New Applications of Data Science for Intelligent Transportation Systems

by

ALVARO CABREJAS EGEA

Thesis

Submitted to the University of Warwick

for the degree of

Doctor of Philosophy

Mathematics for Real-World Systems Centre for Doctoral Training

November 2020

Contents

List of Tables

List of Figures

Acknowledgments

This thesis would not have been possible without the immense help and guidance provided my by supervisor, Dr. Colm Connaughton. He has been the most encouraging and supporting supervisor one could wish for, demonstrating the highest scientific curiosity and human kindness while always being there to bounce ideas about any topic. My research, the time spent at the University of Warwick and my scientific outlook have benefited in major ways from his counsel and advice. I would also like to thank the staff and the members of the community at the Centre for Complexity Science and the Mathematics for Real-World Systems Centre for Doctoral Training, who have managed to create an outstandingly welcoming place for research and study, full of brilliant minds and an amazing place to call home for the past few years. Very special thanks to Maxim Smilovitysky, Ayman Boustati, Roger Hill, Raquel Gonçalves and Chris Davies for the walks, chats and general frolicking.

Thanks to The Alan Turing Institute for their support, and very special thanks to Jules Manser, Jade Thompson, Jessie Wand, Dr. Marya Bazzi, Dr. Sebastian Vollmer, Dr. Bilal Mateen, Dr. Franz Kiraly and Dr. Neil Walton.

Also, big thanks to Ignacio Gómez Pérez and Iñigo Aguirre de Carcer García from the Universidad Politécnica de Madrid, since they shaped the academic experience that drove me towards science and eventually here, spurring my curiosity and showing me that research and play can indeed overlap.

Thanks to my old friends for their understanding and support, and although I cannot possibly name every one here, special thanks to Luis Sánchez Gilabert, Enrique Golmayo Bolíbar and Gonzalo Campos García-Herraiz for always being there. A very special thanks to Jose J. Fernández Cano for teaching me the value of being part of a real team, but also that leading is not always standing in the front, and that no matter how long it's been, sharing is still winning.

Thanks to my mother Carmen for showing me that there are no wrong questions, ever; to my father Rafael for teaching me to be hard on my opinions and always examine the facts; and to my brother Mario for all the times he's been there.

Lastly, I am out of words to express my gratitude to Chihiro Ikegami for constantly being my greatest supporter and cheerer, and for always going above and beyond. Thank you for choosing to walk this road with me.

Declarations

This thesis is submitted to the University of Warwick in support of my application for the degree of Doctor of Philosophy. It has been composed by myself unless stated otherwise, and has not been submitted in any previous application for any degree. Parts of this thesis have been published by the author or created within a collaborative project as stated below:

- Parts of Chapter 3 were written during a collaboration with Thales UK, and have been published in [1] Alvaro Cabrejas Egea, Peter De Ford, and Colm Connaughton. Estimating Baseline Travel Times for the UK Strategic Road Network. In *IEEE 21st International Conference on Intelligent Transportation Systems, Proceedings, ITSC*, volume 2018-Novem, pages 531–536. IEEE, 2018. ISBN 9781728103235. doi: 10.1109/ITSC.2018.8569924.

- Parts of Chapter 4 were written during a collaboration with Thales UK, and have been published as [2] Alvaro Cabrejas Egea and Colm Connaughton. Wavelet Augmented Regression Profiling (WARP): improved long-term estimation of travel time series with recurrent congestion. In *IEEE 23rd International Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2020. doi: 10.1109/itsc45102.2020.9294318.
  The corresponding pre-print [3] is available as Alvaro Cabrejas Egea and Colm Connaughton. Wavelet Augmented Regression Profiling (WARP): improved long-term estimation of travel time series with recurrent congestion. *arXiv preprint arXiv:2006.13072*, 2020.

- Parts of Chapter 5 were written during an Enrichment Year in the Alan Turing Institute (London, UK), some of these counted with the collaboration of Raymon Zhang and/or Neil Walton. Some sections of this Chapter have been accepted for publication as [4] Alvaro Cabrejas Egea, Raymond Zhang, and Neil Walton. Reinforcement Learning for Traffic Signal Control: Comparison with Commercial Systems. In *14th Conference on Transport Engineering*, 2021. doi: pending.

The corresponding pre-print [5] is available as Alvaro Cabrejas-Egea, Raymond Zhang, and Neil Walton. Reinforcement Learning for Traffic Signal Control: Comparison with Commercial Systems. *arXiv preprint arXiv: 2104.10455*, 2021.

My contributions include development of the testing environment, architecture and implementation of baseline Reinforcement Learning agents, development of all the components of the Value-Based Reinforcement Learning agents, definition and implementation of States, Rewards and Actions, development of the test maps and their calibration, development of the training schedule and hyperparameter tuning, and development of the of the modular training environment, cyclic controllers and general deployment utilities.

- Parts of Chapter 6 were written during an industrial collaboration with Vivacity Labs, and have been published as [6] Alvaro Cabrejas Egea, Shaun Howell, Maksis Knutins, and Colm Connaughton. Assessment of Reward Functions for Reinforcement Learning Traffic Signal Control under Real-World Limitations. In *IEEE International Conference on Systems, Man and Cybernetics, Proceedings, SMC*. doi: 10.1109/smc42975.2020.9283498.

  The corresponding pre-print [7] is available as Alvaro Cabrejas-Egea, Shaun Howell, Maksis Knutins, and Colm Connaughton. Assessment of Reward Functions for Reinforcement Learning Traffic Signal Control under Real-World Limitations. *arXiv preprint arXiv:2008.11634*, 2020.

  My contributions include the definition, development and testing of the individual rewards, hyperparameter tuning, and modification of the existing testing environment codebase to admit the tested quantities.

- Parts of Chapter 7 were written during an industrial collaboration with Vivacity Labs, and are currently under review as [8] Alvaro Cabrejas Egea and Colm Connaughton. Reward Functions for Real-World Pedestrian and Vehicular Intersection Control through Reinforcement Learning. In *IEEE 24th International Conference on Intelligent Transportation Systems, ITSC proceedings*. IEEE, 2021. doi: pending.

The corresponding pre-print [9] is available as Alvaro Cabrejas-Egea and Colm Connaughton. Assessment of Reward Functions in Reinforcement Learning for Multi-Modal Urban Traffic Control under Real-World limitations. *arXiv preprint arXiv:2010.08819*, 2020.

My contributions include the definition, development and testing of the individual rewards, hyperparameter tuning, and modification of the existing testing environment codebase to admit the tested quantities.

Abstract

Streets and motorways are the basic blocks in the core of our transportation networks. In recent years, increases in available sensory and computing power have allowed us to start massive gatherings of data related to their use and performance, and to obtain insightful information via data science. This, in turn, has increased our ability to create systems that estimate the state of the transportation networks and provide us with control capabilities over it, giving rise to the concept of Intelligent Transportation Systems. These systems aim to provide deeper levels of observability to our transportation networks so that their capacity can be increased without the need of further heavy investment to develop traffic infrastructure, especially in terms of laying new roads and streets.

In this thesis we aim to contribute to this process in both urban and interurban settings. Here we propose two different algorithms to estimate and forecast expected travel time in motorways over the long term, ranging from hours to a week. The first of them is centred around the identification of the different traffic regimes and leveraging their specific characteristics to improve estimation and forecasting. The second of them looks further into the differentiation between recurrent and non recurrent congestion from the point of view of statistical analysis in the frequency space, using the natural frequencies of the traffic system to tell them apart and exert prediction. We also delve into how Intelligent Transportation Systems can affect our cities, looking at how reinforcement learning can create independent agents capable of controlling traffic lights at intersections. We do this by first looking at the most effective agent architectures in different junctions of increasing complexity. Then we dive into the difference in performance for agents in charge of vehicular intersections, provided by an array of reward functions that use different measures obtained from the traffic flow. Finally, we expand these systems to also take pedestrians into account, investigating the rewards that produce the lowest waiting times when serving different modes of transportation with opposing needs.

Abbreviations

AC: Actor Critic

A2C: Adaptive Actor Critic

A3C: Asynchronous Advantage Actor Critic

ADAM: ADAptive Moment estimation

AI: Artificial Intelligence

ANN: Artificial Neural Network

API: Application Programming Interface

ARIMA: AutoRegressive Integrated Moving Average

CNN: Convolutional Neural Network

COM: Component Object Model

CPU: Central Processing Unit

CWT: Continuous Wavelet Transform

DQN: Deep Q-Network

DDQN: Double Deep Q-Network

D3QN: Double Dueling Deep Q-Network

DRL: Deep Reinforcement Learning

DWT: Discrete Wavelet Transform

EU: European Union

EWMA: Exponentially Weighted Moving Average

FFT: Fast Fourier Transform

GPS: Global Positioning System

GPU: Graphics Processing Unit

HE: Highways England

IQR: Inter-Quantile Range

ITS: Intelligent Transportation Systems

LWR: Locally Weighted Regression

LOESS: LOcally wEighted Scaterplot Smoothing

MAPE: Mean Absolute Percentage Error

MARE: Mean Absolute Relative Error

MARL: Multi-Agent Reinforcement Learning

MC: Monte Carlo

MCTS: Monte Carlo Tree Search

MDP: Markov Decision Process

MO: Maximum Occupancy

NN: Neural Network

NTIS: National Traffic Information Service

PER: Prioritised Experience Replay

PG: Policy Gradient

PLN: Phase-Length Normalisation

RL: Reinforcement Learning

SARSA: State-Action-Reward-State-Action

SRN: Strategic Road Network

STL: Seasonal Trend decomposition using LOESS

RMSE: Root Mean Squared Error

TD: Temporal Difference

TSC: Traffic Signal Control

UTC: Urban Traffic Controller

UK: United Kingdom

VA: Vehicle Actuated System-D

WARP: Wavelet Augmented Regression Profiling

WT: Wavelet Transform

CHAPTER 1

Introduction

The world in which we live is becoming progressively more interconnected. One driving force in this process is our ever increasing ability to gather data from the events happening around us, and leveraging it to obtain deeper insights about them via Data Science. This tendency is having an enormous impact on the way that we interact with different systems around us. Especially as they transition from their traditional form to new "smart" approaches which attempt to improve the efficiency and uses of said systems. We can see examples of this on the recent evolution of applications aimed at sectors such as utility grids, medicine, online retail, smart assistants and Intelligent Transportation Systems.

Intelligent Transportation Systems (ITS), which are at the core of the research here presented, were defined by the European Union as "systems in which information and communication technologies are applied in the field of road transport, including infrastructure, vehicles and users, and in traffic management and mobility management, as well as for interfaces with other modes of transport" [10]. Effectively, ITS allow us to make better use from our current transportation infrastructure in two main ways: on one hand, it is possible to divert some of our increasing sensing capabilities towards our road networks to better understand their internal workings in an attempt to estimate their typical behaviour and use this to generate predictions, and on the other hand, we can direct some of the newly available computing power to actively manage these systems in real time, achieving increases in capacity that would not be possible with traditional transportation systems. Here, both approaches will be used in conjunction with Data Science for different problems, looking to generate better estimates of travel times for inter-urban networks, and also at how can urban networks be actively managed in real-time.

Improvements on the reliable estimation and forecasting of travel times in inter-urban road networks allow road users to forward plan journeys to minimise travel time, potentially increasing overall system efficiency. On busy motorways, however, congestion events can cause large, short-term spikes in travel time. These spikes make direct forecasting of travel time using standard time series models difficult on the timescales of hours to days that are relevant to forward planning. The problem is that some such spikes are caused by unpredictable incidents and should be filtered out, whereas others are caused by recurrent peaks in demand and should be factored into estimates.

Improving the operation of urban road networks can significantly reduce traffic congestion, as well as promote active travel and public transport within a city and reduce emissions. Optimising the timings of adaptive traffic signals can play a large role in achieving this by making effective use of green lights in response to the demand levels on each approach to a junction and by promoting progression across multiple junctions.

However, most currently used algorithms have been improved only incrementally since their initial development in the 1980s, and so do not take advantage of either modern traffic data sources, nor modern computational resources. Recent improvements in CPU and especially GPU power are allowing for vision-based sensors to gather large amounts of real-time data that a few years ago seemed unattainable, such as individual vehicle position and speeds, at a much lower marginal cost than would be feasible with traditional actuated sensors. As a side effect of these developments the area covered by sensors is ever increasing. It is also becoming possible to direct some of these towards pedestrians, which are now also starting to be actively factored in these systems with similar importance to vehicles. This process is allowing the development of novel smart control approaches, harnessing the power of real-time data to deliver cheap and responsive systems that can adapt to a variety of situations.

In this thesis, we study different ways in which Data Science can be applied to create or improve realistic Intelligent Transportation Systems in motorways and urban areas, aiming at increasing our estimation, forecasting and control of road traffic based systems.

In Chapter 2 we introduce the context and basic mathematical concepts that will be required for the development of the rest of the thesis.

In Chapter 3, we present a new method for long-term estimation of the expected travel time for links on highways and their variation with time. The approach is based on a time series analysis of travel time data from the UK's National Traffic Information Service (NTIS). Time series of travel times are characterised by a noisy background variation exhibiting the expected daily and weekly patterns punctuated by large spikes associated with congestion events. Some spikes are caused by peak hour congestion and some are caused by unforeseen events like accidents. We present an algorithm that uses thresholding to split the data into background and spike signals, each of which is analysed separately. The the background signal is extracted using spectral filtering. The periodic part of the spike signal is extracted using locally weighted regression (LWR). The final estimated travel time is obtained by recombining these two. We assess our method by cross-validating in several UK motorways. We use 8 weeks of training data and calculate the error of the resulting travel time estimates for a week of test data, repeating this process 4 times. We find that the error is significantly reduced compared to estimates obtained by simple segmentation of the data and compared to the estimates published by the NTIS system.

Chapter 4 presents a deeper study of the troublesome spikes uncovered in Chapter 3. Here we introduce the Wavelet Augmented Regression Profiling (WARP) method for long-term estimation of typical travel times. We look at how Wavelets can be used to analyse a travel time signal, looking at its statistical properties in the frequency-time space, finding which data points are outliers with respect to the base dynamics of the road section and classifying them into background and spikes in a more sophisticated manner than it was done in the previous chapter. It then further separates the spikes into contributions from recurrent and residual congestion. The linear separation of the components is achieved using a combination of wavelet transforms, spectral filtering and locally weighted regression. The background and recurrent congestion contributions are then used to estimate typical travel times with horizon of one week in an accurate and computationally inexpensive manner. We train and test WARP on the M6 and M11 motorways in the United Kingdom using 12 weeks of link level travel time data obtained from the UK's National Traffic Information Service (NTIS). When evaluating the algorithm via rolling forecast evaluations, WARP compares favourably to estimates produced by a simple segmentation method and to the currently published estimates.

In Chapter 5, a number of modern deep reinforcement learning architectures are considered for the task of traffic light control in urban junctions, and compared against commercial systems when using a traffic profile based on realistic demand scenarios. The focus is primarily placed on two Deep Reinforcement

Learning classes: Deep Q-Networks (DQN) and Synchronous Actor-Critic (A2C) methods. DQN focuses on estimating the optimal value function from the present state. A2C estimates the optimal policy and maintains a running estimate of the optimal value from the present state. In both cases a neural network is used to guide the algorithm towards optimal decisions. Modern implementations and extensions of these systems are used to this end: in the case of DQN, Double Deep Q-Learning, Dueling architectures and Prioritized Experience Replay are used; for Actor-Critic methods, a entropy penalty function is applied to the policy search. Each of these additional methods improves training time and the quality of the solutions. These learning algorithms are implemented on a custom RL platform built to interface with the PTV Vissim simulator. To this end, software has been designed on top of the PTV Vissim COM (Component Object Model) API (Application Programming Interface), so that the simulations and input data behave in a manner that is similar to a multi-agent implementation of OpenAI Gym interface [11]. This assists in simplifying and rationalizing the training loop for our algorithms. All the source code is publicly accessible via a GitHub repository [12]. In Chapter 5, the performance of these algorithms will be favourably compared against that obtained with the MOVA, Surtrac and Balance commercial systems. The work there presented took place in the context of a project in smart mobility between The Alan Turing Institute and the Toyota Mobility Foundation.

In Chapter 6, different reward functions for Reinforcement Learning agents operating Urban Traffic Controllers are compared in a simulation of a junction in Greater Manchester, UK, across various demand profiles, subject to real world constraints: realistic sensor inputs, controllers, calibrated demand, intergreen times and stage sequencing. Several authors have surveyed the reward functions used in the literature of Reinforcement Learning for Traffic Signal Control, but attributing outcome differences to reward function choice across works is problematic as there are many uncontrolled differences, as well as different outcome metrics. The reward metrics considered are based on the time spent stopped, lost time, change in lost time, average speed, queue length, junction throughput and variations of these magnitudes. The performance of these reward functions is compared in terms of total waiting time. It is found that speed maximisation resulted in the lowest average waiting times across all demand levels, displaying significantly better performance than other rewards previously introduced in the literature. The work developed in this and the following chapter took place in the context of an industrial collaboration with Vivacity Labs, after which several of the agents presented there have been deployed to the real-world junction to control real traffic.

Chapter 7 extends the work of Chapter 6, shifting the focus to multi-objective optimisation with the introduction of simulated pedestrian demand, in addition to the vehicular one. Hence, the requirements of both classes, which are in opposition, must be balanced by the agent to minimise global delay. The chapter presents a robust comparison between 30 different Reinforcement Learning reward functions for controlling intersections serving vehicles and pedestrians. A calibrated model in terms of demand, sensors, green times and other operational constraints of a real intersection in Greater Manchester is used. Sensor inputs are restricted to what can be achieved with current vision-based sensors. The rewards can be broadly classified in 5 groups depending on the quantities used: queues, waiting time, delay, average speed and throughput in the junction. The performance of different agents, in terms of waiting time, is compared across different demand levels ranging from normal operation to saturation of traditional adaptive controllers. We find that those rewards maximising the speed of the network obtain the lowest waiting time for vehicles and pedestrians simultaneously, closely followed by queue minimisation, demonstrating better performance than other methods proposed in the literature.

Finally, in Chapter 8 the main contributions of this thesis are summarised and reviewed. Further, potential new avenues in which these contributions could be extended and continued are explored.

CHAPTER 2

Background

2.1 Time Travel Estimation and Forecasting

This section will introduce the most important ideas and the mathematical background that will constitute the framework from which different approaches to estimation and forecasting of travel times in road traffic will be developed in following chapters.

Within road traffic, the focus will be placed in *travel times*, understood as the time needed to transverse a given section of road: how they vary in scales ranging from minutes to weeks and how their baseline variation and behaviour can be estimated, and this estimation, in turn, used for forecasting.

The main challenges encountered revolve about the noisy and bursty dynamics displayed in these time series. The majority of sites evaluated display both recurring and non-recurrent congestion, affecting the higher quantiles of their travel times distribution. Distinction between these two classes is not trivial and has a high impact on the general accuracy of estimation and forecasting methods. In order to better classify time series, estimate their baseline values, and forecast their future behaviour, techniques introduced in this chapter that are traditionally used for different purposes will later be combined in novel ways.

The first half of the chapter will proceed as follows: Section 2.1.1 will introduce the road network object of the study. Section 2.1.2 will expand on travel times, their calculation and basic characteristics. Section 2.1.3 will provide a brief history of travel time estimation in science. Section 2.1.4 will introduce basic mathematical background relating to the tools that will be used in later time series analysis, namely decision trees, seasonal estimation, spectral filtering and the wavelet transform. Lastly, and not in this section, problem-specific literature that directly relates to the time-series problems at hand are introduced in Sections 3.2 and 4.3, to better highlight the place of the experiments here conducted amongst the wider literature.

The progression in the second half will be: Section 2.2.1 will introduce current Traffic Signal Control systems. Section 2.2.2 will introduce how these control systems are simulated. Section 2.2.4 will provide a short historical introduction to approaches taken in Urban Traffic Control, Reinforcement Learning and how the latter became a feasible approach to tackle the former. Section 2.2.5 will introduce the necessary mathematical concepts related to Markov Decision Processes, Reinforcement Learning and modern agents. Finally, and again not in this section, the literature that contextualises the approaches here taken in terms of Reinforcement Learning will be found in Sections 5.2, 6.2 and 7.2.

2.1.1 The Strategic Road Network

The Strategic Road Network (SRN) in England covers about 7080 km (4400 miles) of highways and A roads, carrying over 30% of all traffic in the country, with an average daily use of 4 million vehicles [13]. Highways England (HE), who manages the SRN estimates that the traffic on motorways has grown by 50% since 1993, and is forecasted to grow another 31% by 2041.

In this context, efficient operation of motorways is one key item that allows for increased motorway capacity in absence of momentum to extend the network due to economic, environmental and social reasons. The various subsystems composing the Highways England Traffic Management Systems are connected to the National Traffic Information System (NTIS), a subscription service providing access to incidents, Variable Message Signs, Matrix Signal settings and induction loop data, which provide real-time state information about the traffic flow in the network [14].

The most basic component of the SRN are the *links*, segments of road between 500 and 20000 metres in length. For the past 30 years, multitude of induction loops, the most common sensor used worldwide for traffic control, have been rolled out locations in those links managed by HE in order to gather real-time data of the traffic flow and provide with elevated estimation, forecasting and control abilities to the operators. Additional information about their configuration and set-up can be found in technical reports by The Highways Agency [15]. These induction loops operate by detecting disturbances to the electromagnetic field of a coil that is built into the motorway. Their main drawback is the disruption to traffic during their installation and their loss of performance during stop-and-go situations [16]. Individual induction loops are able to detect the presence of vehicles passing above them, being able to provide car counts from which flows can be extrapolated. Multiple loops in a single site allow for the estimation of the speed of vehicles and overheads. Multiple sites in a single road allow for the estimation of travel times in said road.

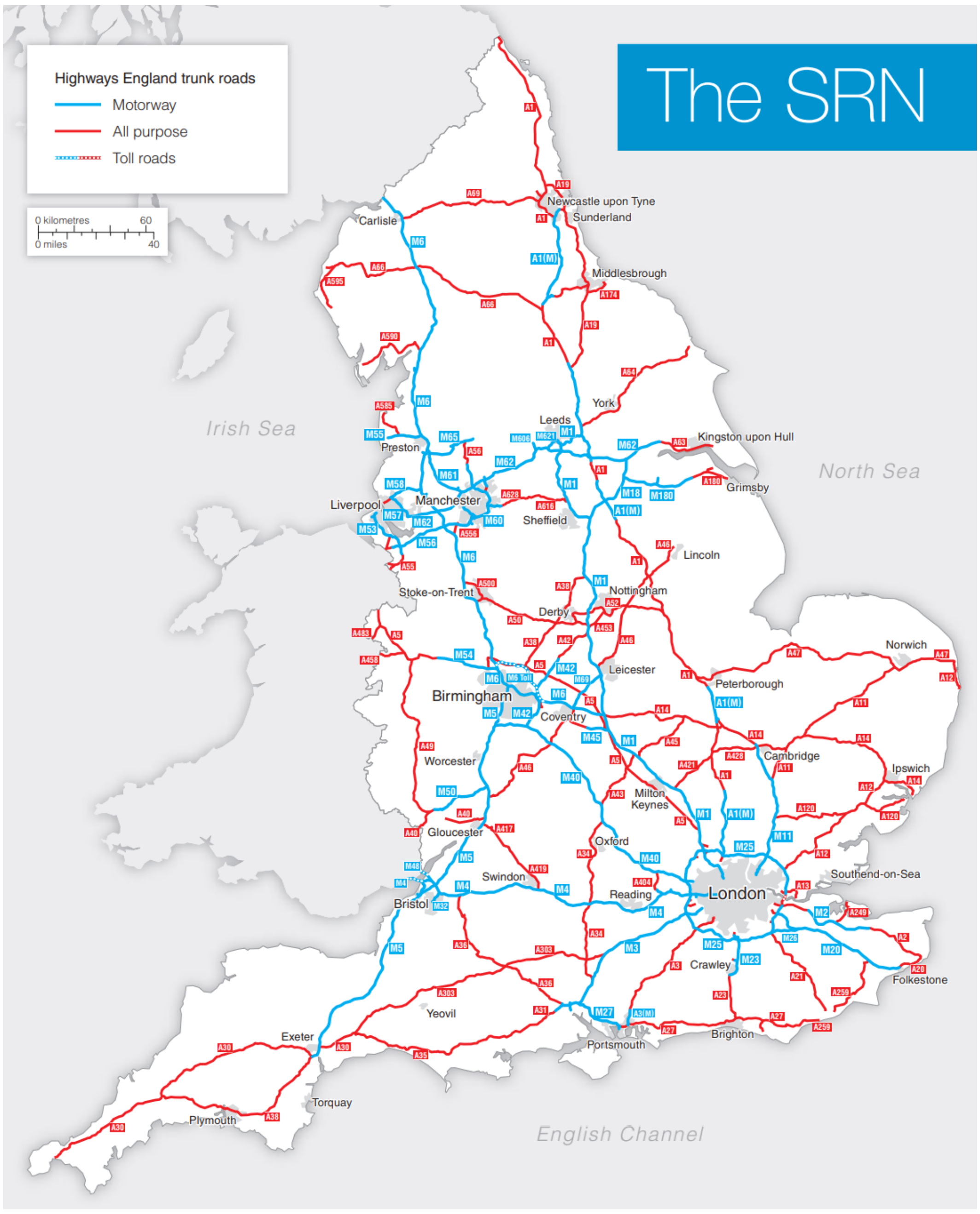


Figure 2.1: Schematic map of the Strategic Road Network in England. Source: Highways England [13] (edited).

2.1.2 Travel Time Profiles

These travel time estimates for a section of road are defined as the expected amount of time needed to travel through said section of a road and are currently obtained by averaging the values of the past 5 minutes on a rolling basis. For each different link, a *travel time profile* can be defined, understood as the typical expected time to transverse said link *at a given point in time*. The main difference between these two is that while *travel time* refers to a single data point for a specific site at a specific time, a *travel time profile* will provide a series of such points over a period of time.

Traffic profiles for each link in the SRN are published to subscribers by NTIS via the DATEX II service together with other information regarding speed, flow, Variable Message Signs, incidents and other data with a minutely resolution [14].

These profiles act as the basic departure point from which all other components of *smart management* build on, by providing baseline measures against which deviations can be identified by using the sensors of the network to, in turn, build a variety of metrics and performance indications. Furthermore, accurate estimates of travel times profiles are extremely useful to both users and operators, since they allow for extended planning of departure, arrival and travel conditions, without which the just-in-time supply chains cannot operate. Hence, accurate estimation and forecasting of expected travel times becomes necessity for advanced transportation and manufacturing systems.

Direct forecasting of travel times is quirky since it is more usual to use speed or flows as principal data source from which travel times are derived. Accurate travel time modelling is important due to being one of the easiest measures to instinctively understand, but also due to their ease to explode into extreme values, quickly and heavily impacting the state of traffic and all associated magnitudes.

An initial challenge to generate these profiles is that the baseline travel times will not be constant over time, depending heavily on a variety of factors affecting both demand and capacity that can make them vary to several times their original value.

A further, and more complex challenge revolves about the previously mentioned bursty nature of travel time series. Their value can stay stable for long periods of time with quick excursions to extreme levels due to congestion. This generates issues with the detection, since it becomes difficult to identify what part of the congestion is recurring and what parts are non-recurring congestion caused by specific unusual non-recurring situations on the road. Both of these situations generate data points that can be found in the highest quantiles of the distribution of travel times for a given location. Previously, approaches that involved removing or ignoring these points have been taken due to the difficulties in the classification between these two components that display such a similar behaviour. However, achieving higher accuracy in the prediction of those travel times in the higher quantiles of the distribution can provide immediate benefits with respect to current estimation and forecasting methods, since those are the sections in which more people are affected by delays.

In terms of scientific approaches, seasonal methods tend to obtain good performance on series following recurrent patterns of human activity (daily and weekly oscillations), however they typically do not generate good results for prediction in high quantiles or during large deviations. As mentioned earlier, these are precisely the areas in which the most benefits of improved estimation can be delivered. Seasonal methods and those based on averaging and smoothing methods are not useful by themselves, as they will capture the non-recurrent component as if it were recurring and predict false traffic jams at times where none will be found. This effect will be further explained in Sections 3.2 and 4.3. Some degree of smoothing, will however be needed to dampen the high frequency and low amplitude oscillations that can be observed in these series. Within the context of distinguishing between recurrent and non-recurrent congestion, the use of decision trees will be useful to classify different traffic regimes, which also can be done via the wavelet transform, as it will be shown in upcoming sections.

In terms of quality quantification, there is no clear answer to the question of what is an optimal traffic profile. For the rest of this document an optimal profile will be defined as the profile that minimises a measure of error when compared to the actual travel times that the link experienced. Further information about current use and calculation of traffic profiles in England can be found in Sections 3.1 and 4.1.1.

2.1.3 Brief history of travel time estimation and forecasting

Prior to the introduction of sensors in the roads, the main methods for estimating travel times were restricted to manual car counting at designated points, floating car approaches in which a car drives with the traffic flow with the specific goal of data collection, and later, inversely working from GPS data to estimate the state of traffic. However all of these methods are suboptimal when compare with direct data gathering by sensors on the road.

Historically, travel times estimated via induction loops in real networks have followed a variety of approaches. Some of these use traffic theory-based models, such as conservation laws [17] which, while providing with realistic theoretical relations, require close attention to entry and exit ramps since they low data quality has a great impact for these approaches. Trajectory-based methods have gathered relevance due to their simplicity and practicality. They require several sensors such that individual vehicles' trajectories can be retraced [18, 19]. An alternative to these are the vehicle reidentification methods. These methods rely on vehicle-specific signatures that are matched across different sites in order to identify said vehicle in different locations. They can rely on vehicle length [20] or inductance patter recognition [21].

In this thesis, the focus will be placed on data-based methods, since they do not require expertise in traffic theory, while providing very good results [22]. Their main drawback is related to the high amounts of data required and the dependence of the accuracy of the results on the granularity and precision of the data source. However, a sufficiently large and dense dataset can heavily mitigate this effect.

Early methods of this kind [23] looked at seasonal approaches such as AutoRegressive Integrated Moving Average (ARIMA) and Box-Jenkins methods [24], although Seasonal-Trend estimation based on LOESS (explained in following sections) is currently considered a more powerful alternative [25] and will be consequently used for seasonal analysis.

For a deeper look into historical estimation of travel times, please check You and Kim [26] and Mori et al. [22].

2.1.4 Mathematical Background

The main mathematical tools used while conducting the time series research presented in this thesis are introduced over the next sections.

2.1.4.1 Decision Trees

Decision trees are decision support tools that use a tree-like model of a sequence of decisions, based on an if-else approach. In their most basic setting, they allow classification of data into classes. They have been traditionally used in operations research to support decision analysis in obtaining strategies that are most likely to obtain a determined goal. Their use within the research here presented will be to classify the state of the series between different regimes (congested or not) in order to guide the combinations between different components of the signal.

A decision tree acts as an n-nary tree flowchart where each node splits into a number of possibilities according to the value of some feature. Successive nodes are traversed until a leaf (terminal node) is reached, which will provide the category of the data that is being analysed.

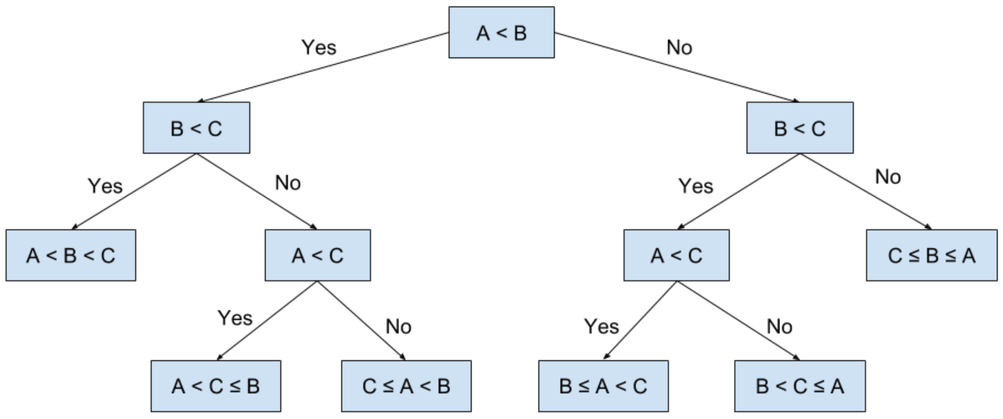


Figure 2.2: Example of a simple decision tree to sort 3 values A, B, C. Source: Niculaescu [27]

Their strengths are that they are very easy to understand and can be easily converted into a set of rules. They are versatile, managing different kinds of data, are non-parametric (no assumptions about internal structure) and can handle missing values. Their weaknesses are more relevant to traffic since the rules that determine their operation need to be set by hand, and these would need to change with the location and specific characteristics of a specific site. Hence, the use of decision trees alone for profile estimation and forecasting for massive-scale deployments is not feasible on its own, but can still be a useful component of other systems if the rules governing the classification can be formulated in general terms as it will be shown in the following chapters.

2.1.4.2 Seasonal Estimation Methods and STL

One of the most intuitive and straightforward correlations with road traffic demand are the patterns of human activity, as it will be shown in later sections of this thesis. This correlation, that traduces into highly structured autocorrelation functions for the time series of individual links' travel times, indicates an underlying seasonality. Hence, in this section different methods of calculating seasonality and their applications to traffic estimation and forecasting will be reviewed.

Seasonal estimation methods are based on the idea that a given signal can be decomposed into a number of component series that can be, additively or multiplicatively, be reconstructed into the original series. Their use within the research here presented is to approximate the difference seasonal patterns that we can observe in the data and their evolution over time.

Time series can be typically decomposed into:

- A trend component T, indicating the long term progression of the series.
- A seasonal component S, representing the seasonal or periodic variation in the series.
- A noise or residual component I, which comprises all the information that cannot be explained in terms of trend or seasonality.

Hence, any point in the series x_i can be understood as

$$x_i = T_t + S_t + I_t, \tag{2.1}$$

in the case of an additive decomposition, or as

$$x_i = T_t \times S_t \times I_t \tag{2.2}$$

if the decomposition is multiplicative.

Nonparametric regression is a category of regression in which the predictor variable does not have a priori form, but is instead constructed according to information stemming from the data. In general, it requires greater amounts of data than parametric regression models since the model structure is also generated from the data, instead of only generating model estimates.

Seasonal-Trend decomposition based on LOESS (LOcally Estimated Scatterplot Smoothing), also known as STL was first proposed by Cleveland et al. [28]. STL aims to decompose a time series signal into Trend, Seasonal and Noise components. Unlike previous seasonal estimation methods such as ARIMA and TBATS, STL can assume seasonalities of any period, or combination of periods.

The STL algorithm is composed of the LOESS smoother, and two recursive procedures codified in an inner loop and an outer loop, and depends on the following internal parameters: observations per cycle of the seasonal component $n_{(p)}$, number of passes of the inner loop $n_{(i)}$, number of passes of the outer loop $n_{(o)}$, smoothing parameter for low-pass filter $n_{(l)}$, smoothing parameter for trend component $n_{(t)}$ and the smoothing parameter for the seasonal component $n_{(s)}$.

LOESS smoother: Assuming x_i, y_i; $i = 1, ..., n$ are measurements of an independent and dependent variables, the LOESS regression curve $\hat{g}(x)$ is a smoothing of y given x. To calculate $\hat{g}(x)$, an integer q assumed $q \leq n$ is used to choose the q nearest neighbours of x, which are given weights based on their distance from x. With $\lambda_q(x)$ being the distance of the q-th farthest x_i from x, and W being the weight function:

$$W(u) = \begin{cases} (1 - u^3)^3 & \text{for } 0 \leq u < 1 \\ 0 & \text{for } u \geq 1 \end{cases} \tag{2.3}$$

The neighbouring weights are:

$$v_i(x) = W\left(\frac{|x_i - x|}{\lambda_q(x)}\right), \tag{2.4}$$

meaning those x_i closest to x will have the highest weights, becoming zero at the location of the $q - th$ farthest point. Then, a polynomial of degree $d = 1$ or 2 is fit to the data, obtaining $\hat{g}(x)$.

Now, assuming $q > n$, let $\lambda_n(x)$ be the distance from x to the farthest x_i, we can define $\lambda_q(x)$ as:

$$\lambda_q(x) = \lambda_n(x)\frac{q}{n}, \tag{2.5}$$

which will be used as before in the definition of neighbouring weights.

Assuming each measurement (x_i, y_i) has a reliability weight ρ_i, this can be incorporated into the LOESS smoothing by using $\rho_i v_i(x)$ as the weights in the local least-squares fitting.

Inner loop: In each pass through the inner loop, the seasonal and trend components are updated once. This is repeated $n_{(i)}$ times. Let S_i^k and T_i^k, $i = 1, ...n$ be the seasonal and trend components at the end of the k-th pass. The steps to follow for the next update are:

1. Detrending: detrended series $x_i - T_i^k$.

2. Cycle-subseries smoothing: Each subseries cycle of the detrended series is smoothed by LOESS with $q = n_{(s)}$ and $d = 1$, obtaining a temporary series C_i^{k+1}.

3. Low-pass filter: The smoothed cycle-subseries are smoothed using 2 successive moving averages of length $n_{(p)}$, then a moving average of length 3 and finally a LOESS smoothing with $d = 1$ and $q = n_{(l)}$, obtaining an output L_i^{k+1}.

4. Detrending of smoothed cycle-subseries: The seasonal component is updated $S_i^{k+1} = C_i^{k+1} - L_i^{k+1}$. The role of L_i^{k+1} is preventing low-frequency power from entering the seasonal component.

5. Deseasonalising: A deseasonalised series $x_i - S_i^{k+1}$ is calculated.

6. Trend smoothing: Smoothing of the deseasonalised series with LOESS using $q = n_{(l)}$ and $d = 1$ generating T_i^{k+1}.

Outer loop: Each pass of the outer loop requires a previous pass of the inner loop, but not vice-versa. The outer loop executes a computation of robustness weights that are used in the following run of the inner loop to reduce the effect of transient/aberrant behaviour in the seasonal and trend components. After a pass of the inner loop, estimates are generated for S_i^{k+1} and T_i^{k+1}. Then the remainder is $I_i^{k+1} = x_i - S_i^{k+1} - T_i^{k+1}$. The robustness weight for point i will be:

$$\rho_i = B\left(\frac{|I_i|}{h}\right), \tag{2.6}$$

where

$$B(u) = \begin{cases} (1-u^2)^2 & \text{for } 0 \leq u < 1 \\ 0 & \text{for } u \geq 1 \end{cases} \tag{2.7}$$

and

$$h = 6 \text{ median}(|I_i|). \tag{2.8}$$

After this, the inner loop is repeated, but in the smoothing steps the neighbouring weights are multiplied by the robustness weights.

Lastly, both weight functions $W(u)$ and $B(u)$, with their quick decay, act as filters for the STL algorithm, preventing the influence of points that are classified as too far as per the function definition. Although these two functions are defined here following the original design by Cleveland et al. [28], other functional forms can be used in their place as long as they fulfil the same purpose of penalising input data that is too far apart in time in the case of $W(u)$, or that are greater than a certain threshold multiplier of the median value of the remainder in the case of $B(u)$.

2.1.4.3 Spectral Methods and Fast Fourier Transform

Spectral methods for time series revolve around the idea of representing a series as a superposition of sinusoidal functions of different frequencies. Their use in the research presented in this thesis will be limited to a transformation of the original series in the time domain into a complex-valued function in the frequency domain by using the Fast Fourier Transform (FFT) [29] algorithm to then filter certain frequencies before performing the inverse transformation.

The FFT is an algorithm such that the Discrete Fourier Transform of a function can be calculated in $O(N \log N)$ operations instead of the $O(N^2)$ required by simply evaluating the definition below.

A Discrete Fourier Transform of a series x_i into a complex series X_k with $i = 0, ..., n-1$, $k = 1, ...n$, which, using j as the imaginary unit, is defined as:

$$\begin{aligned} X_k &= \sum_{i=0}^{n-1} x_i \cdot e^{-\frac{j2\pi}{n}ki} \\ &= \sum_{i=0}^{n-1} x_i \cdot \left[\cos\left(\frac{2\pi}{N}ki\right) - j \cdot \sin\left(\frac{2\pi}{N}ki\right)\right] \end{aligned} \tag{2.9}$$

The Discrete Fourier Transform, is an invertible linear transformation, in which the inverse transform is formulated as

$$x_i = \frac{1}{N}\sum_{k=0}^{n-1} X_k \cdot e^{j2\pi ki/N}, \tag{2.10}$$

being necessary to bring the series back into the time domain after filtering.

The main limitation of the Fourier Transform variants is that it provides no information about when the different frequencies in a signal are important in non-stationary signals. This prevents the Fourier Transform from being used to locate abrupt changes or beginnings and ends of events. In order to overcome this limitation, the Windowed (Short-Time) Fourier Transform was introduced, dividing a longer signal into segments of equal length and then computing the Fourier Transform separately for each segment. This allows for a degree of time resolution, but still displays limits to the certainty of the calculations determined by the Gabor limit or Fourier Uncertainty Principle, which in a manner not dissimilar to the Heisenberg Uncertainty Principle states that it is impossible to simultaneously localise a signal in both frequency and time in certain terms, as discussed by Hall [30], establishing limitations in terms of frequency resolution and temporal resolution. Here, the classical Fourier Transform allows measurements with measuring all frequencies, but in doing so, all information relative to the temporal component is lost, making the time at which a specific frequency appeared unknown. By taking a windowed approach, this is mitigated, but the underlying problem remains, as the precision in both time and frequency is determined by the size of the window.

A useful alternative to obtain both kinds of information is the Wavelet Transform, which takes advantage of the intermediate cases before reaching the Gabor limit, as will be explained in the following section.

2.1.4.4 Continuous Wavelet Transform

A wavelet can be thought of as a brief oscillation, with an amplitude that has effectively limited duration, starting and ending at zero. The specific way in which it varies over time depends on the specific choice of wavelet family. Wavelets are localised in time and frequency, allowing for the identification of sparse events. Most wavelets families are created to have specific qualities that make them suitable for signal processing.

Examples of the components of a complex-valued Morlet wavelet are shown in Fig 2.3.

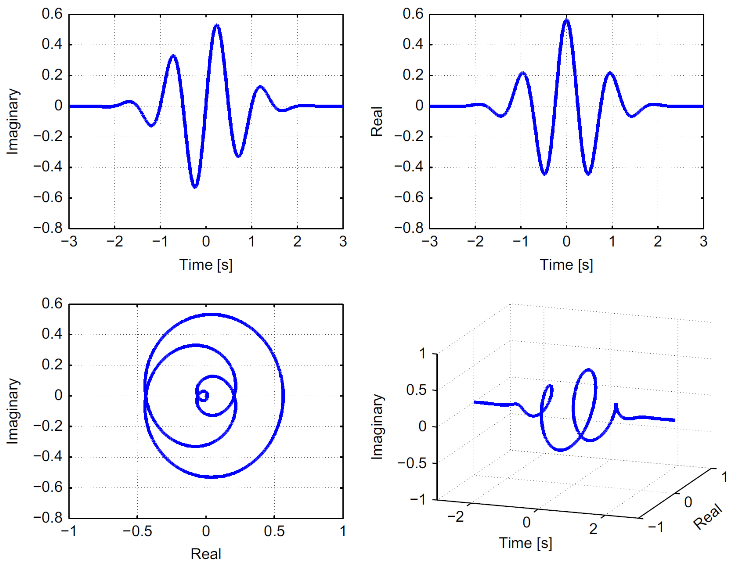


Figure 2.3: Example of a complex-valued Morlet wavelet. Source: Karimi et al. [31].

The Continuous Wavelet Transform [32, 33] is a linear transformation that results of the convolution of the input series with a set of functions derived from the chosen wavelet family. This permits the localisation

Let $L^2(\mathbb{R})$ be the space of square integrable functions, let $\overline{\psi}$ be the complex conjugate of ψ, and let $*$ represent a convolution. The Continuous Wavelet Transform of a function $f \in L^2(\mathbb{R})$ is defined as:

$$\mathcal{W}_f(s_c, t_r) = \int_{-\infty}^{\infty} f(t)\overline{\psi_{s_c,t_r}}(t)dt = f * \psi_{s_c,t_r}; \tag{2.11}$$

where ψ represents a complex-valued window function, also called *mother wavelet* parametrised using a scale (dilation) parameter s_c and a shift (translation) parameter t_r into specific wavelets ψ_{s_c,t_r}.

$$\psi_{s_c,t_r}(x) = \frac{1}{\sqrt{s_c}}\psi\left(\frac{x - t_r}{s_c}\right) \tag{2.12}$$

Full derivation details can be found in Kaiser [34]. To be classified as a wavelet, a function ψ must satisfy three different criteria [35]:

1. It must have finite energy:

$$\mathcal{E} = \int_{-\infty}^{\infty} \|\psi_{s_c,t_r}(t)\|^2 dt < \infty \tag{2.13}$$

2. Admissibility condition, the admissibility constant C_g must be finite, implying the wavelet has no zero-frequency component. If X_ψ^{fr} is the Fourier Transform of $\psi_{s_c,t_r}(t)$, then:

$$C_g = \int_0^{\infty} \frac{\|X_\psi^{f_r}\|^2}{f_r} df_r < \infty \tag{2.14}$$

3. For complex wavelets, the Fourier Transform must both be real and zero for negative frequencies.

Then, the inverse CWT is defined as:

$$f(t) = \frac{1}{C_g}\int_{-\infty}^{\infty}\int_0^{\infty} \mathcal{W}_f(s_c, t_r)\psi_{s_c,t_r}(t)\frac{ds_c\, dt_r}{s_c}^2 \tag{2.15}$$

The CWT discretises scale by fixing a base, typically of the form $2^{1/v}$, $v > 1$, where v is the number of *voices per octave*. This name is used because increasing the scale by an octave (doubling the scale) requires v intermediate scales, providing more granularity as v increases. In contrast, the Discrete Wavelet Transform (DWT) only uses powers of 2 for the discretisation of the scale, hence the *continuous* nature of the CWT refers to the progression across scales. To obtain said different scales, this initial base is raised to positive integer powers, e.g. $2^{l/v}$, $l = 1, 2, ..., n$. The translation parameter t_r is discretised to integer values, spanning the length of the series, and making the imposition of contour conditions necessary, which in turn defines an uncertain area surrounding the margins of the series.
In a basic manner, the CWT functions as follows:

1. Generate initial wavelet ψ_{s_c,t_r} .

2. Compare the wavelet against a section at the start of the signal, with its length defined by s_c and its center defined by t_r, by calculating the complex-valued $\mathcal{W}_f(s_c, t_r)$, representing how closely correlated f and ψ_{s_c,t_r} are in this section.

3. Shift the wavelet to the right by increasing t_r and repeat 2 until the end of the signal.

4. Scale (stretch) the wavelet by increasing s_c to target lower frequencies, and repeat steps 2-3 for the next voice.

This process produces a series of wavelet coefficients $\mathcal{W}_f(s_c, t_r)$, corresponding with the different values of scale (including all octaves) and shift that represent the similitude between the shifted and scaled versions of the mother wavelet and the original signal. These coefficients given an indication of how influential are the oscillations of a given frequency (scale) is at a given point in time (translation) and can be represented in a scaleogram, usually expressed in terms of power $P = \|\mathcal{W}_f(s_c, t_r)\|^2$. Intuitively, in the cases where the sampling period of the signal equals a unit of time, $P\left(W_f(s_c, t_r)\right)$ will represent how influential for the dynamics of the signal during a window of length s_c centred at the temporal point t_r.

Here, the Generalised Morse wavelets, a family of exactly analytical wavelets will be used, based on its suitability for time-frequency analysis, and specifically for their usefulness for analysing localised discontinuities. The CWT can be thought of as the inner product of the source signal with the wavelets used as basis, where the wavelet coefficients represent the level of similarity of the signal and the wavelet at a given scale. If the basis wavelets were orthogonal, energy would be preserved by the transformation, with the wavelet coefficient being the correlation coefficient between signal and wavelet at that point, times the energy of the signal.

While the Fourier transform always projects the signal onto an orthogonal set of components, making it energy preserving (making them useful in de-noising applications), as it is the case with the Discrete Wavelet Transform (when using orthogonal wavelets), in the case of the CWT this is only true in its integral form, but not in the case of numerical computation [36]. When the CWT is computed numerically, as it will be the case in this thesis, regardless of the normalisation used to ensure that the transformed signal will have the same energy at every scale, the CWT is not an orthonormal transformation, making it non-energy preserving when taking the inner product of the function with the basis wavelets. L1 normalisation is used in this case for a more accurate representation, ensuring that equal amplitude oscillatory components in the data at different scales, will have equal magnitude in the CWT [36].

From a practical point of view, orthogonality is not required for time-frequency analysis, its effect being limited to the generation of informational redundancies, being shown as highly correlated output data [37]. Beyond the choice of a family of wavelets for a task (i.e. not using a discontinuous wavelet for continuous data), "the choice is largely one for the researcher, and is likely limited by the software being used" [38].

The Wavelet Transform gives good time resolution for high frequency events, and good frequency resolution for low frequency events, which is well suited for analysing many real-world signals. It can be more useful than the Fourier Transform to approximate signals with discontinuities, due to the time-localised behaviour. Regarding the numerical implementation, which is not trivial, calculations presented here use the MATLAB baseline implementation in the Wavelet Toolbox [36].

Their use within the research here presented is to identify and separate the recurrent and non-recurring congestion of the travel time signal, making use of its additive characteristics. Their advantage with respect to traditional spectral methods is that instead of identifying the spectral power of a given frequency over the length of a signal, the Wavelet Transform provides this power spectrum as a function of time.

2.2 Reinforcement Learning for Traffic Signal Control

Although there is a growing research literature for Reinforcement Learning (RL), arguably much less investment has been placed on application of these algorithmic developments in Traffic Signal Control (TSC). Nonetheless, as we discuss in this thesis, there is a potentially equally important role to be played by the development of RL methods for traffic signal control.

TSC is essential to the smooth movement of traffic in towns and cities. Junctions in dense urban areas often experience conflicting traffic movements. These are movements where the path of a vehicle will cross the path of another vehicle if its movement is not interrupted. When load traffic is relatively light or when there is sufficient space, this can be managed with fixed road signs or by constructing roundabouts. However, more complex junctions in dense urban areas are managed by traffic signals. Here traffic lights are used to select phases which consist of several non-conflicting movements. Urban Traffic Controllers (UTCs) are either fixed plan, or adaptive.

The sequence of phases selected and their duration are managed by an UTC system, which coordinates these phases both within a junction and between junctions. The simplest form of traffic control is to have a fixed timed cycle. Although as we will discuss, this control strategy both in theory and in practice is less efficient in adapting to changing traffic patterns. Adaptive traffic control systems have been developed over a number of years. Adaptive traffic control systems sense the present state of congestion at a junction through loop detectors and cameras, and then choose phases with durations that reacts to current traffic demands.

A further ingredient that is an important consideration in the design of TSC systems is simulation. Modern TSC systems are evaluated by simulating the movement of vehicles through different junctions. In this way the performance of a traffic signal control system can be evaluated and improved before it is released on the roads. RL provides a systematic method for training and optimizing algorithms using simulation, so it is only natural to attempt to combine the two. The research presented in the last 3 chapters of this thesis aims to understand how we can better utilize this process used by traffic engineers. Reinforcement learning has seen a surge in interest in recent years due to several notable success stories in Deep Reinforcement Learning (DRL). In it, a neural network approximates the optimal value function and policy function of a Markov Decision Process. These algorithms can perform at super-human level in ATARI games [39] and can beat professionals in long-term games requiring advanced planning such as the game of Go [40] and StarCraft 2 [41]. These approaches have existed for a number of decades prior, see Tesauro [42] and Bertsekas and Tsitsiklis [43]. Although human performance is not the benchmark in the study of TSC systems, it can be argued that similarly significant improvements may be possible when compared with existing TSC systems.

2.2.1 Current Traffic Signal Control Systems

Traffic signaling was originally used in rail transport.The first road traffic signal was developed by John Peake Knight in 1868. Electric traffic lights were installed in Cleveland USA in 1914. Automatic lights appeared in Wolverhampton UK in 1926. Vehicle actuated traffic lights we installed in London in 1932. Since then the use of traffic lights has grown dramatically, with modern cities having thousands of traffic lights on their streets. A good history of TSC can be found in Hamilton et al. [44].

Regarding modern TSC, MOVA (Microprocessor Optimised Vehicle Actuation) is an adaptive signal control systems for single intersections developed by the Transport Research Laboratory [45]. MOVA is primarily used in the UK with thousands of junctions using the system. BALANCE [46] is a proprietary UTC owned by PTV Group. Balance is designed to coordinate a network of junctions. The method was proposed by Friedrich et al. [47] and uses Genetic Algorithms (GA) for the optimization of signal timings. The BALANCE system is primarily implemented in Poland and Germany.

TRANSYT and SCOOT (Split Cycle and Offset Optimisation Technique) are two further variants of MOVA developed by the Transport Research Laboratory. TRANSYT is a tool used for optimizing fixed time signals [48, 49, 50]. SCOOT is a vehicle-actuated UTC from Transport Research Laboratory, which attempts to optimize the Split and Offset of traffic signals for sets of junctions using loop detector inputs [51, 52]. PTV EPICS is a signal control systems that optimizes traffic signal timings using similar methods to the TRANSYT system [53].

SCATS [54] is a similar system to the SCOOT system, attempting to optimise Split and Offset of traffic signals based on loop detector input. It is primarily used in Australia.

Utopia, Rhodes and OPAC each apply hierarchical optimal control. Utopia performs a closed loop control optimization of individual traffic lights using loop detector information and then aggregates to a larger scale optimization [55, 56]. The Rhodes system developed in Mirchandani and Head [57], providing a hierarchical architecture for wide area traffic signal control. The local control used is described in Head [58]. OPAC follows a two level dynamic programming approach to traffic signal control [59], being trialed in Virginia and Arizona.

PRODYN and SurTrac both implement forward solving dynamic programming to individual traffic junctions. These systems do not use cyclic phases. PRODYN is a Dynamic Programming based traffic signal optimizer [60]. It uses forward solving Dynamic Programming algorithm. It was trialed in France and implemented in a number of countries. The SurTrac system was developed in Carnegie Mellon University by Smith and Barlow [61]. The system solves a forwards implementation of dynamic programming typically used to optimize computer CPU operations.

2.2.2 Simulation of Traffic Signal Control systems

Simulation data is required to train RL algorithms for TSC, since the deployment of untrained systems to generate real-world data is unfeasible from a practical point of view.

There are three main systems for simulating urban traffic control: SUMO, Aimsun Next and PTV Vissim. Each of them offers a python API which allows for the retrieval of different magnitudes in the simulation in real time(Other simulators such as JCT LinSig and Systra Paramic are used for evaluation of UTC. However, API access is not available). Aimsun and PTV Group were, respectively, purchased by Siemens in 2018 and Porsche SE in 2017. This has been interpreted as part a move from car and infrastructure providers into a broader mobility services space.

SUMO is an open source traffic simulation software originally developed by the Deutsches Zentrum für Luft- und Raumfahrt (DLR, German Center for Air- and Space-flight) in Berlin [62]. The system is coded in C++ and source can be accessed on from its website.[1] Because code is open source, it is easier to implement parallel algorithms and thus reduce the time required to gain simulation results. This is useful for the training of larger scale RL algorithms, such as Asynchronous Actor Critic (A3C) [63] that require a central agent and a number of workers performing parallel simulations.

Aimsun provides simulation software for microscopic, mesoscopic and macroscopic simulation of traffic. Aimsun Next is the current main incarnation of this software which runs predominantly on desktop computers. Though products for live traffic prediction are now available. Aimsun as increasing support for the simulation of self-driving cars. It supports the connection with externally provided UTCs such as SCATS, SCOOT, and UTOPIA.

PTV Vissim is a microscopic simulator from PTV group. Macrosimulation is provided through its PTV Vissum software. It supports the connection with externally provided UTC algorithms such as SCATS, SCOOT, and UTOPIA. PTV are developing cloud platform variants of their software which, again, will be important for the training of RL algorithms for larger scale UTCs.

Aimsun and PTV Vissim offer similar functionalities. From our conversations with traffic modellers and controllers, Aimsun was primarily used for junction design, where as Vissim was more frequently used for the evaluation and optimisation of TSC. Large cities such as London use PTV Vissim for UTC modelling. However, both systems offer a much broader and comparable set of uses, and their use in traffic modelling will vary from region to region.

In the research presented in this thesis, Vissim was used for those projects in collaboration with Transport for London and the Toyota Mobility Foundation, for easier knowledge transfer with the transport authorities. SUMO was used on those others done in collaboration with Vivacity Labs due to the focus on performance and large-scale parallel simulations.

Finally, we note that unlike autonomous vehicles, where there are a multitude of packages for the comparison and evaluation of the Reinforcement learning algorithms [64, 65], there is very little software available for this purpose in TSC,

[1] https://www.eclipse.org/sumo/

leading to the requirement of implementing or setting them up manually, hence reducing the amount of feasible comparisons.

2.2.3 Description of Current Traffic Signal Control Systems

In this section, the TSCs that will be used for comparison against the controllers proposed in this thesis will be introduced.

2.2.3.1 Maximum Occupancy

Maximum Occupancy (MO) is a heuristic approach to TSC. It is capable of obtaining great performance, despite its simplicity. In an intersection being managed by a MO controller, when a new green stage is to be selected, all queues (or lane occupancies) are calculated, and the stage that would serve the greatest amount of aggregated queuing vehicles is selected. To do this either induction loops, occupancy sensors, or any other form of sensors able to provide these measures are required.

2.2.3.2 Vehicle Actuated - System D

According to the Traffic Advisory Leaflet of the UK's Department for Transport, "it is still probably the most common form of control for isolated junctions" [66]. It can either use induction loops at specific distances upstream from the intersection (12, 25, and 39 metres) or Above Ground Detectors to perceive incoming vehicles.

A vehicle that approaches a non-green signal will be detected and will register a demand for green in the controller, which will follow a cyclic schedule, being able to skip stages with no registered demand. Whenever two phases must follow, the first will trigger a demand for the second one. After a stage reaches the maximum green time, a demand is registered for that stage to be implemented once other demands are met. If vehicles are detected moving towards a green signal, the duration of the stage can be extended. "On expiry of the last extension and with no more vehicles detected, the controller will answer a demand for another stage, either at the end of the minimum green period, or immediately if this has already expired. If vehicles continue to extend the green period and a demand exists for another stage, the green signal will be terminated on expiry of a preset maximum period after the demand has been received. If there are no demands for another stage the signals will normally not change. However, in the absence of demands, they can revert to a pre-determined stage, say, the main road, or an all-red" [66].

2.2.3.3 Microprocessor Optimised Vehicle Actuation (MOVA)

MOVA was developed by Transport Research Laboratory, and requires induction loops placed about 40 (this being called the X-detector) and 100 metres (IN detector) upstream from the intersection in all lanes. The detectors are connected directly to the MOVA computer system, a self-contained, microprocessor-based module, which interacts with the traffic controller. MOVA requires grouping lanes into links of 1-3 adjacent lanes that are on the same traffic phase (the displayed signal is set jointly for them).

MOVA takes a sequential approach to its decision making, conditioned on flow and queue information derived from the detectors, so a green stage is composed of:

1. The minimum green time (7 seconds in the UK), plus a minimum per link green to serve vehicles that have surpassed the closest detector to the junction.

2. A period when the queue on at least one link is deemed to be discharging at saturation flow, based on the size of the 'gap' between vehicles as measured per the closest detector to the junction. This lasts until all relevant links' (those that will see their corresponding signal changed to red in the next phase) discharge rate have at least one lane which has fallen below saturation flow.

3. A period when MOVA, according to an internal model that estimates vehicles' positions from detector data every half second, weights the benefits of further extending the green time based on estimations of vehicle counts and arrival flows. This step considers different options, aiming to choose that which minimises the expected wait for vehicles in the affected links that would need to wait and that of the remaining vehicles waiting in red stages.

This description covers the basic operation of MOVA. There are a series of extra considerations in the case of oversaturated links, where MOVA seeks to maximise capacity by allowing green signals to continue as long as possible, if the rate of discharge is above a certain threshold. Extended details can be found in Vincent and Peirce [67] and UK Department for Transport [68].

2.2.3.4 Split Cycle Offset Optimisation Technique (SCOOT)

SCOOT is not used, nor compared against at any point in this document, being included exclusively for descriptive purposes.

SCOOT makes intensive use of detectors upstream for any given controlled junction or "node". One or more of these nodes can be run jointly as a "region". SCOOT generates information about flow and occupancy based on this sensor data, coding it in a "link profile unit", which are used to create "cyclic flow profiles" of these units for each link.

SCOOT uses three separate optimisation procedures:

1. Split Optimiser: at every stage change it assesses the current timings and determines whether they should be advanced or retarded by 1 to 4 seconds per stage, or whether they should stay the same.

2. Offset Optimiser: It uses the cyclic flow profiles for each link, to assess once per cycle on each node, whether the current action times should be advanced, retarded, or stay the same, using 4 second increments.

3. Cycle Time Optimiser: It operates per region, either every 2.5 or 5 minutes. It identifies a critical node in the region and tries to adjusts cycle time to keep it over 90% saturation rate. It can change cycle times by 4, 8, or 16 seconds.

Extra details regarding installation, components and other information can be found in UK Department for Transport [69].

2.2.4 Academic Literature

In this section, previous academic literature is reviewed, focusing first of research and optimisation methods applied to TSC systems, then on RL with a particular focus on deep reinforcement learning, and finally on approaches using deep reinforcement learning used to develop UTC.

2.2.4.1 Traffic Signal Control Research

The approaches here presented can be roughly categorised as follows: queueing theoretic, heuristic optimization, mathematical programming, and optimal control. Early research will be introduced first, to then focus on the indicated categories.

Early Work: Academic work on TSC systems first emerged in the late 1950s and 1960s. Here are variety of approaches and modelling solutions were proposed. An interesting personal perspective on these developments is given in Newell [70]. The paper of Wardrop and Whitehead [71] is seen as the seminal work on the modelling of road traffic networks. Wardrop defines his famous notion of a user optimum and system optimum and also studies traffic signal timing. Following this, important works on fixed signal timings include Webster [72] and Miller [73]. Both of which are important in the development of the TRANSYT and MOVA systems. A number of the calculations in these works are backed up by Newell [74]. The earliest work applying a Markov Decision Processes (MDP) to traffic signal control appears to be in 1967 by Martin-Löf [75].

Queueing Theory: A number of papers look into queueing theoretic models of traffic. Here an emphasis is placed on understanding the stationary distribution of queue sizes under different network configurations. This can provide useful insights into performance, but is limited to idealized settings. Fixed cycle timings are certainly tractable, see McNeil [76] for early analysis and Boon et al. [77] for a recent analysis. Additional methods are required to then optimize these mathematical model. The stability of queueing networks can be counter intuitive [78] and this feeds into the work on TSC systems by Lämmer and Helbing [79].

Mathematical Programming: Mathematical programming approaches, cast the TSC problem as the optimization of a function subject to constraints. This is useful particularly for fixed time plans, focusing on the time splitting. However, such approaches are not best suited for forward planning over a time horizon. Linear programming and mixed-integer programming approaches are given by Gartner et al. [80, 81]. A multi-objective version is given by Dujardin et al. [82]. An formulation as a Linear Complementary Problem is given by De Schutter [83]. Quadratic programming approaches are given in Li et al. [84].

Heuristic Optimization: Heuristic algorithms are often used to find good solutions large and combinatorial optimization problems. Numerous heuristics exist and have been applied to TSC these include –but are not limited to– simulated annealing [85], evolutionary and genetic algorithms [86, 87, 88], cross entropy method [89] and neural networks [90]. Existing controllers such as PTV Balance use genetic algorithms. Further, as we will discuss shortly, the cross entropy method and neural networks can be applied in reinforcement learning to help optimize Markov decision process.

Optimal Control: TSC can be framed as an optimal control problem. Here a dynamic program or MDP must be solved over time. To the best knowledge of the author, the earliest paper advocating this approach is Martin-Löf [75]. There are a number of UTCs using systems based around the solution of DP, see [91, 57]. Within this there are a number of different approaches of which Reinforcement Learning is just one. For instance, Model Predictive Control (MPC) where the optimal next step is take for a model over a finite time horizon [92, 93, 94, 95, 96, 97, 98]. Forward

Dynamic programming is used by the PRODYN and SurTrac UTC systems. Here the task of processing batches of cars can be recast a shortest path problem which can be solved using a forward Bellman recursion.

2.2.4.2 Reinforcement Learning

Reinforcement Learning combines the study of Markov Decision Processes with supervised learning. Much of the theory of DP and MDP was laid-out by Richard Bellman [99]. Here the task of an agent is to optimize the sum of a sequence of rewards for the states reached by a process over time. On a high level the process is as follows: the agent receives a representation of the state of the system, the agent then selects an appropriate action. The system then advances to the next state, and generates a reward based on the effect of the action chosen by the agent. This process is repeated until the finalisation of the control problem. When the distribution of dynamics and rewards are known then iterative solution of the Bellman equation yields an optimal control. When the distribution of rewards and dynamics are unknown then these must be estimated, either explicitly or implicitly.

This distinction, regarding the knowledge about the environment, sets apart the DP and RL approaches. While DP algorithms require a model of the MDP (in terms of transition dynamics and rewards) and normally work offline, RL algorithms are model-free and can work both online or offline, estimating the distribution of rewards from extended interaction with the environment.

When state spaces and action spaces are large, maintaining estimates for each state and action becomes impractical. Instead the value of the estimated future reward can be approximated through a function approximation. Whenever there is an estimation or approximation of the reward or transitions of the environment, these algorithms are referred to as Approximate DP and RL with Function Approximation. The main differences between the different variants of DP, including Approximate DP and RL stem from the knowledge about the environment mentioned earlier.

DP algorithms are essentially "planning" methods, that can iteratively compute an optimal policy, and value function from their knowledge about the environment. This is done by breaking the optimisation process into a series of simpler sub problems at different points in time and "going backwards in time" from the best outcomes, although this can turn computationally expensive. In the case of Approximate DP, optimisation and simulation are used in combination, generating approximations of the optimal values of Bellman's equations and optimal policies. This is done by generating sample paths over states, "stepping forwards in time" for a number of iterations, using them to learn the value functions and allow for generalisation over the space of states.

RL, in turn, requires a set of samples of actions in the environment, the transitions it generates and the corresponding rewards, which can be generated either online or offline. RL uses these sampled experiences to update its estimates about the reward that will be generated by a certain action by optimising the parameters in its function approximator. This learning, unlike with DP, does not need to occur at the end of an control episode, but can instead take place immediately after every time step. Given that every transition generates a reward, in RL this reward can be immediately used to generate a target, indicating how far the previous estimate was from reality and allowing for immediate improvement.

The simplest function approximation is a linear function approximation, and surprisingly, theoretical guarantees on performance are possible [100]. The policy can also be directly estimated and improved in policy gradient methods [101] and actor-critic methods [102], which map states to probability distributions over the available actions.

More regression models are possible, the most popular of these is neural networks. Success was achieved in this direction by Tresauo's TD-Gammon [42], which built on the work of Sutton [103]. More recently, efforts have been made to improve training for deep neural networks. These include the use of replay memory and fixing targets during training runs to make Deep Q-Networks [39] (DQN) more stable, and the asynchronous implementation of actor-critic algorithms [63], double Q-learning [104, 105], prioritized experience replay [106] and dueling architectures [107].

Excellent treatment of RL and historical accounts can be found in [108] and [43]. Important milestones in the development of reinforcement learning are Q-Learning [109], SARSA [110] and Temporal difference methods [111]. These tabular methods apply for moderately small state and action spaces.

These techniques are discussed in greater detail later in this document, but essentially these methods have the effect of de-correlating Stochastic Gradient Descent (SGD) steps taken by the algorithm, and separating out the task of prediction and estimation.

2.2.4.3 Reinforcement Learning in TSC

Early works applying reinforcement learning to traffic signal control emerged in the late 70's [112]. Though more tabular RL methods, such as SARSA and Q-learning, above began to be applied in the mid-to-late 90's, see [86, 113, 114, 115].

Function approximation in RL was applied throughout the 2000's. Bingham [90] is the earliest work the author could find applying neural networks for reinforcement learning in UTC. However, RL approximations are applied to TSC before these works, for instance, see Mikami and Kakazu [86]. A more developed application of neuro-dynamic programming to traffic signal control is given by Srinivasan et al. [116]. Multi-agent approaches to UTC are first considered by Wiering [117] and Actor-Critic are employed by Richter et al. [118]. Linear function approximation and TD methods used employed by Cai et al. [119] and a single layer neural network for Q-function approximation is used by Arel et al. [120].

With the advent of deep learning, there has been a rapid growth in research applying of DRL to generate UTCs in the 2010's. The first DQN implementation is in the Masters project of Rijken [121], quickly followed by others who make incremental additions to the approach. In 2014, El-Tantawy et al. [122] review previous early approaches and simulate five intersections with realistic geometry, proposing a variety of state representations and rewards. In 2016, van der Pol and Oliehoek [123] studied DQN applied to simple networks with 2 phases per junction. In 2017, Gao et al. [124] introduces the target network and experience replay to the problem and Mousavi et al. [125] produce states directly from raw pixels in a simple intersection, using a Convolutional Neural Network (CNN) as function approximator and finding value-based methods and policy gradient methods to have comparable performance in this setting. In 2018, Liang et al. [126] uses a CNN approach, using pseudo-pixels by partitioning the intersection into cells, the values of which represent the presence or absence of vehicles.

The PhD work of Genders is a good example of a SUMO implementation [127, 128, 129, 130], where it is shown that denser state representations do not necessarily have an impact on the performance of a trained agent.

Relatively few modern works in the field compare results with fixed-time or modern UTC systems (both centralised and distributed), thus it is hard to qualify the quality of these techniques. Notable exceptions being Chu et al. [131] which expands deep learning actor-critic methods to multiple junctions and compare with fixed time plans using Aimsun, and Cai et al. (2009) [119], who compare results a fixed time system optimized by TRANSYT 12.0.

We present further problem-specific literature review of reinforcement learning in TSC in Section 5.2, Section 6.2 and Section 7.2, aiming to better identify the gaps in the literature that the research there presented attempts to cover.

2.2.5 Deep Reinforcement Learning

We now give a brief mathematical description of the reinforcement learning problem, function approximation and the algorithms that have been implemented. These descriptions will be further covered in following chapters to link it to the problem at hand.

2.2.5.1 Markov Decision Processes

A Markov Decision Process (MDP) is an optimization over time, looking for a map between actions and states that maximises some measure of future returns. It is defined in terms of the tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \mathcal{R} >$. Here there are a set of possible states $s \in \mathcal{S}$ that the system can take, and a set of actions $a \in \mathcal{A}$ allowed to an agent. The selection of actions by the agent is modelled as a map which we will refer to as policy (π) that specifies the action $a = \pi(s)$ to be taken. When joining an MDP with a policy in this way, the combination behaves as a Markov Chain since the actions are only determined by the state s.

The state evolves according to the probabilistic transition function $\mathcal{T}$, which depends on the current state s_t and the agent's action a_t, that is $s_{t+1} = \mathcal{T}(s_t, a_t)$. Further, the agent receives a reward $r \in \mathcal{R}$, the value of which depends on the state and action chosen $r(s_t, a_t)$. Often rewards are discounted by a factor $\gamma \in (0, 1)$, which is used to ensure more recent rewards are more important and defines a temporal horizon to the problem.

Given initial state s_0, an MDP looks to maximize the expected discounted return via the following optimization:

$$V_\pi(s_0) = \text{ Maximize } \quad \mathbb{E}[G_t] \qquad t = 0, 1, 2, ... \tag{2.16}$$

Where the return is defined as:

$$G_t = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \tag{2.17}$$

This represents the total expected discounted reward, starting from state s_0 and following policy π until the end of the control problem. The function $V_\pi(s_0)$ in Eq. 2.16 is known as the value function of the MDP and satisfies the Bellman equation

$$0 = \max_{a_t \in \mathcal{A}} \quad r(s_t, a_t) + \gamma \mathbb{E}_{s_t, a_t}[V_\pi(s_{t+1})] - V_\pi(s_t) \tag{2.18}$$

for all s_t, where s_{t+1} is the next state after taking action a_t in state s_t as per the probabilistic transition function of the system $\mathcal{T}$, and $\mathbb{E}_{s_t,a_t}$ takes the expectation over its probability distribution. This can be equivalently written as

$$0 = r(s_t, a_t) + \gamma\mathbb{E}\Big[\max_{a_{t+1}\in\mathcal{A}} Q(s_{t+1}, a_{t+1})\Big] - Q(s_t, a_t) \tag{2.19}$$

where $V(s_t) = \max_{a_t\in\mathcal{A}} Q(s_t, a_t)$. Here $Q(s_t, a_t)$ is known as the Q-value and is the expected return received after initially taking action a_t while in state s_t.

Further, if π is a probabilistic policy, here $\pi(a|s)$ denotes the probability for choosing action a in state s. Then the expected reward from that policy V_π satisfies

$$0 = r(s_t, a_t) + \mathbb{E}_{s_t,\pi}[\gamma V_\pi(s_{t+1})] - V_\pi(s_t) \tag{2.20}$$

where $\mathbb{E}_{s_t,\pi}$ denotes the expectation of the next state s_{t+1} given s_t and the policy $\pi(a|s)$.

2.2.5.2 TD-methods and Q-learning

We now describe the basic RL methods, before diving into how they work in conjunction with deep learning.

In RL typically we do not have a closed form for the distribution of the next state s_{t+1} given the current state and action, s_t and a_t. Instead it is evaluated by simulation. Thus typically we must numerically evaluate V_π and V to know the reward of a policy and to calculate the optimal reward.

To evaluate a policy, for each simulated s_{t+1} from state s_t and action a_t generated from policy $\pi(\cdot|s_t)$, we perform the update

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha\left(r(s_t, a_t) + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)\right) \tag{2.21}$$

Here and hereafter, the arrow refers to one update to $V(s_t)$ by the algorithm. The resulting algorithm is called $TD(0)$ [108] and is a basic routine for evaluating the value of policy π, V_π. If the parameter α which we call the learning rate decreases sufficiently slowly then convergence to V_π is guaranteed.

If we wish to evaluate the optimal Q-values then we update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big(r(s_t, a_t) + \gamma \max_{a_{t+1}\in\mathcal{A}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\Big). \tag{2.22}$$

This algorithm is known as Q-learning [109] and provided states and actions are visited infinitely often and, again, α decreases suitably slowly then convergence to the optimal Q-values and value function is guaranteed [132].

2.2.5.3 Function Approximation and Neural Networks

Often it is not possible to maintain a table of all states and actions: first, because not all states will be visited frequently enough to gain a reasonable prediction; and second, because there may not be sufficient memory to store this information. Hence instead it is natural to apply some form of approximation to summarize data received so far and to generalise decisions for thus far unobserved states.

In particular, we represent the relationship between a vector of inputs s and a real-valued (or vector-valued) output y through a parameterized function $\hat{y} = f_\theta(s)$ for a number p of parameters or weights $\theta \in \mathbb{R}^p$. Here $\hat{y}$ serves as a estimation of y from input s. The loss between output and the prediction is given by a loss function $L(y, \hat{y})$, typical choices are quadratic function in the case of value-estimation, or the relative entropy in the case of policy gradients. The objective is then to find weights that minimize the loss between inputs and outputs

$$\min_{\theta \in \mathbb{R}^p} \ \mathbb{E}\Big[L(y, f_{\theta(x)})\Big] . \tag{2.23}$$

For the purposes of the research here presented it is sufficient to assume that the function $f_\theta(s)$ is differentiable in θ so that we can apply Stochastic Gradient Descent [133] (SGD) on these parameters to minimize the above objective. To do this each time we gain data (s, y) we calculate $\hat{y} = f_\theta(s)$ and perform the update

$$\theta \leftarrow \theta - \alpha \frac{\partial L}{\partial \hat{y}}(y, \hat{y}) \nabla_\theta f_\theta(s) . \tag{2.24}$$

The algorithms used to train neural networks are essentially variants of this update rule.

We consider the setting where the function $f_\theta(s)$ is described by a neural network. Here $f_\theta(s)$ consists of the composition of a number of simpler functions. Specifically we take $a_j^{(0)} = s_j$ for $j = 1, ..., p$ and then recursively we take

$$a_k^{(l)} = g_k^{(l)}(z_k^{(l)}) \quad \text{where} \quad z_k^{(l)} = \sum_{k=1}^{K_{l-1}} w_{kj}^{(l)} a_j^{(l-1)} \qquad \text{for } k = 1, ..., K_l \tag{2.25}$$

where, finally, we define $f_\theta(s) = a^{(L)}$. We think of each pair (k, l), $k = 1, ..., K_l$, $l = 1, ..., L$ as representing a neuron. Here each l represents a layer of our neural network and k indexes each neuron at that layer. Each function $g_k^{(l)}$ is called an activation function. Typical activations are the logistic function, tanh, ReLU (Rectified Linear Unit) and linear. Each weight represents the significance of an activation on a neuron. Some weights can be set to zero in order to reduce the complexity of the model. Further a weight dependent functions can be added to the objective [134] in order to reduce the complexity of the model. This is called regularization.

With state and target (s, y) data the weights of the neural network can be updated with the SGD. The procedure required to calculate the derivatives in Eq.2.24 is called BackPropogation [135], and takes care of the book-keeping required to apply chain rule to the composite function $f_\theta(s)$. We note that in practice SGD is typically not used to fit a neural network, instead a number of purpose built variants exist. The research here presented uses ADAM (ADAptive Moment estimation) [136] for the training processes, which has a more complex update rule when compared to SGD.

The above description is sufficient for the purposes of describing the general basis of our algorithms and network architectures. Further experiment-specific details will be given in the methods sections of the following chapters.

2.2.5.4 Deep Q-Networks (DQN)

Although a significant number of neural network RL methods existed, the Deep Q-Network (DQN) was amongst the first to show significant success during the growth of deep learning. It is described originally by Mnih et al. [39] and further implementation details are described in Roderick et al. [137]. Subsequently there have been a number of extensions to this method which we will also discuss and study. DQN is a variant of Q-learning whereas A2C and A3C, discussed in the following sections, are a policy gradient methods.

DQN uses a neural network to estimate the action with the highest return based on $Q_\theta(s, a)$ and a different second network to maintain an estimate of the Q-values, $Q_{\theta'}(s, a)$, where as before θ and θ' denotes the weights of our neural networks. Over a number of update simulation episodes, the weights in the target network θ' are fixed and new weights are trained θ. The basic step of DQN takes a 4-tuple (s_t, a_t, r_t, s_{t+1}) where s_t is a state, a_t is the action taken $a_t = f_\theta(s_t)$, r_t is the reward received and s_{t+1} is the next state reached, and performs the following update

$$\theta \leftarrow \theta + \alpha \left(r + \max_{a_{t+1}} Q_{\theta'}(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t) \right) \nabla_w Q(s_t, a_t) \,. \tag{2.26}$$

Note that this is the SGD step, with input (s, a), output $y = r + \max_{a_{t+1}} Q_{\theta'}(s_{t+1}, a_{t+1})$ provided by the target network, and $L(y, \hat{y}) = (y - \hat{y})^2$. This approach is called fixed Q-targets, and it stabilises learning, avoiding the rapid changes of target that would be obtained when updating our objective estimation weights every time step. How often the update $\theta' \leftarrow \theta$ is performed will be an important hyperparameter of the system.

The next main feature of the algorithm is how the 4-tuples (s_t, a_t, r_t, s_{t+1}) are selected for learning. Each individual 4-tuple is taken from a simulation and is stored in memory, called the replay memory, the capacity M of which is another of the hyperparameters of the system. These memories are in principle sampled from the replay memory at random (although more sophisticated sampling methods have been developed which we will discuss shortly). The policy implemented by the DQN agents often is an ϵ-greedy policy. Here when, in state s_t with probability ϵ action a_t is selected at random with probability ϵ and with probability $1-\epsilon$ the action that maximizes the Q-function $Q_w(s_t, a_t)$ is selected. The parameter ϵ is decreased over a number of simulation runs. This is called the *cooling schedule* of the algorithm.

There are a number of extra hyperparameters that must be selected prior to training. These include learning rate α, the cooling schedule for ϵ, the number or layers and activation functions of the neural network. We will discuss each of these when presenting the experiments.

Below, some simple variants and additions to the basic DQN problem are introduced. Each of these add some (marginal) improvement to the basic algorithm.

Prioritized Experience Replay [106]: The idea here is to rank all the 4-tuples (s_t, a_t, r_t, s_{t+1}) in the replay memory. This is done by recording the absolute value of the TD error of each (s_t, a_t, r_t, s_{t+1}) in memory and then forming a ranking from highest to lowest. The TD error is given by

$$\delta = r + \gamma \max_{a_{t+1}} Q_{\theta'}(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t) \tag{2.27}$$

There are two basic variants: first, the highest ranked error in memory is selected; second and most common, the n-th highest ranked memory is sampled according to a probability depending on a parameter η:

$$P_n = \frac{n^{-\eta}}{\sum_{m=1}^{M} m^{-\eta}} \quad n = 1, 2, ..., M \tag{2.28}$$

A value of $\eta = 0$ corresponds to uniform sampling and $\eta = \infty$ corresponds to highest ranked. Both these rules introduce bias to our estimates. To correct for this we apply importance sampling to get an unbiased gradient update that is the same as uniformly sampling. The weights w_n are:

$$w_n = (\frac{1}{M} \cdot \frac{1}{P(n)})^{\beta} \tag{2.29}$$

So the basic Q-learning step becomes:

$$\theta \leftarrow \theta + \eta \left(\frac{1}{MP_n} \right)^{\beta} \delta \nabla_{\theta} Q(s,a) \, . \tag{2.30}$$

Note that if $\beta = 1$ then this is the correct (unbiased) importance sampling, however, it could be argued that earlier in training the gradient updates for TDs with large (and probably previously unseen) experience are more important. So taking $\beta \approx 0$ initially and linearly increasing to $\beta = 1$ is recommended. Another strategy is to set $\beta = 0$, as training with ultimately be most effected by large TD errors and by biasing towards these it can be attempted to deal with these errors as best as possible.

Dueling Architectures [107]: Given a value function and a Q-function, the advantage of a state action pair is given by

$$A_{\theta}(s,a) = V_{\theta}(s) - Q_{\theta}(s,a), \qquad s \in \mathcal{S}, a \in \mathcal{A}. \tag{2.31}$$

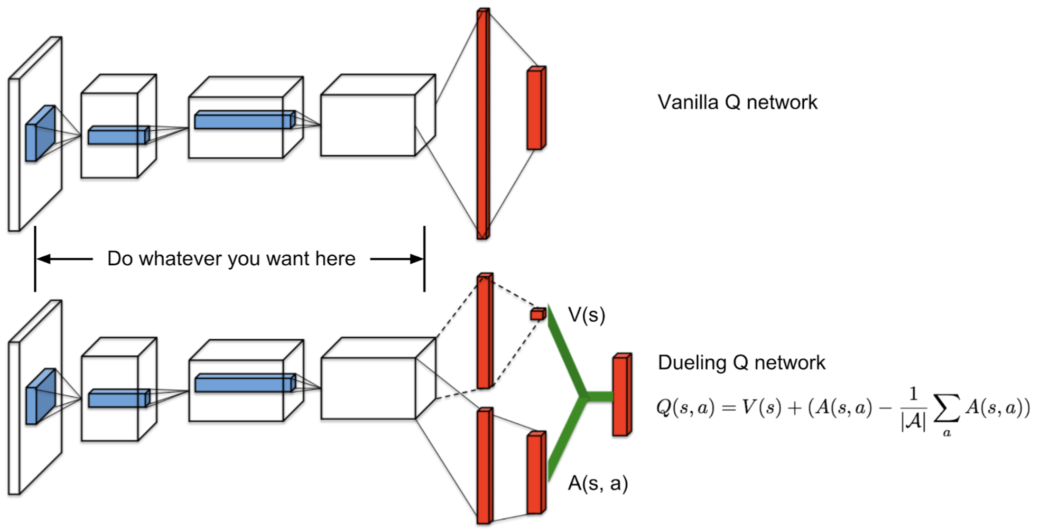

Vanilla Q network
Do whatever you want here
V(s)
Dueling Q network
$Q(s,a) = V(s) + (A(s,a) - \frac{1}{|\mathcal{A}|} \sum_{a} A(s,a))$
A(s, a)

Figure 2.4: Duelling architecture in Deep Q-Networks. Note that it only refers to the last layers indicated in red. Source: Wang et al. [107]

Recalling from Eq. 2.16 and Eq. 2.17, V_{π} represents the value of a state, without accounting for the choice of action; and from 2.2.5.1, $Q(s,a)$ represents the value of choosing a specific action while in a state. Hence, the advantage function calculates how beneficial is to take a specific action relative to all other actions available given a current state. Dueling architectures split the problem of approximating Q-values into the problem of approximation the optimal value function and the optimal advantage function.

Here the penultimate layer of the neural network is split with one neuron corresponding to the value function $V(x)$ and a further $\mathcal{A}$ neurons acting to estimate the advantage functions $A(s,a)$, $a \in \mathcal{A}$. These are then recombine using the definition above to give an estimate to the Q-values.

Often the size of $V(s)$ and $A(s,a)$ are different orders of magnitude. For some states all actions can be of comparable value if the optimal policy is followed thereafter. So $A(s,a)$ tends to be small while the value function can be large. By applying a dueling architecture, the problem of finding $A(s,a)$ and the problem of finding $V(s)$ are separated, which is good for ensuring stability during training. Further $V(s)$ is updated in every update in a dueling architecture while only one $Q(s,a)$ is updated in the traditional DQN setting.

Double Q-learning [104, 105]: Q-learning and Deep Q-learning tend to over estimate Q-values due to the maximisation step over the estimated state-action values. Double Q-learning is a way of rectifying this and instead underestimating Q-values. This is more desirable, since it promotes higher degree of exploration than in the case in which the values are overestimated. In addition, overestimates tend to be large while underestimates tend to be more controlled. The original 2010 version did this by maintaining two Q-values and applying two sets of weights $\theta^{(A)}$, $\theta^{(B)}$ which are updated as follows:

$$\theta^{(A)} \leftarrow \theta^{(A)} + \alpha \Big(r + Q_{\theta'^{(A)}}(s_{t+1}, a_{t+1}^{(B)}) - Q_{\theta^{(A)}}(s_t, a_t) \Big) \nabla_\theta Q_{\theta^{(A)}}(s_t, a_t) \tag{2.32}$$

$$\theta^{(B)} \leftarrow \theta^{(B)} + \alpha \Big(r + Q_{\theta'^{(B)}}(s_{t+1}, a_{t+1}^{(A)}) - Q_{\theta^{(B)}}(s_t, a_t) \Big) \nabla_\theta Q_{\theta^{(B)}}(s_t, a_t) \tag{2.33}$$

where:

$$a_t^{(A)} \in \text{argmax}_{a \in \mathcal{A}} Q_{\theta^{(A)}}(s_t, a_t) \qquad \text{and} \qquad a_t^{(B)} \in \text{argmax}_{a \in \mathcal{A}} Q_{\theta^{(B)}}(s_t, a_t)\,. \tag{2.34}$$

The second version from 2016, which is implemented in later chapters, we have a model network parametrised by its weights θ, and a target network θ'. The target network θ' is used to select the action and θ is responsible for state-action value evaluation, with the estimated target being:

$$\hat{y_t} \equiv r + \gamma Q_{\theta'}(s_{t+1}, \mathrm{argmax}_a Q_\theta(s_{t+1}, a)) \tag{2.35}$$

and addressing the problem of overestimation, and in general providing the agents with more stable and reliable learning.

2.2.5.5 Policy Gradients and Actor-Critic Algorithms

DQN and its variants seek to parametrise and then estimate the values of different states and action. We then use these estimate to derive policies. In contrast, policy gradient algorithms estimate policies directly from states without attempting to learn a value structure.

Following the notation previous introduced, $\pi_\theta(a|s)$ is the probability of choosing action a in state s with a neural network defined by its weights vector θ. Let J be a measure of the performance of the agent, defined as the the total accumulated reward over an episode (one execution of the control problem):

$$J = \sum_a [Q(s_0, a)\pi(a|s)] \tag{2.36}$$

The update rule will be of the general form

$$\theta \leftarrow \theta + \alpha \nabla J(\theta). \tag{2.37}$$

The next step will be to calculate the gradient of J (for full derivation details, please refer to Sutton and Barto [108]), resulting in:

$$\nabla J(\theta_t) = G_t \cdot \nabla_\theta \ln \pi_\theta(a_t|s_t) \tag{2.38}$$

resulting in an update rule:

$$\theta \leftarrow \theta + \alpha(G_t \cdot \nabla_\theta \ln \pi_\theta(a_t|s_t)). \tag{2.39}$$

Actor Critic (AC) [102] methods arise from the combination of update rules as the one just described with the specific state-action value knowledge from Q-Learning. AC is composed of two different sub-agents operating in unison. The first one learns the policy and is known as the Actor. The second learns the Q-Values of state-action pairs and is known as the Critic. This results in a modified update rule to our neural network weights vector:

$$\theta \leftarrow \theta + \alpha(Q(s_t, a_t) \cdot \nabla_\theta \ln \pi_\theta(a_t|s_t)) \tag{2.40}$$

Advantage Actor Critic (A2C): A2C is a synchronous evolution of the Asynchronous Advantage Actor Critic (A3C) introduced in Mnih et al. [63]. The main difference between them is that A3C uses multiple independent agents, each using their own weights and operating on their own copies of the environment in parallel. A2C does not use multiple agents. The improvements allow the agent to compute the *advantage* of selecting a specific action $a \in \mathcal{A}$. To this end, the Q-Value in the previous update rule can be substituted with the advantage of an action as defined previously in Eq. 2.31.

This would typically require two different approximators, one for $Q(s, a)$ and another for $V(s)$, but taking into account their relation as per the Bellman optimality equation:

$$Q(s_t, a_t) = \mathbb{E}[r_{t+1} + \gamma V(s_{t+1})] \tag{2.41}$$

Then it can results in the advantage:

$$A(s_t, a_t) + r_{t+1} + \gamma V(s_{t+1}) - V(s_t), \tag{2.42}$$

which only requires one estimator for $V(s)$, but needs to be evaluated from the simulation via Monte Carlo policy evaluation:

$$V(s_t) \approx \sum_t r(s_t, a_t). \tag{2.43}$$

From here, it follows that, in the case of A2C, the gradient that will be finally required for weight updates is:

$$\nabla J(\theta) \approx \sum_t \nabla \ln \pi_\theta(a_t|s_t) A(s_t, a_t) \tag{2.44}$$

CHAPTER 3

Estimating Traffic Profiles in the Strategic Road Network

3.1 Introduction

The UK collects and processes data from its Strategic Road Network (SRN) in real time and makes it available through the National Traffic Information Service (NTIS) [14]. NTIS collects speed, flow and travel time data using sensors on the road and in vehicles. Parts of this information come from the MIDAS system, system samples from the distribution of speeds over time. The basic building blocks of the SRN are called links - segments of motorway between 500 and 20000 metres in length. NTIS data is used to assign a travel time profile to each link. Profiles should represent the typical time to traverse a link at a given time of day on a given day of the week and are the topic of study of this paper.

Profiles clearly vary with day of the week and time of the day but should be relatively stable from one week to the next. This stability over time means calculating profiles is different from the problem of short-term forecasting. While there is no a-priori mathematical definition of a typical travel time, we take it to mean the value that minimises the mean absolute relative error (MARE) with respect to subsequently measured travel times. NTIS publishes profile values alongside measured travel times although the methodology used to calculate them is proprietary, and hence not in the public domain, so it will not be discussed in this piece of research. In this paper, we present a novel method for generating profiles for a complete week. Our approach is based on statistical analysis of previous data and does not require any a-priori segmentation of days into classes. Rather patterns of intra- and inter-day variability are learned directly from the data.

The available literature on travel times is extensive but most recent research focuses on short-term forecasting and with fewer studies looking into the long term

[138, 139]. From a methodological point of view statistical methods and machine learning take most of the attention [140]. Within the last group, neural networks are getting a level of relevance [141, 142] . Others take closer approaches to this paper: using historical data [143, 144], differentiating between peak and non-peak [145], performing spectral analysis [146] or Locally Weighted Regressions [147, 148, 149, 150, 151, 152]. Comparisons between some of these studies and others can be found in Mori et al. [22], Nikovski et al. [153], Van Hinsbergen et al. [154] and Lana et al. [155]. Most of these methods have been specifically tuned for the conditions on which they have been developed and transferability to other sites is often not evaluated. To assess the transferability of our method, it is tested on three different motorways. The method here presented, uses a combination of spectral analysis, tree decisions and Locally Weighted Regression (LWR).

3.2 Examples of travel times

The most direct variable for measuring the state of traffic over a length of road is the vehicles' travel times. The instantaneous travel time for a given segment of road is the average time that the vehicles currently in it are taking since they enter the segment until they exit it.

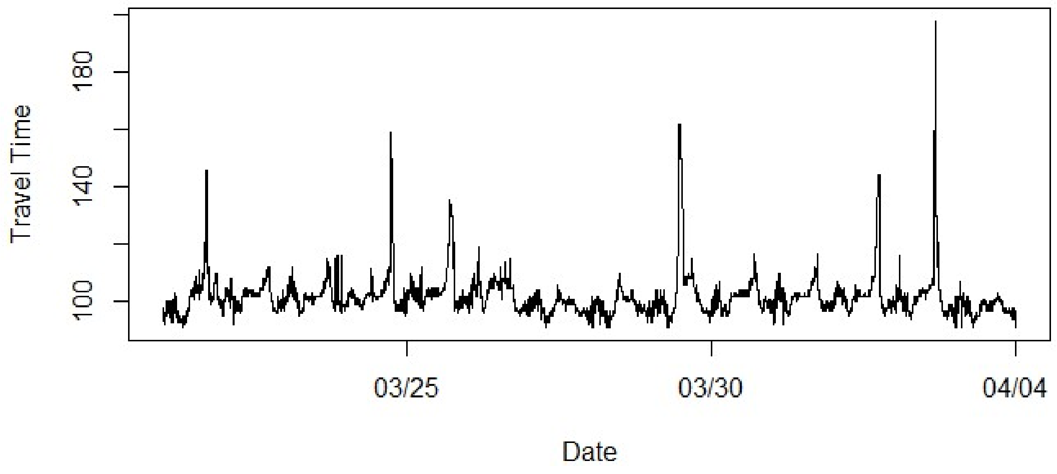


Figure 3.1: Travel times on link 117007401 in the M6 over three weeks

As it can be seen in Figure 3.1 the travel time remains within a vaguely predictable pattern most of the days, with bounded minima during nights corresponding with the free flow time and with some outliers below this value corresponding to speeding drivers. Travel time will meaningfully rise as people leave to work and add load to the motorways. This collective behaviour will create the morning traffic jams, which in our data, are partially replicated during the evening rush hour, normally finding a plateau in between. After this, the travel times slowly decay towards the night period of free-flow regime. In Figure 3.1, it can also be seen that there are a series of excursions out of this oscillating yet bounded typical behaviour. In these, the travel time can increase several times fold the usual values. These extreme oscillations are much less predictable both in intensity and inter-oscillation period than the recurrent congestion described previously.

3.3 Basic methods for profile estimation

3.3.1 Use of Exponentially Weighted Moving Average (EWMA)

The most basic approach to profile estimation is to apply an EWMA on the same minute of every day, with the implicit assumption that similar behaviour is expected at the same time of the day. In this approach, the estimated profile $\hat{x}(i, d+1)$ on the $i - th$ minute of a given day d, for a memory parameter $\alpha \in [0, 1]$ and with measured travel time x_i^d, will be:

$$\hat{x}_i^{d+1} = \alpha * x_i^d + (1 - \alpha) * \hat{x}_i^d \tag{3.1}$$

The main problem with EWMA is the manner in which the memory decays. Recent measurements are weighted more heavily than events in the past. Thus if an extreme fluctuation occurs, the following profile estimates will be biased, partially replicating this event and over-estimating the travel times until enough new measurements arrive to dissipate this effect.

3.3.2 Segmentation

In addition, to acknowledge the specific differences between some special dates in the year, and the difference between days of the week, this family of methods requires the use of heavy date based segmentation. The EWMA will be applied across dates which fall in the same category (i.e. Mondays, weekends, Christmas Day, ...). If this is combined with the shortcomings presented in Section 3.3.1, some long reaching effects are generated, which can propagate for weeks into the future predictions, but do not have any reflection on the observed travel times. A possible instance of this effect can be seen in Figure 3.2. Furthermore, in order to generate a valid segmentation, an experienced team is necessary, since the needs of the process can geographically vary, given that the EWMA approach tends to better approximate

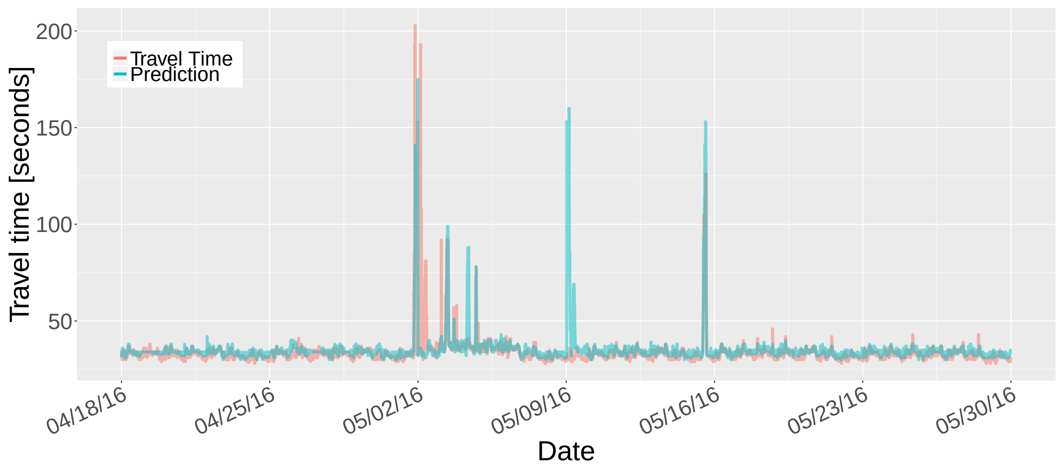


Figure 3.2: A single large random spike in travel time can lead to over-estimation of subsequent profile estimates

endemic congestion. This dependence can lead to the creation of legacy systems which may not be well understood after a few years, decreasing their usability over time unless extra effort is put into transmitting this knowledge and continually train new staff.

3.4 Data

3.4.1 Data Gathering and Selection

Data was gathered from 3 different Motorways in England. The M6 and M11 were selected due to their high usage and combination of recurrent and non-recurrent congestion. The M25 was selected on the base that it is the most used Motorway in England on a daily basis, suffering from chronic congestion.

- The dataset for M6 comprises 90 days (12 complete weeks) of data (07/03/2016-05/06/2016) across 14 links.
- The dataset for d M11 comprises 90 days (12 complete weeks) of data (07/03/2016-05/06/2016) across 25 links.
- The dataset for M25 comprises 75 days (10 complete weeks) of data (07/04/2017-20/06/2017) across 61 links.
- Links were discarded if they had more than 10% of missing data, if they contained any large incidents logged (e.g. accidents) or if they contained any entry or exit ramps.

- Whenever missing data was detected for 10 or less minutes, it was linearly interpolated.
- Whenever missing data was detected for over 10 minutes, it was left as missing values.

The only occasion where data imputation has taken place is in the previously mentioned periods of up to 10 minutes. Given their extremely small scale when compared with the prediction horizon (10080 minutes), resulting in lengths of a maximum of 0.1% of said horizon, the effects of imputation of missing data were considered very minimal.

In the case of the M6 and the M11 the first 8 weeks are used to predict one complete week ahead. One week later, the process is repeated, deleting the oldest week of training data and incorporating measurements of the week predicted in the previous step. This is performed 4 times. In the case of the M25, 6 weeks of data were used for training and the process as described above is performed 3 times.

3.4.2 Data Contents

For each link on a specific date, the required input data consists of one entry per minute, containing:

- Measured travel time
- Profile (expected) travel time

3.5 Travel Time Prediction Algorithm

Given the cyclic nature of traffic, the aim was a prediction algorithm that could account for the periodic variations and endemic congestion while being resilient to fluctuations and rare events. This algorithm also should:

- Be robust, mitigating the propagation of isolated events into future forecasts, unlike methods using EWMA.
- Not require the use of time segmentation and be valid for regular and "special" dates.
- Be location agnostic, the internal parameters should be set algorithmically based on the data.
- Have Gaussian, mean 0, uncorrelated residuals.

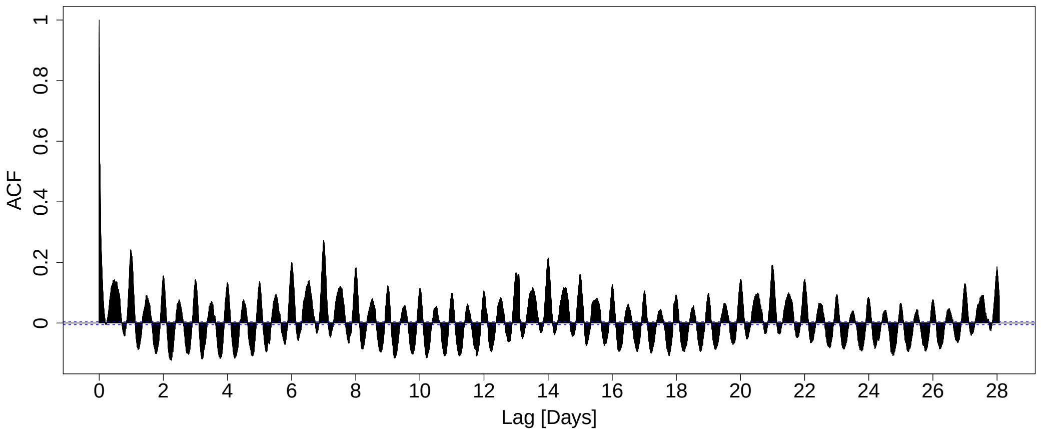


Figure 3.3: Autocorrelation function of a link in the M6 over a period of 4 weeks, showing seasonal patterns on the daily and weekly periods.

- Near flat Trend term, given the different time scales between seasonal cycles and changes in the general motorway flow.

While there is a clear spatial and temporal dependency in the data (cars advancing through successive links over time) that could be exploited with certain types of models and analysis; after several conversations with our partner company, Thales UK, it was decided to steer away from such approaches, opting instead for an approach based exclusively on local per-link data aiming to obtain location-agnostic methodologies for long-term estimation, as the current method is, and capable of potentially being a successor to said method based on EWMA. Incidentally, this also has a positive effect for the availability of data, since such spatial-temporal modelling require of upstream data to calculate. Given the low quality of the real-world data available and how many links had to be discarded during the initial data analysis, taking such an approach would have meant that links downstream from those discarded, up to the limit of the spatial dependency inbuilt in such a model, would have needed to be discarded too, thus reducing the overall amount of available data.

Regarding the stationarity of the time-series, often it is assumed that processes including seasonality are non-stationary. However, in the case of stable seasonal patters, they can fit within the class of cyclostationary series. Cyclostationary series are non-stationary in the sense that their statistical properties will change over time, however it can be expected for these properties to be the same within the same section of the seasonal pattern across different periods of said pattern. These processes are characterised by a periodic mean and autocorrelation functions [156], fitting with the time series at hand based on the information shown in Figs. 3.1 and 3.3. Regarding its suitability for analysis given its characteristics, for tools such as ARIMA, it is possible to make the series stationary by taking differences as required, and for other tools such as STL no such considerations are needed, since the model is meant to be used with this or even more complex data including heavily changing trends.

3.5.1 Naive Segmentation

To obtain an accurate comparison of the performance of the algorithm developed in the following subsections, an example of basic segmentation was coded. This involved a weighted combination of the training data points using uniform weights. In this way, for the $i-th$ minute of a week and using a training set composed of the previous of n weeks, the Naive Segmentation (NS) profile is:

$$\hat{x}(i,n) = \sum_{\text{week}=1}^{n} \frac{x_n^i}{n} \tag{3.2}$$

3.5.2 Double Seasonal ARIMA

AutoRegresive Integrated Moving Average (ARIMA) [24] models are a generalisation of ARMA [157] models which have been extremely successful for forecasting time series in a wide range of fields and applications. Further extensions to ARIMA include the ability to model series including seasonal terms, resulting in Seasonal ARIMA (SARIMA) models.

To obtain reliable comparisons with the profile calculation methodology presented in the following section, a SARIMA model using two seasonal terms was fit to the travel time data and used for forecasting:

$$\text{model} = \text{SARIMA}(P_1, D_1, Q_1)_1 \text{ x } (P_2, D_2, Q_2)_{1440} \text{ x } (P_3, D_3, Q_3)_{10080}. \tag{3.3}$$

Here, $(P, D, Q)_i$ will refer to the P number of lag observations in the model, the number of times D that the source data was differenced, and the size Q of the moving average of the model for a seasonal length of i. Hence, the part using the sub-index 1440 will deal with daily seasonality, and the part using the sub-index 10080 will deal with the weekly seasonality.

Most common implementations of SARIMA in R and their related tools are however not suitable for this type of modelling. Firstly, "ARIMA models don't work well for very long time series", and in these cases non-parametric models can be more suitable [158]. Secondly, standard SARIMA implementations in R, such as those in the "Stats" or "Forecast" packages "will allow a seasonal period up to $m = 350$ (lags) but in practice will usually run out of memory whenever the seasonal period is more than about 200" [159] [160] [161]. Thirdly, said packages do not have support for multiple seasonal terms [160] [161] [162] (R. Hyndman, personal communication, March 24, 2021), which are necessary in this case based on what the seasonal periods observed in Fig. 3.3, with the daily seasonality covering 1440 lags and the weekly seasonality covering 10080 lags respectively. Based on the three points above and after expert consultations (R. Hyndman, personal communication, March 24, 2021), the most adequate tool was deemed to be the "msarima" model class from the R package "smooth" [163], although its user manual does not recommend usage for series having extremely long seasonal periods in terms of lags [164] such as the series currently at hand.

Standard modern-day compatible methods for parameter estimation in the model [165] based on information criteria proved ineffective due to excessive memory consumption, forcing computing systems crashes due to the excessive length of the time series and the elevated number of ARIMA parameters over which the algorithm must optimise. An additional challenge was found regarding computation times, since computation times for a single SARIMA model using 4 weeks of travel time data to forecast a week ahead, as required to be in close conditions to those of the other profiles here presented, takes approximately 2 hours and 8 minutes to calculate on a current-day desktop computer, and approximately 1.5 GB of memory to digitally store.

Based on these limitations that will hamper any optimisation procedure, an alternative strategy was devised, in which a sequential approach was taken to estimate the 9 parameters indicated in Eq. 3.3. An initial selection of the data was performed, and a sub-series covering the morning plateau of a typical Tuesday with no congestion events was isolated from the original series of one link of the M6 displaying both recurrent and non-recurrent congestion. In this manner, it is possible to obtain a sub-series that is much shorter than our seasonal terms, to attempt to approximate the behaviour of the time series in absence of any seasonality. Following this, the Sventunkov-Boylan algorithm [165] was applied, obtaining the best fit for an ARIMA$(0, 1, 4)$ based on lowest Bayesian Information Criterion (BIC), which also happened to obtain the lowest Mean Absolute Percentage Error. Next, a sub-series covering 3 days (Tuesday, Wednesday, Thursday), was isolated and the process was repeated, this time using a SARIMA model with a daily seasonality term. After an optimisation process within a reduced parameter

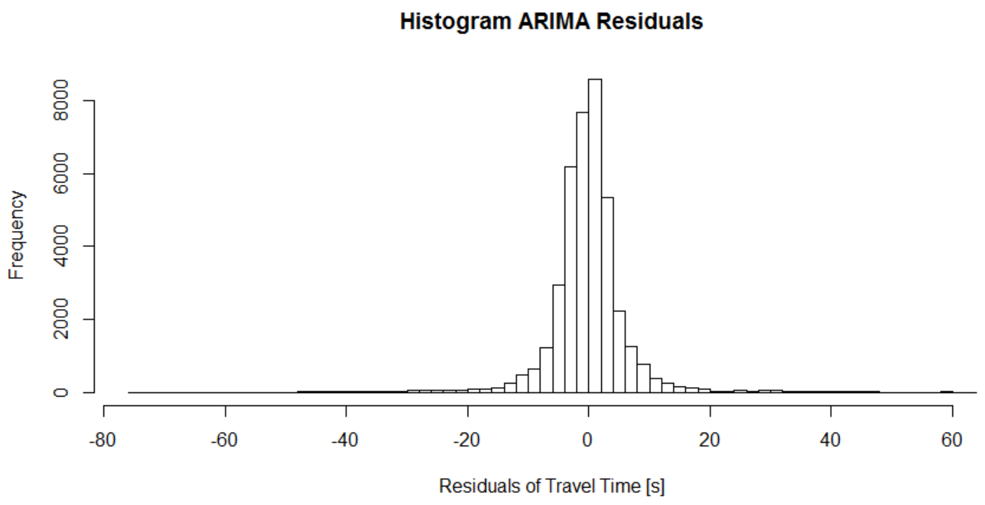


Figure 3.4: Histogram of the residuals from SARIMA model across prediction weeks.

search space using the same algorithm, the best fit to the data was achieved for a SARIMA$(0, 1, 4)_1$ x $(0, 0, 3)_{1440}$ based on the same criteria. It's important here to remark that while in absence of seasonality the Sventunkov-Boylan algorithm estimates 3 parameters, during the second step of the search all 6 parameters were jointly estimated, obtaining the same values as in the first case as the best fit for the non seasonal part of the model. While this does not provide a guarantee that said estimated parameter values will be the best choice for a model including both seasonal terms, it gives an indication that these values can provide a reasonable approximation to the underlying process. Lastly, the parameters for the non-seasonal and daily seasonal components were fixed, and the Sventunkov-Boylan algorithm was used to estimate the parameters for weekly seasonality modelling, obtaining a SARIMA$(0, 1, 4)_1$ x $(0, 0, 3)_{1440}$ x $(0, 1, 0)_{10080}$ model. The overall process required of 2 differences terms, one for non-seasonal and another for weekly seasonality in order to make the series stationary.

The histogram for the residuals of the full SARIMA model can be found in Fig. 3.4, and the autocorrelation function (ACF) can be observed in Fig. 3.5. The distribution of residuals was compared with Normal and Laplace distributions via Kolmogorov-Smirnov test, where in the one sample case the test statistic D quantifies a distance between the empirical distribution and the cumulative distribution function of the reference distribution, with lower values representing closer resemblance. The test for Normal distributions obtained the closest match with a $N(\mu = 0, \sigma = 4.2)$ (statistic $D = 0.02993$), and for Laplace distributions with

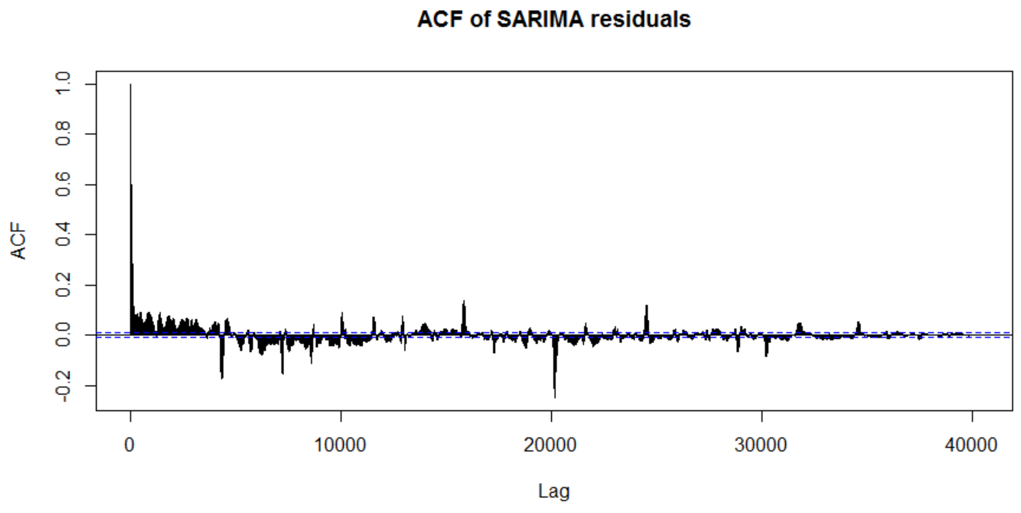


Figure 3.5: Autocorrelation of the residuals from SARIMA model across prediction weeks.

a Laplace($\mu = 0, \sigma = 4$) (statistic $D = 0.04904$), leading to a Normal distribution being shown to be a better approximation. While the residuals are close to normally distributed, the ACF of the residuals shows that while there is no pattern consistently emerging over time, a Light positive spike in week 1 and negative spike in week 2 (10080 and 20160 lags) point at potential seasonality information being left in the data, although the correlations are weak. Based on the information presented on Section 3.6.3, regarding computational cost and general performance, in addition to those presented earlier in this section regarding the adjustments to the standard process needed to estimate the parameters of the model, it was decided against refitting the model, and its improvement and generalisation to entire motorways is left as future work.

3.5.3 Decomposition in Background and Spikes

During exploratory data analysis we found that, from the point of view of travel times, traffic operates in two differentiated regimes that we denominated Background and Spikes.

- Background:
  - Stable around a mean value.
  - Oscillates with small amplitude and high frequency.
  - Suitable for spectral filtering.

- Spikes:
  - Zero most of the time. Quickly go to extreme values.
  - Oscillates with great amplitude and low inter-oscillation frequency, creating far reaching effects.
  - Suitable for seasonal decomposition.

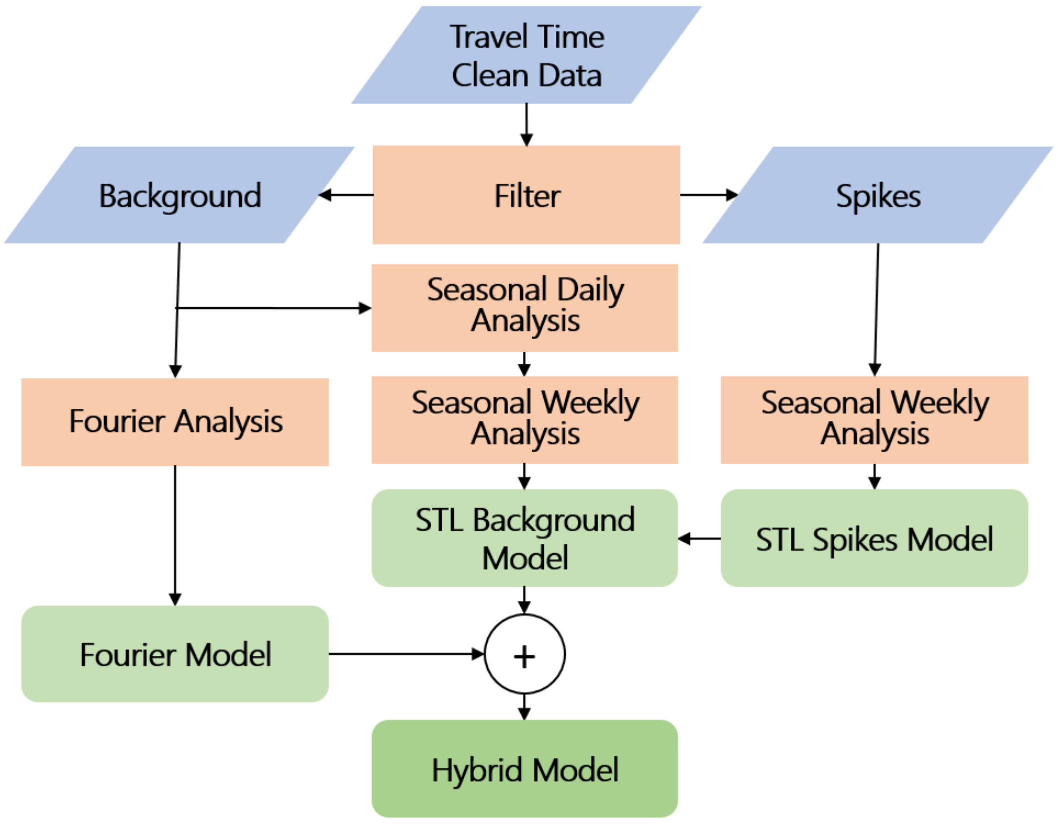


Figure 3.6: Schematic of data streams in the algorithm.

In this context, assuming Gaussian noise ξ, and that the components operate in an additive way, generated a signal of the form:

$$\text{Travel Time}_t = \text{Background}_t + \text{Spikes}_t + \xi \tag{3.4}$$

The objective was to separate them in such a way that the moments of smooth and normal operation are captured as part of the Background and used for spectral analysis, attempting to mitigate the prediction error induced by the high frequency oscillations and obtaining a basic view of what can be daily observed. Meanwhile, the spikes, including the recurring and non-recurring congestion, were treated separately searching for seasonality in larger time scales than those in which the travel time signal oscillates, as suggested by Fig. 3.3. Ideally, after this final step of the decomposition, the remainder should only contain isolated large rare events deviating from the profile and white noise.

To prevent the differing lengths of the links from affecting this decomposition, for this step, all travel times were normalized according to their corresponding link's free flow time, defined as the time to transverse the length of the link at the maximum legal speed allowed by the motorway. However, this step only mitigates the non-regularity of the time series, since there are drivers who do not follow these limits. A threshold was heuristically set, as seen in Table 3.1, to separate the two components in the different regimes from the normalised travel times on a per-motorway basis by using a 1 level decision tree as introduced in Section 2.1.4.1, although better results can be obtained by setting each link individually. Intuitively, this threshold scales with the amount of recurring congestion in a link. Whenever a data point was above the threshold, it was flagged as belonging to a spike.

Table 3.1: Travel Time Normalisation Thresholds

Motorway	**Threshold**
M6	1.1
M11	1.2
M25	1.4

For this purpose, an indicator function was defined, taking for every minute the value:

$$\delta_t^{spike} = \begin{cases} 1, & \text{if } x_t > \text{Threshold} \\ 0, & \text{otherwise} \end{cases} \tag{3.5}$$

3.5.4 Spectral Component

The main difficulty when dealing with the Background signal is the low amplitude, high frequency fluctuations that can be found almost ubiquitously. Signal smoothing can be performed by completely removing a range of frequencies while the information bearing bands are retained. For this task, the Fast Fourier Transform (FFT) [29] introduced in Section 2.1.4.3 was used as follows:

1. Calculate Background Power Spectrum using FFT

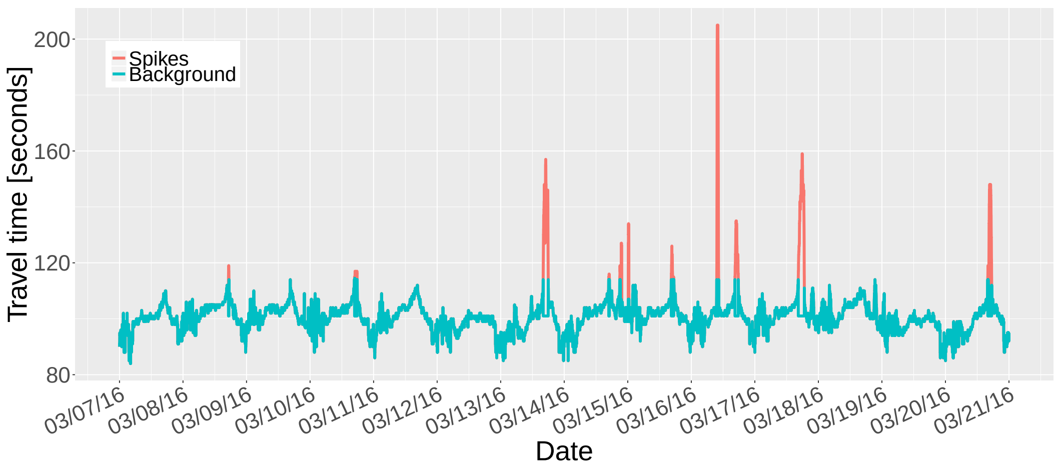


Figure 3.7: Decomposition time series in background and spikes

2. Remove frequencies corresponding to periods under 4 hours and over 1 week
3. Repeat for all n weeks in training set
4. Apply EWMA to the modified weekly Power Spectra
5. Compute the Inverse Transform

3.5.5 Seasonality Component

Seasonal-Trend Decomposition based on LOESS (STL) [28], as previously introduced in Section 2.1.4.2, was the chosen algorithm for the seasonality analysis since it can handle any type of seasonality, allowing the user to control how it changes over time as well as the smoothness of the trend-cycle while being robust to outliers [25].

Below, the sequence of steps taken to extract the Seasonality Components that can be observed in Fig. 3.3 is described in line with the schematic data streams represented in Fig. 3.6. Note that this should be applied to the n training weeks as a single time series.

1. Decomposition of Background for daily seasonality
2. Extract and sum the series corresponding Trend and Remainder from Step 1
3. Decomposition of output from 2 for weekly seasonality
4. Add daily and weekly Seasonal components from Step 1 and Step 3 to obtain global seasonality

5. Average seasonality across training weeks

6. Linearise Trend term from output of Step 3

7. Add linearised trend to seasonality obtained in Step 6

8. STL Decomposition of Spikes for weekly seasonality

9. Extract Spike's Seasonality corresponding to the number of weeks for forecasting

10. Add Spike's Seasonal component to the output of 8

To ensure a successful decomposition, after Step 3 Background's Remainder should ideally be zero mean, Gaussian distributed, although this is not a requirement of STL. After checking the distribution of the remainders for Link 1 of the M6, it was found that their mean equals 0.8 seconds, and that upon examination under a two-sided Kolmogorov-Smirnov test, it results in a p-value=2.2E-16, rejecting the hypothesis that the residuals are normally distributed. However, the data being used comes from the real world, with the model directly attempting to filter out large deviations, which is then reflected in the distribution of the residuals in the form of a heavy tail. This development will not necessarily subtract from the predictive power of the model, since the filtering of these large events precludes the existence of the mentioned heavy tail, and the actual distribution of residuals from the finished profile is characterised, as it will be seen in following sections, by uncorrelated residuals with close to zero mean.

Background's Trend should have a near zero slope. These outcomes should also hold if performed again after Step 8, with the addition that the trend should also be close to zero in absolute value, since the time scales in which such a global trend can meaningfully vary should be much greater than the prediction scope of this algorithm.

The decomposition performed by STL as defined above is additive. In order to check for potential suitability of a multiplicative approach to the decomposition, an alternative decomposition was performed after applying a logarithmic transformation to the travel time series. An alternative seasonal series is created this way, which will be referred to using the suffix "Log". The only difference between the two profiles is the previously mentioned logarithmic transformation to background and spikes series prior to the seasonal-trend decomposition and transformations in the frequency space, paired with an inverse transformation prior to the recombination step, creating the new HybridLog profile as described in the next section.

3.5.6 Seasonal-Spectral Hybrid Profiles

In order to create the final Seasonal-Spectral Hybrid profile (Hybrid), one of the two forecasts generated in the previous points is taken, depending on what is the identified regime:

$$\text{Hybrid} = \text{Seasonal} * \delta_{\text{spike}} + \text{Spectral} * (1 - \delta_{\text{spike}}) \tag{3.6}$$

Similarly, the Seasonal-Spectral HybridLog profile (HybridLog), will be:

$$\text{HybridLog} = \text{SeasonalLog} * \delta_{\text{spike}} + \text{SpectralLog} * (1 - \delta_{\text{spike}}) \tag{3.7}$$

3.6 Accuracy Results

In this section the accuracy of the algorithm described above is compared against the Published Profiles and the NS Model. For each temporal point i, the Mean Average Relative Error (MARE) is defined below:

$$\text{MARE} = \frac{\left(\sum_{i=1}^{n} \frac{\|x_i - \hat{x_i}\|}{x_i}\right)}{n} \tag{3.8}$$

The MARE is the same as the Mean Absolute Percentage Error (MAPE), but instead of reporting it as a percentage, it is reported as a fraction of a unit. Both of them are measures of prediction accuracy when producing forecasts in time series. In order to calculate it, it is necessary to average the deviation from the individual forecasted points with respect to the actual measurement. For complete Motorways, the Root Mean Squared Error (RMSE) has been calculated for each temporal point i as:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \hat{x_i})^2}{n}} \tag{3.9}$$

In both cases the error for a Link is defined as the average MARE or RMSE

Table 3.2: Hybrid Profile, MARE Distribution Per Motorway

% rel. error	<-25%	-25% **to** -15%	-15% **to** -5%	-5% **to** 5%	5% **to** 15%	15 % **to** 25%	>25%
M6	1.58	0.60	3.77	88.01	5.97	0.06	0.01
M11	0.80	0.35	4.07	86.15	7.97	0.49	0.15
M25	3.85	2.73	10.42	75.12	7.29	0.33	0.28

across all prediction points. The error for a Motorway is defined as the average of the error across all its links.

3.6.1 Accuracy by Quantile

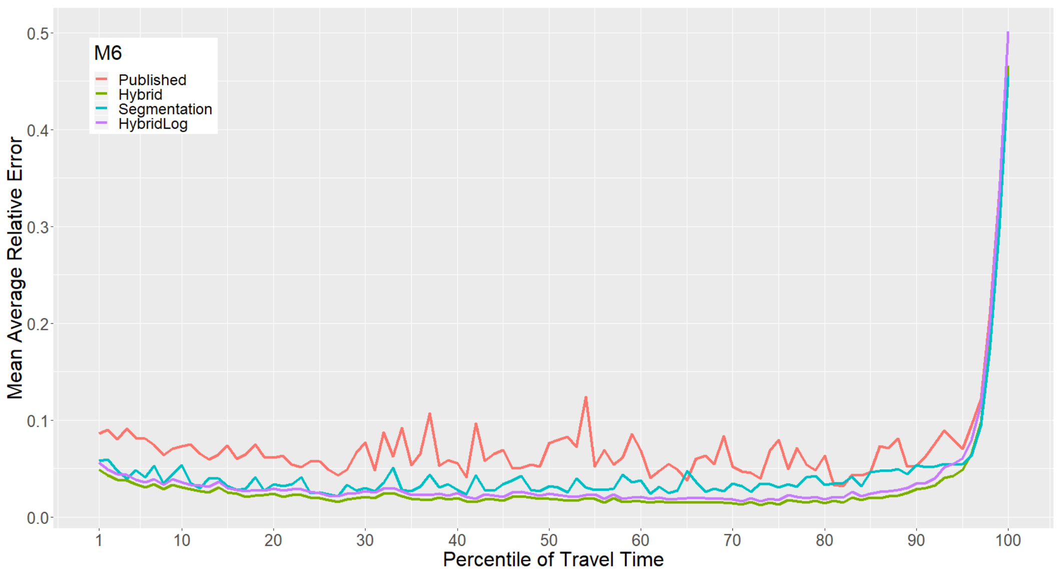


Figure 3.8: Average accuracy results in M6 across percentiles of travel time.

Here, the accuracy of the algorithm is compared against the Published Profiles, HybridLog and the NS Profile across all percentiles of travel time, with the goal of assessing the prediction quality as the actual measurements of the predicted quantities increase. For this goal, Figs. 3.8-3.10, present the same results as shown in Figs. 3.11-3.13, but instead of grouping the measurements by the time of the day in which they are taken, 100 bins are generated, and travel times are placed in the bins according to a quantile function $Q : f(\mathcal{R} \rightarrow [0, 1])$. Then, for the bin corresponding to each quantile q of travel time, containing n travel time measurements, the MARE will be calculated as:

$$\text{MARE}_q = \frac{\left(\sum_{i=1}^{n} \frac{\|x_i^q - \hat{x_i^q}\|}{x_i^q} \right)}{n}. \tag{3.10}$$

As it can be seen in Figs. 3.8, 3.9 and 3.10, the Hybrid Profile has a higher accuracy than the Published Profiles, HybridLog Profile and the NS Profile for all percentiles of travel time except for the most extreme values where they all perform poorly.

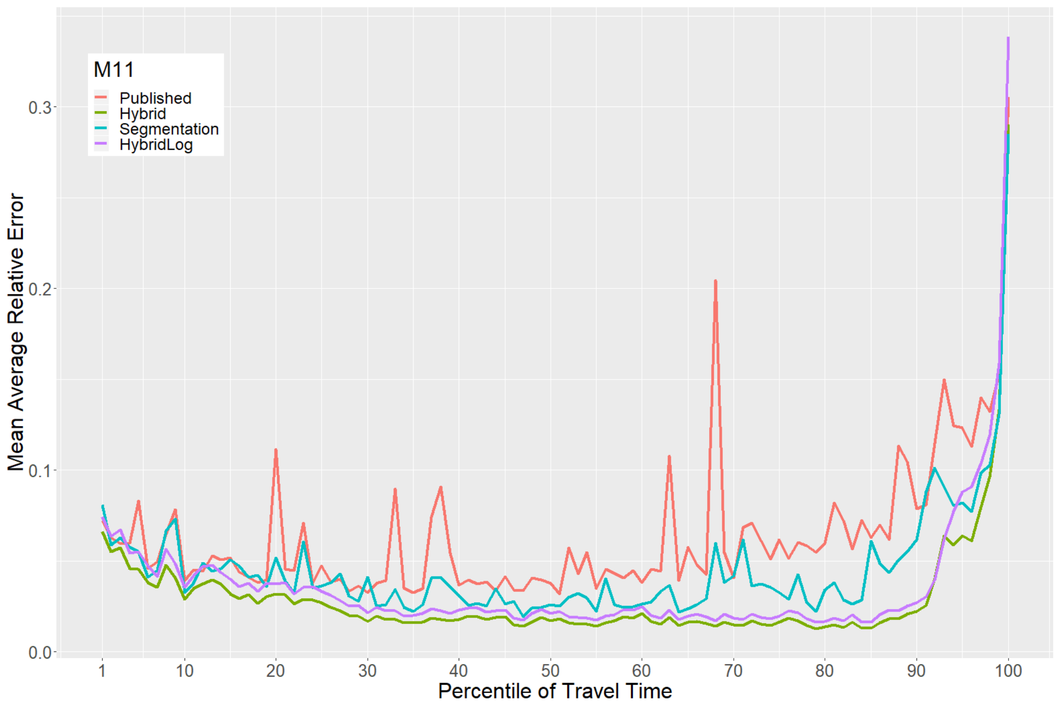


Figure 3.9: Link average accuracy results in M11 across percentiles of travel time.

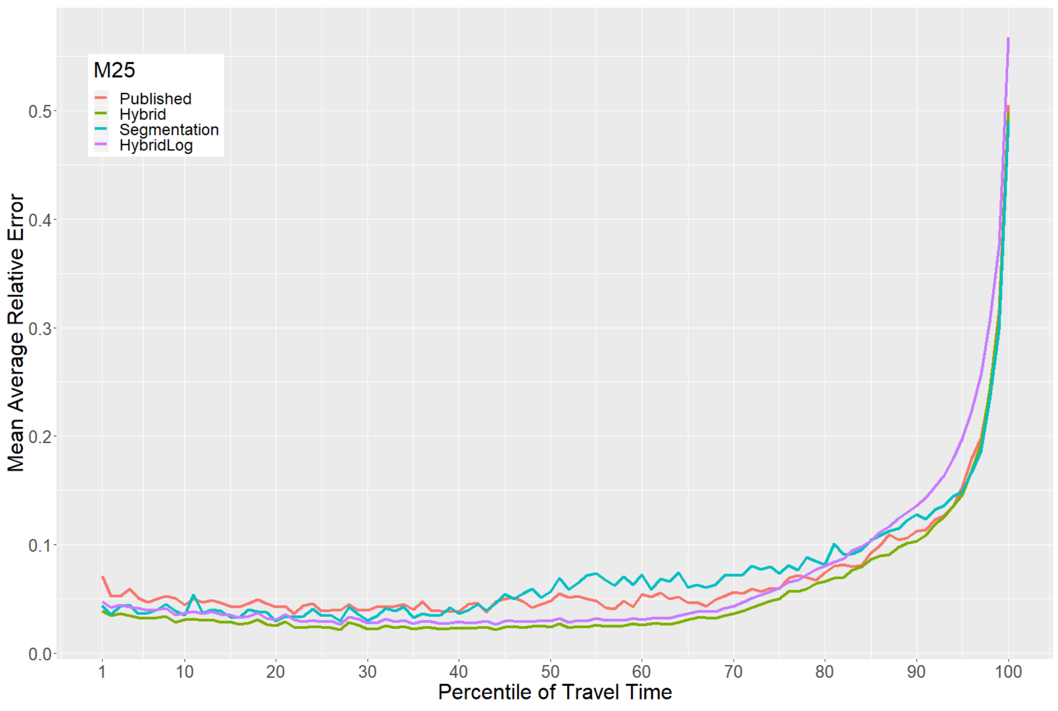


Figure 3.10: Link average accuracy results in M25 across percentiles of travel time.

The most meaningful difference occurs between percentiles [50 − 95], where the Published Profile starts to suffer from higher inaccuracy.

3.6.2 Daytime Error

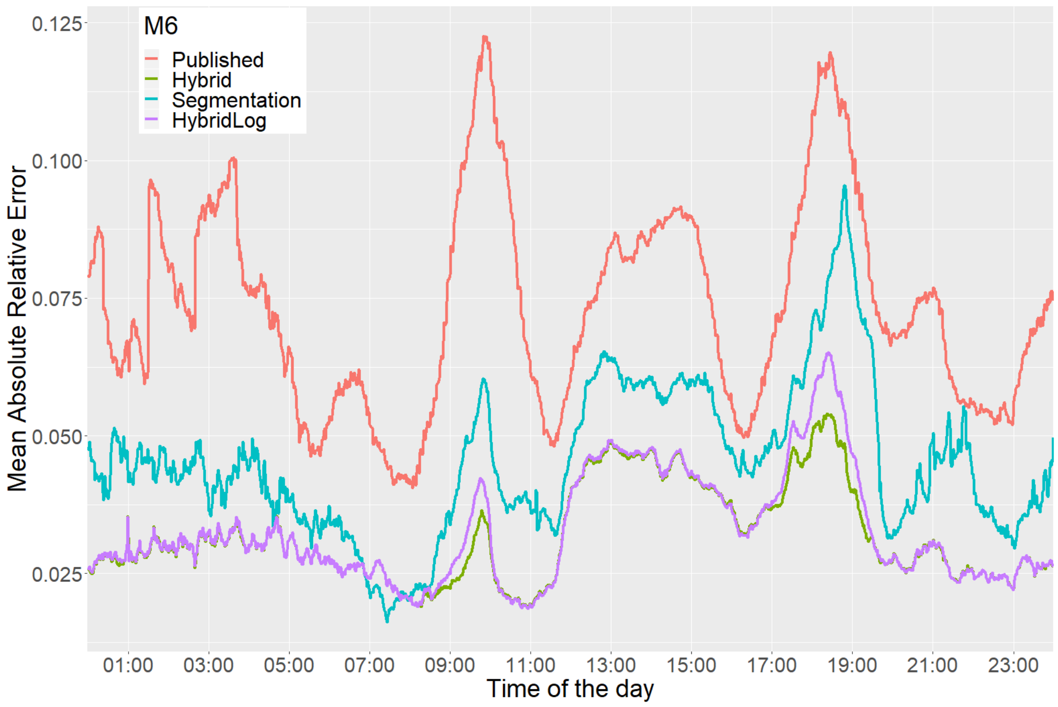


Figure 3.11: Link average accuracy results in M6 across times of the day.

Here, the accuracy of the algorithm is compared against the Published Profiles, HybridLog Profile and the NS Profile across the times of the day. As it can be seen in Figures 3.11, 3.12 and 3.13, the Hybrid Profile clearly displays a higher accuracy than the Published Profiles and the NS Profile for all times of the day, for all locations and training lengths, except for brief windows at 7 AM and 11PM (this last one only in the case of M11) in where the NS Profile performs slightly better, and a very brief period at 8 AM in the M25 where all three profiles obtain similar performance. The HybridLog Profile displays a similar performance when compared to the Hybrid Profile in the case of the M6 and M11, its predictions suffer higher errors during the morning and evening peaks, where both models differ. While the performance of the HybridLog profile is slightly superior than that of the Hybrid profile for a period of a few minutes in the M11 around 6:30PM and 7:30PM, for all other locations and times the Hybrid profile scores equal or better. In the case of the M25, the performance of the HybridLog Profile is found to be worse, performing

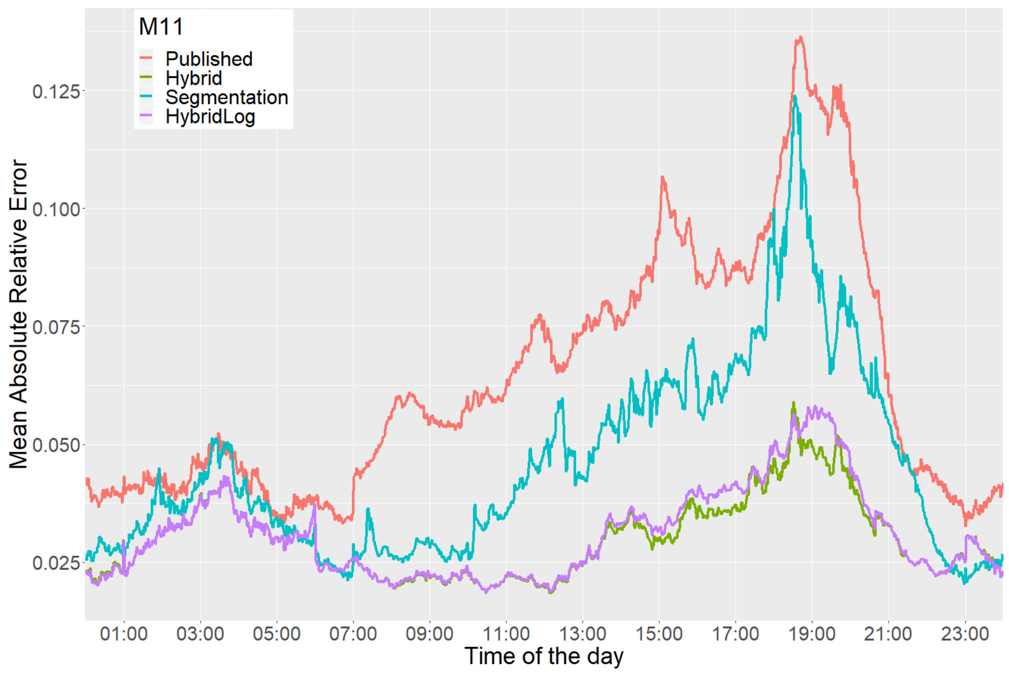


Figure 3.12: Link average accuracy results in M11 across times of the day.

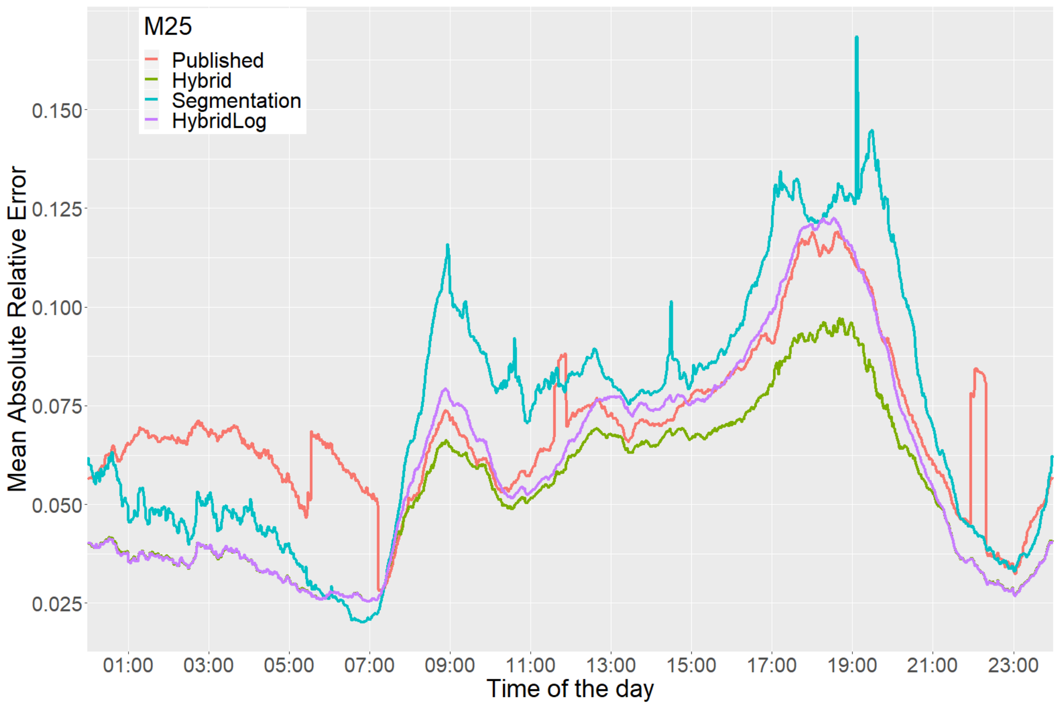


Figure 3.13: Link average accuracy results in M25 across times of the day.

in a comparable level to the Published Profile.

The greatest improvement in prediction, with respect to the Published Profile, occurs during the morning and evening peak hours, where the Hybrid Profile presented in this paper does not suffer a meaningful worsening in its performance relative to the morning plateau when compared with the other two profiles.
In the case of the M6 and M11, where the training set is richer, the error at peak times is reduced by at least 50% in all cases, reaching as much as 68.7% in the case of the M6 morning rush. In the case of the M25, which is congested on a regular basis, the errors in the Published Profile during peak times are slightly lower than on the other cases, indicating that, given the use of EWMA for its calculation, recurrent congestion is better captured by it. Even in this case, the proposed algorithm performs significantly better than any of the other two, except for a brief window between 6-7AM when it is outperformed by the NS Model.

3.6.3 SARIMA Results

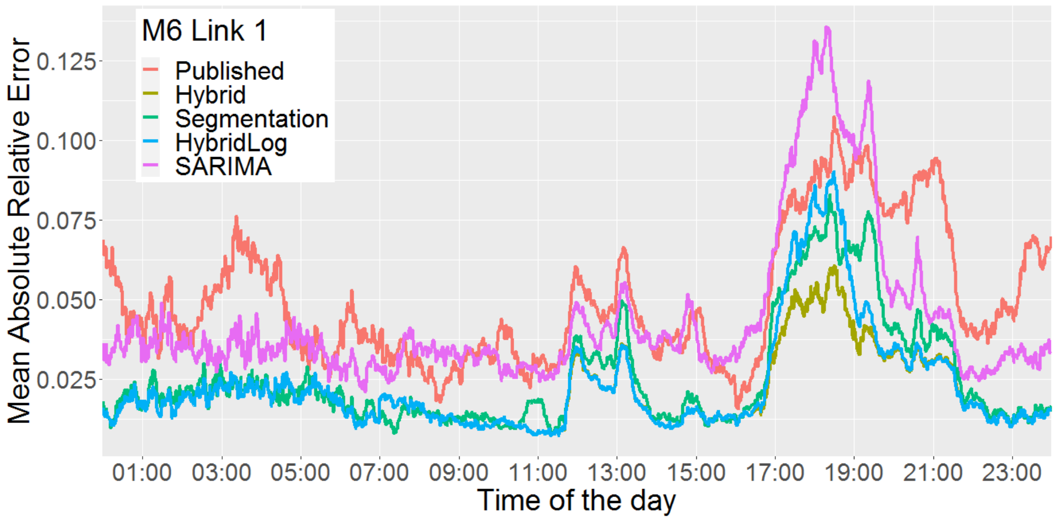


Figure 3.14: Average forecast accuracy across times of the day in Link 1 of the M6

In Fig. 3.14, it can be observed that while the accuracy of the SARIMA profile when calculated for link 1 of the M6, is superior to that of the Published Profile for a section of the day, it falls short of the accuracy provided by the other reference profile, Naive Segmentation, as well as the Hybrid and HybridLog profiles introduced in this paper for all times of the day.

Furthermore, the computational time required to calculate the SARIMA profile is between 3 and 4 orders of magnitude greater than that used to calculate the other profiles. Assuming a constant calculation speed of 2.12 hours of wall-time per week of forecast (8.48 hours per link), as previously obtained experimentally, and taking into account the number of links for which data is available (14 for the M6, 25 for the M11 and 61 for the M25, for a total of 100), as well as the fact that 4 rolling forecasts are produced (for a total of 400 forecasting iterations), it is estimated that approximately 35.40 days of CPU time are required to perform equivalent forecasts to those of the Hybrid profile when using SARIMA. While this is far from ideal, once it was additionally taken into account that the corresponding time to calculate said Hybrid forecasts has been measured at 9.16 minutes of CPU time (approximately 5.5 seconds per link), and given the lower (albeit still acceptable) performance of the SARIMA profile, it seemed reasonable to discontinue its calculation for other links in order to prevent assigning an extreme amount of the limited available computing resources to an under-performing method and focus on the development of those models with increased predictive power and lower computation times.

3.6.4 Hybrid Profile Residuals

Figures 3.15-3.18 show the expected and measured travel times for Links 1 and 3 in the M6, as well as the residuals of the Hybrid profile over the evaluation horizon. These residuals also include those measurements in moments of non-recurring congestion that the algorithm does not try to estimate or forecast, aiming to leave them out. Hence, a long tail of such events can be observed in their distribution.

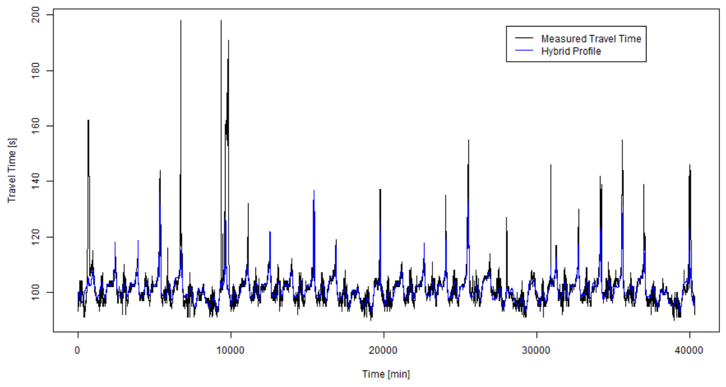


Figure 3.15: Observed Travel Times and Hybrid Profile Prediction for M6 Link 1.

Usually, in time series analysis the general aim is to have such residuals being as close as possible to normally distributed. However, in this case due to the initial aims, as stated in previous sections, of being resilient to the extreme non-

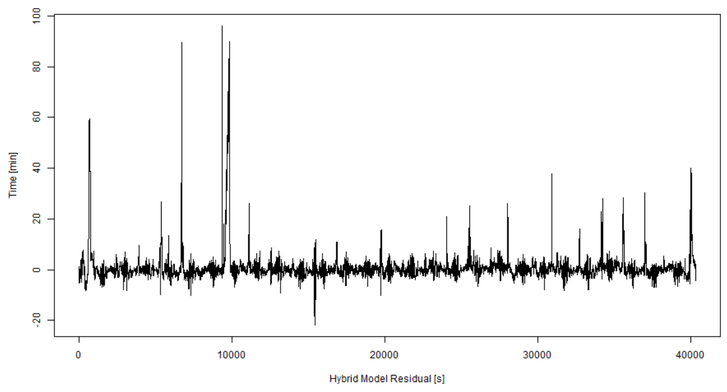


Figure 3.16: Residuals of Hybrid Profile over time for M6 Link 1.

linearities associated with the often more extreme non-recurring congestion events, as well as the fact that the extreme length of the series and its density, as it was seen during the SARIMA approximation attempt, led to higher weight being placed in the empirical performance of the model, both in terms of its accuracy, and the residuals being close to zero and stable over time. As it can be observed in Figs. 3.15-3.18, the residuals oscillate around zero, with little to no drift over the course of each estimated week for Links 1 and 3 of the M6. Furthermore, when looking at the paired Travel Times plots, it can be seen that excursions far from zero occur whenever there is a large ongoing event associated with non-recurring congestion that the model was purposely built so it was filtered out. These links were chosen based on each of them providing a good representation of links showing high and low levels of recurring and non-recurring congestion respectively.

3.6.5 Model Noise Assumptions

The analysis of the residuals of the SARIMA reference profile, shown in Sec. 3.5.2, hints at the existence of a more complex noise source in the data than purely normally distributed.

In the initial problem formulation in Eq.3.4 a single noise source was assumed. In some items of previous traffic research, travel time noise has been estimated using linear mixtures of normal distributions [166, 167] with varying numbers of components depending on the assumptions of the underlying model being used.

Since the model here proposed assumes two different traffic regimes, it is reasonable to wonder if each regime will operate under its own noise distribution. In order to assess this assumption, an alternative formulation for the model is proposed,

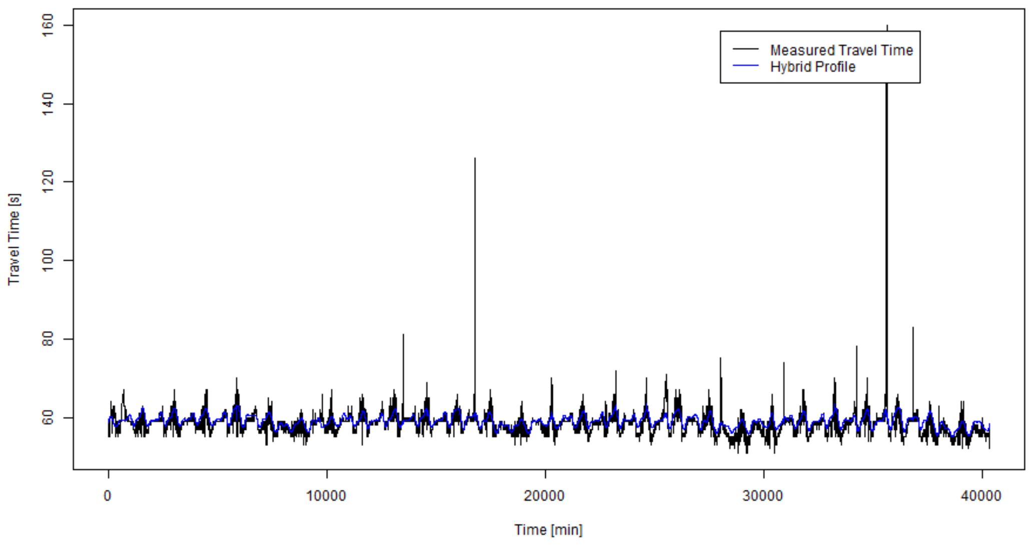


Figure 3.17: Observed Travel Times and Hybrid Profile Prediction for M6 Link 3.

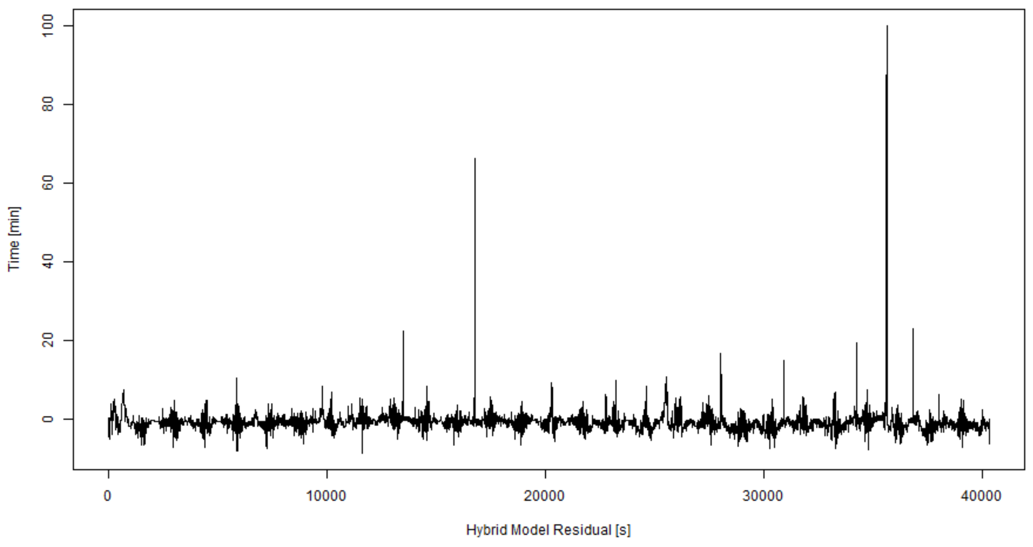


Figure 3.18: Residuals of Hybrid Profile over time for M6 Link 3.

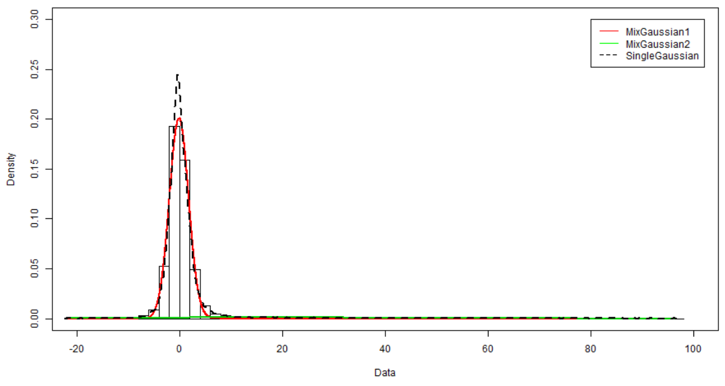


Figure 3.19: Mixed Distribution Estimation of residuals in M6 Link 1

in which background and spikes have their own sources of noise, resulting on an overall noise that will be observed as a linear mixture of Gaussian distributions:

$$\text{Travel Time}_t = \text{Background}_t + \text{Spikes}_t + \xi_{\text{Background}} + \xi_{\text{Spikes}} \tag{3.11}$$

To examine this hypothesis, the residuals of the Hybrid profile in the M6 were analysed, and the noise components were estimated by obtaining a feasible parametrisation via Expectation Maximisation. In Figs.3.15-3.22 we can observe: the mixture distribution estimation for residuals, empirical distribution of residuals showing the logarithm of the densities for the theoretical and empirical distributions for Links 1 and 3 of the M6. These figures associated with these two are representative examples of the two classes we can find in the remaining links of the motorway, in terms of the amount of recurrent congestion they suffer. Among the class suffering more congestion, it was found that the in resulting mixture distribution, the component modelling the noise for spikes had a weight of 4.6%, while the estimated spike prevalence as per the delta function in the Hybrid Profile is 3.3%. While this estimation provides a reasonable fit to the source data, albeit using the second distribution to fit the heavy tail of the distribution, it comes with its own additional set of problems.

The first issue comes from the choice of the number of noise terms, the definition of two distinct source noises has been made a priori, but other number of noise terms could be proposed (e.g. a noise term for the congestion onset and a different one for congestion clearing), in line with different existing assumptions regarding traffic flow that are beyond the scope of this work. In this way, while every addition and parameter estimation for an additional noise source would improve the

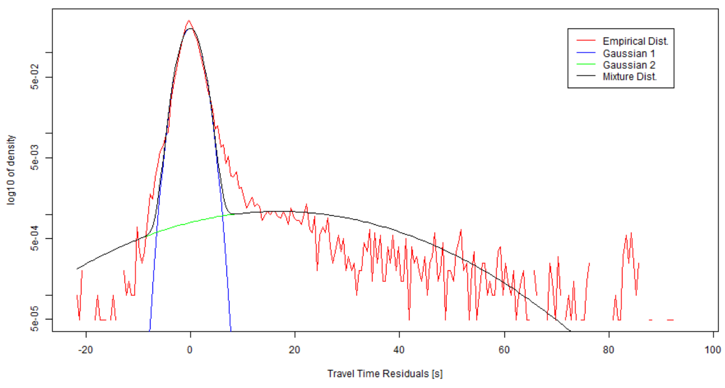


Figure 3.20: Logarithm of density for theoretical and empirical distributions of residuals of travel times for M6 Link1.

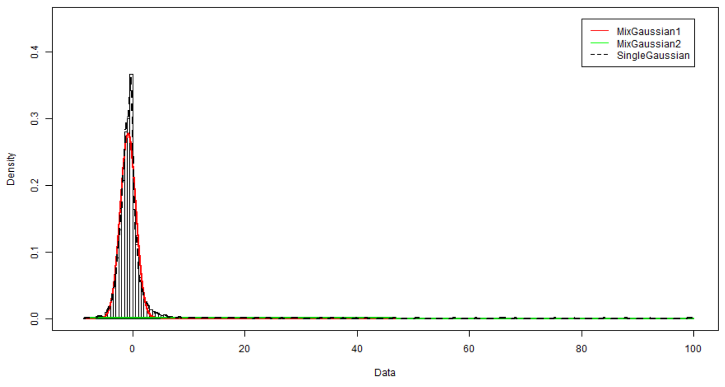


Figure 3.21: Mixed Distribution Estimation of residuals in M6 Link 3

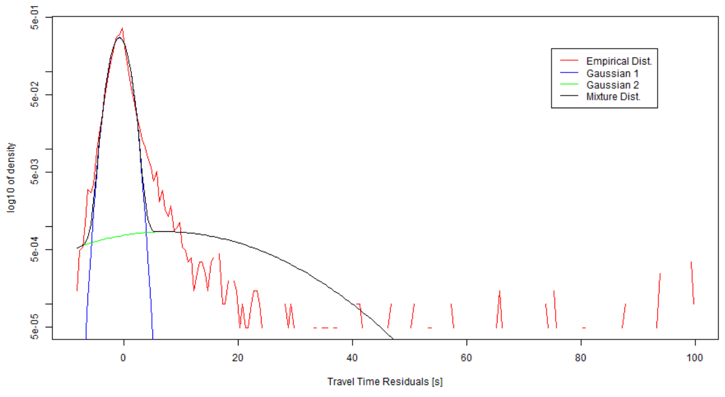


Figure 3.22: Logarithm of density for theoretical and empirical distributions of residuals of travel times for M6 Link3.

fit to the data, it could easily lead to overfitting, especially in this case where the parametrisation is done per-link. In the limit, this would allow the addition of an arbitrary number of components with quasi-zero weight, that while perfectly fitting our source data, would fail to generalise to entire motorways, which is the ultimate goal. This is a relevant concern in this case, since the average weight of the second distribution was estimated at 4% of the mixture distribution.

Second, the Expectation-Maximisation algorithm used [168] for estimation is susceptible to initial conditions, reaching slightly different parametrisations for the model depending on the initialisation, light identifiability issues arise commonly in the form of label switching, although these tend to not pose serious issues when handled with care. However, small variations in the estimated parameters due to random initialisations can lead to the creation of a series of closely related distributions that are equally good for explaining the data, now creating serious identifiability issues.

Furthermore, the residual distributions will change over time, first and on shorter timescales, based on the natural occurrences of non-recurring congestion (which our model does not attempt to approximate, consequently being sent to the residuals) and the noise of the system on a week-to-week basis, and on longer timescales, based on the change over time of global traffic flows. This can lead to further issues regarding the parametrisation of the estimated distribution, even leading to potential changes in the class of a given link.

Lastly, incorporating information about these per-link distributions into the forecasting algorithm would violate the third requirement for the algorithm (as stated in 3.5) regarding the lack of specific per-link parametrisation, while the proposed approach is centred around generalisation for entire motorways.

While a more detailed study of the noise would allow for a better a priori characterisation of the uncertainties in the model, under the current formulation this would have no effect in the forecasts, given that the Hybrid profile estimates the baseline dynamics of the process and deterministically project them forward in time as expected travel times. Based on the issues and considerations covered in this section, it was decided to keep the initial assumption of a single noise source for the next chapter.

3.7 Conclusion and Future Work

This chapter has introduced an algorithm for estimation of baseline travel times in UK highways, capable of accurately forecasting travel times up to 4 weeks in advance. Said algorithm obtains lower forecasting error than the currently Published Profiles and a reference naive approach. The increase in performance is consistent across the three motorways testes, being able to adapt to each of the motorways' dynamics: morning and evening rush hours in the case of the M6 links, evening rush hour in the case of the M11 and high recurrent congestion in the case of the M25, which may explain why this is the only motorway in which the Segmentation Profile is the worst performing one. The reduction in the forecasting error is most noticeable during the previously mentioned rush hours, during which the Hybrid profile produces similar errors to that of the Published Profile during low demand regimes. The algorithm presented above meets the requirements described in Section 3.5 except for the need of an heuristically set threshold.

The implementation of an alternative Hybrid Profile using a multiplicative decomposition (HybridLog) by performing a logarithmic transformation on the travel time data has not shown to improve the performance of the model and hence, will not be used in the following chapter, continuing research with an additive approach to the seasonal-trend decomposition.

One potential way of reaching compliance with these requirements is to perform the decomposition by applying a Wavelet Transform and separating background from spikes based on statistical analysis of the transformed time series in terms of how their Wavelet coefficients fluctuate over time within a scale level. Another potential extension is the performance of a sensitivity analysis explore the limits of the algorithm in terms of minimum training data set (although during testing, acceptable performance was not found under 4 weeks), as well as maximum performance with increased training, for which additional data should be gathered.

CHAPTER 4

Wavelet Augmented Regression Profiling (WARP)

4.1 Introduction

4.1.1 Background

In this chapter, we follow up on and expand upon the work introduced in Chapter 3, presenting more elaborate and suitable solutions to the problematic of handling spikes in motorway travel time series. Chapter 3 introduced an approach to classify data points in travel time series into two differentiated regimes. While the utility and information carried by the background is straightforward, the fact that the spikes contain both recurrent and non-recurrent congestion makes extracting valuable information from them more troublesome. In the previous chapter, the information-bearing subset of the spikes component is extracted in a quasi-black box approach, searching for any seasonality that can be extracted with a seasonal-trend approach, similarly as it is done with the background, but without going into the temporal structure of the spikes.

This chapter aims to dig deeper, defining a clear method to identify what part of the spikes is indeed due to recurrent congestion, based on the variation of the time travel signal across different timescales, using spectral and statistical analysis of recent historical data, while relying on pattern discovery for intra- and inter-day variability which it computes directly from input data.

This is done while still providing an alternative approach for generating one week of expected travel times, using a single parameter and using only publicly available data from NTIS, and also removing the requirement of any prior form of segmentation of times and days into different classes.

4.1.2 Previous Work

There is an extensive literature on travel times, although recent research is more focused on shorter term forecasting, with far fewer long term estimation studies [138, 139]. Machine learning and statistical analysis methods receive the most attention [140]. In machine learning, neural networks are recently attracting a lot of interest [141, 142]. Other approaches are closer to the methods presented here: making use of historical data [143, 144], differentiating between rush and non-rush hour [145], using spectral methods [146] or Locally Weighted Regressions [147, 148, 149, 150, 151, 152]. The Wavelet Transform has been previously found to be useful in combination with Kalman filters [169], neural networks [170, 171, 172, 173] and statistical analysis [174, 175]; but these either cover short predictions, use the Wavelet analysis of travel times for other purposes such as incident detection, or do not focus on the spectral properties across timescales or are not real-world data based. Comparisons involving some of these studies are performed in [22, 153, 154, 155], but they either focus on short-term prediction or do not produce an overall best-performer. The method presented here is distinct from the previous work mentioned, since it does not use a sample of individual trips, but all conducted over 12 weeks in multiple sites. The aggregation period is 5 minutes, the prediction resolution is 1 minute (since the aggregation step is rolling), and the prediction horizon is larger. Among these methods, the transferability to other locations is not often evaluated, being tuned for a specific location with its own specific conditions. In order to ensure transferability, our method is examined on 39 individual locations across two motorways and independently scored for each site. The work presented here makes use of a combination of Continuous Wavelet Transform [33], tree decisions, spectral analysis, and Locally Weighted Regression (LWR).

4.2 Travel times on Motorways

As introduced in Section 3.2, from a user perspective, vehicle travel times are the most relevant measure of the state of the traffic flow on a road link. The average travel time in a given minute of the day on a given link will be the average time to travel from the entry to the exit loop sensor for all vehicles that passed through.

From Figure 4.1, it can be observed that most of the time, the travel time on a link follows a repeating pattern with minima located at night matching the bounded free flow time (except for speeding drivers). As the morning rush starts, travel time will rise as traffic jams are generated from the collective drivers' behaviour. More effects of these collective dynamics can be observed in the afternoon during the

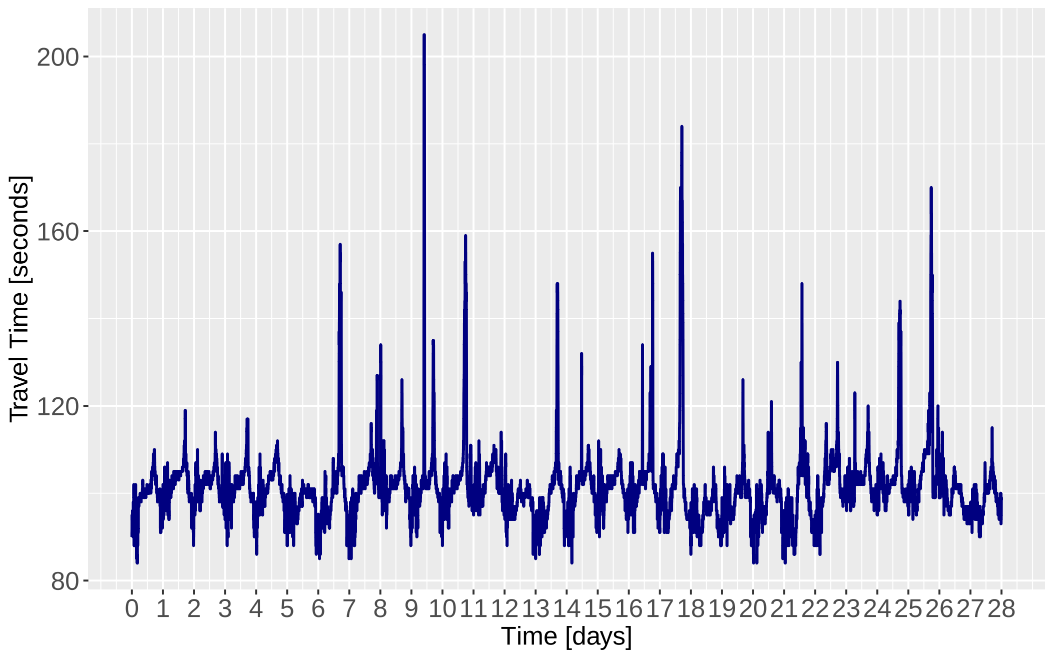


Figure 4.1: Link 1170079, M6. Travel Time over 28 days of minutely data between 07/Mar/2016 and 04/Apr/2016.

evening rush, normally being possible to find a plateau between these two peaks. Finally, travel times progressively decay towards the night's free-flow regime. In Figure 4.1, we observe a series of spikes found outside of this normally bounded yet oscillating behaviour. Travel time in these events can climb up to several times the normal amount. The predictability in terms of duration and amplitude of these spikes is much lower than the periodic component described above.

Lastly, looking back to the Autocorrelation Function of the Travel Time series, which was plotted in Figure 3.3, we can observe a double seasonal pattern with periods of 1 day and 1 week. This regularity seen in time travel time series can and is often used by modellers to approximate and forecast travel times as is explored over the next sections.

4.3 Basic methods for profile estimation

4.3.1 Exponentially Weighted Moving Average for Profiles

As explained in [1] and reviewed in the previous chapter, a basic approach to estimating profiles is to apply an Exponentially Weighted Moving Average (EWMA) across a given minute of the available days, assuming that similar behaviour is to be expected at similar times on different days. Then, the profile estimation $\hat{x}(i, d+1)$ for the i-th minute of a date d, controlling our memory parameter $\alpha \in [0, 1]$ to balance the memory of the process and with measured travel time x_i^d, will be:

$$\begin{aligned} \hat{x}_i^{d+1} &= \alpha x_i^d + (1-\alpha)\hat{x}_i^d \\ &= \alpha x_i^d + \alpha^2 x_i^{d-1} + (1-\alpha-\alpha^2)\hat{x}_i^{d-1} \\ &= \ldots \end{aligned} \tag{4.1}$$

EWMA-based profiles have a main issue: the way in which the memory decays. If a large non-recurrent deviation from the baseline patter occurs, subsequent estimations will have a bias towards partially replicating the deviation and consistently predicting over-estimates until newer data is included and the effect dissipates.

This, when put together with the extra limitations introduced by time-based segmentation, as explained in Section 3.3.2, introduce long lasting perturbations to the series, as it was shown in Figure 3.2.

Consequently, significant operational and geographical expertise about specific roads is needed to create a valid segmentation, given that the EWMA approach is effective exclusively for recurrent congestion. These requirements often lead to the generation of legacy systems which become increasingly difficult to maintain as time passes, their usefulness declining over time or requiring additional efforts for continually training new staff to maintain the system. These modelling and operational limitations may make segmentation and EWMA based profiles suboptimal both in performance and operation.

4.4 Data Selection and Contents

The M6 and M11 motorways in England are chosen due to their high use and their display of both recurrent and unusual congestion, being key in several heavily used commuting routes. The dataset shares the contents of M6 and M11 with that introduced in Chapter 3.

- The dataset aggregates 90 days (12 complete weeks) minutely entries (07/03/2016-05/06/2016).
- Links with over 10% of data missing or containing access ramps were discarded.
- The previous condition left 14 different links in the M6 and 25 links in the case of the M11.
- Entries missing for 10 minutes or less were linearly interpolated.

- Entries missing for over 10 minutes were left as missing values.

The algorithm uses 8 weeks of data to predict one entire week ahead. After a week, the oldest week is deleted, and the most recent one is incorporated, producing a new estimate for the subsequent week. This procedure is simulated 4 times. For each link-date pair, the data comprises minutely data, containing:

- Average vehicle travel time in seconds
- Profile (expected) travel time in seconds
- Traffic Flow in cars/hour
- Vehicle headway in metres

4.5 Background and Spikes

4.5.1 Characteristics

As described in [1], if we look at the travel times, we find they operate in two different regimes that we call background and spikes. The background is stable with small high-frequency fluctuations about a time-varying mean value. This makes it suitable for seasonal analysis and spectral filtering (smoothing). In contrast, the spikes are zero most of the time but can quickly climb to extreme values. They have much greater amplitude and much lower frequency, creating long reaching effects. Although they are non periodic in the time domain, a non-harmonic seasonality contribution associated with recurrent congestion can be extracted via non-parametric regression, as it will be shown shortly.

In this context, if we assume Gaussian noise ξ, and given the additive properties of wavelet decomposition, the decomposition for every minute t of the day, will be of the form:

$$\text{Travel Time}_t = \text{Background}_t + \text{Spikes}_t + \xi \tag{4.2}$$

The objective is to separate the signals such that the times of smooth non-congested operation, together with the recurring congestion, are captured in the background and passed through spectral smoothing, mitigating the estimation errors created by the high frequency oscillations and achieving a view of what can be daily observed, so seasonal patterns can be extracted in the shorter and longer periods shown in Fig. 3.3. Meanwhile, the spikes, containing the non-recurring congestion, can be searched for any seasonality left on time scales larger than the period in which the travel times oscillate, as also suggested by Fig. 3.3. If performed correctly, the remainder after this seasonal extraction step of the decomposition should contain only isolated events with large deviations from the profile and white noise.

4.5.2 Wavelet Time Series Decomposition

To perform the decomposition, we will take advantage of the additive properties of the wavelet transform, as introduced in Section 2.1.4.4. First the time series $\vec{x_t}$ of length t, and elements x_i with $i \in [0, t]$ is turned into a zero mean series $\vec{x_t} - mean(\vec{x_t})$ and used as input for a Continuous Wavelet Transform (CWT) [176, 177] using a Morse wavelet [178] and 140 timescales levels, as introduced in Section 2.1.4.4. The output of this first transform is a complex matrix $\vec{W}$ of dimensions (levels, t), for the elements of which we calculate their modulus ρ, phase ϕ and power P. A heatmap of P^2 for the original time series can be seen in the first subplot in Figure 4.2. In the figure we can observe that the most influential dynamics occurring during the series length (x axis) happen at the timescales (y axis) where it was expected based on Figure 3.3, namely 1 day (1440 minutes) and 1 week (10080 minutes). In the figure, we also observe that surges in power across different wavelet timescales occur at the same time as the non-recurring congestion, since in order to approximate this signal, the CWT algorithm needs to combine several wavelets with smaller periods than those dominating the recurring part of the series.

In order to isolate this non-recurring component, the series is sequentially assessed over all the time domain by taking a horizontal slice for a single timescale level l, and generating a series $\vec{x_t}^l$. After fixing l, we calculate the Median and Inter Quantile Range of $\vec{x_t}^l$ to search for outliers. A maximum threshold value is set equal to $T = median(\vec{x_t}) + \alpha * IQR(\vec{x_t}^l)$, with $\alpha \in [0, \inf)$ being a parameter that defines how aggressively we target spikes. Then, the individual elements of $\vec{x_t}^l$, x_i^l are individually evaluated: if found below the limit, they are stored in a background container $B_i^l = x_i^l$; otherwise, the fraction below the threshold will be nonetheless passed on to the background $\vec{B_i}^l = median(\vec{x_t}^l) + \alpha * IQR(\vec{x_t}^l)$, with the remaining part going into the spikes storage $\vec{S_i}^l = x_i^l - B_i^l$.

In the extremes, for $\alpha = 0$ we would find that any deviation from the median is taken into the spikes signal, and with $\alpha = \infty$ only points infinitely distant from the mean would be taken into the spikes. This parameter could be optimised per-link to obtain the best results, with relatively low calibration effort when starting from a plausible baseline. However a per-link optimisation step would go against the initial goals stated in Sec. 3.5, regarding the algorithm being location agnostic in its parametrisation. For this reason, a heuristic optimisation was performed via

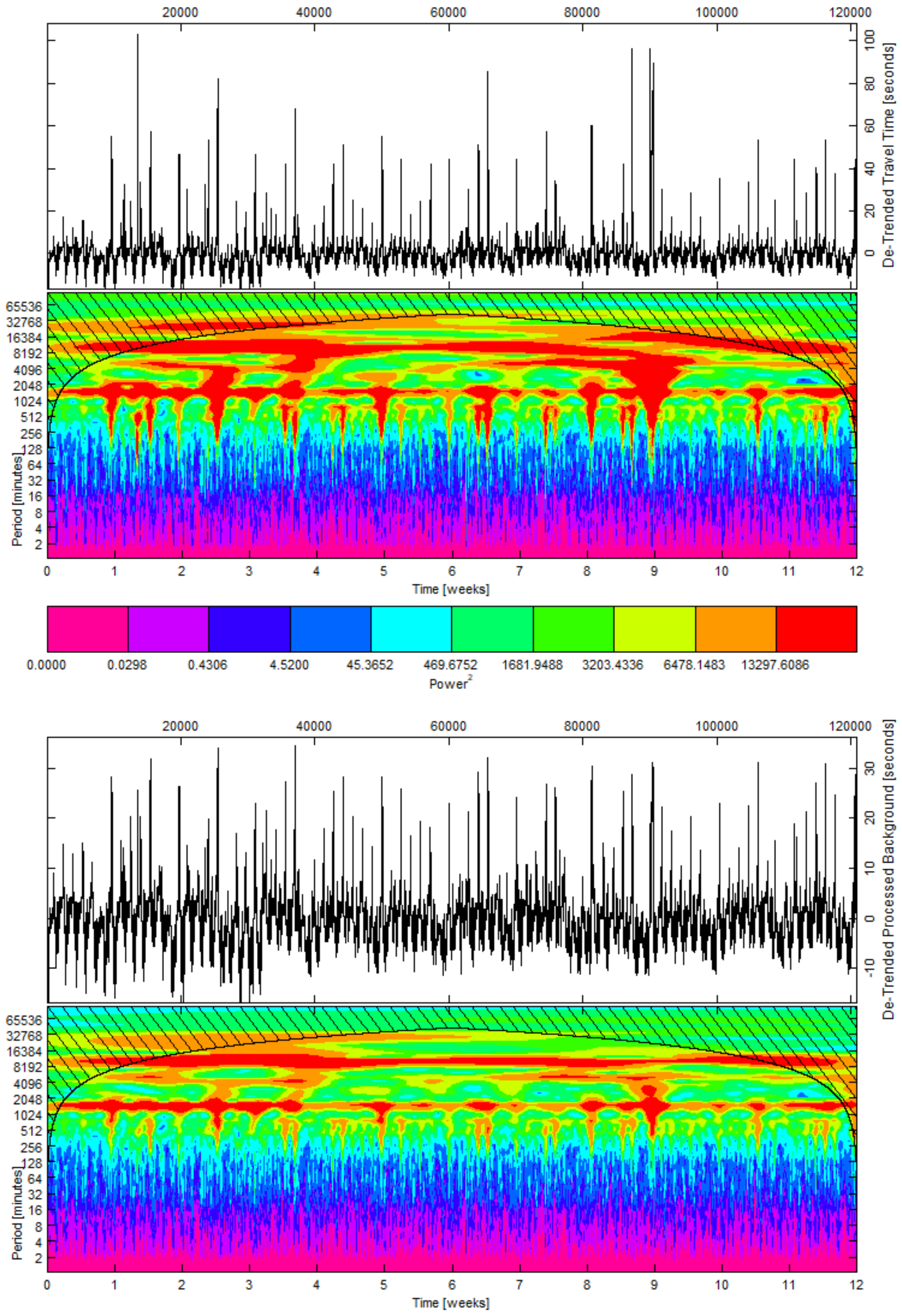


Figure 4.2: Top: Power of CWT of the original Travel Time series for Link 1170079 of the M6. Bottom: Power of CWT of the extracted Background $\vec{B}_t$ series, as detailed in Section 4.5.2, for Link 1170079 of the M6

grid search, aiming to obtain the value of α that minimises forecasting error when evaluating via rolling forecasts for all motorways jointly. Best results were obtained for $\alpha = 1$, hence this will be the value used for the results presented over the following sections.

Once all levels have been processed in this manner, we use the previous information about ϕ to reconvert the two series from being characterised in terms of (ρ, ϕ) to the complex components of the DWT. After this step, we apply the Inverse DWT to $\vec{B}_t$ and $\vec{S}_t$, obtaining the two series that can be observed in Figure 4.3.

4.5.3 Series recombination and analysis of background

The results of the separation of the background can be seen in Fig. 4.2. Here it can be observed that the separated background is still characterised by seasonality dominating the weekly (10080 minutes) and daily (1440 minutes) timescales, and keeping a very similar structure in the regions occupied by the faster dynamics ($<$ 120 minutes) to that of the original series. Simultaneously, the surges in power across timescales of the original DWT-transformed series (left), which are associated with congestion events, have been greatly reduced or eliminated altogether in the second subplot.

This demonstrates the ability to process a time series showing multiple seasonalities and punctuated by large deviations from the normal dynamics of the process into two separate series, one of which will exclusively contain the baseline dynamics of the process, respecting its structure and natural variability, and the other, which will only contain those large events that occur outside of the smooth operational region.

To reduce noise introduction during the inverse transform, a threshold is applied to $\vec{S}_t$, where elements representing spikes less than 3 seconds in amplitude are set to zero when looking at future estimation.

For future prediction steps, an indicator function is defined for every entry in the series, taking value:

$$\delta_i^{spike} = \begin{cases} 1, & \text{if } x_i > \text{Threshold} \\ 0, & \text{otherwise.} \end{cases} \tag{4.3}$$

4.6 WARP Travel Time Prediction Algorithm

The objective now is, given the seasonality and separation of the original signal provided above, to generate a time travel prediction model that accounts for the cyclic

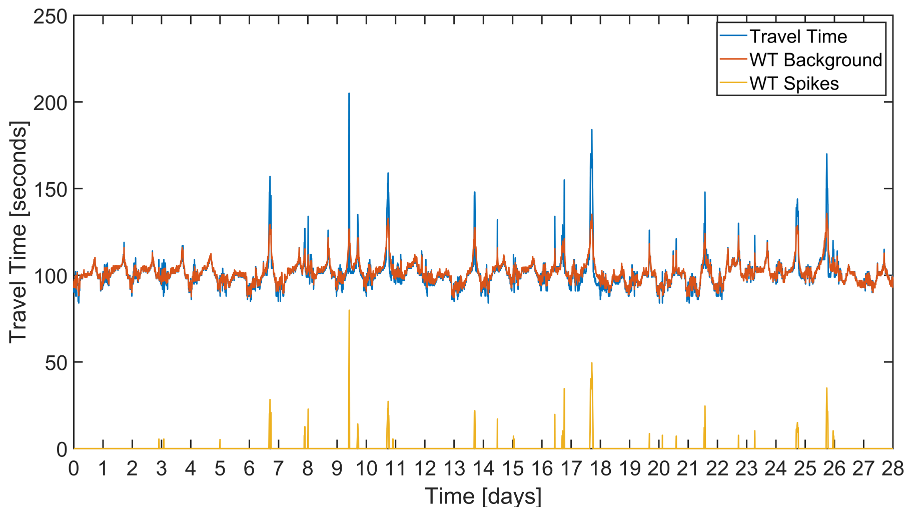


Figure 4.3: Recombination post Inverse Wavelet Transform

variations and recurrent congestion but remains resilient to unexpected deviations and rare events.

We aim to provide a robust algorithm that mitigates the propagation of extreme events into the future (unlike EWMA). It must work for all locations and not require the use of time segmentation. The trend term must be nearly flat, based on the difference in timescales for the growth of demand on a motorway level and the seasonalities concerning this chapter. Finally, it must have uncorrelated residuals, Gaussian distributed with mean 0. Following these requirements, we introduce the WARP algorithm (Wavelet Augmented Regression Profiling) from section 4.6.1.

4.6.1 WARP: Spectral Component

The background signal shows oscillations of high frequency (order of minutes) and low amplitude (usually a few seconds) almost ubiquitously. It can be smoothed by discarding, in the frequency domain, the frequencies that in which the oscillations occur and those outside the scope of this study (over 4 weeks and under 4 hours), while keeping those in the information bearing bands by using the Fast Fourier Transform (FFT) [29]. Once this step is performed, the modified power spectra can be passed through the Inverse FFT Transform and, since large events have been removed previously, now an EWMA can be safely applied to the series in the time-domain in order to obtain the background prediction.

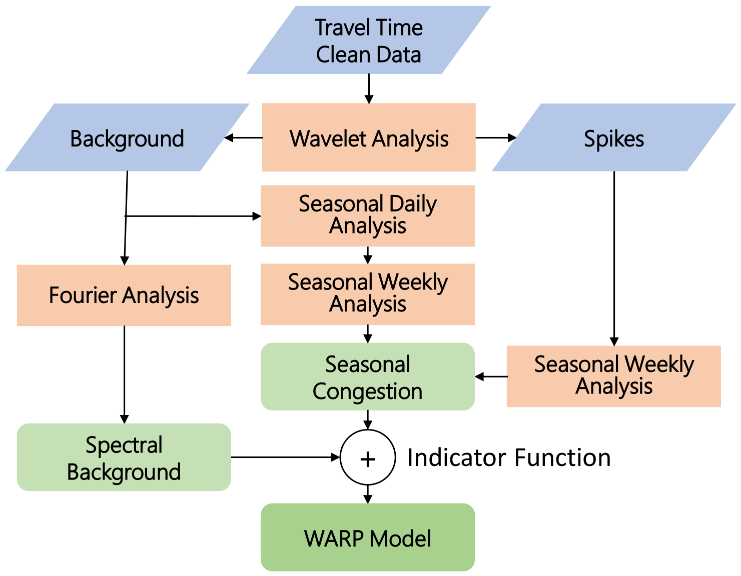


Figure 4.4: Flowchart of the data streams in the algorithm

4.6.2 WARP: Seasonal Component

The seasonal component is computed via Seasonal-Trend Decomposition based on LOESS (STL) [28]. STL is resilient to outliers and can manage any combination of seasonalities, allowing to control their change over time as well as the smoothness of the trend [25]. We begin by isolating the daily seasonality S_d from the entire background training series using STL. Trend and remainder are summed and re-analysed for weekly seasonality S_w. This step also produces a trend series T_r and a remainder which should be Gaussian distributed, zero mean. Global seasonality is calculated as $S_g = S_d + S_w$. We then average the seasonality over the training weeks to obtain a value for each minute of the week we are to estimate. Then, T_r, which is nearly flat after the decomposition, is linearised to obtain a baseline series B_t, and the Baseline prediction is $B_p = B_t + G_s$. Finally, the spikes are searched for any seasonality left on the weekly level Sp_w, discarding the trend and remainder terms, and obtaining the final seasonal component as $SEASONAL = B_t + Sp_w$.

4.6.3 WARP Hybrid Profile

The final WARP profile, containing the spectral and seasonal components, depends on the regime as per Eq. (4.4):

$$\text{WARP} = \text{Seasonal} * \delta_{\text{spike}} + \text{Spectral} * (1 - \delta_{\text{spike}}) \tag{4.4}$$

Table 4.1: MARE Distribution in M6 prediction

Profile / MARE	$(> -25\%]$	$(-25\%, -15\%]$	$(-15\%, -5\%]$	$(-5\%, 5\%)$	$[5\%, 15\%)$	$[15\%, 25\%)$	$[> 25\%)$
Published	1.58	0.54	5.69	56.21	30.79	3.17	2.02
Wavelet	1.57	0.51	3.22	81.73	12.14	0.70	0.13

Table 4.2: MARE Distribution in M11 prediction

Profile / MARE	$(> -25\%]$	$(-25\%, -15\%]$	$(-15\%, -5\%]$	$(-5\%, 5\%)$	$[5\%, 15\%)$	$[15\%, 25\%)$	$[> 25\%)$
Published	0.85	1.15	19.21	62.83	13.21	1.65	1.10
Wavelet	0.78	0.32	3.29	81.33	12.47	1.09	0.73

4.7 WARP model predictive accuracy

In this section we assess the performance of the WARP model in out-of-sample validation and compare it against a null model based on simple segmentation and against the published NTIS profiles (the details of which are not in the public domain.)

4.7.1 Simple Segmentation - null model

Our null model for assessment of the performance of the WARP model is a basic segmentation model that applies uniform weights to the training data in a given time interval from previous weeks. On the i-th minute of a week and using the previous n weeks (here $n = 8$) as training, the simple segmentation (SS) profile is:

$$\hat{x}(i, n) = \sum_{\text{week}=1}^{n} \frac{x_n^i}{n} \tag{4.5}$$

4.7.2 Rolling forecast evaluation

Standard cross-validation procedures applied to Machine Learning (defining test and train datasets) cannot be directly translated to time series. For time series, standard cross-validation is based on forecasts with a rolling origin, using variable amounts of training data, given that the same model is used for prediction at different points in time [25]. The model here proposed differs from most traditional time-series models in that, the way that it achieves variation of the seasonal component and remains responsive to new data is by being recalculated from zero for every forecasting week.

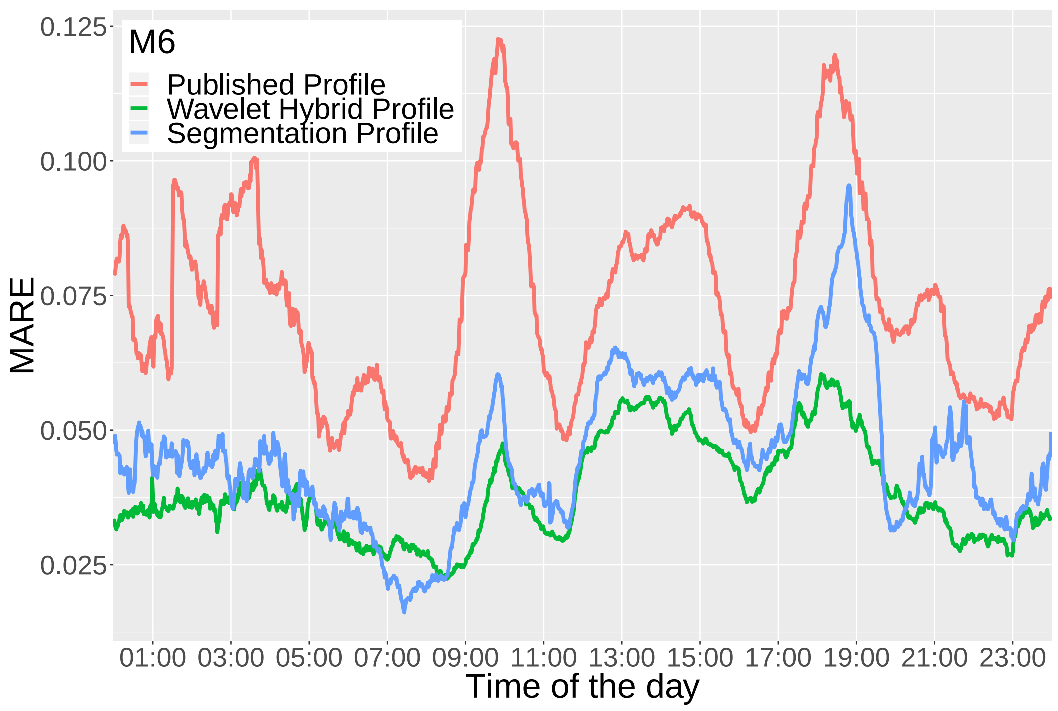


Figure 4.5: MARE over the times of the day for the M6

In this sense, each "predicted" week of data is obtained by using a new iteration of the model, having identical parametrisation to its predecessors but different training data (discarding one week of measurements to incorporate a new one), that generates a deterministic estimation for *baseline travel times* a single time, and this baseline is then projected forward in time as a forecast. The main evaluation criteria for the model is the predictive performance in the very long term, hence, rolling forecast evaluation tests have been performed to compare the WARP profile values against the subsequently measured travel time.

We choose to use the Mean Absolute Relative Error (MARE), as defined in Eq. (3.8) to quantify performance since it allows for a fair comparison of links of different lengths. The Root Mean Square Error (RMSE), as defined in Eq. (3.9) has also been calculated on a motorway level as the average of the RMSE across its links.

As shown in Figures 4.5 and 4.6, and supported by Tables 4.1 and 4.2, WARP shows lower predictive error than the published profiles and the SS Model for all times and motorways. This is most relevant for morning and evening rush hours, where the others' predictive error soars, WARP suffers no meaningful performance worsening relative to the plateau in the middle of the day. The error at rush hours is

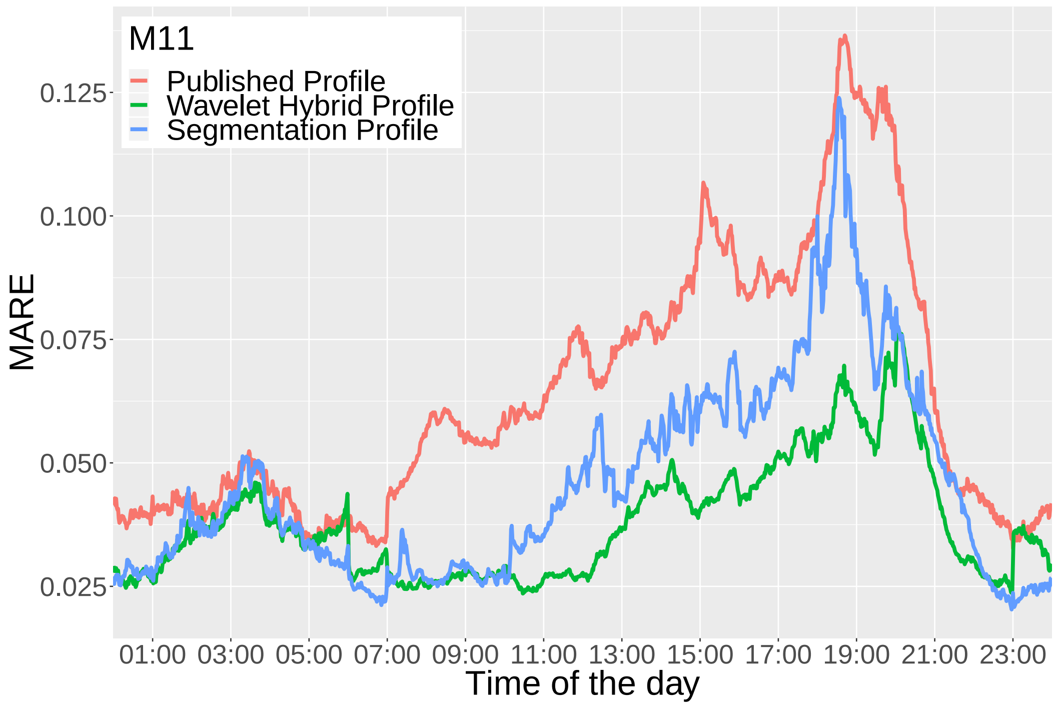


Figure 4.6: MARE over the times of the day for the M11

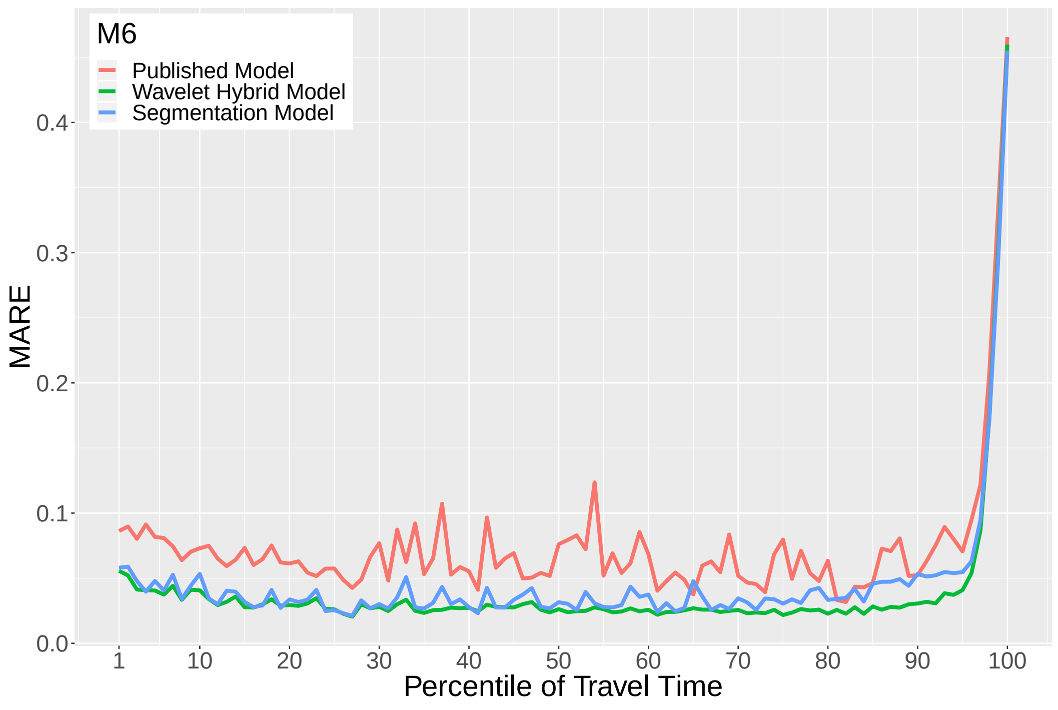


Figure 4.7: MARE across percentiles of travel time for the M6.

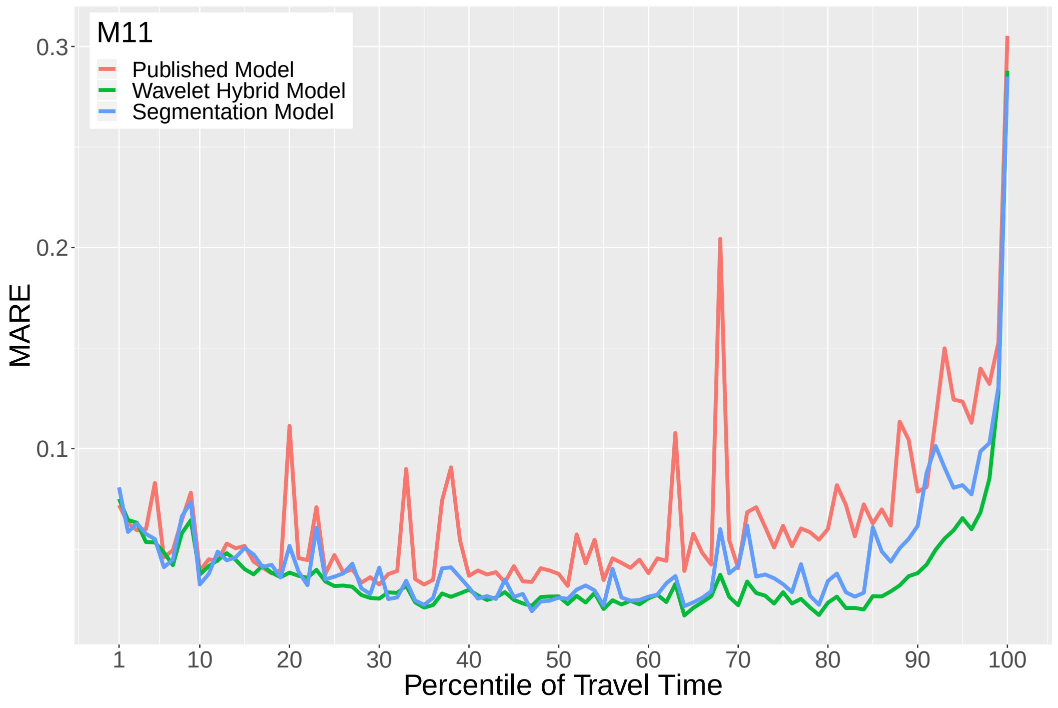


Figure 4.8: MARE across percentiles of travel time for the M11.

reduced by a minimum of 50% across all cases, reaching as high as 63.4% in the case of the M6 morning rush. In Figs. 4.7 and 4.8 it can be observed that the accuracy of the WARP profile is significantly higher than that of the published profiles or the SS Model across nearly all percentiles of travel time. WARP is most competitive in the upper percentiles of the travel time distribution since it explicitly accounts for the predictable contribution of recurrent congestion to travel time spikes. All three models eventually suffer similar errors under the most extreme deviations since these are true outliers and are not amenable to data-driven forecasting.

4.7.3 Conclusion and Future Work

This chapter has tackled the problem uncovered in Chapter 3 of additively splitting a travel time signal showing multiple seasonalities and interspersed with large deviations into a baseline signal, capturing most of its recurrent dynamics, and a spike signal, containing the developments occurring outside the baseline regime. This is done by performing an statistical analysis of the time travel signal after performing a CWT. The separated components can be recombined and processed to generate long term estimations in similar ways of those presented in Chapter 3 to produce estimates of baseline travel times in the motorway and to generate long term fore-

Table 4.3: Average MARE per Link on M6 and M11

Links M6	**MARE**
117007401	0.0290
117007501	0.0680
117007601	0.0293
117007801	0.0826
117007901	0.0435
117008401	0.0484
117009102	0.0338
117011901	0.0295
117012001	0.0584
117012101	0.0370
117012201	0.0605
117012301	0.0379
117016001	0.0496
123025901	0.0427

Links M11	**MARE**
199048301	0.0510
199048701	0.0279
199048801	0.1053
199048901	0.0239
199049002	0.0306
199049101	0.0213
199049402	0.0223
199049501	0.0181
199049702	0.0297
199049801	0.0408
199050002	0.0445
199050101	0.0272
199050202	0.0288
199050901	0.0433
199063301	0.1203
199063701	0.0500
199063801	0.0265
199064203	0.0230
199065202	0.0259
200021668	0.0280
200024801	0.0233
200028639	0.0241
200028641	0.0188
200028645	0.0435
200028648	0.0499

Table 4.4: Average MARE & RMSE per Motorway

Motorway	**MARE**	**RMSE [s]**
M6	0.0385	3.90
M11	0.0379	4.42

casts which prove more accurate than the currently used Published Profiles. This is done in a fast and computational inexpensive manner.

Regarding the accuracy results, forecasts result marginally more accurate in the case of the M6, even if the results for the M11 are very similar. The algorithm does not seem affected by the fact that the gathered M6 data suffers morning and evening travel time peaks, while the data from the M11, which was gathered in a single direction, sees only one peak a day. The algorithm shows worst performance during said peaks, but still obtains very significant prediction error reductions when compared to the Published Profiles.

There are relatively simple approaches that could a priori be used to fine-tune the algorithm, such as automatic tuning of the parameter α based on the distribution of travel times on a per-link basis. However, these should be formulated carefully, given that travel time distributions on motorways often display bimodal distribution, and aggressively targeting the data points around in the second mode will make all congested regime, part of which can indeed belong into the baseline dynamics of the motorway, into the spike signal and hence being treated as non-recurrent.

CHAPTER 5

Agent Architectures in Reinforcement Learning for Traffic Signal Control

5.1 Introduction

Traffic Signal Control (TSC) can be used to ensure the safe and efficient utilisation of the road network at junctions, where traffic can change directions and merge, having to manage conflicting individual priorities with the global needs of the network.

Sub-optimal TSC can cause numerous problems in ever-growing cities such as the increase of delays, congestion, waste of energy and air pollution. Traffic congestion has a major financial impact. A study by Inrix [179] shows that traffic congestion in 2019 cost £6.9 billion in the UK alone, with an average of 115 hours wasted per driver, resulting on an average cost of £894 per driver. Similar patterns are observed in other developed countries, with traffic causing Germany a loss of €2.8 billion and 46 hours lost per driver, and a cost for the US of $88 billion and 99 hours lost per driver. That number could reach $3400 billion yearly worldwide by 2030. Other studies [180] suggest a more modest cost, around $29 billions dollars for the United State each year.

In order to improve their traffic conditions, populous cities around the globe are exploring the deployment of smart Urban Traffic Controllers (UTCs) that use real time data to adjust their stage schedule and green time duration. Traditionally, fixed time plans have been used. Those fixed time plans can be optimised by systems that try to optimise the green time splits in a deterministic manner such as TRANSYT [48]. These types of systems require site-specific knowledge of the traffic lights placement and typical demand profiles to be able to provide effective control. These methods are not easily scalable and deteriorate over the years as the traffic demand changes [181].

With the development of induction loops that can detect the presence and estimate the speed of vehicles at a given position, real time (actuated) UTCs were created in two variants: those that optimise single isolated intersections with systems such as MOVA (Microprocessor Optimised Vehicle Actuation) [45], and those that cover multiple intersections such as SCOOT (Split Cycle and Offset Optimisation Technique) [51]. The latter group of systems uses data provided by induction loops around a partitioned area or set of intersections, an inability to overlap these areas makes it not easily scalable.

To remedy the scalability problem, other systems based on the varying state of the managed intersections and those neighbouring it were developed. These methods are based on local rules that generate a self-organising area traffic controllers. One such method is SurTrac [61], which solves a forwards implementation of Dynamic Programming. Other systems are based on physics, such as the different versions building on the concept of BackPressure [182, 183, 184] to maximise junctions' throughput.

With the recent breakthrough of Deep Reinforcement Learning (DRL) on high complexity problems such as Atari games or Go [185, 186, 187], attention has been turned towards the adaptation of these approaches to generate industry-grade controllers for traditionally noisy and difficult to control systems such as TSC.

The purpose of this chapter is to reproduce some of the results of the main and most successful RL approaches on intersections and networks of increasing complexity. A secondary objective is to compare different architectures of DRL TSC agents, since, given the complexity of their implementation, most available literature only deals with a single class.

The chapter is organised as follow: Section 5.2 contains an introduction of the reinforcement learning framework , Section 5.3 introduces the basic necessary mathematical background, in Section 5.4 basic reinforcement learning algorithms are presented, in Section 5.4.6 the process to implement an modular RL environment on the traffic simulator Vissim is presented, then the experimental results and comparisons between the different reinforcement learning agents (DQNs and A2C) with MOVA and SurTrac are presented on different junction and network configurations. Sections 5.5 through 5.8 present different configurations tested and show the results obtained. Lastly, Sections 5.9 and 5.10 introduce the conclusions, limitations and set out the future work.

5.2 State of the Art

5.2.1 Reinforcement Learning Methods

As introduced in Section 2.2.5, Reinforcement Learning is an area of Machine Learning that tries to imitate how biological entities learn. In it, an independent agent evolves in an unknown environment and learns how to perform a task for which no prior information is given based on its interactions said environment via a set of allowed actions. The goal of the agent is to maximise the total reward signal that it receives as a feedback for each of its actions.

RL methods have already successfully been applied to TSC in experimental setups. A good review of early methods can be found here [188] (without recent deep learning function approximations). More recent works [126, 130, 189] use neural networks as function approximators to avoid the limitations regarding dimensionality and computing time of table based methods in large state-action spaces, and they show that DRL TSC can indeed be more efficient then previous actuated methods.

While there is a variety of approaches in the literature that craft successful RL-based TSC systems, most of them do no present direct comparisons against commercial systems that are the concern of this chapter.

Gao et al. [124] used a CNN and discrete cell encoding with a Target Network for a value-based agent. The results were compared against a fixed time and a heuristic system (longest queue first), finding RL to perform better.

In Mousavi et al. [125], raw pixels were used as input for a CNN that parametrises two agents: a policy-gradient agent and a value-based agent. The variation in the delay between actions was used as reward, and while both agents were found to have near-identical performance, they were not compared against any reference system.

Later, in Wan and Hwang [190] a DQN using discrete cell encoding as state was implemented. It used a CNN architecture and a delay-based reward. It was compared against a fixed time system, obtaining better performance.

Liang et al. [126], used the same approach and included speed information in the state, using a reward based on variations on aggregated wait time for all vehicles. It compared against two different fixed-time systems, ranking better than both and providing some evidence of the benefits of using Double DQN, Duelling architecture and Prioritised Experience Replay.

In Genders and Razavi [128] different state spaces were evaluated using a policy-gradient algorithm, including: occupancy and speed for each incoming road, queue and a measure of density of incoming roads, and Discrete Cell Encoding, partitioning the incoming roads into cells of fixed lengths, in this case 2.5 metres. Genders found little difference in the performance of the agents as a result of the change in the magnitudes observed, but it could be argued that a discrete cell encoding would greatly benefit from a Convolutional Neural Network (CNN) architecture in the agent, which is not used.

Regarding comparisons with established systems, in Stevanovic and Martin [191] the authors compare SCOOT with a Genetic Algorithm-based control method. It is shown that SCOOT's performance can be surpassed by more adaptive Genetic Algorithms that, in turn, tend to be less effective at learning than RL methods.

Despite of the amount of previous work, most results are hard or impossible to reproduce given the lack of industry standards in terms of simulators, performance metrics, test-bench suites, the lack of availability of commercial algorithms for comparison, given the fierce protection of their internal workings and the lack of open-source code of proposed RL models. The objective of the chapter henceforth is to provide a reliable comparison in a variety of setups that can be used as a baseline reference for future improvements in the field.

5.2.2 Commercial Traffic Signal Control Optimisers

5.2.2.1 MOVA

MOVA (Microprocessor Optimised Vehicle Actuation) [45] is a traffic control software designed by TRL Software. It purpose is to reduce delay on isolated intersections and junctions. The basic functioning of MOVA involves two induction loop detectors estimating the flow of vehicles in each lane of the intersection. The system makes a virtual cell representation of the lanes within MOVA as that shown in Fig. 5.1, and then it computes a performance index based on the delays calculated. If the index results lower than a certain threshold, the signal is changed to the next stage, otherwise the stage it is extended.

5.2.2.2 SURTRAC

Scalable URban TRaffic Control (SURTRAC) [61] is an adaptive TSC system published in 2013. A real-world deployment on 20 intersections in Pennsylvania showed an improvement of 20-40%. Surtrac operates in a decentralised manner, with each intersection allocating its green time independently and asynchronously based on incoming flows. Each intersection is controlled by a local scheduler and communi-

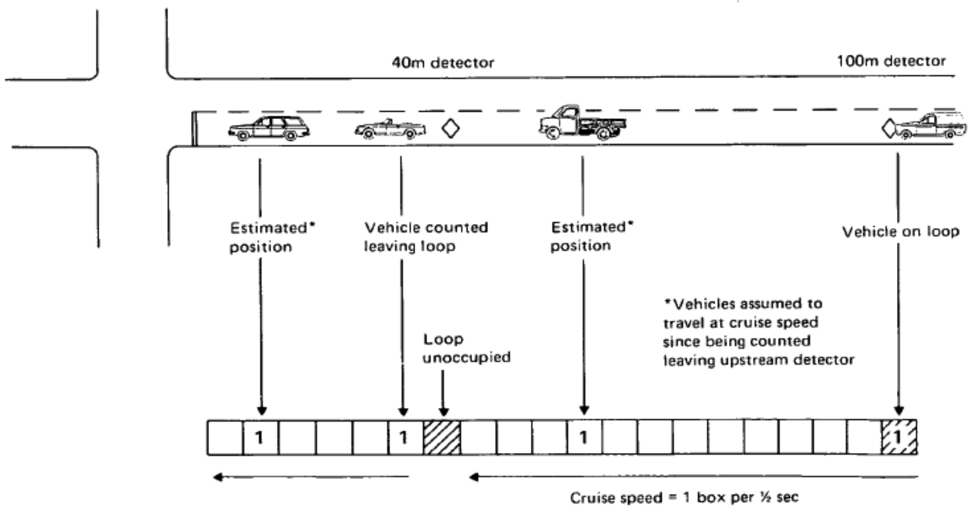

40m detector
100m detector
Estimated* position
Vehicle counted leaving loop
Estimated* position
Vehicle on loop
Loop unoccupied
*Vehicles assumed to travel at cruise speed since being counted leaving upstream detector
1
1
1
1
Cruise speed = 1 box per ½ sec

Figure 5.1: Internal cell partitioning of a lane within MOVA. Source: TRL[45].

cates projected outflows to the to downstream neighbouring junctions, modelling vehicles as a sequence of clusters or platoons in which the vehicle order must be preserved. This communication allows for locally balancing competing flows while creating larger "green corridors". The scheduling problem is to find an optimal sequence such that the input jobs (ordered clusters) are cleared while minimising the joint waiting time of all vehicles. These schedules are recomputed once per second.

5.3 Traffic Control as a Markov Decision Process

The control of a traffic intersection can be formulated as a Markov Decision Process (MDP). As seen in Section 2.2.5.1, the MDP is defined in terms of a 5-tuple:

- A set of possible environment states $s \in \mathcal{S}$.
- A set of actions of the agent $a \in \mathcal{A}$.
- A stochastic transition function $\forall a \in \mathcal{A}, \mathcal{T}^a_{s,s'} \triangleq \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a)$.
- A scalar real valued reward function $R(s_t, s_{t+1}, a_t)$ that provides a performance measure to the transition generated by progressing into the state s_{t+1} after taking action a_t while in state s_t
- A discount factor γ that will provide the balance between immediate exploitation and approaches that aim to maximise returns over time.

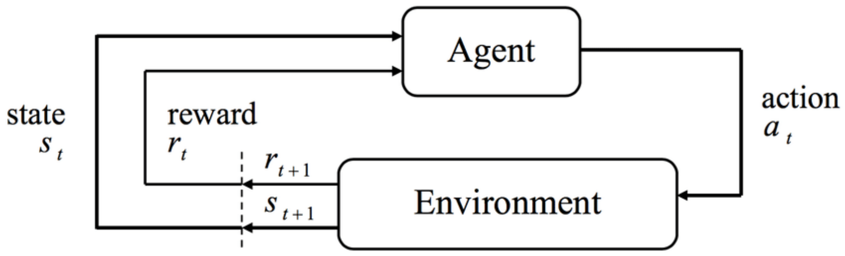


Figure 5.2: Schematic agent-environment interaction in a Markov Decision Process. Source: Sutton and Barto [108].

Each time an action is required, the agent will receive a state vector s_t from the environment. Based on this state, the agent will produce an action a_t, which will be implemented in the simulator. The environment will then advance time until a next action is required, according to its dynamics represented by $\mathcal{T}^a_{s,s'}$. At this point, the next state s_{t+1} will be observable. Both states will be used to generate a reward r_t to serve as feedback to the agent. The agent will receive the state observation s_{t+1} and the cycle will start again.

The transition function of an MDP can be deterministic (e.g. Go, chess), known (e.g. backgammon) or unknown (e.g. finance, traffic). The observability of the system by the agent can be complete, knowing all relevant variables of the environment (e.g. Go, chess) or partially observable (e.g. Starcraft, traffic). Arguably, with modern sensors, self-driving cars to eliminate the human factor, and adaptive TSC systems, traffic could become completely observable in the near future. However, for the purpose of this research, we will treat urban traffic as a partially observable process being guided by an unknown stochastic transition function.

From here on, it is assumed that the traffic environment displays the Markov property, i.e. the process is memoryless, with the next state only depending on the current state and the action taken. This assumption does not hold when we look at the state of an urban intersection over a period of days to weeks, since patterns tend to arise on these timescales, as indicated in previous chapters, displaying similar states at similar times of the day. However, this assumption holds when the temporal horizon of the control problem is very small when compared with the temporal dimension of the seasonality with the smallest period, such as the case of traffic where seasonality is measured in days to weeks and timespan of an action is a few seconds.

5.4 Methods

5.4.1 Reinforcement Learning

Within this frame, the goal of the agents will be to maximise their future discounted return, as defined in Eq. (2.17) $G_t = \sum_{t=0}^{+\infty} \gamma^t r(s_t, a_t)$ with $\gamma \in [0, 1]$. This is done by learning a policy π, parametrised by the weights θ of the neural network performing the approximation of the reward function (to partially avoid the curse of dimensionality associated with table-based methods) and mapping states to actions: $\pi : f_1(s) \to a$. The reward function provides the agent with feedback about its performance, mapping an action given a state to a scalar value: $r : f_2(s, a) \to \mathbb{R}$.

The value function of a state is $V_\pi(s) = \mathbb{E}_\pi[G_t | s_t = s]$. It describes how good a state is under a policy, the higher the value is the better the state.

The action-value function or Q-value is $Q_\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a]$. It represents the total episodic return by following policy π after being in state s and taking action a.

5.4.2 Value-based Reinforcement Learning Methods

Tabular value-based methods, such as Q-Learning, attempt to learn an optimal policy $Q^*_\pi = \max_\pi \mathbb{E}[r_t | s_t = s, a_t = a]$ by iteratively performing Bellman updates on the Q-values of the individual state-action pairs:

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha\big(y_t - Q_\pi(s_{t+1}, a_{t+1})\big) \tag{5.1}$$

where α is the learning rate and y_t is the Temporal Difference (TD) target for the value function:

$$y_t = r_t + \gamma \max_{a_{t+1}} Q_\pi(s_{t+1}, a_{t+1}) \tag{5.2}$$

5.4.3 Deep Q-Network Agents

The Deep Q-Network method is an evolution of Q-Learning, popularised in Mnih et al. [39] by successfully playing Atari games from raw pixels, some of them at super-human level. The purpose of the agent is to find an approximation of $Q_\pi*$ by tuning the weights θ of a neural network. The agent keeps a second neural network parametrised by θ' to generate the TD targets, that will be:

$$y_t = r_t + \gamma \max_{a_{t+1}} Q_\pi(s_{t+1}, a_{t+1}, \theta') \tag{5.3}$$

To do so it requires:

- For a state s, an action a is taken following an ϵ-greedy policy on Q_θ. Repeat until the end of the episode.
- The transitions (s_t, a_t, r_t, s_{t+1}) are stored in a replay memory $\mathcal{M}$ with capacity m.
- A mini batch of size b of (s, a, r, s') is randomly sampled from $\mathcal{M}$
- Perform a step of Stochastic Gradient Descent of θ.
- Every N learning steps, copy the weights to the target network: $\theta' \leftarrow \theta$.

The experience replay memory is used to increase the stability of the training, obtaining samples that cover a wider amount of situations. It can also increases the data efficiency since the same transition can be used several times for gradient descent.

There are two additions that have been used as extensions to the basic agent that generally improve performance, Prioritised Experience Replay, and Dueling Architecture, as defined in Section 2.2.5.4. The difference between a baseline DQN, the DuelingDQN (also known as DDQN) agent and the DuelingDoubleDQN (also known as DDDQN or D3QN) agent is the use of the Dueling and Double Q Learning modules as described in Section 2.2.5.4.

5.4.3.1 DQN Agents Implementation Hyperparameters

Two variants of the DQN agent have been implemented, one uses the improvements described above and is described in Algorithm 1, the second uses additional Double Q-Learning and is described in Algorithm 2. The agents implemented used the parameters described in Table 5.1.

Fully connected layers	2
Activation Function	ReLU
L2 kernel regularisation	0.001
Copy weight frequency	20
α	0.005
γ	0.95
PER η	0.6
PER β	0.4

Table 5.1: Hyperparameters of DQN-based agents

Algorithm 1 Operation of the implemented Dueling DQN Agent with PER

```
Initialise agent network with random parameters $\theta$
Initialise target network with random parameters $\theta$'
Initialise memory $M$ with capacity $m$
Define frequency $N$ for copying weights to target network
for each episode do
    measure initial state $s_0$
    while episode not done do
        choose $a_t = \pi(s_t)$ according to $\epsilon$-greedy policy
        implement $a_t$ and advance until next action needed
        measure $s_{t+1}$, and calculate $r_t$
        store transition $(s_t, a_t, r_t, s_{t+1})$ in $M$
        calculate TD error $\delta = r + \gamma \max_{a_{t+1}} Q_{\theta'}(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t)$
        calculate priority sampling weight and store in $M$
        $s \leftarrow s_{t+1}$
    end
    $b \leftarrow$ sample batch of transitions from $M$ according to priority weights
    for each memory $m_i = (s_i, a_i, r_i, s_{i+1})$ in $b$ do
        $\hat{y}_i = r_i + \gamma \max_a Q_{\theta'}(s_{i+1}, a')$
    end
    Stochastic Gradient Descent on $\theta$ over all $(x_i, y_i) \in b$
    if number of episode is multiple of $N$ then
        $\theta' \leftarrow \theta$
    end
end
```

Algorithm 2 Operation of the implemented Dueling Double DQN Agent with PER

```
Initialise agent network with random parameters $\theta$
Initialise target network with random parameters $\theta$'
Initialise memory $M$ with capacity $m$
Define frequency $N$ for copying weights to target network
for each episode do
    measure initial state $s_0$
    while episode not done do
        choose $a_t = \pi(s_t)$ according to $\epsilon$-greedy policy
        implement $a_t$ and advance until next action needed
        measure $s_{t+1}$, and calculate $r_t$
        store transition $(s_t, a_t, r_t, s_{t+1})$ in $M$
        calculate TD error $\delta = r + \gamma \max_{a_{t+1}} Q_{\theta'}(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t)$
        calculate priority sampling weight and store in $M$
        $s \leftarrow s_{t+1}$
    end
    $b \leftarrow$ sample batch of transitions from $M$ according to priority weights
    for each memory $m_i = (s_i, a_i, r_i, s_{i+1})$ in $b$ do
        $\hat{y}_t = r_t + \gamma Q_{\theta'}(s_{t+1}, \text{argmax}_a Q_\theta(s_{t+1}, a))$
    end
    Stochastic Gradient Descent on $\theta$ over all $(x_i, y_i) \in b$
    if number of episode is multiple of $N$ then
        $\theta' \leftarrow \theta$
    end
end
```

5.4.4 Policy Gradient Reinforcement Learning Methods

Policy Gradient in RL is based on the idea that sometimes obtaining a direct policy $\pi(s)$ mapping states to action is easier than estimating the value function $V(s)$ or the state-action values $q(s, a)$. It has an added benefit in the fact that, unlike DQN, it can learn stochastic policies, generating a probability distribution over the potential actions.

The goal is to find the policy that increases the amount of reward. To do so one has to perform gradient ascent on the performance measure $J = \sum_a [Q(s_0, a)\pi(a|s)]$ introduced in Eq. (2.36).

5.4.5 Advantage Actor Critic Agent

The Advantage Actor Critic (A2C) method tries to reduce the variance in the policy method by combining the direct mapping from actions with the value-based approximation method. The goal is to learn an actor

$$\pi_\theta = \mathbb{P}_\theta[a_t = a | s_t = s] \tag{5.4}$$

and a critic

$$V_\pi^\theta(s) = \mathbb{E}_\theta[G_t | s_t = s], \tag{5.5}$$

both of which are parametrised by the neural network weights vector θ.

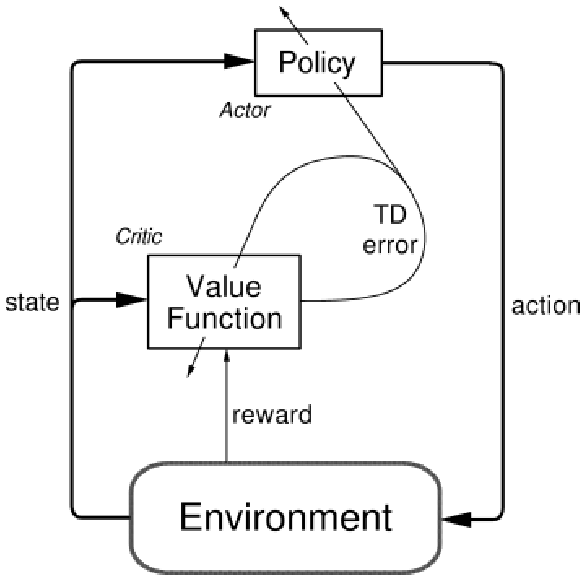


Figure 5.3: Internal architecture of Actor Critic. Source: Sutton and Barto [108].

5.4.5.1 Actor Critic Agents Implementation and Hyperparameters

The implementation of the A2C agent is presented in the pseudocode in Algorithm 3 and the parameters used for the implementation of the A2C agent are summarised in Table 5.2.

Fully Connected layers for value	2
Size of neural network layers	48
Fully Connected layers for policy	2
Activation Function	ReLU
n-return steps	16
Cross-entropy loss	0.5
Value loss coefficient	0.5
γ	0.95
α	10^{-5}

Table 5.2: Hyperparameters of AC-based agents

Algorithm 3 Operation of the implemented Advantage Actor Critic Agent

```
Initialise actor network with random parameters $\theta_a$,
Initialise critic network with random parameters $\theta_c$,
for each episode do
    reset gradients $d\theta_c = d\theta_a = 0$ measure initial state $s_0$
    choose action $a_t = \pi(s_t)$ according to $\pi_\theta(a_t|s_t)$,
    while episode not done do
        implement action $a_t$,
        advance simulator until next action needed,
        measure new state $s_{t+1}$, and calculate reward $r_t = V_{\theta_c}(s_t)$,
        choose action $a_{t+1} = \pi(s_{t+1})$ according to $\pi_{\theta_a}(a_{t+1}|s_{t+1})$,
        update actor $\theta_a = \theta_a + \alpha \nabla_{\theta_a} \ln \pi_{\theta_a}(a_i|s_i) Q_{\theta_c}(s_i, a_i)$,
        calculate TD error $\delta \leftarrow r_t + Q_{\theta_c}(s_{t+1}, a_{t+1}) - Q_{\theta_c}(s_t, a_t)$,
        update critic $\theta_c = \theta_c + \alpha\delta\nabla_{\theta_c} Q_{\theta_c}(s, a)$,
    end
end
```

5.4.6 Simulation Interface

In this section, more detail will be given on the simulation setup that was used.

All the experiments presented in this chapter were simulated using the commercial traffic simulator PTV Vissim [192]. The main interface between Vissim and python is the COM (Component Object Model) interface. However, there are many functionalities that are not provided and so it was needed to add additional code to collate information and distribute control in order to allow for multi-agent training. A virtual server was set up using the Windows COM interface, allowing the connections from different RL UTCs implemented in Python. These UTCs can execute all different stages of control allowed for a given map, allowing controllers to interact and correspond to multiple intersections. The interface has to handle these updates of the different signals, gathering the state space and the reward for the reinforcement learning process and signal changes and state update have to managed asynchronously.

To this end the interface is partitioned in two different blocks (environment and agent), with 2 levels each.

The first block is the one acting on the simulator side (indicated in red in Fig. 5.4). The lowest level is what has been called the *Environment*. The Environment itself consists of several Signal Control Units and several functions over the whole simulator such as resetting the simulator or varying the aggregate flow of vehicles. It is responsible for dispatching and maintaining the COM server, scanning and identifying different components and intersections in the network, and passing information between the simulation and the Signal Control Units.

Next are the *Signal Control Units* (SCUs). The SCUs act as an abstraction of a junction, which accounts for the region of control and monitoring of an agent. When initiated, the signal control unit is provided with a dictionary that informs the signal control unit of which signals and phases it is responsible for, how long each stage should last for safe operation, which state and reward information it should gather during this time. When information is gathered and new action is required, the control unit makes a request from the environment for new a new stage to implement. This request contains all relevant information to the agent in terms of simulator state. Once a new stage is given by the agent, the control unit send this information to the simulator. In terms of practical traffic control, they are analogous of the micro-controllers on the traffic lights.

On the second block we have an *Agent* (indicated in blue in Fig. 5.4), which receives a representation of the state of the simulator from the SCU and uses its internal neural networks to select which action is to be implemented by the SCUs. Other than this, the agent is only responsible for storing the memory tuples and performing updates on the weights of its neural networks. A level above the agent there is what came to be called the *Master Agent*. The Master Agent can contain one or more individual agents, is responsible for saving, loading and

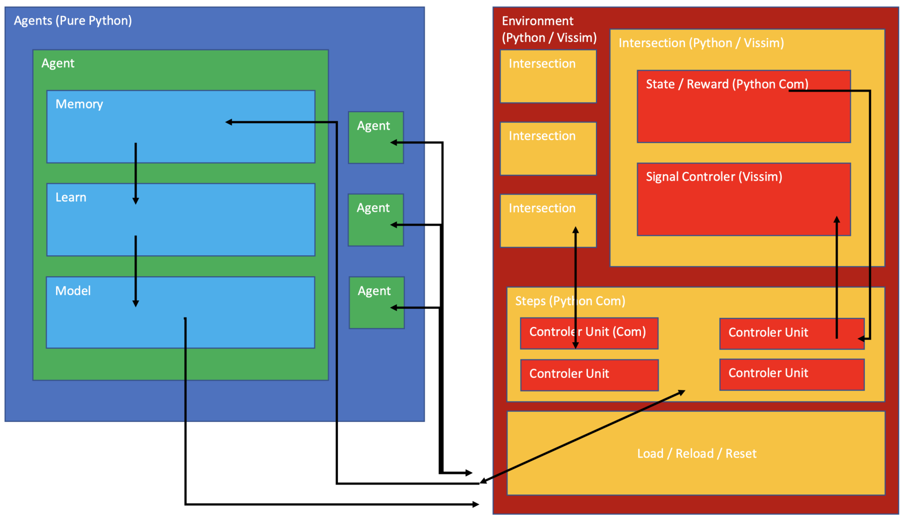


Figure 5.4: The simulation interface.

keeping track of the metrics of the agents. Furthermore, all training, testing and memory population are originated in the Master Agent, which was written similar to the OpenAI gym interface, acting as the modern standard for RL training. The agents are allowed to prepopulate memories, train, test (run without training and gathering more simulation data) and demo (run with grapical interface).

This set up is quite natural and similar to the way modern UTC systems for isolated junctions operate as well as modern AI training environments. We note that our implementation of a control unit significantly increased the speed of training by a several orders of magnitude and further allowed for more distributed training and updating. The OpenAI style of interface simplifies the interaction between agents and the environment. Further, our multi-agent environment allows us to flexibly test and train a variety of different agents.

5.4.6.1 State, Action and Rewards of the agents

The experiments presented in the following sections all use the same descriptions for simulator state and reward calculation, although they differ in the number of actions available to them.

The state of an intersection of l lanes will be presented to the agents as a state vector $s \in \mathbb{R}^{l+1}$. While it could be interesting to use the incoming flows as state variables, this was experimentally tested not to be feasible while using Vissim due to an great worsening of computation times. As per Allsop [193], as cited in Heydecker [194]:"The number of vehicles in the vertical queue associated with a stream of traffic can be interpreted as the mean rate of delay in that stream and provides an objective that bears direct economic interpretation". Hence, each component will contain the length of the queue of vehicles measured upstream from the traffic light in metres. The last component will be the numeric ID of the current stage that the agent is implementing. Given the lack of sharp turns or other elements that can force a vehicle to decelerate for a different reason than having another vehicle stopping in front of it, a vehicle will be considered in queue once its speed drops below 20 km/h and will be considered no longer in queue once its speed surpasses 30 km/h.

As per the statement of Heydecker [194], as reported in the previous paragraph, queues are reasonable choice of measure both for states and rewards, being able to transmit useful information to the agent relative to the mean rate of delay of the system. Based on this, the reward after an action will be calculated as the negative sum of the length of the queues of all lanes immediately upstream from the intersection:

$$r_t = -\sum_l q_l. \tag{5.6}$$

There were also initial attempts to set the reward function to either the stop delay or the delay of the vehicles (which are the direct evaluation metrics), but it was found that using this configuration hampered agent convergence and greatly increased the computational requirements without providing an advantage in the few occasions it worked. Consequently the research line was dropped.

The agent has a set of actions $\mathcal{A}$ that varies depending on the intersection to control. Once the agent chooses an action a, the stage corresponding with the ID of a is implemented. The green time is set to a minimum of 6 seconds. Once this time has passed, the agent is requested a new action. If the agent chooses the same action again, the current stage is extended for a further 3 seconds. There are no inbuilt limitations as to how many times an agent can extend a stage, leaving it for the agents to learn. If the agent chooses a different action than the currently active one, a 3 seconds amber stage is implemented in the lights that were green, after which, the new stage is implemented.

5.4.6.2 Saturation Rates and Benchmarking

Saturation flows in Vissim are determined based on the values of the parameters given to the Wiedemann's Car Following Model [195] which it uses internally. The default parameters, which were used ($bx_{\text{Add}} = 2m$, $bx_{\text{Mult}} = 3m$) result in a satu-

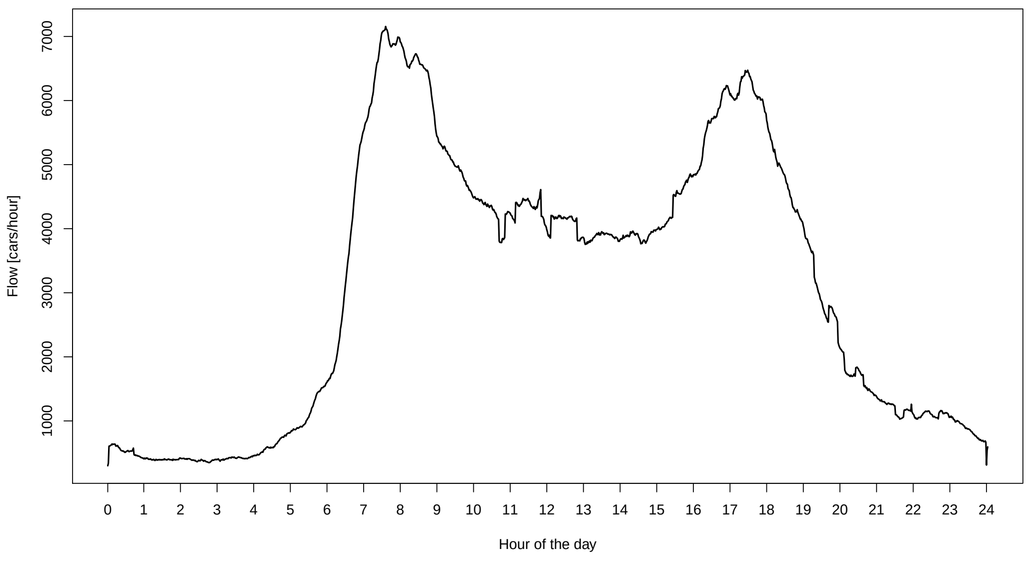


Figure 5.5: Daily profile demand created using the Hybrid Profiling Algorithm [1]. Demand values for the intersections are derived from this shape between 6AM and 10PM.

ration rate of 2144.13 vehicles * lane / hour of green time, obtained experimentally while averaging over 1800 seconds of simulated traffic. Given that the saturation flow in Vissim is defined by these parameters, the saturation flow obtained in this experiment can be extrapolated to all models used throughout this chapter by multiplying the saturation rate per lane by the number of lanes feeding into any given junction.

Regarding benchmarking, in order to compare each signal controller policy and agent, we need to define a specific testing framework. For each model except the last, a demand profile will be created. This will follow the shape found in a *typical day* as described in previous chapters. The profile will be split on 10 segments of length 6 minutes. Each of these segments will correspond with a level of demand. The levels of demand are obtained by setting out what will be the maximum demand the intersection will suffer, setting that magnitude to coincide with the peaks of the distribution that could be found on a *typical day* obtained by using the algorithms presented in the previous chapters and adjusting all other values proportionally. The general shape can be observed in Fig. 5.5, and the specific demand levels will be specified in each experiment.

Random seeds are changed and updated after every simulation episode, training or testing.

There are metrics that have been traditionally used in traffic management to quantify the performance of UTC systems, some of the main ones include reserve capacity and the mean rate of delay [193, 194, 196]. However, these measures and their calculation originate from the basis of delay estimation [193], since at the time these measures were defined, delay could not be directly and accurately measured. In the research presented in this thesis, traffic simulators are used on all instances where traffic signals are being optimised. These simulators are able to produce extremely granular and accurate measures of delay, even on a per-vehicle basis if required. Hence, it seems reasonable to use measurements directly extracted from the simulator (such as average delay per vehicle) as a direct representation of the performance of the network instead of turning to secondary measures that are derived based on prior impossibility to obtain accurate and complete observations. Furthermore, based on the statements of Allsop [193], as cited in Heydecker [194], as seen in Section 5.4.6.1, it appears reasonable to also take queue lengths (also directly measurable from the simulator with great granularity) as a secondary alternative performance measure.

The main quantitative metrics on which the system will be evaluated are the Global Cumulative Delay and the Global Cumulative Stop Delay generated by all vehicles during the execution of the evaluation. The first one considers as delay any deviation from the maximum speed allowed in the link in which it is placed. The second one account for the number of seconds spent in a queue. Queues over time will be shown in those models in which they are informative, since as the number and size of the intersection grows, this quickly becomes untractable and its usefulness diminishes.

5.5 Single Cross Straight

The first test is conducted on the simplest junction configuration. This is given in Fig 5.6. The junction referred to as *Single Cross Straight* is composed on 4 lanes distributed in 4 arms coinciding with the cardinal directions of the model. These will be referred to as *North*, *East*, *South* and *West*, matching the direction from where the vehicles are inserted in the model upstream from the intersection. The controller for the junction has two stages, a north-south stages and an east-west stages, and the vehicles are not allowed to turn. The testing aim was to perform an initial performance comparison of deep reinforcement learning algorithms against MOVA from the Transport Research Laboratory and SUTRAC from RapidFlow

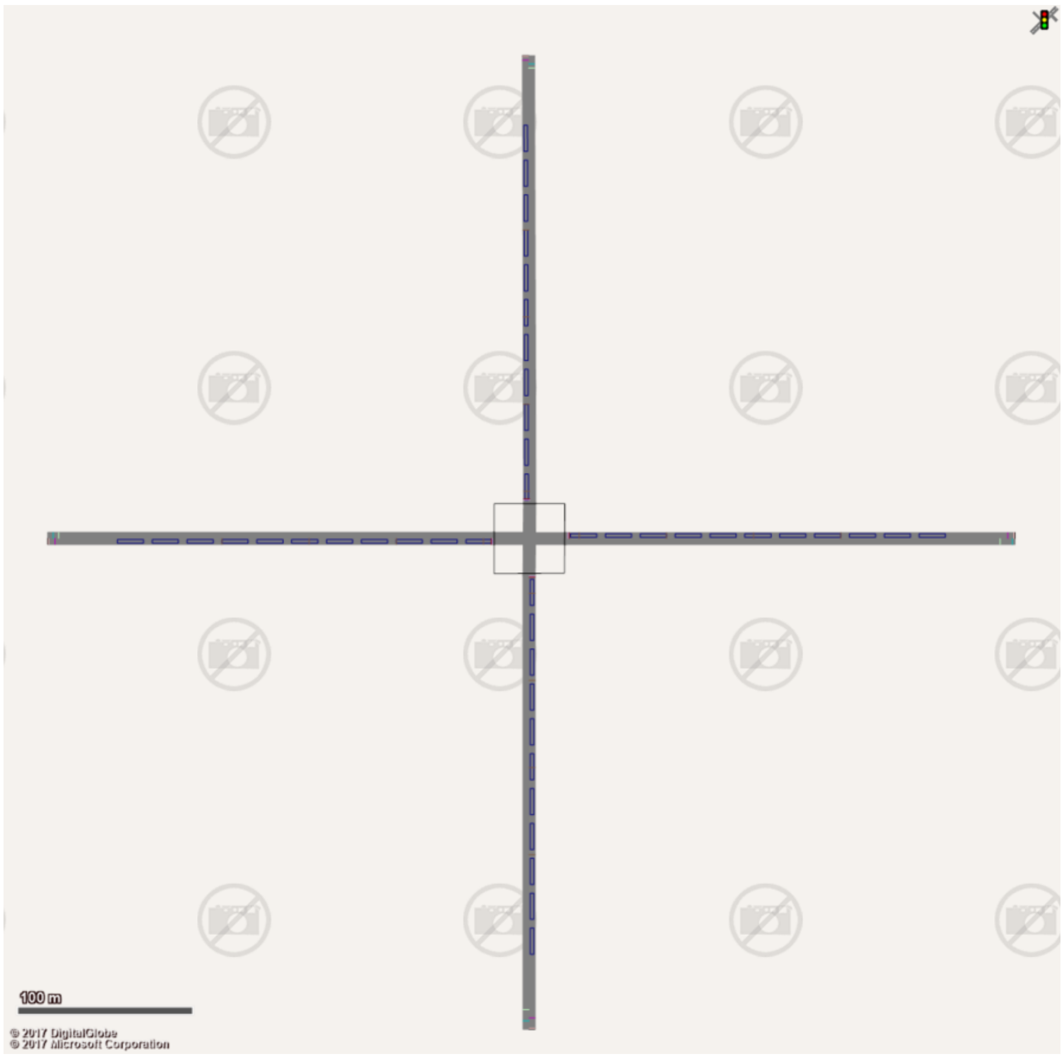


Figure 5.6: The Single Cross Straight model

Technologies in a setup that was as simple as possible and was required to exert fine adaptive timing control, rather than using complicated transitions between stages that would rarely, if ever, appear in sequence in cyclic control.

For the configuration of PC MOVA, loop detectors were placed at 30 and 90 metres upstream from the intersection, which corresponded to distances recommending in the MOVA Traffic Control Manual. A minimum 5 second green time was set with one second amber, which again is a standard recommended timing. The implementation of SURTRAC used in this chapter is based on the work of Xie et al. [197].

A Duelling Deep Q-Network with PER, a Dueling Double Deep Q-network with Prioritized Experience Replay as described in Section 2.2.5.4, and an Actor-

Critic algorithm as detailed in Section 2.2.5.5 and using the same parameters.

5.5.1 Scenario Description and Experimental Setup

In each timestep when an action is required, the queue sizes at the lanes are concatenated with the current action of the junction, and this was passed to the agents as input. The optimization objective of all learning algorithms was to minimize the queue between control decisions.

In order to train the models, they were ran using a fixed vehicle demand of 400 vehicles per hour on each of the incoming lanes, for a total aggregated demand (the flow of vehicles the UTC is meant to serve) of 1600 vehicles/hour. Both variants of the DQN agent were trained for 400 episodes of one hour of simulated demand, using an ϵ geometrically annealed from 1 to 0.001. The average reward of each training episode was recorded, and the best agents were selected using the best performing agents at their best performance during their training runs. The A2C agents were trained for 100 episodes until they converged, with the best performing A2C agents being found around episode 40.

The agents were then evaluated in scenarios lasting one hour, in which full data regarding queues, stops, stop delay and global delay are recorded on a per-second basis. This evaluation run is partitioned in 10 periods of 6 minutes each, with stochastic demand within the intervals centred around a mean value (see Table 5.3). According to Vissim's Manual, when using the Stochastic arrivals setting "stochastic fluctuations of the traffic volume may occur" [192]. While this can create some irregularities in specific instances of evaluation of quantities such as queues or per-lane throughput, especially so in the case of fixed-cycle controllers which could see moments of asymmetric demand, adding a level of stochasticity in the demand is expected to have a positive effect in the training of RL agents by increasing the variety of states that they will see. During evaluation, an average of 2120 vehicles are inserted in the model, with 2 peaks of demand of 3000 vehicles/hour for 6 minutes each.

The main evaluation metrics that will be used will be:

- Global Cumulative Delay: Understood as the deviation from the theoretical time in seconds a vehicle would take to cover the distance given by its route (model entrance $\rightarrow$ model exit) by circulating at the maximum allowed speed, aggregated for all vehicles.

- Global Cumulative Stop Delay: Understood as the total aggregated time in seconds that all vehicles have spent stopped.

Table 5.3: Demand in vehicles/hour per cardinal direction over the benchmark.

Time period [min]	North	East	South	West
0-6	200	200	200	200
6-12	400	400	400	400
12-18	900	500	900	500
18-24	1000	500	1000	500
24-30	700	500	700	500
30-36	500	700	500	700
36-42	500	1000	500	1000
42-48	500	900	500	900
48-54	400	400	400	400
54-60	200	200	200	200

- Queue length: Length in metres of the queues detected by the simulator sensors. If the queue overflows the lane in which it originates, the sensor will show a saturated value matching the length of the lane.

5.5.2 Experiment 1: Single Cross Straight

The first step was to determine the optimum cycle length for the fixed controller, imposing the condition of cycle times being fixed and equal for both stages, following the way in which the demand varies over the course of the scenario as show in Table 5.3. The optimum cycle length was determined to be 56 seconds, following the methodology shown in Salter [198]. This cycle length will be used for comparison with the adaptive controllers in this experiment, and henceforth be referred to simply as Cyclic Controller.

Figures 5.7 and 5.8 show the Global Cumulative Delay and the Global Cumulative Stop Delay for the network while being controlled by a trained Actor-Critic agent (A2C), a Dueling Deep Q-Network (DuelingDQN), a Dueling Double Deep Q-Network (DuelingDDQN), SURTRAC, MOVA, and the previously mentioned reference cyclic controller on a 56 seconds cycle respectively. Figures 5.9 and 5.10 present the same information without the cyclic controller for clarity. Each was tested against a range of loads on each lane as per the section above. As it can be seen, the cyclic solution is clearly beaten by adaptive UTCs such as MOVA, Surtrac and RL. The different UTCs are on a par with a slight advantage for the DuelingDDQN which saves the community 3000 sec compared to MOVA on this hour of simulation, which represents on average 1 or 2 sec for each car. The RL

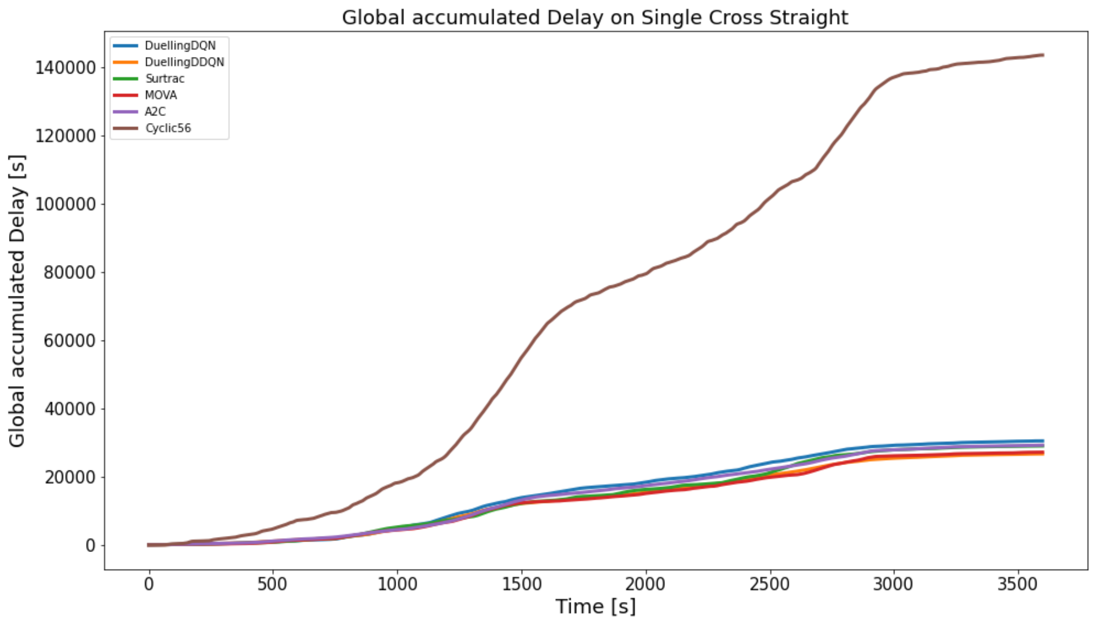


Figure 5.7: Global Cumulative Delay in Single Cross Straight.

agent also seems slightly more robust against changes in demand, producing lower slopes in the delay graphs in sections of extreme demand.

Table 5.4: Cumulative Delay and Cumulative Stop Delay in seconds across Controllers on Single Cross Straight.

Controller	**Cumulative Delay** [s]	**Cumulative Stop Delay** [s]
Cyclic	143660.50	76538.66
MOVA	27187.53	14361.43
SURTRAC	29008.36	15023.25
A2C	26382.14	11018.80
DDQN	28303.94	11211.24
DDDQN	21286.86	8847.86

Figures 5.11 - 5.16 display the queue lengths in each lane as a result of the different UTCs in operation. Given the symmetric demand described in Table 5.3, the asymmetries in the measured queues originate in the extra stochasticity introduced into the arrival rates by the stochastic arrival parameter in Vissim based on the choice of random seed. It has been experimentally tested that these fluctua-

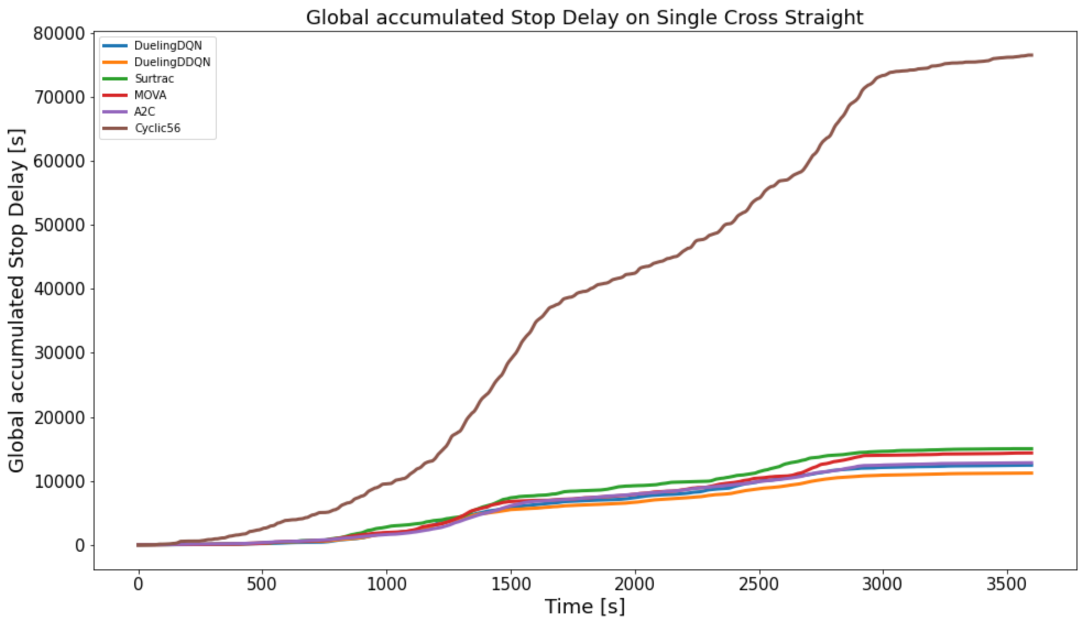


Figure 5.8: Global Cumulative Stop Delay in Single Cross Straight.

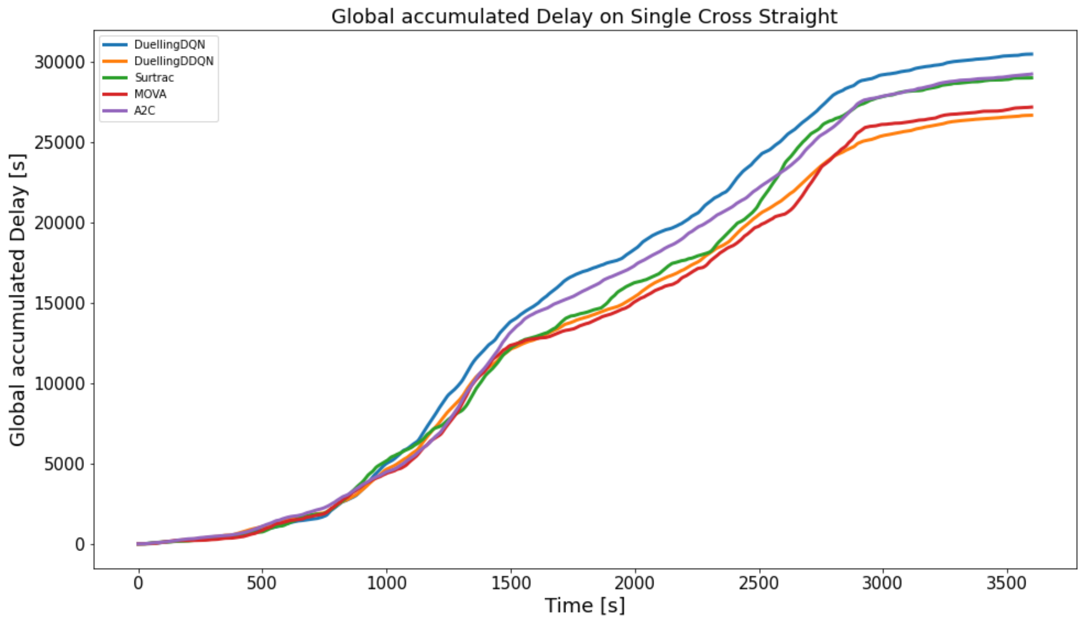


Figure 5.9: Global Cumulative Delay in Single Cross Straight without Cyclic Controller.

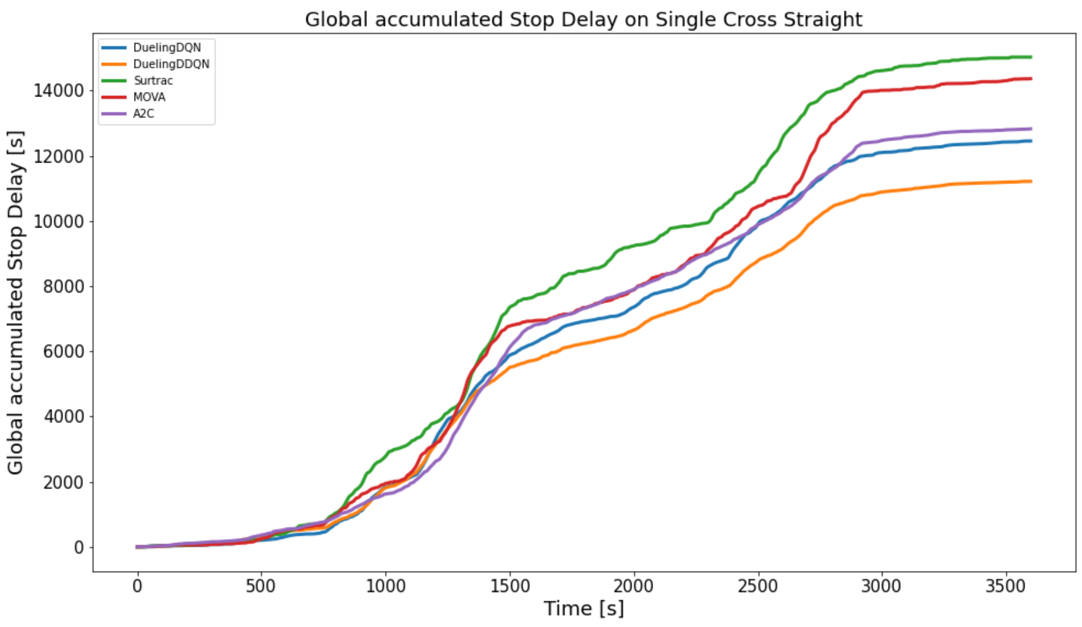


Figure 5.10: Global Cumulative Stop Delay in Single Cross Straight without Cyclic Controller.

tions, and the potential induced asymmetries in measures are entirely dependent on the choice of random seeds, meaning that while it will still be possible to observe them in results that only involve use of a single random seed (such as the graphs mentioned earlier in this paragraph), they will not have meaningful effects in the training of agents spanning several hundred episodes using different seeds.

Saturation on the sensors occurs when the queue reaches approximately 250 metres, coinciding with the end of the lane. In the figures it can be seen how all non-RL controllers are incapable of dealing with the two peaks in demand. The cyclic UTC results in saturated lanes during both peaks and queues in excess of 100 metres during a great part of the simulation. The UTC using MOVA suffered two moments in which at least a sensor was saturated coinciding with the peaks in demand, however the queues were close to lengths of around 50 metres during the most part of the simulator.

The UTC using Surtrac follows a similar pattern that only has a single lane saturated coinciding with the second peak in demand. The queue distribution is more irregular than in the previous case, but with similar average values. None of the experiments using RL agents as UTCs suffered of saturation in any of their lanes

Table 5.5: Average global queue length in metres across controllers in Single Cross Straight.

Controller	**Average Queue Size [m]**
Cyclic	132.37
MOVA	60.59
SURTRAC	72.41
A2C	56.07
DDQN	50.11
DDDQN	49.42

during the length of the evaluation. They all managed a more balanced distribution of queues in their respective lanes, displaying a higher ability to balance loads even during peak times.

The results of the custom implementation of Surtrac are quite impressive considering that it has not had any kind of parameter optimisation. Furthermore Surtrac is designed to work on clusters and because this is a single intersection model, clusters are less likely to form. Because of the simplicity of this 2 actions intersection, there is not a lot of delay difference between adaptive UTCs. As it will be appreciated shortly, these results will change when we consider more complex junctions.

Given the level of difference in performance between the adaptive and cyclic controllers, how this is more accentuated on more complex intersections and the increasing difficulty in properly setting them in big intersections (which incidentally motivated this research), the cyclic controller will be omitted for the next examples. Given that the A2C agent has been clearly outperformed in this experiment by those based on the DQN architecture, the following experiments will focus on the performance of this architecture compared with commercial systems.

5.6 Single Cross Triple

The intersection referred to as *Single Cross Triple*, as shown in Fig. 5.17 displays a much higher complexity than the intersection presented in the previous section. It is composed of 4 incoming links of 3 lanes each.

The links will be identified, as *North*, *East*, *South* and *West*, following the approach seen in Section 5.5. The individual lanes in a link will be identified as *Left*, *Centre* and *Right*, as they would be seen by a vehicle that was travelling by

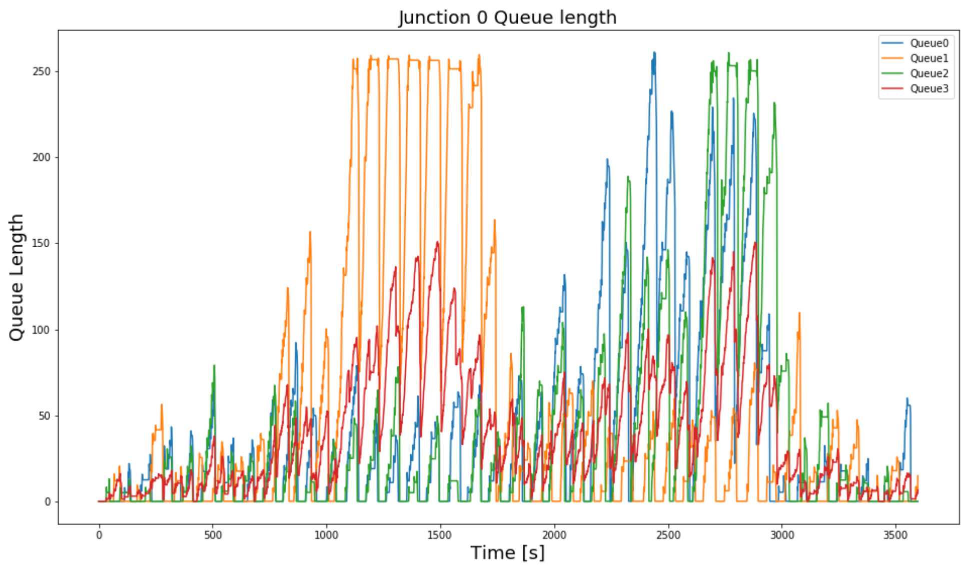


Figure 5.11: Queues over time in upstream directions from the intersection over an evaluation run using the Cyclic UTC

said link.

In each incoming link, the left lane serves a dedicated nearside turning lane, the central allows for forward travel and the right lane allows for both offside turning and going straight.

5.6.1 Scenario Description and Experimental Setup

The models were trained on a fixed demand of 300 vehicles per hours on each incoming link. The DQNs were trained for 400 episodes of one hour of simulated time, with ϵ annealed geometrically between 1 and 0.001. The A2C agents were trained for 100 episodes until they converged, and the A2C agents were found around episode 40. During the hour of evaluation, the demand profile from the last experiment was used with a scaling factor of 1.5, an average of 3180 vehicles were introduced to the model, with 2 peaks of demand of 4500 vehicles/hour for 6 minutes each.

5.6.2 Experiment 2: Single Cross Triple - 4 actions

Due to limitations in how Vissim internally treats the queues, it is not possible to obtain a straightforward measurement of the lane queues in links that have more than one lane. In these cases, the system returns the length of the longest queue.

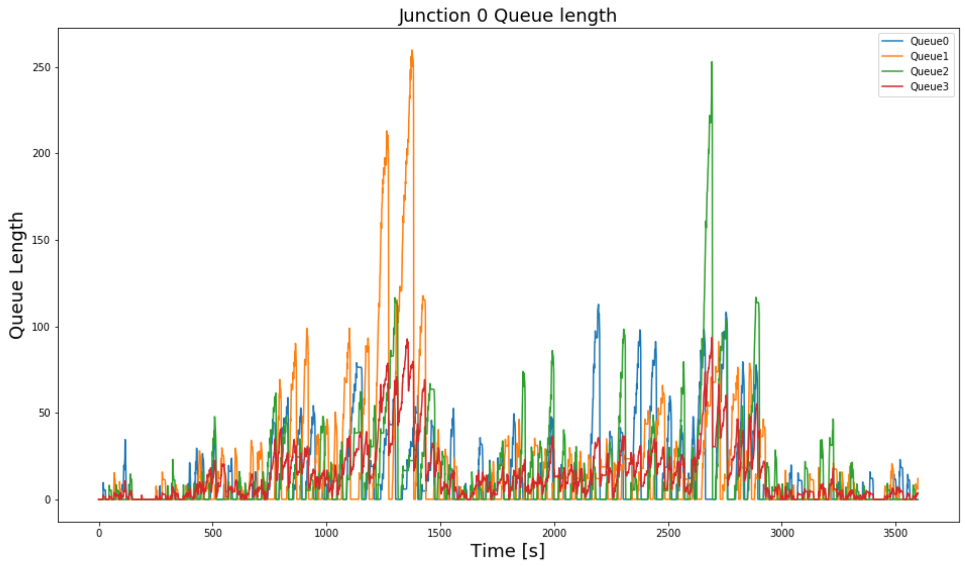


Figure 5.12: Queues over time in upstream directions from the intersection over an evaluation run using the MOVA UTC

To mitigate this, the first experiment was run with agents that would take 4 queue inputs, plus the state of the traffic signal as state input.

Due to the limitations in how the agents perceive the state of the intersection, the action set was limited to 4 different actions, being allowed only those that set to green the 3 traffic lights serving the lanes of the same incoming link. This allows for the vehicles to perform turns, but prevents more sophisticated stages from happening.

Table 5.6: Cumulative Delay and Cumulative Stop Delay in seconds across Controllers on Single Cross Triple - 4 actions.

Controller	**Cumulative Delay [s]**	**Cumulative Stop Delay [s]**
MOVA	260257.65	215527.96
DDQN	135220.91	104521.26
DDDQN	155563.22	122184.40

As it can be seen in Figures 5.18 and 5.19, the UTC using MOVA performs poorly compared to the RL agents DuellingDQN and DuellingDDQN. During this hour of simulation RL agents halve the cumulative delay, which saves over 100000

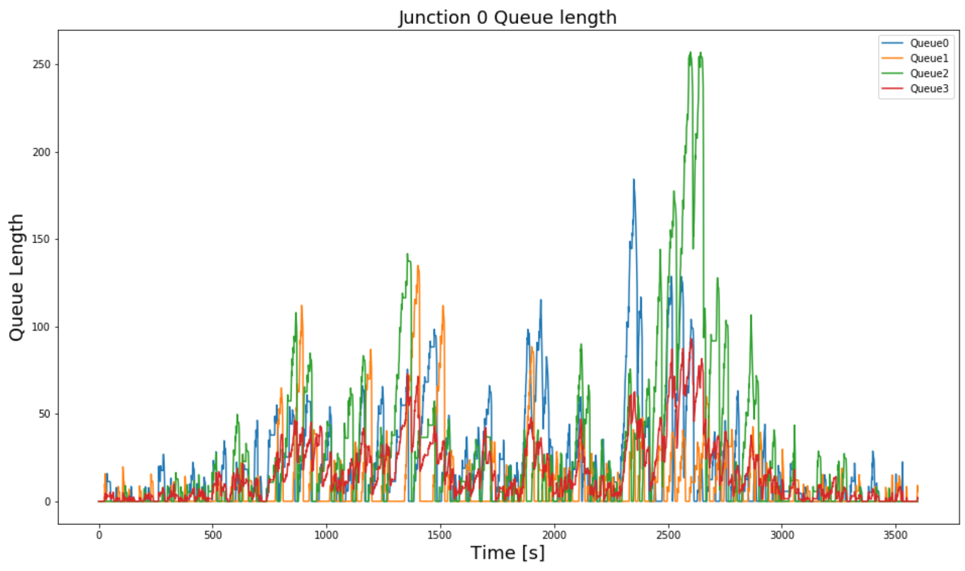


Figure 5.13: Queues length over time in upstream directions from the intersection over an evaluation run using the Surtrac UTC

seconds for all vehicles involved, which is over 27 hours, meaning an average of over 32 seconds of travel time saved per vehicle.

Table 5.7: Average of longest global queue length in metres across controllers in Single Cross Triple - 4 actions.

Controller	**Average Queue Size [m]**
MOVA	179.27
DDQN	128.20
DDDQN	153.58

The Figures 5.20, 5.21 and 5.22 present the queue length per cardinal direction as seen by the agent during evaluation. Here again the length of the queues in those intersections controlled by RL agents during the test scenario were lower than the ones controlled by MOVA. Additionally it can be seen that the agent using Dueling Double Q-Learning has a more stable performance than that Dueling Q-Learning.

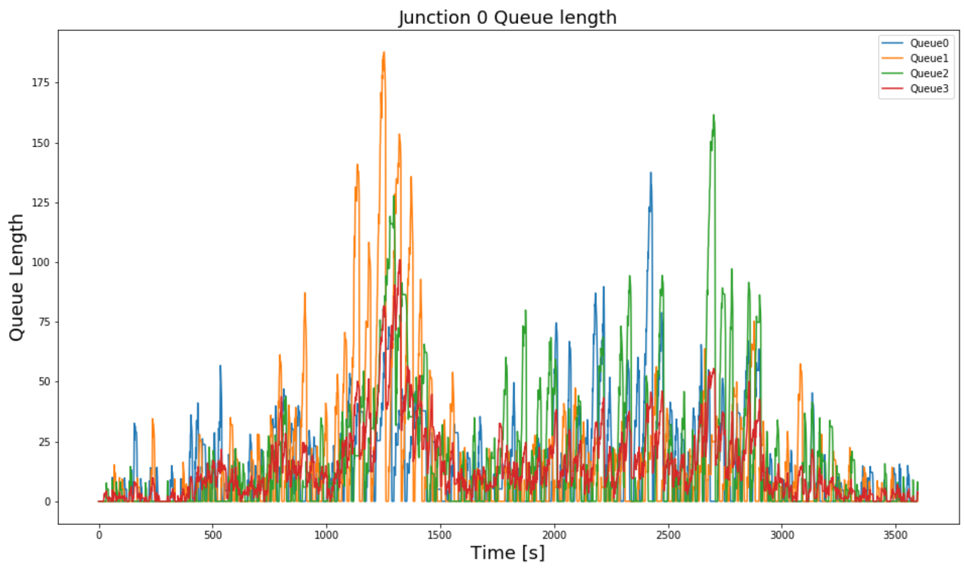


Figure 5.14: Queues length over time in upstream directions from the intersection over an evaluation run using the A2C UTC

5.6.3 Experiment 3: Single Cross Triple - 8 actions

A series of workarounds were explored to overcome the limitation in queue length measurements, including external measurement and attempting to interface with the simulator's internal variables. These were only partially successful, correct measurements were obtained but with a subsequent general slowing down of the simulation to levels in which the generation of the raw amounts of data necessary for RL was no longer feasible.

As an alternative, the map was reworked. All original lanes were partitioned into their own independent links, and new links with a length of 100 metres were added, as shown in Fig. 5.23. This allows the vehicles to change lanes according to their routes after being places in the edges of the network but before reaching the point in which lanes are split into independent links, since lane changing after this split is no longer possible for the vehicles.

Given this setup and aiming to provide the RL agents with the greatest amount of room to act, 8 different stages are available as represented in Fig. 5.24. No specific stage order is enforced, and the agents are free to change between any combination of stages.

While these modifications allowed using information from all lanes in an

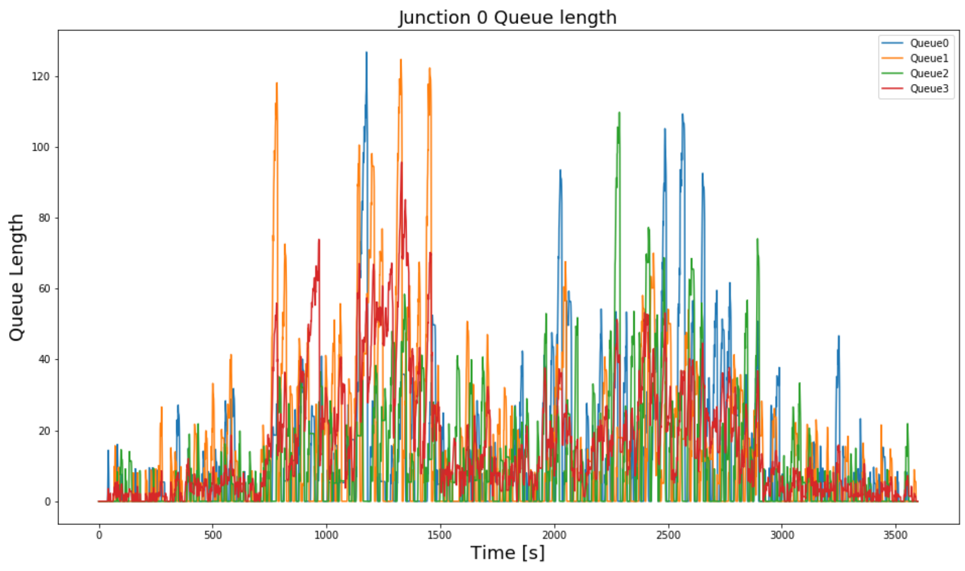


Figure 5.15: Queues length over time in upstream directions from the intersection over an evaluation run using the Dueling DQN UTC

akin manner to what modern sensors would achieve, the following results can not be directly compared with those of the previous sections. Both models share name and rough geometry, but the layout of lanes is changed and so are the routing possibilities open to the vehicles.

The results presented below, use DQN agents taking 12 queue length inputs plus the state of the signal.

Table 5.8: Cumulative Delay and Cumulative Stop Delay in seconds across Controllers on Single Cross Triple - 8 actions.

Controller	**Cumulative Delay** [s]	**Cumulative Stop Delay** [s]
MOVA	165456.44	139929.55
DDQN	72642.59	56233.30
DDDQN	71245.61	53855.98

The RL agents display a similar gap in performance with MOVA as in the previous experiment. RL agents manage to generate about a third of the delay produced by MOVA, following the same trend as in the previous experiments.

While this appears to be a great success, these results have to be put into

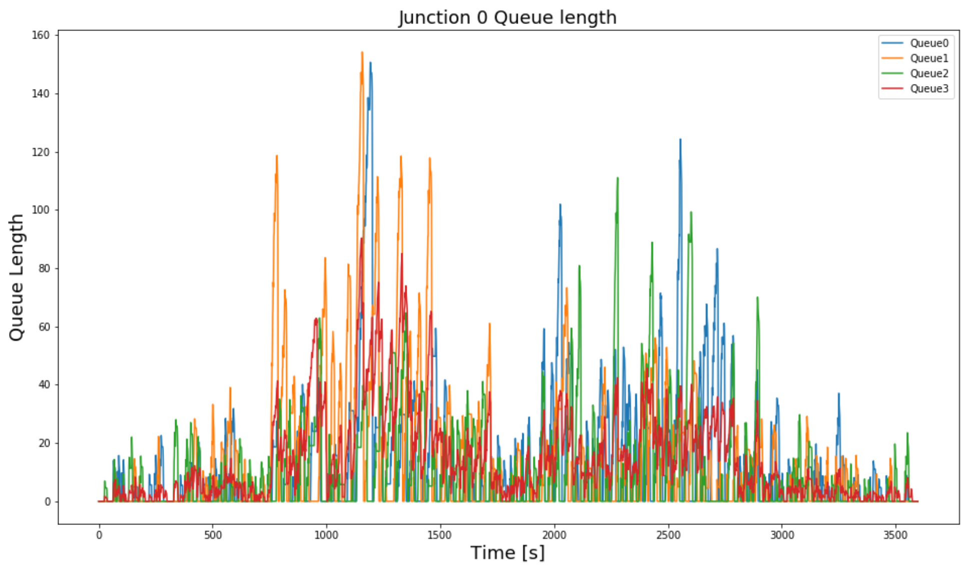


Figure 5.16: Queues length over time in upstream directions from the intersection over an evaluation run using the Dueling DDQN UTC

context. On one hand, MOVA is not designed to control so many phases and lanes. MOVA has a lot of internal parameters meant to be fine tuned by a traffic engineer. In this experiment, MOVA was configured in its most basic operation method, with many of its parameters set to their default values, since the values that a traffic engineer would assign them were unknown. The configuration regarding detectors, speed limits, priorities and phases compatibility was implemented carefully following the MOVA manuals that are made available with the software. These parameters did produce a successful control loop, operating in line with what was expected of the configuration process. On the other hand, none of the RL agents has been fine tuned to the level that would be expected during commercial operations. The distribution of layers and neurons was not optimised, nor were the activation functions, learning rate or discount factor, meaning that the RL agents can still be improved upon.

5.7 Scaling up to Five Intersections

The *Five Intersection* model is the first one with multiple junctions. It is composed of 5 concatenated copies of the *Single Cross Triple* map in a cross configuration, as it can be seen in Fig. 5.27. This model does not include inter-agent communication of any kind, hence the focus of this test is the generalisation abilities of pre-trained

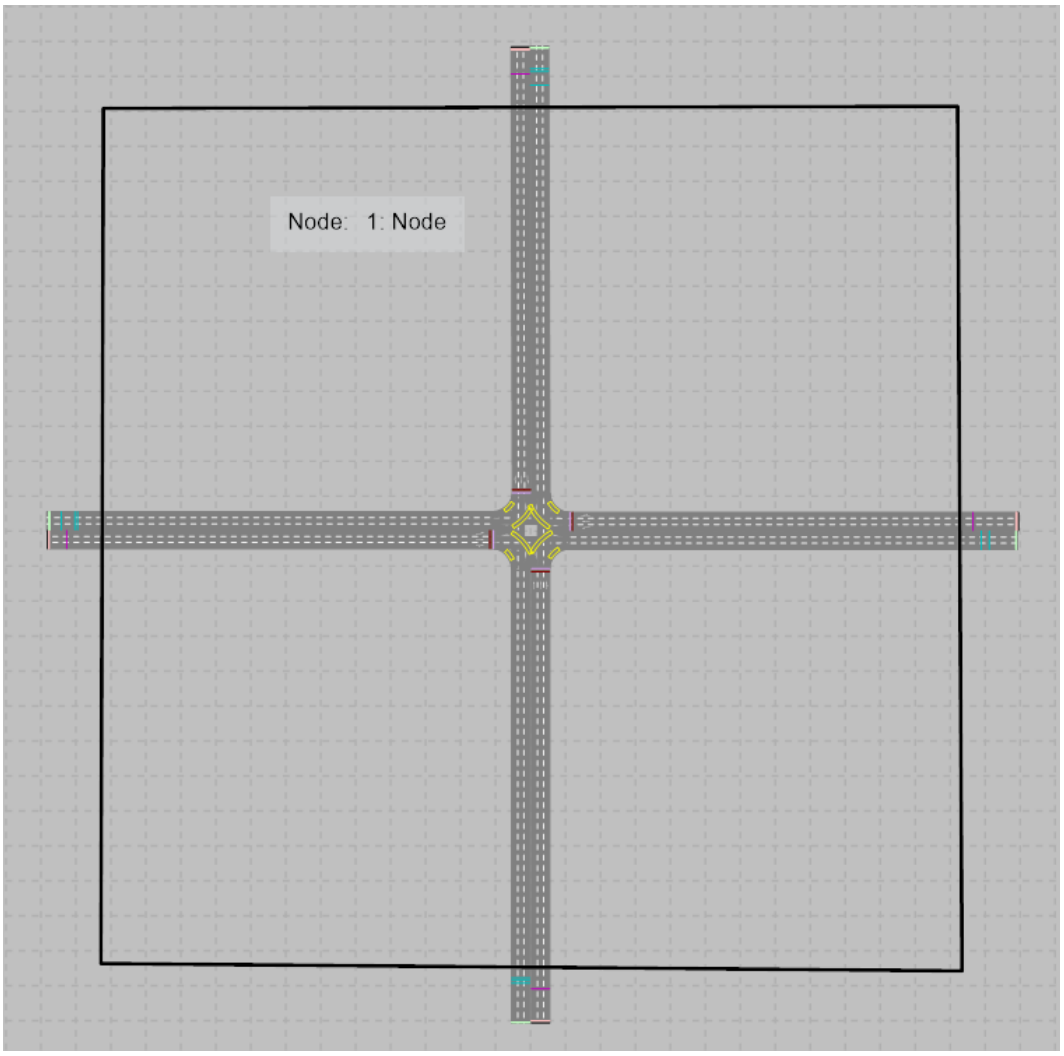


Figure 5.17: The Single Cross Triple model

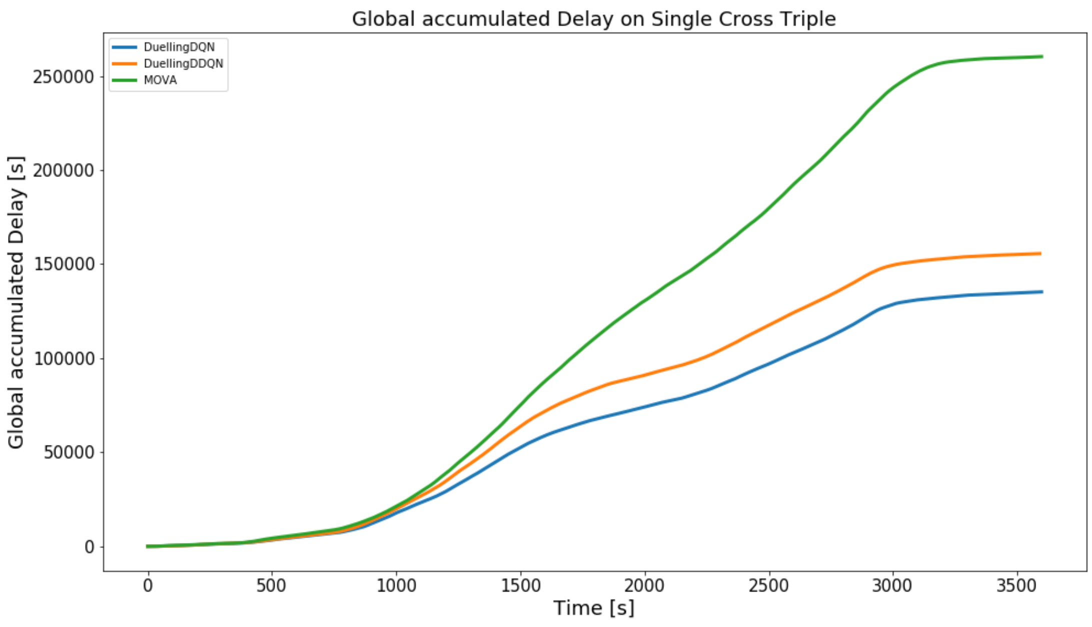


Figure 5.18: Global Cumulative Delay in Single Cross Triple - 4 actions.

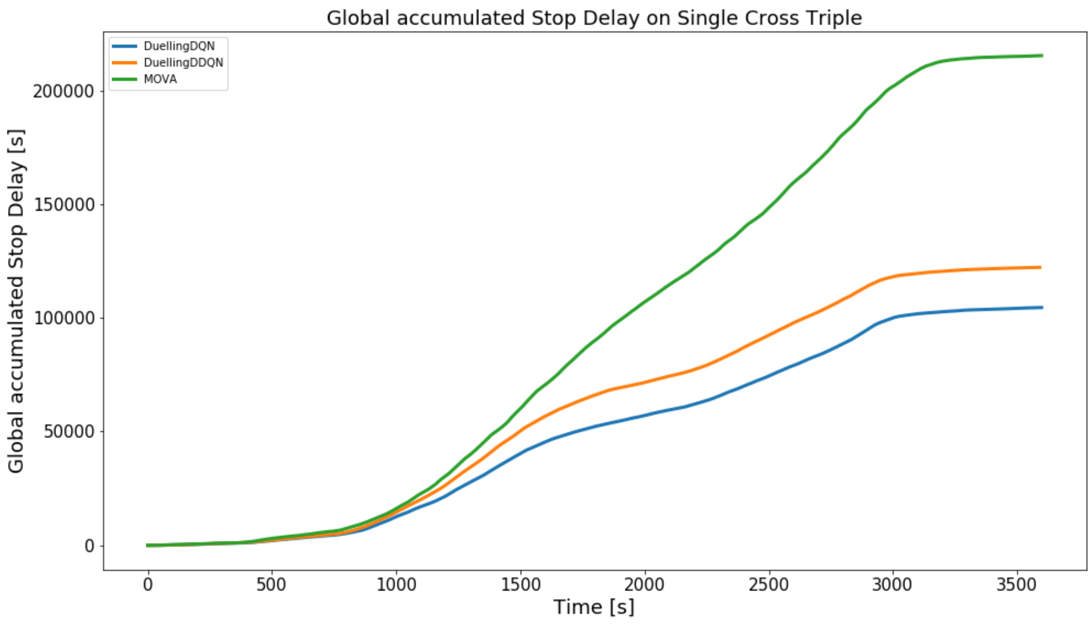


Figure 5.19: Global Cumulative Stop Delay in Single Cross Triple - 4 actions.

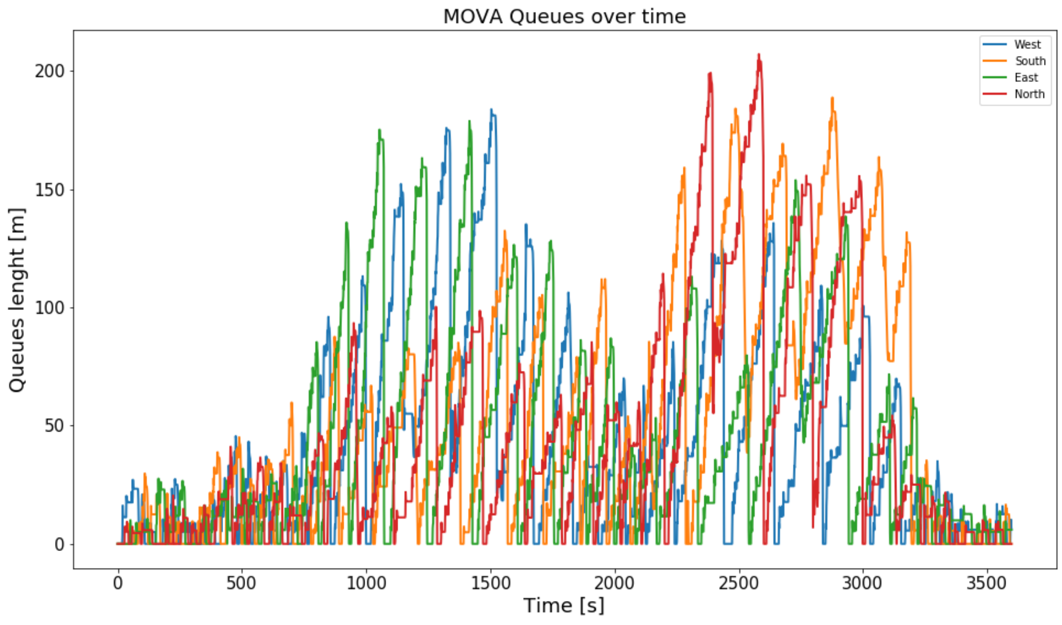


Figure 5.20: Aggregated queues over time in the upstream direction from the intersection using the MOVA UTC.

agents in more complex situations.

Here, the fluctuations in neighbouring junctions will create variations over time in the amount of vehicles that arrive to the traffic lights. Further, all the previous models had cars inserted to the model in the same link that they would be once they reach the intersection. This implies that the arrival times of the vehicles followed exactly the same distribution of the input methods, in this case a Poisson Point Process with variable (over time) mean arrival rates. In this case, those intersections in the fringes of the map will still see vehicles arriving following the same distribution as above, however, the central intersection will have to deal vehicles arriving in platoons and with a variable arrival rates, since these will be dictated by the operation and efficiency of those UTCs located upstream from the central intersection.

5.7.1 Experiment 4: Five Intersections - 8 actions

The test conducted on the Five Intersection map focused on how easily learning can be transferred and extrapolated by RL agents in the context of TSC and on measuring the benefits of previous not fully adequate learning, when compared with

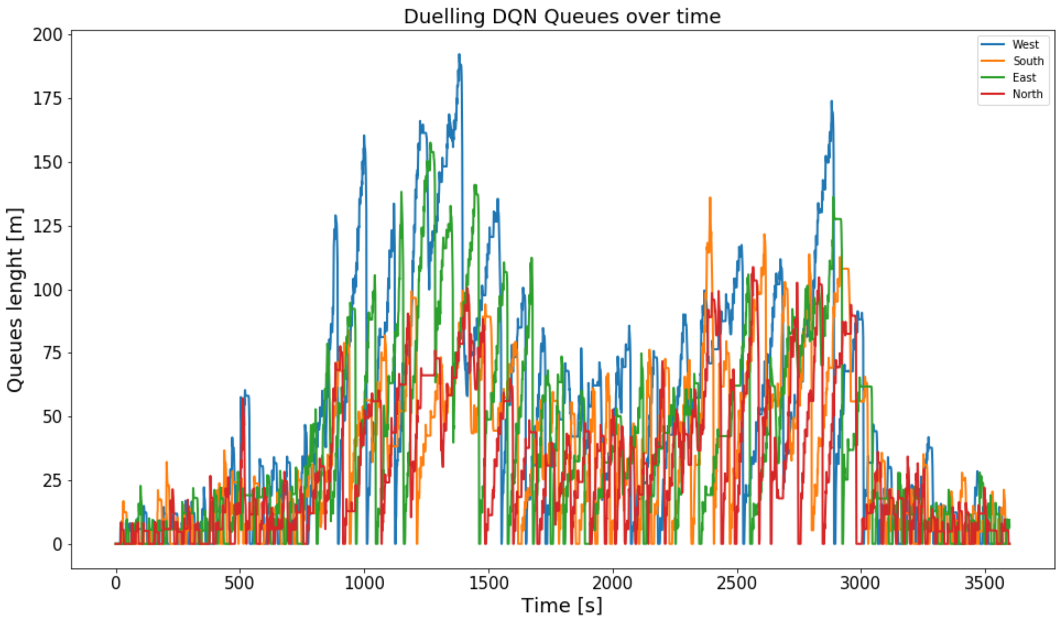


Figure 5.21: Aggregated queues over time in the upstream direction from the intersection using the DuelingDQN UTC.

learning directly on the job. To this end, a zone controller composed of 5 copies of the DuelingDDQN agent used in the Experiment 3 of this chapter was configured. No further training was performed in this configuration prior to the evaluation. A second controller, composed of 5 DuelingDDQN agents was trained directly on the intersection. Their performance is compared against that of the corresponding MOVA UTC.

The mean arrivals over time follow a distribution with the same shape as in the previous experiments, using a scaled demand to account for the increased size of the network. The test is conducted with an average demand of 5920 vehicles/hour (roughly 5 times the demand of the previous model), including 2 peaks with arrival rates of 8000 vehicles/hour, lasting 6 minutes each. A detailed description of the demand per input point and time period can be found in Table 5.9.

The turning ratios were set such that upon arriving at each intersection, 20% of vehicles will perform an offside turn while being in the corresponding lane, 20% of vehicles will use the nearside turn lane to perform said turn, 20% of vehicles will go straight while sharing the nearside lane with the turning vehicles, and the remaining 40% will go straight while using the central lane.

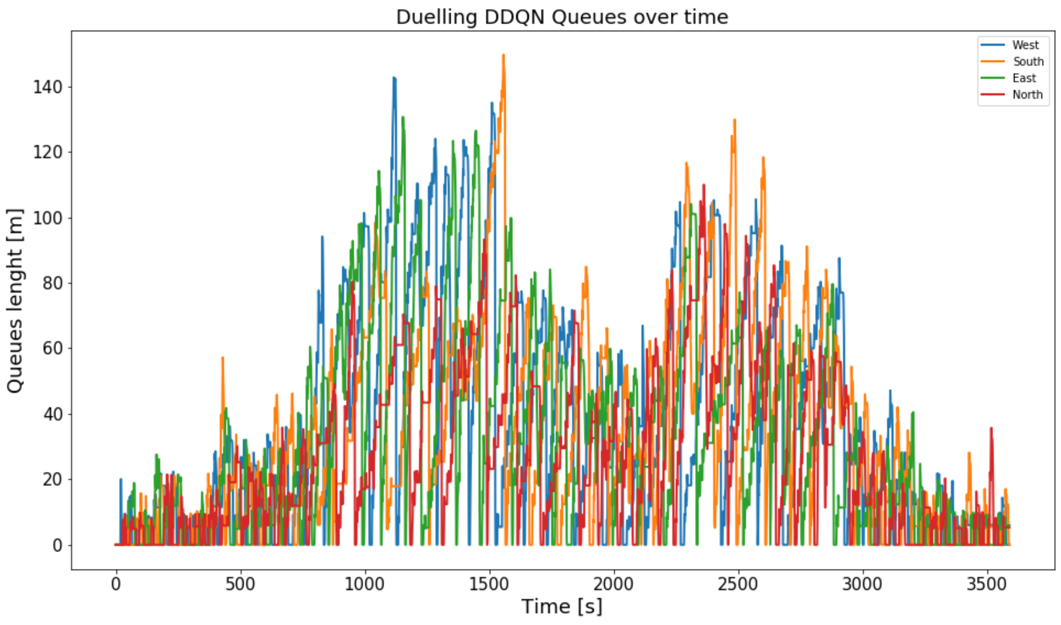


Figure 5.22: Aggregated queues over time in the upstream direction from the intersection using the DuelingDDQN UTC.

Table 5.9: Demand in vehicles/hour inserted in the model for each input ID.

Period / Input	1	2	3	4	5	6	7	8	9	10	11	12
min 0-6	200	200	200	200	200	200	200	200	200	200	200	200
min 6-12	400	400	400	400	400	400	400	400	400	400	400	400
min 12-18	500	900	500	500	900	500	500	900	500	500	900	500
min 18-24	500	1000	500	500	1000	500	500	1000	500	500	1000	500
min 24-30	500	700	500	500	700	500	500	700	500	500	700	500
min 30-36	500	700	500	500	700	500	500	700	500	500	700	500
min 36-42	500	1000	500	500	1000	500	500	1000	500	500	1000	500
min 42-48	500	900	500	500	900	500	500	900	500	500	900	500
min 48-54	400	400	400	400	400	400	400	400	400	400	400	400
min 54-60	200	200	200	200	200	200	200	200	200	200	200	200

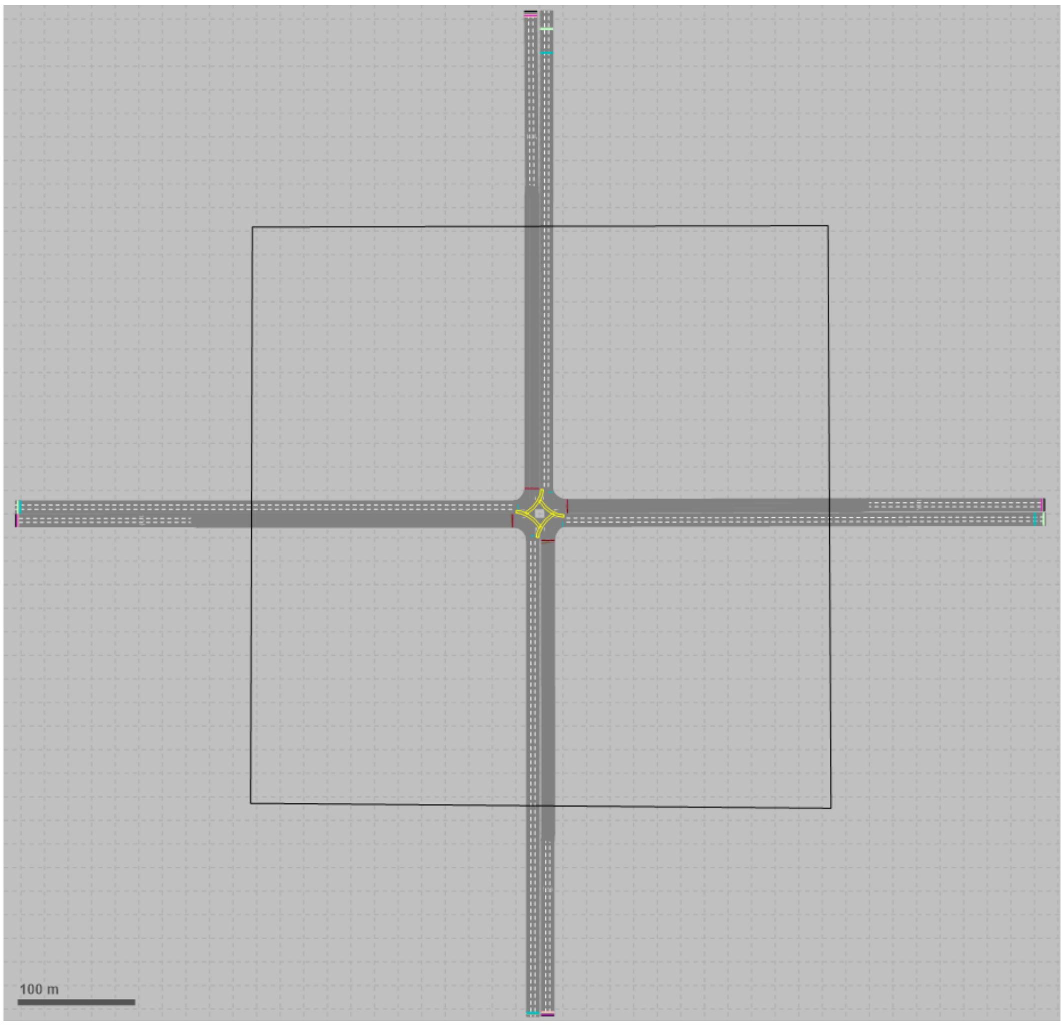


Figure 5.23: Modified Single Cross Triple for 8 actions.

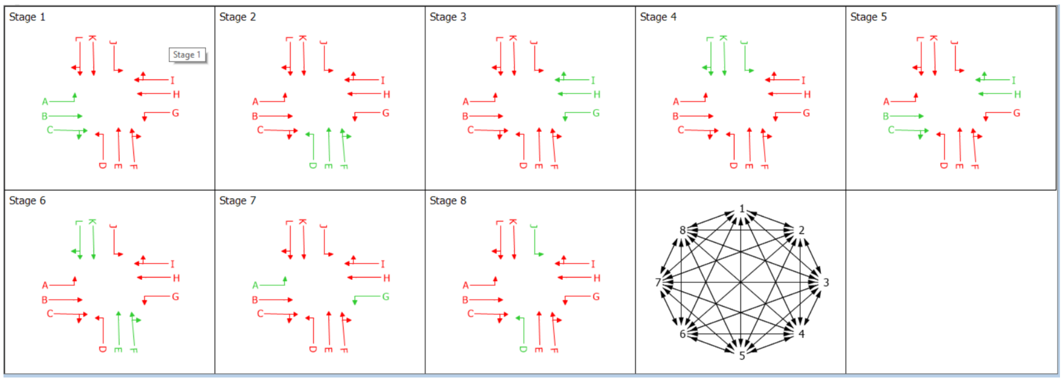


Figure 5.24: Allowed stages of the Single Cross Triple model and allowed transitions between stages.

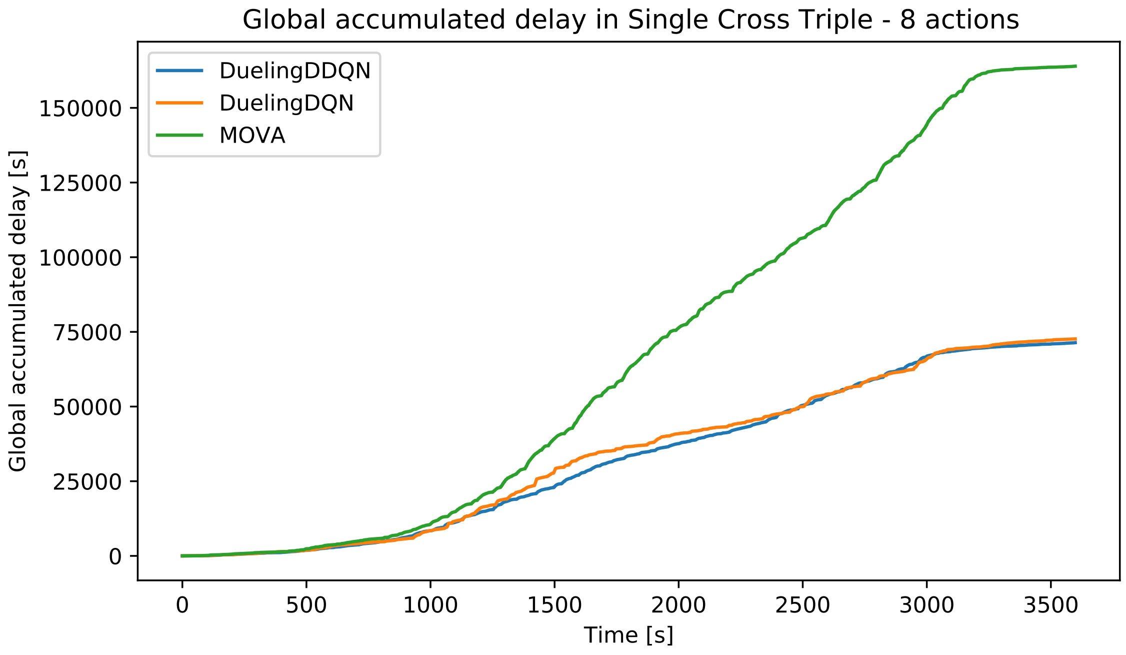


Figure 5.25: Global Cumulative Delay in Single Cross Triple - 8 actions.

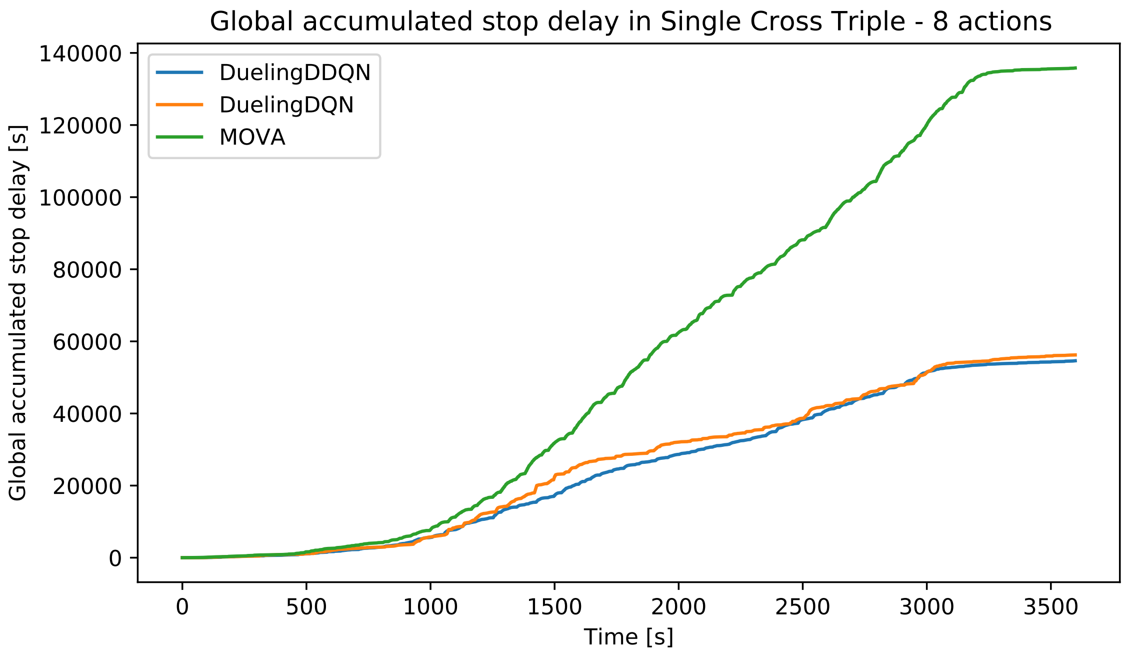


Figure 5.26: Global Cumulative Stop Delay in Single Cross Triple - 8 actions.

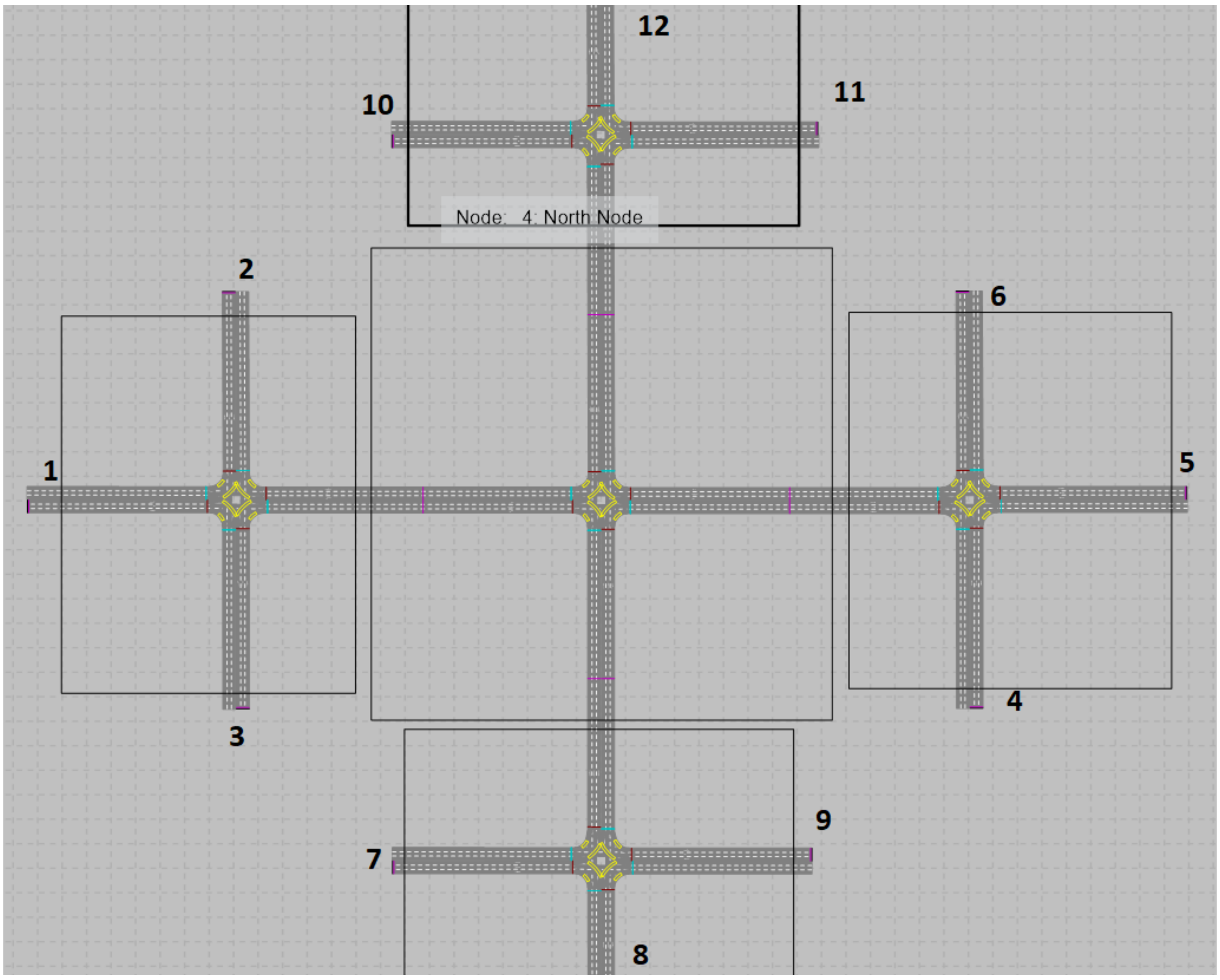


Figure 5.27: The Five Intersection model with IDs for vehicle input points.

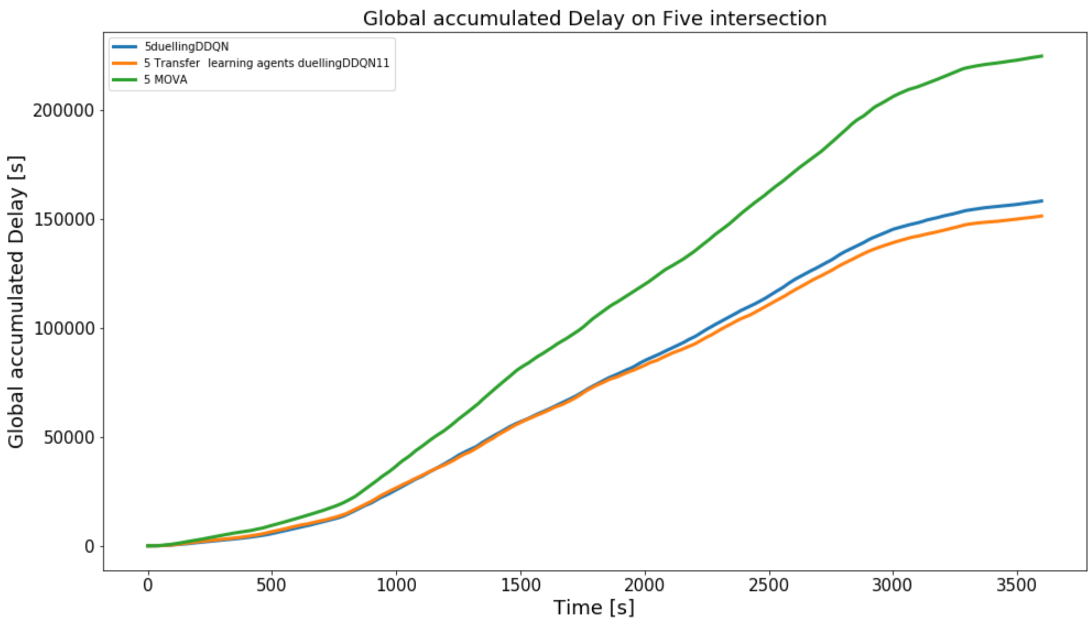


Figure 5.28: Global Cumulative Delay in Five Intersections - 8 actions

During training, it was observed that the RL agents would have been able to handle more demand (over 12000 vehicles/hour), but the level tested was near the saturation levels of MOVA, so testing on higher loads was not feasible.

Table 5.10: Cumulative Delay and Cumulative Stop Delay in seconds across Controllers on Five Intersections map.

Controller	**Cumulative Delay**	**Cumulative Stop Delay**
MOVA	224687.90	159965.86
DDDQN TL	151306.23	86958.68
DDDQN	158206.40	94800.08

In the results presented in Figs. 5.28 and 5.29 we can observe that the DQN agents still have better performance, generating lower cumulative delay and cumulative stop delay than MOVA. The difference in performance between both methods is reduced when compared to the prior experiments and taking into account the extra size of the network, it can be argued that the lower performance of RL agents is due especially to the middle intersection. Since the flows observed by this middle intersection depend on the performance of the neighbouring intersections (traffic will

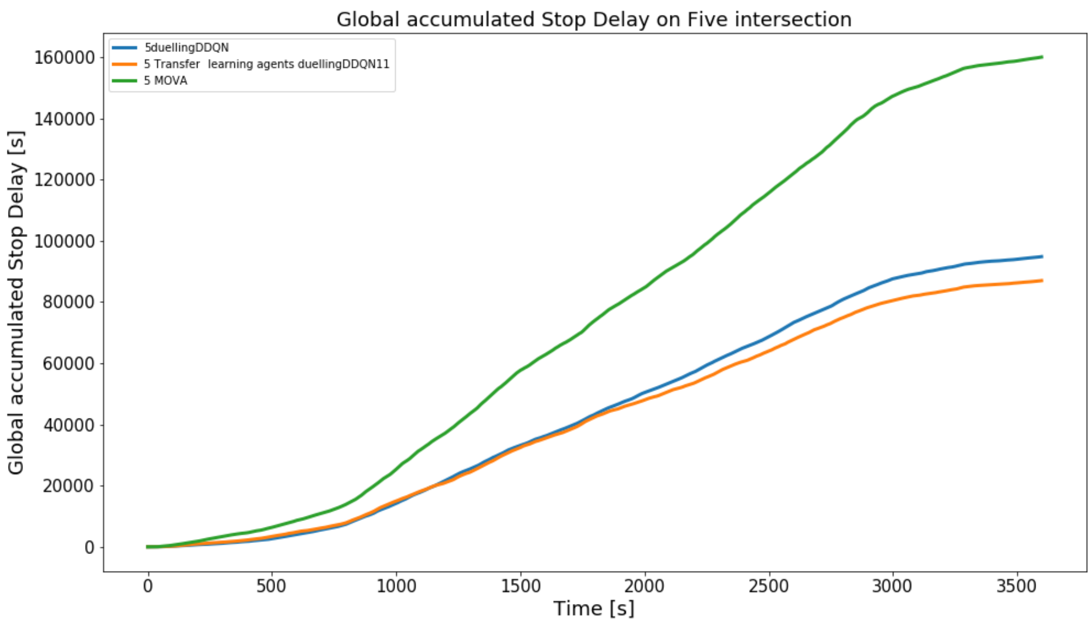


Figure 5.29: Global Cumulative Stop Delay in Five Intersections - 8 actions

only reach it after being given way by one of the other controllers), both the state variables observed, as well as the expected reward response to the agent's actions will not stabilise until the provided demand profiles by the surrounding controllers stabilise too. In turn, this will only occur when said surrounding controllers are sufficiently trained. Based on this, it can be argued that since both DQN-based controllers received the same amount of training, but having the agents controlling the surrounding intersections fully trained will have a beneficial effect on that training of the central agent, the 5 controllers trained on site could benefit of more extended training.

Less surprisingly that it would appear at first, it is found that the agent previously trained on a single intersection obtains marginally better results than the agent that learnt on the job. While theoretically it will always be best for an agent to have at least part of its training period spent on the task it will have to perform, the value of pre-training an algorithm is widely accepted in the ML and RL communities. However, pre-training on close to operational conditions will have, in principle and disregarding the effects of randomness in the training, more utility to the learning process of the agent, since it is an equal task to what the agent needs to carry out rather than close to it.

While these results underline the utility of pre-training RL TSC agents in problems that encapsulate only part of the challenges that they will face in the final problem, it can be argued that a pre-trained agent that later received some final fine-tuning on the complete model could achieve even better performance, based on the exposition over the previous two paragraphs. In this sense, both DQN-based agents here presented can be considered under trained with respect to an optimally trained agent: the transfer learning agent due to lack of on-the-job experience, and the agent trained on site due to a lack of extended training to offset the disrupted flows experienced by the central agent while the surrounding agents train. As it will be seen in the following section, the use of pre-training has limits, and the noise and oscillations that pre-trained agents can induce in a sufficiently complex system, are enough to steer the system towards catastrophic failure.

Lastly, and regarding robustness, RL UTCs are inherently robust to lanes being closed, since a well-trained agent will not serve a lane for which no demand is perceived. Beyond this, and while well trained DRL agents are great at extrapolating in situations outside of their training envelope, before any deployment in the real world, the agents should still go through an entire robustness assessment process. While the description and detailing of such process is beyond the scope of the research here presented, it should cover good operation under all edge-cases, and ensure the safety limitations as defined by the competent authorities.

5.8 Balance Network

The last experiment was conducted in the *Balance Network*, which is the testbed that PTV uses to showcase the power of their network-wide adaptive timing optimiser called *Balance* [46]. Data on its internal workings is not widely available, it is based on a two-layer approach, the first of which uses a microscopic simulation to derive densities for each available route, a second mesoscopic simulation evaluates the performance of the network and further optimises the green time splits.

The network, a model of a real urban network, is composed of 14 intersections of various configurations, using between 3 and 14 managed lanes.

5.8.1 Scenario Description and Experimental Setup

Due to the issues introduced above regarding obtaining queues from individual lanes, the model was reworked in a similar fashion of that in Experiments 3 and 4. Links with more than one lane were allowed space for merging and changing lanes, and were then partitioned in individual links of a single lane, so data could be retrieved.

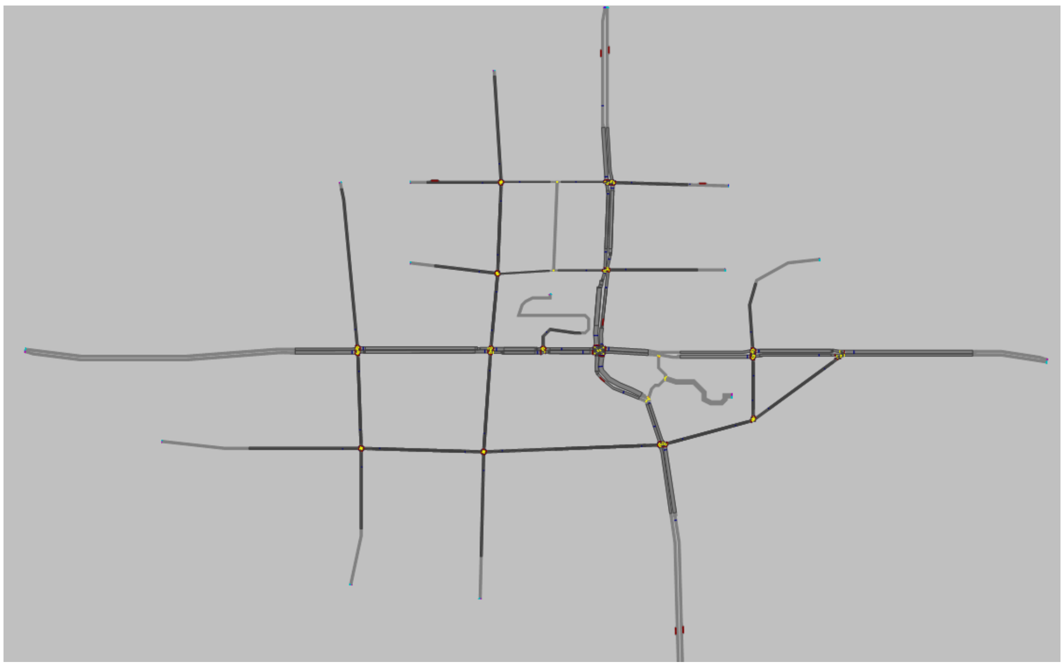


Figure 5.30: The Balance model

The demand profile on the network has been set by the traffic engineers in PTV to accurately represent the traffic situation at the real-world location of the network. The simulation covers 2.5 hours during the morning rush. It starts with a total average demand of of 4009.4 vehicles/hour distributed between all edges in the boundaries of the simulation. On the peak of demand, the simulation operates for 1 hour attempting to serve a demand of 7978.1 vehicles/hour.

There was an initial attempt at training either all, or part of the RL agents directly on the network. This quickly turned unfeasible, as the amount of data that the agents and the simulator exchange is increased to a point that the simulation grinds to a halt, obtaining simulating speeds of $1-2\times$ real time. Since the agents require around 400 hours of simulated training time to learn the task, and there are no a priori guarantees of convergence, an alternative solution was searched.

5.8.2 Experiment 5: Balance Network

As an alternative, all intersections were pre-trained in an isolated setting. This involved manually recalculating all possible routes in the simulation (combination between all entry links with all exit links including alternatives through longer but less used routes), in order to obtain the demand that each individual intersection goes through during the simulation and the turning ratios in every link.

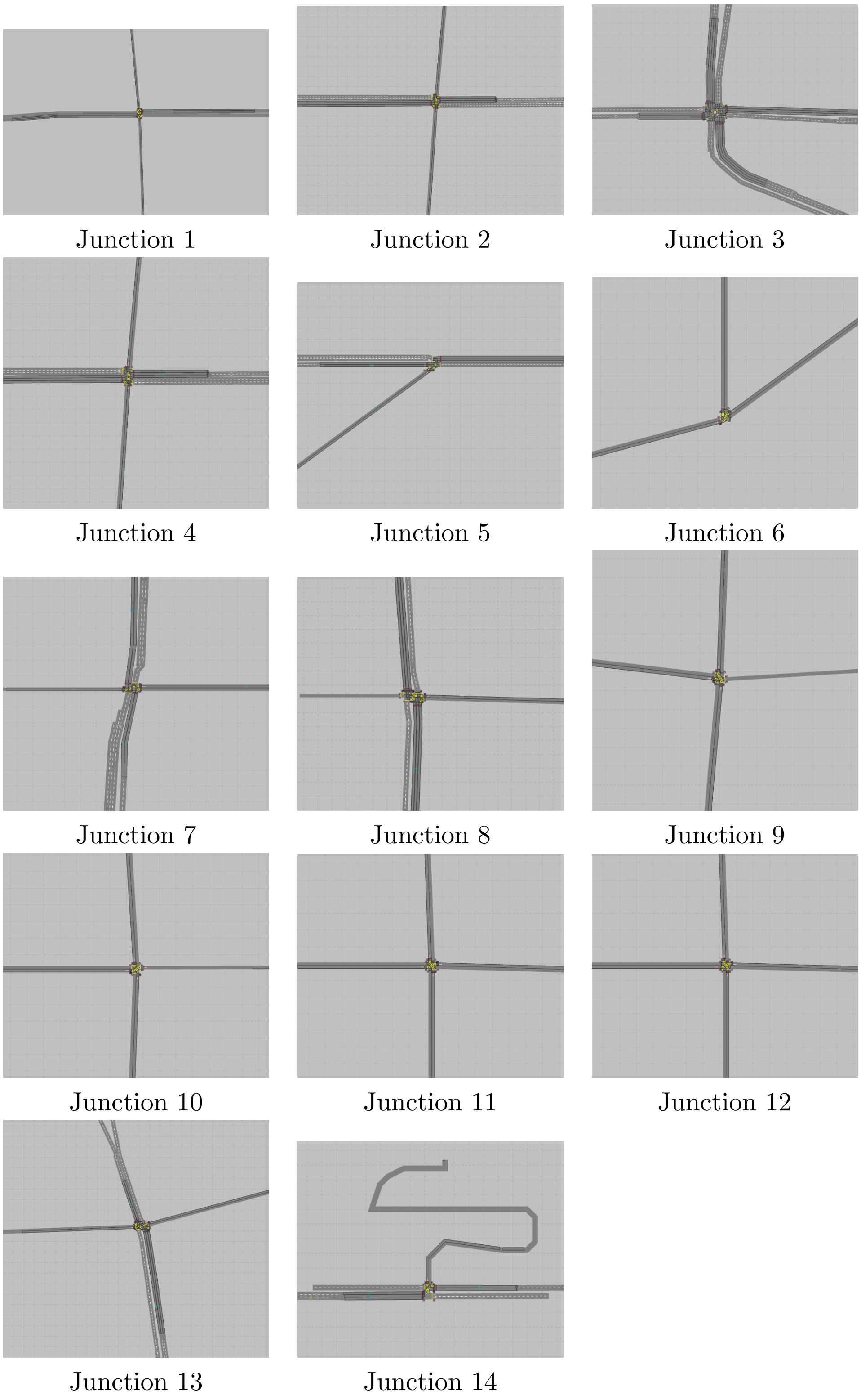


Figure 5.31: Partitioning of the Balance Network and individual junction status after rework.

While individual agents on isolated junctions were able to converge without issue to policies providing good local control, once the agents were recombined in an adapted version of the complete network, they suffered the same earlier issues as the agents managing the UTCs in the *Five Intersections* map, regarding the instabilities in their performance.

Instabilities, backed-up queues, and disrupted flows and demand profiles propagated to the network over time, easily triggering catastrophic breakdowns, especially on the links between junctions 2, 3 and 14, as it can be observed in Fig. 5.32, that was beyond the RL agents' ability to recover.

This was aggravated by the discrepancies of inter-junction distances in the models of individual intersections when re-integrated into the network map. When the network was initially partitioned into independent intersections, and the links directly upstream from the intersections were partitioned in independent links (covering a lane each to allow for retrieval of per-lane state representations), the links directly upstream of those were stretched to allow space for vehicle inputs into the model and lane changing. This extra space was meant to offset the fact that once a link has been partitioned into their own per-lane links, lane changing between them is not possible. Due to this extra space, these problems did not arise until the agents were reintegrated into the network map for joint control.

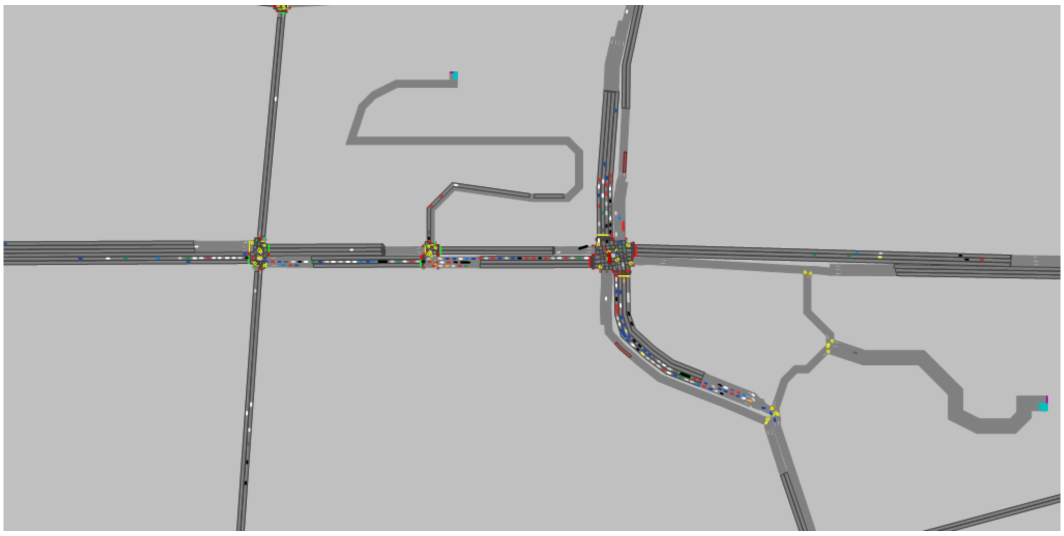


Figure 5.32: Example of irrecoverable catastrophic failure in the area of junctions 2 (left), 3 (right) and 14 (centre)

This collapse in performance is mainly attributable to a few clustered controllers, although all of them will provide suboptimal responses given the previously mentioned disruptions to the demand profiles. Here, only a few failing controllers are enough to prevent the system from exerting effective control. The main reasons for the catastrophic failure can be observed in Fig. 5.32, in which we can see that the vehicles approximating junctions 3 and 14 from the West and South are attempting to change lanes in a situation where there is no space for this (the blinkers can be seen activated in the vehicle models indicating an intention to change lanes). This lack of space caused individual drivers to stop and block traffic, rather than take an undesired route, causing traffic to back up and overflow into the junctions upstream, creating a cascading effect that could not be recovered from. This is clearly an effect of the modifications that were necessary to perform on the original map, as mentioned earlier, in order to make feasible the introduction of RL controllers that were capable of using all the available traffic stages by using per-lane variables instead of per-link variables, while running at high speeds.

The catastrophic jam around junction 14 is further explained by the fact that the main part of the state representation available to these agents is a vector of queue lengths, and the reward is the negative sum of queues. Given the lack of an inbuilt mechanism to force the agents to serve each line at least once per some predefined period, these representations can cause agents to disregard sustained queues in a single short lane to serve lanes with lower waiting times overall, but longer queues. While this is generally not an issue on junctions with balanced lengths for their incoming links, as in all the previous examples in this chapter, it becomes important in the case of junctions 3 and 14, where it can be seen how the lanes causing the breakdown are all among the shorter ones of those being served. An example of this can also be observed in Fig. 5.32, where the short incoming inside lane to junction 14 is saturated, while any other lanes that would be served with it (remaining West lanes and all West lanes) are empty, while there is a significant demand, in comparison, present in the North-South direction covering the lanes that cross the junction without turning. As a mitigation strategy, it was attempted to change the reward function of the offending agents to one that penalised more longer queues, by using the negative sum of squared queues as a reward function, however this was proven to heavily impact agent convergence rates during training and had to be abandoned due to lack of results and high computational demands of re-training.

Although it can be argued that different state representations could give the agents better quality or less myopic observations about the traffic flow, this alone would not solve the problem as the agents have been able, while using their current state representation, of dealing with all of the previously tested junctions

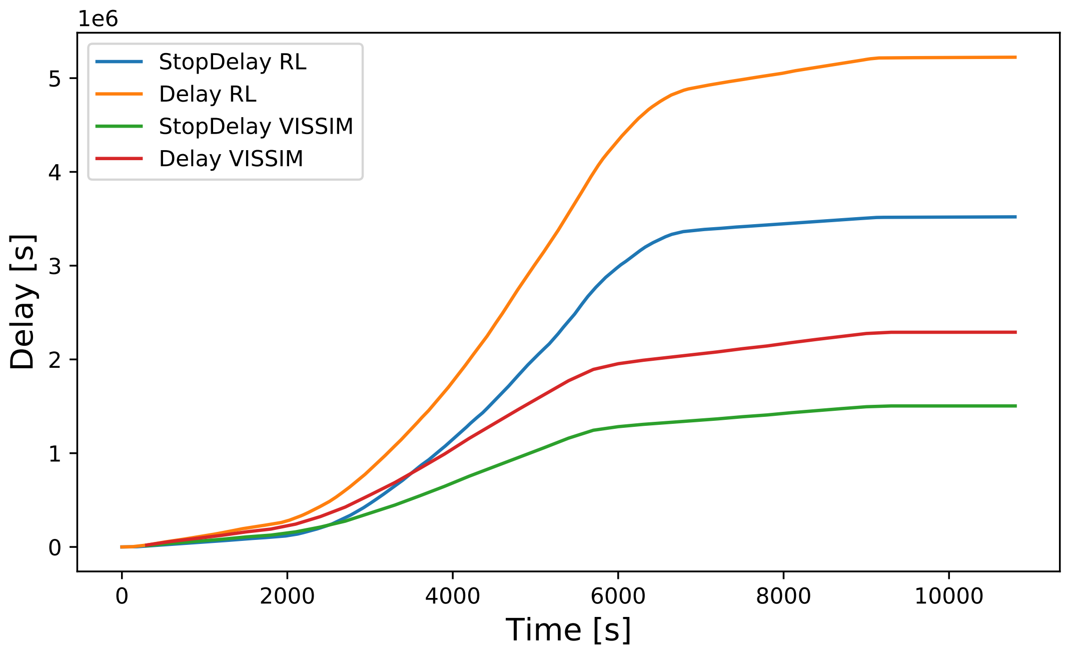


Figure 5.33: Global Cumulative Delay over time and Global Stop Delay for DuelingDDQN Agents and Balance UTCs.

in an isolated manner, showing that this representation can generate agents that exert reasonable control and indicating that while this can be one of the issues present, it is not inherently causing the collapse. There is another argument about how creating inbuilt limitations about regarding minimum frequencies with which to serve each group of lanes could be beneficial. While this is true in this specific failure case, and would be required by transport authorities in the case of a live deployment to address some robustness concerns, taking RL TSC closer to earlier methods, it would also fail to solve the problem regarding lane-changing that is the main root of issues in this case. Other options involving further modifying the map geometry, would be taking the map even further away from its original counterpart and further degrading the ability to perform accurate comparisons.

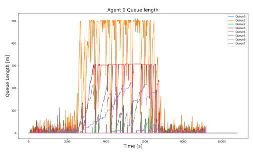
Figure 5.33 shows the aggregated performance of all the agents during the whole simulation in terms of delay and stop delay. In it, it can be observed that during approximately the first hour if simulation, RL agents outperform the Balance controller in terms of stop delay. This means that while RL agents were assigning the available green time in a more effective manner than the Balance controller, they were under performing in terms of inter-junction coordination and green wave generation, leading to higher delay overall. This behaviour and their under-performance in terms of coordination was expected, since Balance specifically optimises to obtain this coordination, and experimentation was halted due to the issues here described regarding map geometry before the RL agents were extended to allow for inter-agent communication.

In Figure 5.34 it can be seen how, in this specific case, the agents 1, 2, 3, and 14 were not able to prevent collapse from happening in their junctions. The agents have been found to be highly dependent on the performance of their neighbours. This makes the most intuitive solution, re-training those agents under performing and testing again, less useful than it would seem, since they often cause a third controller that was operating fine to fail and require further retraining. This approach did not cause any performance improvement in the area of agents 2-14-3, always leading to a collapse due to an overflow caused by the vehicles attempting to access the innermost lane in the West link of junction 3. After several such cycles, this approach was abandoned due to its high computational demands and diminishing returns.

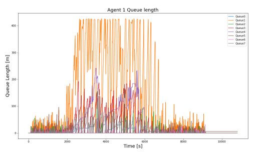
Some of the issues here presented could be overcome, such as preventing the collapse of the agent 1 by training it jointly with agent 2, or by successfully training while using a non-linear combination of queues that penalise more heavily those that are longer with respect to their capacity. However, this would only involve a mitigation of the main issue: the required modifications to the map due to the inability of the simulator to provide per-lane information to the agents while running at speeds that are acceptable for DRL. The solution to this main issue would be to integrate the agents with a different microscopic traffic simulator that can provide the required measurements, such as SUMO, which is left for future work.

5.9 Results and Discussion

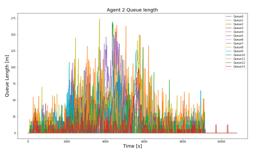
In this chapter, several network architectures for RL UTCs were tested. It was found that agents do not require extensive or complex neural networks to be able to exert adequate control over traffic junctions. In terms of training, it was most effective to carefully tune learning rate, exploration rate, minibatch sizes and frequency and amount of learning performed. More training episodes were not necessarily better, having a heavy dependence on hyperparameter values. This in not surprising given the range of prior application areas, but still is an important point for future development of these algorithms. The Reinforcement Learning agents also showed great stability and robustness to control situations outside of their training envelope.



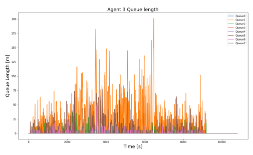
Agent 1



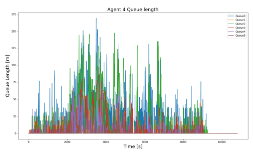
Agent 2



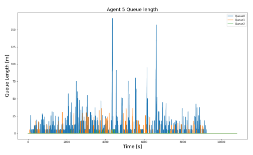
Agent 3



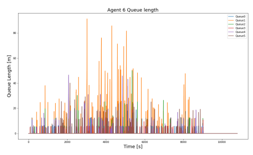
Agent 4



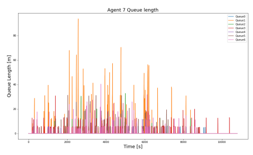
Agent 5



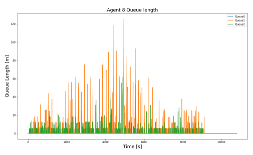
Agent 6



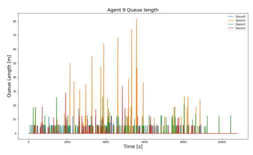
Agent 7



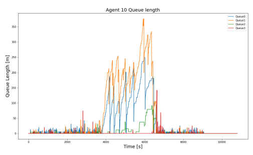
Agent 8



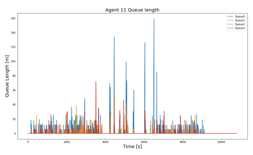
Agent 9



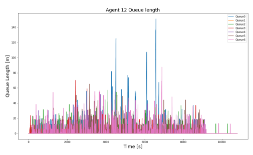
Agent 10



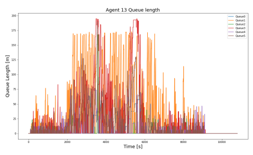
Agent 11



Agent 12



Agent 13



Agent 14

Figure 5.34: Queues over time in the 14 DuelingDDQN agents running simultaneously in the Balance network.

They also show an ability to extrapolate to deal with situation never seen during training. Additionally, it is rather impressive that agents trained on relatively low uniform demand can perform better than commercial systems during evaluation tests that included variable demand several times higher than anything experienced before (e.g. the Single Cross Triple agents were trained on a 300 vehicles/hour per link, while it successfully handled a demand of 4500 cars/hour during evaluation).

Experiment 1 provided evidence that fixed time systems perform worse than adaptive and RL UTCs in simple junctions. In this case, MOVA and the RL agent following a DuelingDDQN architecture obtained very similar results, with a slight advantage for the RL agent.

Experiment 2 provided similar evidence about a smaller number of controllers, in a situation where queues were measured on a per-link basis rather than per-lane. This implies less granularity in the data and makes the control task more challenging. The results followed the same pattern with a DuelingDDQN agent obtaining the best performance, despite the lower quality of the input data.

Experiment 3 required modifications of the map in order to obtain said per-lane queues. This model saw the introduction of a much more complex junction, with a multitude of actions available to the agent, some of them serving the same lanes in different ways, and letting the agent decide on the sequence. Once again RL agents obtained better results than MOVA, with the DuelingDDQN agent obtaining the lowest global and stop delay. The gap between the performance of MOVA and RL agents is increased here with respect to the last experiment. Most likely reasons are higher granularity in the data and extra actions being available to the agent, allowing it to display more complex sequences of actions.

Experiment 4 saw the introduction of a multi-intersection and tested UTCs that were trained on the job and others that were transferring the knowledge that they had acquired while performing a similar task. The gap in performance between MOVA and DQN agents was once again reduced, but RL methods still obtained the best performance. The results from this experiment illustrates well how difficult it is to train sequences of agents that will influence each other. While the agents on the fringes were able to exert reasonable control, performance towards the middle intersection, where the traffic conditions are different. The fact that the middle intersection will not receive a normal demand profile until the intersections upstream from it have been sufficiently trained, and thus the general principle that learning for agents acting in series cannot effectively happen until those agents upstream from it have sufficient knowledge to provide with accurate and stable learning environments, both stem from here.

Experiment 5 saw an attempt to individually train and then integrate a multitude of agents that had been initially pre-trained individually. The approach was a failure, precisely due to the knock-on-effects indicated about Experiment 4, together with the modifications that were required for the map. The experiment resulted a failure in which the RL agents were displaying a joint performance much lower than what would be expected given their individual records. Given the general performance of the model, such an approach would require further joint training of all the agents, which is currently not technically feasible within the scope of this project.

Reinforcement Learning applied to UTC could be a real life solution to improve traffic conditions in urban environments, even though the sensors needed are a bit more sophisticated than simple induction loop and there is still work to find measures against unlawful cars and to find solutions for more fairness in green time allocation. Nevertheless, UTC is not the only solution to traffic congestion, better road networks can be designed as the capacity (veh/hour) is limited regardless of how good is the UTC. The future urban transportation could be personal car-less with the development of public transportation and self driving cars. But meanwhile, these experiments show that Reinforcement Learning based UTC could be the next generation solution for reducing traffic congestion.

5.10 Limitations and Future Work

The results presented in this chapter suffer from several limitations. It was not possible during research to find a suitable parametrisation of the A2C UTC such that complex junctions could be controlled, although there exists evidence that this is indeed possible. The hyperparameters for the agents were not exhaustively fine tuned, and more work into learning, exploration and discount rates could easily improve the performance of all the agents. While different architectures for the neural network have been tested, this is limited to great changes in what the different layers are supposed to do. This testing did not include an optimisation step over the size, depth and width of the neural network used as approximator, while recent research suggests that neural topology greatly affects task performance.

Further than this, only one state and reward have been tested. Although there exists evidence that much more complex state representations yield diminishing returns, the state space used here is rather limited, and could be improved to give the agent with marginally better representations of the system it is trying to control. Exploration of the effect of potential different reward functions is left as future work.

This piece of research has clearly underlined the current limitations of agents that will produce decisions in series over a continuous object, and how partially trained agents create disturbances in the state variables that propagate upstream from them through the network. These disruption waves can be extremely damaging and entirely prevent learning from happening, leading to a blockage of the simulation. Agents trained in such manner should have extended training phases once integrated to ensure harmonious joint operation. Private companies addressing similar issues for the same problem have found this approach to partially or entirely remove this issue.

The multi-agent networks could benefit from either local communication with neighbours so state information can be exchanged, or a reformulation of the reward functions, making them based on local and neighbouring information that is relevant to the performance of the entire network, searching for a more bottom-to-top approach in which local clearance rules can provide emerging control for a wider network than any of the parts receives information from.

In addition, the capabilities and limitations of a given simulator need to be assessed in full before undertaking such lengthy projects. In this chapter, as the junctions tested became more complex, the maps required progressively heavier adaptations in order to perform the required experimentation. This process ended with the conclusion that it would not be possible to train the Balance Network without heavy changes that, in themselves, prevented the experimentation from being feasible as it was discovered later. In this sense, and including Vissim up to version 20, its use is not recommended for situations in which per-lane measurements are needed, where other simulators should be used (e.g. SUMO).

Lastly, other baselines such as SCOOT or TRANSYT should be compared in this setting, especially in the case of multiple junctions, as MOVA is not designed to coordinate the traffic lights. Nevertheless, those multiple junctions systems require great configuration efforts, specifically designed databases, and a great amount of specialist knowledge that is not openly available.

CHAPTER 6

Assessment of Reward Function Choices for Reinforcement Learning Agents in Real-World Isolated Intersections

6.1 Introduction

Reinforcement learning (RL) has been investigated as a potential next step in urban traffic control (UTC) systems, demonstrating the potential to outperform even well-calibrated systems currently in use [199], such as 'Microprocessor Optimised Vehicle Actuation' (MOVA) for isolated junctions, and 'Split, Cycle and Offset Optimisation technique' (SCOOT) for regions of up to 30 signalised junctions. However, most of the work to date is not intended to be directly applied to the real world. As such, all observed works in the literature overlook operational limitations of this application of RL. Further, there is a gap in the literature regarding the choice of reward function for such an RL system, which is a critical aspect. This chapter provides a robust comparison of reward functions for RL, in the context of a junction in Greater Manchester, UK, a simulation of which has been calibrated using extensive data from Vivacity Labs vision-based sensors. This research is directly translatable to real-world applications of the technology, and has since been deployed to manage real traffic.

The chapter is structured as follows: Section 6.2 reviews earlier work in the field and enumerates different reward functions used in the literature. Section 6.3 states the mathematical problem and the Reinforcement Learning theoretical background. Section 6.4 describes the implementation and characteristics of the agents and environment. Section 6.5 describes and provides the analytic expressions of the different reward functions being tested. Section 6.6 gives detail on the training, selection and evaluation of the agents. Section 6.7 shows the results of the experiments in terms of average waiting time of the vehicles.

6.2 Related Work

Previous studies have considered RL for UTC without focusing specifically on the choice of reward functions. Initial research was centered around proof-of-concept, with studies such as Wiering [117] and Abdulhai et al. [200] advocating for its potential use, and the ability of Q-Learning to perform better [201, 202] than traditional UTC methods such as MOVA [45], SCOOT [51] and SCATS [54]. Later research looked into neural networks as a function approximator to estimate the value of state-action pairs whilst addressing discretisation issues raised previously [118, 203]. Recent research makes use of deep RL to estimate the state-action values for each state [188, 123, 127, 124, 190, 126, 204] or to learn a policy directly that maps states to actions [118, 125, 128, 130, 205].

A number of publications have listed available RL methods for TSC. In [122], early table based methods are summarised; in [188] reward functions in a multi-junction network (delay between actions, difference in delay between actions, minimisation and balancing of queues, and minimising stops) are compared, and in [128] three different state representations are compared, finding similar performance with each. As the outcome was found not to be sensitive to state representation, the present work keeps the state representation constant. More recent studies, such as Yau, K. L. A., Qadir, J., Khoo, H. L., Ling, M. H., & Komisarczuk [206] focus on different RL approaches to this problem, while Wei et al. [207] consider state representation, reward function, action definition and model specific distinctions (online-offline, policy-value, tabular-function approximation) in a survey across the field, but without performing comparisons. Previous work typically approximates in terms of road geometry, traffic demand, and operational constraints, creating models of intersections that preclude real-world applicability.

This points to a gap in the literature of directly comparing a broad range of reward functions, in a well-calibrated, geometrically-accurate simulation which accounts for real-world limitations (e.g. safety constraints): this is the topic of this chapter.

6.3 Problem Background and Definition

6.3.1 Reinforcement Learning

An intelligent agent is an entity which acts towards a goal based on observations and a decision making process. RL is an area of Machine Learning which focuses on how agents can learn a policy π based on extended interaction with an environment. This approach treats the problem as a Markov Decision Process (MDP), defined in terms of the $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathrm{R}, \gamma >$ tuple (States, Actions, Transition Function, Rewards, Discount Factor), in accordance to what is shown in Fig. 6.1.

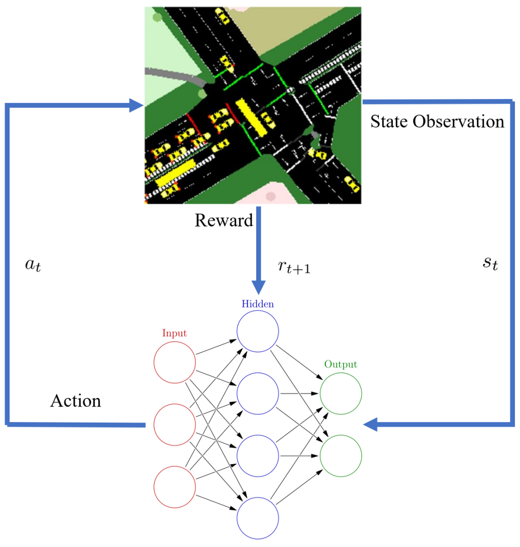


Figure 6.1: Schematic representation of information flows between Environment and agent in a Reinforcement Learning framework.

The goal of the agent is to learn an optimal policy π^* that maximises the expected future reward. The discounted future reward at time t, R_t, is defined in equation (6.1).

$$R_t = \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right] \tag{6.1}$$

6.3.2 Q-Learning and Deep Q-Learning

Q-Learning [109] defines the value of a state action pair as the Q-Value $Q(s, a)$, which represents the value of taking a certain action a while in state s, resulting in a transition to a new state s_{t+1}. $Q(s, a)$ is approximated by successive Bellman updates:

$$Q(s,a) \leftarrow Q_t(s,a) + \alpha[r + \gamma \max_{a'} Q_t(s_{t+1}, a_{t+1})] \tag{6.2}$$

If $Q(s,a)$ is known, Eq. (6.2) can be solved to obtain π^* however its value is usually unknown, and so, estimating it is the task of Q-learning. Deep Q-Learning [39], thereby, uses a deep neural network as this function estimator.

6.4 Methods

6.4.1 Agent

The agent uses a Deep Q Network (DQN) implemented in PyTorch [208] with 2 fully connected hidden layers of 500 and 1000 neurons respectively, and an output layer of 2 neurons, one per allowed action. All layers use ReLU as an activation function. The network weights are optimised using Stochastic Gradient Descent [133], using ADAptive Moment Estimation (ADAM) [136] as the optimizer with a learning rate of $\alpha = 10^{-5}$. The discount factor was set to $\gamma = 0.8$ for all experiments. The discount factor has dimensions T^{-1}, affecting how much the agent cares about rewards far in the future when compared with those in the very close future. An agent with $\gamma = 0$ will be myopic, aiming to produce the best immediate rewards. An agent with γ close to the unit will seek for the greater rewards over time. The determination of an optimal discount factor in RL for TSC is still an open problem. In traffic control, the temporal horizon for the task is infinite, so it makes no sense to tune the discount factor to the expected duration of the training episode, since this episodic manner in which the training proceeds will not be encountered during real operation. While this points towards discount factors close to the unit, it can be argued that this concern for rewards in the far future should not go to such lengths that the agent is caring for actions that are well beyond the point where its own influence would matter (e.g. a controller taking actions for a busy intersection at noon, which are being conditioned on maximisation over the actions that will take place around midnight when the traffic would be extremely low and in no real need of active optimisation). Based on the above, a grid search was performed for different values of γ, obtaining the best results for $\gamma = 0.8$, which was consequently chosen to be the value used by all agents.

6.4.2 Environment

Our environment is a real four-arm junction located in Greater Manchester, UK. This junction is modelled using the microscopic traffic simulator SUMO [62] and

Algorithm 4 Vivacity DQN agent

```
Initialise agent network with random parameters $\theta$,
Initialise target network with random parameters $\theta$',
Initialise memory $M$ with capacity $L$,
Define frequency $F$ for copying weights to target network,
for each episode do
    measure $s_0$,
    while episode not done do
        choose $a_t$ according to $\epsilon$-greedy policy,
        implement action $a_t$,
        advance simulator until next action needed,
        measure $s_{t+1}$, and calculate $r_t$,
        store transition tuple $(s_t, a_t, r_t, s_{t+1})$ in $M$,
        $s \leftarrow s_{t+1}$,
    end
    $b \leftarrow$ sample batch of transitions tuples from $M$,
    for each transition $x_i = (s_i, a_i, r_i, s_{i+1})$ in $b$ do
        $y_i = r_{i+1} + \gamma \max_a Q(s_{i+1}, a', \theta')$
    end
    Stochastic Gradient Descent on $\theta$ over all $(x_i, y_i) \in b$,
    if number of episode is multiple of $F$ then
        $\theta' \leftarrow \theta$
    end
end
```

calibrated using 3.5 months of flow and journey time data collected by vision-based sensors. Specific details of the model tuning, such as saturation rates, turning ratios and vehicle mixture are proprietary information of Vivacity Labs and Immense.AI and are, at the time this document is to be made public, covered by an Non-Disclosure Agreement. A stage is defined as a group of non-conflicting green lights (phases) in a junction. The agent decides which stage to select next and requests this from an emulated traffic signal controller, which moves to that stage subject to its limitations. These limitations are primarily safety-related and examples include enforcing minimum green times, minimum intergreen times, and stage transitions that match reality, including stopping ambers of 3 seconds and starting ambers lasting 2 seconds [209]. The site features four vision-based sensors which can provide flow, queue length and speed data. The data available to the agent is restricted to what can be obtained from these sensors, so approaches such as taking average approaching flows are not feasible within this scope.

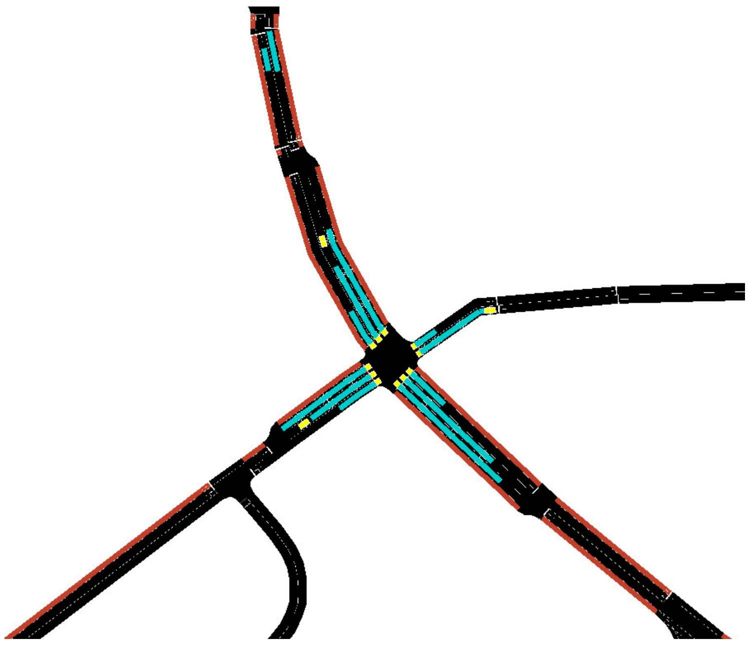


Figure 6.2: Intersection model in SUMO.

This chapter does not consider pedestrians, thus promoting comparability with prior work; pedestrians will be considered in Chapter 7.

6.4.3 State Representation

The input to the agent is a combination of two parts: sensor data and traffic controller state. The sensor data is the occupancy of each lane area detector, while the

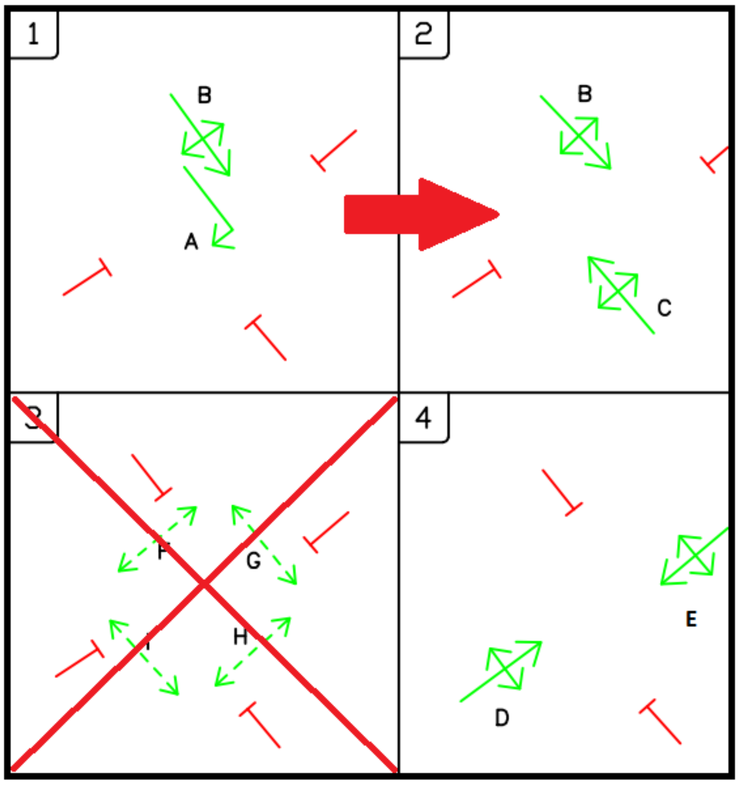


Figure 6.3: Allowed stages in the intersection (Stage 3 only serves pedestrians so isn't used). Stage 1 is an intermediate stage required to reach stage 2. This is known as a *late start*.

controller state is a one-hot (binary valued vector) representation of which stage is active. A 12-second buffer at 0.6s resolution of both parts is provided to the agent.

While there are other state representations provided in the literature which are more information dense, many features of these cannot practically be obtained in the real world by the available sensors. Moreover, recent findings [128] indicate that the gain from more information-dense states is marginal, meaning an agent can manage an isolated intersection with relatively simple state inputs. The state representation has been kept constant across the different experiments presented in this chapter.

6.4.4 Action Set

The junction is configured to have 4 available stages. The agent is able to choose Stage 2 or Stage 4, yielding an action space size of 2. Stage 1 serves a leading offside turn phase from the main road, and was excluded by suggestion of the transport authority, since it is an intermediary stage that the controller will go through in order to reach Stage 2, which serves the main road. Stage 3 only serves pedestrians, which are not considered here, so was also excluded. Stage 4 serves the side roads, which do experience significant demand. In each timestep when a stage has been active longer than the minimum green time, the agent generates state-action values for each potential stage and the highest value is chosen according to an ϵ-greedy policy [108]. If the agent chooses the same stage, that stage is extended by 0.6s, otherwise the controller begins the transition to the other stage. The extension can be chosen indefinitely, as long as the agent identifies it as the best action. The length of the minimum green times for each phase, as enumerated in Fig. 6.3 are listed in Table 6.1.

Table 6.1: Minimum green phase lengths in seconds.

Phase	**Minimum length**
A	4s
B, C, D, E	7s

The complexity in the decision-making stems from the combination of using Stage 1 as an intermediate state and the extensions to the stage duration. Traditional RL for UTC regards each Stage as an action for the agent to take, based on the instantaneous state of the system. However, in the case of the intermediate Stage 1, the agent has to choose when to start the transition without knowledge of the future state when Stage 2 begins. Regarding the extensions, given that their length is smaller than that of the initial phase, their impact on the state will be smaller, generating a distribution of reward and state-action value outcomes that the agent needs to approximate.

6.5 Reward Functions

In this section the individually tested reward functions are introduced.

Let N be the set of lane queue sensors present in the intersection, with individual lanes identified by n. Let V_t be the set of vehicles on incoming lanes in the intersection at time t, with $v \in V_t$ representing individual vehicles. Let s_v be their individual speeds, τ_v their waiting times, and ρ_v the flow of vehicles across the intersection over the length of the action. Let t^p be the time at which the previous action was taken and t^{pp} the time of the action before that.

6.5.1 Queue Length based Rewards

6.5.1.1 Queue Length

The reward will be the negative sum over all n sensors of the queues (q) at time step t. The punishment signal will grow proportionally to the growth of the queues in the junction, being a linear combination in which all items have equal weight. Similar to the reward introduced in [202] but without the need for thresholding the queue values, and used in [205].

$$r_t = -\sum_{n \in N} q_t^n \tag{6.3}$$

One of the first published in the field of Q-Learning, this reward function has low sensor requirements and although is implementable using just induction loops, as shown in Prashanth and Bhatnagar [202], these would provide only coarse grain congestion levels based on thresholding, making more advanced sensors desirable, such as those used in this piece of research.

6.5.1.2 Queue Squared

Introduced in [130], this function squares the result of adding all queues. This generates a reward that is a non-linear combination of the queues, in which all have the same weight. This increasingly penalises actions that lead to longer queues.

$$r_t = -\left(\sum_{n\in N} q_t^n\right)^2 \tag{6.4}$$

6.5.1.3 Delta Queue

The reward will be the difference between the previous and current sum of queues, turning positive for those action that decrease the queue size and negative when it increases. Similar approach to the rewards which are shown in Eqs. (6.7) and (6.10).

$$r_t = \sum_{n\in N} q_{t^p}^n - \sum_{n\in N} q_t^n \tag{6.5}$$

6.5.2 Waiting Time based Rewards

6.5.2.1 Wait Time

The reward will be the negative aggregated time in queue (with τ representing the individual vehicle's time in queue) that the vehicles at the intersection have accumulated since the last action.

$$r_t = -\sum_{v\in V_t} \tau_t^v \tag{6.6}$$

This function is more information-dense than queues, scaling with individual waiting times, but requires more advanced hardware for individual vehicle recognition. Additionally, this function is aligned with the evaluation objective.

6.5.2.2 Delta Wait Time

Similar to Eq. (6.5), used in [126]. The reward will be the difference between the aggregated waiting time of all vehicles in the junction τ between the current time and the previous action.

$$r_t = \sum_{v\in V_t} \tau_{t^p}^v - \sum_{v\in V_t} \tau_t^v \tag{6.7}$$

6.5.2.3 Waiting Time Adjusted by Demand

The reward will be the negative aggregated waiting time as above, but in this case it is divided by an estimate of the current demand ($\hat{d}$) or arrival rate, implicitly accepting that given a wait time as a result of an action, the penalty should scale with the difficulty of the task.

$$r_t = -\frac{1}{\hat{d}} \sum_{v \in V_t} \tau_t^v \tag{6.8}$$

6.5.3 Time Lost based Rewards

6.5.3.1 Time Lost

Used in [190], the reward will be the negative aggregated delay accumulated by all vehicles upstream from the intersection, understanding the delay as deviations from the vehicle's maximum allowed speed (s_{max}). Assuming a simulator time step of length δ:

$$r_t = -\sum_{v \in V_t} \sum_{t^p}^{t} \delta(1 - \frac{s_v}{s_{max}}) \tag{6.9}$$

This reward provides a more accurate representation of the total delay caused, since it also accounts for all deceleration happening around the intersection.

6.5.3.2 Delta Time Lost

Introduced in [201] and used in [188, 127, 124, 125, 128]. Similar to Eq. (6.7). The reward will be the change of global delay in the vehicles around the intersection since the last action was taken.

$$r_t = \sum_{v \in V_t} \sum_{t^{pp}}^{t^p} \delta(1 - \frac{s_v}{s_{max}}) - \sum_{v \in V_t} \sum_{t^p}^{t} \delta(1 - \frac{s_v}{s_{max}}) \tag{6.10}$$

This reward function provides both punishment and reward centered around zero.

6.5.3.3 Delay Adjusted by Demand

The reward will be the same as in the point above, but divided by an estimate of the demand level ($\hat{d}$).

$$r_t = -\frac{1}{\hat{d}} \sum_{v \in V_t} \sum_{t^p}^{t} \delta(1 - \frac{s_v}{s_{max}}) \tag{6.11}$$

6.5.4 Average Speed based Rewards

6.5.4.1 Average Speed

This reward seeks to maximise the average joint speed of all vehicles around an area of influence around the intersection.

$$r_t = \frac{1}{|V_t|} \sum_{v \in V_t} (\frac{s_v}{s_{max}}) \tag{6.12}$$

6.5.4.2 Average Speed Adjusted by Demand

The reward will be, as in the previous section, but multiplied by an estimation of the demand ($\hat{d}$). This function scales the reward with the difficulty of the task.

$$r_t = \frac{\hat{d}}{|V_t|} \sum_{v \in V_t} (\frac{s_v}{s_{max}}) \tag{6.13}$$

6.5.5 Throughput based Rewards

6.5.5.1 Throughput

The reward will be the total number of vehicles that cleared the intersection between the last time that an action was taken and now. As previously introduced in Section 6.5, ρ_v represents the flow of vehicles through the intersection over the length of the action.

$$r_t = \sum_{t_p}^{t} \rho_v \tag{6.14}$$

6.5.6 Other reward functions

There are several other reward functions which were not considered. For example minimising the frequency of signal change [123, 210], and accident avoidance [123]: both of these concerns are already addressed by traffic signal controllers. Also, pressure, "defined as the difference of vehicle density between the incoming lane and the outgoing lane"[199] has been used as a reward function in the control of a large network [199, 206, 207, 211, 210, 212] and achieved good results; however, this requires data from upstream and downstream of the target junction, so is beyond the current scope. Lastly, while the mean-rate of delay [213] could potentially be used as a reward metric which to minimise, its calculation is dependent on estimation of the current flow levels at the instant the decision is taken. The demand estimation proposed for use in previous subsections is only used here as a constant value through each training episode and not offered by sensors nor dynamically updated. Should this measure be being used to calculate the mean-rate of delay, it would produce static cycles in the best case scenario, and completely random behaviour in the worst, as all actions could be perceived as having the same value. This behaviour places it in a different category from typical RL controllers which are meant to update their knowledge about the system each time a decision is needed, and more akin to traditional adaptive controllers that change their cycles over longer timescales. Furthermore, the need to know a priori the length of the proposed cycle to estimate said rate of delay, would make it incompatible with the current implementation of industry-grade controllers being tested, that do not have this knowledge, since for them a stage extension is considered a new decision, taken using updated data, that extends the current cycle.

6.6 Experimental Setup

6.6.1 Training Process

In each training run, the agent is subject to a training curriculum including a variety of scenarios, including sub-saturated, near-saturated, and over-saturated situations. The agent is first shown sub-saturated episodes, before the difficulty level is increased. Each episode runs for 3000 steps of length 0.6 seconds, for a total simulated time of 30 minutes (1800 seconds).

Ten training runs were conducted for each of the reward functions described in the following section, to capture variance in training run outcome.

6.6.2 Evaluation and Scoring

After each training run is complete, the agent's performance is evaluated and compared to that of reference agents in terms of average waiting time as defined by Vivacity Labs. While the rate of loss of time to the community at a junction can be calculated as the excess amount of traffic in the vicinity (also known as the rate of delay) [213], the expression used for the calculation assumes that a constant proportion of the cycle will be effectively green, which is not the case in the case of RL agents since their cycle times can have great variation. Instead, delay measures provided by the simulator were used as evaluation metric, following approaches of recent RL literature in the topic [122, 125, 126, 128]. The approach here described will be carried over into the next chapter.

Table 6.2: Average waiting time in seconds across demand scenarios

Reward Function	Low	Medium	High
Queues	6.59 ± 0.46	8.97 ± 1.27	16.21 ± 5.07
Queues Squared	6.35 ± 0.53	8.56 ± 1.26	16.22 ± 5.08
Delta Queues	6.47 ± 0.41	8.96 ± 1.59	18.87 ± 5.07
Stopped Time	6.64 ± 0.52	11.88 ± 3.39	26.27 ± 4.61
Stopped Time AD	6.70 ± 0.46	9.60 ± 1.66	17.68 ± 4.95
Delta Stopped Time	7.15 ± 0.81	16.59 ± 4.95	29.21 ± 3.67
Time Lost	6.79 ± 0.42	9.23 ± 1.15	15.84 ± 4.36
Time Lost AD	6.59 ± 0.46	8.97 ± 1.27	16.21 ± 5.07
Delta Time Lost	8.27 ± 1.48	13.48 ± 4.04	22.54 ± 5.54
Average Speed	6.24 ± 0.39	8.61 ± 1.07	14.95 ± 3.40
Average Speed AD	6.13 ± 0.44	8.22 ± 1.24	14.33 ± 4.97
Throughput	28.02 ± 9.36	51.16 ± 7.23	55.72 ± 7.02
Vehicle Actuated	8.70 ± 0.62	14.76 ± 1.69	27.9 ± 6.05
Maximum Occupancy	6.32 ± 0.51	9.51 ± 1.87	21.33 ± 5.77

Two reference agents have been implemented to give context to the RL performance: Maximum Occupancy (longest queue first) and a Vehicle Actuated Controller. The Vehicle Actuated algorithm was an implementation of System D [214], a common algorithm in the UK. For each training run, the RL and reference agents are each tested on 300 scenarios: this is 100 scenarios at each of 3 demand levels. In each repetition the average waiting (stopped) time for all vehicles was computed.

The first scenario involves a demand of 1714 vehicles/hour (1 vehicle/2.1 seconds), and will be referred as Low Demand scenario. The second scenario uses a demand of 2117 vehicles/hour (1 vehicle/1.7 seconds), and will be referred as Medium Demand scenario, although this level of demand is around the observed peak and the saturation level for the junction. The third scenario test a demand level slightly above what the junction can currently serve. It uses a demand of 2400 vehicles/hour (1 vehicle/1.5 seconds) and will be referred as High Demand scenario.

For each reward function, the best result of the 10 training runs was selected, based on the evaluation method stated in the previous two paragraphs.

6.7 Results

In this section, the distribution of average waiting times per vehicle across the different scenarios is presented for each reward function.

Figures 6.4 - 6.6 show for each agent, the distribution of average waiting times per vehicle across 100 repetitions of each scenario as described in the section above. In the plot, the box encompasses the two intermediate quartiles, with a dashed line indicating the mean and a solid line indicating the median. The whiskers show the range of the data, with the remaining data outside them being represented as flier points. The throughput reward function is not shown to improve graph readability, as it performed poorly by all metrics. Complete corresponding results are reported in Table I.

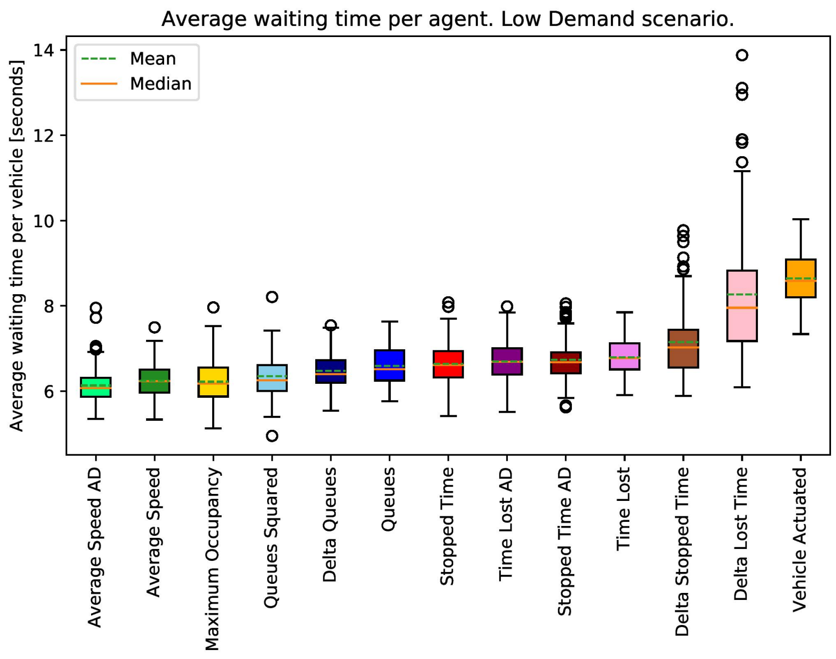


Figure 6.4: Distribution and medians of average waiting time in seconds across agents in Low Demand scenario. Sub-saturation demand of 1714 vehicles/hour (1 vehicle/2.1 seconds).

Reward functions optimising the average speed around the intersection were found to be best performing across all three scenarios and consistently outperformed the reference agents. Adjusting the average speed by an estimate of the demand level shows to improve the performance of this reward function in all cases. From those rewards using queues, Queues Squared performed best, except for the High Demand scenario in which the sum of Queues obtains marginally lower waiting times. The results from agents using reward functions based on either Wait Time or Time Lost are inconsistent across different demand levels. While sum of Time Lost is the third best-performing agent at High Demand, it is sixth in Medium Demand and Tenth in Low Demand. Sum of Wait Time follows an inverse, yet less extreme trajectory as the demand is lowered, from worse to better performance. Throughput was found

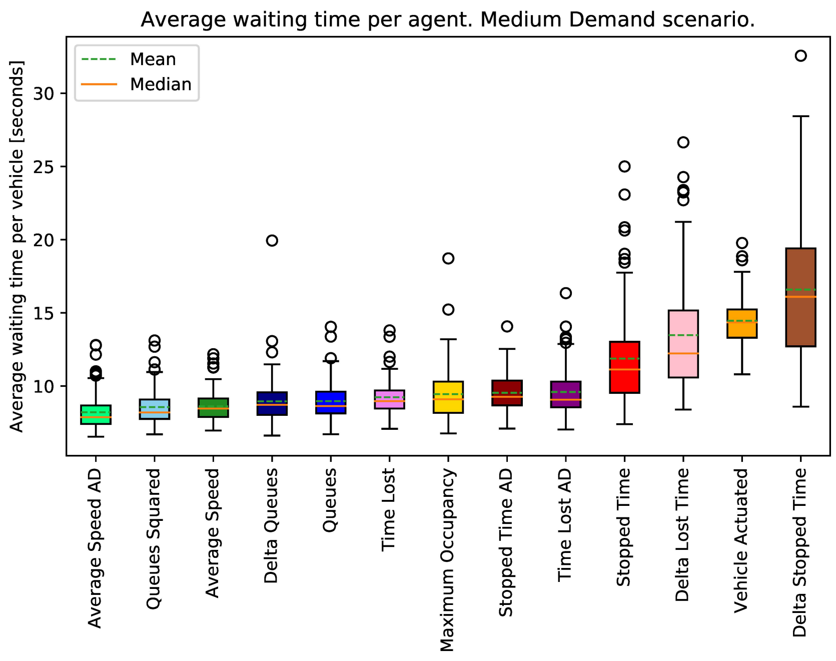


Figure 6.5: Distribution and medians of average waiting time in seconds across agents in Medium Demand scenario. Near-saturation demand of 2117 vehicles/hour (1 vehicle/1.7 seconds).

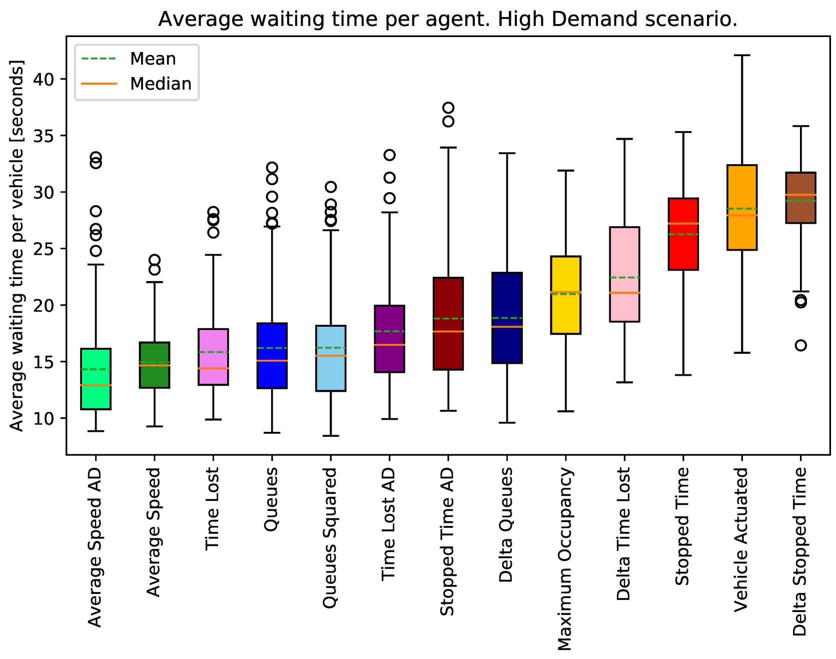


Figure 6.6: Distribution and medians of average waiting time in seconds across agents in High Demand scenario. Over-saturation demand of 2400 vehicles/hour (1 vehicle/1.5 seconds).

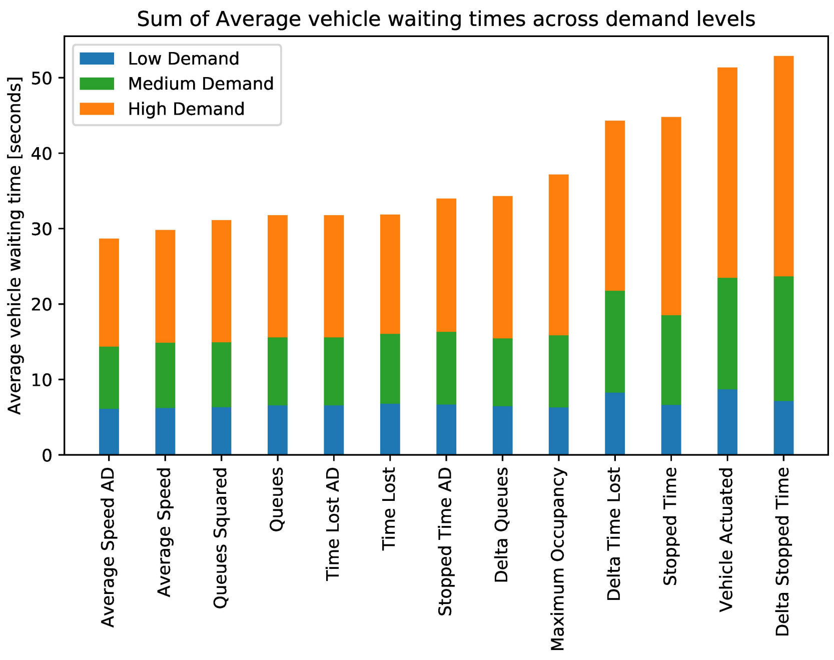


Figure 6.7: Stacked bar chart of Average Waiting Time across scenarios.

to be consistently the worst-performing of the reward functions tested. The rewards based on difference in delay (Delta Wait Time and Delta Time Lost), which were used in seven previous items of research as a most desirable reward function, were amongst the worst performers.

Lastly, Figure 6.7 presents the aggregate results for the different demand levels as a general performance guide from the perspective of individual drivers and not as a community, since the total community waiting time is proportional to the number of vehicles as well as their mean delay.

6.8 Discussion and Conclusion

This chapter has explored the performance of several reward functions for deep Q-learning agents in the context of Urban Traffic Control. The chapter reaffirms earlier findings that RL outperforms deterministic baseline algorithms. Overall, the proposed reward functions based on maximisation of the average speed of the vehicles in the network resulted in the lowest waiting times across demand levels, including both baseline algorithms. The performance of queue length and stopped time rewards was comparable, but when reliability is taken into account, average speed is clearly preferred since it produces a smaller variation in waiting times.

In sub-saturated conditions, a simple approach of always serving the largest queue probably suffices, although rewarding based on queue length had similar performance, and rewarding for average speed yielded slight improvements. In busier conditions, which are more important to transport authorities, RL provides significant improvements in waiting time, with speed-based rewards again performing best, followed by queue-based rewards in near-saturation conditions, and followed by penalising for lost time in saturated conditions.

Rewarding based on delay minimisation or stopped time may perform worse than speed-based rewards as they encode information about previous timesteps, whereas queue lengths and vehicle speeds are snapshots at a single timestep, which better suits the underpinning Markov Decision Process framing of the problem. Speed maximisation may have yielded better results than queue minimisation under high demand as it penalises the agent for vehicles which are moving slowly forward in congested conditions, which would not have been penalised by the queue length reward. This would also not have been captured by the stopped time reward, which may explain its worse performance overall. One key finding is that whilst rewards based on difference in delay were found to perform well in earlier works, this was found to perform poorly in the present work. The Maximum Occupancy reference agent performs well at low demand, but mediocre at medium demand and poorly at high demand. It does not extend stage durations in busy traffic when the amount of vehicles of vehicles using the active stage is slightly lower than that of stationary queuing traffic. This results in shorter than optimal stage lengths and so too much time is wasted in interstage transitions. However, it did consistently perform better than the Vehicle Actuated reference agent. The Vehicle Actuated algorithm extends stages until the related loops have been unoccupied for 1.5s, hence extending stages in congested traffic. However, its performance suggests that this standard extension length, coupled with the maximum stage durations found empirically to perform relatively well, is overly eager to extend stages. From a visual inspection via the simulator of the best RL agent's behaviour, it appears to learn an 'adaptive stage extension' behaviour: learning when to prioritise a large queue waiting to be served and when to prioritise avoiding the cost of transitioning stages. A key benefit to the DQN approach is that it can learn how eagerly it should extend stages under varying conditions at a specific site, rather than requiring manual calibration. This automated calibration can be repeated regularly so that the site continues to perform well long-term without manual intervention.

Regarding robustness against previously unseen situations, at the core of the agents we find neural networks which are specially well suited for extrapolation, this being a characteristic that has driven their growth in usage over the previous years. Based on this, slightly differing state variables from those previously observed during training should not pose any kind of risk for the robustness of operation of such an agent. Moving onto major disruptions and incidents, the main great change that an agent could find while in real operation, not previously seen in training relates to one or more of the lanes feeding into the intersection being closed. The agents here presented obtain the environment states based on lane occupancy, used as a proxy for the level of demand. In such a situation, the measured occupancy for the affected lanes would be zero, indicating to the agent that there is no demand for said lanes. This in turn, will lead the agent not to serve said lanes, meaning that the agent would be completely robust against lane closures, performing normally for the remaining lanes. If the focus is placed instead into driver-caused accidents in the middle of the intersection, blocking it; unless there are were some sensors covering said area, and proper routines in place for this eventuality, there would be no immediate adaptation as the controllers, similarly as with current day systems. Should said sensors and emergency routines be in place, there is no reason why an initial alert to the competent authorities, paired with a cautionary closure of the entire junction cannot be executed by the agent itself, instead of depending on alerts given by the affected drivers or other nearby people. When this is paired by the fact that during RL agent deployments for traffic control, the previous control system is not overwritten and it is possible to switch back to it with a few keystrokes, it can be stated that a priori there are no reasons why RL agents for TSC should be less robust nor secure that their traditional counterparts, and that there are several reasons pointing to these agents potentially being able to offer higher levels of robustness and security for road users without the need of further expenditure than that required for their standard deployment.

6.8.1 Limitations and Future Work

One limitation of the study is that the results presented only relate to the training of a deep Q-learning agent as specified in Section 6.4, and are not necessarily representative of the performance in other type of architectures such as those representing the state as an image and processing it with Convolutional Neural Networks, or Policy Gradient methods. Another limitation is that most intersections found in an urban setting do not exclusively serve vehicles, this will be tackled in the following chapter.

RL for TSC, as it is treated by most literature, has some deep differences with the set of problems that the seminal version of these algorithms were created to solve, which hamper their performance and progression. While the intention of this chapter is not to assess the theoretical suitability of the reward functions introduced, but to gather previously proposed reward functions and others recently proposed by industry and empirically compare their performance for the first time, there are open points regarding reward shaping in RL for TSC that are worth covering.

One such issue is that DQN agents assume an episodic nature of finite length of the task at hand, while traffic control in practice behaves as an infinite horizon control task. For this reason, most RL TSC literature assumes that the best parametrisations for the discount factor is to take values close to the unit. As this chapter shows, often better results can be obtained by setting it to lower values, with the implicit assumption that the temporal scales for which there exists a correspondence between actions and states over which to optimise action choices are indeed finite (i.e when an intersection is empty, the choice of action does not matter, breaking the continuity of the need for optimisation over time). However, the linkage between some measure of temporal structure in the state variables (which ultimately condition the action choice) and the specific value of the discount factor has not been solved yet. Hence, the specific choice of discount factor is, as stated earlier, still an open problem in RL for TSC. Even so, often reasonable estimates of the best performing discount factor for a given task can be obtained, although at a high computational cost, by performing grid searches.

A second problem relates to the lack of temporal correspondence between the passage of time in the environment and how the agent perceives this temporal evolution. In most RL applications, it is found beneficial for the general performance of the agents that these are able to act with as high of a frequency as it is possible to obtain updated data for them (e.g. Go, Atari games, self-driving vehicles, electrical grid control, etc.), so they can fine-tune their actions with improved information. In these cases, one time step for the environment equals one state estimation-action-reward cycle for the agent, with the implicit assumption that every action lasts a single time step. This is not the case in RL for TSC, where normally actions have, at least, a minimum length, and the possibility of being extended for a longer time (with the minimum extension time not needing to be equal to the minimum stage times). This means that once an action is chosen, the agent is committed to maintain it as its chosen action for the duration, regardless of the evolution of the environment during this process. The fact that actions are fixed after being chosen, creates uncertainty as to how to approach the experience management of the agent over time, being able to either only record the transitions in terms of the initial state that led to the action being taken and the end state at the end of the action, or to record every intermediate time step, keeping the action fixed for the duration and calculating a reward value for every temporal increment in the simulator while disregarding the choices of action of the agent. There are arguments supporting the first choice based on the idea that it would produce experience transitions that better represent the effects of a given action at the cost of some granularity. Opposing arguments adduce that recording every transition multiplies the experience available by a factor located between the minimum stage extension length and minimum phase length, at the cost of a degree of global view from the agent. Ultimately, the question as to which approach produces superior results remains open and is left for future work. The first design philosophy of the two here presented has been followed through the RL-focused chapters in this thesis, matching the most extended choice in the field's literature. An analogous discussion can be made for whether extensions should be treated together with the initial action as a whole, or whether they should be treated as a separate action. This discussion is also currently open in RL for TSC.

Further, the changes in the state of the environment will not occur uniformly during the lifespan of an action: a traffic light that just turned green will have minimal effect on the queue length of its corresponding lane during the first second, since only those vehicles at its front will be accelerating from an initially stopped position. Equally, if an action is such that the related queues are cleared before the end of the stage, the impact of said action on the affected lanes will be non-existing since the moment of clearing, while the likely arrival of further vehicles in other lanes will make this part of the action look detrimental to performance from a reward point of view.

These facts, when paired with the design methodology chosen for this work, mean that from the point of view of the simulator the agent will provide actions with variable frequency, while from the point of the agent it will result in the observed distributions of rewards having, in general, a higher variance that what would be observed if the time-lines for agent and environment were shared. While it is undeniable that a wider distribution of stored rewards for a given action will have make training more difficult, there is a large amount of evidence available both in the literature, as well as in products being deployed in the real-world that successful learning and successful control are in no way impeded by this.

All the rewards presented in this chapter, except for those based on average speeds, are affected by the lack of correspondence between agent's and environment's accounting of time to some degree. This is one possible reason for the dominance shown by average speed based rewards. This effect will increase the observed variance of the distributions of rewards within a class, compared to what would be observed under fixed action lengths. The reward functions that will be more severely affected are those based on delay, stopped time and throughput. In the first two cases, this is be due to the fact that their variation in a single time step will be roughly proportional to the number of vehicles below maximum speed and stopped respectively, and while any extensions will reduce the amount of stopped vehicles, the remaining vehicles will keep accumulating delay during said extension. In the last case, the variation in the observed reward will be approximately proportional to the total extension length, since these will occur at a moment when cars are discharging their link at close to the saturation rate. Given that the agent is not aware of this lack of correspondence in the passage of time between itself and its environment, and that extensions are treated as longer single actions, this equates telling a throughput based agent that the saturation rate is effectively unknown, variable and unpredictable within a range. This is one potential reason for the bad performance in these experiments of throughput based agents. Queue based rewards will be affected to a lesser degree, since the number of vehicles discharged from a queue will raise with the length of the extension, but the number of vehicles arriving at the back of the remaining queues will partially offset this.

In terms of functional forms, those estimating the variation between actions, and specifically those based on time lost and delay could see the errors in their estimation affect the sign of the reward, since they are the only rewards can can take both positive and negative values. This will, in turn generate more suboptimal choices of actions leading to greater variance in the agent's performance.

One potential solution to this issue is to modify the reward calculation procedure to include a multiplicative term inversely proportional to the length of the corresponding stage plus any extensions, effectively having the agent try to approximate the rate of reward in reward units per second. This approach will be tested in the next chapter, but ultimately this is a research question that remains open.

One last concern can be raised about the use of a fixed pre-calculated factor (estimation of demand) being applied to the calculation of any given reward function. The first consideration here is that the choice of action itself is not critically affected by this factor most of the time, as the factor in the experiments here presented varied between 1.5 and 2.1 (matching the number of vehicles inserted into the model per second), creating a maximum deviation in the expected value of an action that is usually well below the difference in reward between a "good" choice of action and a "bad" one (i.e. in the best case scenario, assuming the estimated state-action value is the underlying truth, after a changing the demand from one extreme to the opposite, will generate an estimation error of $2.1 - 1.5 = 0.6$ times the true value). By introducing such a factor, if we increase the demand by said 60%, the agent using queues will receive the same reward value for a queue that is 60% longer than it was under low demand. While an statement such as "60% lower demand means 60% lower queues/delays/stopped times" does not acknowledge the non-linearities in traffic and would not hold when experimentally checked, adding such an approximation to the agents has, in general, helped reduce the variance of the reward distribution to estimate, and improved agent performance, especially in high demand situations. This was initially a highly experimental approach that has not been previously shown in the literature, and it was kept since the empirical results showed that in a great majority of cases, it helped reduce the variance in the performance of the agents when compared to those using the same measures but not including this factor. As such experimental addition, there are characteristics of it that could be the subject of further study. Additionally, there is room to expand and improve it, searching for ways to calculate reward levels relative to the level of demand that would help stabilise the learning and behaviour of RL TSC agents while increasing their robustness. This too is outside the scope of this piece of research and left for future work.

Overall, the ability of RL agents to effectively deal with ambiguities such as those introduced in this section, plus those regarding their required ability to extrapolate when in complex state-action spaces is the reason that they are displaying such a success, leading to their popularisation over the past few years. While these issues remain open, valid and relevant, just as well as other current concerns about neural-networks (e.g. explainability), neither of them ultimately prevent RL agents from having the ability to exert control over highly complex systems.

CHAPTER 7

Assessment of Reward Functions in Reinforcement Learning Agents for Real-World Multi-Modal Isolated Intersections

7.1 Introduction

Reinforcement Learning (RL) approaches have been showing promising results in the field of Traffic Signal Control. However, most of the existing works do not try to jointly optimise vehicular and pedestrian travel times, even though pedestrians are allowed and present in the great majority of urban intersections.

This chapter compares the performance of RL agents using 30 different reward functions split into 5 different classes based on the inputs they use, when controlling a simulation of a real-world junction in Greater Manchester (UK) that has been calibrated using 3.5 months of data gathered from s Labs vision-based sensors.

The chapter is structured as follows: Section 7.2 reviews previous literature in the field. Section 7.3 states the mathematical framework used and provides some theoretical background. Section 7.4 reviews the environment, the agents and their implementation. Section 7.5 introduces the reward functions tested in this chapter and provides their analytical expressions. Section 7.6 contains details about the training and evaluation of the agents. Lastly, Section 7.7 provides the experimental results and discusses them.

7.2 Related Work

RL for UTC has been previously explored and discussed in a variety of research, aiming to eventually substitute existing adaptive control methods such as SCOOT[51], MOVA[45] and SCATS[54]. The field has evolved from early inquiries about its theoretical potential use [117, 200, 202, 201, 215], to progressively more applied and realistic scenarios that look towards real-world use and deployment. Recent works use different inputs for the reward function of the controlling agents (delay, queues, waiting time, throughput, ...), however, it is not clear what benefits are provided from choosing which. The different inputs used as reward are thoroughly indexed in Yau, K. L. A., Qadir, J., Khoo, H. L., Ling, M. H., & Komisarczuk [206], Haydari and Yilmaz [216], and Wei et al. [207], although no direct performance comparisons are made. Different methods are taken regarding inputs, such as pixel-based vectors passed to a CNN [217, 124, 125], per-lane state signals using fully connected neural networks [122, 205, 129], or hybrid approaches [127, 130, 190]. Recent research suggests that more complex state representations only provide marginal gains, if any[128], so in this paper the second approach is taken.

A common thread in most previous works is the need for approximations about the network being studied and the lack of pedestrian modelling and joint optimisation for vehicles and pedestrians travel times. As indicated in [216], pedestrian implementation has a high impact on learning performance, being often discarded as unimportant or left for future work save for two exceptions [88, 189], the first of which uses a genetic algorithm instead of RL, and the second explores a single reward function. In this paper we attempt to cover this gap in the literature, providing a robust performance assessment of RL agents serving both vehicles and pedestrians, using a variety of rewards, both novel and from the literature, attempting to uncover what state variables should be used in the reward to obtain the best performance. These are applied to a RL agent in a calibrated model of a real-world junction, using real geometry, calibrated demand, realistic sensor inputs and emulated traffic light controllers, to which some of these agents have been deployed to control real traffic in it since these experiments took place. This paper delivers the future work deferred from [6] in terms of shifting the focus towards pedestrians and multi-objective optimisation, while keeping the problem grounded in the real world.

7.3 Problem Definition

7.3.1 Markov Decision Processes and Reinforcement Learning

The problem is framed as a Markov Decision Process (MDP) as introduced in Section 2.2.5.1, satisfying the Markov property: given a current state s_t, the next state s_{t+1} is independent of the succession of previous states $\{s_{t-1}, s_{t-2}, ..., s_0\}$. An MDP is defined by the 5-element tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma >$.

The objective of an MDP optimisation is to find an optimal policy π^*, mapping states to actions, that maximises the sum of the expected discounted reward,

$$R_t = \mathbb{E}\Bigg[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\Bigg]. \tag{7.1}$$

As stated before, in the case of RL for UTC, $\mathcal{T}$ is unknown, making it necessary to approach it from a model-free RL perspective.

7.3.2 Q Learning and Value-Based RL

As introduced in Section 2.2.5.2, Q-Learning [109] is an off-policy model-free value-based RL algorithm. For any finite MDP, it can find an optimal policy which maximises expected total discounted reward, starting from any state [132]. Q-Learning aims to learn an optimal action-value function $Q^*(s, a)$, defined as the total return after being in state s, taking action a and then following policy π^*.

$$Q^*(s, a) = \max_{\pi} \mathbb{E}_{\pi*}\big[R_t | s = s_t, a = a_t\big] \tag{7.2}$$

Traditional table-based Q-Learning approximates $Q^*(s, a)$ recursively through successive Bellman updates,

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha\big(y_t - Q(s_{t+1}, a)\big) \tag{7.3}$$

with α the learning rate and y_t the Temporal Difference (TD) target for the Q-function:

$$y_t = r_t + \gamma \max_{a_{t+1}} Q_\pi(s_{t+1}, a_{t+1}) \tag{7.4}$$

This table representation is not useful for high dimensional cases, since the size of our table would increase exponentially, nor for continuous cases, since every distinct $s \in \mathcal{S}$ would require an entry.

7.3.3 Deep Q Network

One way of addressing the issues of Q-Learning in high dimensional spaces is to use neural networks as function approximators. This approach is called Deep Q-Network (DQN) [39], and it was previously introduced in Section 2.2.5.4. The Q-function approximation is denoted then in terms of the parameters θ of the DQN as $Q_\theta(s,a)$. DQN stabilises the learning process by introducing a Target Network that works alongside the main network. The main network with parameters θ, approximates the Q-function, and the target network with parameters θ' provides the TD targets for the DQN updates. The target network is updated every number of episodes by copying the weights $\theta^- \leftarrow \theta$. With $Q_{\theta'}(s_{t+1}, a_{t+1})$ representing the target network, it results in a TD target to approximate:

$$y_t = r_t + \gamma \max_{a_{t+1}} Q_\theta(s_{t+1}, a_{t+1}). \tag{7.5}$$

7.4 Methods

7.4.1 Reinforcement Learning Agent

The agent used to obtain these results is a standard implementation of a DQN in PyTorch [208], optimising its weights via Stochastic Gradient Descent [133] with ADAM [136] used as optimizer. The learning rate is $\alpha = 10^{-5}$ and the discount factor is $\gamma = 0.8$ for all simulations. The Neural Network in the agent uses 2 hidden, fully connected layers of sizes 500 and 1000 respectively, using ReLU as an activation function.

7.4.2 Reinforcement Learning Environment

The environment is modelled in the microscopic traffic simulator SUMO [62], representing the same real-world intersection in Greater Manchester that was used in the previous Chapter. The junction consists of four arms, with 6 incoming lanes (two each in the north-south orientation, and one each in the east-west orientation) and 4 pedestrian crossings. The real-world site also contains 4 vision-based sensors, able to supply occupancy, queue length, waiting time, speed and flow data. The demand and turning ratios at the junction have been calibrated using 3.5 months of journey time and flow data collected by these sensors. Specific details of the model tuning, such as saturation rates, turning ratios and vehicle mixture are proprietary information of Vivacity Labs and Immense.AI and are, at the time this document is to be made public, covered by an Non-Disclosure Agreement.

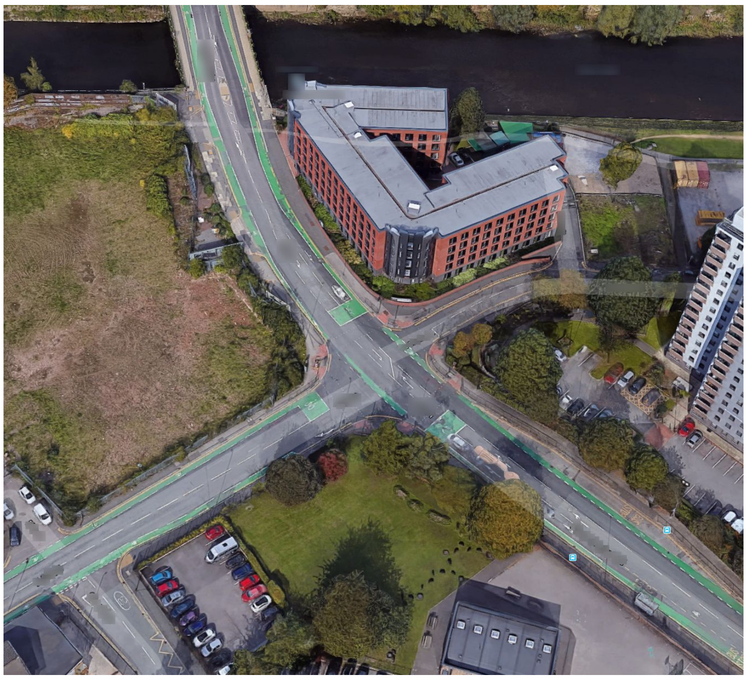


Figure 7.1: Aerial view of Study Junction from Google Earth.

The environment includes an emulated traffic signal controller, responsible for changing between the different stages in the intersection and enforcing the operational limitations, which are focused on safety. This includes enforcing green times, intergreen times, as well as determining allowed stages. Stopping ambers are 3 seconds and starting ambers last 2 seconds. A stage is defined as a group of non-conflicting green lights (phases) in a junction which change at the same time. The agent decides which stage to select next and requests this from an emulated traffic signal controller, which moves to that stage subject to its limitations, which are primarily safety-related. The data available to the agent is restricted to what can be obtained from these sensors, as explained in the previous chapter, so approaches such as taking average approaching flows are not feasible within this scope.

7.4.3 State Representation

The agent receives an observation of the simulator state as input, using the same state information across all experiments here presented. Each observation is a combination of the state of the traffic controller (which stage is active) and data from the sensors. The data from the sensors is comprised of the occupancy in each lane area detector and a binary signal representing whether the pedestrian crossing button has been pushed. Given that this is the only representation of the state of the pedestrians, it is not possible to treat them as one of the streams of traffic. The agent receives a concatenation of the last 20 measurements at a time, covering the previous 12 seconds at a resolution of 0.6 seconds.

7.4.4 Actions of the Agent

The junction is configured to have 4 available stages. The agent is able to choose Stage 2, Stage 3 or Stage 4, yielding an action space size of 3. The action space has an extra action available when compared with that presented in Chapter 6, since now pedestrians are fully simulated. Stage 1 services a protected offside turn coming from the north. It is used by the traffic light controller, as a transitional step for reaching Stage 2, as defined by the transport authority. Stage 2 deals with the traffic in the north-south orientation. Stage 3 is the pedestrian stage, setting all pedestrian crossings to green, and all other phases to red. Stage 4 services the roads in the east-west orientation, which have considerable demand.

Once the controller has had a stage active for the minimum green time duration, the agent is requested to compute the value of all potential state-action pairs (i.e. the value of other stages given the current state) once per time-step. From these, the action with the highest expected value is selected following an ϵ-greedy policy [108]. Should the agent choose the same action, the current stage will be extended for a further time-step (0.6 seconds). There is no built-in limit to the maximum number of said extensions, leaving it for the agent to learn the optimal green time for any given situation. If a different stage is chosen, then the controller will proceed to the intergreen transition between them. The length of the minimum green times for each phase, as enumerated in Fig. 7.2 are listed in Table 7.1.

Table 7.1: Minimum green phase lengths in seconds.

Phase	**Minimum length**
A	4s
G	5s
B, C, D, E, F, H, I	7s

There are 2 situations that further add to the complexity of this control process:

1. Variable number of extensions, and hence length of the stages, creates a distribution of values over the state-action pairs in most rewards, which the agent

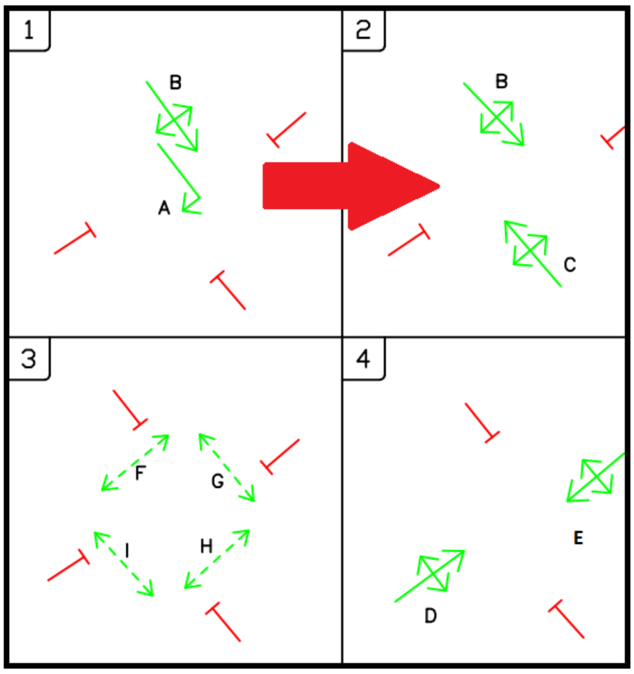


Figure 7.2: Allowed stages and the phases that compose them. Stage 1 is an intermediate Stage, which is necessary to go through to reach Stage 2.

must approximate. The variance of this distribution will be higher than the variance that would be obtained using constant stage length.

2. The requirement that Stage 1 must be used as an intermediate step to reach Stage 2 implies less certainty in the control process than in other stages, since there is an unaccounted dilated temporal horizon between the state that triggered the action, and the effects of said action over the state variables.

7.4.5 Modal Prioritisation and Adjusting by Demand

The agent serves vehicles and pedestrians arriving at the intersection, seeking to jointly optimise the intersection for both modes of transport.

All the reward functions presented in this chapter follow the same structure. The reward, as seen by the agent controlling the intersection, will be a linear combination of an independently calculated reward for the vehicles and another for the agents, as it can be seen in Eq. (7.6).

$$r_t = a * R_t^v + b * R_t^p; \qquad a + b = 1 \tag{7.6}$$

In this way, a and b are the Modal Prioritisation coefficients for our rewards, with

R^v, R^p being respectively the vehicular and pedestrian rewards.

This is a simple yet effective way of accounting for different opposed objectives during the control process. Due to the configuration of the intersection, there is no phase that serves simultaneously both vehicles and pedestrians. For this reason, a reward of the form presented in Eq. (7.6) can provide an effective learning environment for agents, since different components of it will capture the state metrics of the different modes of transportation. There is no reason why this approach cannot be reasonably extended to encompass a few classes. This can lead to interesting takes from the traffic light operators in which bikes or public transportation vehicles can have weights that benefit their travel times, giving them priority over individual cars, and creating extra incentives to use these modes of transportation.

Of the rewards presented in the following section, those that were more sensitive towards the relative ratio of the demand between pedestrian and vehicles require manual tuning of the modal prioritisation parameters. While undesirable from a modeller and operator point of view since it partially counters the benefits that RL provides in terms of self-adjustment, they are provided so potential users and researchers can evaluate the trade-offs between potential increased performance and increased configuration effort. The mentioned series will be identified by the weight applied to the pedestrians. As such, series identified as P80 and P95 represent those in which the weights were $a = 0.2$, $b = 0.8$, and $a = 0.05$, $b = 0.95$ respectively. Those series without an identifier do not require tuning the modal prioritisation weights ($a = b = 0.5$).

The need for specific prioritisation coefficients is closely related to the choice of inputs for the reward function and the lack of balance between pedestrian and vehicular demand, with the vehicular demand being greater. Rewards such as those based on queues do not require of a specific choice of prioritisation weights in order to converge to effective policies. In this case, it occurs because the agent will learn to serve each phase before certain levels of unacceptable queues are reached for a certain lane or pedestrian area, and the queues measures grow by a unit each time a vehicle or pedestrian joins one. A similar reasoning can be made for, for example, average speed based rewards. However, those rewards based on input quantities that inherently grow over time are more susceptible of being unbalanced in this scenario. For example, if we look at those based on waiting time under the assumption of greater volume of vehicle demand than pedestrian demand, we find that it is likely that the amount of delay jointly accumulated by all vehicles in the intersection while they are stopped during the pedestrian phase, is much greater than what can be obtained during any other phase in which pedestrians will be waiting and at least some of the vehicles will be moving. Following this reasoning, it is easy to see how certain rewards can find convergence for all classes more easily when the relative importance for the agent of one mode of transportation is boosted through prioritisation weights.

One last addition that can be made to the rewards is to add a term scaling the difficulty with the demand level, implicitly accepting that higher demand typically worsens the performance of a network, independent of the actions of the controlling agent. These series are identified with the suffix AD (Adjusted by Demand).

7.5 Reward Functions

All reward functions tested are presented in this section with their analytical expressions.

Let N be the set of lane queue sensors present in the intersection. Let M be the set of pedestrian occupancy sensors in the junction. Let V_t and P_t be respectively the set of vehicles in incoming lanes, and the set of pedestrians waiting to cross in the intersection at time t, with individual vehicles and pedestrians identified by v and p. Let s_t^v be the individual speed of vehicle v at time t, τ_t^v and τ_t^p the waiting times of vehicles and pedestrians at time t, respectively. Let ρ_v and ρ_p be the vehicular and pedestrian flows across the junction over the length of the action. Let t^p be the time at which the previous action was taken and t^{pp} the time of the action before that. Lastly, let t_e^v and t_e^p be the entry times of vehicles and pedestrians to the area covered by sensors.

7.5.1 Queue Length based Rewards

7.5.1.1 Queue Length

Similar to [202], used in [205], the reward is the negative sum at t of queues (q) over all (n, m) sensors.

$$r_t = -\sum_{n \in N} q_t^v - \sum_{m \in M} q_t^p \tag{7.7}$$

7.5.1.2 Queue Squared

Seen in [130], this function squares the result of adding all queues.

$$r_t = -\left(\sum_{n \in N} q_t^v\right)^2 - \left(\sum_{m \in M} q_t^p\right)^2 \tag{7.8}$$

7.5.1.3 Queues Phase Length Normalisation

As Queue length, but dividing the sum by the phase length (Phase Length Normalisation), approximating the reward that the action generates by unit of time it is active.

$$r_t = -\frac{1}{t - t^p} \sum_{n \in N} q_t^v - \sum_{m \in M} q_t^p \tag{7.9}$$

7.5.1.4 Delta Queue

The reward is the variation of the sum of queues between actions. Similar to Eqs. (7.13) and (7.16).

$$r_t = \left(\sum_{n \in N} q_{t^p}^v - \sum_{n \in N} q_t^v \right) + \left(\sum_{m \in M} q_{t^p}^p - \sum_{m \in M} q_t^p \right) \tag{7.10}$$

7.5.1.5 Delta Queue PLN

As Delta Queue, but dividing the sum by the phase length (Phase Length Normalisation).

$$r_t = -\frac{1}{t - t^p} \left(\big(\sum_{n \in N} q_{t^p}^v - \sum_{n \in N} q_t^v \big) - \big(\sum_{m \in M} q_{t^p}^p - \sum_{m \in M} q_t^p \big) \right) \tag{7.11}$$

7.5.2 Waiting Time based Rewards

These rewards require Modal Prioritisation weights.

7.5.2.1 Wait Time

The reward is the negative sum of time in queue accumulated since the last action by all vehicles. This function is aligned with the evaluation objective.

$$r_t = -\left(a \sum_{v \in V_t} \tau_t^v + b \sum_{p \in P_t} \tau_t^p \right) \tag{7.12}$$

7.5.2.2 Delta Wait Time

Seen in [126], similar to Eq. (7.10). The reward is the variation in queueing time between actions.

$$r_t = a \left(\sum_{v \in V_t} \tau_{t_p}^v - \sum_{v \in V_t} \tau_t^v \right) + b \left(\sum_{p \in P_t} \tau_{t_p}^p - \sum_{p \in P_t} \tau_t^p \right) \tag{7.13}$$

7.5.2.3 Waiting Time Adjusted by Demand

Negative sum of waiting time, adding a factor to scale it accordingly with an estimate of the demand ($\hat{d}$).

$$r_t = -\frac{1}{\hat{d}}\left(a \sum_{v\in V_t} \tau_t^v + b \sum_{p\in P_t} \tau_t^p\right) \tag{7.14}$$

7.5.3 Delay based Rewards

These rewards require Modal Prioritisation weights.

7.5.3.1 Delay

Seen in [190]. Negative weighted sum of the delay by all entities. Delay is understood as deviation from the maximum allowed speed. For the pedestrians, the time in queue is used given that, from the point of view of the sensors, pedestrian presence is binary. Assuming a simulator time step of length δ:

$$r_t = -\left(a \sum_{v\in V_t} \sum_{t_e^v}^{t} \delta(1 - \frac{s_t^v}{s_{max}}) + b \sum_{p\in P_t} \tau_t^p\right) \tag{7.15}$$

7.5.3.2 Delta Delay

First seen in [201] and used in [188, 127, 124, 125, 128]. The reward is the variation between actions, as calculated in Eq. (7.13) but using the delay instead of the waiting time.

$$\begin{aligned} r_t = a\left(\sum_{v\in V_t} \sum_{t_e^v}^{t^p} \delta(1 - \frac{s_t^v}{s_{max}}) - \sum_{v\in V_t} \sum_{\max(t_e^v, t^p)}^{t} \delta(1 - \frac{s_t^v}{s_{max}})\right) \\ + b\left(\sum_{p\in P_t} \tau_{t^p}^p - \sum_{p\in P_t} \tau_t^p\right) \end{aligned} \tag{7.16}$$

7.5.3.3 Delay Adjusted by Demand

Same as in Eq. (7.15), introducing a scaling demand term.

$$r_t = -\frac{1}{\hat{d}}\left(a \sum_{v\in V_t} \sum_{t^p}^{t} \delta(1 - \frac{s_t^v}{s_{max}}) + b \sum_{p\in P_t} \tau_t^p\right) \tag{7.17}$$

7.5.4 Average Speed based Rewards

7.5.4.1 Average Speed, Wait Time Variant

The vehicle reward is the average speed of vehicles in the area covered by sensors and normalised by the maximum speed. The pedestrian reward is the minimum between the sum of the waiting time of the pedestrian divided by a theoretical desirable maximum waiting time τ_{max} and 1. This produces two components of the reward $R_p, R_v \in [0, 1]$.

$$r_t = \frac{\sum_{v \in V_t} \frac{s_t^v}{s_{max}}}{|V_t|} + \min\big(\sum_{p \in P_t} \frac{\tau_t^p}{\tau_{max}}, 1\big) \tag{7.18}$$

7.5.4.2 Average Speed, Occupancy Variant

Vehicle reward as in the previous entry. Pedestrian reward is the minimum between the sum of pedestrians waiting divided by a theoretical maximum desirable capacity p_{max} and 1.

$$r_t = \frac{\sum_{v \in V_t} \frac{s_t^v}{s_{max}}}{|V_t|} + \min\big(\sum_{P_t} \frac{p}{p_{max}}, 1\big) \tag{7.19}$$

7.5.4.3 Average Speed Adjusted by Demand, Demand and Occupancy Variants

As in the previous two entries, adding a multiplicative factor equal to the historical estimation of the demand for the current time of the day $\hat{d}$, scaling the reward with the difficulty of the task.

7.5.5 Throughput based Rewards

These rewards require Modal Prioritisation weights.

7.5.5.1 Throughput

The reward is the sum of the pedestrians and vehicles that cleared the intersection since the last action.

$$r_t = a \sum_{t_p}^{t} \rho_v + b \sum_{t_p}^{t} \rho_p \tag{7.20}$$

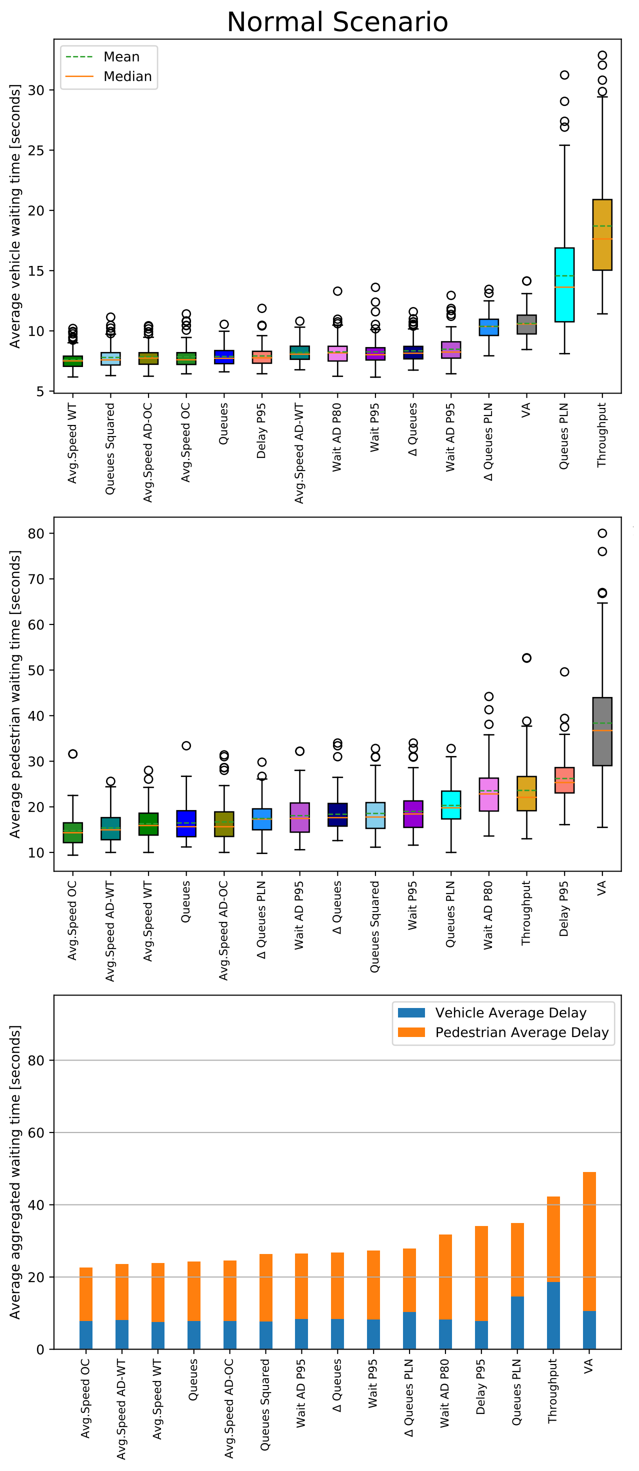


Figure 7.3: Vehicular, pedestrian and aggregated waiting times in seconds for the fifteen best performing agents in the Normal Scenario.

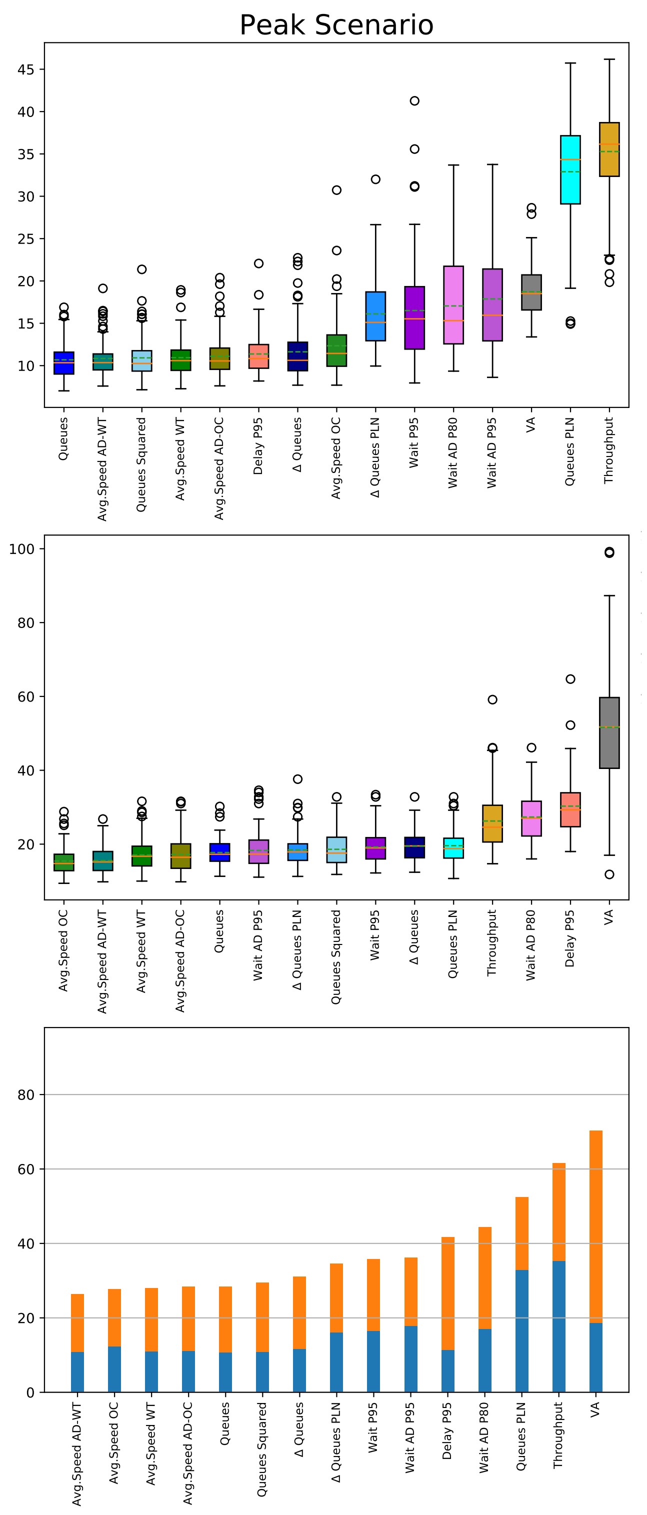


Figure 7.4: Vehicular, pedestrian and aggregated waiting times in seconds for the fifteen best performing agents in the Peak Scenario.

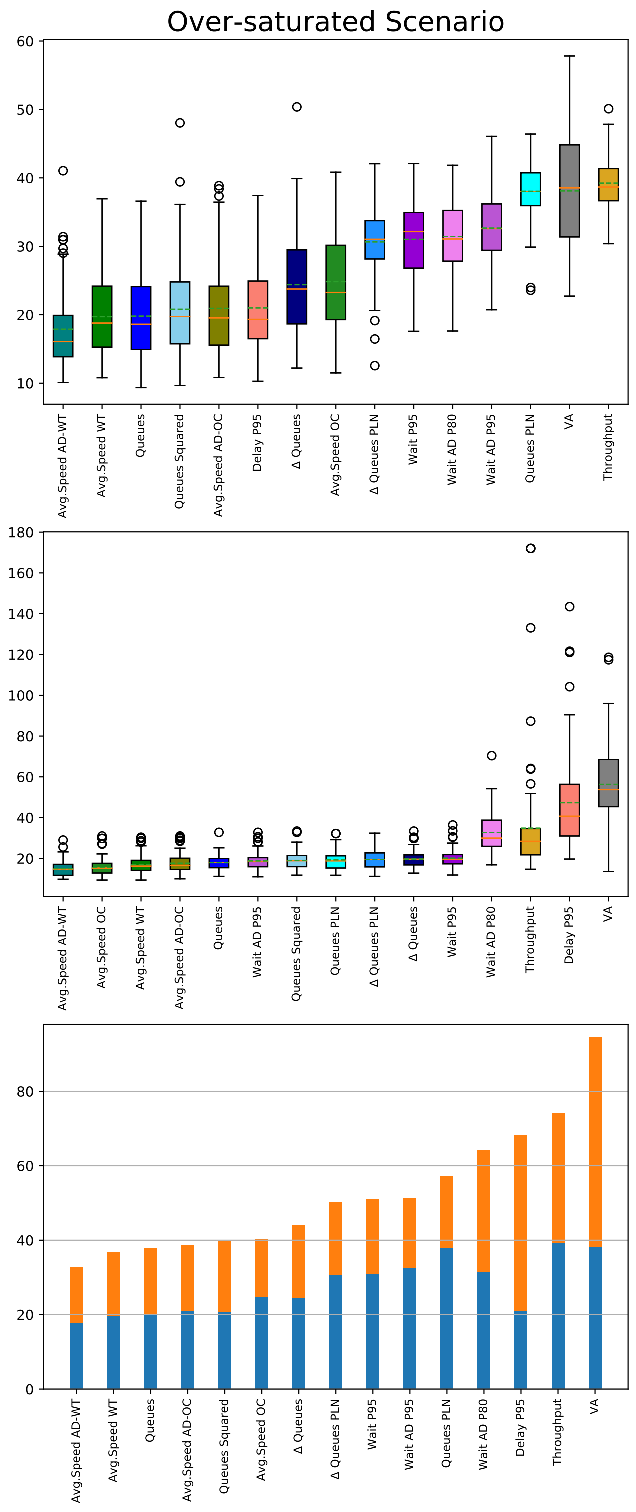


Figure 7.5: Vehicular, pedestrian and aggregated waiting times in seconds for the fifteen best performing agents in the Oversaturated Scenario.

Table 7.2: Average waiting time in seconds across demand scenarios

	Normal Scenario		Peak Scenario		Oversaturated Scenario	
Scenario	Vehicles	Pedestrians	Vehicles	Pedestrians	Vehicles	Pedestrians
Queues	7.87 ± 0.83	16.47 ± 3.95	10.68 ± 2.06	17.73 ± 3.64	19.80 ± 6.01	17.94 ± 3.48
Queues Sq.	7.79 ± 0.93	18.55 ± 4.47	10.92 ± 2.41	18.60 ± 4.81	20.80 ± 6.88	19.02 ± 4.38
Queues PLN	14.57 ± 4.91	20.31 ± 4.94	32.90 ± 6.36	19.59 ± 4.78	38.04 ± 3.93	19.28 ± 4.87
Δ Queues	8.34 ± 1.04	18.37 ± 3.94	11.63 ± 3.09	19.45 ± 3.75	24.40 ± 7.20	19.70 ± 3.80
Δ Queues PLN	10.37 ± 1.10	17.45 ± 3.59	16.11 ± 4.38	18.44 ± 4.45	30.64 ± 5.00	19.49 ± 4.32
Avg. Speed - Wait	7.61 ± 0.84	16.31 ± 3.82	10.94 ± 2.19	17.05 ± 4.67	10.62 ± 1.17	38.36 ± 12.66
Avg. Speed - Occ	7.86 ± 0.94	14.79 ± 3.97	12.34 ± 3.44	15.43 ± 3.84	24.84 ± 7.31	15.56 ± 4.49
Avg. Speed AD - Wait	8.20 ± 0.80	15.37 ± 3.48	10.85 ± 2.11	15.55 ± 3.84	17.89 ± 5.68	14.95 ± 3.80
Avg. Speed AD - Occ	7.85 ± 0.88	16.68 ± 4.83	11.10 ± 2.44	17.20 ± 4.93	20.93 ± 6.83	17.66 ± 5.01
Wait Time	7.80 ± 0.90	41.05 ± 19.40	14.65 ± 4.73	110.34 ± 59.56	28.82 ± 4.83	228.46 ± 159.81
Wait Time P80	8.20 ± 1.26	28.80 ± 9.05	14.94 ± 4.81	54.29 ± 35.00	30.01 ± 4.84	113.68 ± 52.00
Wait Time P95	8.26 ± 1.15	19.00 ± 4.67	16.51 ± 5.94	19.24 ± 4.44	31.02 ± 5.26	20.14 ± 4.16
Wait Time AD	7.83 ± 0.99	56.00 ± 30.22	14.84 ± 4.84	169.11 ± 92.44	27.52 ± 5.01	324.12 ± 212.37
Wait Time AD P80	8.25 ± 1.13	23.52 ± 5.73	17.05 ± 5.91	27.35 ± 6.29	31.43 ± 4.95	32.69 ± 9.67
Wait Time AD P95	8.48 ± 1.19	18.07 ± 4.75	17.88 ± 6.25	18.30 ± 5.02	32.67 ± 5.28	18.78 ± 4.37
Δ Wait Time	9.12 ± 1.23	82.57 ± 36.55	15.28 ± 5.09	326.07 ± 175.84	24.16 ± 6.77	594.03 ± 273.64
Δ Wait Time P80	8.94 ± 1.38	33.35 ± 17.34	16.68 ± 4.65	81.64 ± 49.48	30.38 ± 4.50	149.79 ± 105.07
Δ Wait Time P95	10.02 ± 1.66	42.36 ± 16.59	16.27 ± 5.33	72.27 ± 44.89	26.88 ± 6.22	174.85 ± 109.01
Delay	6.39 ± 0.40	849.52 ± 318.33	8.59 ± 0.89	849.52 ± 318.33	14.43 ± 3.16	849.52 ± 318.33
Delay P80	8.39 ± 1.10	46.78 ± 16.52	11.52 ± 2.36	78.91 ± 35.73	20.93 ± 7.02	143.27 ± 72.39
Delay P95	7.92 ± 0.89	26.21 ± 4.86	11.38 ± 2.42	30.30 ± 7.64	20.99 ± 6.29	47.34 ± 23.27
Delay AD	6.71 ± 0.43	811.38 ± 352.38	8.79 ± 0.96	811.38 ± 352.38	14.08 ± 3.31	811.38 ± 352.38
Delay AD P80	7.74 ± 0.81	44.55 ± 17.51	10.68 ± 1.95	122.05 ± 112.18	18.92 ± 6.46	404.54 ± 252.47
Delay AD P95	7.83 ± 0.84	48.76 ± 24.28	11.62 ± 3.02	180.59 ± 123.18	21.77 ± 6.82	425.35 ± 234.33
Δ Delay	11.18 ± 2.93	211.41 ± 116.86	26.98 ± 6.75	546.51 ± 263.71	34.97 ± 3.45	393.81 ± 267.95
Δ Delay P80	10.62 ± 2.34	66.46 ± 30.32	20.51 ± 6.04	180.64 ± 107.80	29.70 ± 4.97	307.76 ± 218.11
Δ Delay P95	8.23 ± 1.29	99.40 ± 59.76	15.22 ± 4.97	221.92 ± 133.24	25.35 ± 6.43	398.13 ± 240.03
Throughput	18.71 ± 4.79	23.60 ± 6.88	35.28 ± 5.60	26.26 ± 8.14	39.24 ± 3.72	34.86 ± 28.54
Throughput P80	35.53 ± 10.87	51.96 ± 31.20	47.60 ± 5.99	65.91 ± 37.86	47.85 ± 5.15	84.93 ± 49.08
Throughput P95	26.28 ± 8.81	101.07 ± 65.21	56.39 ± 10.72	130.98 ± 84.11	74.10 ± 13.94	74.46 ± 57.96
VA-System D	10.62 ± 1.17	38.36 ± 12.66	18.73 ± 2.92	51.62 ± 16.50	38.10 ± 8.26	56.32 ± 19.58
Max. Occupancy	6.92 ± 0.54	196.09 ± 130.04	10.02 ± 1.75	397.20 ± 213.06	21.57 ± 5.10	596.32 ± 253.80

7.6 Experiments

7.6.1 DQN Agents Training

The training process covers 1500 episodes running for 3000 steps of length $\delta = 0.6$ seconds for a simulated time of 30 minutes (1800 seconds). The traffic demand is increased as the training advances, with the agent progressively facing sub-saturated, near-saturated and over-saturated scenarios, with a minimum of 1 vehicle / 3 seconds (1200 vehicles/h) and a maximum of 1 vehicle / 1.4 seconds (2571 vehicles/h).

For each reward function, 10 copies of the agent are trained, and their performance was compared against two reference systems. These are Maximum Occupancy (MO, longest queue first) and Vehicle Actuated System D (VA-System D) [214] (vehicle-triggered green time extensions), which is commonly used in the UK. The agent performing best against the reference systems in each class is selected for detailed scoring.

7.6.2 Evaluation and Scoring

Each selected agent is tested and its performance scored over 100 copies of 3 different scenarios with different demand levels. Each evaluation is the same length as the training episodes, with the demand kept constant during each run. These three scenarios are aimed to test the agents during normal operation, peak times and over-saturated conditions, and will be henceforth referred to as Normal, Peak and Over-saturated Scenarios. Peak Scenario uses the level of demand observed in the junction that results in saturated traffic conditions under traditional controllers.

The Normal Scenario uses an arrival rate of 1 vehicle / 2.1 seconds (1714 vehicles/h). Peak Scenario uses an arrival rate of 1 vehicle / 1.7 seconds (2117 vehicles/h). Over-saturated Scenario uses an arrival rate of 1 vehicle / 1.4 seconds (2400 vehicles/h)

7.7 Results and Discussion

The results from the simulations of the different reward functions are summarised in Figs. 7.3, 7.4 and 7.5, including the performance of the 15 rewards found to have lower waiting times and seeming most desirable in practice. They are detailed for all 30 rewards in Table 7.2. In Figs. 7.3-7.5, the distribution of pedestrian and vehicle waiting times, and the combination of mean performances for both modes of transportation across 100 repetitions of each demand level are presented. Table

7.2 shows the mean waiting time for each distribution and their standard deviation, also calculated across all three demand levels.

The results display further evidence that RL agents can reach better performance than reference adaptive methods, more evidently so when pedestrians are added. In the case of MO, the bad performance can be framed within the need of having more pedestrians queued than vehicles in any sensor in order to start the pedestrian stage. VA suffers due to its predisposition towards extending green times by 1.5s in the presence of any vehicle, making it more difficult to reach a state in which the pedestrian stage can be started. Both of these characteristics make the vanilla reference methods less suited for intersections including pedestrians than the RL methods presented in Figs. 7.3-7.5, especially in situations of high demand.

At a global level, methods based on maximisation of the average network speed show the lowest global waiting times for pedestrians and vehicles combined across all demand levels, while also obtaining some of the lowest spreads, as shown in the case with no pedestrians [6]. Their performance is closely followed by Queue minimisation, which obtains the lowest average waiting times for vehicles in the Normal and Peak Scenarios, but falls behind in Over-saturated conditions and when dealing with pedestrians. Queue Squared minimisation has a comparable yet slightly worse performance, followed by Delta Queues and Delta Queues PLN. This last reward has been shown to obtain better performance with higher demand, which is consistent with it generating less variance in the reward distribution, since it is approximating the arrival rate minus the exit rate of the intersection given an action, making it an option that could be further explored for permanently congested intersections.

Prioritised rewards based on Waiting Time show acceptable performance, but also a high sensitivity to the changes in the modal prioritisation weights. This is similar to the behaviour shown by the Delay-based rewards, which overall perform worse, potentially due to the need to use Wait Time for pedestrians, mixing the state variables, although this does not seem to be an issue for average speed based rewards. Without a weight configuration heavily favouring the pedestrians, these reward functions were found to converge for vehicles only, obtaining the lowest vehicle waiting times overall in the case of the Delay functions, at the expense of rarely, if ever, serving pedestrians. The suitability of a given choice of modal prioritisation weights is further affected by the functional form of the reward. In the results, it can be observed that while in general the choice $(a = 5, b = 95)$ obtains better results (e.g. Wait Time and Delay), for certain functional choices the prioritisation $(a = 20, b = 80)$ is the one producing the best results, which would not

Table 7.3: Results of independent two-sample t-tests for failed prioritisations.

Series	Scenario	Mean P80	SD P80	Mean P90	SD P90	t-statistic	p-value
	Normal	33.35	17.34	42.36	16.59	-3.7352	0.0002*
Delta Wait Time	Peak	81.64	49.48	72.27	44.89	1.3959	0.1643
	Oversaturated	149.79	105.07	174.85	109.01	-1.6468	0.1012
	Normal	44.55	17.51	48.76	24.28	-1.3991	0.1635
Delay AD	Peak	122.05	112.18	180.59	123.18	-3.4934	0.0006*
	Oversaturated	404.54	252.47	425.35	234.33	-0.601	0.5485
	Normal	66.46	30.32	99.40	59.76	-4.8907	2.60E-6*
Delta Delay	Peak	180.64	107.80	221.92	133.24	-2.394	0.0175
	Oversaturated	307.76	218.11	398.13	240.03	-2.7724	0.0061*
	Normal	51.56	31.24	101.07	65.21	-6.7593	3.31E-10*
Throughput	Peak	45.91	37.86	130.94	84.11	-7.0197	9.28E-11*
	Oversaturated	84.93	49.08	74.46	57.96	1.3719	0.1717

be the case if the suitability of the weights was only affected by the relative demand ratios between vehicles and pedestrians. This is the case with Throughput based functions, which, unlike the Wait and Delay functions, obtained lower waiting times with equal modal weights, and a general wait time increase as the weights become more skewed towards the pedestrians.

Rewards using Differences in Delay or Wait Time, having good performance in the literature, were found either not to converge for pedestrians or to produce mediocre results.

The addition of a demand scaling term generates, in general, a slight improvement in waiting times across the rewards using Wait Time and Delay, particularly at higher demand levels.

The subset of the proposed reward functions for which prioritisation weights seemed ineffective includes: Delta Wait Time, Delay AD, Delta Delay, and Throughput. This can raise questions regarding whether these results actually mean that an increase in the evaluative weight of pedestrians can actually have a negative impact over the performance of the agents.

To check this, a series of two-tailed independent sample Welch's t-tests, which do not assume equal variance, were carried out between the two pedestrian prioritisations for each scenario of the affected reward functions. In them the null hypothesis is that their distributions of waiting times have equal means, meaning that the prioritisation caused no significant observable effect in terms of average waiting times. In all cases, both samples were of size 100, implying df=198, and the significance level is set at 0.01. Statistical significance of the results is indicated with an asterisk (*).

The results are displayed on Table 7.3, where it can be observed that in 50% of the cases, a higher weight being allocated to pedestrian generated a statistically significant increase in waiting times. However, there is no discernible pattern regarding in which cases this is true, given the lack of consistency in the differences being significant across reward functions or scenarios.

Given these results, there is no definitive evidence to state that in the case of the referred reward functions, the changes to the modal prioritisation weights had a consistent statistically significant effect neither positive nor negative in the general case. This, when taking into account that these same agents did achieve reasonable levels of control for one of the tasks, points in the direction of the training of the agents, in these cases, not having achieved convergence to a policy capable of simultaneously managing both tasks.

Overall, the dominance shown by speed maximisation methods could be attributed to several factors. Average Speed based functions, as Queue based functions, obtain an instantaneous snapshot of a quantity that does not intrinsically grow over time, as opposed to Delay, Wait and Throughput, so it exclusively encodes information about the moment the action is requested. It can also be argued that speed maximisation rewards are not affected by the correspondence between agent actions and time-steps in the environment . In the specific case of RL for UTC, the values of the reward received by the agent using a reward based on Queues, Delay, Wait or Throughput are a function of the length of the phase that generated them, making them theoretically less suitable for the underlying MDP than speed maximisation.

Lastly, speed maximisation and queue minimisation have an extra benefit that makes them into serious candidates for expansive real-world use: the lack of need for modal prioritisation tuning. One of the main selling points of ML and RL methods stems from their ability to perform equal or better than traditional systems at a lower cost in a variety of situations. However, a lengthy manual tuning process in order to find the exact weights for a given junction is not only untranslatable to any other intersection, but may also not result in reduced planning and execution times compared with traditional control. The lack of need for manual tuning, especially in the case of Average Speed functions, which are specifically crafted to avoid this, make them in our view more applicable in a wider and faster manner than any of the other reward functions here presented.

Regarding robustness, given that the agents used in this chapter do not differ significantly from those presented in the previous chapter, other than the inclusion of pedestrians, the same considerations regarding robustness and security apply here as they did in Section 6.8.

One limitation of the research presented in this chapter is that the results are only relevant in the case of value-based DQN agents as introduced in Section 7.3 and Section 7.4, and not for CNN or Policy Gradient architectures. This work could be extended to account for other modes of transportation, performing a similar optimisation based on different vehicle classes (buses, cyclists, personal vehicles, trucks, etc.). The optimisation could seek to prioritise them based on different criteria (e.g. priority to cyclists and public transport during rush hours or weighting vehicles according to the expected number of passengers).

CHAPTER 8

Conclusions

Modern Intelligent Transportation Systems generate and allow for the logging of vast amounts of data, which can in turn be used to further improve and optimise these systems. This is of direct interest for users, operators and stakeholders of any industry that requires transportation of goods at any level, and those that provide transportation as a service. Throughout this thesis we have studied different applications of Data Science to improve modern Intelligent Transportation Systems in both motorway and urban settings.

This has involved investigating the characteristics of the different regimes of motorway traffic, looking for ways in which these can be identified and isolated into components from initial series of travel times, and how we can treat said components in order to extract valuable information for estimation and forecasting.

In Chapter 3 we attempted this separation by normalising the travel times per section of motorway and then defining a threshold over which points were classified as spikes. We then used a combination of non-parametric regression and spectral methods to estimate the underlying behaviour of the different motorway sections and generate traffic profiles. These generated profiles, demonstrated a higher forecasting power than similar methods currently used in the motorways in the United Kingdom. During this chapter, it was uncovered that this separation mechanism, while effective, failed to extract some valuable information into the spikes signal.

This fact was latter addressed in Chapter 4, where we looked at how Wavelets can be used to perform the separation of the travel times signal into components looking at the statistical structure of the signal in the frequency-time space. Here, we defined certain data points as outliers with respect to the baseline dynamics of the section of road, classifying them as spikes in a more sophisticated way than in the previous chapter. This decomposition was seamlessly integrated into the previously introduced forecasting algorithm to generate predictions with higher accuracy than the currently published profiles even in the absence of granular per-site tuning.

This proposed method could benefit of further optimisation on a per-link level by performing a more sophisticated analysis of the distribution of frequencies needed to approximate points suffering recurrent or non-recurring congestion. Further, given that the prediction frequency and resolution of modern short-term forecasting methods is similar, it would be interesting to compare them in equal footing, since the proposed model achieving over 80% of predictions within 5% of the ground truth could make it competitive also in this setting.

Overall, Chapters 3 and 4 aimed at producing new purpose-built estimation and forecasting methods that could substitute one specific, currently used method for travel time profile estimation in UK motorways. The methods here developed discard the complex time based segmentation requirements of the current method that they try to improve upon, instead making a more sophisticated analysis of traffic regimes and their cyclic variation over time. On a global view, they clearly obtain better performance scores, which was the main goal and metric, than the current method and other baseline approaches considered.

In Chapter 5 the focus was moved to an urban setting, in which different architectures for value-based and policy-based reinforcement learning agents are compared for the task of controlling the signalled intersections in several maps. Here we observed how various reinforcement learning configurations have the ability to outperform even well tuned commercial traffic optimisation algorithms currently in use, providing effective adaptive real-time control. In this chapter, we also tested the limits of training uncoordinated agents trying to control entire networks, observing how simultaneously training all agents in a network can generate instabilities in the demand received by agents downstream preventing effective learning. We also experimentally checked that for very regular networks, copies of the same agent can be deployed in neighbouring intersections achieving satisfactory control that can beat commercial systems. Some of these experiments would benefit from replication in a different simulator that allows per-lane metrics retrieval. Most of the agents presented there could also benefit from recently developed extension modules for reinforcement learning agent, increasing their convergence ability and control performance. Lastly, it would be very interesting to explore different avenues to include inter-agent communication in extended networks, investigating what information exchanges provide the highest utility to neworked agents. This could also be compared with recently developed self-organising methods that provide a different paradigm than what was used in traditional controllers.

Lastly, Chapter 6 and Chapter 7 focused on how to obtain the pers performance when controlling a real-world intersection in Manchester, UK via reinforcement learning. These two chapters explored the performance of agents using different reward functions for the intersection modelled without and with pedestrians respectively. In both cases we found that reward functions that encourage higher average speeds throughout an area of interest in the junction provide the lowest waiting times for both vehicles and pedestrians. We also found that reward functions previously indicated in the literature as great options, performed sub-optimally in our tests. While the scope for modification of the agent was limited due to its origin in commercial applications, it would be interesting to further modify the inputs that the agents receive to make them simpler, hopefully obtaining better convergence properties. This same approach, maximising speed, could be applied to wider networks providing area control, steering the agents towards the discovery of the creation and use of green waves.

Some of the agents developed in Chapter 7 have been since deployed to control real traffic for extended periods of time, in the order of hours, which is a great success in bringing research into the real world.

Regarding the relationship of this work on RL for TSC with prior art, the work here presented does not aim to provide a complete approach to building complete UTCs, but to contribute to the exploration of a new paradigm of control that has been developing over the past few years, both in terms of suitability of theoretical approaches in the field (Chapter 5), and in terms of the practical considerations needed to move it to the real-world (Chapters 6 and 7). From the different results and discussions we can conclude that RL for TSC is showing great capacity for improvement over previous isolated systems, both in terms of delay minimisation and simplicity of operation and configuration. Similarly, it shows potential to also outperform distributed systems once the issues encountered in the last experiment of Chapter 5 are overcome. Further, the agents obtaining the best performance in Chapter 7, have since been deployed in the real world, obtaining improvements over previous systems and demonstrating their potential to be replacement for the current generation of methods for urban traffic control at intersections.

Bibliography

[1] Alvaro Cabrejas Egea, Peter De Ford, and Colm Connaughton. Estimating Baseline Travel Times for the UK Strategic Road Network. In *IEEE 21st International Conference on Intelligent Transportation Systems, Proceedings, ITSC*, volume 2018-Novem, pages 531–536. IEEE, 2018. ISBN 9781728103235. doi: 10.1109/ITSC.2018.8569924.

[2] Alvaro Cabrejas Egea and Colm Connaughton. Wavelet Augmented Regression Profiling (WARP): improved long-term estimation of travel time series with recurrent congestion. In *IEEE 23rd International Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2020. doi: 10.1109/itsc45102.2020.9294318.

[3] Alvaro Cabrejas Egea and Colm Connaughton. Wavelet Augmented Regression Profiling (WARP): improved long-term estimation of travel time series with recurrent congestion. *arXiv preprint arXiv:2006.13072*, 2020.

[4] Alvaro Cabrejas Egea, Raymond Zhang, and Neil Walton. Reinforcement Learning for Traffic Signal Control: Comparison with Commercial Systems. In *14th Conference on Transport Engineering*, 2021. doi: pending.

[5] Alvaro Cabrejas-Egea, Raymond Zhang, and Neil Walton. Reinforcement Learning for Traffic Signal Control: Comparison with Commercial Systems. *arXiv preprint arXiv: 2104.10455*, 2021.

[6] Alvaro Cabrejas Egea, Shaun Howell, Maksis Knutins, and Colm Connaughton. Assessment of Reward Functions for Reinforcement Learning Traffic Signal Control under Real-World Limitations. In *IEEE International Conference on Systems, Man and Cybernetics, Proceedings, SMC*. doi: 10.1109/smc42975.2020.9283498.

[7] Alvaro Cabrejas-Egea, Shaun Howell, Maksis Knutins, and Colm Connaughton. Assessment of Reward Functions for Reinforcement Learning Traffic Signal Control under Real-World Limitations. *arXiv preprint arXiv:2008.11634*, 2020.

[8] Alvaro Cabrejas Egea and Colm Connaughton. Reward Functions for Real-World Pedestrian and Vehicular Intersection Control through Reinforcement Learning. In *IEEE 24th International Conference on Intelligent Transportation Systems, ITSC proceedings*. IEEE, 2021. doi: pending.

[9] Alvaro Cabrejas-Egea and Colm Connaughton. Assessment of Reward Functions in Reinforcement Learning for Multi-Modal Urban Traffic Control under Real-World limitations. *arXiv preprint arXiv:2010.08819*, 2020.

[10] EU Parliament. Directive 2010/41/EU of the European Parliament and of the Council of 7 July 2010. *Official Journal of the European Union*, (2010/40/EU): 352–355, 2010. doi: 10.1007/978-1-137-54482-7_33.

[11] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[12] Alvaro Cabrejas Egea, Raymond Zhang, and Neil Walton. Code Repository - Vissim RL Traffic Light Control, 2019. URL https://github.com/ACabrejas/Vissim{_}RL{_}Traffic{_}Lights{_}Control.

[13] Highways England. Highways England Strategic Road Network Initial Report. Technical Report December, 2017. URL https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment{_}data/file/666884/Highways{_}England{_}Strategic{_}Road{_}Network{_}Initial{_}Report{_}-{_}WEB.pdf.

[14] The Highways Agency. National Traffic Information Service Publish Services. pages 1–10, 2011. doi: NISPTIH009v1.0.

[15] The Highways Agency. Specification for the Installation of Detector Loops on Motorways and All-Purpose Trunk Roads. Technical Report F, 2006. URL http://www.ukroads.org/webfiles/mch1540f.pdf.

[16] B DALLA CHIARA, F DEFLORIO, and I PINNA. A comparing analysis of traffic monitoring systems. *EUROTRANSPORT*, (4), 2010.

[17] Do H Nam and Donald R Drew. Traffic dynamics: Method for estimating freeway travel times in real time from flow measurements. *Journal of Transportation Engineering*, 122(3):185–191, 1996.

[18] Benjamin Coifman. Estimating travel times and vehicle trajectories on freeways using dual loop detectors. *Transportation Research Part A: Policy and Practice*, 36(4):351–364, 2002.

[19] J. W.C. Van Lint and N. J. der Zijjpp. Improving a travel-time estimation algorithm by using dual loop detectors. *Transportation Research Record*, 1855 (1):41–48, 2003.

[20] Benjamin Coifman, David Beymer, Philip McLauchlan, and Jitendra Malik. A real-time computer vision system for vehicle tracking and traffic surveillance. *Transportation Research Part C: Emerging Technologies*, 6(4):271–288, 1998.

[21] Baher Abdulhai and Seyed M Tabib. Spatio-temporal inductance-pattern recognition for vehicle re-identification. *Transportation Research Part C: Emerging Technologies*, 11(3-4):223–239, 2003.

[22] Usue Mori, Alexander Mendiburu, Maite Álvarez, and Jose A. Lozano. A review of travel time estimation and forecasting for Advanced Traveller Information Systems. *Transportmetrica A: Transport Science*, 11(2):119–157, 2015. ISSN 23249943. doi: 10.1080/23249935.2014.932469.

[23] Mengert P, Brown P, and Yuan L. Prediction Algorithms for Urban Traffic Control. Technical Report DOT-TSC-HW913-PM79-6, Transportation Systems Center, 1979.

[24] George E P Box, Gwilym M Jenkins, and Gregory C Reinsel. *Time series analysis: forecasting and control*, volume 734. John Wiley & Sons, 2011.

[25] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2013.

[26] Jinsoo You and Tschangho John Kim. Towards developing a travel time forecasting model for location-based services: A review. In *Methods and Models in Transport and Telecommunications*, pages 45–61. Springer, 2005.

[27] Oana Niculaescu. Classifying data with decision trees, 2018. URL `https://elf11.github.io/2018/07/01/python-decision-trees-acm.html`.

[28] Robert B Cleveland, William S Cleveland, Jean E McRae, and Irma Terpenning. STL: A seasonal-trend decomposition. *Journal of official statistics*, 6 (1):3–73, 1990.

[29] James W. Cooley and John W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297, 1965. ISSN 00255718. doi: 10.2307/2003354.

[30] Matt Hall. Resolution and uncertainty in spectral decomposition. *First Break*, 24(12), 2006.

[31] Hamid Reza Karimi, Witold Pawlus, and Kjell G Robbersmyr. Signal reconstruction, modeling and simulation of a vehicle full-scale crash test based on Morlet wavelets. *Neurocomputing*, 93:88–99, 2012.

[32] Jean Morlet. Sampling theory and wave propagation. In *Issues in acoustic Signa lmage processing and recognition*, pages 233–261. Springer, 1983.

[33] Alex Grossmann and Jean Morlet. Decomposmon of hardy functions into square integrable wavelets of constant shape. *Fundamental Papers in Wavelet Theory*, 15(4):126–139, 2009. doi: 10.1515/9781400827268.126.

[34] Gerald Kaiser. *A friendly guide to wavelets*. Springer Science & Business Media, 2010.

[35] Paul S Addison. Introduction to redundancy rules: the continuous wavelet transform comes of age, 2018.

[36] MathWorks. Continuous 1-D wavelet transform, 2016. URL https://uk.mathworks.com/help/wavelet/ref/cwt.html?s{_}tid=doc{_}ta.

[37] Petr Klapetek, David Necas, and Christopher Anderson. Gwyddion user guide. *Czech Metrology Institute*, 2007:2009, 2004.

[38] Patrick M Crowley. An intuitive guide to wavelets for economists. *Bank of Finland Research Discussion Paper*, (1), 2005.

[39] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. ISSN 14764687. doi: 10.1038/nature14236.

[40] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda

Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. ISSN 14764687. doi: 10.1038/nature16961.

[41] The AlphaStar Team. AlphaStar: Grandmaster level in StarCraft II using multi-agent reinforcement learning, 2019. URL https://deepmind.com/blog/article/AlphaStar-Grandmaster-level-in-StarCraft-II-using-multi-agent-reinforcement-learning.

[42] Gerald Tesauro. TD-Gammon: A Self-Teaching Backgammon Program. In Alan F Murray, editor, *Applications of Neural Networks*, pages 267–285, Boston, MA, 1995. Springer US. ISBN 978-1-4757-2379-3. doi: 10.1007/978-1-4757-2379-3_11. URL https://doi.org/10.1007/978-1-4757-2379-3{_}11.

[43] Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*, volume 5. Athena Scientific Belmont, MA, 1996.

[44] Andrew Hamilton, Ben Waterson, Tom Cherrett, Andrew Robinson, and Ian Snell. The evolution of urban traffic control: changing policy and technology. *Transportation Planning and Technology*, 36(1):24–43, 2013. ISSN 03081060. doi: 10.1080/03081060.2012.745318.

[45] R A Vincent and J R Peirce. *'MOVA': Traffic Responsive, Self-optimising Signal Control for Isolated Intersections*. Traffic Management Division, Traffic Group, Transport and Road Research, 1988.

[46] PTV Group. PTV Balance. URL http://vision-traffic.ptvgroup.com/en-us/products/ptv-balance/.

[47] B Friedrich, J Mertz, O Ernhofer, M Clark, C Toomey, T McLean, and Others. TABASCO Deliverable 9.4: Urban Traffic Control with PT Priority: Final Evaluation Report, 1998.

[48] D. I. Robertson. A traffic network study tool. *RRL report*, 253, 1969.

[49] Dennis I Robertson. TRANSYT: a traffic network study tool. 1969.

[50] D. I. Robertson and P. Gower. User Guide To Transyt Version 6. Technical Report 255, 1977. URL https://trl.co.uk/sites/default/files/LR888.pdf.

[51] P B Hunt, D I Robertson, R D Bretherton, and M Cr Royle. The SCOOT on-line traffic signal optimisation technique. *Traffic Engineering & Control*, 23(4), 1982.

[52] P. B. Hunt, D. I. Robertson, R. D. Bretherton, and R. I. Winton. Scoot - a Traffic Responsive Method of Coordinating Signals. *TRRL Laboratory Report (Transport and Road Research Laboratory, Great Britain)*, pages 1–45, 1981. ISSN 03051293. doi: citeulike-article-id:563207.

[53] Dipl Joachim Mertz. *Ein mikroskopisches Verfahren zur verkehrsadaptiven Knotenpunktsteuerung mit Vorrang des öffentlichen Verkehrs Inhalt*. Technische Universität München, 2000.

[54] P R Lowrie. Scats, sydney co-ordinated adaptive traffic system: A traffic responsive method of controlling urban traffic. 1990.

[55] Vito Mauro and C Di Taranto. Utopia. *Control, computers, communications in transportation*, pages 245–252, 1990.

[56] F Donati, V Mauro, G Roncolini, and M Vallauri. A Hierarchical-Decentralized Traffic Light Control System. The First Realization: "Progetto Torino". *IFAC Proceedings Volumes*, 17(2):2853–2858, 1984. ISSN 1474-6670. doi: https://doi.org/10.1016/S1474-6670(17)61415-0. URL http://www.sciencedirect.com/science/article/pii/S1474667017614150.

[57] Pitu Mirchandani and Larry Head. RHODES: A real-time traffic signal control system: architecture, algorithms, and analysis. *Transportation Research Part C: Emerging Technologies*, 9(6):415–432, 2001.

[58] K. Larry Head. Event-based short-term traffic flow prediction model. *Transportation Research Record*, (1510):45–52, 1995. ISSN 03611981.

[59] N.H. Gartner. OPAC: Strategy for Demand-responsive Decentralized Traffic Signal Control. *IFAC Proceedings Volumes*, 23(2):241–244, 1990. ISSN 14746670. doi: 10.1016/s1474-6670(17)52677-4. URL http://www.sciencedirect.com/science/article/pii/S1474667017526774.

[60] J. J. Henry, J. L. Farges, and J. Tuffal. Prodyn Real Time Traffic Algorithm. In *IFAC Proceedings Series*, volume 16, pages 305–310. Elsevier, sep 1984. ISBN 0080293654. doi: 10.1016/s1474-6670(17)62577-1. URL http://dx.doi.org/10.1016/S1474-6670(17)62577-1.

[61] Stephen F Smith and Gregory J Barlow. SURTRAC : Scalable Urban Traffic Control. *Transport Researach Board*, 2013.

[62] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie WieBner. Microscopic traffic simulation using sumo. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2575–2582. IEEE, 2018.

[63] Volodymyr Mnih, Adria Puigdomenech Badia, Lehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *33rd International Conference on Machine Learning, ICML 2016*, 4:2850–2869, 2016.

[64] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017. URL http://arxiv.org/abs/1711.03938.

[65] Cathy Wu, Aboudy Kreidieh, Kanaad Parvate, Eugene Vinitsky, and Alexandre M Bayen. Flow: A Modular Learning Framework for Autonomy in Traffic. *CoRR*, abs/1710.0, 2017. URL http://arxiv.org/abs/1710.05465.

[66] UK Department for Transport. General Principles of Traffic Control by Light Signals. *Traffic Advisory Leaflet 1/06*, 1-4(March), 2006. URL http://tsrgd.co.uk/documents/traffic-advisory-leaflets.

[67] R. A. Vincent and J. R. Peirce. 'MOVA': Traffic responsive, self-optimising signal control for isolated intersections. Technical Report 170, 1988.

[68] UK Department for Transport. The " MOVA " signal control system. (3/97): 1–7, 1997.

[69] UK Department for Transport. The " SCOOT " Urban Traffic Control System. *Traffic Advisory Leaflet 7/99*, 1999.

[70] G. F. Newell. Memoirs on highway traffic flow theory in the 1950s. *Operations Research*, 50(1):173–178, 2002. ISSN 0030364X. doi: 10.1287/opre.50.1.173.17802.

[71] J G Wardrop and J I Whitehead. Correspondence. Some Theoretical Aspects of Road Traffic Research. In *Proceedings of the Institution of Civil*

Engineers, volume 1, pages 767–768. Thomas Telford, 1952. doi: 10.1680/ipeds.1952.11362.

[72] F. V. Webster. Traffic signal settings. *Road Research Technical Paper, No.39, Road Research Laboratory, London*, (39):44, 1958. URL https://trid.trb.org/view.aspx?id=113579.

[73] Alan J. Miller. Settings for Fixed-Cycle Traffic Signals. *or*, 14(4):373, 1963. ISSN 14732858. doi: 10.2307/3006800.

[74] G. F. Newell. Approximation Methods for Queues with Application to the Fixed-Cycle Traffic Light. *SIAM Review*, 7(2):223–240, 1965. ISSN 0036-1445. doi: 10.1137/1007038. URL http://www.jstor.org/stable/2027270.

[75] Anders Martin-Löf. Computation of an Optimal Control for a Signalized Traffic Intersection. *Transportation Science*, 1(1):1–5, 1967. ISSN 0041-1655. doi: 10.1287/trsc.1.1.1.

[76] Donald R. McNeil. A solution to the fixed-cycle traffic light problem for compound Poisson arrivals. *Journal of Applied Probability*, 5(3):624–635, 1968. ISSN 0021-9002. doi: 10.2307/3211926.

[77] M. A.A. Boon, A. J.E.M. Janssen, J. S.H. van Leeuwaarden, and R. W. Timmerman. Pollaczek contour integrals for the fixed-cycle traffic-light queue. *Queueing Systems*, 91(1-2):89–111, 2019. ISSN 15729443. doi: 10.1007/s11134-018-9595-9. URL https://doi.org/10.1007/s11134-018-9595-9.

[78] Maury Bramson. Stability of queueing networks. *Probability Surveys*, 5(1): 169–345, 2008. ISSN 15495787. doi: 10.1214/08-PS137.

[79] Stefan Lämmer and Dirk Helbing. Self-control of traffic lights and vehicle flows in urban road networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(4):P04019, 2008. ISSN 17425468. doi: 10.1088/1742-5468/2008/04/P04019.

[80] Nathan H. Gartner, John D.C. Little, and Henry Gabbay. Optimization of traffic signal settings by mixed-integer linear programming: Part I: The network coordination problem. *Transportation Science*, 9(4):344–363, 1975. ISSN 00411655. doi: 10.1287/trsc.9.4.344.

[81] Nathan H. Gartner, John D.C. Little, and Henry Gabbay. Optimization of Traffic Signal Settings By Mixed-Integer Linear Programming - 2. the Network

Synchronization Problem. *Transportation Science*, 9(4):344–363, 1975. ISSN 00411655. doi: 10.1287/trsc.9.4.344.

[82] Yann Dujardin, Florence Boillot, Daniel Vanderpooten, and Pierre Vinant. Multiobjective and multimodal adaptive traffic light control on single junctions. In *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, pages 1361–1368. IEEE, 2011. ISBN 9781457721984. doi: 10.1109/ITSC.2011.6082977.

[83] Bart De Schutter. Optimal traffic light control for a single intersection. *Proceedings of the American Control Conference*, 3(3):2195–2199, 1999. ISSN 07431619. doi: 10.1109/acc.1999.786344.

[84] Yanning Li, Edward Canepa, and Christian Claudel. Optimal control of scalar conservation laws using linear/quadratic programming: Application to transportation networks. *IEEE Transactions on Control of Network Systems*, 1(1): 28–39, 2014. ISSN 23255870. doi: 10.1109/TCNS.2014.2304152.

[85] Toshihiko Oda, Toru Otokita, Tomomitsu Tsugui, and Yoshito Mashiyama. Application of Simulated Annealing to Optimization of Traffic Signal Timings. *IFAC Proceedings Volumes*, 30(8):733–736, 1997. ISSN 14746670. doi: 10.1016/s1474-6670(17)43908-5.

[86] Sadayoshi Mikami and Yukinori Kakazu. Genetic reinforcement learning for cooperative traffic signal control. In *IEEE Conference on Evolutionary Computation - Proceedings*, volume 1, pages 223–228. IEEE, 1994. doi: 10.1109/icec.1994.350012.

[87] Ghulam Q. Memon and A. G.R. Bullen. Multivariate optimization strategies for real-time traffic control signals. *Transportation Research Record*, 1554 (1554):36–42, 1997. ISSN 03611981. doi: 10.1177/0361198196155400105.

[88] Ayad M Turky, M S Ahmad, Mohd Zaliman Mohd Yusoff, and Baraa T Hammad. Using genetic algorithm for traffic light control system with a pedestrian crossing. In *International Conference on Rough Sets and Knowledge Technology*, pages 512–519. Springer, 2009.

[89] Mike Maher, Ronghui Liu, and Dong Ngoduy. Signal optimisation using the cross entropy method. *Transportation Research Part C: Emerging Technologies*, 27:76–88, 2013. ISSN 0968090X. doi: 10.1016/j.trc.2011.05.018.

[90] Ella Bingham. Reinforcement learning in neurofuzzy traffic signal control. *European Journal of Operational Research*, 131(2):232–241, 2001. ISSN 03772217. doi: 10.1016/S0377-2217(00)00123-5.

[91] Nathan H. Gartner. OPAC: A demand-responsive strategy for traffic signal control. *Transportation Research Record*, 1983(906):75–81, 1983. ISSN 03611981.

[92] K. Aboudolas, M. Papageorgiou, and E. Kosmatopoulos. Store-and-forward based methods for the signal control problem in large-scale congested urban road networks. *Transportation Research Part C: Emerging Technologies*, 17 (2):163–174, 2009. ISSN 0968090X. doi: 10.1016/j.trc.2008.10.002.

[93] K. Aboudolas, M. Papageorgiou, A. Kouvelas, and E. Kosmatopoulos. A rolling-horizon quadratic-programming approach to the signal control problem in large-scale congested urban road networks. *Transportation Research Part C: Emerging Technologies*, 18(5):680–694, 2010. ISSN 0968090X. doi: 10.1016/j.trc.2009.06.003.

[94] T. Tettamanti, I. Varga, B. Kulcsár, and J. Bokor. Model predictive control in urban traffic network management. In *2008 Mediterranean Conference on Control and Automation - Conference Proceedings, MED'08*, pages 1538–1543, jun 2008. ISBN 9781424425051. doi: 10.1109/MED.2008.4602084.

[95] Tamás Tettamanti and István Varga. Distributed traffic control system based on model predictive control. *Periodica Polytechnica Civil Engineering*, 54(1): 3–9, 2010. ISSN 15873773. doi: 10.3311/pp.ci.2010-1.01.

[96] Tams Tettamanti, Istvan Varga, and Tamas Peni. MPC in Urban Traffic Management. *Model Predictive Control*, 2010. doi: 10.5772/9922.

[97] Tung Le, Hai L. Vu, Yoni Nazarathy, Quoc Bao Vo, and Serge Hoogendoorn. Linear-quadratic model predictive control for urban traffic networks. *Transportation Research Part C: Emerging Technologies*, 36(0):498–512, 2013. ISSN 0968090X. doi: 10.1016/j.trc.2013.06.021.

[98] Shu Lin, Bart De Schutter, Yugeng Xi, and Hans Hellendoorn. Fast model predictive control for urban road networks via MILP. *IEEE Transactions on Intelligent Transportation Systems*, 12(3):846–856, 2011. ISSN 15249050. doi: 10.1109/TITS.2011.2114652.

[99] Richard Bellman. The Theory of Dynamic Programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954. ISSN 02730979. doi: 10.1090/S0002-9904-1954-09848-8.

[100] John N. Tsitsiklis and Benjamin Van Roy. Analysis of temporal-difference learning with function approximation. *Advances in Neural Information Processing Systems*, 42(5):1075–1081, 1997. ISSN 10495258.

[101] Ronald J. Willia. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8(3):229–256, 1992. ISSN 15730565. doi: 10.1023/A:1022672621406.

[102] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. *Advances in Neural Information Processing Systems*, pages 1008–1014, 2000. ISSN 10495258.

[103] Richard S. Sutton. Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3(1):9–44, 1988. ISSN 15730565. doi: 10.1023/A:1022633531479.

[104] Hado V Hasselt. Double Q-learning. In *Advances in neural information processing systems*, pages 2613–2621, 2010.

[105] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-Learning. In *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, pages 2094–2100, 2016. ISBN 9781577357605.

[106] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[107] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Frcitas. Dueling Network Architectures for Deep Reinforcement Learning. In *33rd International Conference on Machine Learning, ICML 2016*, volume 4, pages 2939–2947, 2016. ISBN 9781510829008.

[108] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

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

[110] G A Rummery and M Niranjan. *On-line q-learning using connectionist systems cued/f-infeng/tr 166*, volume 37. University of Cambridge, England, 1994.

[111] R S Sutton and a G Barto. Temporal credit assignment in reinforcement learning. *Computer Science*, page 210, 1984.

[112] C. P. Pappis and Ebrahim H. Mamdani. A Fuzzy Logic Controller for a Traffic Junction. *IEEE Transactions on Systems, Man and Cybernetics*, 7(10):707–717, 1977. ISSN 21682909. doi: 10.1109/TSMC.1977.4309605.

[113] Charles W. Thorpe, Thomas L. Anderson. Traffic Light Control Using SARSA with Three State Representation. Technical report, IBM Technical Report, 1996.

[114] Baher Abdulhai and Rob Pringle. Machine learning based adaptive signal control using autonomous Q-learning agent. In *Proceeding of the IASTED International Conference. Intelligent Systems and Control. Honolulu, Hawaii, USA*, pages 320–327, 2000.

[115] Mark D. Pendrith. Distributed reinforcement learning for a traffic engineering application. In *Proceedings of the International Conference on Autonomous Agents*, pages 404–411. Citeseer, 2000. doi: 10.1145/336595.337554.

[116] Dipti Srinivasan, Min Chee Choy, and Ruey Long Cheu. Neural networks for real-time traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 7(3):261–272, 2006. ISSN 15249050. doi: 10.1109/TITS.2006.874716.

[117] Marco Wiering. Multi-Agent Reinforcement Learning for Traffic Light Control. In *Proc Intl Conf Machine Learning*, number JANUARY 2000 in ICML '00, pages 1151–1158, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1558607072. doi: 10.1038/ijo.2010.228. URL http://igitur-archive.library.uu.nl/math/2007-0330-200425/wiering{_}00{_}multi.pdf.

[118] Silvia Richter, Douglas Aberdeen, and Jin Yu. Natural actor-critic for road traffic optimisation. In B Schölkopf, J C Platt, and T Hoffman, editors, *Advances in Neural Information Processing Systems*, pages 1169–1176. MIT Press, 2007. ISBN 9780262195683. doi: 10.7551/mitpress/7503.003.0151. URL http://papers.nips.cc/paper/3087-natural-actor-critic-for-road-traffic-optimisation.pdf.

[119] Chen Cai, Chi Kwong Wong, and Benjamin G. Heydecker. Adaptive traffic signal control using approximate dynamic programming. *Transportation Research Part C: Emerging Technologies*, 17(5):456–474, 2009. ISSN 0968090X. doi: 10.1016/j.trc.2009.04.005.

[120] I. Arel, C. Liu, T. Urbanik, and A. G. Kohls. Reinforcement learning-based multi-agent system for network traffic signal control. *IET Intelligent Transport Systems*, 4(2):128–135, 2010. ISSN 1751956X. doi: 10.1049/iet-its.2009.0070. URL http://digital-library.theiet.org/content/journals/10.1049/iet-its.2009.0070.

[121] Tobias Rijken. DeepLight: Deep reinforcement learning for signalised traffic control, 2015.

[122] Samah El-Tantawy, Baher Abdulhai, and Hossam Abdelgawad. Design of reinforcement learning parameters for seamless application of adaptive traffic signal control. *Journal of Intelligent Transportation Systems*, 18(3):227–245, 2014.

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

[124] Juntao Gao, Yulong Shen, Jia Liu, Minoru Ito, and Norio Shiratori. Adaptive traffic signal control: Deep reinforcement learning algorithm with experience replay and target network. *arXiv preprint arXiv:1705.02755*, 2017.

[125] Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. Traffic light control using deep policy-gradient and value-function-based reinforcement learning. *IET Intelligent Transport Systems*, 11(7):417–423, 2017.

[126] Xiaoyuan Liang, Xunsheng Du, Guiling Wang, and Zhu Han. Deep Reinforcement Learning for Traffic Light Control in Vehicular Networks. XX(Xx):1–11, 2018. URL http://arxiv.org/abs/1803.11115.

[127] Wade Genders and Saiedeh Razavi. Using a Deep Reinforcement Learning Agent for Traffic Signal Control. pages 1–9, 2016.

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

[129] Wade Genders and Saiedeh Razavi. Asynchronous n-step Q-learning adaptive traffic signal control. *Journal of Intelligent Transportation Systems: Technology, Planning, and Operations*, 23(4):319–331, 2019. ISSN 15472442. doi: 10.1080/15472450.2018.1491003. URL https://doi.org/10.1080/15472450.2018.1491003.

[130] Wade Genders. *Deep reinforcement learning adaptive traffic signal control.* PhD thesis, 2018.

[131] Tianshu Chu, Jie Wang, Lara Codeca, and Zhaojian Li. Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):1086–1095, 2020. ISSN 15580016. doi: 10.1109/TITS.2019.2901791.

[132] Francisco S Melo. Convergence of Q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep*, pages 1–4, 2001.

[133] Jack Kiefer, Jacob Wolfowitz, and Others. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3): 462–466, 1952.

[134] Seán Mc Loone and George Irwin. Improving neural network training solutions using regularisation. *Neurocomputing*, 37(1-4):71–90, 2001.

[135] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[136] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[137] Melrose Roderick, James MacGlashan, and Stefanie Tellex. Implementing the deep q-network. *arXiv preprint arXiv:1711.07478*, 2017.

[138] Joo Mendes-Moreira, Alípio Mário Jorge, Jorge Freire De Sousa, and Carlos Soares. Comparing state-of-the-art regression methods for long term travel time prediction. *Intelligent Data Analysis*, 16(3):427–449, 2012. ISSN 1088467X. doi: 10.3233/IDA-2012-0532.

[139] G. A. Klunder, A. P. Baas, and F. J.Op De Beek. Long term travel time prediction algorithm using historical data. In *14th World Congress on Intelligent Transport Systems, ITS 2007*, volume 2, pages 1191–1198, 2007. ISBN 9781617387777.

[140] Howard R. Kirby, Susan M. Watson, and Mark S. Dougherty. Should we use neural networks or statistical models for short-term motorway traffic forecasting? *International Journal of Forecasting*, 13(1):43–50, 1997. ISSN 01692070. doi: 10.1016/S0169-2070(96)00699-1.

[141] Ying Lee. Freeway travel time forecast using artifical neural networks with cluster method. In *2009 12th International Conference on Information Fusion, FUSION 2009*, pages 1331–1338. IEEE, 2009. ISBN 9780982443804.

[142] Dongjoo Park, Laurence R. Rilett, and Gunhee Han. Spectral basis neural networks for real-time travel time forecasting. *Journal of Transportation Engineering*, 125(6):515–523, 1999. ISSN 0733947X. doi: 10.1061/(ASCE)0733-947X(1999)125:6(515).

[143] John Rice and Erik Van Zwet. A simple and effective method for predicting travel times on freeways. *IEEE Transactions on Intelligent Transportation Systems*, 5(3):200–207, 2004. ISSN 15249050. doi: 10.1109/TITS.2004.833765.

[144] Steven I.Jy Chien and Chandra Mouly Kuchipudi. Dynamic travel time prediction with real-time and historic data. *Journal of Transportation Engineering*, 129(6):608–616, 2003. ISSN 0733947X. doi: 10.1061/(ASCE)0733-947X(2003)129:6(608).

[145] Yang Zhang and Yuncai Liu. Analysis of peak and non-peak traffic forecasts using combined models. *Journal of Advanced Transportation*, 45(1):21–37, 2011. ISSN 01976729. doi: 10.1002/atr.128.

[146] H. Nicholson and C. D. Swann. The prediction of traffic flow volumes based on spectral analysis. *Transportation Research*, 8(6):533–538, 1974. ISSN 00411647. doi: 10.1016/0041-1647(74)90030-6.

[147] Billy M. Williams, Priya K. Durvasula, and Donald E. Brown. Urban freeway traffic flow prediction: Application of seasonal autoregressive integrated moving average and exponential smoothing models. *Transportation Research Record*, 1644(1644):132–141, 1998. ISSN 03611981. doi: 10.3141/1644-14. URL http://trrjournalonline.trb.org/doi/10.3141/1644-14.

[148] H Sun, H X Liu, H Xiao, and B Ran. Short-term traffic forecasting using the local linear regression model. 2002.

[149] Ming Zhong, Satish Sharma, and Pawan Lingras. Refining genetically designed models for improved traffic prediction on rural roads. *Transportation Planning and Technology*, 28(3):213–236, 2005. ISSN 03081060. doi: 10.1080/03081060500120340.

[150] Nihad Karim Chowdhury, Rudra Pratap Deb Nath, Hyunjo Lee, and Jaewoo Chang. Development of an effective travel time prediction method using modified moving average approach. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5711 LNAI, pages 130–138. Springer, 2009. ISBN 3642045944. doi: 10.1007/978-3-642-04595-0_16.

[151] Pietro Dell'acqua, Francesco Bellotti, Riccardo Berta, and Alessandro De Gloria. Time-Aware Multivariate Nearest Neighbor Regression Methods for Traffic Flow Prediction. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):3393–3402, 2015. ISSN 15249050. doi: 10.1109/TITS.2015.2453116.

[152] S. Vasantha Kumar and Lelitha Vanajakshi. Short-term traffic flow prediction using seasonal ARIMA model with limited input data. *European Transport Research Review*, 7(3), 2015. ISSN 18668887. doi: 10.1007/s12544-015-0170-8.

[153] D. Nikovski, N. Nishiuma, Y. Goto, and H. Kumazawa. Univariate short-term prediction of road travel times. In *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, volume 2005, pages 1074–1079, 2005. ISBN 0780392159. doi: 10.1109/ITSC.2005.1520200.

[154] C. P.I.J. Van Hinsbergen, J. W.C. Van Lint, and F. M. Sanders. Short term traffic prediction models. In *14th World Congress on Intelligent Transport Systems, ITS 2007*, volume 7, pages 5013–5030. ;, 2007. ISBN 9781617387777.

[155] Ibai Lana, Javier Del Ser, Manuel Velez, and Eleni I. Vlahogianni. Road Traffic Forecasting: Recent Advances and New Challenges. *IEEE Intelligent Transportation Systems Magazine*, 10(2):93–109, 2018. ISSN 19411197. doi: 10.1109/MITS.2018.2806634.

[156] William A Gardner, Antonio Napolitano, and Luigi Paura. Cyclostationarity: Half a century of research. *Signal processing*, 86(4):639–697, 2006.

[157] GEORGE E P Box, Gwilym M Jenkins, and G Reinsel. Time series analysis: forecasting and control Holden-day San Francisco. *BoxTime Series Analysis: Forecasting and Control Holden Day1970*, 1970.

[158] R. J. Hyndman. Fitting models to long time series, 2014. URL https://robjhyndman.com/hyndsight/long-time-series/.

[159] R. J. Hyndman. Forecasting with long seasonal periods, 2010. URL https://robjhyndman.com/hyndsight/longseasonality/.

[160] R Core Team and Others. R: A language and environment for statistical computing. 2013.

[161] Rob J Hyndman, Yeasmin Khandakar, and Others. Automatic time series forecasting: the forecast package for R. *Journal of statistical software*, 27(3): 1–22, 2008.

[162] R. J. Hyndman. Two seasonal periods in ARIMA using R, 2013. URL https://stats.stackexchange.com/questions/47729/two-seasonal-periods-in-arima-using-r.

[163] Ivan Svetunkov. Statistical models underlying functions of'smooth'package for R. 2017.

[164] Ivan Svetunkov. Forecasting Using State Space Models, 2021. URL https://cran.r-project.org/web/packages/smooth/smooth.pdf.

[165] Ivan Svetunkov and John E Boylan. State-space ARIMA for supply-chain forecasting. *International Journal of Production Research*, 58(3):818–827, 2020.

[166] J W C Van Lint and Henk J van Zuylen. Monitoring and predicting freeway travel time reliability: Using width and skew of day-to-day travel time distribution. *Transportation Research Record*, 1917(1):54–62, 2005.

[167] Zhenliang Ma, Luis Ferreira, Mahmoud Mesbah, and Sicong Zhu. Modeling distributions of travel time variability for bus operations. *Journal of Advanced Transportation*, 50(1):6–24, 2016.

[168] Tatiana Benaglia, Didier Chauveau, David Hunter, and Derek Young. mixtools: An R package for analyzing finite mixture models. *Journal of Statistical Software*, 32(6):1–29, 2009.

[169] Sheng Li. Nonlinear combination of travel-time prediction model based on wavelet network. In *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, volume 2002-Janua, pages 741–746. IEEE, 2002. ISBN 0780373898. doi: 10.1109/ITSC.2002.1041311.

[170] A. Samant and H. Adeli. Feature Extraction for Traffic Incident Detection Using Wavelet Transform and Linear Discriminant Analysis. *Computer-Aided Civil and Infrastructure Engineering*, 15(4):241–250, 2000. ISSN 10939687. doi: 10.1111/0885-9507.00188.

[171] Samanwoy Ghosh-Dastidar and Hojjat Adeli. Wavelet-clustering-neural network model for freeway incident detection. *Computer-Aided Civil and Infrastructure Engineering*, 18(5):325–338, 2003. ISSN 10939687. doi: 10.1111/1467-8667.t01-1-00311.

[172] Xiaomo Jiang and Hojjat Adeli. Dynamic wavelet neural network model for traffic flow forecasting. *Journal of Transportation Engineering*, 131(10): 771–779, 2005. ISSN 0733947X. doi: 10.1061/(ASCE)0733-947X(2005)131:10(771).

[173] Abhijit Dharia and Hojjat Adeli. Neural network model for rapid forecasting of freeway link travel time. *Engineering Applications of Artificial Intelligence*, 16(7-8):607–613, 2003. ISSN 09521976. doi: 10.1016/j.engappai.2003.09.011.

[174] Tigran T. Tchrakian, Biswajit Basu, and Margaret O'Mahony. Real-time traffic flow forecasting using spectral analysis. *IEEE Transactions on Intelligent Transportation Systems*, 13(2):519–526, 2012. ISSN 15249050. doi: 10.1109/TITS.2011.2174634.

[175] Hang Yang, Yajie Zou, Zhongyu Wang, and Bing Wu. A hybrid method for short-term freeway travel time prediction based on wavelet neural network and markov chain. *Canadian Journal of Civil Engineering*, 45(2):77–86, 2018. ISSN 12086029. doi: 10.1139/cjce-2017-0231.

[176] D. H. Griffel and Ingrid Daubechies. Ten Lectures on Wavelets. In *The Mathematical Gazette*, volume 79, page 224, 1995. doi: 10.2307/3620105.

[177] Stephane Mallat. *A Wavelet Tour of Signal Processing*. Elsevier, 2009. ISBN 9780123743701. doi: 10.1016/B978-0-12-374370-1.X0001-8.

[178] Sofia C. Olhede and Andrew T. Walden. Generalized Morse wavelets. *IEEE Transactions on Signal Processing*, 50(11):2661–2670, 2002. ISSN 1053587X. doi: 10.1109/TSP.2002.804066.

[179] INRIX. Scorecard, 2019. URL http://inrix.com/scorecard/.

[180] S A C S Jayasooriya and Y M M S Bandara. Measuring the Economic costs of traffic congestion. *3rd International Moratuwa Engineering Research Conference, MERCon 2017*, (February):141–146, 2017. doi: 10.1109/MERCon.2017.7980471.

[181] M C Bell and R D Bretherton. Ageing of fixed-time traffic signal plans. In *International conference on road traffic control*, 1986.

[182] Tichakorn Wongpiromsarn, Tawit Uthaicharoenpong, Yu Wang, Emilio Frazzoli, and Danwei Wang. Distributed traffic signal control for maximum network throughput. In *2012 15th international IEEE conference on intelligent transportation systems*, pages 588–595. IEEE, 2012.

[183] Tung Le, Péter Kovács, Neil Walton, Hai L. Vu, Lachlan L.H. Andrew, and Serge S.P. Hoogendoorn. Decentralized signal control for urban road networks. *Transportation Research Part C: Emerging Technologies*, 58:431–450, 2015. ISSN 0968090X. doi: 10.1016/j.trc.2014.11.009. URL http://dx.doi.org/10.1016/j.trc.2014.11.009.

[184] Hsu-Chieh Hu and Stephen F Smith. Softpressure: a schedule-driven backpressure algorithm for coping with network congestion. *arXiv preprint arXiv:1903.02589*, 2019.

[185] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[186] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, and Others. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[187] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[188] Patrick Mannion, Jim Duggan, and Enda Howley. An experimental review of reinforcement learning algorithms for adaptive traffic signal control. In *Autonomic road transport support systems*, pages 47–66. Springer, 2016.

[189] Ying Liu, Lei Liu, and Wei Peng Chen. Intelligent traffic light control using distributed multi-agent Q learning. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2018-March:1–8, 2018. ISSN 2153-0009. doi: 10.1109/ITSC.2017.8317730. URL http://arxiv.org/abs/1711.10941.

[190] Chia-Hao Wan and Ming-Chorng Hwang. Value-based deep reinforcement learning for adaptive isolated intersection signal control. *IET Intelligent Transport Systems*, 12(9):1005–1010, 2018.

[191] Aleksandar Stevanovic and Peter T Martin. Split-cycle offset optimization technique and coordinated actuated traffic control evaluated through microsimulation. *Transportation Research Record*, 2080(1):48–56, 2008.

[192] PTV Group. PTV Vissim 20, 2019. URL https://www.ptvgroup.com/en/solutions/products/ptv-vissim/.

[193] Richard E. Allsop. Delay-minimizing settings for fixed-time traffic signals at a single road junction. *IMA Journal of Applied Mathematics (Institute of Mathematics and Its Applications)*, 8(2):164–185, 1971. ISSN 02724960. doi: 10.1093/imamat/8.2.164.

[194] Benjamin G Heydecker. Objectives, stimulus and feedback in signal control of road traffic. *Journal of Intelligent Transportation Systems*, 8(2):63–76, 2004.

[195] R Wiedemann. Simulation Des strassenverkehrsflusses. Institut für Verkehrswesen der Universität Karlsruhe. 1974.

[196] Richard E Allsop. Estimating the traffic capacity of a signalized road junction. *Transportation Research/UK/*, 6(3), 1972.

[197] Xiao Feng Xie, Stephen F. Smith, Liang Lu, and Gregory J. Barlow. Schedule-driven intersection control. *Transportation Research Part C: Emerging Technologies*, 24:168–189, 2012. ISSN 0968090X. doi: 10.1016/j.trc.2012.03.004.

[198] R J Salter. Optimum cycle times for an intersection. In *Highway Traffic Analysis and Design*, pages 304–310. Springer, 1996.

[199] Hua Wei, Chacha Chen, Guanjie Zheng, Kan Wu, Vikash Gayah, Kai Xu, and Zhenhui 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*, pages 1290–1298, 2019.

[200] Baher Abdulhai, Rob Pringle, and Grigoris J Karakoulas. Reinforcement learning for true adaptive traffic signal control. *Journal of Transportation Engineering*, 129(3):278–285, 2003.

[201] Samah El-Tantawy and Baher Abdulhai. An agent-based learning towards decentralized and coordinated traffic signal control. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, pages 665–670, 2010. ISSN 2153-0009. doi: 10.1109/ITSC.2010.5625066.

[202] L A Prashanth and Shalabh Bhatnagar. Reinforcement learning with function approximation for traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 12(2):412–421, 2010.

[203] Sahar Araghi, Abbas Khosravi, Michael Johnstone, and Doug Creighton. Intelligent Traffic Light Control of Isolated Intersections Using Machine Learning Methods. *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 3621–3626, 2013. doi: 10.1109/SMC.2013.617. URL http://ieeexplore.ieee.org/document/6722370/.

[204] Pengyuan Zhou, Tristan Braud, Ahmad Alhilal, Pan Hui, and Jussi Kangasharju. ERL: Edge Based Reinforcement Learning for Optimized Urban Traffic Light Control. *2019 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2019*, pages 849–854, 2019. doi: 10.1109/PERCOMW.2019.8730706.

[205] Mohammad Aslani, Mohammad Saadi Mesgari, Stefan Seipel, and Marco Wiering. Developing adaptive traffic signal control by actor–critic and direct exploration methods. In *Proceedings of the Institution of Civil Engineers-Transport*, volume 172, pages 289–298. Thomas Telford Ltd, 2019.

[206] P Yau, K. L. A., Qadir, J., Khoo, H. L., Ling, M. H., & Komisarczuk. A Survey on Reinforcement Learning Models and Algorithms for Traffic Signal Control. *ACM Computing Surveys*, 50(3), 2017. doi: https://dl.acm.org/doi/pdf/10.1145/3068287.

[207] Hua Wei, Guanjie Zheng, Vikash Gayah, and Zhenhui Li. A survey on traffic signal control methods. *arXiv preprint arXiv:1904.08117*, 2019.

[208] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, and Others. Pytorch: An imperative style, high-performance deep

learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[209] UK Department for Transport. *General Principles of Traffic Control by Light Signals*. Department for Transport London, UK, 2006.

[210] Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui 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, pages 2496–2505, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3220096. URL https://doi.org/10.1145/3219819.3220096.

[211] Pravin Varaiya. *The Max-Pressure Controller for Arbitrary Networks of Signalized Intersections*, pages 27–66. Springer New York, New York, NY, 2013. ISBN 978-1-4614-6243-9. doi: 10.1007/978-1-4614-6243-9_2. URL https://doi.org/10.1007/978-1-4614-6243-9{_}2.

[212] Chacha Chen, Hua Wei, Nan Xu, Guanjie Zheng, Ming Yang, Yuanhao Xiong, Kai Xu, and Zhenhui Li. Toward A Thousand Lights: Decentralized Deep Reinforcement Learning for Large-Scale Traffic Signal Control. *AAAI*, pages 3414–3121, 2020. URL www.aaai.org.

[213] Richard E. Allsop. SIGSET: A computer program for calculating traffic signal settings. *Traffic Engineering & Control& Control*, 1971.

[214] Highways Agency. Siting of Inductive Loops for Vehicle Detecting Equipments at Permanent Road Traffic Signal Installations. (C), 2002.

[215] Monireh Abdoos, Nasser Mozayani, and Ana L C Bazzan. Traffic light control in non-stationary environments based on multi agent Q-learning. In *2011 14th International IEEE conference on intelligent transportation systems (ITSC)*, pages 1580–1585. IEEE, 2011.

[216] Ammar Haydari and Yasin Yilmaz. Deep Reinforcement Learning for Intelligent Transportation Systems: A Survey. *arXiv preprint arXiv:2005.00935*, 2020.

[217] Xiaoyuan Liang, Xunsheng Du, Student Member, and Guiling Wang. A Deep Reinforcement Learning Network for Traffic Light Cycle Control. *IEEE Transactions on Vehicular Technology*, 68(2):1243–1253, 2019. doi: 10.1109/TVT.2018.2890726.