting in a maximization of network throughput if the assumptions of the underlying traffic model hold (see Section 2.1.5). To enable a fair comparison, this method is allowed to change phases every 10 seconds, which corresponds to the same action duration for all trained TransferLight variants (see Section 4.1.3).
- **`PressLight`** [9]: This RL-based approach employs a theoretically motivated definition for rewards and states, a concept also adopted in TransferLight with some adaptations (see Section 4.1). Since PressLight implements its policy as a MLP, it is only trained and tested for the `arterial` scenario, as this scenario exclusively features homogeneous intersections, allowing for parameter sharing during training. In a scenario akin to the `arterial` scenario, the authors of PressLight showcase the efficacy of their reward and state definitions. They demonstrate that their method can learn to establish a green wave, competing favorably with the `FixedTime` baseline determined via the Webster [40] and GreenWave [41] methods.

5.2.3 Evaluation Metrics

To assess the performance of the compared methods, the following two network-level metrics, previously discussed in Section 2.1.2, are employed:

- **Travel time**: This is the most commonly used metric in the community to evaluate the performance of RL-based TSC approaches. It is measured as the time in seconds it takes a vehicle to complete a journey from a start lane to a destination lane. It is important to note that the evaluation of travel time is restricted to vehicle trips that arrive at their destination successfully. Vehicle trips that do not reach their destination will not be taken into account.

- **Throughput**: If the TSC method is not able to cope with the traffic demand, the road network may be heavily blocked so that no vehicle can proceed. As a result, these blocked vehicles cannot reach their destination and their travel time cannot be determined. Exclusive use of the travel times from arrived trips before the blockage can therefore lead to a distorted picture of the actual average travel time. To account for such worst-case scenarios, the methods are therefore also evaluated in terms of throughput. In this work, throughput is measured as the number of vehicles arriving in the last 60 seconds (i.e., the last minute). If there is then a blockage at any point in the road network, throughput drops, reflecting the fact that vehicles cannot reach their destination.

In the context of addressing research question Q3, all TransferLight variants are additionally compared with regard to their model uncertainty. For this purpose, the uncertainty of the Q-value (TransferLight-DQN) or probability (TransferLight-A2C) of each selected action is estimated from 100 MC dropout samples (see Section 2.2.5).

5.3 Evaluation Results

In this section, the evaluation framework developed in the previous section is utilized to address the research questions posed. Firstly, the performance evolution of the TransferLight models during training is analyzed to gain insights into the rate and stability of the models' performance improvements in the training configurations, and whether the performance remains consistent at some point. This analysis is also employed to investigate whether the specific training of TransferLight on the fixed scenario leads to overfitting. Subsequently, the zero-shot transfer performance of the TransferLight models on the developed test scenarios is examined and and compared to the considered baselines. A closer examination of the `arterial` test scenario is then conducted to investigate how well the compared methods are able to produce a green wave. Lastly, the model uncertainty is compared between the TransferLight models

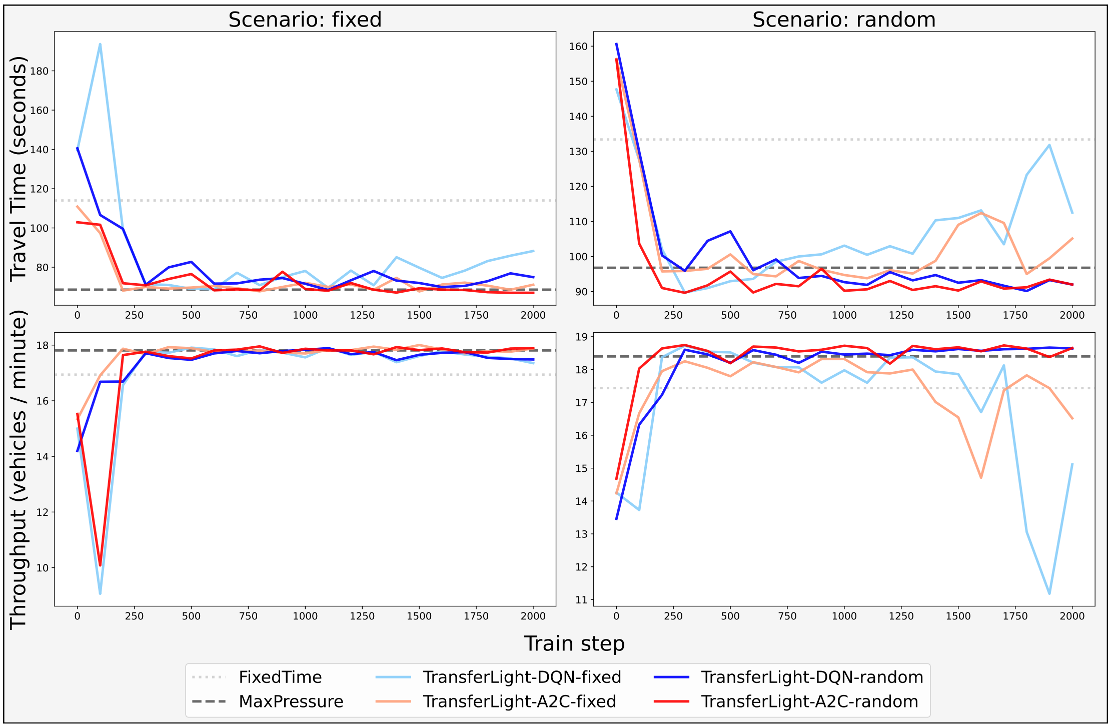


Figure 5.2: Learning curves for different evaluation metrics and training scenarios

on the test scenarios to examine whether training with the proposed DR mechanism leads to more confident predictions.

5.3.1 Performance Evolution during Training

To investigate the evolution of the compared TransferLight models throughout the training process, their performance is evaluated after every 100th training step. This evaluation is conducted for both the fixed scenario used to train `TransferLight-DQN-fixed` and `TransferLight-A2C-fixed`, as well as 10 newly sampled scenarios from the proposed DR mechanism that were not observed by any TransferLight model during training. Therefore, the 10 random scenarios represent new traffic conditions for all TransferLight models, while the fixed scenario is only novel for `TransferLight-DQN-random` and `TransferLight-A2C-random`. The resulting learning curves for both the average travel time and the average throughput are depicted in Figure 5.2. To facilitate the classification of the results, the performance of `FixedTime` and `MaxPressure` in the corresponding scenarios is marked with gray dotted and dashed horizontal lines, respectively.

The learning curves indicate that all TransferLight models exhibit rapid improvement in their respective training configuration in the first 250 training steps, both in terms of average travel time and throughput, despite an initial drop in performance of `TransferLight-DQN-fixed`, from which the model can quickly recover. Subsequently, the initial rapid improvement of all models plateaus and reaches a relatively stable performance. Both `TransferLight-DQN-fixed` and `TransferLight-A2C-fixed` level off at an average travel time of under 75 seconds and an average throughput of roughly 18 vehicles per minute on the fixed scenario. Hereby, they outperform `FixedTime` and more or less reach the same level of performance as `MaxPressure`. `TransferLight-DQN-random` and `TransferLight-A2C-random`, on the other hand, level off at an average travel time below 100 seconds and an average throughput above 18 vehicles per minute on the random scenarios, outperforming both `FixedTime` and `MaxPressure`.

By evaluating the evolution of both TransferLight variants trained using DR on the fixed scenario, it can be observed that they are capable of competing with the other two TransferLight variants that are specifically trained on the fixed scenario. Notably, this observation does not hold true in the opposite direction. If the evolution of the TransferLight variants trained solely on the fixed scenario is evaluated on the random scenarios, they initially exhibit improvement with respect to these scenarios as well. However, after around 1,250 training steps, their performance significantly drops in the random scenarios, even underperforming `FixedTime` in terms of average throughput, despite still performing well in the fixed scenario.

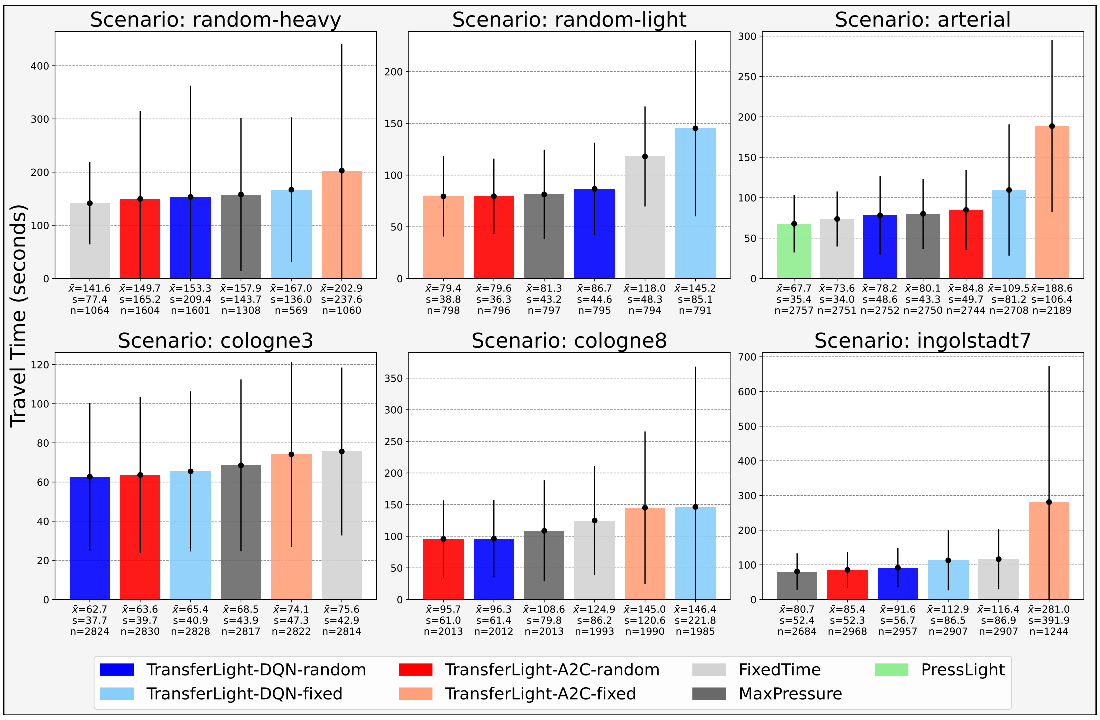
With regards to research question Q1, the findings of this analysis provide compelling evidence that despite the extensive utilization of parameter sharing between agents and individual components of the graph-structured state representation, training TransferLight exclusively on a fixed scenario does, in fact, results in overfitting and, as a consequence, diminished generalization to novel traffic conditions. However, if TransferLight is trained with the proposed DR mechanism, the results show that TransferLight achieves a stable and competitive performance across scenarios from the randomization space. This suggests that the employed DR mechanism leads to better generalization of TransferLight to novel traffic conditions.

5.3.2 Zero-Shot Transfer Performance

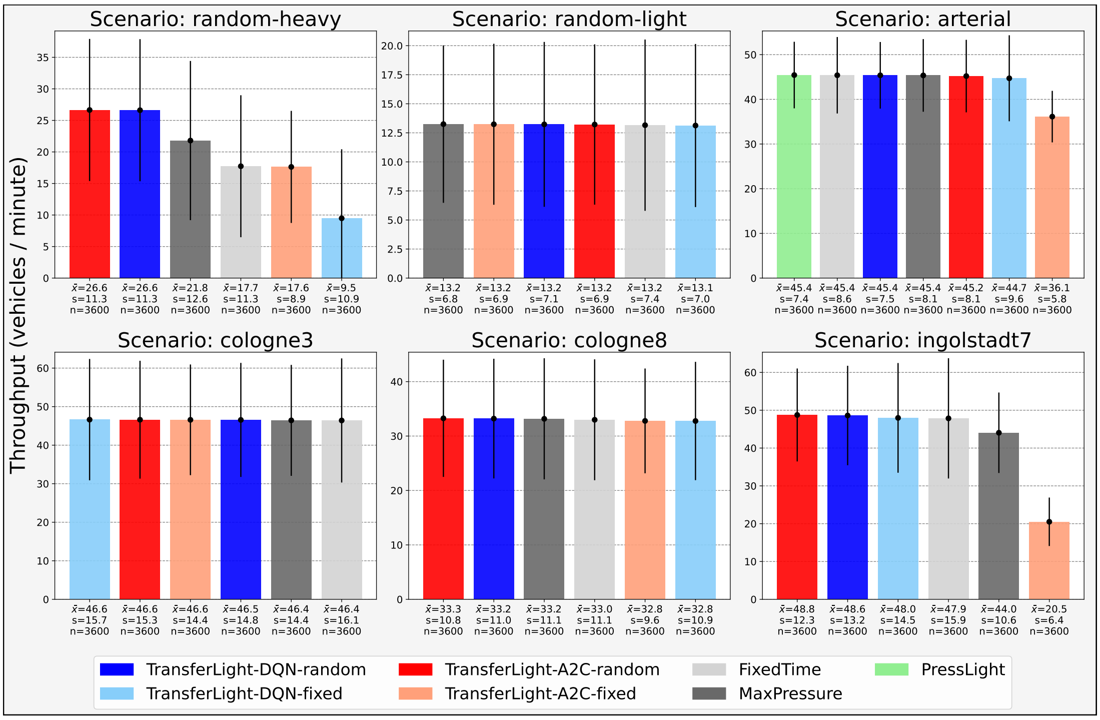
The previous section has demonstrated that TransferLight better generalizes to novel scenarios from the DR space by employing DR during training. However, it is important to note that the individual randomization parameters of the space are bounded. Consequently, the set of scenarios that can be sampled from the DR space is limited and may not encompass real-world traffic dynamics. To better evaluate how TransferLight transfers to such scenarios, it is therefore crucial to assess its performance on scenarios that lie outside of the space considered during training. For this purpose, Section 5.2.1 presented a set of synthetically generated and real-world test scenarios that are used in this section to evaluate the zero-shot transfer performance of TransferLight. The bar charts in Figure 5.3 illustrate the performance of the compared methods in these test scenarios with respect to travel time (Figure 5.3(a)) and throughput (Figure 5.3(b)). To facilitate a more rigorous quantitative comparison, the statistical parameters of the sample, including the mean, standard deviation, and sample size, denoted by $\bar{x}$, s, and n, respectively, are presented below each bar.

When examining the individual graphs, it is evident that the TransferLight models trained exclusively on a single scenario exhibit significant differences in performance across various test scenarios. For instance, `TransferLight-A2C-fixed` demonstrates reasonable performance in the `random-light` scenario and is competitive with `TransferLight-A2C-random` and `MaxPressure` in terms of average travel time. However, its performance drops significantly in alternative scenarios, with the average travel time sometimes doubling (`arterial` scenario) or even tripling (`ingolstadt7` scenario) compared to the best method. Similarly, the average travel time of `TransferLight-DQN-fixed` in the scenario `cologne3` lags slightly behind the best performing models, `TransferLight-A2C-random` and `TransferLight-DQN-random`. In all other test scenarios, however, `TransferLight-DQN-fixed` performs significantly worse and is consistently outperformed by the baseline methods used for comparison.

When TransferLight is trained utilizing the proposed DR approach instead, the bar charts reveal that the resulting policy appears to be more consistent and competitive, without manifesting a substantial decline in performance in any of the test scenarios considered. Notably, in many of the test scenarios, both `TransferLight-A2C-random` and `TransferLight-DQN-random` surpass alternative methods. This is particularly evident in the scenarios from Cologne, where the models result in vehicles arriving on average around 5 seconds faster in scenario `cologne3` or even over 12 seconds faster in scenario `cologne8` compared to the next best baseline `MaxPressure`. Also in the `random-heavy` scenario, both `TransferLight-A2C-random` and `TransferLight-DQN-random` exhibit a significantly higher average throughput of 26.6 vehicles per minute (almost 5 vehicles per minute more than the next best baseline `MaxPressure`),



(a) Travel Time



(b) Throughput

Figure 5.3: Performance of compared methods on test scenarios with respect to different performance metrics

a metric that is otherwise broadly similar between the compared methods in the other scenarios. Nevertheless, in other situations, `TransferLight-A2C-random` and `TransferLight-DQN-random` exhibit marginally inferior performance compared to a subset of alternative methods. This discrepancy is notably observed in the `arterial` scenario, where `PressLight` and `FixedTime` outperform with average travel times of 67.7 seconds and 73.6 seconds, respectively. Nevertheless, it is crucial to note that `PressLight`'s parameters and `FixedTime`'s signal timing configuration were specifically tailored to perform well in the `arterial` scenario. In contrast, `TransferLight-DQN-random` and `TransferLight-A2C-random` must adapt to the `arterial` scenario based on acquired knowledge from other scenarios, yet still demonstrate competitive average travel times of 78.2 and 84.8 seconds, respectively, exceeding or closely matching the performance of `MaxPressure` (80.1 seconds). Another prominent observation is that `FixedTime` seems to outperform both `TransferLight-A2C-random` and `TransferLight-DQN-random` in the `random-heavy` scenario with respect to average travel time. However, upon closer examination, it becomes apparent that the average travel time estimates for all methods, except `TransferLight-A2C-random` and `TransferLight-DQN-random`, are derived from conspicuously small sample sizes. Specifically, while the mean travel time estimates for `TransferLight-A2C-random` and `TransferLight-DQN-random` are based on 1604 and 1601 vehicle trips, respectively, the estimate for `MaxPressure` is based on a sample size of merely 1308 trips, and the sample size of the other methods is even lower. These small samples suggest that some trips that arrive when using `TransferLight-A2C-random` or `TransferLight-DQN-random` do not reach their destination when using one of the other approaches. This observation is also supported by the average throughput, which, as previously mentioned, is highest for both `TransferLight-A2C-random` and `TransferLight-DQN-random` and significantly lower for the other methods. A demonstration using the *sumo-gui*[2] tool reveals the cause of this phenomenon. With the exception of `TransferLight-A2C-random` and `TransferLight-DQN-random`, the signal control policy employed by the other methods results in a congested road network from which no vehicle can extricate (see Appendix A.1 for more details). Consequently, `TransferLight-A2C-random` and `TransferLight-DQN-random` are the only methods that can manage the high traffic demand in the `random-heavy` scenario and ultimately prevent the congestion.

Overall, the outcomes of the performance comparison in the test scenarios suggest that the signal control policy learned by TransferLight can adapt to a variety of traffic

---

[2] `https://sumo.dlr.de/docs/sumo-gui.html` (accessed 30.11.2023)

conditions, including practical ones, more effectively if the proposed DR technique is utilized during training. This allows TransferLight to perform on par with established baselines in many situations in a more consistent manner and even outperform them in some cases. Furthermore, the fact that TransferLight can prevent congestion in a high demand setting, where all other methods fail, suggests that the DR mechanism enables TransferLight to learn a more robust policy that can better cope with elevated demand. With regard to research question Q2, this strongly implies that the proposed DR mechanism indeed leads to a policy whose zero-shot transfer performance is significantly improved. However, the superior performance of other custom-tailored methods in specific situations suggests that the DR approach does not endow TransferLight with the requisite knowledge to operate optimally across scenarios.

5.3.3 Signal Progression

The performance evaluation conducted in the preceding section has revealed an improvement in TransferLight's adaptability to unobserved traffic conditions through the proposed DR approach. However, when compared to methods specifically tuned for the `arterial` scenario (i.e., `FixedTime` and `PressLight`), TransferLight exhibited a performance drawback. To gain deeper insights into the underlying causes of this performance deficit, this section delves into an introspective analysis of the policy employed by the individual methods in navigating traffic through the arterial street. To this end, Figure 5.4 presents *time-space diagrams* for different signal control methods utilized in the arterial scenario, including `TransferLight-DQN-arterial`, a new TransferLight-DQN model trained specifically on the `arterial` scenario to investigate whether the model architecture has sufficient capacity to learn a similar behavior to the best performing method `PressLight`. Each diagram depicts the trajectories of vehicles moving along the arterial road from the southernmost to the northernmost lane within a time frame of 900 to 1,200 seconds from the beginning of the simulation. The signal timing sequence (i.e., the red, yellow, green sequence) for the arterial movements of each intersection is plotted along the time axis at the distance where the intersections are situated. To facilitate a quantitative comparison, the upper section of each diagram presents the mean and standard deviation of the green times, the green time offsets of consecutive intersections, and the travel times of the vehicles traversing the arterial road.

The diagrams demonstrate that the methods specifically tailored to the arterial scenario, namely `FixedTime`, `PressLight` and `TransferLight-DQN-arterial`, are

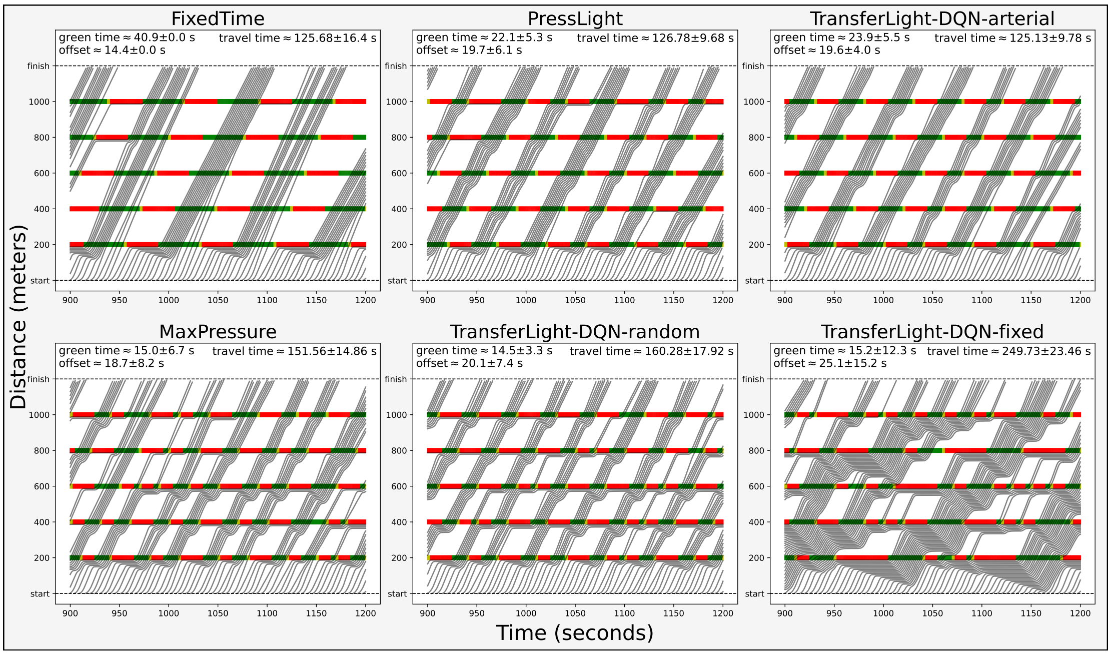


Figure 5.4: Time-space diagrams for different TSC methods employed in the `arterial` scenario

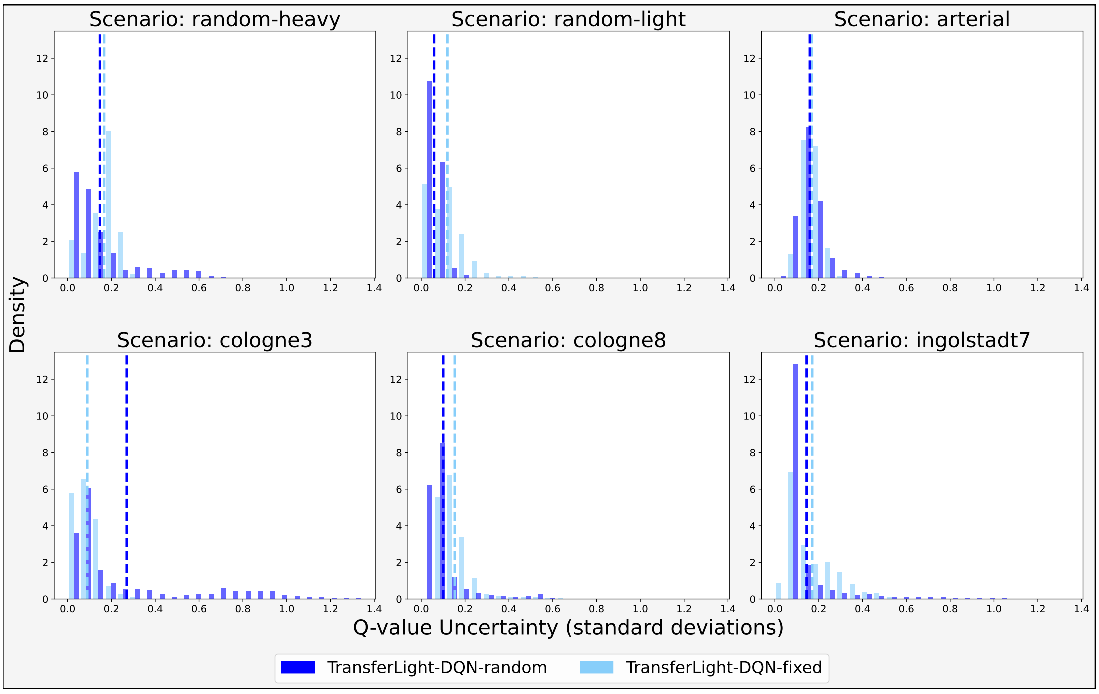
highly effective in generating green waves. This is evident from the individual vehicle trajectories, which indicate that vehicles can traverse the entire arterial without having to decelerate sharply, with only occasional stops at intersections to wait for the next green signal indication. As a result, the average travel time along the arterial using these methods is quite low compared to the other methods, at around 125-127 seconds. An interesting observation is that the learned signal timing parameters of `PressLight` and `TransferLight-DQN-arterial` are quite similar, while they differ substantially from those of `FixedTime`. For example, the average green time of `PressLight` and `TransferLight-DQN-arterial` is 22.1 and 23.9 seconds, respectively, whereas the green time of `FixedTime` is almost twice as long, at 40.9 seconds. Furthermore, it is evident that both `PressLight` and `TransferLight-DQN-arterial` do not use a consistent green time, but instead adjust the green time according to the current demand, which is also reflected in the increased standard deviation. Similarly, the average green time offset of the learned solution is roughly 5 seconds longer than that of `FixedTime`. One possible explanation for the difference in signal timing parameters could be that both `PressLight` and `TransferLight-DQN-arterial` can only switch phases every 10 seconds. Due to this limited granularity, the policies may not be able to generate a signal timing sequence that matches that of `FixedTime` and may instead have learned a policy that best exploits these restrictions. However, as demonstrated in the performance comparison in Section 5.3.2, these restrictions do not impede `PressLight` (and most likely `TransferLight-DQN-arterial`) from producing a signal timing sequence that results in a lower average travel time than `FixedTime`.

In contrast to the methods tailored to the `arterial` scenario, the signal timing sequences of the other methods used appear noticeably more irregular often involving green times lasting less than 10 seconds. Despite this irregularity, `MaxPressure` and `TransferLight-DQN-random` exhibit a green time offset of 18.7 and 20.1, respectively, similar to the one of `PressLight` and `TransferLight-DQN-arterial`. However, the cascading of green times more often appears suboptimal with green times ending too early such that platoons of vehicles cannot pass through the arterial in their entirety. As a consequence, vehicles are queuing more frequently waiting for the next signal indication. This is conspicuously present for `TransferLight-DQN-fixed`, where vehicle queues extend dangerously close to previous intersections. Due to this suboptimal signal progression, the average travel time for vehicles traversing the arterial route increases substantially, measuring 151.56 seconds for `MaxPressure`, 160.28 seconds for `TransferLight-DQN-random` and even more pronounced 249.73 seconds for `TransferLight-DQN-fixed`.

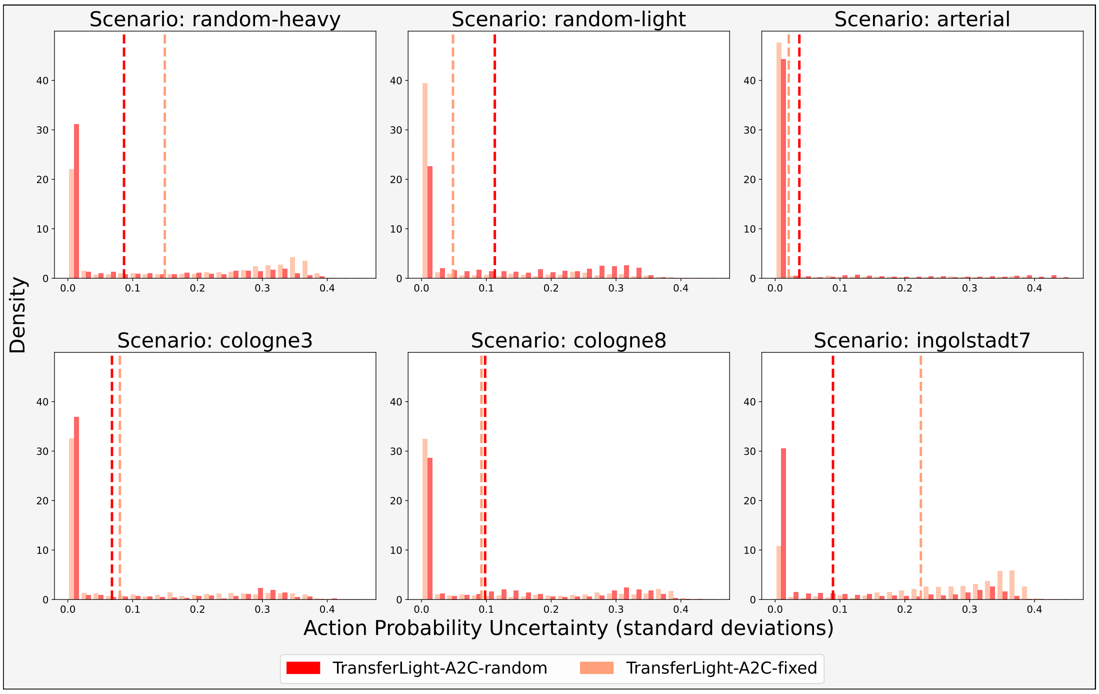
Concerning research question Q2, the time-space diagrams for the `arterial` scenario affirm the findings from the previous section. Despite the observed performance improvement, the proposed DR approach does not necessarily enable TransferLight to perform optimally when compared to tailored solutions. However, the results also indicate that the model architecture of TransferLight is not the root cause of the performance deficit in the `arterial` scenario. TransferLight can learn a more ideal behavior, matching that of `PressLight`, if specifically trained on it. This observation may suggest that the proposed domain randomization does not fully leverage the capacity of TransferLight.

5.3.4 Model Uncertainty

The proposed DR method substantially expands the spectrum of traffic conditions from which TransferLight can acquire knowledge. As a result, it is highly probable that the uncertainties associated with the model's response to variations in unfamiliar traffic conditions diminishes. This section aims to investigate whether and to what extent this is the case. To this end, Figure 5.5 shows histograms, each representing the distribution



(a) `TransferLight-DQN-fixed` vs. `TransferLight-DQN-random`



(b) `TransferLight-A2C-fixed` vs. `TransferLight-A2C-random`

Figure 5.5: Histograms of the model uncertainty estimates recorded by different TransferLight models in different test scenarios

of the recorded MC dropout uncertainty estimates for a particular TransferLight model in a particular test scenario. While Figure 5.5(a) compares the Q-value uncertainties between `TransferLight-DQN-fixed` and `TransferLight-DQN-random`, Figure 5.5(b) compares the action probability uncertainties between `TransferLight-A2C-fixed` and `TransferLight-A2C-random`. In all diagrams, the corresponding mean uncertainties are indicated by dashed vertical lines.

Upon a closer examination of Figure 5.5(a), a noteworthy initial observation is that the distributional patterns of Q-value uncertainties, both for `TransferLight-DQN-random` and `TransferLight-DQN-fixed`, exhibit a lack of consistency across various test scenarios. For instance, in the `random-light` scenario, `TransferLight-DQN-random` displays a relatively high level of certainty, as evident by the concentration of Q-value uncertainties around zero. Conversely, in alternative scenarios, the model demonstrates increased uncertainty, characterized by a distributional peak shifted towards higher uncertainties (`arterial` scenario) a more dispersed pattern with an elongated tail towards higher uncertainties (`random-heavy` scenario), or a combination of both phenomena (all real-world scenarios). By comparing the uncertainty distributions of `TransferLight-DQN-fixed` and `TransferLight-DQN-random`, it can be observed that in certain test scenarios, `TransferLight-DQN-random` exhibits less uncertainty in the Q-values than `TransferLight-DQN-fixed`. This trend is particularly noticeable in the random-heavy scenario, where `TransferLight-DQN-random`, despite having a broader spread, attains a higher density of uncertainties proximate to 0 compared to `TransferLight-DQN-fixed`, whose uncertainty distribution demonstrates a distinctly elevated peak just below 0.2. A similar pattern is discernible in scenarios such as `random-light`, `cologne8`, and `ingolstadt7`, where `TransferLight-DQN-random` manifests an augmented density towards lower uncertainties in contrast to `TransferLight-DQN-fixed`. Yet, in the `arterial` and `cologne3` scenarios, despite the advantage in average travel time of `TransferLight-DQN-random` over `TransferLight-DQN-fixed` (see Section 5.3.2), the former does not necessarily exhibit superior certainty in these contexts compared to the latter. In fact, their uncertainty distributions appear quite similar in the `arterial` scenario, while in the `cologne3` scenario, `TransferLight-DQN-random` displays a significantly broader spread towards higher uncertainties than `TransferLight-DQN-fixed`, resulting in a mean uncertainty that is notably lower for `TransferLight-DQN-fixed` than for `TransferLight-DQN-random`.

In contrast to the uncertainty distributions associated with Q-values, the distributions characterizing uncertainties in action probabilities, as depicted in Figure 5.5(b), exhibit a relatively greater degree of uniformity across TransferLight-A2C models and various test scenarios. Notably, these distributions manifest a conspicuous and concentrated peak of uncertainties around 0, with a flat tail extending to uncertainties reaching as high as roughly 0.4. The principal distinction among these distributions lies predominantly in the relative densities assigned to the tail in relation to the peak. In specific test scenarios, the peak of `TransferLight-A2C-random` conspicuously exceeds that of `TransferLight-A2C-fixed`, implying that the former learned a more confident policy. This discrepancy is particularly pronounced in the `ingolstadt7` scenario and, to a lesser extent, evident in the `random-heavy` scenario. Nonetheless, analogous to the uncertainty distributions associated with Q-values, there exist counterexamples that challenge a definitive assertion of heightened confidence in action probabilities for `TransferLight-A2C-random` over `TransferLight-A2C-fixed` across varying traffic conditions. Specifically, in the `arterial`, `cologne3`, and `cologne8` scenarios, the uncertainty distributions appear relatively similar, while in the `random-light` scenario, `TransferLight-A2C-fixed` exhibits a greater degree of certainty than `TransferLight-A2C-random`, as evidenced by the prominently higher peak of `TransferLight-A2C-fixed`.

In addressing the third research question Q3, the findings from the histograms indicate that the integration of the proposed DR mechanism does not result in a TransferLight model capable of generating predictions with consistently low uncertainty across various traffic scenarios. While there is a discernible trend towards lower uncertainties, particularly for the TransferLight-A2C models, it is noteworthy that under certain test scenarios, the distributions shift or spread more prominently towards higher uncertainties, suggesting that such scenarios encompass traffic conditions the model is unable to respond to with high confidence. The cause of this lack of confidence could be attributed to various factors. One possibility may be that such testing conditions differ too strongly from the conditions the model encountered during training. This could indicate that the diversity or range of scenarios included in the randomization during training is not extensive enough to provide the model with sufficient examples resembling such traffic conditions. Apart from the DR mechanism, it is plausible that the TransferLight architecture may lack the capacity to effectively reduce uncertainty associated with specific types of traffic conditions. For instance, this limitation could arise due to the absence of crucial connections in the graph-structured state representation.

It is also noteworthy that TransferLight models trained solely on fixed scenarios exhibit comparable or even superior confidence levels in certain test scenarios, despite having substantially fewer traffic conditions to explore during training. However, this apparent confidence may be attributed to overfitting, wherein these models might have developed an inflated certainty in their predictions based on the specific traffic conditions encountered during training. This overconfidence could have misleadingly carried over into the test scenarios, suggesting that the models' perceived certainty in decision-making does not necessarily correlate with their actual effectiveness in these scenarios. Moreover, it may be generally misleading to compare uncertainty distributions between different models in this manner, as the models act independently, leading to substantially different trajectories and, accordingly, completely different traffic situations observed along the way. A more accurate comparison of uncertainties across models would involve examining uncertainties for the same states. This would provide a more direct comparison of how certain models handle identical traffic conditions. Due to timing constraints, however, a follow-up analysis following this approach was not feasible in this study. This could be an interesting avenue for future research, offering a more direct comparison of how certain different models handle the same traffic conditions.

6
Conclusion

6.1 Summary

The primary objective of this thesis was to evaluate the effectiveness of DR in improving the transferability of a signal control policy to unfamiliar traffic scenarios not encountered during training. To achieve this, a DR mechanism was developed, introducing variability in environmental parameters associated with the road network and traffic for each training episode. Additionally, a model for the policy of an intersection, named TransferLight, was proposed, to foster generalization across intersection types and traffic conditions.

In pursuit of the primary goal of this study, the following research questions were addressed:

Q1: *Does TransferLight exhibit overfitting when trained exclusively on a fixed scenario?*

Q2: *To what extent does the zero-shot transfer performance of TransferLight experience improvement when trained in conjunction with the suggested DR mechanism, and what are the potential limitations associated with the effectiveness of this approach?*

Q3: *To what extent does the integration of the proposed DR mechanism into the training procedure diminish the uncertainty in the predictions made by TransferLight when confronted with novel traffic scenarios?*

Evaluation results indicated that TransferLight indeed suffered from significant overfitting when trained solely on a fixed scenario. This overfitting led to inconsistent performance outcomes on scenarios that differed from the training scenario, with the model often performing substantially worse than established baselines.

In contrast, when TransferLight was trained using the proposed DR mechanism, it exhibited improved generalization across a wide range of traffic conditions. Additionally, DR resulted in an enhanced zero-shot transfer performance of TransferLight, as demonstrated by consistently strong travel time and throughput results across a set of test scenarios specifically chosen for their deviation from the training conditions. In these scenarios, the model competed favorably with established baselines and, in many instances, outperformed them. However, within the context of an arterial road network scenario, the approach fell short of methods specifically designed for this setting. Further investigation revealed that these specialized methods excelled in forming green waves, leading to fewer vehicle stops along the arterial road and, consequently, reduced travel time.

An analysis of recorded model uncertainty estimates did not imply that the integration of the proposed DR mechanism enables TransferLight to consistently act with high confidence across traffic situations. Although a trend towards low uncertainties was observed, the uncertainty distributions varied between test scenarios. Some scenarios showed a significant spread or shift towards higher uncertainties, suggesting that these scenarios presented more unfamiliar traffic conditions to the model. Interestingly, if TransferLight was exclusively trained on a fixed scenario, providing substantially less traffic conditions to explore, it often exhibited comparable or even superior confidence in its predictions, despite its performance deficit. This could arguably be attributed to a misleadingly inflated confidence resulting from overfitting on the fixed scenario.

In light of these insights, it can be concluded that the proposed DR mechanism in this work provides an effective method to prevent a signal control policy from overfitting, thereby enhancing its zero-shot transfer performance to traffic scenarios not encountered during training. Nevertheless, it is important to note that despite the performance improvement, the mechanism does not necessarily equip the signal control policy with the required knowledge to act ideally and with high confidence across all traffic conditions, as the results of this work have indicated.

6.2 Limitations

While the proposed DR mechanism has shown promising results, it is important to acknowledge certain limitations of this work.

A significant limitation of this study lies in the assumptions made about traffic. The training process was confined to scenarios involving only passenger cars, overlooking the diversity of real-world traffic participants such as public transport vehicles, emergency vehicles, pedestrians, and cyclists. This could limit the practical applicability of the learned signal control policy, as it may not fully account for the unique behaviors and needs of these varied participants. Additionally, the training assumed a uniform speed limit of 50 kilometers per hour, which does not reflect the variability of speed limits in real-world scenarios due to factors like road type, location, and local regulations. This could potentially affect the adaptability of the learned signal control policy to different speed limit scenarios.

An additional limitation pertains to the assumptions made about the capabilities of traffic detectors. The training process was conducted under the assumption that traffic detectors installed at intersections can sense traffic with fine granularity and without any measurement errors. However, in practical scenarios, these detectors may not always provide such precise data and are often subject to various types of measurement errors. This discrepancy between the assumed and actual capabilities of traffic detectors could potentially impact the applicability and reliability of the learned signal control policies when applied in real-world scenarios.

Furthermore, this study acknowledges the limitation concerning the evaluation of the learned signal control policy. The evaluation was conducted on a limited number of test scenarios, which may not comprehensively represent all possible traffic conditions. There is a possibility that the learned policy may not perform as expected in untested scenarios. In addition, the effectiveness of the learned signal control policy was measured only in terms of travel time and throughput. However, there are other significant metrics such as waiting time, fuel consumption and emission levels that were not considered in this study. The exclusion of these metrics may limit the comprehensiveness of the evaluation and the practical applicability of the learned signal control policy.

6.3 Future Work

While this study has made various contributions in terms of zero-shot transfer learning in the context of TSC, there are several promising avenues for future research that could further advance this field.

One potential direction is to enhance the DR mechanism by extending the diversity of traffic conditions. This could be achieved by expanding the ranges of domain parameters, such as lane length or the alpha and beta parameters characterizing traffic flows. Another avenue for augmentation involves introducing variability to variables that were previously fixed, such as the number of simulated vehicles in each episode. Furthermore, integrating variations in different types of traffic participants, speed limits, and accounting for measurement errors from traffic detectors could add a layer of realism and versatility to the mechanism. These enhancements might further improve the model's zero-shot transfer performance and increase its confidence, possibly enabling it to respond more ideally across a wider range of traffic conditions and enhancing its robustness. Consequently, the reliability of the signal control policy could be significantly strengthened, making it more suitable for real-world deployment.

In addition to enhancing the DR mechanism, potential adjustments to the model could further bolster its capacity and coordination abilities for improved performance in diverse traffic scenarios. One avenue of exploration could be drawn from previous works that leverage neural message passing for intersection communication to enhance their cooperation [21, 8]. For TransferLight, this could be achieved by connecting nodes of traffic movements at one intersection to corresponding nodes of downstream or upstream traffic movements at neighboring intersections within the graph-structured state representation. Another extension to this, demonstrated by previous researchers [14, 24, 25], involves integrating the history of past state representations using a spatio-temporal GNN. These enhancements would not only augment the model's capacity but also improve the coordination ability of the joint policy, potentially reducing model uncertainties associated with insufficient capacity. Additionally, a model designed to accommodate varying traffic sensing granularity levels for different detector types could be explored. This could, for example, be achieved by distinguishing between segment node types representing different lengths based on detector characteristics and training the architecture in an end-to-end fashion. Such adaptability could enhance the model's applicability to a broader range of intersection types and detection technologies, further advancing its real-world deployment potential.

Another avenue for future research could be the development of a standardized testing benchmark to assess the zero-shot transfer capabilities of learned signal control policies more comprehensively. As discussed earlier, the limitations of the evaluation approach in this study suggest the need for such a framework to cover a more diverse set of traffic scenarios and include assessment metrics that incorporate more real-world considerations. Guidance, such as that from the *Handbuch für die Bemessung von Straßenverkehrsanlagen* [107] could be instrumental in constructing a benchmark that incorporates real-world considerations, offering an established grading system for the effectiveness of signal controllers based on the average waiting time for different traffic participants. Building such a benchmark could benefit future researchers by allowing them to compare their solutions in a more fair and nuanced way. Additionally, it could provide a way to better assess how trustworthy such systems are, further reducing hurdles to bring these systems to production.

6.4 Implications

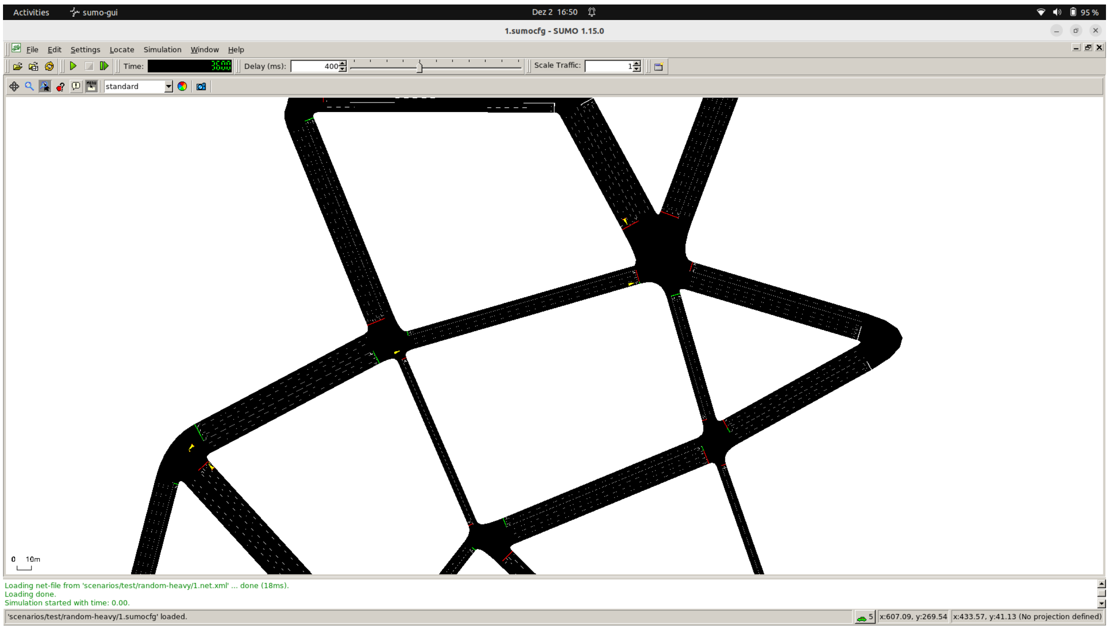
There is still a long way ahead until learned signal controllers are reliable enough to be deployed into the real world. In the ongoing advancements in this field, this work contributes to the growing body of research by demonstrating the critical importance of training signal control policies on a diverse range of traffic scenarios to prevent the policies from overfitting and ensure that they can adapt seamlessly to dynamic and varied traffic conditions without substantially sacrificing performance. The DR mechanism, proposed in this work, stands out for its simplicity and effectiveness in generating sufficiently varied traffic scenarios during training. Future works could adopt or explore the extension of the proposed mechanism, leveraging its ideas to enhance the robustness of learned signal controllers in facing the complexities of real-world traffic scenarios. As these advancements accumulate, it is conceivable that the insights gained from this research may, in time, contribute a crucial puzzle piece, potentially bringing us closer to the reliable deployment of intelligent signal control systems in real-world traffic management capable of effectively handling the ever-increasing traffic demand in urban areas.

A

Appendix

A.1 Traffic Congestion in the `random-heavy` Scenario

The subsequent figures show screenshots of the *sumo-gui*[1] tool after 3,600 seconds of simulated traffic in the scenario `random-heavy` when different TSC methods are used. The yellow triangles correspond to individual vehicles. Evidently, TransferLight models trained with DR produce an essentially unimpeded road network (Figure A.1-A.2). Conversely, the use of alternative methods leads to a pronounced traffic congestion that makes it impossible for vehicles to continue their journey (Figure A.3-A.6).



Time: 3600

Delay (ms): 400

Scale Traffic: 1

```
Loading net-file from 'scenarios/test/random-heavy/1.net.xml' ... done (18ms).
Loading done.
Simulation started with time: 0.00.
'scenarios/test/random-heavy/1.sumocfg' loaded.
```

5

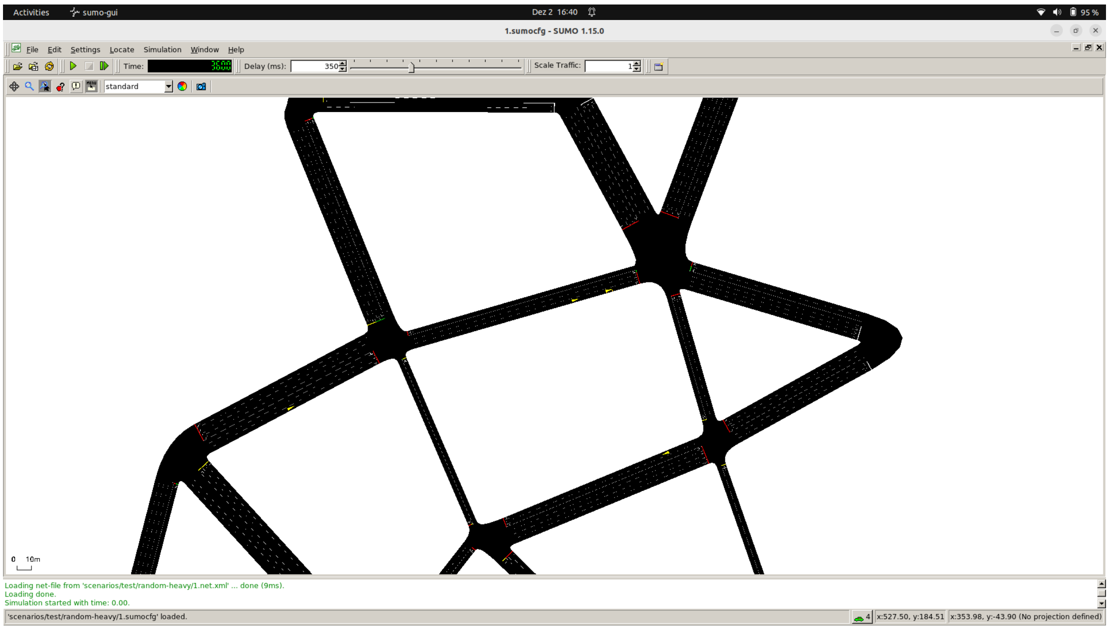
x:607.09, y:269.54

x:433.57, y:41.13 (No projection defined)

Figure A.1: sumo-gui screenshot after 3,600 seconds of simulated traffic in the `random-heavy` scenario if `TransferLight-DQN-random` is employed

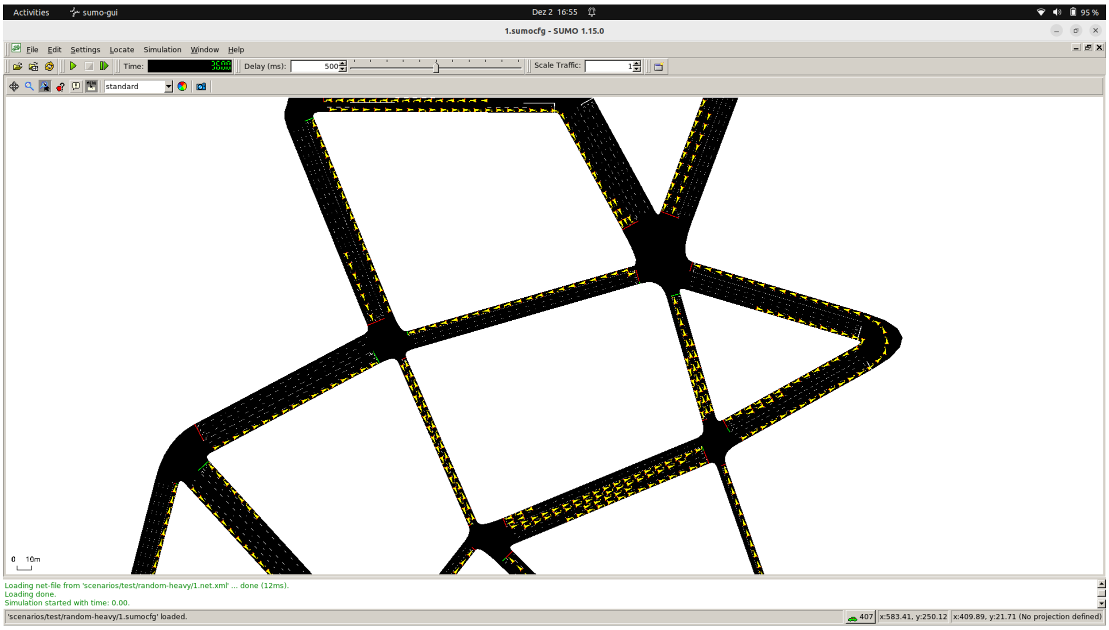
---

[1] `https://sumo.dlr.de/docs/sumo-gui.html` (accessed 30.11.2023)



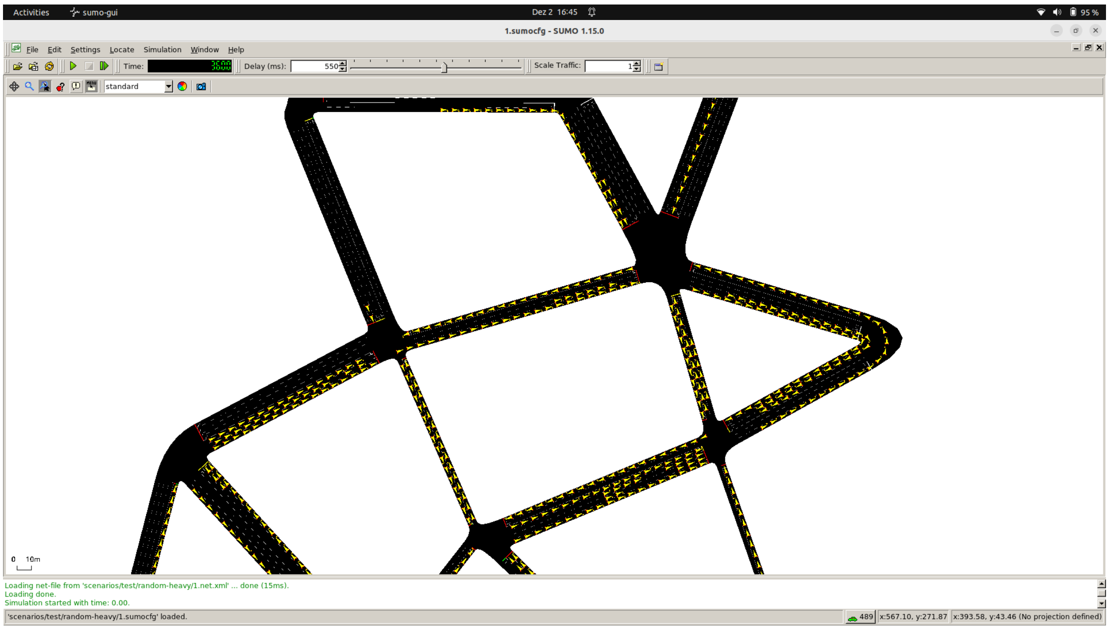
```
Loading net-file from 'scenarios/test/random-heavy/1.net.xml' ... done (9ms).
Loading done.
Simulation started with time: 0.00.
'scenarios/test/random-heavy/1.sumocfg' loaded.   4   x:527.50, y:184.51   x:353.98, y:-43.90 (No projection defined)
```

Figure A.2: sumo-gui screenshot after 3,600 seconds of simulated traffic in the `random-heavy` scenario if `TransferLight-A2C-random` is employed



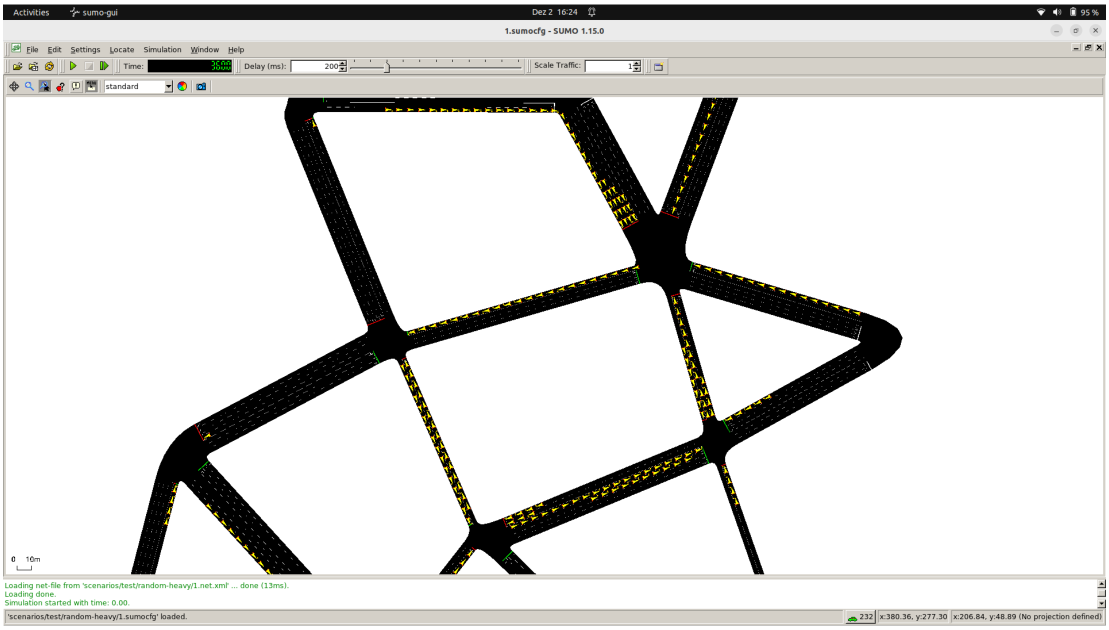
```
Loading net-file from 'scenarios/test/random-heavy/1.net.xml' ... done (12ms).
Loading done.
Simulation started with time: 0.00.
'scenarios/test/random-heavy/1.sumocfg' loaded.   407   x:583.41, y:250.12   x:409.89, y:21.71 (No projection defined)
```

Figure A.3: sumo-gui screenshot after 3,600 seconds of simulated traffic in the `random-heavy` scenario if `TransferLight-DQN-fixed` is employed



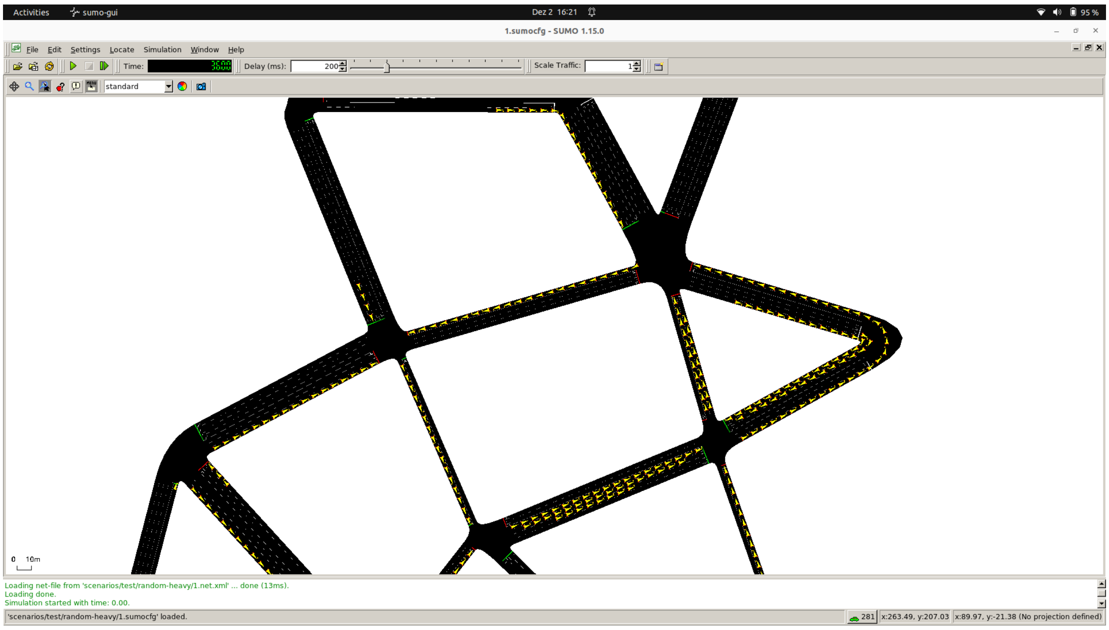
```
Loading net-file from 'scenarios/test/random-heavy/1.net.xml' ... done (15ms).
Loading done.
Simulation started with time: 0.00.
'scenarios/test/random-heavy/1.sumocfg' loaded.
```

Figure A.4: sumo-gui screenshot after 3,600 seconds of simulated traffic in the `random-heavy` scenario if `TransferLight-A2C-fixed` is employed



```
Loading net-file from 'scenarios/test/random-heavy/1.net.xml' ... done (13ms).
Loading done.
Simulation started with time: 0.00.
'scenarios/test/random-heavy/1.sumocfg' loaded.
```

Figure A.5: sumo-gui screenshot after 3,600 seconds of simulated traffic in the `random-heavy` scenario if `MaxPressure` is employed



Time: 3600
Delay (ms): 200
Scale Traffic: 1

```
Loading net-file from 'scenarios/test/random-heavy/1.net.xml' ... done (13ms).
Loading done.
Simulation started with time: 0.00.
'scenarios/test/random-heavy/1.sumocfg' loaded.
```

281 x:263.49, y:207.03 x:89.97, y:-21.38 (No projection defined)

Figure A.6: sumo-gui screenshot after 3,600 seconds of simulated traffic in the `random-heavy` scenario if `FixedTime` is employed

B

Abbreviations

A2C Advantage Actor-Critic

ANN Artificial Neural Network

DL Deep Learning

DQN Deep Q-Network

DRL Deep Reinforcement Learning

DR Domain Randomization

GPI Generalized Policy Iteration

GNN Graph Neural Network

Tanh Hyperbolic Tangent

i.i.d. independent and identically distributed

MC Monte Carlo

MSE Mean Squared Error

MDP Markov Decision Process

MG Markov Game

ML Machine Learning

MLE Maximum Likelihood Estimation

MLP Multilayer Perceptron

TE Transportation Engineering

PDF Probability Density Function

ReLU Rectified Linear Unit

RL Reinforcement Learning

TSC Traffic Signal Control

WGAN Wasserstein Generative Adversarial Network

C

List of Figures

2.1 Signalized four-way intersection 6
2.2 Computation graph of a single artificial neuron and a network of artificial neurons 16
2.3 Computation graph of a neural message passing layer 19
2.4 Two layers of neural message passing applied on a bidirectional example graph 21
2.5 Markov Decision Process 27

4.1 An example intersection and its corresponding graph-based state representation 47
4.2 Computation graph of a neural message passing layer in the graph embedding module 51
4.3 Computation graph of the MLP used in the message, scoring and update functions 52
4.4 Randomly generated road networks for different seeds 58
4.5 Random traffic generated for different traffic flows 60

5.1 Road networks of the test scenarios 66
5.2 Learning curves for different evaluation metrics and training scenarios . 71
5.3 Performance of compared methods on test scenarios with respect to different performance metrics 74
5.4 Time-space diagrams for different TSC methods employed in the `arterial` scenario 77

5.5 Histograms of the model uncertainty estimates recorded by different TransferLight models in different test scenarios 79

A.1 sumo-gui screenshot after 3,600 seconds of simulated traffic in the `random-heavy` scenario if `TransferLight-DQN-random` is employed . . 89
A.2 sumo-gui screenshot after 3,600 seconds of simulated traffic in the `random-heavy` scenario if `TransferLight-A2C-random` is employed . . 90
A.3 sumo-gui screenshot after 3,600 seconds of simulated traffic in the `random-heavy` scenario if `TransferLight-DQN-fixed` is employed . . . 90
A.4 sumo-gui screenshot after 3,600 seconds of simulated traffic in the `random-heavy` scenario if `TransferLight-A2C-fixed` is employed . . . 91
A.5 sumo-gui screenshot after 3,600 seconds of simulated traffic in the `random-heavy` scenario if `MaxPressure` is employed 91
A.6 sumo-gui screenshot after 3,600 seconds of simulated traffic in the `random-heavy` scenario if `FixedTime` is employed 92

D

List of Tables

2.1 Network-level TSC performance metrics 10
2.2 Intersection-level TSC performance metrics 11
2.3 Activation functions frequently used in DL 17
2.4 Cost functions frequently used in DL 22

4.1 Node / edge features of the graph-based state representation 49
4.2 Layers of the graph embedding module 50
4.3 Hyperparameters of the independent Q-learning algorithm 55
4.4 Hyperparameters of the independent Advantage Actor-Critic (A2C) algorithm 57

E

List of Algorithms

4.1 Independent Q-learning algorithm 54
4.2 Independent Advantage Actor-Critic (A2C) Algorithm 56

F

Bibliography

[1] B. Pishue, “2022 INRIX Global Traffic Scorecard.” INRIX, 2022. `https://inrix.com/scorecard` [Accessed: 19.07.2023].

[2] D. A. Hennessy and D. L. Wiesenthal, “Traffic congestion, driver stress, and driver aggression,” *Aggressive Behavior: Official Journal of the International Society for Research on Aggression*, vol. 25, no. 6, pp. 409–423, 1999.

[3] K. Zhang and S. Batterman, “Air Pollution and Health Risks due to Vehicle Traffic,” *Science of The Total Environment*, vol. 450–451, p. 307–316, Apr. 2013.

[4] M. Krzyzanowski, B. Kuna-Dibbert, and J. Schneider, *Health Effects of Transport-Related Air Pollution.* WHO Regional Office Europe, 2005.

[5] J. I. Levy, J. J. Buonocore, and K. von Stackelberg, “Evaluation of the public health impacts of traffic congestion: a health risk assessment,” *Environmental Health*, vol. 9, Oct. 2010.

[6] M. Barth and K. Boriboonsomsin, “Real-World Carbon Dioxide Impacts of Traffic Congestion,” *Transportation research record*, vol. 2058, no. 1, pp. 163–171, 2008.

[7] H. Wei, G. Zheng, H. Yao, and Z. Li, “Intellilight: A Reinforcement Learning Approach for Intelligent Traffic Light Control,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD ’18, ACM, July 2018.

[8] H. Wei, N. Xu, H. Zhang, G. Zheng, X. Zang, C. Chen, W. Zhang, Y. Zhu, K. Xu, and Z. Li, “CoLight: Learning Network-level Cooperation for Traffic Signal Control,” in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM ’19, ACM, Nov. 2019.

[9] H. Wei, C. Chen, G. Zheng, K. Wu, V. Gayah, K. Xu, and Z. Li, “Presslight: Learning Max Pressure Control to Coordinate Traffic Signals in Arterial Network,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD ’19, ACM, July 2019.

[10] G. Zheng, X. Zang, N. Xu, H. Wei, Z. Yu, V. Gayah, K. Xu, and Z. Li, “Diagnosing Reinforcement Learning for Traffic Signal Control,” *arXiv preprint arXiv:1905.04716*, 2019.

[11] G. Zheng, Y. Xiong, X. Zang, J. Feng, H. Wei, H. Zhang, Y. Li, K. Xu, and Z. Li, “Learning Phase Competition for Traffic Signal Control,” in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM ’19, ACM, Nov. 2019.

[12] A. Oroojlooy, M. Nazari, D. Hajinezhad, and J. Silva, “Attendlight: Universal Attention-based Reinforcement Learning Model for For Taffic Signal Control,” in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 4079–4090, Curran Associates, Inc., 2020.

[13] C. Chen, H. Wei, N. Xu, G. Zheng, M. Yang, Y. Xiong, K. Xu, and Z. Li, “Toward a Thousand Lights: Decentralized Deep Reinforcement Learning for Large-Scale Traffic Signal Control,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, p. 3414–3421, Apr. 2020.

[14] Y. Wang, T. Xu, X. Niu, C. Tan, E. Chen, and H. Xiong, “STMARL: A Spatio-Temporal Multi-Agent Reinforcement Learning Approach for Cooperative Traffic Light Control,” *IEEE Transactions on Mobile Computing*, vol. 21, p. 2228–2242, June 2022.

[15] X. Zang, H. Yao, G. Zheng, N. Xu, K. Xu, and Z. Li, “MetaLight: Value-based Meta-Reinforcement Learning for Traffic Signal Control,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, p. 1153–1160, Apr. 2020.

[16] J. Ault and G. Sharon, “Reinforcement Learning Benchmarks for Traffic Signal Control,” in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks* (J. Vanschoren and S. Yeung, eds.), vol. 1, Curran, 2021.

[17] L. Yan, L. Zhu, K. Song, Z. Yuan, Y. Yan, Y. Tang, and C. Peng, “Graph Cooperation Deep Reinforcement Learning for Ecological Urban Traffic Signal Control,” *Applied Intelligence*, vol. 53, p. 6248–6265, July 2022.

[18] B. Abdulhai, R. Pringle, and G. J. Karakoulas, “Reinforcement Learning for True Adaptive Traffic Signal Control,” *Journal of Transportation Engineering*, vol. 129, p. 278–285, May 2003.

[19] P. LA and S. Bhatnagar, “Reinforcement Learning with Function Approximation for Traffic Signal Control,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, p. 412–421, June 2011.

[20] S. El-Tantawy, B. Abdulhai, and H. Abdelgawad, “Multiagent Reinforcement Learning for Integrated Network of Adaptive Traffic Signal Controllers (MARLIN-ATSC): Methodology and Large-scale Aplication on Downtown Toronto,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, p. 1140–1150, Sept. 2013.

[21] T. Nishi, K. Otaki, K. Hayakawa, and T. Yoshimura, “Traffic Signal Control based on Reinforcement Learning with Graph Convolutional Neural Nets,” in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, Nov. 2018.

[22] H. Wei, C. Chen, K. Wu, G. Zheng, Z. Yu, V. Gayah, and Z. Li, “Deep Reinforcement Learning for Traffic Signal Control along Arterials,” *Proceedings of the 2019, DRL4KDD*, vol. 19, 2019.

[23] T. Chu, J. Wang, L. Codeca, and Z. Li, “Multi-Agent Deep Reinforcement Learning for Large-Scale Traffic Signal Control,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, p. 1086–1095, Mar. 2020.

[24] L. He, Q. Li, L. Wu, M. Wang, J. Li, and D. Wu, “A Spatial-Temporal Graph Attention Network for Multi-Intersection Traffic Light Control,” in *2021 International Joint Conference on Neural Networks (IJCNN)*, IEEE, July 2021.

[25] L. Wu, M. Wang, D. Wu, and J. Wu, “DynSTGAT: Dynamic Spatial-Temporal Graph Attention Network for Traffic Signal Control,” in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM ’21, ACM, Oct. 2021.

[26] E. Van der Pol and F. A. Oliehoek, “Coordinated Deep Reinforcement Learners for Traffic Light Control,” *Proceedings of Learning, Inference and Control of Multi-Agent Systems (at NIPS 2016)*, vol. 8, pp. 21–38, 2016.

[27] J. Gao, Y. Shen, J. Liu, M. Ito, and N. Shiratori, "Adaptive Traffic Signal Control: Deep Reinforcement Learning Algorithm with Experience Replay and Target Network," *arXiv preprint arXiv:1705.02755*, 2017.

[28] S. S. Mousavi, M. Schukat, and E. Howley, "Traffic Light Control Using Deep Policy-gradient and Value-function-based Reinforcement Learning," *IET Intelligent Transport Systems*, vol. 11, p. 417–423, Sept. 2017.

[29] J. Yoon, K. Ahn, J. Park, and H. Yeo, "Transferable Traffic Signal Control: Reinforcement Learning with Graph Centric State Representation," *Transportation Research Part C: Emerging Technologies*, vol. 130, p. 103321, 2021.

[30] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Sept. 2017.

[31] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2018.

[32] S. Pouyanfar, M. Saleem, N. George, and S.-C. Chen, "Roads: Randomization for Obstacle Avoidance and Driving in Simulation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[33] H. Niu, J. Hu, Z. Cui, and Y. Zhang, "Dr2l: Surfacing Corner Cases to Robustify Autonomous Driving via Domain Randomization Reinforcement Learning," in *Proceedings of the 5th International Conference on Computer Science and Application Engineering*, CSAE 2021, ACM, Oct. 2021.

[34] X. Wang, S. Sanner, and B. Abdulhai, "A Critical Review of Traffic Signal Control and A Novel Unified View of Reinforcement Learning and Model Predictive Control Approaches for Adaptive Traffic Signal Control," *arXiv preprint arXiv:2211.14426*, 2022.

[35] H. Wei, G. Zheng, V. Gayah, and Z. Li, "A Survey on Traffic Signal Control Methods," *arXiv preprint arXiv:1904.08117*, 2019.

[36] T. Urbanik, A. Tanaka, B. Lozner, E. Lindstrom, K. Lee, S. Quayle, S. Beaird, S. Tsoi, P. Ryus, D. Gettman, S. Sunkari, K. Balke, and D. Bullock, *Signal Timing Manual.* Transportation Research Board, 2 ed., Sept. 2015.

[37] M. Eom and B.-I. Kim, “The Traffic Signal Control Problem for Intersections: A Review,” *European transport research review*, vol. 12, pp. 1–20, 2020.

[38] R. L. Gordon and W. Tighe, “Traffic Control Systems Handbook,” tech. rep., United States. Federal Highway Administration. Office of Transportation, 2005.

[39] P. Varaiya, “Max Pressure Control of a Network of Signalized Intersections,” *Transportation Research Part C: Emerging Technologies*, vol. 36, pp. 177–195, 2013.

[40] F. V. Webster, “Traffic Signal Settings,” tech. rep., Road Research Laboratory, Department of Scientific and Industrial Research, 1958.

[41] R. P. Roess, E. S. Prassas, and W. R. McShane, *Traffic Engineering.* Pearson/Prentice Hall, 2004.

[42] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[43] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychological Review*, vol. 65, no. 6, p. 386–408, 1958.

[44] K. Fukushima, “Cognitron: A Self-Organizing Multilayered Neural Network,” *Biological Cybernetics*, vol. 20, no. 3–4, p. 121–136, 1975.

[45] K. Fukushima, “Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position,” *Biological Cybernetics*, vol. 36, p. 193–202, Apr. 1980.

[46] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation Applied to Handwritten Zip Code Recognition,” *Neural Computation*, vol. 1, p. 541–551, Dec. 1989.

[47] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning Representations by Back-propagating Errors,” *Nature*, vol. 323, p. 533–536, Oct. 1986.

[48] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural Message Passing for Quantum Chemistry,” in *Proceedings of the 34th International*

Conference on Machine Learning - Volume 70, ICML'17, p. 1263–1272, JMLR.org, 2017.

[49] K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, vol. 2, p. 359–366, Jan. 1989.

[50] G. Cybenko, "Approximation by Superpositions of a Sigmoidal Function," *Mathematics of Control, Signals, and Systems*, vol. 2, p. 303–314, Dec. 1989.

[51] W. L. Hamilton, "Graph Representation Learning," *Synthesis Lectures on Artifical Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159, 2020.

[52] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," *arXiv preprint arXiv:1409.0473*, 2014.

[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[54] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 448–456, PMLR, 07–09 Jul 2015.

[55] X. Glorot and Y. Bengio, "Understanding the Difficulty of Training Deep Feedforward Neural Networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 249–256, PMLR, 13–15 May 2010.

[56] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[57] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact Solutions to the Nonlinear Dynamics of Learning in Deep Linear Neural Networks," *arXiv preprint arXiv:1312.6120*, 2013.

[58] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[59] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[60] A. E. Orhan and X. Pitkow, "Skip Connections Eliminate Singularities," *arXiv preprint arXiv:1701.09175*, 2017.

[61] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[62] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[63] E. Hüllermeier and W. Waegeman, "Aleatoric and Epistemic Uncertainty in Machine Learning: An Introduction to Concepts and Methods," *Machine Learning*, vol. 110, p. 457–506, Mar. 2021.

[64] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 1050–1059, PMLR, 20–22 Jun 2016.

[65] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[66] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* MIT press, 2018.

[67] R. Bellman, "A Markovian Decision Process," *Journal of Mathematics and Mechanics*, pp. 679–684, 1957.

[68] F.-X. Devailly, D. Larocque, and L. Charlin, "IG-RL: Inductive Graph Reinforcement Learning for Massive-Scale Traffic Signal Control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, p. 7496–7507, July 2022.

[69] F.-X. Devailly, D. Larocque, and L. Charlin, "Model-based Graph Reinforcement Learning for Inductive Traffic Signal Control," *arXiv preprint arXiv:2208.00659*, 2022.

[70] H. Zhang, C. Liu, W. Zhang, G. Zheng, and Y. Yu, “Generalight: Improving Environment Generalization of Traffic Signal Control via Meta Reinforcement Rearning,” in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM ’20, ACM, Oct. 2020.

[71] W. Genders and S. Razavi, “Using a Deep Reinforcement Learning Agent for Traffic Signal Control,” *arXiv preprint arXiv:1611.01142*, 2016.

[72] C. J. C. H. Watkins, *Learning from Delayed Rewards.* PhD thesis, University of Cambridge, 1989.

[73] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” *arXiv preprint arXiv:1312.5602*, 2013.

[74] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level Control through Deep Reinforcement Learning,” *Nature*, vol. 518, p. 529–533, Feb. 2015.

[75] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous Control with Deep Reinforcement Learning,” *arXiv preprint arXiv:1509.02971*, 2015.

[76] B. T. Polyak and A. B. Juditsky, “Acceleration of Stochastic Approximation by Averaging,” *SIAM Journal on Control and Optimization*, vol. 30, p. 838–855, July 1992.

[77] H. Van Hasselt, A. Guez, and D. Silver, “Deep Reinforcement Learning with Double Q-Learning,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, Mar. 2016.

[78] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, “Dueling Network Architectures for Deep Reinforcement Learning,” in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 1995–2003, PMLR, 20–22 Jun 2016.

[79] W. R. Thompson, “On the Likelihood that one Unknown Probability Exceeds Another in View of the Evidence of two Samples,” *Biometrika*, vol. 25, p. 285–294, Dec. 1933.

[80] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Advances in Neural Information Processing Systems* (S. Solla, T. Leen, and K. Müller, eds.), vol. 12, MIT Press, 1999.

[81] R. J. Williams, “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning,” *Machine Learning*, vol. 8, p. 229–256, May 1992.

[82] A. G. Barto, R. S. Sutton, and C. W. Anderson, “Neuronlike Adaptive Elements that can Solve Difficult Learning Control Problems,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, p. 834–846, Sept. 1983.

[83] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous Methods for Deep Reinforcement Learning,” in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 1928–1937, PMLR, 20–22 Jun 2016.

[84] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, “Scalable Trust-Region Method for Deep Reinforcement Learning using Kronecker-Factored Approximation,” in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[85] M. L. Littman, “Markov Games as a Framework for Multi-Agent Reinforcement Learning,” in *Machine Learning Proceedings 1994*, p. 157–163, Elsevier, 1994.

[86] L. S. Shapley, “Stochastic Games,” *Proceedings of the National Academy of Sciences*, vol. 39, p. 1095–1100, Oct. 1953.

[87] K. Zhang, Z. Yang, and T. Başar, “Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms,” *Handbook of Reinforcement Learning and Control*, pp. 321–384, 2021.

[88] C. Claus and C. Boutilier, “The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems,” in *Proceedings of the Fifteenth National/Tenth*

Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, AAAI ’98/IAAI ’98, (USA), p. 746–752, American Association for Artificial Intelligence, 1998.

[89] G. Papoudakis, F. Christianos, A. Rahman, and S. V. Albrecht, “Dealing with Non-Stationarity in Multi-Agent Deep Reinforcement Learning,” *arXiv preprint arXiv:1906.04737*, 2019.

[90] J. D. Little, M. D. Kelson, and N. H. Gartner, “MAXBAND: A Versatile Program for Setting Signals on Arteries and Triangular Networks,” *Transportation Research Record Journal of the Transportation Research Board*, vol. 795, pp. 40–46, 12 1981.

[91] P. Lowrie, *SCATS, Sydney Coordinated Adaptive Traffic System: A Traffic Responsive Method of Controlling Urban Traffic.* Roads and Traffic Authority NSW, Darlinghurst, NSW Australia, 1990.

[92] P. Hunt, D. Robertson, R. Bretherton, and R. Winton, “SCOOT - A Traffic Responsive Method of Coordinating Signals,” tech. rep., Transport and Road Research Laboratory (TRRL), 1981.

[93] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum Learning,” in *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML ’09, ACM, June 2009.

[94] Y. Xiong, G. Zheng, K. Xu, and Z. Li, “Learning Traffic Signal Control from Demonstrations,” in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM ’19, ACM, Nov. 2019.

[95] C. Finn, P. Abbeel, and S. Levine, “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks,” in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, pp. 1126–1135, PMLR, 06–11 Aug 2017.

[96] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein Generative Adversarial Networks,” in *Proceedings of the 34th International Conference on Machine Learning*, ICML’17, p. 214–223, JMLR.org, 2017.

[97] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph Attention Networks,” *arXiv preprint arXiv:1710.10903*, 2017.

[98] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic Traffic Simulation using SUMO," in *The 21st IEEE International Conference on Intelligent Transportation Systems*, IEEE, 2018.

[99] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[100] G. Van Rossum and F. L. Drake Jr, *Python Reference Manual.* Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[101] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[102] M. Fey and J. E. Lenssen, "Fast Graph Representation Learning with PyTorch Geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[103] S. Krauß, *Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics.* PhD thesis, Universität zu Köln, 1998.

[104] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array Programming with NumPy," *Nature*, vol. 585, p. 357–362, 2020.

[105] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[106] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, no. 3, p. 90–95, 2007.

[107] FGSV, *Handbuch für die Bemessung von Straßenverkehrsanlagen: HBS*. FGSV.: Forschungsgesellschaft für Straßen- und Verkehrswesen, FGSV-Verlag, 2015.

Declaration of Academic Integrity

Thesis:	**Domain Randomization of Deep Reinforcement Learning Environments for Zero-Shot Traffic Signal Control**
Surname:	Dreyer
First name:	Frank
Date of Birth:	16.10.1994
Matriculation No.:	227030

I herewith assure that I wrote the present thesis independently, that the thesis has not been partially or fully submitted as graded academic work and that I have used no other means than the ones indicated. I have indicated all parts of the work in which sources are used according to their wording or to their meaning.

I am aware of the fact that violations of copyright can lead to injunctive relief and claims for damages of the author as well as a penalty by the law enforcement agency.

Date: ....

(Signature)