Center for Advanced Multimodal Mobility Solutions and Education

Project ID: 2022 Project 07

DEVELOPING ROBUST SMART TRAFFIC SIGNAL CONTROL

Final Report

by

Tianxin Li, Ph.D. (ORCID ID: https://orcid.org/0000-0002-3061-8077)
Graduate Research Assistant
The University of Texas at Austin
301 E. Dean Keeton Street, Stop C1761, Austin, TX 78712
Phone: 1-512-471-4541; Email: tianxinli@utexas.edu

Randy Machemehl, Ph.D., P.E. (ORCID ID: https://orcid.org/0000-0002-6314-2626)
Professor
The University of Texas at Austin
301 E. Dean Keeton Street, Stop C1761, Austin, TX 78712
Phone: 1-512-471-4541; Email: rbm@mail.utexas.edu

for

Center for Advanced Multimodal Mobility Solutions and Education
(CAMMSE @ UNC Charlotte)
The University of North Carolina at Charlotte
9201 University City Blvd
Charlotte, NC 28223

September 2023

ACKNOWLEDGEMENTS

This project was funded by the Center for Advanced Multimodal Mobility Solutions and Education (CAMMSE @ UNC Charlotte), one of the Tier I University Transportation Centers that were selected in this nationwide competition, by the Office of the Assistant Secretary for Research and Technology (OST-R), U.S. Department of Transportation (US DOT), under the FAST Act. The authors are also very grateful for all of the time and effort spent by DOT and industry professionals to provide project information that was critical for the successful completion of this study.

DISCLAIMER

Table of Contents

List of Figures

List of Tables

EXECUTIVE SUMMARY

Traffic signal control is a crucial element of urban mobility that profoundly influences transportation network efficiency and safety. Traditional traffic signal control systems rely on fixed-time or actuated signal timings, often failing to adapt to dynamic traffic demands and congestion patterns. This technical report explores the application of Reinforcement Learning (RL) algorithms to traffic signal control, aiming to enhance traffic flow efficiency and alleviate congestion.

The research develops a simulation model of a signalized intersection and trains RL agents to dynamically adjust signal timings based on real-time traffic conditions. These RL agents are designed to learn from experience, adapt to changing traffic patterns, and optimize traffic flow, even in scenarios with unexpected traffic incidents.

The study examines the benefits of RL algorithms in optimizing traffic signal control, both in scenarios with and without traffic incidents. To achieve this, an incident generation module is integrated into an open-source traffic signal performance simulation framework that relies on the Simulation of Urban MObility SUMO software. This module introduces the presence of emergency response vehicles and randomly generates traffic incidents within the network. By exposing RL agents to this environment, they can learn and fine-tune traffic signal control to minimize system delay.

Initially, the research focuses on a single intersection scenario, employing the DQN algorithm to form the RL agent traffic signal controller. The training process is enhanced through the utilization of experience replay and target network techniques, addressing the limitations of the DQN algorithm. Hyperparameter tuning identifies the optimal parameter combination for training, with results showcasing the superiority of DQN over other controllers in terms of system-wide and intersection-specific queue distribution and vehicle delay.

The study is subsequently extended to encompass a small corridor featuring two intersections and a grid network with a 2x2 intersection configuration. The incident generation module introduces various traffic scenarios to the RL agent, and once again, hyperparameter tuning confirms the DQN model's effectiveness in reducing congestion and enhancing system performance. Robustness testing under varying demands demonstrates the consistent performance of the DQN model.

In summary, this technical report underscores the potential of RL algorithms in optimizing traffic signal control, both in scenarios with and without traffic incidents. The incident generation module creates a realistic learning environment for RL agents, resulting in improved system performance and reduced congestion. Furthermore, the importance of hyperparameter tuning is emphasized as a critical component in establishing a strong foundation for RL training processes.

Chapter 1. Introduction

Traffic signal systems play an essential role in the transportation network to minimize the number of traffic accidents and maintain orderly traffic flow. Traffic signal control methods include three broad categories: pretimed control, actuated control, and adaptive control. Pretimed control has fixed cycle lengths and phasing, so it is not responsive to traffic demand fluctuations. Actuated control is designed to respond to variable traffic demands but the nature of the potential response is constrained by the combination of detection capability and fixed controller settings. Adaptive control could lead to better performance if the proposed methods can accurately predict future traffic patterns.

With the explosive development of computing power and data accessibility, as well as the advanced development of artificial intelligent (AI), there are more possibilities to improve existing traffic signal control performance (Winston, 1992; Russell and Norvig, 2002). Three categories of AI technologies have been used commonly: supervised learning, unsupervised learning, and reinforcement learning. Because of the characteristics of the traffic signal control problems, reinforcement learning fits our needs best (Arel, 2010). Reinforcement learning is designed to handle optimization problems by learning the interaction between agents and the environment. The core concept behind reinforcement learning is to take advantage of the machine computing capability to discover the relationships between the traffic signal control, the intersection environment and the traffic flow patterns by trial and error. Due to the complexity of the problem, it would be impractical for human beings to complete such complicated calculations in a short period of time.

Deep Q-learning is one of the most commonly used methods in reinforcement learning because of its ease of implementation and better performance as the data scale increases. Deep Q-leaning is a combination of Q-learning and deep neural network. Q-learning represents a method to use a determined or approximated Q-table to guide the actions of the agents. With the training process, the Q-table is updated and reaches convergence so that every action taken in the future will be the best choice of the agent in order to maximize the long-term cumulative rewards.

Combined with deep learning, reinforcement learning can explore more complicated relationships between the agents and their environment to provide potentially better performance (LeCun et al., 2015; Goodfellow et al., 2016; Kamilaris and Prenafeta-Boldú, 2018). Deep learning relies on a neural network, which mimics the thinking and decision-making process of neuron activation (Wang, 2003; Abiodun et al., 2018). The more layers of the neural network used, the more complicated patterns between the inputs and outputs can be detected. However, more layers of neural network also require more computing power during the calculation. One must consider the tradeoff between the calculation effort and representation of the relationship between the inputs and outputs.

For a complicated problem, a deterministic Q-table is impossible to generate so a deep neural network is utilized to approximate the Q-table. Rather than having a concrete Q-table, deep Q-learning uses the neural network between the inputs and outputs to approximately

represent the Q-table. Therefore, the training process will update the coefficients associated with the neural network to improve the prediction accuracy.

Although considerable research has focused on using deep Q-learning to improve the performance of actuated traffic signals in a network, one key question has not been targeted yet. That is how disruptions within the network should be addressed. The concern is that traffic signal control based on deep Q-learning and normal traffic condition settings might not be able to adequately respond to traffic flow disruptions caused by traffic incidents (crashes, disabled vehicles or objects dropped on the roadway). This dissertation is developed to fill this gap, developing a robust traffic signal control equipped with the deep Q-learning while also considering traffic incidents in the network.

1.1 Problem Statement

Without testing the performance when traffic incidents occur, the robustness of the deep Q-learning traffic signal control for the network cannot be guaranteed. When traffic incidents occur, the network will suffer a sharp and temporary capacity shortage on the involved link(s) causing diversion to other links. Traffic signal timing is the only practical means of responding to incident disruptions to reduce the negative impact of traffic incidents. Due to the complexity of the network problems, operators from the traffic management center often cannot provide an optimal traffic signal plan in a short period of time. Traffic incidents can occur in any location and any situation in the network, making the previous experience less useful.

The core problem of the traffic incident case is sudden unmatched travel demand and supply. If traffic signals can utilize the real-time objective inputs from their environment, including traffic flows and intersection performance (e.g., queue length and total control delays), to take actions quickly enough, the network performance could be improved immediately.

Reinforcement learning methods promise to provide solutions for this kind of problem. Each AI agent keeps exploring the relationship between traffic signal control and vehicle queue length in its intersection and once the knowledge collected is enough to produce an accurate approximation, the action chosen by the AI agent (maintain the current phase or shift to another phase) will maximize the improvement of the intersection performance in terms of the chosen measure of effectiveness such as queue length or delay reduction.

If the agent has never experienced traffic incident impacts, it must encounter the situation enough times to “understand” the impact of incidents and how to take optimal actions responsively. Creating traffic incidents in the real network to train the deep Q-learning algorithm is problematic so the simulation method comes in handy. There are no available simulation tools on the market to allow users to combine the application of reinforcement learning and traffic network incidents.

This study is to fill this gap. By developing an incident responsive network in an open-source microscopic simulator and exploring the advanced deep Q-learning method, a robust AI-assisted actuated traffic signal control system will be developed.

1.2 Objectives

The objectives of this dissertation include the following key components:

1. Build a traffic incident responsive simulator based on an open-source microscopic traffic simulation software system. This simulator will characterize an incident occurring in the network and blocking a lane that is part of a link. In addition, the simulator will simulate the impact of emergency service vehicles (an abstraction of police cars, EMS, etc.) in response to the incident. In this way, the full impact of the incident and the rescue process can be evaluated quantitatively based on traditional measures of effectiveness, such as queue lengths and total system delay.

2. A deep Q-learning model will be developed and trained with the data from the simulation process. The proposed deep Q-learning method will take advantage of the most advanced methods in the market, including the prioritized experience replay and dueling network. The deep Q-learning model will be trained in a single intersection without traffic incidents.

3. The well-trained deep Q-learning model will be applied to all the traffic signals of a grid network where all intersections are identical to the single intersection in the previous study and the network will encounter traffic incidents occurring randomly in time and location. Transfer learning methods will be applied to reduce the calculation tasks to allow the deep Q-learning model to perform well in a different environment.

1.3 Expected Contributions

To achieve these goals, this study explores the application of RL, specifically Q-learning integrated with deep neural networks, to enhance traffic signal control. It explores RL's capacity to enhance traffic flow and alleviate congestion, effectively addressing the shortcomings of conventional fixed-time and actuated signal systems.

1.4 Report Overview

This dissertation is organized as follows. Chapter 1 describes the motivation, problem statement, objectives, and research scope. Chapter 2 is a comprehensive literature review of the history of traffic signal control, the common framework of traffic signal control, and the most advanced research on traffic signal control based on reinforcement learning methods. Chapter 3 presents the proposed deep Q-learning model as well as advanced tools to improve its performance. Chapter 4 explains the open-source micro simulation software, the traffic incident analysis module, network choice, demand generation, and details from the incident module. Chapter 5 describes the model performance for a single intersection without traffic incident disruptions. Chapter 6 describes the transfer of the proposed deep Q-learning model to a network with adjustments to enable evaluation of traffic incident handling. Chapter 7 concludes the dissertation and suggests future work.

Chapter 2. Literature Review

2.1 Introduction

An intersection is where vehicle paths cross sharing a common space. In earlier days, there were no traffic control devices to facilitate the common space sharing, so users had to compete for the right of way. To improve safety and facilitate orderly space sharing, traffic control devices were introduced. Traffic control signals are commonly used by agencies to improve intersection safety and operational efficiency.

Generally, signalized intersections accommodate all ground transportation modes, including passenger cars, bicycles, and pedestrians. However, the purpose of this paper is building fundamental reinforcement learning traffic signal control based on simulation methods so only passenger cars are considered throughout the paper.

The following section summarizes important literature for the development of traffic signal control methods, including pretimed, actuated, adaptive, and machine learning control.

2.2 Traditional Traffic Signal Control Methods

The basic logic behind traffic signal timing is to provide optimal amounts of green signal time to conflicting movements to reduce conflicts and decrease the likelihood of traffic accidents and to improve efficiency usually measured by fewer delays.

There are three types of traffic signal control methods commonly used today: pretimed, actuated, and adaptive. None of them is superior to the others since they perform different roles for different types of intersections as well as traffic arrival patterns. Therefore, all of them can have a significant impact on the traffic network in terms of safety and efficiency.

2.2.1 Pretimed Signal Control

Pretimed traffic signal control is defined as a predetermined traffic signal schema with fixed green time for each phase as well as fixed cycle length and fixed phase patterns. The signal cycle length needs to be tuned to minimize the control delay and the green time split for each approach is normally based on the flow ratios between different phases (Kell and Fullerton, 1991).

Pretimed traffic signal control methods are commonly used for both isolated intersections and networks (Bell, 1992; Slinn et al., 1998). Webster proposed a closed-form formula to split the green time proportionally by taking into account the historical traffic flow ratios between phases (Webster, 1958). The cycle length is tuned based on the characteristics of the intersection to minimize the total delay. No real-time data from the field is required and the historical traffic flow needs to be aggregated.

Coordination of pretimed controllers to produce traffic progression can improve network efficiency decreasing unnecessary stops and reducing delays. The GreenWave was developed (Roess et al., 2004) as an extension of the Webster methods by considering the travel time at a chosen speed between intersections (called offset) to reduce numbers of vehicular stops. This

method requires all associated intersections to have the same cycle length, which is usually the maximum cycle length from all intersections.

Practitioners have developed different types of extensions of pretimed traffic signal control. For example, intersections could have different pretimed traffic signal schemes during different times of the day and different schemes for weekdays and weekends (Mirchandani and Head, 2001). Instead of having only one signal plan for a specific intersection, as many as 20 different plans could be applied and could be automatically chosen by the signal controller based on either time of day or traffic demand (Roess, 2004).

Pretimed signal control is an offline method which means there is no need to collect any real-time information from the field. It relies on historical traffic data to adjust the green time split, cycle length, and phase patterns. It is easy to maintain compared to other more advanced traffic signal control methods which require field data, including flow and queue length to tune their parameters. Therefore, pretimed signal control is still the most commonly applied method in the traffic network.

2.2.2 Actuated Signal Control

Since traffic demand constantly varies, the basic objective of signal control is to accommodate demand variability. Pretimed methods can address this variability by choosing among many stored timing plans by time of day or volume thresholds (if detection is provided). Actuated signal control measures real time traffic flows for all actuated phases and is designed to be flexible enough to change green times in response to demand (Fellendorf, 1994). Every actuated phase has a maximum green time so if demand is heavy on all phases, every phase will be the maximum green and the actuated scheme evolves into pretimed operation. Actuated control does not work well in coordinated signal systems, since time-based coordination requires every signal to have the same cycle length, that is, to provide time based coordination in a network, actuated signals must “act like” pretimed signals (Yin, 2007).

Actuated signal control collects real time data from the intersection approaches, such as queue length or traffic flow, to extend the current phase duration or terminate the current phase to start the next phase as needed. In actuated traffic signal control, several key components could be varied, including phase sequences, green time for each phase, and cycle length, which does not coordinate with other adjacent intersections (Roess, 2004).

The benefits of implementing actuated signal control are obvious. It can adjust the current plan to the varied traffic conditions, such as flow fluctuation or changing traffic demand patterns, to minimize control delay and improve efficiency. It is recommended to use actuated traffic control in a non-oversaturated traffic flow scenario. Because if the traffic flow is approaching the capacity of the road and stable, especially during the peak hours, pretimed signal control programmed proportionally to the critical flow will be equivalent but less expensive and require less maintenance.

Semi-actuated control refers to actuated control with detectors only on the minor road so the green rests on the major road until a vehicle is detected on the minor road. In this way, the associated intersection can maintain the green time for the major road and also provide service to

the minor street when needed. This method is appropriate for intersections where the traffic pattern has a noticeable difference in volume from the major and minor roads. If the actuated green phase for the minor roads is called too many times, the vehicles from the major roads suffer significant delay and more stops, which is against the purpose of implementing the semi-actuated control.

Fully actuated control includes detectors for all signal phases and allows real time adjustment of the signal plan to accommodate traffic for all intersection approaches rather than only the minor roads. By taking into account real time traffic flow from detectors, the signal plan can extend or terminate any phase as needed. This helps the intersection to respond to varied traffic flow from all approaches (Lin, 1985).

Researchers have investigated methods of using actuated control in coordinated networks, however, as noted earlier, coordination methods generally require actuated control to limit the flexibility they are designed to provide. Two famous fully actuated signal control methods are Self-Organizing Traffic Light (SOTL) and max pressure. Self-organization used here represents the concept of signal control for intersections in a network that can interact with each adjacent one and achieve dynamically a global optimum (Gershenson, 2005). Cools et al. (2013) proposed an on-demand responsive actuated signal control with varying traffic demand and overcame the traditional green wave method of allocating unnecessary green time or deficient green time for the incoming vehicles. It is a method that helps traffic flow move as platoons by forcing the vehicles to wait at the stop line until a queue size threshold is met. Once the approaches accumulate queues surpassing the predefined threshold, the green signal shifts to move those vehicles potentially more efficiently than the existing green wave method.

Max pressure control was introduced into actuated signal control (Varaiya, 2013). This method monitors the pressure from all approaches, as the difference between flow for incoming lanes and outgoing lanes of each approach, and chooses the maximum pressure releasing phase to allow the maximum number of vehicles to enter the intersection and hence ensure the minimum pressure for the phase duration. This method requires vehicle flow information from adjacent intersections as a precise measure of the pressure. Weighted queue lengths are needed to calibrate intersections to achieve the best performance.

Actuated control could be categorized into two broad classes, including isolated and coordinated. Isolated intersections with actuated control only focus on improving efficiency and safety of one intersection, while coordinated intersections will deploy a reasonable offset and other parameters to reduce the unnecessary stops and delay for the coordinated network.

Semi-actuated can be used for an arterial corridor since the major traffic flow would use the arterial street and minor cross streets would be served with green only when needed (Skabardonis, 1998). The majority of the green time and capacity should be assigned to the corridor rather than the minor movements. Fully actuated control would be most beneficial for isolated intersections where traffic demands from all approaches vary heavily.

Implementing actuated signal control to adjust the control plan in real-time has limitations. A complicated program must be provided for the controllers to take the inputs and adjust the control plan accordingly. The cost of installation is more expensive than pretimed

signal control and maintenance costs are another problem. The induction loop detectors commonly used in these methods are installed under the pavement surface and if the pavement structure moves either vertically or horizontally the inductance loop detector wires break and the cost of replacing them is not trivial.

2.2.3 Adaptive Signal Control

Adaptive signal control refers to the technology that collects real-time data from the installed detectors to dynamically determine green phase and its duration in response to current and predicted future traffic demands based on programmed algorithms to increase the performance of the intersections.

Adaptive signal control is considered to be advantageous over actuated signal control for providing lower control delay and better intersection throughput performance (Gayah, 2014). The key component of adaptive signal control is to dynamically adjust the parameters based on the future traffic flow prediction (Klein, 2001). The most famous traffic signal control frameworks based on adaptive signal control include TRANSYT, Split Cycle Offset Optimization Technique (SCOOT), Sydney Coordinated Adaptive Traffic System (SCATS), Optimized Policies for Adaptive Control (OPAC), Real-Time Hierarchical Optimized Distributed and Effective System (RHODES), and ASC Lite. The following content will review all of these frameworks.

In 1969, Robertson proposed a fixed-time traffic signal control algorithm based on the traffic flow passing through a road network to minimize the sum of the average queues in the network (Robertson, 1969). It is an off-line method that uses macro-simulation since it relies on historic flow data. It was one of the earliest traffic signal control methods that relied on a digital computer program to help researchers and practitioners to optimize the traffic signal control, including offset and green time split. It can be used to control up to 50 intersections in a network.

The core components in TRANSYT are based on cyclic flow profiles (CFP) that estimate queue lengths based on historic data so to evaluate the performance of alternative signal timings. The CFP measures the one-way traffic flow from one approach and averages the flow over a specific duration. The estimated queue length and clearing time from the CFP are used to predict the impact of offset and green splits to find the best signal timing parameters.

Based on the TRANSYT, SCOOT was introduced to overcome some of the limitations of TRANSYT (Hunt, 1981). As mentioned before, TRANSYT is an offline method and relies on historic data. In contrast, SCOOT takes advantage of technology development as vehicle sensors have become available. Detectors are installed upstream to obtain traffic flow information so to improve the estimation of queue length accuracy. In addition, since SCOOT relies on real-time traffic information and calculates the signal timing parameters quickly, it is an online method. Lastly, SCOOT adjusts the signal timing parameters gradually to avoid significant timing fluctuations that could disrupt flow (Hunt, 1981). Transport and Road Research Laboratory (TRRL) tested the performance of SCOOT in England cities in 1975 by floating cars and found that SCOOT reduced average queue length by 12% (Robertson, 1986).

SCATS was introduced in Sydney, Australia in 1990. It utilizes the traffic flow inputs collected by installed detectors to understand the real-time traffic. It also has a library that records the pre-defined signal plans based on the traffic flow patterns to help dynamically adjust the signal timing parameters in a short period. The adjustable parameters include phase split, cycle length, and offsets (Lowrie, 1990). SCATS has been implemented in Australia for controlling more than 1,800 signals and has achieved significant improvement in terms of reducing delay and queue length. In addition, SCATS has been implemented in multiple other areas and has achieved promising performance improvement (Akcelik et al., 1998; Stevanovic, 2009; Dutta et al., 2010).

In the early 80s, researchers at the University of Lowell with support from the U.S. Department of Transportation developed OPAC, varying signal timing plans dynamically to accommodate real-time traffic demand patterns. It ignores the cycle concept and only considers the split time of sequential phases by either extending the current phase or starting the next phase earlier (Gartner, 2002). Implementation of this method requires predetermining the phases for each intersection. To improve performance, OPAC considers both real-time data collected from upstream detectors and historic flow data for better queue and delay estimation. Dynamic programming and rolling horizon optimization are used to find the optimal solutions for the signal timing parameters in response to traffic patterns. OPAC can be implemented for distributed individual intersections to achieve network-level optimization (Gartner, 2001).

RHODES is another famous adaptive signal control framework that can be implemented for a distributed system. RHODES utilizes an hierarchical control structure for connecting different components in traffic signal optimizing problems, including network loading, network flow estimation, and traffic signal control activation by exploiting the modern technologies and availability of real-time data (Head, 1992). RHODES not only considers the software for traffic signal control, but also the hardware components, such as the communication system. RHODES applies a dynamic programming method to optimize the single intersection signal timing plan for splits and cycle length while implementing the REAMBAND model for platoon progression optimization (Dell, 1995).

To reduce the costs of installation and operation while keeping the benefits of traditional adaptive signal control frameworks, the Federal Highway Administration (FHWA) (Ghaman, 2007) developed ASC Lite to integrate the process of traffic flow monitoring and signal plan optimizing accordingly. ASC Lite focuses on linear and arterial networks. The developed control module has been included in the CORSIM simulation software for users to deploy and test their signal timing strategies.

Performance of ASC Lite has been evaluated (Shelby, 2008) by field implementation, including Gahanna, OH, Houston, TX, Bradenton, FL, and EI Cajon, CA. The evaluation shows that ASC Lite has been demonstrated effective in terms of reduction of delay, arterial travel time, fuel consumption, and vehicle stops. In addition, ASC Lite was also evaluated by field implementation in Albany, New York, showing that the system provided benefits of delay reduction in the core area of the analytical network, but not the boundary (Ban, 2014).

In conclusion, adaptive signal control has attracted a large number of researchers and practitioners to develop various frameworks and test their performance in real scenarios.

Evolution of these strategies is mainly due to modern technologies and algorithms, including faster computing machines and more efficient mathematical algorithms. With the rapid development of learning algorithms and lower costs of data storage and computing, reinforcement learning has been adopted to improve signal control performance.

2.3 Reinforcement Learning Traffic Signal Control

Machine learning tackles the problems that relate to detecting patterns and drawing conclusions from historic experience. Reinforcement learning, one of the most famous machine learning techniques, focuses on optimization problems by directly converting input data into action choices without modeling the environment. For example, in the traffic signal control process, the adaptive control methods require the prediction of the queue length or vehicle arrival patterns from the adjacent network through mathematical models. In a complex system, particularly in the transportation network, to fully understand the relationship between vehicle arrival patterns and traffic signal phase durations is difficult, sometimes even impossible. However, in reinforcement learning, models of this kind of detail are not required and hence attract many researchers to explore its capability in the signal control domain (Abdulhai, 2003).

Reinforcement learning collects experience from the interaction between an agent and its environment. Without building a model for the environment, the agent could extract useful information from the environment and use trial and error to come up with a solution to improve its behavior to achieve a long-term goal (Sutton and Barto, 2018).

Reinforcement learning includes model-free and model-based algorithms. In our scenario, where traffic signal control responds to varying traffic demand, model-based algorithms will require modelers to pre-specify the models for the intersection as well as the vehicle arrival patterns, which is difficult. Therefore, model-free approaches achieve a significant focus in the traffic signal control field, especially Q-learning. The Q-learning agent, the signal controller, collects the state from its interacting environment to adjust its behavior to improve its performance measured by a performance index. It has many advantages facilitating the improvement of traffic signal control. Watkins proved that Q-learning would converge to optimal action-values with the probability of 1 as long as all state and action pairs are repeated in the samples (Watkins, 1992). More details for Q-learning will be introduced in Chapter 3. Here we focus on the applications of reinforcement learning to traffic signal control.

2.3.1 Isolated Intersections

Abdulhai (2003) proposed a simple yet powerful Q-learning model for traffic signal control associated with an isolated intersection. The traffic demand contains two straight movements, including east-west and north-south. The state includes the queue lengths from all approaches as well as the elapsed green time of the current phase. The traffic control agent can choose two actions, either remain in the current phase or shift to the next one. Cycle length was not fixed but minimum and maximum green splits are chosen for each phase for practical reasons. The reward function was measured by a power function of the queue length from all approaches to discourage longer queues for some approaches. The proposed Q-learning traffic signal controller was compared with the pretimed signal control through simulations with three different traffic demand profiles to reflect traffic pattern variation including off-peak and peak hours. When traffic demand is constant and near the intersection capacity, the Q-learning

controller performed on a par with the pretimed signal control, however, when the traffic becomes variable, the Q-learning controller reduced system delay by 40% on average. This research laid the foundation for implementing Q-learning in the traffic signal control field.

El-Tantawy et al. extended Abdulhai's work by fine-tuning the parameters used in the Q-learning traffic signal control model with a real case study in Downtown Toronto in a simulation environment (El-Tantawy et al., 2014). The proposed model outperformed optimized pretimed traffic signal control and actuated signal control by saving about 50% average vehicle delay.

One limitation of the Q-learning signal controller mentioned above is that the model requires the full representation of the state collected from the intersection. If the model is extended to the network level, this method would not be able to be computed efficiently. To tackle this limitation, Prashanth and Shalabh (Prashanth, 2010) developed a Q-learning technique with a function approximation method to reduce the size of inputs and significantly reduce the computing time to get the model to converge to optimal conditions. Instead of precise observation of queue lengths, the function approximation method abstracted the demand level and waiting time into several categories. The proposed model was compared to the prior work and the results show that Q-learning with function approximation provided better performance in terms of less computing time and data storage while maintaining the control performance (Abdulhai, 2003).

Lu et al. evaluated the performance of Q-learning for an isolated intersection with transition curve theory to estimate the delay for each approach (Lu et al., 2008). The state is total delay for the single intersection. The action sets include four phases with 2 seconds interval alternation. The reward function is the same as the state, which is total intersection delay. The proposed model was compared with the fixed signal settings and the results show a car in the system can save 21 seconds per cycle.

Chin et al. also applied a Q-learning algorithm to an isolated intersection. The state is the different levels of queue length and the number of phases in the signal plan (Chin et al., 2011). Actions were defined by the green time choice of each phase in a 5-second duration. Rewards were measured by the number of vehicles in the queue from all approaches. Various traffic conditions including flow saturation levels were examined in the simulations. The results showed that total delay could be reduced even in peak times compared to the optimized fixed-time signal.

2.3.2 Coordinated Intersections

Rather than focusing on improving the performance of adaptive signal control on an isolated intersection with the help of Q-learning algorithms, some research explored its benefits in the context of the transportation network with multiple intersections.

Balaji et al. designed a distributed multi-agent-based Q-learning traffic signal control for improving the existing adaptive signal control in an urban arterial network in the Central Business District of Singapore (29 intersections) to reduce the total delay and travel time (Balaji et al., 2010). Data collected from all intersections share information with adjacent intersections so the expected vehicle arrival patterns could be evaluated accurately. Parameters used in the model were fine-tuned with real-time information for the reinforcement learning model.

Simulation results showed significant delay reduction compared to other network traffic control systems.

Abdoos et al. explored the performance of multi-agent Q-learning for the network where peak traffic patterns do not appear and conventional traffic signal timing does not provide an efficient solution (Abdoos et al., 2011). Average queue length from all approaches in a fixed cycle was used as the state representation in the Q-learning model. Cycle time of all intersections remained the same during the optimization and the actions refer to the choice of remaining in the current phase or changing to the next one with a fixed, small amount of green time. The reward used in the model is inversely proportional to the average queue length from all links in the network, normalized to the range of 0 to 1, which could reduce the weight computing time in the Q-learning models. Fifty intersections were used to test the performance of the proposed model and two traffic profiles were used in the simulations, showing the proposed model outperformed the fixed signal timing plan in the same network by reducing the queue length and delay time. However, the size of the network exerted a significant computing burden on the system, and model improvement is needed.

Abdoos et al. developed a two-level hierarchical control model based on Q-learning (Abdoos et al., 2014). The bottom level comprises multiple intersections from a smaller region in the network and performs Q-learning to optimize the signal timing plan individually, while the top-level implemented tile coding to reduce the size of the state from the bottom level and abstract the model to a computing degree that field implementation of the proposed multi-agent Q-learning model could be practical. A network with 9 (3x3) intersections was used to show the performance of the proposed model and concluded that while maintaining the optimization of signal timing for bottom level intersections and also achieving promising results in reducing delay in the system from the top-level coordination.

2.3.3 Deep Q-learning

With the advent of the deep neural network, Q-learning has been improved and deep Q-learning models could yield more promising results.

With high-dimensional inputs available from intersections, such as camera images from surveillance cameras, simple Q-learning has difficulties representing the complex sensory inputs and actions and generalizing past experiences to new situations (Mnih, 2015). To mimic the human and animal brain learning process, a hierarchical neural network, termed deep neural network (DNN), was introduced to handle the extremely high complexity of input data and actions. Combined with Q-learning, deep Q-learning (DQN) was proposed for the existing Q-learning models. The prototype of DQN tested in the Atari 2,600 games significantly outperformed the previous Q-learning model based on pixel image data extracted from the games.

Since the publication of DQN, its application in traffic signal control has been evaluated. Genders and Razavi developed a DQN with experience replay for optimizing the signal timing of an isolated intersection (Genders and Razavi , 2016). Due to the advantages of DNN which can handle information-dense inputs efficiently, the state represented in this research contains the discrete cell representation of the road segment. Three vectors associated with each cell,

including vehicle presence status, the speed of each vehicle, and the current traffic signal phase, were used as the state, forming an information-dense input. Instead of only considering the queue length or average vehicle delay normally applied in the Q-learning methods, this kind of information-dense input could help the agent learn more from the complex input and generalize the experience to the new situations better, achieving a faster convergence with less computing time with similar parameter settings. Experience replay was another important technique used in this research, which uses extra memory to save the past experiences, a tuple of action, state, and reward, to train the model more efficiently. The proposed DQN for isolated intersection signal control improved the intersection performance compared to a one-layer Q-learning model, showing the benefits of applying DNN.

Ge et al. proposed a cooperative DQN with Q-value transfer for multi-agent-based adaptive signal control (Ge et al., 2019). Individual intersections relied on the deep Q-learning model to optimize their performance respectively. The cooperative mechanism was triggered when the centralized control system combined the latest optimal performance of each intersection and transferred the Q-value from adjacent intersections for a quicker learning process and less computing time.

In conclusion, deep Q-learning either for a single intersection or a network with multiple intersections can improve model performance by taking into account more complex sensory data and actions at the expense of more computing time. However, both existing research for Q-learning and deep Q-learning fail to consider the network with traffic incidents and hence prevent practitioners from understanding their performance in this situation.

2.3.4 Traffic Incident Management in Traffic Signal Control

Traffic congestion can be classified into two categories: recurring and non-recurring. Recurring congestion is due to traffic demand pattern variations throughout the day, such as traffic demand in the peak hours that exceeds capacity. Recurring congestion tends to occur daily and allows traffic management personnel to seek solutions. Non-recurring congestion comes from special events, such as traffic incidents and activities that increase travel demand such as major sports events. Traffic incidents, for example, occur in various locations, under different traffic patterns, and rarely repeat so traffic management agencies do not have enough experience to plan traffic signal adjustments required to deal with the real-time scenarios. (Mao, 2019).

Traffic incident management (TIM) aims to detect the incident rapidly and recover the transportation infrastructure capacity as quickly as possible (Carson, 2010). Various tools and strategies are proposed to facilitate traffic incident management, including manually adjusting adjacent traffic signals to temporally increase capacity to accommodate the traffic patterns under the impact of traffic incidents. However, due to the characteristics of traffic incidents, such as random locations, times of day, and traffic patterns, manually adjusting the traffic signals is almost impossible.

Logi and Ritchie proposed a knowledge-based system for non-recurring traffic congestion supporting traffic management personnel to select integrated traffic control plans, including traffic diversion and signal timing adjustment, to decrease traffic flow metering from the incident locations and increase capacity for the congested approaches (Logi and Ritchie,

2001). This traffic congestion management tool relied on the knowledge collected from a set of predetermined incident locations by varying the inputs, such as flow saturation degree and traffic signal timing parameters, to increase the uncertainty of the environment to mimic the real-time scenario. The model provides a selection of control plans for the users as well as the reasoning logic for the target goals. However, the model did not include enough detail about how to choose the adjacent signalization intersections, and algorithms for adjusting signal timing parameters were not provided.

Wirtz et al. evaluated the impact of traffic signal adjustment from a preplanning perspective for a full road closure on I-94 (Wirtz et al., 2005). Dynamic traffic assignment-based simulation was used to compare the traffic delay in time before and after manually adjusting the traffic signal plans near the incident locations. The results show that the preplanning of the traffic incidents in terms of traffic signal control adjustment could reduce the traffic delay and recover the roadway capacity faster than the original traffic signals operating in normal conditions.

Ban et al. developed a decision-making tool to determine if adaptive signal control is better than the existing actuated signal control system in real-world situations by using a regression model and support vector machines (Ban et al., 2016). However, this research failed to discuss the impact of traffic incidents in the comparison.

Mao et al. proposed genetic algorithms to optimize adaptive traffic signal control under severe incident conditions (Mao et al., 2019). This research first fine-tuned the model parameters in a recurrent traffic condition and then implemented the improved model in non-recurring situations. The results concluded that the proposed genetic algorithm reduces the traffic delay by over 40%.

2.4 Summary

Reinforcement learning is advantageous compared to conventional signal control methods. Data that is currently available to characterize the intersection or network state can become intensive and conventional methods cannot make use of this information as efficiently as reinforcement learning which relies on the computing capability of modern machines. For example, reinforcement learning could directly use camera images as the inputs for the learning model to extract useful information and output the model results. Second, reinforcement learning can take advantage of the model-free techniques, such as Q-learning, to avoid a need to explicitly model intersection characteristics to reduce errors of input interpretation. Third, reinforcement learning can autonomously improve itself as long as the computer runs which leads to the continuous improvement.

Although many research efforts have implemented reinforcement learning models in normal traffic conditions to show its advantages over conventional signal optimization methods, the analysis of reinforcement learning-based signal control under traffic incidents has largely been ignored. This dissertation contributes to filling this gap by building a reinforcement learning model, particularly the Q-learning model in traffic signal control by considering traffic incidents in the network to improve network delay reduction performance. Two main contributions will be included: 1. An independent incident generation and emergency vehicle response module will be developed in a microsimulation platform to generate incidents randomly in the network with random duration to provide the learning agent enough experience with

network traffic incidents. 2. The parameters of the DQN model used in the single intersection will be optimized. The derived model will be transferred into a network with 16 intersections (4x4) with little computing time to perform cooperative adaptive signal control to alleviate traffic impacts of traffic incidents. This would build the foundation for evaluating the deep Q-learning performance in the network settings in response to the random occurrence of traffic incidents in the network.

Chapter 3. Deep Reinforcement Learning Algorithm

3.1 Introduction

In this chapter, key concepts of reinforcement learning are illustrated as well as Q-learning and its variants for improvement.

3.2 Reinforcement Learning

Reinforcement learning is a process through which an AI agent takes sequential actions by interacting with its environment by trial and error to solve a task, which is often modeled as a Markov Decision Process (MDP). An MDP is a mathematical framework for modeling decision making in a discrete and stochastic control process (Howard, 1960).

At each time step t, the agent observes a state s from the environment, where $s \in S$ and S represents all possible states in the environment. The agent takes an action a by following some predetermined rules where $a \in A(s)$ and $A(s)$ represents all potential actions for the agent to perform at state s. The environment shifts to another state s^' with the impact of the performed action and sends a numerical signal, termed reward $R(s, a, s')$, to the agent to inform whether the action is promising as expected. The process repeats until the terminal state is reached. The whole process can be described in Figure 3.2-1.

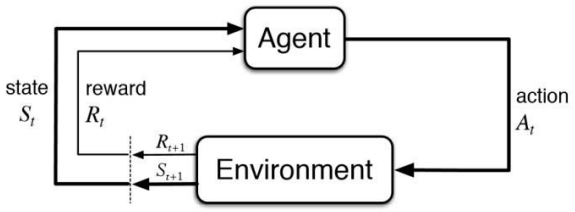

Figure 3.2-1. Closed Loop of Reinforcement Learning Process (Sutton and Barto, 2018)

State S is a member of the set of all observations of the environment represented by the model. Take traffic signal control as an example, if the longest queue length from each approach is determined to be used to represent an isolated intersection, the possible states can be represented by an integer array with a size of 4 and each integer represents the longest queue length of each approach with only straight movement traffic demands.

The set of actions A defines the choices of the agent to exert on the environment. In the case of traffic signal control, the agent is the traffic signal controller. The actions the agent can perform include extending the current phase green time or shifting to the next phase from the available phases. For some cases, the agent can skip phases if conditions are satisfied, which is predetermined by the modeler.

The Markov property states that the future is independent of the past (Markov, 1954). Therefore, the transition function P models the transition probability of new state S' based on the current state S and chosen action A as follows:

$$P(S' = s'|S = s, A = a) = p(s'|s, a)$$

In the case of traffic signal control, assume the current state is s and the signal controller chooses to extend the current phase. If we knew the transition function, we could get a deterministic new state. However, the transition function of a dynamic environment, such as a a transportation network, is hard to obtain and estimation is required to solve this issue.

Reward R is an immediate quantified signal that the agent receives from the environment as the result of taking an action, and it directly notifies the agent if the action is good or not. In the case of traffic signal control, if we choose to extend the current phase, but the current state has no traffic demand for it, this environment should send a negative reward to the agent that it is not a good choice.

To be more specific, the agent first observes its environment and senses the inputs, termed state. The agent takes an action from its action set based on the existing information and knowledge learned from its past experiences. The environment changes to the next state from the previous one based on the action performed by the agent. The new environment will send a signal, termed reward, to the agent telling the agent whether the taken action is good or not. If the reward is good, the agent will learn it and use a method to save this joy experience so it tends to perform the same action the next time it experiences the same state. Otherwise, the agent would not take the same action to avoid punishment, e.g., negative rewards. For a deterministic environment, where states and actions are limited, as long as the agent experiences all the possible states and actions, the agent will fully understand the relationship between its states and actions and hence achieve the maximum accumulative rewards through the process. Then, the agent has completed the learning task.

One of the challenges of reinforcement learning is when the MDP cannot be fully determined in terms of the transition function. Two common learning methods are used to overcome this issue. The first one isto build a model of the MDP and find the optimal policy. The second approach is to gain knowledge through experience (a tuple of state, action, reward, and new state) and estimate the optimal policy.

In the finite MDP, an episode denotes a process from the beginning state to the end state. In the case of traffic signal control, one round of simulation of a traffic demand with the signal control process can be called one episode. During each episode, the trajectory of the reinforcement learning process could be represented by a series of states, actions, and rewards. If the learning process is finite and the final time step is denoted by T, the whole learning trajectory could be expressed as:

$$S_1, A_1, R_1, S_2, A_2, R_2, \dots, S_T, A_T, R_T$$

The goal of learning is to maximize the total rewards, termed as returns denoted by G_t at time step t.

$$G_t = \sum_{i=t}^{T} R_i$$

In the above equation, every reward from time t is equally important since there is no weighting factor for each one. However, in reality, the rewards might not be the same. For example, which one will you choose, $1,000 now or $1,000 one year later? The answer is definitely obvious. You will choose to get the money as soon as possible because money tends to depreciate in the long run. The same concept was introduced to the. Discount factor, γ is used to quantify this effect and the return can be calculated by the following equation:

$$G_t = \sum_{i=t}^{T} \gamma^{i-t} R_i$$

The discount factor is a value between 0 and 1, inclusively. If it is set to 0, only the immediate reward will be considered. If it is set to 1, future rewards have the same value as the current one. Normally, the discount factor is set to be a value slightly less than 1 so we treat future rewards as less important than the immediate ones and it will eventually decay to 0 if the time steps are large enough.

The discounted returns can also be expressed as the following by considering that the MDP is executed one time step at a time:

$$G_t = R_t + \gamma G_{t+1}$$

This equation reflects the relationship between two consecutive returns. Note that all the rewards, R_i, where $i = t, t+1, \dots, T$, in this equation have not been observed so they are random variables. We use r_t to denote the observed reward. The randomness of R_t comes from two sources. First, the action can be randomly chosen if exploring the environment early in the training stage. The other one is due to the randomness of the new state from the environment.

3.3 Q-Learning

Since R_t is a random variable with respect to the states and actions starting at time step t, the returns G_t is also a random variable with respect to the states and actions. To calculate G_t, we need a way to estimate future rewards. Q-learning is the most common algorithm to calculate returns based on the Temporal Difference (TD) learning concept. TD learning is a combination of Monte Carlo (MC) estimation and Dynamic Programming (DP). MC estimation allows the agent to learn from its experience without explicitly modeling the environment (model-free) and update estimates based on other estimates, while DP can be used to calculate the returns based on parts of observations and parts of estimations (Sutton and Barto, 2018).

The Q-value, known as the action-function value, $Q_\pi(s_t, a_t)$, is used to represent the expectation of returns G_t with respect to the state and action at time t as:

$$Q_\pi(s_t, a_t) = E(G_t | S_t = s_t, A_t = a_t)$$

Since we have $G_t = R_t + \gamma G_{t+1}$, we can express the Q-value as follows:

$$\begin{aligned} Q_\pi(s_t, a_t) &= E(R_t + \gamma G_{t+1} | S_t = s_t, A_t = a_t) \\ &= E(R_t | S_t = s_t, A_t = a_t) + \gamma(E(G_{t+1} | S_t = s_t, A_t = a_t)) \\ &= E(R_t + \gamma Q_\pi(S_{t+1}, A_{t+1})) \end{aligned}$$

There must exist at least one policy that leads to the maximum action-value function and we use $Q^*$to indicate this optimal action-value function. Whatever policy is used, we cannot improve the action-value function by taking action a_t at the given state s_t. Normally, we can remove the π from Q to simplify the expression.

$$Q^*(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t)$$

The best action leads to the maximum action-value function which can be expressed by:

$$a^* = \operatorname{argmax}_a Q^*(s_t, a_t)$$

Since we do not know the expected value of rewards R_t and returns from the next time step, we use the observed r_t and $Q_\pi(s_{t+1}, a_{t+1})$ to estimate the Q-value (Watkins, 1989). Combined with the DP concept to update the action-value function based on parts of the observations and parts of the estimations, we have the Q-learning expression (Watkins, 1989), defined by:

$$Q^*(s_t, a_t) = (1 - \alpha) Q^*(s_t, a_t) + \alpha[r_t + \gamma \max_a Q^*(S_{t+1}, a)]$$

Where α is called learning rate, a hyper-parameter that is not learned from the learning process but determined by the modelers in advance. The learning rate determines how much the old Q-value should be changed based on the estimated Q-value. Q-learning trains the optimal action-value function $Q^*(s, a)$. In the above equation, the second part of the equation is called the TD target, which is a combination of the observed reward by executing one time step and the estimated optimal Q-value from the next time step, expressed by:

$$y_t = r_t + \gamma \max_a Q^*(S_{t+1}, a)$$

TD error represents the difference between the target value and the existing value, expressed by:

$$\delta_t = Q^*(s_t, a_t) - y_t$$

Therefore, the Q-learning update equation can be expressed by:

$$Q^*(s_t, a_t) = Q^*(s_t, a_t) - \alpha \delta_t$$

Using the TD learning method reduces the difference of the TD error through experience. Once the error cannot be reduced anymore (smaller than a threshold), the learning is considered

to be converged and the learning process can be terminated. The Q-learning method will converge as long as each state-action pair can be visited enough times (Watkins, 1992).

To enable the agent to explore efficiently early in the learning process, ε-greedy policies are used by giving all nongreedy actions the minimal probability, $\frac{\varepsilon}{|A(s)|}$, where ε is a value between 0 and 1 and the denominator is the size of the possible actions. For greedy actions, the probability is set to $1 - \varepsilon - \frac{\varepsilon}{|A(s)|}$. As learning proceeds when the agent has more knowledge, the action choice will be cleverer and more efficient by lowering the probability of choosing random actions. This is normally realized by applying ε decay methods.

3.3.1 Tabular Q-learning

For a simple environment with a small number of state-action pairs, one can use the tabular method to solve the Q-learning problem. This method uses a table, termed Q-table, to save the Q-value of each state-action pair during the learning process. Once the algorithm converges, the final Q-table can be used to guide the agent to choose an action at any given state to achieve the maximum expected returns.

An algorithm for solving the Q-learning problem by the tabular method is listed below:

Algorithm: Q-learning (Watkins, 1989)

Set hyperparameters, including
  learning rate $\boldsymbol{\alpha} \in (0, 1]$,
  discount factor $\boldsymbol{\gamma} \in [0, 1]$,
  greedy factor $\boldsymbol{\epsilon} > 0$,
  number of episodes $\boldsymbol{n\_episodes}$
Initialize $\boldsymbol{Q^*(s, a)}$, for all $\boldsymbol{s} \in \boldsymbol{S}$, $\boldsymbol{a} \in \boldsymbol{A(s)}$, arbitrarily except that $Q(terminal, \cdot) = 0$
For episode = 1, $\boldsymbol{n\_episodes}$ do
  Initialize sequence $\boldsymbol{s_1}$
  For time step $\boldsymbol{t} = 1, \boldsymbol{T}$ do
    With probability $\boldsymbol{\varepsilon}$ select a random action $\boldsymbol{a_t}$ by following the **$\boldsymbol{\varepsilon}$-greedy policy**
    Otherwise select $a_t = \underset{a}{\operatorname{argmax}}\, Q(s_t, a)$
    Execute action $\boldsymbol{a_t}$ and observe reward $\boldsymbol{r_t}$ and state $\boldsymbol{s_{t+1}}$
    Update ***Q-table*** by
      $$Q^*(s_t, a_t) = (1 - \alpha)Q^*(s_t, a_t) + \alpha[r_t + \gamma \max_a Q^*(S_{t+1}, a)]$$
    Set $s \leftarrow s_{t+1}$
  End
End

Figure 3.3-1. Bus line simulation demonstration Algorithm: Q-learning

3.3.2 Deep Neural Network and Deep Q-learning

For a more complex environment, when the number of state-action pairs is too large to be stored with a Q-table or when it is impossible to visit each state-action pair, the optimal action-value function ($Q^*(s,a)$) can be approximated. Hence DQN was introduced to improve the capability of Q-learning (Mnih et al., 2015).

$$Q^*(s,a) \approx Q(s,a;\theta)$$

Here θ represents the learning parameters in the DQN. The essence of DQN is the deep neural network (DNN). DNN is comprised of at least three layers of artificial neural network, as shown in Figure 3.3-2. Activations of one layer determines activations of the next layer and the inputs proceed. Each layer detects a pattern from the previous layer. With a large number of hidden layers, the model can detect more sub patterns, compared to the model with a small number of hidden layers. If the number of hidden layers is less than what is required to extract important features from the inputs, the model might under fit the data. Otherwise, overfitting could occur. The number of hidden layers tends to correspond to the complexity of the input layer, which is a hyper-parameter to be tuned by experimentation.

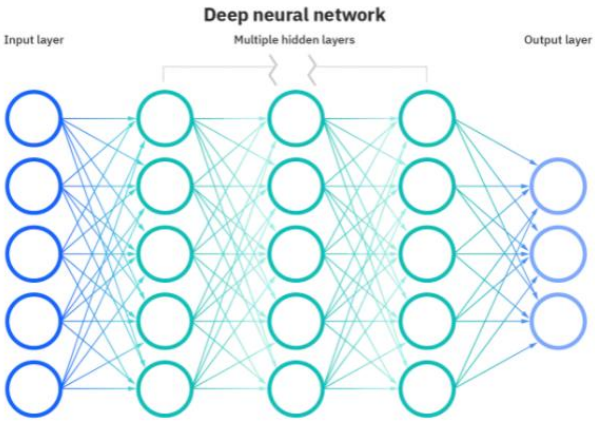

Figure 3.3-2. Deep neural network with three hidden layers (https://www.ibm.com/cloud/learn/neural-networks)

Each neural network includes a certain number of nodes and each node is called an artificial neuron. Each neuron takes the outputs from the previous layer and outputs a number between 0 and 1 by normalization to reduce the computing time. For example, the first layer of the DQN is the input layer and the value of each node is only dependent on the inputs. Each node in the hidden layers is initialized with an arbitrary weight, as a connection between the nodes in adjacent layers, but cannot be the same for all nodes otherwise the model cannot distinguish the importance of them. Each node in the next layer is the weighted sum of all the nodes from the previous layer, similar to a linear regression model. Each layer will be assigned an activation rule and if the value of a node meets this activation rule, it will be activated and its value will be passed to the next layer as an input. All weights between the hidden layers are termed parameters

in the DNN. The concept of "learning" is a process of updating weights associated with each node in the hidden layers to minimize the cost function, also known as the loss function, so that the learning model can accurately predict and maximize returns from the inputs. The number of nodes in each hidden layer is a hyper-parameter that must be fine-tuned as is the number of hidden layers in the DNN.

As mentioned before, the parameters in the DNN are randomly selected by initialization. How does the model learn from the inputs? No matter what models of machine learning one uses, one must have a predicted value and a target value. In deep Q-learning, the target value is based on observation of one time-step reward and the estimated optimal Q-value from the next state. The predicted value is the current Q-value updated by the Q-learning update equation. The difference between the predicted value and target value tells the performance of the current learning model. Gradient descent is the most commonly used method to find an improving direction to lower the loss function and this process is called forward propagation, while back propagation is used to update the weights in the network based on the loss function and the weights.

The process of solving Q-learning with the DQN approximation has the following steps:

Algorithm: Q-learning with DQN (Mnih et al., 2015)

Set hyperparameters, including
  learning rate $\boldsymbol{\alpha} \in (0, 1]$,
  discount factor $\boldsymbol{\gamma} \in [0, 1]$,
  greedy factor $\boldsymbol{\epsilon} > 0$,
  number of episodes $\boldsymbol{n\_episodes}$,
  number of hidden layers $\boldsymbol{n\_layers}$,
  number of nodes for each hidden layer $\boldsymbol{n_{nodes}}$
Initialize $\boldsymbol{Q(s, a; \theta)}$, for all $\boldsymbol{s} \in \boldsymbol{S}$, $\boldsymbol{a} \in \boldsymbol{A(s)}$
For episode = 1, $\boldsymbol{n\_episodes}$ do
  Initialize sequence $\boldsymbol{s_1}$
  For time step $\boldsymbol{t} = 1, \boldsymbol{T}$ do
    With probability $\boldsymbol{\varepsilon}$ select a random action $\boldsymbol{a_t}$ by following the **$\boldsymbol{\varepsilon}$-greedy policy**
    Otherwise select $a_t = \underset{a}{\operatorname{argmax}}\, Q(s_t, a; \theta)$
    Execute action $\boldsymbol{a_t}$ and observe reward $\boldsymbol{r_t}$ and state $\boldsymbol{s_{t+1}}$
    Set the TD target $y_j = \begin{cases} r_j & if\ episode\ terminates\ at\ step\ j+1 \\ r_j + \gamma \max\limits_{a} Q(s_{j+1}, a; \theta) & Otherwise \end{cases}$
    Perform gradient descent on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to $\boldsymbol{\theta}$
    Set $s \leftarrow s_{t+1}$
  End
End

Figure 3.3-3. Algorithm: Q-learning with DQN

3.4 Deep Q-learning Variations

Although deep Q-learning has achieved promising results for many applications, it may be unable to converge when implementing a neural network (McClelland et al., 1995). Two main reasons can lead to this issue.

First, we use a transition, (s_t, a_t, r_t, s_{t+1}), in the deep Q-learning mentioned earlier to train the model. Successive transitions can be correlated with each other and hence make the model update highly correlated. Second, every time one transition is used to update the model, it will be discarded and will never be used again. Experiments have shown that using consecutive transitions without any improvement to train the DQN can result in inefficient training regardless of training time. Therefore, experience replay was introduced (Lin, 1992).

3.4.1 Experience Replay

The purpose of implementing experience replay is to reduce the impact of correlated transitions for the training process. To implement experience replay, a data structure which is a list of tuples (past transitions), termed a replay buffer, is used. The size of the replay buffer, N, is a hyper-parameter that must be tuned and cannot be trained by the learning model.

The replay buffer stores N past transitions, a tuple of (s_t, a_t, r_t, s_{t+1}) called experience e_t. The model will not begin the training until the replay buffer is filled with past experiences with the size N. A minibatch (certain number of experiences), termed batch size, will be randomly and uniformly selected from the replay buffer to train the model. The chosen experiences are equally important in terms of improving the model. To maintain the size of the replay buffer, the oldest experiences are replaced by the latest ones.

3.4.2 Target network

Another known limitation of the DQN is overestimating the Q-value (Van Hasselt et al., 2016). Recall the TD target in DQN is defined by:

$$y_t = r_t + \gamma \max_a Q(s_{t+1}, a; \theta)$$

The TD target is partly based on the observation r_t and partly on the estimate of DQN for the state at the next time step. Since we always choose an action to maximize the Q-value, the model will overestimate the TD target and hence overestimate the Q-value overall. To solve this issue, the concept of the target network was proposed (Mnih et al., 2015). Instead of using the DQN parametrized by θ, to calculate the TD target, the target network uses another DQN with parameter θ^-, which will be fixed in a certain amount of time steps for the agent to have a fixed target to learn from. The target network will be updated every C time steps, a hyper-parameter. The loss function associated with implementing the target network can be defined as follows to minimize the mean square error:

$$L_t(\theta_t) = E[(r_t + \gamma \max_a Q(s_{t+1}, a; \theta_t^-) - Q(s_t, a_t; \theta_t))^2]$$

Gradient descent is the common method to reduce the loss function by following the direction of the derivative of the loss function with respect to θ_t:

$$\frac{\partial L_t(\theta_t)}{\partial \theta_t} = -E[(r_t + \gamma \max_a Q(s_{t+1}, a; \theta_t^-)) \frac{\partial Q(s_t, a_t; \theta_t)}{\partial \theta_t}]$$

With the combination of experience replay as mentioned earlier, random samples from the replay buffer will be extracted to update the DQN parameter, termed stochastic gradient descent to break the correlation of successive experiences.

3.5 Summary

The DQN with experience replay and target network was introduced by Minh (2015) and will be adopted to train the traffic signal controller for the single intersection scenario in this research.

The pseudocode of the algorithm used in this dissertation is listed below:

Algorithm: Deep Q-learning with Experience Replay and Target Network (Mnih et al., 2015)

Set hyperparameters, including
  learning rate $\boldsymbol{\alpha} \in (0, 1]$,
  discount factor $\boldsymbol{\gamma} \in [0, 1]$,
  greedy factor $\boldsymbol{\epsilon} > 0$,
  replay memory capacity $\boldsymbol{N}$,
  minibatch size m,
  update steps $\boldsymbol{C}$,
  number of episodes $\boldsymbol{n\_episodes}$,
  number of hidden layers $\boldsymbol{n\_layers}$,
  number of nodes for each hidden layer $\boldsymbol{n_{nodes}}$

Initialize replay memory $\boldsymbol{D}$ to capacity $\boldsymbol{N}$
Initialize action-value function $\boldsymbol{Q}$ with random weights $\boldsymbol{\theta}$
Initialize target action-value function $\widehat{\boldsymbol{Q}}$ with weights $\boldsymbol{\theta^-} = \boldsymbol{\theta}$
For episode = 1, $\boldsymbol{n\_episodes}$ do
  Initialize sequence $\boldsymbol{s_1}$
  For time step $\boldsymbol{t} = 1, \boldsymbol{T}$ do
    With probability $\boldsymbol{\epsilon}$ select a random action $\boldsymbol{a_t}$
    Otherwise select $a_t = \underset{a}{\operatorname{argmax}}\, Q(s_t, a; \theta)$
    Execute action $\boldsymbol{a_t}$ and observe reward $\boldsymbol{r_t}$ and state $\boldsymbol{s_{t+1}}$
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\boldsymbol{D}$
    Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from $\boldsymbol{D}$
    Set the TD target $y_j = \begin{cases} r_j & if\ episode\ terminates\ at\ step\ j+1 \\ r_j + \gamma \max\limits_{a} Q(s_{j+1}, a; \theta_t^-) & Otherwise \end{cases}$
    Perform stochastic gradient descent on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to $\boldsymbol{\theta}$
    Set $\widehat{\boldsymbol{Q}} = \boldsymbol{Q}$ every $\boldsymbol{C}$ steps
  End
End

Figure 3.5-1. Algorithm: Deep Q-learning with Experience Replay and Target Network

Chapter 4. Simulation Preparation

4.1 Overview

Simulation is a primary method by which municipal traffic engineers establish confidence in innovative traffic signal timing concepts. This confidence is ordinarily established by characterizing the field network, collecting traffic demand data and testing potential signal retiming policies to analyze network performance (e.g., average vehicle delay). In recent years, as the development of reinforcement learning methods has evolved, a goal-oriented machine learning process can be applied decreasing analysis effort. AI traffic signal controllers have been studied and a variety of control techniques have been tested through traffic simulations.

The core concept of reinforcement learning algorithms is to explore the relationship between the agent's actions and its evolving environment by trial-and-error methods. Feedback from the environment measured by so-called rewards can help the agent adjust its behavior so as to achieve more rewards in the future.

The next generation traffic signal control system is far from the field application since many aspects, including traffic incidents, have not been tested thoroughly. One major reason is that collecting field network data associated with traffic incidents and validating the proposed models are expensive and time-consuming. For example, collecting historical traffic incident characterizations and emergency vehicle response data are rarely feasible for most researchers, who might want to focus more on the development of traffic signal retiming.

The agent can learn from interaction with the environment regarding impacts of traffic incidents in the network. However, a large amount of experience is required to enable the agent to optimally respond to all possible incident situations so implementation of an AI traffic signal without sufficient experience would be disastrous. An implemented AI traffic signal should perform at least better than existing signal timing plans with/without traffic incidents.

A more inexpensive and practically feasible traffic simulation tool with traffic incident/response and AI signal control module would be helpful in promoting a smarter more robust traffic signal control system. Therefore, we provide a Python extension based on SUMO to allow micro-simulation of an AI signal control system in a network experiencing incidents randomized in both time and space.

We begin by highlighting some existing efforts in developing the next generation traffic signal control systems and available simulation software with traffic incident/response capabilities. This is followed by the extraordinary features of SUMO and the framework we used to extend SUMO with a traffic incident/response module coded in Python. Experiments are presented to show the use of the extended module.

4.2 Literature Review

Traffic signal retiming plays a significant role in improving the network performance when traffic incidents occur. Due to the complicated inputs and short period of time for making decisions, traffic simulations have been commonly adopted to test potential traffic signal

retiming policies before field implementation. Liu and Hall proposed a Windows operating system-based computer simulation software for simulating highway traffic incidents as well as emergency vehicle dispatching (Liu and Hall, 2000). Traffic delay is the only factor considered in the model and queue spill back effect is not simulated. It could help researchers and practitioners to broadly understand the impact of traffic incidents and determine the emergency dispatch strategies as needed. However, there are several key limitations. It only focused on the highway so local networks are not simulated. Users cannot simulate traffic signals in the software, which limits the usage of this simulation tool.

Kaan and Bartin developed a complete traffic incident simulation tool in Siman language to generate incidents in the network and to send emergency vehicles to respond accordingly (Kaan and Bartin, 2003). Real network and real-world data were collected to test their proposed simulation tool. This work allows users to implement different TIM strategies to reduce the impact of traffic incidents in the network. However, the programming language Siman is rarely used in the data scientist and machine learning modeling field these days since most of the advanced machine learning methods are written in Python. It will prevent users from effectively testing their TIM strategies which will include the latest machine learning technologies.

Ozbay et al. proposed Rutgers Incident Management System (RIMS) to simulate traffic incidents and to test various incident response strategies based on the cell transmission model developed by Daganzo (Ozbay et al., 2009). The results indicate that computer simulation methods could significantly reduce the traffic delay triggered by a traffic incident in the network. However, this tool lacks traffic signal timing, limiting the usage of it.

Huang and Pan proposed to use a GIS engine to facilitate traffic incident and incident response optimization management. The idea was tested with real cases and commercial traffic simulation software (Huang and Pan, 2007).

Wirtz et al. proposed a simulation-based method to test traffic incident management strategies in Visual Interactive System for Transport Algorithms (VISTA), a dynamic traffic assignment (DTA) embedded tool (Wirtz et al., 2005). The DTA offers the opportunity for modelers to accurately estimate the impact of the traffic incidents by considering the dynamic change of road capacity and link travel time, where the static traffic assignment models fail to perform. Network and traffic demand were extracted from the Chicago Area Transportation Study and various incidents in terms of locations and durations were simulated around I-94. Eleven surrounding traffic signals were manually adjusted based on Webster's formula to split the green time to accommodate the changed traffic flow pattern once the incident occurs. Incident response actions of closure of a certain number of ramps upstream of the incident locations were analyzed to find the best traffic delay alleviation strategies. The idea of this research is to preplan the traffic incident management strategies and take corresponding incident response actions once the incident occurs. This method might be helpful when the incident locations are fixed and traffic demand patterns could be predicted. However, in reality, due to the complexity of the network and scale of inputs, including traffic demand, network characteristics, and existing traffic signal control methods, it is hard to find the optimal traffic incident strategies within a short period of time without human interactions.

Reinforcement learning methods have been adopted in the field of traffic signal retiming. The main advantage of the reinforcement learning methods is to allow use of deep neural networks to perform approximation of inputs from the environment and estimate cumulative long-term expected rewards with a model-free method. To achieve the accuracy of high-level function approximation, large amounts of data need to be prepared.

Common limitations of existing traffic incident simulation tools are:

- The tools have not been maintained and published so that other users find it hard to replicate the experiments or design new experiments to test traffic signal control strategies. A free and open-source simulation software is needed.
- The existing tools are not able to generate a test network and associated traffic demand so as to minimize the costs of preparing the base scenario. Most existing experiments use a single or multiple real data points to simulate the traffic incident environment. This scale of inputs is not enough to train the machine learning models.
- The functions in the existing tools are not comprehensive enough to test proposed strategies from different angles, including vehicle rerouting and traffic signal retiming.
- The existing tools are not available for multi-cross platforms, preventing the use of high-performance computing advantages these days.
- The simulation environment is closed, meaning it is hard for the users to customize and extend.
- Measurement of Effectivenesses (MOEs) are limited and do not catch up with the network performance measurement nowadays when vehicle emission and fuel consumptions are required to be considered.
- Manually generating test networks, traffic demand, and incident occurrence is not efficient for training machine learning algorithms for traffic signal retiming.

There is a need for a simulation testbed that incorporates the traffic signal retiming and traffic incidents/response system to develop a more robust AI traffic signal control system. The purpose of this work is to provide a highly automated process to generate random traffic incidents in the given network as well as the corresponding emergency service vehicles as an extension based on the existing popular microscopic traffic simulation software SUMO. Key components in the extension include random traffic incident generation, traffic incident detection, emergency service vehicle generation, and emergency service vehicle dispatching. By conducting simulations in this kind of setting, new traffic signal control methods considering the traffic incident impact in the network can be tested and tuned as preparation for field application.

4.3 Simulation Platform

SUMO simulation requires at least two files, including a network file and a route file. The network file defines the road network, including intersections, edges, and connection rules. The traffic signals can be also included in the network file. There are several common types of traffic signals provided in SUMO, including pretimed, actuated, adaptive, and other more

advanced (self-organized traffic signals) control frameworks. Detectors are also provided with the user's definition, including loop, area detector, etc. Users can also customize the traffic control algorithms as needed, including the reinforcement learning traffic signal control methods.

4.3.1 Network

Another benefit of using SUMO is that it provides a network generation library (NETGENERATE) so that users can easily build a grid-like network. This library allows users to determine the number of intersections in horizontal and vertical directions in the network. Users can also choose the number of lanes and length of each approach for each intersection.

Pretimed traffic signals can also be added to the target intersections in the network. The tool provides a way to set up the cycle length, left turn protection phases, green split, yellow time, and all red time durations to mimic the practical applications as needed.

4.3.2 Traffic Demand

SUMO provides another important and useful Python script to prepare traffic demand randomly based on the developed network if users cannot get access to any trip information of the network. It is convenient to the users who focus on evaluating a more generalized traffic signal control algorithm so they do not have to spend time collecting field data. The tool allows users to set the ratio of internal and external traffic demand as needed. In this study, we assume that all traffic demand is external traffic so the ratio between the through and internal traffic demand is set to an extremely large integer, meaning all the traffic is starting and ending from the fringe of the network. The trip table will be saved into a XML file so the experiments can be repeated.

Another commonly used way in SUMO to generate the traffic demand is to dynamically add vehicles to the system. The problem with this method is that the generated traffic will calculate the shortest path in the network dynamically so it might be able to detour around the incident location and hence decrease the traffic impact.

In order to isolate the impact of traffic signal retiming provided by the AI traffic signal agent, we need to lock the traffic routes so that when there is a traffic incident in the system, the traffic would not shift routes. This is not the case in reality where travelers would shift routes to avoid being stuck in a long queue in the network. However, we assume that no travelers would change routes for two reasons. First, the benefits of optimizing the signal plan based on the AI traffic signal agent need to be calculated. If the travelers are allowed to shift routes, the net benefits of signal plan optimization are difficult to quantify. The other reason is the ratio of travelers who shift routes and remain on the original ones relate to the traffic demand pattern and characteristics of intersections, which are hard to quantify for this research.

The same thing should not happen to the traffic demand generated later after the incident time. Therefore, this paper decided to use the first method mentioned above and edit the original traffic demand file (XML format) to add traffic incidents, including incident locations, incident durations, and emergency vehicle response.

4.3.3 Incident Generation

SUMO provides three methods to simulate traffic incidents in the network: 1. Stop a car at a designated location for a specific period; 2. Reduce the road capacity of associated edges; 3. Reduce the design speed of the associated road edges. The easiest and more realistic manner is the first one since it will require the route file to be edited with one line of code to reflect the stop of an incident vehicle. Figure 4.3-1 shows the added traffic incident information in the route file. In this example, the vehicle with ID 2 will stop at Lane "C2C1_1" 20 meters from the end of this lane for 1500 seconds.

```
<vehicle color="1,0,0" depart="2.00" id="2">
    <route edges="right2C2 right2C2.100.00 C2C1 C2C1.100.00 C1C0 C1C0.100.00 C0right0"
    />
<stop duration="1500" endPos="20" lane="C2C1_1" parking="false" /></vehicle>
```

Figure 4.3-1. Settings of Stopping Vehicle in Route File

4.3.4 Emergency Service Vehicles Simulation in Sumo

In addition to the incident vehicle generation, we also provide a way to generate emergency service vehicles in the simulation once the traffic incident is detected.

During normal traffic movement, no vehicle will stop at a location for a long period of time, a user-defined time threshold (e.g., 5 minutes). Once the system has detected that a vehicle is stuck in the network for more than a specific period of time, the emergency service vehicles will be generated and dispatched. Users can choose the number of emergency service vehicles to reflect the reality process, such as multiple police cars and EMS vehicles.

We can generate multiple individual vehicles to mimic the police and EMS vehicles, but the problem of this is some of the vehicles might not be able to reach the incident location due to associated traffic congestion. To overcome this issue, we decided to edit the emergency vehicle length to mimic multiple emergency service vehicles being needed.

The default length of per emergency service vehicle is 7.5 meters, including 5 meters for the vehicle length and 2.5 meters for the clear space. For example, if 3 police cars and 1 EMS vehicle are required to deal with a traffic incident, that is a total of 4 emergency service vehicles, a vehicle with length of 30 meters, will be generated and dispatched in the simulation and hence it will block 30 meters of the incident lane to reflect the combination impact of multiple emergency services vehicles in practice.

In SUMO, there are several important concepts of network components. Edge defines the approach of the intersection. Edge includes a certain number of lanes. The lanes are named based on the edgeID and lane index.

The route of the emergency service vehicle is defined before dispatching it into the network. To generalize the implementation of emergency vehicle response, we randomly select an origin for its route from the fringe of the network. The destination of the route is the incident

edge. During the incident response service, the emergency vehicle will occupy the lane next to the incident vehicle. For example, if the incident vehicle stops at the middle lane of an edge and there are 3 lanes for this edge, the emergency service vehicle will randomly stop in either the first (straight and right turn lane) or the third lane (left turn lane in our experiment) of the same edge.

The emergency vehicle will arrive at the incident location after the incident vehicle has been detected and the travel time from its origin to the incident location. And then the emergency service vehicle will stop for the same duration as the incident vehicle stops. Once the emergency service vehicle completes its service it will finish its route and reach the intersection of the destination edge.

In Traci, the function to generate a route based on the origin and destination edges is traci.simulation.findRoute(origin_edge, destination_edge). Once the two parameters are given, the function will find a feasible and probably the shortest route in the network. The route information could be called to show the edges used in this route by calling the route.edges property.

To dispatch the emergency service vehicle in the system, the function traci.route.add(routeID, route_edges) needs to be called to add the edges of the emergency service vehicle route into the route file. The emergency service vehicle then can be added to the route file by calling traci.vehicle.setStop(vehicle_id, route_edges, stop_lane_index, stop_duration). Users can customize the traffic signal to allow emergency service vehicle priority so that it can arrive at the incident location as quickly as possible.

For a two-lane edge, the whole edge will be blocked by both the incident car and emergency service vehicle, while for a three or more-lane edge, two lanes will be blocked and its capacity will be reduced significantly. We examined the impact of only considering the incident vehicle without the emergency service vehicle in the system and the delay impact is significantly different, showing that having the emergency service vehicle in the system should be more realistic. The pseudo code for the simulation is shown in Figure 4.3-2.

```
Step = 0
hasIncident = False
While numberOfVehicles() > 0:
        executeSimulation() # 1 step, e.g., 1 second per step
        If incident detected() and not hasIncident:
                getIncidentInformation() # including location and stopping duration
                generate emergency service vehicle() and dispatch()
                hasIncident = True # this will avoid generating emergency service vehicle
        multiple times
        Step += 1
```

Figure 4.3-2. Pseudocode for the Simulation Framework

4.4 Simulation Procedure

Once the network and traffic demand are prepared, the customized incident Python script will read the route XML and randomly select a vehicle to generate a traffic incident. Then the simulation starts and once the incident vehicle is detected in the network stopping for more than 5 minutes (a user defined threshold), emergency service vehicles will be generated by calling DISPATCH_EMERGENCY_VEHICLE() function. The default color of the emergency vehicle is set to be blue and the length of it is determined by the number of emergency service vehicles needed for this incident multiplied by 7.5 meters. Therefore, the incident vehicle will be shown in red and the emergency service vehicle will be shown in blue when viewing the animation of the simulation process.

The developed incident generation and emergency service vehicle response Python script is published in the following GitHub repository. The Networks directory includes a 4x4 grid network generated by calling the NETGENERATE command aforementioned, as shown in Figure 4.4-1. The main functions of incident generating and emergency service vehicle response are in incidentRoute.py located in the root directory of this GitHub repository.

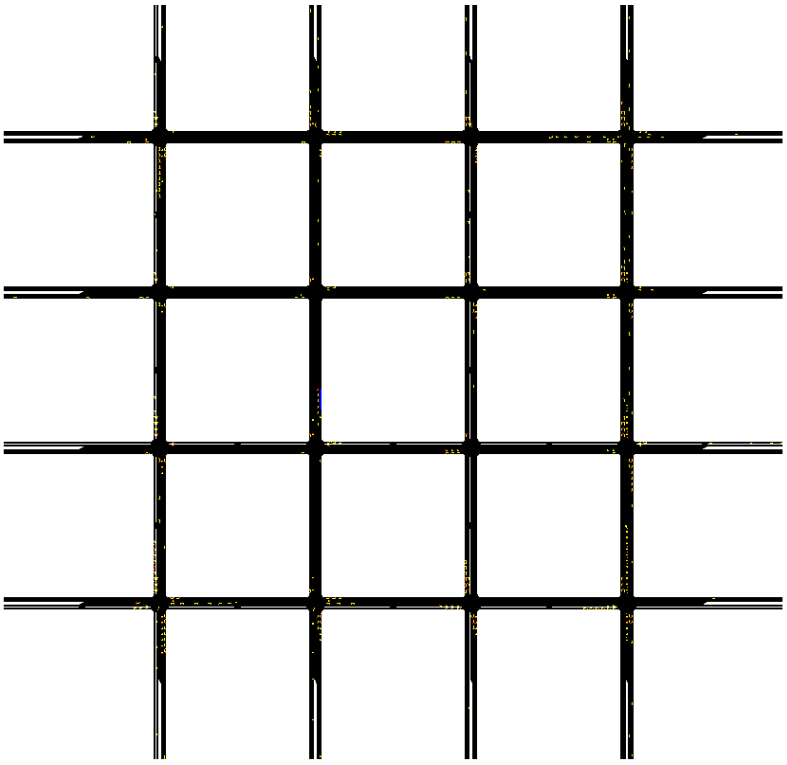

Figure 4.4-1. 4x4 Grid Network with Traffic Incident

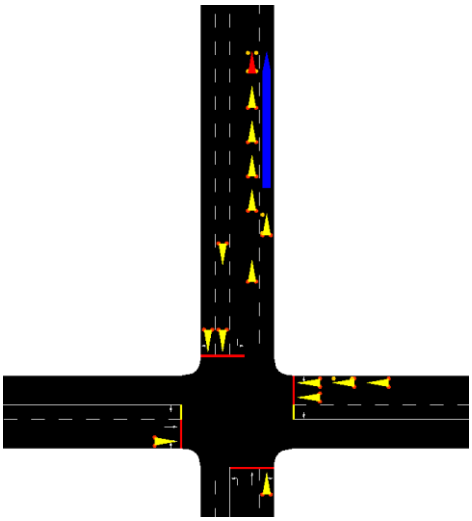

Figure 4.4-2. An Example of the Incident Vehicle (Red) and Emergency Service Vehicles

In Figure 4.4-1, the 4x4 grid network is shown as well as the incident vehicle and emergency service vehicle. Figure 4.4-2 shows a larger view of the incident vehicle (RED) and emergency service vehicle (BLUE).

The current Python script is for single traffic incident preparation. Users could extend it to include multiple incidents as needed.

4.5 Implementation

A 4x4 grid network can be created by running the NETGENERATE module provided by SUMO to generate a grid-like network with user settings. All the parameters as well as their meaning can be found in Appendix 1.

Traffic demand associated with the 4x4 grid network can be prepared by running the trip generating Python script in SUMO. Some of the key parameters that users can define include the ratio of internal and external trips, hourly traffic flow, and turning ratios. The commands used in this paper can be found in Appendix 2 as well as the explanation of the parameters.

Traffic demands are all external traffic, meaning the origins and destinations of all trips are at the network fringe.

Intersections are all controlled by pretimed traffic signals with 4 phases, including east-west straight movement phase, east-west left turn phase, north-south straight movement phase, and north-south left turn movement phase. Right turn movement is allowed and included in the straight movement phases. Cycle time for every intersection is the same, 90 seconds. The cycle is split 50%-50% between the east-west and north-south directions with the straight/right movement receiving 27 seconds and the left turn phase receiving 13 seconds with both yellow clearance intervals 3 seconds and 2 second all reds. More traffic signal setting information could be found in the network file.

To add random incidents in the created network, the python script must be called. Several inputs need to be defined before calling the extension, Including the network XML file provided by the NETEDIT function, the traffic demand XML file produced by calling the randomTrip.py tool provided by SUMO, and the corresponding SUMO configuration file.

The traffic incident will be generated before the simulation starts by randomly selecting a vehicle from the first one third of the simulation period. The vehicle must have a route crossing the center of the network so that the stopped lane is not located in the fringe of the network to prevent vehicles from entering the network. The traffic incident will last for a random period of time from 15 minutes to 30 minutes with a 5-minute increment. Once the vehicle reaches the incident location, it will fully stop and block the traffic behind.

The traffic simulation system will record all vehicles' stop duration in the network. Once it detects one vehicle stopped for more than 5 minutes (a tunable parameter) in the same location, it assumes a traffic incident exists.

A number of emergency medical service and police cars will be generated as a single long length vehicle to abstract their impact. The origin of this emergency vehicle will be a random location along the network fringe and its destination is the incident location. The emergency vehicle will be stopped for the same amount of time as the incident vehicle . For a two-lane edge, the incident vehicle and emergency vehicle will fully block the road.

The extension could be customized easily if multiple incidents are required for any scenario.

4.6 Summary

This work provides a convenient Python script for SUMO extension. Rather than only considering the traffic incident impact in the network, this research also provides a way to simulate the emergency service vehicle impact in the network. As shown in the experiment results, the combination impact of traffic incidents and corresponding emergency service vehicle response could cause significantly more delays than only considering the traffic incident itself in the network. This tool will help researchers to provide more realistic traffic incident management strategies to reduce the impact of traffic incidents and optimize the traffic management methods, including vehicle rerouting and traffic signal retiming.

Chapter 5. Single Intersection Deep Reinforcement Learning Traffic Signal Control

5.1 Overview

This chapter employs the proposed deep reinforcement learning signal control algorithm to a single intersection simulation scenario, highlighting the potential advantages of using deep Q-learning for the traffic signal controller problem. Additionally, to assess the performance of the proposed algorithm, we compare it against three traditional non-learning traffic signal control algorithms, including Max-pressure, Webster's, and Uniform. To evaluate the effectiveness of the traffic signal controllers, we utilize measures of effectiveness (MoEs) such as system travel time, queue length distribution, and delay distribution of vehicles during the simulation.

The subsequent sections of this chapter are structured as follows. The second section provides an introduction to the deep Q-learning algorithm, which is utilized along with three traditional non-learning traffic signal controllers. The simulation platform settings and code preparation are then detailed. In the Results section, a comprehensive analysis is conducted to compare the performance of the learning and non-learning traffic signal control algorithms. The final section provides a discussion and concluding remarks on the applicability of the deep Q-learning algorithm to the single intersection scenario.

5.2 Deep Q-Learning Model

As Chapter 3 explains, Q-learning is a type of reinforcement learning that does not rely on a pre-existing model and allows for learning the value of actions in a given state. In certain situations, a Q-table can be used to explore all possible state and action combinations, allowing the agent to develop a coherent policy that maximizes cumulative rewards once the model has reached convergence. However, for traffic signal control problems, where the number of states and action pairs is exceptionally large, a Q-table may not be practical.

To provide an example, let's consider a single intersection where the number of states and action pairs is dependent on factors such as the number of vehicles in each lane, the number of lanes per approach, the capacity of each lane, and the signal phasing patterns. When dealing with multiple intersections, it is preferable to have an optimized system solution. Instead of obtaining actual rewards for each state-action pair, we can use a deep neural network (DNN) to estimate the performance of an action in a given state. This combination of Q-learning and DNN is called the deep Q-network (DQN). In this section, we will introduce our DQN, which includes defining the state, action, and reward, as well as specifying the DNN and associated hyperparameters.

The key components of reinforcement learning are:

- Agent: The entity that interacts with the environment and learns to take actions based on the observed states to maximize the reward.
- Environment: The external world in which the agent interacts and receives feedback in the form of rewards.
- State: The current configuration of the environment that the agent observes.

- Action: The decision made by the agent to transition from one state to another.
- Reward: The feedback signal that the agent receives from the environment after taking an action. The reward represents the immediate benefit or cost of the action taken by the agent.
- Policy: The strategy that the agent uses to determine its actions based on the current state of the environment.

5.2.1 Agent

In machine learning, an agent is an entity that interacts with an environment to achieve a specific goal. The agent can receive observations or data from the environment, take actions based on that information, and receive feedback or rewards that indicate how well it is achieving its goal. The agent's objective is typically to learn a policy, which is a mapping from observations to actions, that maximizes its long-term cumulative reward in the environment. Agents can be implemented using a variety of techniques, including reinforcement learning, supervised learning, and unsupervised learning, depending on the nature of the task and the available data.

The traffic signal controller is represented as the agent in DQN, which aims to achieve the maximum cumulative reward by interacting with the intersection and traffic demand through its learned policy.

5.2.2 Environment

The environment comprises everything except the agent, such as the geometry of the intersection, vehicle arrival rate, queue lengths, delay, and other factors that are beyond the agent's control. In our case, the intersection and its characteristics serve as the environment that the agent interacts with during the learning process.

5.2.3 State

The inputs in DQN are represented by the state, denoted as s_t, which belongs to the state space S and $t \in T$, where T represent the time period for the learning process. T is fixed in our experiment for the single, isolated intersection, meaning our learning process is a finite Markov decision process.

A suitable state must capture the essential features of the environment. In DQN, the state should include essential information from the intersection that the traffic signal controller can learn to improve its policy. Common measures used for state representation in traffic signal control include queue length, queue density, delay, vehicle waiting time, and their variations and combinations. Some more advanced states can be represented by the image of the intersection with vehicle positions which allows the model to extract information that humans might not be able to detect. However, more advanced and complicated inputs are difficult to obtain in practice.

By considering the complexity and ease of implementation of algorithms in practice, we choose normalized density of each lane (both incoming and outgoing), normalized queue length

of each lane (for both incoming and outgoing), and the most recent green phase as the state. The reason for normalizing the density and queue is to constrain the value to the range between 0 to 1. The normalized values for the inputs of machine learning models will generally decrease the training time to get the model converged (Goodfellow et. al, 2018).

Normalized density, $density_{l \in L}$, is defined by the ratio between total vehicles and lane capacity, where l denotes the lane and L represents a set of all incoming and outgoting lanes associated with the intersection. Normalized queue, $queue_{l \in L}$, is calculated as the ratio between the number of stopped vehicles and lane capacity. One-hot encoding of the most recent green phase is applied, plus the all red phases.

To summarize, the state in our single intersection case can be defined as below:

$$s_t = [density_{l_1}, \dots, density_{l_n}, queue_{l_1}, \dots, queue_{l_n}, phase_1, \dots, phase_m]$$

Where n denotes the number of incoming and outgoing lanes and m represents the total number of green phases and one all red phase, subject to $density_l \in [0, 1]$, $queue_l \in [0, 1]$, $phase_i = [0, 1]$, and $\sum_m phase_i = 1$.

5.2.4 Action

Action is defined as the choices that the agent can make and hence it is the phases that can be selected in our single intersection scenario. Action is represented by $a_t \in A$, where a_t denotes the action being chosen at time t and action space A is a set of all selectable phases (green phases and the all red phase). We have four green phases in our single intersection scenario, including East-West straight movement green phase with unprotected left-turn green ($phase_{straight-EW}$), North-South straight movement green phase with unprotected left-turn green ($phase_{straight-NS}$), East-West protected left-turn green phase ($phase_{EW-left-turn}$), and North-South protected left-turn green phase ($phase_{NS-left-turn}$).

Since we randomly generate demand for the experiment including random ODs, we include two protected left-turn phases into the action space. For regular cyclic traffic signal controllers, the pattern will be fixed so left-turn green phases will be applied, while for the DQN and Max-pressure controllers, left-turn phases might be less used due to the traffic pattern.

In the DQN mode, each chosen phase will be up for at least t_{green} seconds. If the next chosen phase is the same, it extends the current phase by adding another t_{green} seconds. If a different phase is selected, the corresponding amber phase will be chosen and hence all red time thereafter, where $t_{yellow} = 3$ seconds and $t_{red} = 2$ seconds are fixed.

5.2.5 Reward

Reward at time t, r_t, serves as a numeric signal to train the DQN so the agent can quantify its action given a state and improve its performance by choosing the right action for maximizing long-term value measured by the reward.

As summarized in Chapter 2, commonly used reward representations include total delay and its variation, total stops, total queue length and its variation, and some combinations of those measurements. A number of research efforts have chosen total delay or its variation as the reward based on the assumption that ultimately, the system level performance will be measured by the total delay, so using the same measurements as the reward will directly guide the agent to improve its performance.

However, total delay requires knowing each driver's desired speed and their actual speed through the network so that the difference could represent total delay. Taking into account this obstacle when implementing the DQN algorithm, we use a queue related reward in our model. There are multiple forms of using queue length as the reward representation, and we use the quadratic form of queue difference between each incoming lane and outgoing lane, as shown below:

$$r_t = \sum_n queue\_length^2_{t-1,incoming_i} - \sum_n queue\_length^2_{t,incoming_i}$$

$The\ term\ queue\_length_{t,incoming_i}$ denotes the number of vehicles stopping in ith incoming lane. The quadratic form is used to penalize the long queues to avoid having unfair phase selection for those vehicles from the minor demand approaches. We also use the previous sum of squared queue length minus the current one so if an action reduces the value, the reward is positive and vice versa.

The goal of the agent is to maximize the cumulative rewards, defined by the following formula:

$$R = max \sum_{t=0}^{T} r_i$$

5.2.6 Policy

A policy is a function that maps the current state of an agent to an action to be taken by that agent. The policy defines the agent's behavior and determines what actions the agent should take in response to the environment.

There are two main types of policies in reinforcement learning: deterministic policies and stochastic policies. A deterministic policy maps each state to a single action. For example, a deterministic policy might always output "move forward" when the agent is in a certain state. A stochastic policy, on the other hand, maps each state to a probability distribution over actions. For example, a stochastic policy might output a probability of 0.7 for "move forward" and a probability of 0.3 for "turn left" when the agent is in a certain state. The agent then selects an action according to the probabilities given by the policy.

The goal of reinforcement learning is to learn an optimal policy that maximizes the agent's long-term reward. This is typically done by using a trial-and-error approach, where the agent explores the environment and updates its policy based on the observed rewards. In our

case, the policy represents the weights, θ, of the DNN which will help to choose an action based on a given state to maximize the cumulative rewards. Once the learning process is done, we can use the value saved in θ to approximately calculate the best actions we should choose given a state. For every step, if we follow this guidance, we will maximize cumulative rewards and hence find the best policy to choose a phase given a state.

5.2.7 DNN Structure

A deep neural network (DNN) is a type of artificial neural network (ANN) that is designed to model complex relationships between inputs and outputs by using multiple layers of processing nodes, or neurons, to learn hierarchical representations of the data. DNNs are composed of layers of interconnected nodes, with each node in a layer receiving input from the previous layer and outputting to the next layer. The nodes use nonlinear activation functions to transform their inputs and create a nonlinear relationship between the inputs and outputs. By stacking multiple layers of these nodes, a DNN can learn to extract increasingly abstract and complex features from the input data. DNNs have been applied successfully in a wide range of machine learning applications, including image and speech recognition, natural language processing, and reinforcement learning.

DNNs consist of multiple layers of interconnected neurons that perform increasingly complex transformations on the input data. Common layers in DNNs include:

- Input Layer: The input layer receives input data and passes it to the next layer in the network. It typically does not perform any computation on the input. In our case, this is defined by the inputs collected from the intersection and can be used by the controller to learn.
- Hidden Layers: Hidden layers process the input data and perform non-linear transformations to extract features and learn patterns in the data. The number of hidden layers and the number of neurons in each layer can vary depending on the complexity of the problem being solved. This will help the controller to analyze the inputs collected from the intersection for various patterns to facilitate the learning process.
- Output Layer: The output layer produces the final output of the network. The number of neurons in the output layer depends on the task being performed. For example, in a binary classification task, the output layer would have a single neuron, whereas in a multi-class classification task, the output layer would have multiple neurons. This will form a list of phases with its estimated value from the model to determine which phase has the maximum cumulative rewards to be chosen for the next phase if exploitation is applied. Otherwise, a random phase will be chosen for the next phase.

A deep neural network represents a neural network structure with one input layer, one output layer, and multiple middle layers, called hidden layers. The size of the input layer is the same as the size of the state. The size of the output layer is equal to the number of actions in the

action space, which is 4, the number of green phases in our model. The size of the hidden layers is determined by the number of layers and number of nodes for each layer. The number of nodes for each layer is fixed to be 64. The number of layers is a hyperparameter which will be tuned.

Every layer is fully connected, meaning each node will be passed as an input for the next layer. Each connection between two nodes is represented by a single value in our DNN parameter, θ. Each node can be seen as a multiple regression model that includes all the node values from the previous layer and is weighted by the parameters in θ corresponding to this layer. The weighted sum will be reculated by the chosen activation function.

The activation function is a mathematical function that introduces non-linearity to the output of a neuron. It determines the output of a neuron based on the weighted sum of its inputs. Some common activation functions used in ANNs include:

- Sigmoid function: The sigmoid function maps any input value to a value between 0 and 1, making it useful for binary classification problems. However, it suffers from the vanishing gradient problem, which can make training deep networks difficult.
- ReLU function: The rectified linear unit (ReLU) function outputs the input directly if it is positive, and outputs 0 if the input is negative. ReLU has become a popular choice in deep learning due to its simplicity and ability to avoid the vanishing gradient problem.
- Tanh function: The hyperbolic tangent (tanh) function maps input values to a range between -1 and 1, making it useful for regression problems. It is similar to the sigmoid function but has a steeper gradient, which can improve the convergence of the training process.
- Softmax function: The softmax function is commonly used in the output layer of a neural network to produce probabilities for each class in a multi-class classification problem. It ensures that the output probabilities sum to 1.0.

In our case, ReLU function is applied to be the activation function for each hidden layer.

Q-learning is a well-known reinforcement learning algorithm that is used to find an optimal policy for an agent in an environment by learning the action-value function. However, traditional Q-learning can face limitations when dealing with high-dimensional state and action spaces.

DNNs, on the other hand, are very good at approximating complex non-linear functions, making them a powerful tool for function approximation in reinforcement learning problems with high-dimensional state and action spaces.

By combining Q-learning with DNNs, we can approximate the action-value function with a deep neural network, a technique known as the deep Q-network (DQN). The DQN algorithm can learn directly from raw high-dimensional sensory inputs, such as images, without requiring a

manual feature extraction step. This can greatly simplify the design process, and enable the agent to automatically learn and extract relevant features from the environment.

Overall, DQN, combining Q-learning with DNNs, can lead to better performance and more efficient learning in complex reinforcement learning tasks with high-dimensional state and action spaces.

5.3 Variations of DQN

One of the limitations of DQN is its susceptibility to overestimation of Q-values, especially in the presence of noisy data. This can occur when the DNN overgeneralizes from the limited training data, resulting in overestimation of Q-values for some state-action pairs. Another limitation is the tendency of DQN to overfocus on specific state-action pairs, leading to suboptimal policies. This can be addressed using various modifications to the basic DQN algorithm, such as experience replay, double DQN, dueling DQN, and distributional DQN, which aim to improve stability, reduce overestimation, and encourage exploration. Two advanced techniques will be used in this research to improve the performance of DQN, including experience replay and double DQN.

5.3.1 Experience Replay

Experience replay is a technique used in deep reinforcement learning to improve the efficiency and stability of the learning process. The basic idea is to store experiences (tuples of state, action, reward, next state) in a replay buffer with size B, which is essentially a large dataset of past experiences. During training, some of the experiences, determined by the variable called batch size (b), are randomly sampled from the replay buffer and used to update the deep neural network, instead of using only the most recent experience. This has several benefits, such as reducing the correlation between consecutive experiences, making the learning process more stable, and enabling the reuse of experiences, which can lead to more efficient use of data.

Experience is defined as a tuple of current state, action, reward, and next state, (s_t, a_t, r_t, s_{t+1}). The experiences that aid the agent in learning from its interaction with the environment are stored in a fixed-size memory buffer. When the buffer is full, the oldest experience is replaced with the newest to retain the most recent experiences. In an ideal scenario, all past experiences could be used to calculate the loss function for improving the model, but this would significantly slow down the learning process. Instead, a batch of experiences is randomly and uniformly selected from the buffer for updating the DQN. The size of the batch is a hyperparameter that requires tuning.

5.3.2 Double DQN

Double DQN is an extension of the original DQN algorithm that addresses the overestimation issue that can occur in Q-learning methods. In traditional Q-learning, the maximum action value for a given state is calculated using the same Q-network used to select actions, which can lead to overestimation of the action values. Double DQN uses two separate Q-networks to calculate the action values: one network is used to select actions, while the other is used to estimate the action values. The second network, called target network, is used to evaluate the action values by taking the maximum action value from the network used to select

actions. This approach reduces the overestimation issue that can occur in Q-learning and improves the stability and accuracy of the Q-values.

Target network update refers to periodically updating the weights of a separate neural network, known as the target network, that is used to estimate the target Q-values in the Q-learning update.

During training, the Q-learning update involves calculating the target Q-value for each action based on the current estimate of the Q-values and the observed reward and next state. In a standard DQN, the same neural network is used to estimate the current Q-values and the target Q-values. However, this can lead to instability in the training process, since the Q-learning update involves using the same neural network to generate the target and the prediction, leading to a feedback loop.

To address this, the target network is updated periodically (e.g., every N steps) by copying the weights of the current Q-network to the target network. This provides a more stable estimate of the target Q-values and prevents the feedback loop. The target network is not used for action selection during the training, only for estimating the target Q-values.

The DQN with experience replay and target network was introduced by Minh (2015) and will be adopted to train the traffic signal controller for the single intersection scenario in this research.

The pseudocode of the DQN with experience replay and target network algorithm used in this dissertation is listed below:

Algorithm: Deep Q-learning with Experience Replay and Target Network (Mnih et al., 2015)

Set hyperparameters, including
  learning rate $\boldsymbol{\alpha} \in (0, 1]$,
  discount factor $\boldsymbol{\gamma} \in [0, 1]$,
  greedy factor $\boldsymbol{\epsilon} > 0$,
  replay memory capacity $\boldsymbol{N}$,
  minibatch size m,
  update steps $\boldsymbol{C}$,
  number of episodes $\boldsymbol{n\_episodes}$,
  number of hidden layers $\boldsymbol{n\_layers}$,
  number of nodes for each hidden layer $\boldsymbol{n_{nodes}}$

Initialize replay memory $\boldsymbol{D}$ to capacity $\boldsymbol{N}$
Initialize action-value function $\boldsymbol{Q}$ with random weights $\boldsymbol{\theta}$
Initialize target action-value function $\widehat{\boldsymbol{Q}}$ with weights $\boldsymbol{\theta^- = \theta}$
For episode = 1, $\boldsymbol{n\_episodes}$ do
  Initialize sequence $\boldsymbol{s_1}$
  For time step $\boldsymbol{t} = 1, \boldsymbol{T}$ do
    With probability $\boldsymbol{\epsilon}$ select a random action $\boldsymbol{a_t}$
    Otherwise select $a_t = \underset{a}{\operatorname{argmax}}\, Q(s_t, a; \theta)$
    Execute action $\boldsymbol{a_t}$ and observe reward $\boldsymbol{r_t}$ and state $\boldsymbol{s_{t+1}}$
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\boldsymbol{D}$
    Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from $\boldsymbol{D}$
    Set the TD target $y_j = \begin{cases} r_j & if\ episode\ terminates\ at\ step\ j+1 \\ r_j + \gamma \max\limits_{a} Q(s_{j+1}, a; \theta_t^-) & Otherwise \end{cases}$
    Perform stochastic gradient descent on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to $\boldsymbol{\theta}$
    Set $\widehat{\boldsymbol{Q}} = \boldsymbol{Q}$ every $\boldsymbol{C}$ steps
  End
End

Figure 5.3-1. Algorithm: Deep Q-learning with Experience Replay and Target Network

5.4 Non-learning Traffic Signal Control Algorithms

To evaluate the effectiveness of the proposed DQN, we compared its performance against that of two traditional traffic control algorithms (Uniform and Webster's) as well as the more recent and advanced Max-pressure algorithm. This was done to generate similar Measures of Effectiveness (MoEs) and to demonstrate the performance of our proposed model.

5.4.1 Uniform Traffic Controller

A uniform traffic controller (UTC) is a type of traffic control system where the signal timing plan is fixed and does not change dynamically based on real-time traffic conditions. The UTC uses a pre-determined signal plan to control traffic at an intersection, which is often designed to provide uniform signal timings for each phase of the traffic signal cycle. This type of traffic control system is widely used in areas with relatively stable traffic demand patterns and limited traffic variations. However, UTC may not be able to adapt to sudden changes in traffic flow or accommodate the needs of different types of road users. As a result, more advanced and adaptive traffic control systems, such as actuated or intelligent traffic control systems, are being developed and implemented to improve traffic efficiency and safety. It is also due to this reason, all traffic signal controllers in this dissertation will apply the same phase patters described in the DQN.

- Here is a general procedure for implementing a uniform traffic controller:
- Set the initial phase to be the first phase in the sequence.
- Allocate equal green time to each phase.
- Monitor traffic and detect when a phase has no demand. When this occurs, skip that phase in the sequence.
- When all phases have been completed, return to the first phase and start the cycle over again.

The only hyperparameter in UTC is green duration for each phase (we use the same green duration for each phase), which will be tuned with the given traffic demand.

5.4.2 Webster's Traffic Controller

Webster's traffic controller, also known as the Webster method, is a type of traffic signal control algorithm developed by Anthony G. Webster in the 1950s. It is a fixed-time control method, where the green times for each phase are predetermined based on traffic flow rates and the geometric characteristics of the intersection.

The Webster method assumes that the traffic flow rates are known and constant, and the signal timing plan is set in advance. The controller calculates the total cycle time and the duration of each green interval based on the traffic demands of each approach, the saturation flow rate, and the intersection geometry. The cycle time is the total duration of one complete signal sequence, while the green interval is the period of time when a particular movement is allowed to proceed through the intersection.

Webster's method is relatively simple and requires minimal input data. It is widely used for low- to moderate-volume intersections and has been the basis for other traffic control methods, such as the fixed-time coordinated method. However, it may not be suitable for high-volume intersections or complex traffic conditions, where adaptive signal control methods may be more effective.

We implement an adaptive Webster's method by collecting traffic demand through a fixed time interval to average the traffic demand and assume that the next interval with the same length will have the similar traffic demand. Therefore, the recalculated green split will be reasonable.

The procedure of Webster's traffic controller can be summarized as follows:

- Collect traffic data: Traffic volume data is collected from the intersection, including the number of vehicles arriving on each lane, the queue length, and the delay time.
- Determine cycle time: The total cycle time for the traffic signal is determined based on the traffic demand and the minimum green time required for each phase.
- Calculate green times: The green time for each phase is calculated based on the traffic demand and the pre-determined fixed time ratios for each phase.

- Implement the signal timings: The signal timings for each phase are programmed into the traffic signal controller, which will operate the traffic signal according to the pre-determined timings.
- Monitor traffic flow: The traffic flow at the intersection is monitored to ensure that the traffic signal timings are effective and efficient. If necessary, the timings can be adjusted based on the traffic data collected.
- Repeat the process: The above steps are repeated on a regular basis, usually daily, to ensure that the traffic signal timings are optimal for the current traffic demand.

The hyperparameters in Webster's method include minimum cycle length, maximum cycle length, saturation flow rate, and time interval to recalculate the green time split. All of these hyperparameters will be tuned to obtain the best performance of Webster's method for the performance comparison with other controllers.

5.4.3 Max-pressure Traffic Signal Controller

Max-pressure is a traffic control algorithm that aims to maximize the flow of traffic through an intersection by prioritizing the lanes with the highest pressure, which is defined as the difference between the number of vehicles entering the lane and the number of vehicles leaving the lane. The Max-pressure algorithm is a decentralized control algorithm, which means that each lane controller makes its own decisions based on local information, without requiring communication or coordination with other controllers. The pressure of a particular phase is defined as the sum of queue length of incoming lane, $queue_{incoming}$, minus the queue length of its corresponding outgoing lane, $queue_{outgoing}$ for all the incoming lanes and outgoing lanes associated, as listed below.

$$\boldsymbol{pressure_i = \sum_l queue_{incoming_l} - queue_{outgoing_l}}$$

As indicated by the previous equation, a phase's pressure can be negative, which implies that the downstream lane has a greater vehicle queue than its incoming lane. Consequently, its pressure value turns out to be negative, making it almost impossible to be selected. The phases with higher pressure are selected more frequently in order to alleviate the pressure in the system. In this study, the Max-pressure algorithm necessitates traffic data from the intersection's surroundings, particularly the length of the outgoing vehicle queue.

The procedure of Max-pressure traffic controller can be explained below.

- At each time step, the controller obtains the current queue length of each outgoing lane and calculates the pressure of each phase. The phase with the highest pressure value is selected as the next phase to be executed. If there are multiple phases with the same highest pressure value, one is selected randomly.
- After a phase is selected, the controller assigns a green time duration for the phase based on a pre-defined green time ratio. The green time ratio is the proportion of

the total green time that a phase is assigned. The total green time is the sum of the green times of all selected phases in a cycle.

- The Max-pressure controller repeats this process in each time step to ensure that the pressure of the system is reduced as much as possible.

There is only one hyperparameter in Max-pressure traffic signal control and that is the minimum green duration for a given phase, denoted as t_{green}. The hyperparameter will be optimized in order to achieve the optimal performance.

5.5 Hyperparameter Tuning

In machine learning, there are two types of parameters: model parameters and hyperparameters.

Model parameters are learned during the training process. They are the weights and biases that the model learns from the data to make predictions. In supervised learning, model parameters are updated using an optimization algorithm to minimize the difference between the predicted outputs and the actual outputs for a given set of input data. For example, in a linear regression model, the model parameters are the slope and intercept of the line that best fits the data.

Hyperparameters are set by the user before training the model. They are not learned from the data, but they affect how the model learns the model parameters. Hyperparameters control aspects of the training process such as the learning rate, regularization strength, and the number of hidden layers in a neural network. For example, in a neural network, the model parameters are the weights and biases of the neurons, while the hyperparameters are the learning rate, the number of hidden layers, the number of neurons in each layer and the activation functions used.

Both model parameters and hyperparameters are important in machine learning, and selecting the right values for both can significantly affect the performance of the trained model.

5.5.1 Hyperparameters in DQN

In machine learning, hyperparameters are parameters that are not learned from the data, but are set by the user before training the model. They are called "hyperparameters" because they determine how the model's parameters (which are learned from the data) will be set during the training process.

Some examples of hyperparameters include:

- Learning rate: determines how much the model weights are updated during training.
- Number of hidden layers: determines how many layers are in the neural network.
- Batch size: determines how many examples are used in each iteration of training.
- Activation function: determines the function used to transform the input data in each layer.

Hyperparameters are typically set using trial and error or more advanced optimization methods such as grid search, random search, or Bayesian optimization. Selecting the right hyperparameters is important because it can significantly affect the performance of the trained model.

There is no universal rule of determining the best combination of hyperparameters due to the complexity of real world environment and therefore to achieve a good machine learning model, hyperparameter tuning is required, although many research efforts do not even mention it. To our knowledge, this is the first time that a full suite of hyperparameter tuning has been applied to the traffic signal control problem and a detailed explanation of the process for applying the reinforcement learning model to traffic signal control has been provided. This contribution will fill this gap so other researchers and engineers who are interested in implementing machine learning algorithms to the traffic signal control problem can have a good starting point.

Because of the large number of hyperparameters in DQN and the many potential values for each, it is impractical to exhaustively search for the best combination. To simplify the process, this research employs the commonly used method of grid search to tune the hyperparameters. Grid search involves testing all possible combinations of hyperparameters from a predetermined list, and training the model with each combination for a small fraction of the total training time required to achieve convergence to an acceptable level. The performance of each combination is then evaluated to identify the set of hyperparameters that produce the best preliminary results, which is used to begin the actual training process.

It should be noted that the value of some hyperparameters may be adjusted during the training process. For instance, the learning rate may be decreased as the agent gains a better understanding of the environment and the model reaches a state where a lower learning rate may allow for more exploration of local areas that were not reachable with the larger learning rate. Table 5.5-1 lists all the tuned hyperparameters along with a brief definition for each.

Table 5.5-1. Hyperparameters tuned in DQN

Hyperparameters	Definition
Reinforcement Learning Related Hyper-parameters	
Learning Rate	To govern the pace the algorithm learns the parameter through previous and current rewards
Discount Factor	Discount the future reward so not to have an infinite calculation
Temporal Difference Steps	Number of steps the reward will be used to calculate the target q value

Neural Network Related Hyper-parameters	
Number of Hidden Layers	Number of layers between the input layer and output layer
Target Frequency	Number of time steps to update the target neural network

5.5.2 Learning Rate

In reinforcement learning, the learning rate is a hyperparameter that determines the degree to which the agent's Q-values are updated based on new experiences. It controls the step size at which the agent updates its estimates of the optimal Q-values for each action. A small learning rate means the agent will change its estimates slowly, while a large learning rate means it will update them more quickly.

The learning rate is typically set to a small value (e.g., 0.1 or 0.01) to ensure the agent learns gradually and avoids overfitting to specific experiences. However, the optimal learning rate can depend on the specific environment and problem being tackled, so it is often a hyperparameter that needs to be tuned through experimentation.

Based on the existing research on single intersection, we select three values for the learning rate to be tuned, including , 10^{-3}, 10^{-4}, and 10^{-5}.

5.5.3 Discount Factor

In reinforcement learning, the discount factor is a parameter that determines the importance of future rewards in an agent's decision-making process.

The discount factor, denoted by γ (gamma), is a value between 0 and 1 that represents how much an agent values future rewards. A value of 0 means that the agent only cares about immediate rewards, while a value of 1 means that the agent values all rewards equally, regardless of when they occur.

Three values are used to find the optimal one, 0.5, 0.9, and 0.99.

5.5.4 Temporal Difference Step

Temporal Difference (TD) is a learning method used in reinforcement learning, where the agent learns to predict the value of the next state by updating its current estimate of the value function based on the difference between the observed reward and the predicted reward. The TD step involves calculating the TD error, which is the difference between the observed reward and the predicted reward, and updating the value function estimate based on this error.

In the TD step, the agent observes the current state, takes an action, and receives a reward and the next state. The agent uses the observed reward and the estimated value of the next state

to calculate the TD error. The TD error is then used to update the value function estimate for the current state. This process is repeated for each time step, allowing the agent to learn to predict the value of the next state based on its current estimate of the value function. The size of the TD step is controlled by the learning rate and the discount factor.

We use two values in the tuning process for the TD Step 1 and 2.

5.5.5 Number of Hidden Layers

The number of hidden layers in a reinforcement learning (RL) algorithm depends on various factors, such as the complexity of the problem and the size of the input and output spaces.

In general, deep reinforcement learning algorithms, which use deep neural networks as function approximators, often have multiple hidden layers. The number of hidden layers can range from a few to dozens, depending on the complexity of the problem and the amount of available training data.

However, it is important to note that the number of hidden layers is not the only factor that affects the performance of an DQN algorithm. Other factors such as the number of neurons in each layer, the activation functions used, and the optimization algorithm also play important roles in the success of a DQN algorithm.

To reduce the number of combinations of hyperparameter tuning, we use a fixed number (64) of nodes in each hidden layer and only tune the number of hidden layers to achieve the goal of tuning the architecture of the DNNs. Based on the existing research about the single intersection scenario as well as the input definition in our simulation, we choose two values for the number of hidden layers, 3 and 6, resulting in 5 and 8 total layers for the DNNs combining with the input and output layers.

5.5.6 Target Frequency

In reinforcement learning, the target frequency refers to how often the target network is updated to match the parameters of the primary network. The target network is a separate copy of the primary network used to estimate the value of the next state in the Q-learning algorithm.

The target network is updated less frequently than the primary network to provide a more stable and consistent target for the Q-learning algorithm. The target frequency is a hyperparameter that determines how often the target network is updated, and it can affect the stability and convergence speed of the algorithm.

A common approach is to update the target network every Csteps, where C is the target frequency hyperparameter. This approach is used in the DQN algorithm, where the target network is updated every fixed number of steps.

We choose two values for the target frequency, 64 and 128.

5.5.7 Minimum Green Duration

We also tuned the minimum green duration. This value determines the minimum green time for each phase. Two values are included in the tuning process, 6 and 12 seconds.

5.5.8 Non Tuned Hyperparameters

Some of the hyperparameters in the DQN are not tuned based on the fact that they are easy to be determined based on the previous research and application. In addition, it is also an effective method to reduce the total number of combinations of hyperparameters in the tuning process and hence significantly reduce the computing time.

The replay buffer with size, denoted by B, saves a certain number of past experiences of the agent to help calculate the loss of DQN and facilitate the model to converge. It seems a larger size of replay buffer favors better model performance. However, this should be varied based on the environment. A good way to determine it is it should be large enough to collect different types of experience so the agent can handle almost every state-action pair. We use 40000 for the replay buffer size in the single intersection scenario.

Batch size, denoted by b, determines the number of experiences randomly selected from the replay buffer to be passed to the model. The minimum value of it can be 1 and the largest is the size of the replay buffer. Normally, this value is equal to the power of 2 to take advantage of the computer memory unit. We use 128 in our DQN.

The greedy factor refers to the degree to which the agent prioritizes exploitation of the current best action versus exploration of new actions. A value of 1 for the greedy factor means the agent always chooses the current best action, while a value of 0 means the agent always chooses a random action. A common approach is to start with a high value for the greedy factor to encourage exploration, and then gradually reduce it over time to focus more on exploitation. This trade-off between exploration and exploitation is a fundamental challenge in reinforcement learning. We apply this logic by using 1 over the simulation time to decrease the value of the greedy factor.

Episode defines the time of the training process, meaning the larger value, the longer experiment will be required. This hyperparameter is not explicitly tuned since we can easily increase the learning time as needed. For the hyperparameter tuning, we use 5000 as the value for the episode. Since each simulation lasts 3 hours, the total training time for each hyperparameter combination is equal to repeating the 3-hour simulation 5000 times, which should be a large enough simulation period to get a sense of the performance of each combination of hyperparameters.

5.5.9 Summary

Table 5.5-2 summarizes the parameters used in the tuning and training process. If there is only one single value, that parameter is not tuned, otherwise, it is a tuned parameter.

Table 5.5-2. Parameters in DQN including hyperparameter values

Parameters	Value/Values
Learning Rate	[0.0001, 0.00001, 0.001]
Discount Factor	[0.5, 0.9, 0.99]
TD Step	[1, 2]
Number of Hidden Layers	[3, 6]
Target Frequency	[64, 128]
Green Duration	[6, 12]
Episodes	5000
Replay Buffer Size	40000
Batch Size	128
Number of Nodes Per Hidden Layer	64
Activation Function	ReLU

Overall, we have total of 144 combinations of hyperparameters in the tuning process. Hyperparameters for non-learning controllers are list in Table 5.5-3.

Table 5.5-3. Hyperparameters for non-learning controllers

Hyperparameters	Values
Uniform Traffic Controller	
Green Duration	range(5, 26)
Webster's Traffic Controller	
Minimum Cycle Length	[40, 60, 80]
Maximum Cycle Length	[160, 180, 200]
Saturation Flow Rate	[0.3, 0.38, 0.44]
Time Interval (recalculate critical flow	[600, 900, 1800]

ratio)	
Max-pressure Traffic Controller	
Green Duration	range(5, 26)

All controllers share the same yellow duration 3 seconds and all red duration 2 seconds.

5.6 Simulation Platform

5.6.1 Network

In accordance with Chapter 4, we utilized SUMO for the training and testing phases. The experiment was performed on a single intersection. The intersection comprises four legs, each of which is inclusive of three lanes: two straight movement lanes and one left-turn lane. The length of each lane is 656 feet, equivalent to approximately 200 meters. To reflect the local street environment, the design speed for all lanes is set at 40 mph. It is important to note that left-turn lanes only permit left turns, but not U-turns.

5.6.2 Demand

As outlined in Chapter 4, the SUMO software generates demand through the associated demand module, which is discussed in section 4.3.2. The origin-destination (OD) pattern is entirely random, meaning that there is no predetermined ratio between straight movement and left turn. The hourly distribution of the demand is modeled using an exponential function with a sine wave to generate a random distribution of the demand and create unpredictable traffic patterns for the simulations. Two types of total demand are used in the evaluation of the traffic signal controller's performance, with approximately 4000 and 6000 vehicles per 3-hour simulation. These demand levels are not so heavy that they cause the network to become entirely congested, allowing for the evaluation of any control optimization to be conducted effectively.

5.6.3 Measures of Effectiveness

The evaluation of the traffic signal controllers' performance is based on several MoEs, including the average travel time, standard deviation of travel time, queue length, and total system delay. Once the training is complete and an acceptable DQN is achieved, we will conduct 50 simulations, each lasting for 3 hours and using random seeds to ensure unbiased results. The average travel time is computed by dividing the total system travel time of all vehicles in the system by the number of vehicles. At the intersection level, we use queue length and total system delay as the MoEs. The queue length represents the total number of vehicles stopped in all incoming lanes of an intersection at a specific time step, while the total system delay is calculated by subtracting the free flow travel time from the sum of all vehicle travel times in the system at a particular time step.

5.6.4 Code

We utilized an existing framework, developed by Genders and Razavi (2019), to compare the performance of various traffic signal controllers, including Uniform, adaptive Webster's, Max-pressure, and our proposed DQN with experience replay and target network. Although the framework already included the code for these controllers, we had to develop our own code for the DQN with a different reward function. However, the original code had not been maintained for years and required significant effort to make it functional. To assist others in implementing their own machine learning and non-machine learning controllers, the source code used in this dissertation can be found in here.

5.7 Results

5.7.1 Hyperparameter Tuning Results

In Figure 1, the results of hyperparameter tuning are depicted for both learning and non-learning traffic controllers. Figure 2 presents a consolidated view of all the hyperparameter tuning results in a single figure.

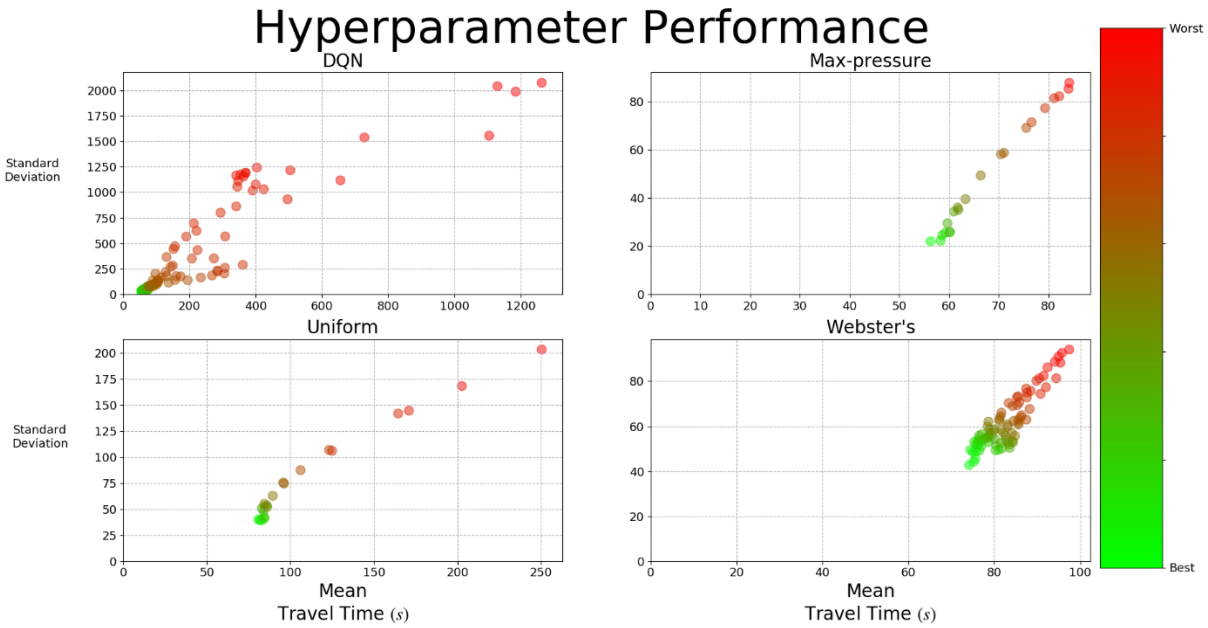

Figure 5.7-1. Hyperparameter tuning results for each controller

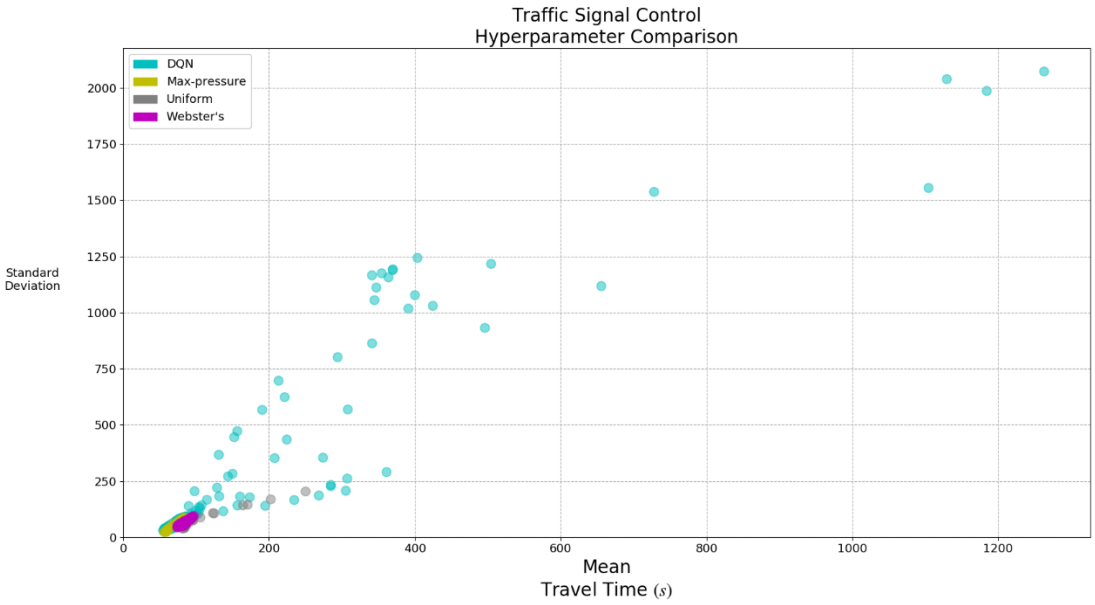

Figure 5.7-2. Hyperparameter tuning results for all controllers in one graph

All sorted hyperparameter tuning results are listed from Appendix 3 to Appendix 6.

It should be noted that the purpose of hyperparameter tuning is to enable the machine learning model to evaluate its initial performance using various combinations of model and hyperparameters. However, in the case of non-learning traffic controllers, the hyperparameter tuning process explores the performance of each hyperparameter configuration to identify the optimal performance for each controller with a specific parameter setting. As a result, our analysis of the hyperparameter tuning results primarily focuses on the DQN model.

The results indicate that DQN performance is significantly affected by the choice of hyperparameters. Therefore, it is highly recommended and necessary to perform hyperparameter tuning before training any machine learning model, as the performance of a learning model cannot be guaranteed by any specific combination of hyperparameters. This is true not only for more complex frameworks that include multiple hyperparameters but also for simpler ones like the reinforcement learning framework, Q-learning.

The presence of a grouping effect can be observed with respect to the learning rate and the discounting factor, where a discounting factor of 0.99, which is close to 1, results in poor performance, regardless of the configuration of other parameters. Additionally, the learning rate is an important hyperparameter that has a significant impact on the model performance. This finding confirms the conclusion of the study that the learning rate plays a crucial role in ensuring that the model converges at an appropriate speed.

Interestingly, the other hyperparameters do not exhibit significant differences across different settings. In the case of the number of hidden layers, it is unlikely that adding three additional hidden layers would be necessary to extract more relevant information and enhance

the model's performance in our single intersection scenario. Similarly, the TD step, which involves calculating one or two immediate rewards to update the DNN model, does not appear to have a significant impact. The same is true for the update frequency, which follows a similar trend.

In summary, our experiments have demonstrated that hyperparameter tuning is crucial for achieving optimal performance when using DQN, as it is highly sensitive to the choice of hyperparameters. Among the hyperparameters that we examined, the learning rate and discount factor were found to be the most important in terms of their impact on the model's performance.

After analyzing the hyperparameter tuning results, we selected the combination of hyperparameters that produced the best preliminary results in terms of the lowest average travel time and standard deviation of travel time for further training the DQN model. The table below provides an overview of all the parameters that were used in the DQN training process, as well as the hyperparameters for the non-learning traffic controllers.

5.7.2 Traffic Controller Performance Comparison

Based on the results of hyperparameter tuning, we established the value of each parameter for the extended training process of our DQN model. The hyperparameter tuning process involved 5000 episodes of 3-hour simulations, which had already demonstrated the agent's potential to outperform other non-learning traffic controllers in terms of average travel time and standard deviation of travel time, as shown in Figure 5.7-2. Consequently, we decided to initiate a new training process using the chosen hyperparameters, but with a larger number of episodes. One reason for this decision was our method of formulating the epsilon, which is the ratio of exploration and exploitation. Instead of continuing to train the best-performing model from the hyperparameter tuning process, we chose to start anew.

As previously stated, we gradually decreased the value of epsilon during the training process to enable the agent to explore more at the early stages and exploit more at the end of the process. By starting a new training process with a larger number of episodes, we can provide the agent with additional opportunities to explore and identify the most effective direction for improving its performance during training by adding more episodes.

Our DQN model underwent training for 20,000 episodes, which equates to 20,000 3-hour simulations based on the given demand of approximately 6,000 vehicles in the single intersection. Figure 5.7-3 depicts the system-level performance of all controllers in terms of average vehicle travel time, mean vehicle travel time, and standard deviation of vehicle travel time.

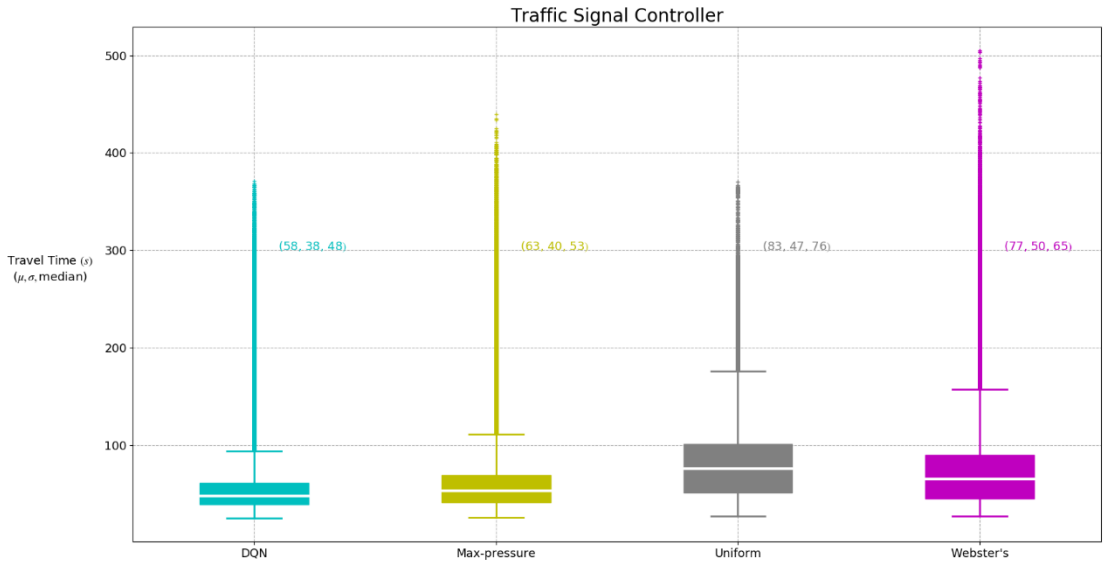

Figure 5.7-3. System Level Performance Comparison of DQN and other Non-learning Controllers with 6,000 Demand

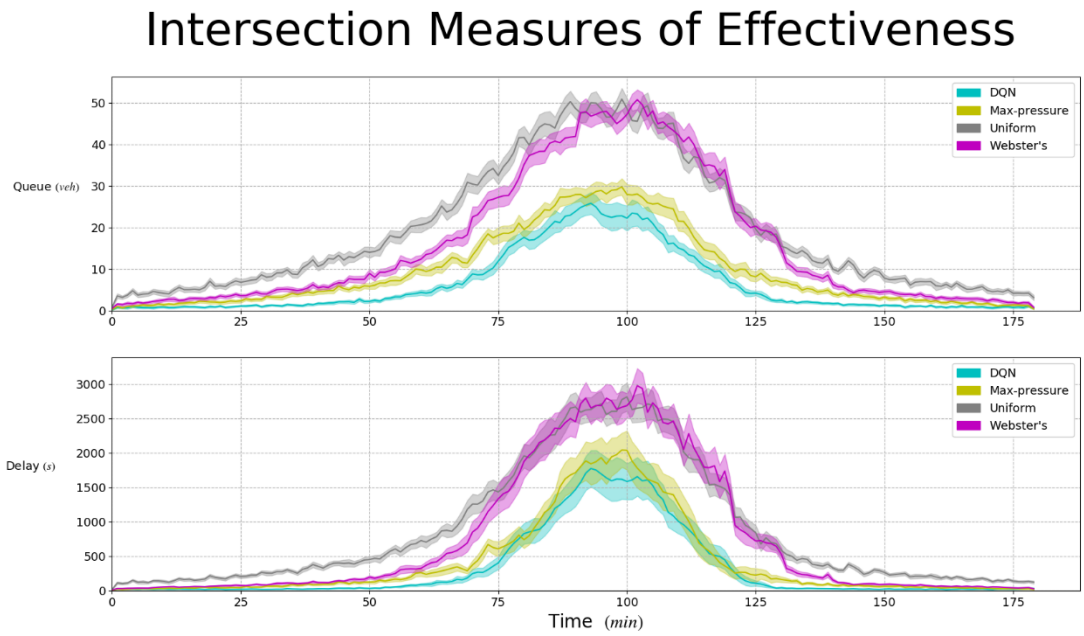

Figure 5.7-4. Intersection Level Performance Comparison of DQN and other Non-learning Controllers with 6,000 Demand

Figure 5.7-4 depicts the intersection-level results, encompassing vehicle queue length and vehicle delays. In a single simulation, we should have 10,800 results for each time step. To smooth out the lines, we calculated the average values every minute, reducing the data to 180.

For each traffic controller, we conducted 32 simulations, resulting in 32 different outcomes. In Figure 5.7-4, the solid line corresponds to the mean value obtained from these 32 results, while the shaded area indicates the 95% confidence interval with alpha = 0.05. During this stage of the analysis, the DQN model does not explore the environment further. Instead, it chooses the phase that maximizes the reward at each time step.

For the intersection-level results, the x-axis represents the simulation time step, which is 180 minutes (3 hours), while the y-axis shows the queue length measured by the number of vehicles in the intersection by summing all stopped vehicles from its incoming lanes in the top graph and total delay of all vehicles in the lower graph.

Based on the results at both the system level and intersection level, our DQN model outperforms all other considered controllers in terms of the chosen MoEs, with Max-pressure coming in second and the uniform traffic controller performing the worst, which is not surprising since we used fully random demand in the experiment. It is expected and well-known that adaptive controllers are more efficient.

It should be noted that machine learning models are often unable to provide a clear explanation for why they perform better than non-learning controllers. This is a limitation of machine learning models, and more research is needed to make their decision-making process more explicit. However, one observation that can be made is the pattern of phase selection by the DQN. Figure 5.7-5 depicts the percentage of frequency each phase is selected in one simulation for the DQN controller, while Figure 5.7-6 shows the same for the Max-pressure. It is evident that the DQN seldom chooses the left-turn movement, accounting for only 5% of the total for the left-turn phases. It is plausible that the DQN recognizes that the straight movement phases have unprotected left-turn green light for those vehicles and that choosing the straight movement with unprotected left-turn phases is more efficient.

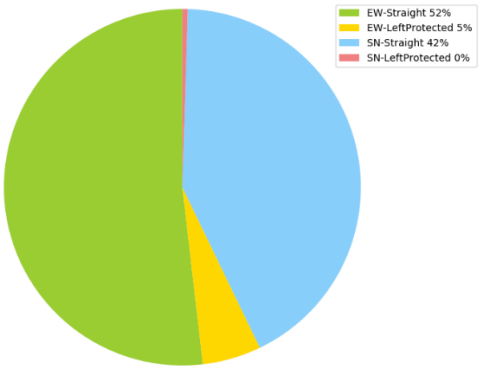

Figure 5.7-5. Frequency of phase selection in one simulation for DQN controller with 6,000 Demand

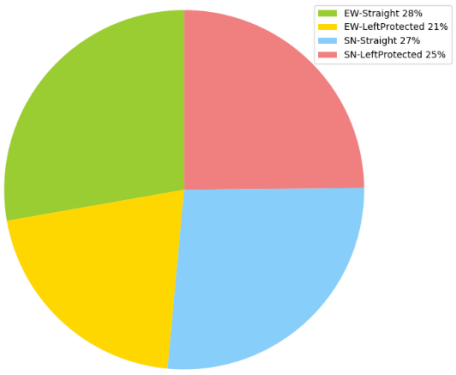

Figure 5.7-6. Frequency of phase selection in one simulation for Max-pressure controller with 6,000 Demand

In addition to presenting the percentage of Max-pressure phases, we include the phase with the highest pressure based on queue length, as it is also aperiodic. In our simulations, traffic demand is random and sometimes results in more left-turning vehicles than those traveling straight, leading to more frequent activation of left-turn phases compared to the DQN controller, as indicated in Figure 5.7-5 and Figure 5.7-6. This demonstrates the benefit of the DQN approach, which receives similar inputs but has learned a policy that utilizes unprotected left-turn green allocations for left-turn traffic instead of resorting to protected left-turn phases that introduce additional delays to the system.

There is another possible reason related to the definition of the state. As we included the vehicle density and queue length for each lane, the agent may have learned to differentiate between them and create predictions to extend certain phases in order to reduce system loss time. It is also possible that the machine learning controller finds something that has not been found by the most smart human beings.

We also tested the model with lower traffic demand (4,000) without hyperparameter tuning and used the same hyperparameter settings from the previous experiment, which also resulted in the best performance compared to non-learning traffic controllers, as shown in Figure

5.7-7 and

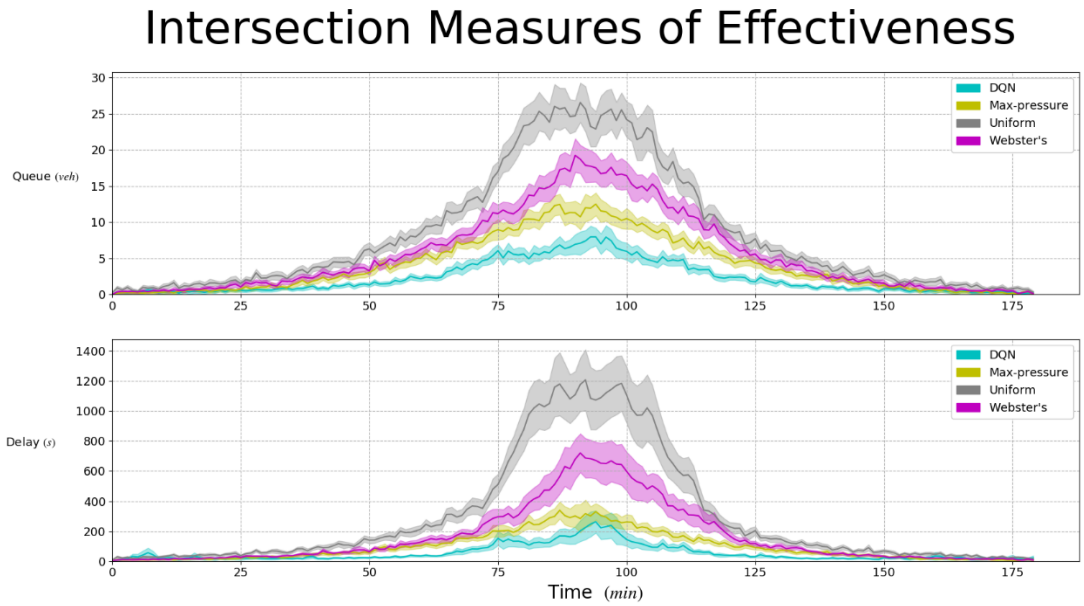

Figure 5.7-8. We did not initiate another round of training but utilized the results from the traffic demand of 6,000 to this lower demand scenario.

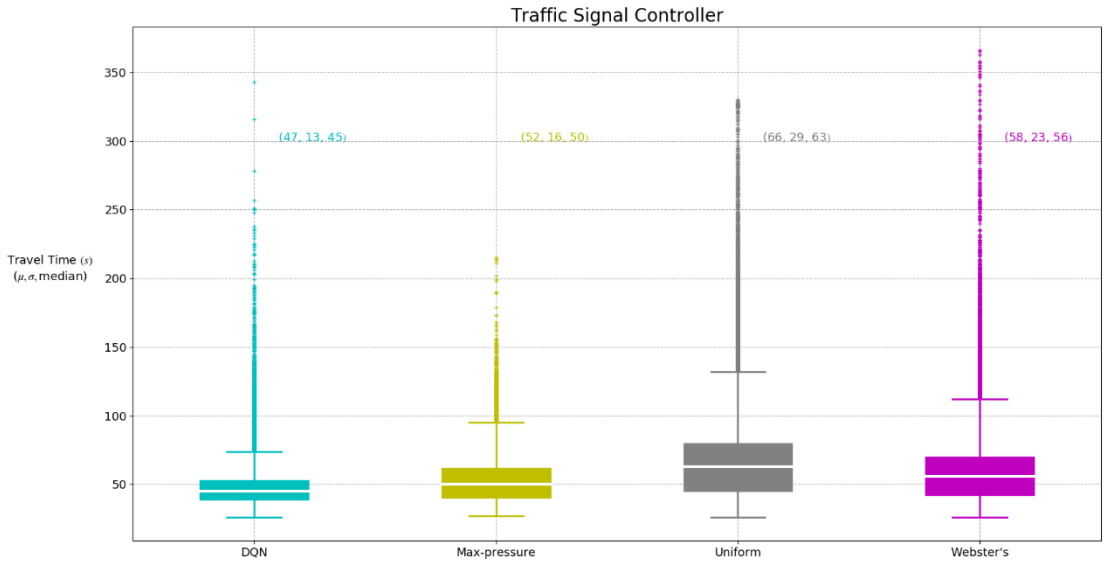

Figure 5.7-7. System Level Performance Comparison of DQN and other Non-learning Controllers with 4,000 Demand

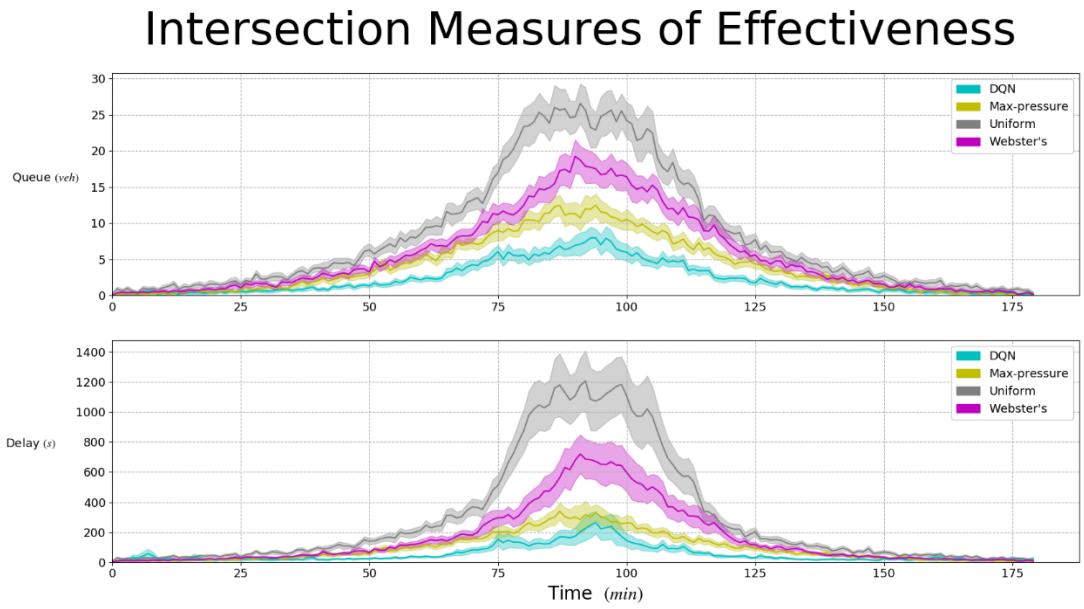

Figure 5.7-8. Intersection Level Performance Comparison of DQN and Non-learning Controllers with 4,000 Demand

It is reasonable that the previously trained DQN model could perform well with lower traffic demand since it had already encountered similar conditions during the training process and learned to generalize its performance. This result highlights the potential of machine learning algorithms, as they cannot only learn from higher traffic demand scenarios but also generalize their performance to lower traffic demand scenarios, outperforming non-learning traffic controllers.

5.8 Conclusion

To ensure the optimal performance of our proposed DQN model, we conducted experiments on various hyperparameter settings. This approach sets our research apart from others, as most of them do not perform hyperparameter tuning before the actual training process. The hyperparameter tuning process not only helped us optimize the non-learning traffic controllers, but also laid a strong foundation for the actual DQN training process.

Based on the findings of the hyperparameter tuning, we discovered that the performance of our proposed DQN is highly influenced by the configuration of its hyperparameters. Specifically, the learning rate and discount factor were identified as the most critical hyperparameters in our single intersection scenario, while other involved hyperparameters appear to be less significant.

Our proposed DQN, equipped with experience replay, target network, and optimized hyperparameters, has been shown through simulation experiments to provide the best performance in terms of MoEs such as average travel time, queue length, and vehicle delays. Moreover, the model is capable of generalizing well to lower traffic demand scenarios, thanks to its training process with higher traffic demand.

Although coding and mathematical knowledge of the reinforcement learning framework are required, transportation knowledge is only minimally necessary to understand DQN. In addition to the traffic controller's principles, traditional complex traffic flow models do not have to be explicitly formulated since the AI can learn them during the training process. This advantage allows anyone interested in traffic signal controller algorithms to explore better machine learning control algorithms without extensive knowledge of the transportation domain. This accelerates the development of better control method performance.

Chapter 6. Grid Network Deep Reinforcement Learning Traffic Signal Control with Incidents

6.1 Overview

Traffic incidents within transportation networks not only raise safety concerns for travelers but also cause significant delays in the system. Traffic Incident Management (TIM) was established to mitigate the adverse effects of such incidents by promptly resolving them and restoring transport infrastructure services. While attempts have been made to use traffic signal coordination to lessen the impact of incidents, the infrequency of these events and the complexity of modeling the relationship between incidents and signal configurations have made it difficult to implement. Consequently, devising a practical solution to adjust traffic controllers and minimize the impact of incidents remains a challenge.

The aim of this study is to explore the application of machine learning techniques to reduce the consequences of traffic incidents, building upon prior research that demonstrated the superiority of DQN with hyperparameter tuning over conventional traffic control techniques such as Uniform, Webster's, and Max-pressure. This approach circumvents the need for explicit modeling of traffic flow and its relationship with signal plan configurations. Our research includes an incident generation module, as described in Chapter 4, which creates random incidents within the network to offer learning opportunities for the AI controller.

We will evaluate the performance of the DQN in the context of traffic incidents by conducting experiments using two distinct network configurations: a two-intersection corridor and a 2x2 grid network, both featuring incident occurrences. Moreover, we will analyze two separate traffic demand scenarios with varying total vehicle counts to confirm the effectiveness of the DQN. To improve the AI controller's capacity to handle traffic incidents, we introduce a new state definition for the DQN.

6.2 Literature Review

Traffic incident management is a critical aspect of transportation system operations. It involves the coordination of multiple agencies to promptly detect, respond to, and clear incidents on road networks to minimize congestion, reduce secondary crashes, and improve overall transportation efficiency. Traffic signal control strategies are essential tools that can be leveraged to facilitate better traffic incident management. This literature review provides an overview of the key research conducted on traffic incident management with a focus on traffic signal plan adjustments.

Carson et al. (2010) conducted a comprehensive review of traffic incident management practices in the United States. The author collected data from various sources, including federal and state departments, transportation agencies, and emergency response organizations. The study identifies best practices across different aspects of traffic incident management, such as incident detection, response, clearance, and communication. The study concludes that adopting best practices in traffic incident management can significantly improve the efficiency and effectiveness of incident response, reduce congestion, and enhance safety for both responders and road users. The report highlights the importance of inter-agency collaboration, real-time

communication, and standardized protocols in achieving better traffic incident management outcomes. Additionally, it emphasizes the need for continuous training, performance measurement, and improvement to ensure the successful implementation of best practices.

Goodall et al. (2013) conducted a qualitative analysis of traffic incident management practices across multiple agencies. Interviews and surveys were used to collect data on collaboration and communication among agencies involved in traffic incident management. The study concludes that effective inter-agency collaboration and communication are crucial to successful traffic incident management. It recommends the development of integrated systems and standardized procedures to facilitate better collaboration among agencies.

These studies show performance improvement of transportation systems can be gained by adjusting the traffic signal control plan during traffic incidents in the network. One of many difficulties is how to provide a solid and reasonable strategy to adjust the traffic signal plan accordingly. Gartner et al. (2001) reviewed existing Traffic-Responsive Plan Selection (TRPS) systems and their methodologies. It also conducted a comparative analysis of their performance in various traffic scenarios. The study finds that TRPS systems can significantly improve traffic signal control during incidents by adapting signal timings to real-time traffic conditions. It highlights the potential of TRPS in reducing congestion and travel times during incidents.

Traditional research uses model-based methods and microsimulation to investigate the proposed signal control plan optimization strategies. Mirchandani and Head (2001) reviewed model-based traffic signal control strategies, focusing on multiple-objective optimization approaches. The authors explore various algorithms and their applicability in incident management scenarios. The study concludes that model-based traffic signal control strategies can effectively address multiple objectives, such as minimizing delays and maximizing throughput, during incidents. It recommends further research on the development and evaluation of these strategies in real-world traffic scenarios.

Aboudolas et al. (2010) explores the use of Model Predictive Control (MPC) as an adaptive traffic signal control strategy. It develops a simulation model to evaluate the performance of MPC in various traffic scenarios, including incidents. The study finds that MPC can effectively adjust traffic signal timings during incidents, leading to reduced delays and improved traffic flow. It recommends further research on the development and evaluation of MPC in real-world traffic scenarios.

One significant limitation of using model-based methods to find the optimal traffic signal control plan for reducing the impact of network traffic incidents is the dependenence on the model accuracy . Due to the complexity of the transportation network system, this is too hard to be practical.

Current research has begun to use machine learning methods to conquer the limitation of traditional model-based methods of traffic signal adjustment. Hadiuzzaman et al. (2012) proposed a methodology for adjusting traffic signal timings during incidents using Artificial Neural Networks (ANNs). It develops an ANN-based model and evaluates its performance in terms of reducing delays and congestion in simulated traffic scenarios. The study concludes that the proposed ANN-based methodology can effectively adjust traffic signal timings during

incidents, leading to reduced delays and improved traffic flow. It recommends further research on the development and evaluation of ANN-based traffic signal control strategies in real-world traffic scenarios. However, the research did not explain the hyperparameters decision which makes the research hard to duplicate and the compared signal controllers do not include more advanced controllers such as Max-pressure.

This paper will fill the gap by investigating the impact of several key hyperparameters in the deep reinforcement learnig, especially Deep Q-Network (DQN), to optimize traffic signal control algorithms with traffic incident occurance. This will provide practical guidance for researchers and enable implementations of deep reinforcement learning signal control algorithms to reduce impacts of incidents in transportation networks.

6.3 Incident Generation

We have developed an incident generation module within the open-source microsimulation platform, SUMO, to expose the machine learning controller to situations involving traffic incidents within the network. This approach creates relevant experiences for the AI controller to learn from and enhance its decision-making process regarding traffic phase selection. To the best of our knowledge, this is the first instance of incorporating the incident concept into SUMO.

In our simulation, we consider both single-vehicle and multiple-vehicle incidents. We represent the number of vehicles involved by using a single vehicle in SUMO with varying lengths, assuming each vehicle measures 5 meters in length with an additional 2.5-meter gap between stopped vehicles. For example, a two-vehicle incident would occupy 15 meters of lane space.

The incident generation module randomly selects a lane connecting two intersections to emulate the coordination impact between traffic controllers. In our scenario, each identical intersection features two straight movement lanes and one left-turn lane. We present two potential incident locations: on the straight movement lane or the left-turn lane. To simplify the learning process and minimize the risk of gridlock, we restrict incident locations to the straight movement lanes.

The incident vehicle's route is generated randomly, adhering to the requirement that it passes through at least two intersections. This ensures that the incident affects multiple intersections rather than just one. Our simulation schedules the incident randomly during the second hour of the three-hour simulation period, allowing most vehicles to complete their trips. Incident durations are assumed to be either 15 or 30 minutes.

Additionally, we incorporate emergency service vehicles into the incident generation module to simulate the rescue process impact. Representing an abstraction of multiple service vehicles, the emergency vehicle varies in length from 22.5 to 45 meters. It is generated 5 minutes after an incident is detected and travels from a random origin, stopping next to the incident location until the incident vehicle moves.

Under these conditions, traffic flow is significantly affected by the incident, allowing us to observe the intersection controllers' responses. The Uniform traffic controller maintains its

fixed pattern and green phase duration, offering no response to the incident. In contrast, the Webster's traffic controller adjusts its phases to accommodate the new traffic pattern by either reducing or extending the current phase. The Max-pressure and DQN traffic controllers, with their acyclic phase selection capabilities, should theoretically perform better in such scenarios, as they can choose suitable phases in any given situation.

6.4 New State

Transportation networks can be significantly disrupted by the occurrence of incidents. One consequence is that vehicles behind the incident point may become stuck, regardless of the amount of green time allocated. Adaptive traffic controllers struggle to account for this feature to enhance their performance. To address this issue, we introduce a new state for the proposed DQN to further improve its capabilities.

The new state is defined as the queue that could potentially be reduced by allocating green time and monitoring vehicles that have not been able to move after experiencing green phases. This approach ensures that the queue information passed to the DQN model is more accurate. We apply this new state only to the DQN, as we have established in the previous chapter that it outperforms other traffic controllers in single intersection scenarios. In this chapter, our goal is to determine the extent of the DQN's performance improvement in situations involving incidents and the application of the new state definition.

To implement the new state collection, information on each individual vehicle's location and the most recent phase is required. If a vehicle's location has not changed compared to the previous time step, and the vehicle has already experienced a green phase for its traveling direction, we can deduce that the vehicle is stuck in the system and will be removed from the queue calculation.

The DQN model employed in this chapter maintains the same structure as the one used in the previous chapter, including its action and reward system, as well as the incorporation of experience replay and target network. The DNN structure also remains similar, utilizing the ReLU activation function and fully connected hidden layers. However, the primary distinction lies in the increased number of hidden layers used in this chapter. This is due to the heightened complexity of corridor and grid networks with traffic incidents, necessitating a more detailed analysis of the relationship between action choices and environmental inputs.

6.5 Simulation Settings

To demonstrate the performance of the DQN, we compare it with three other traffic signal controllers: Uniform, Webster's, and Max-pressure. Definitions and implementation details for each traffic signal controller can be found in the previous chapter, which focuses on single intersection scenarios. It is important to note that the new state definition is applied to both the DQN and Max-pressure controllers, as they both depend on queue information to adjust their phase choices. The simulation spans a three-hour period and is conducted using SUMO.

6.5.1 Network

We employ two network configurations to assess the performance of non-learning traffic controllers and the DQN in scenarios involving incidents: a two-intersection corridor and a 2x2 grid network, illustrated in Figure 6.5-1 and Figure 6.5-2, respectively. Each intersection in these network configurations is identical to the single intersection examined in the previous chapter.

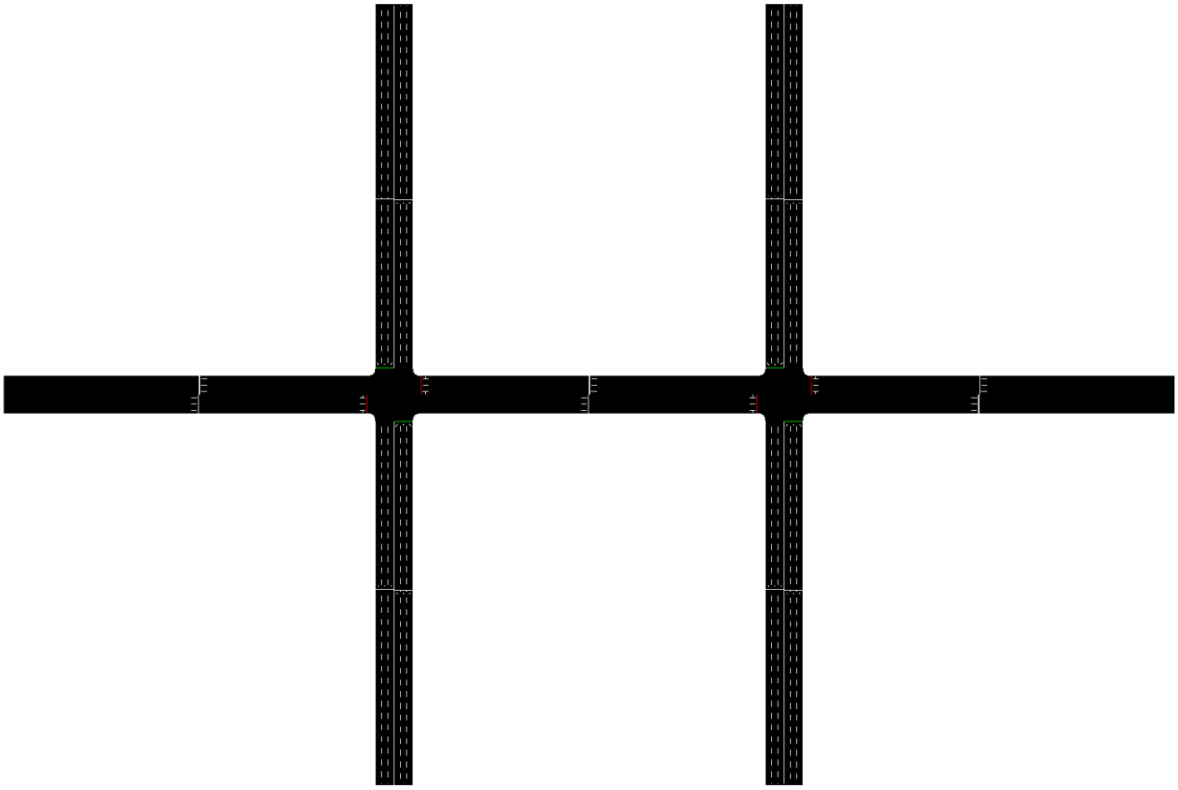

Figure 6.5-1. Corridor with two intersections

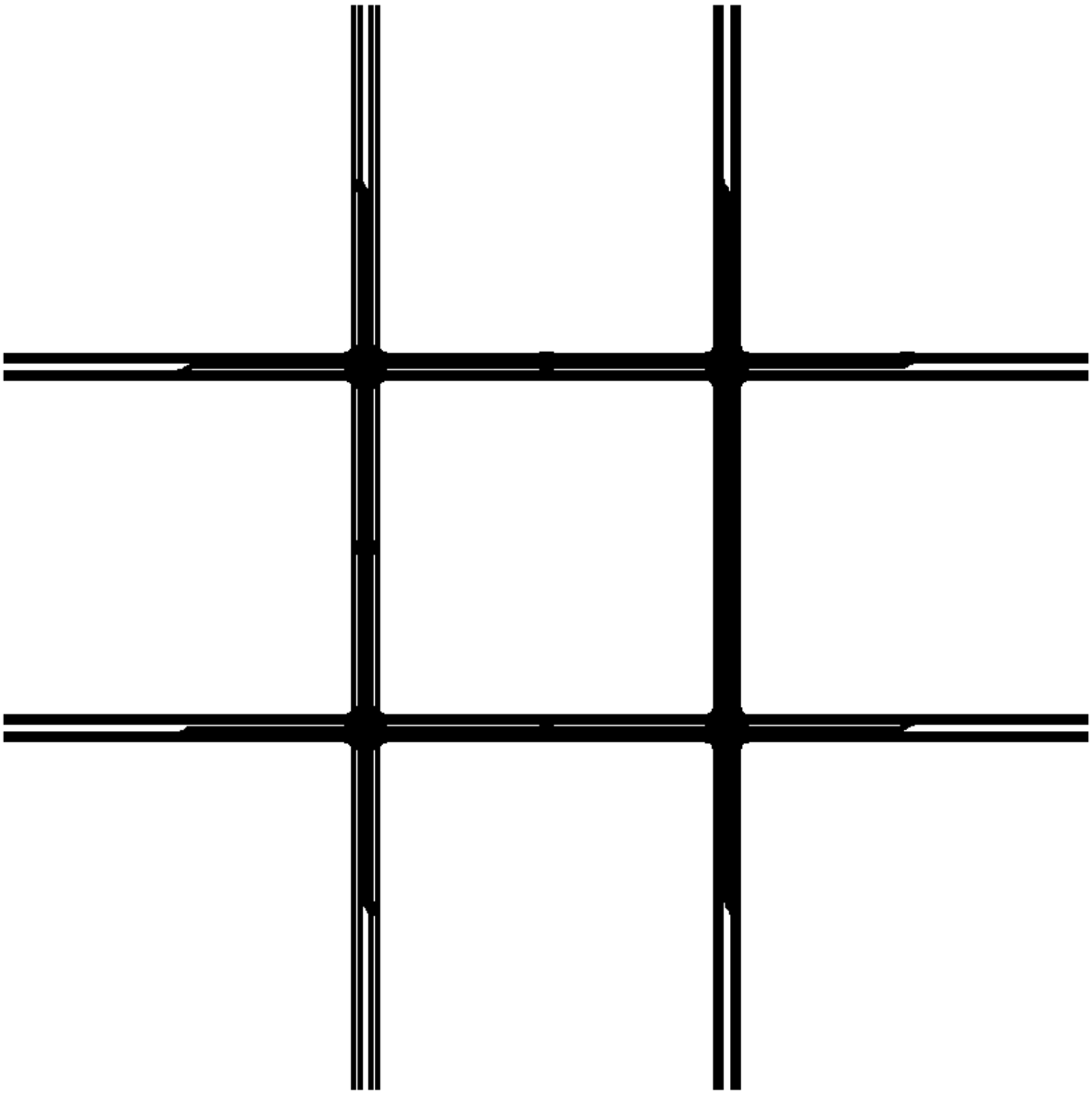

Figure 6.5-2. 2x2 Grid Network

6.5.2 Demand

In the simulations, we will employ two distinct traffic demands, consisting of 4,000 and 6,000 vehicles, respectively. Although these demands are not overly heavy for the two networks in the absence of incidents, the network becomes congested when an incident takes place, presenting an opportunity for traffic controllers to adjust their phase selection and enhance network performance.

The presence of an incident in the network can lead to longer queues at one or more intersections compared to a situation without incidents. This can significantly diminish the network's performance, which can be improved by employing adaptive traffic controllers such as Webster's, Max-pressure, and a trained DQN.

6.6 Hyperparameter Tuning

Drawing on our experience with hyperparameter tuning in the single intersection scenario, we have determined that the most crucial hyperparameters to adjust are the learning rate and discount factor. To minimize computing time, we have limited the hyperparameter tuning list to three values for each parameter. For the remaining parameters, we will use the hyperparameters obtained from the single intersection tuning. Furthermore, we have increased the number of hidden layers in the DNN from 3 to 6 to augment its capacity to capture more complex features from the inputs, thereby improving the performance of the DQN controller. Table 6.6-1 presents the parameters employed during the training process for the corridor and grid networks, as well as other non-tuning parameters used in the DQN.

Table 6.6-1. Parameters used in DQN controller for the corridor and grid network

Parameters	Value/Values
Learning Rate	[0.0001, 0.00001, 0.001]
Discount Factor	[0.5, 0.7, 0.9]
TD Step	2
Number of Hidden Layers	6
Target Frequency	128
Green Duration	6
Episodes	5000
Replay Buffer Size	40000
Batch Size	128
Number of Nodes Per Hidden Layer	64
Activation Function	ReLU

6.7 Results

6.7.1 Hyperparameter Tuning Results

Hyperparameter tuning is carried out for both the corridor and grid (2x2 intersections) networks with a higher traffic demand of 6,000 vehicles. Figure 6.7-1 and Figure 6.7-2 consolidate the results of hyperparameter tuning for each controller into a single figure to facilitate a better understanding of each controller's performance after hyperparameter tuning, where Figure 6.7-3 and Figure 6.7-4 show the combined results. Appendix 11 to 14 provide a comprehensive list of hyperparameter tuning results for all four controllers.

It is important to note that the DQN's performance is not finalized, as we still need to train the model instead of directly applying the preliminary results from hyperparameter tuning. In contrast, the performance of the other three controllers is determined due to their non-learning properties. The rationale behind using the incident scenario to train the DQN is to expose the controller to experiences involving traffic interruptions caused by incidents, thereby enabling it to better adapt its actions for improved performance. Once the DQN has learned from both incident and non-incident periods, it may develop a model capable of handling both situations by adjusting its DNN parameters.

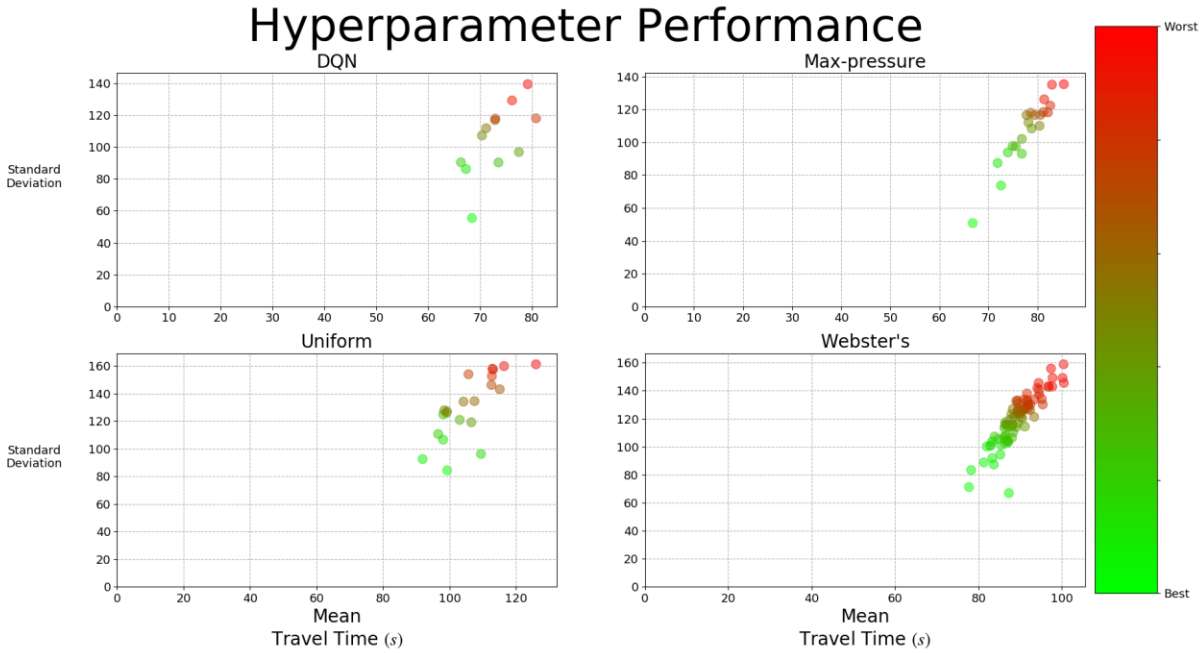

Figure 6.7-1. Hyperparameter tuning results for the corridor network with 6,000 vehicle demand and incident (Separate Graph)

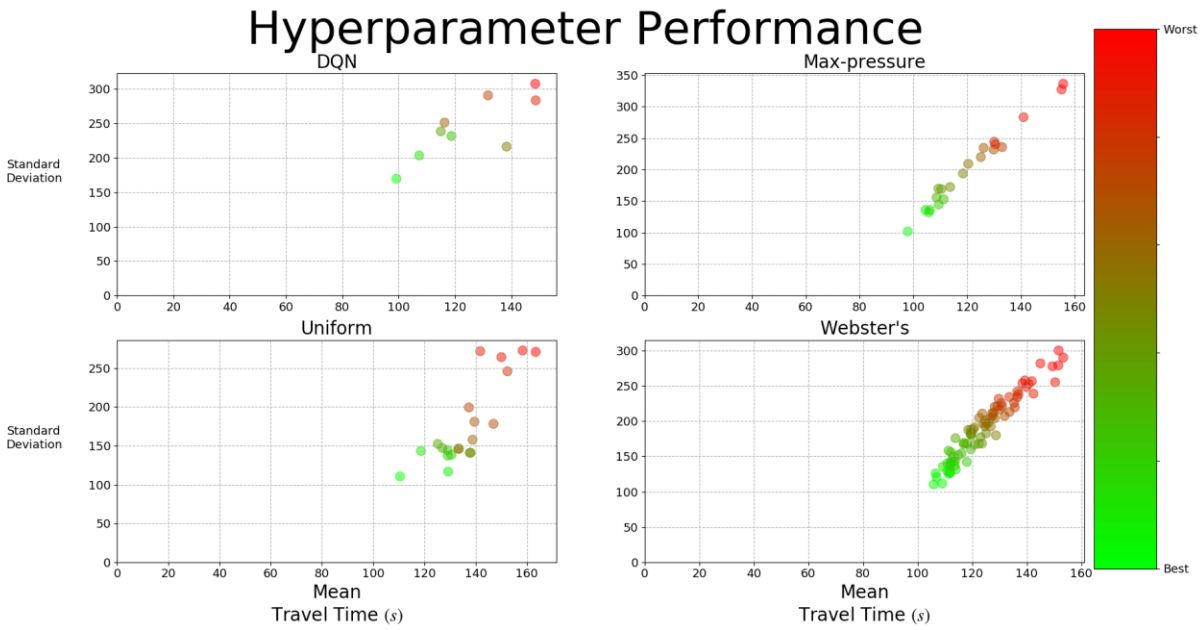

Figure 6.7-2. Hyperparameter tuning results for the 2x2 grid network with 6,000 vehicle demand and incident (Separate Graph)

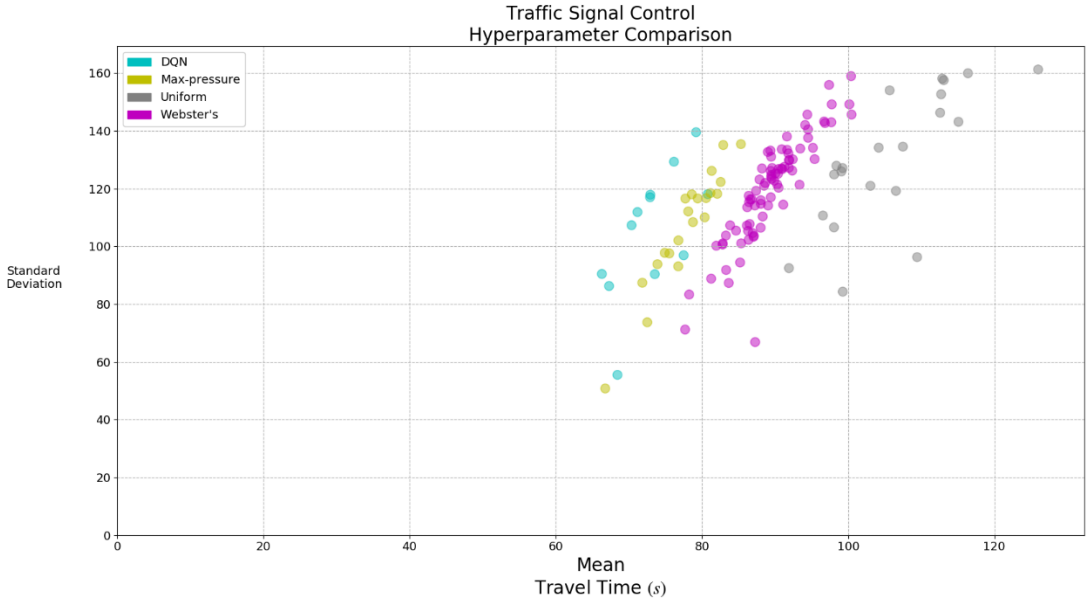

Figure 6.7-3. Hyperparameter tuning results for the corridor network with 6,000 vehicle demand and incident (Combined Graph)

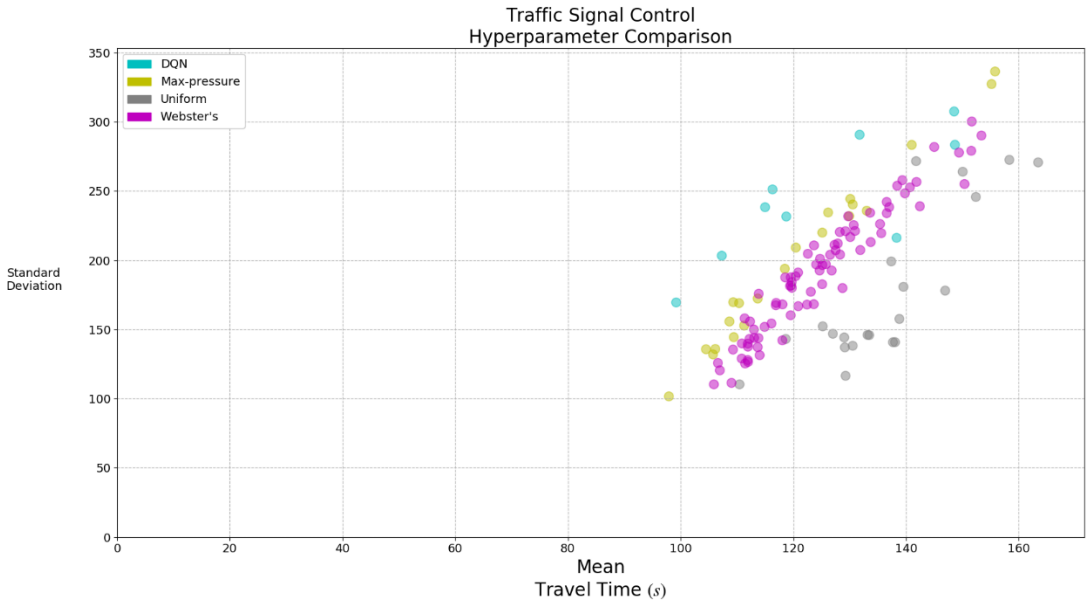

Figure 6.7-4. Hyperparameter tuning results for the 2x2 grid network with 6,000 vehicle demand and incident (Combined Graph)

It is noticeable that the performance of the DQN across different hyperparameter combination settings is not as varied as seen in the single intersection hyperparameter tuning results. This is because we have already narrowed down the potential values to be included in the hyperparameter tuning process. Another observation is that, with proper hyperparameter settings, the performance of the best DQN model can be on par with Max-pressure even without being trained extensively.

To analyze the fact that traditional non-learning controllers struggle to handle varying traffic situations, such as networks with or without incidents even for the same traffic demand, we also conducted hyperparameter tuning for the 2x2 grid network with a 6,000-vehicle demand and no incidents in the network. Figure 6.7-5 and Figure 6.7-6 display the hyperparameter tuning results, while Appendix 15 to 18 provide more detailed information on the results.

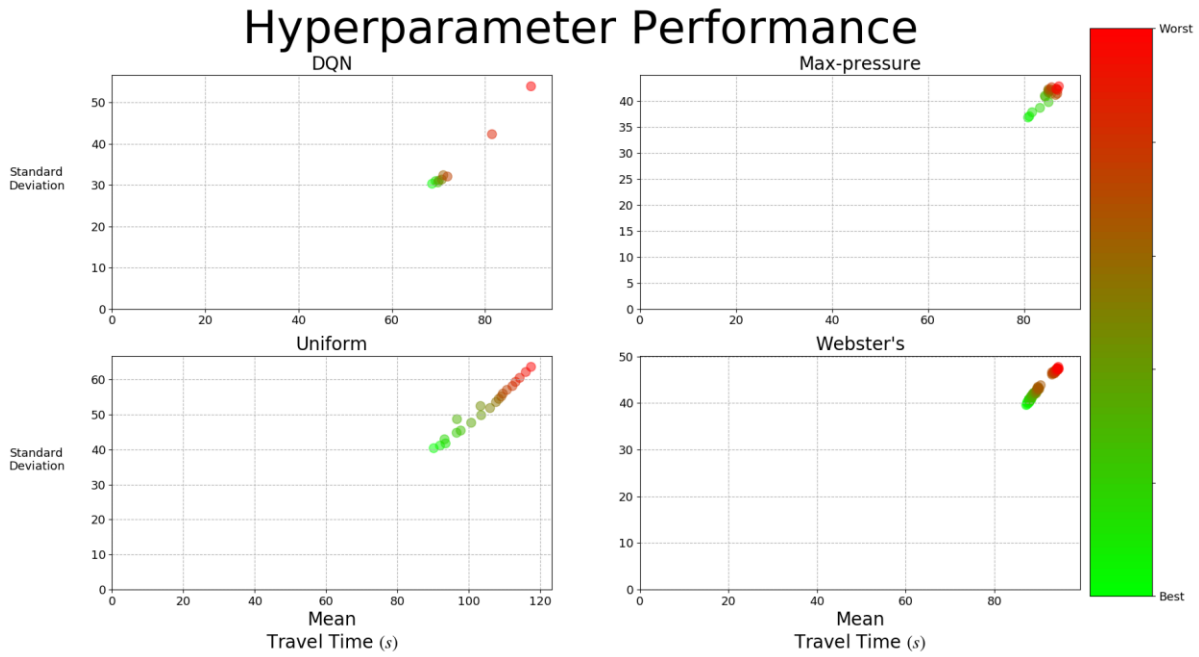

Figure 6.7-5. Hyperparameter tuning results for the 2x2 grid network with 6,000 vehicle demand and no incident (Separate Graph)

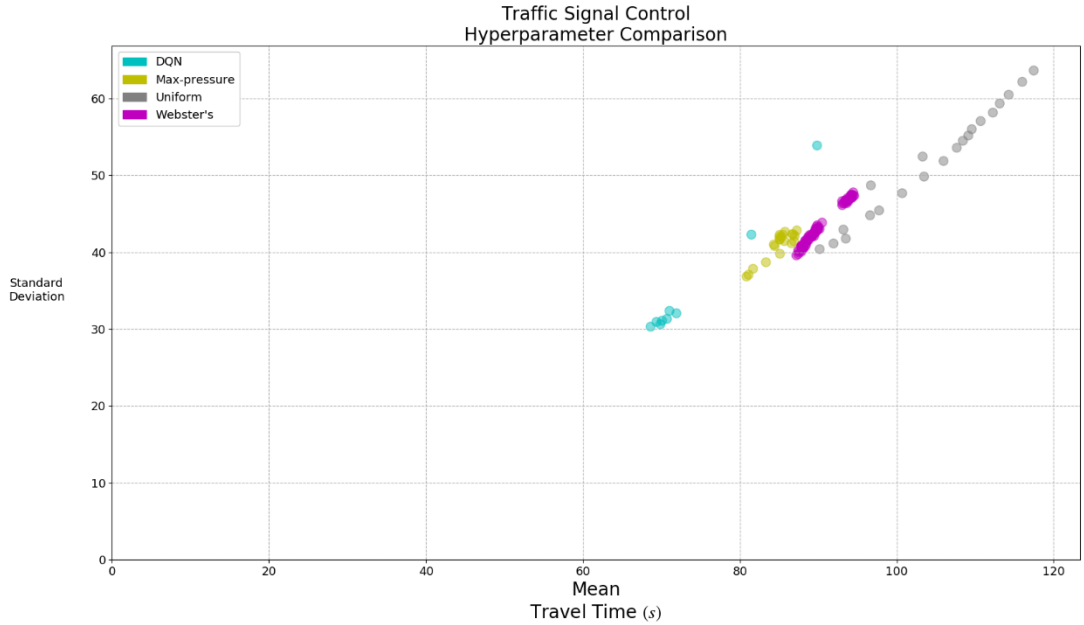

Figure 6.7-6. Hyperparameter tuning results for the 2x2 grid network with 6,000 vehicle demand and no incident (Combined Graph)

By comparing the best hyperparameter settings from all controllers, we can see that the learning model employs almost the same settings, with the only difference being the learning rate. However, the second-best results use the same settings in both incident and non-incident scenarios for the DQN. In contrast, non-learning controllers utilize entirely different settings to compensate for the varying traffic patterns in the two scenarios.

6.7.2 Controller Performance Comparison

Using the best hyperparameter tuning settings for all controllers, we can compare their performance at both the system and intersection levels. For the learning model, DQN, we need to train it to achieve convergence. However, for non-learning controllers, we can simply use the best hyperparameters to generate the results. Each controller will be simulated 32 times to obtain a range of results, overcoming the randomness effect of using just a single simulation to verify its performance. Figure 6.7-7 displays the system-level results, including the mean travel time of all vehicles in the network with incidents in the corridor network, while Figure 6.7-8 illustrates the intersection-level performance.

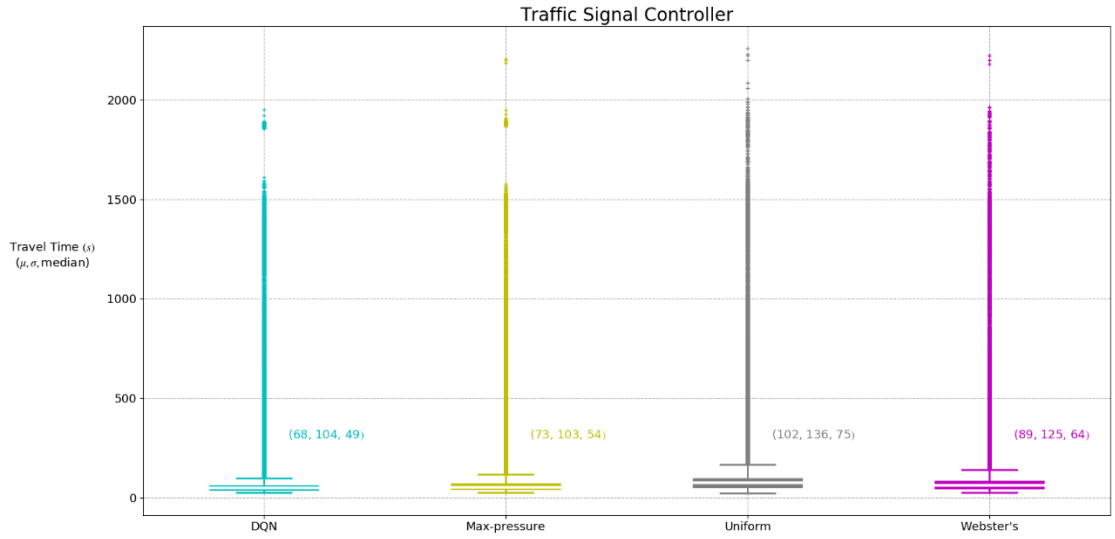

Figure 6.7-7. System Level Performance Comparison of DQN and other Non-learning Controllers with 6,000 Demand and Incident in Corridor Network

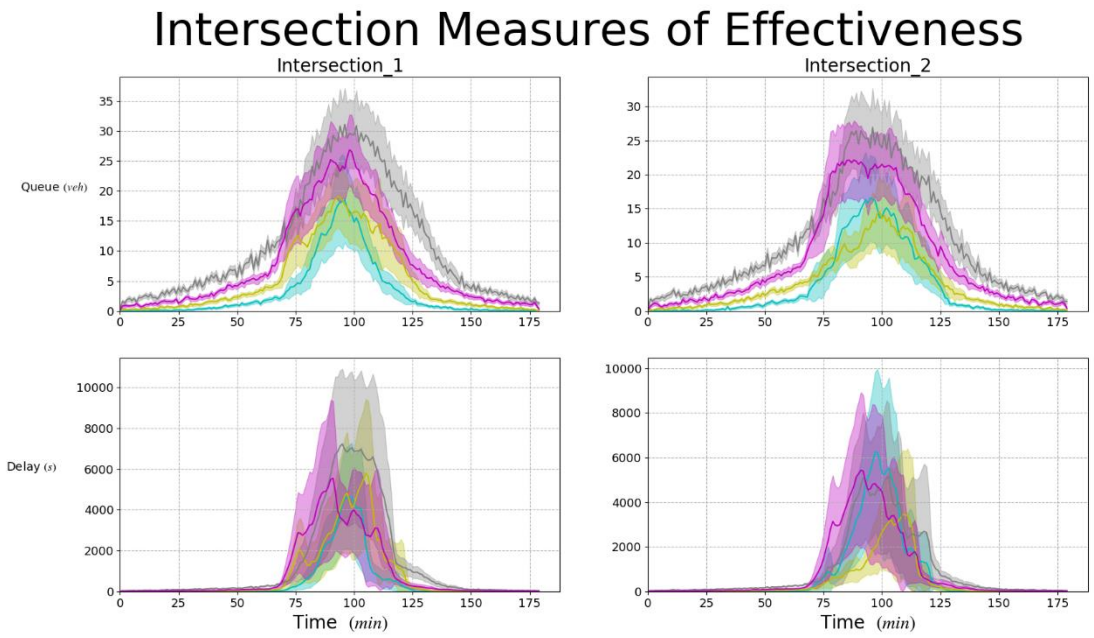

Figure 6.7-8. Intersection Level Performance Comparison of DQN and other Non-learning Controllers with 6,000 Demand and Incident in Corridor Network

Figure 6.7-9 and Figure 6.7-10 depict the same performance measurement for the 2x2 grid network but with the presence of incidents.

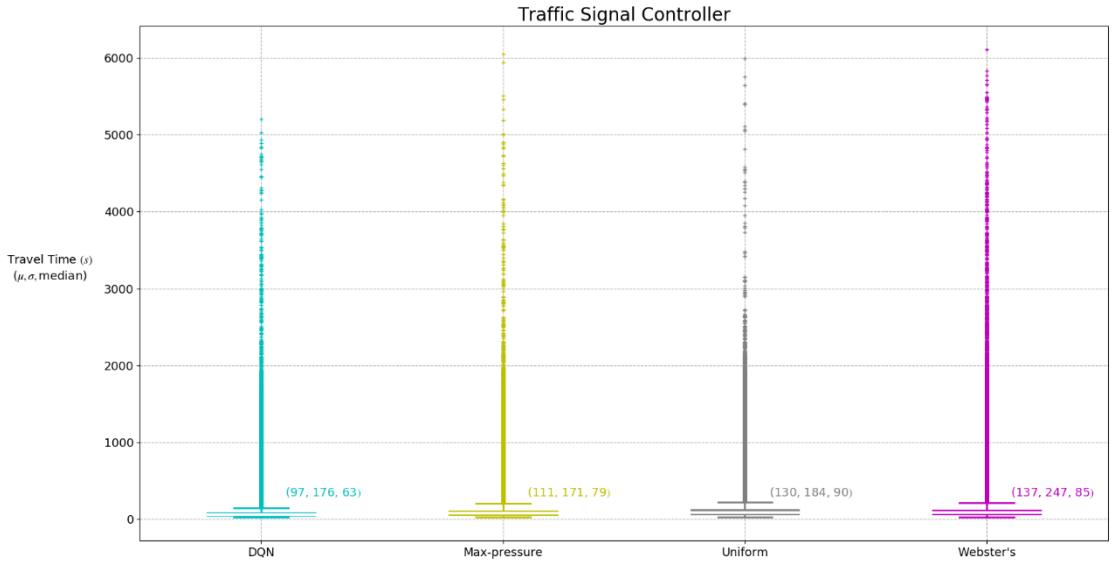

Figure 6.7-9. System Level Performance Comparison of DQN and other Non-learning Controllers with 6,000 Demand and Incident in 2x2 Grid Network

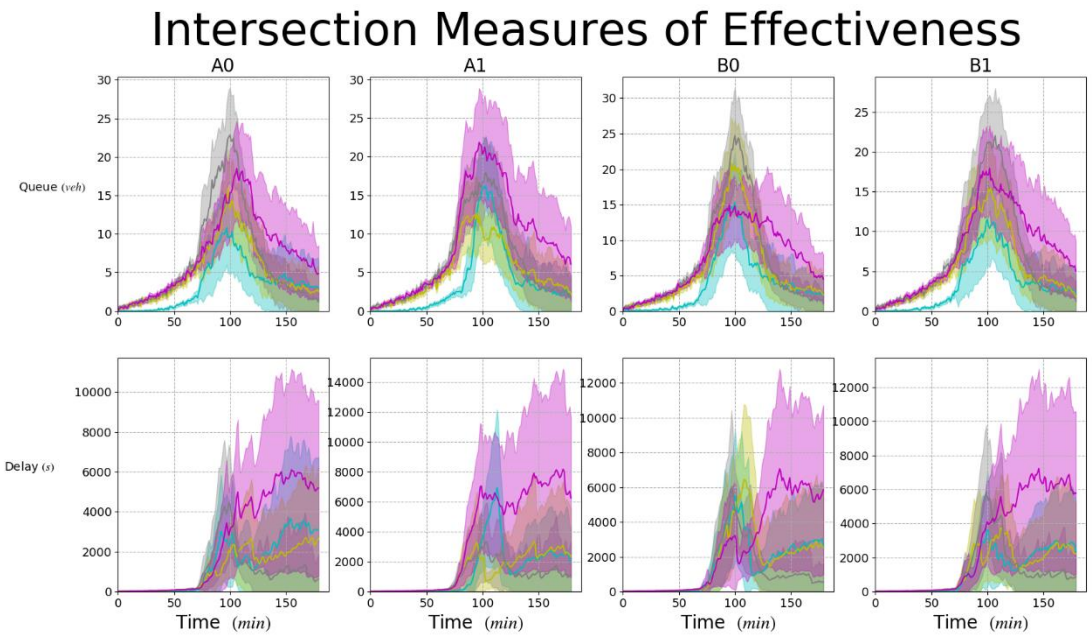

Figure 6.7-10. Intersection Level Performance Comparison of DQN and other Non-learning Controllers with 6,000 Demand and Incident in 2x2 Grid Network

In contrast to the corridor scenario, the incident causes a significant amount of delay in the grid network, which can be seen by the end period of the simulation, where the queue length does not return to 0 due to unfinished vehicles. This is expected, as we did not allocate extra time for the simulation but only allowed a 3-hour simulation for all situations.

Incidents in both networks cause the controller's performance to vary significantly, but the mean travel time clearly shows that the fine-tuned and trained DQN outperforms other controllers with the lowest mean travel time and the lowest standard deviation of mean travel time.

We also applied the model to the same demand without incidents in the network to see if the DQN controller can handle the situation for both incident and non-incident networks, even with the same training model. The logic behind this is that during the training process, the DQN controller also experiences the time when there is no incident in the network, as we only introduce the incident to the network for a certain amount of time out of the 3-hour simulation period.

Figure 6.7-11 to Figure 6.7-14 show the results when applying the incident model directly to the non-incident scenario with the same traffic demand for both networks, including corridor and grid networks.

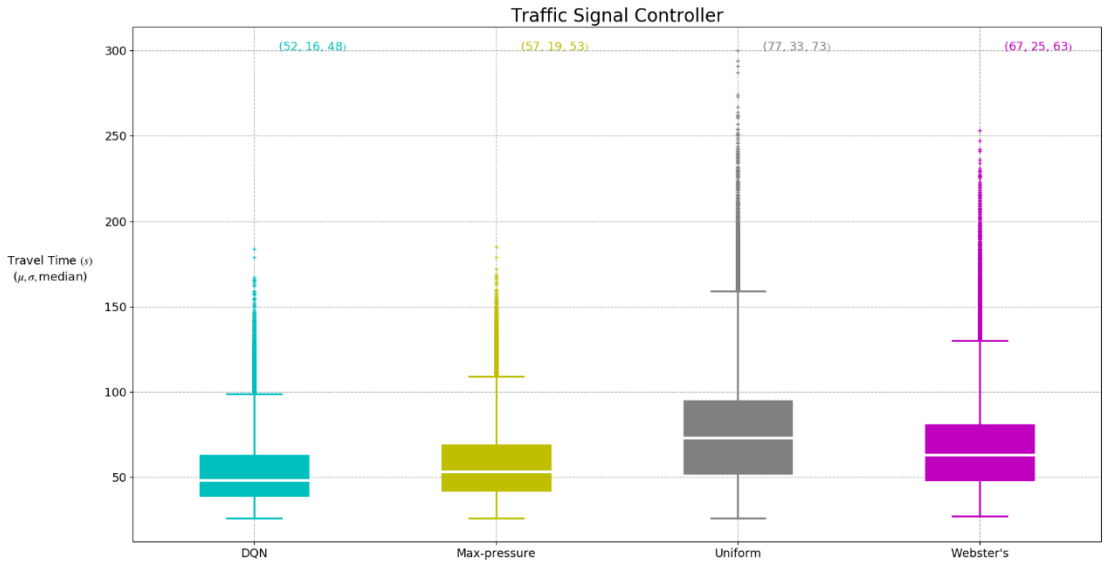

Figure 6.7-11. System Level Performance Comparison of DQN and other Non-learning Controllers with 6,000 Demand and No Incident in Corridor Network

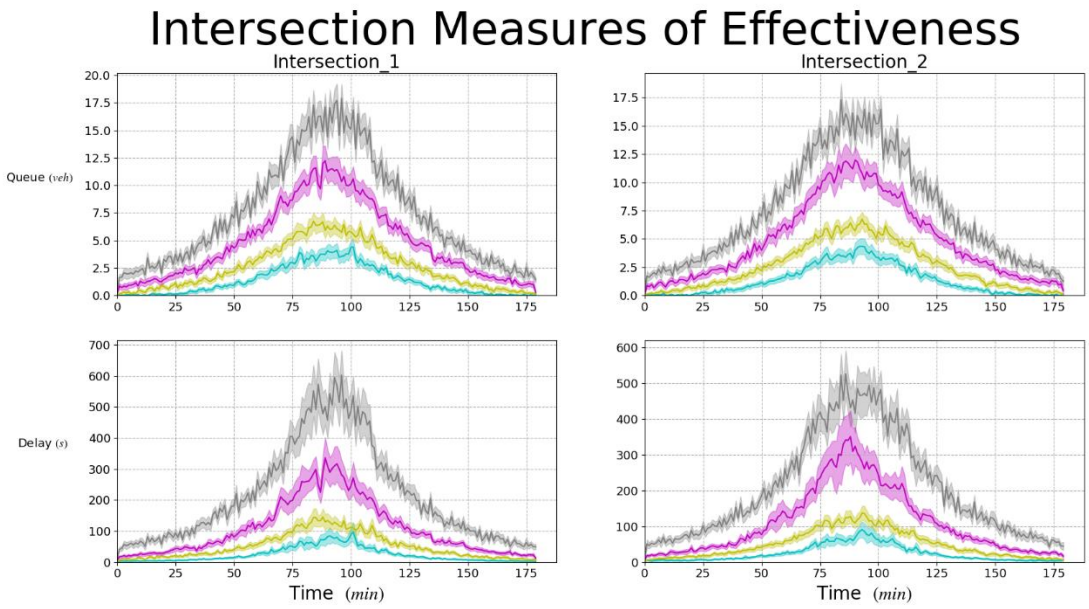

Figure 6.7-12. Intersection Level Performance Comparison of DQN and other Non-learning Controllers with 6,000 Demand and No Incident in Corridor Network

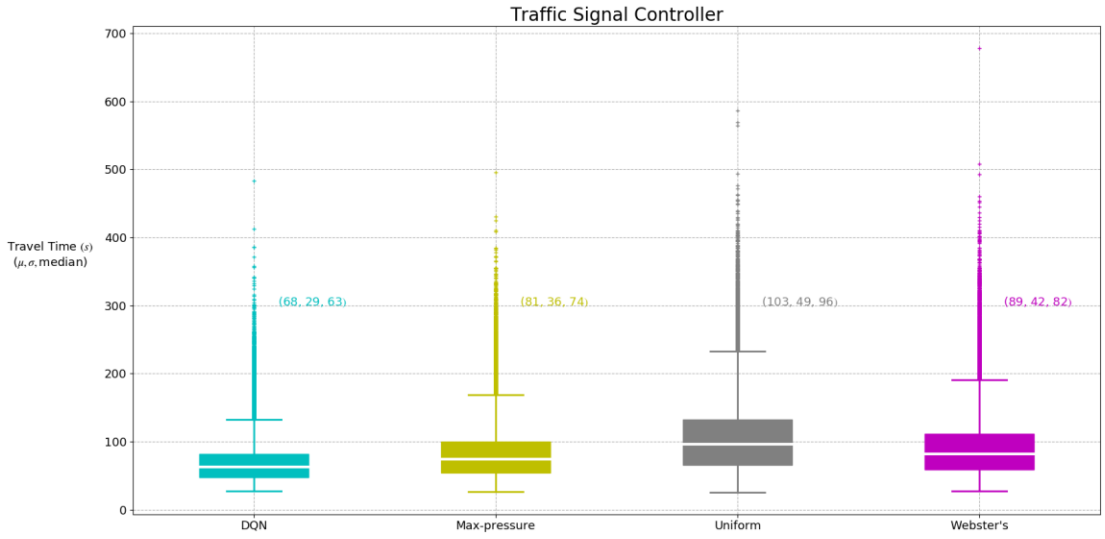

Figure 6.7-13. System Level Performance Comparison of DQN and other Non-learning Controllers with 6,000 Demand and No Incident in 2x2 Grid Network

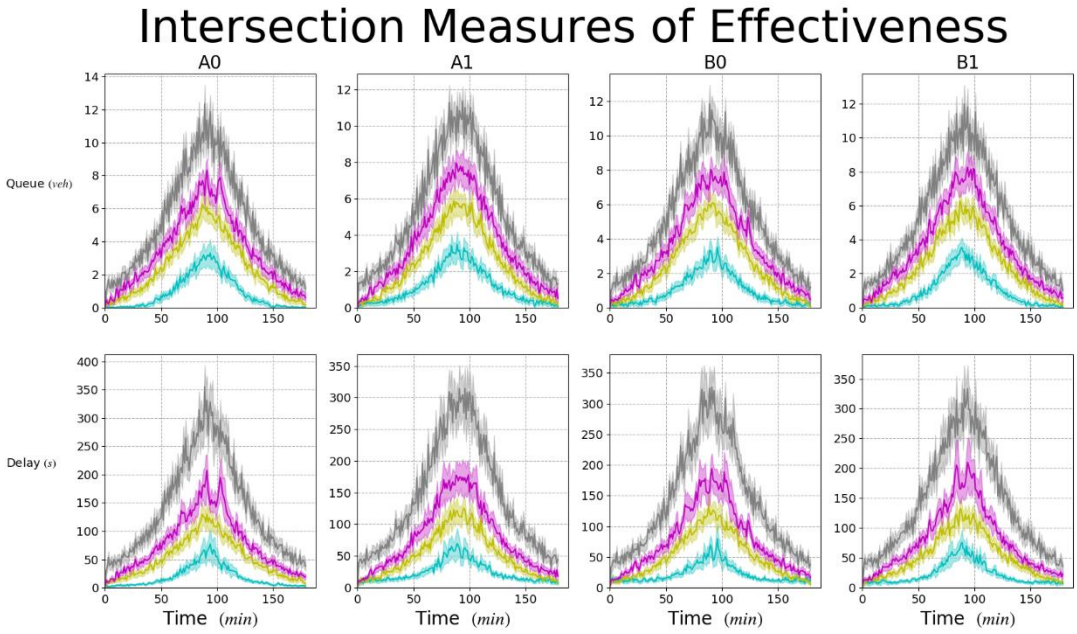

Figure 6.7-14. Intersection Level Performance Comparison of DQN and other Non-learning Controllers with 6,000 Demand and No Incident in 2x2 Grid Network

It is evident that the DQN outperforms the other controllers by directly applying the incident model to the non-incident situation.

In addition to testing the same model in scenarios with and without incidents, we also apply the same model to cases with lower traffic demand. We reduce the demand by half, resulting in about 4,000 vehicle demand in the following simulations. Figure 6.7-15 to Figure 6.7-22 display the performance of each traffic signal controller when applying its model from the 6,000-demand scenario with incidents to the 4,000-demand scenario, both with and without incidents.

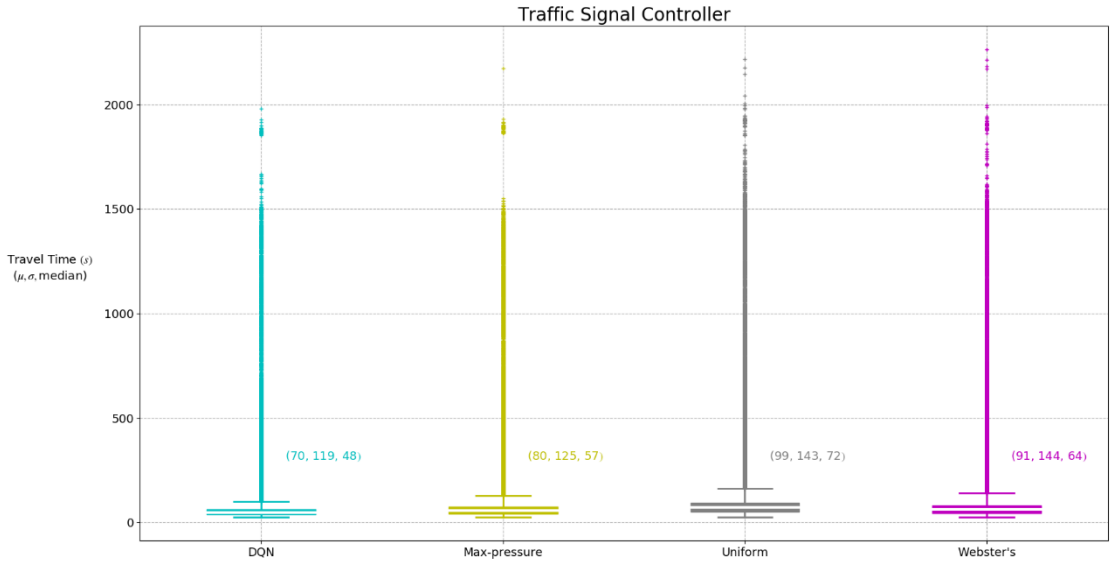

Figure 6.7-15. System Level Performance Comparison of DQN and other Non-learning Controllers with 4,000 Demand and Incident in Corridor Network

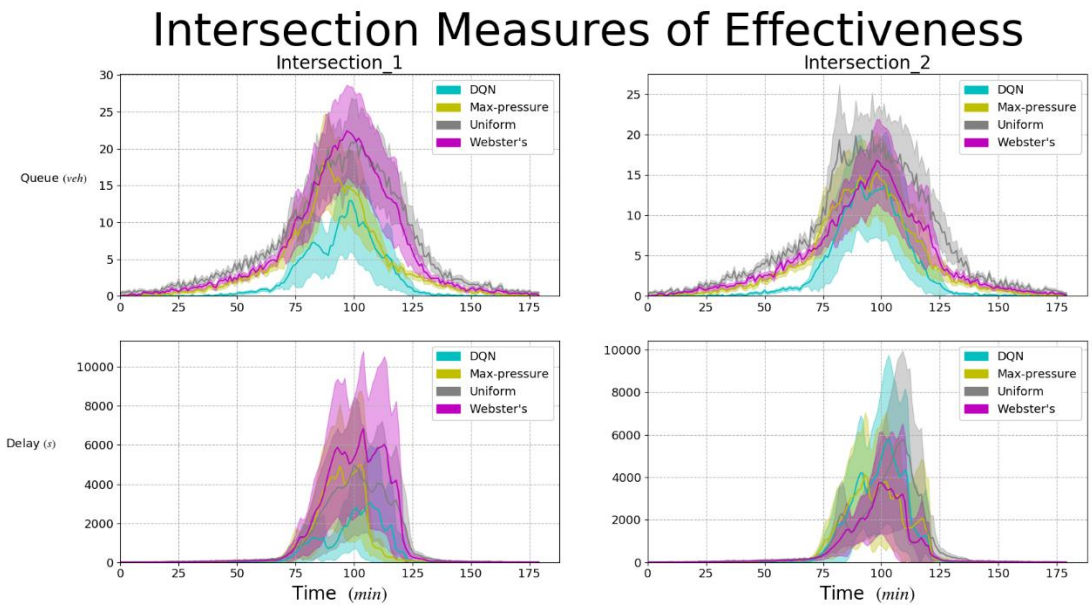

Figure 6.7-16. Intersection Level Performance Comparison of DQN and other Non-learning Controllers with 4,000 Demand and Incident in Corridor Network

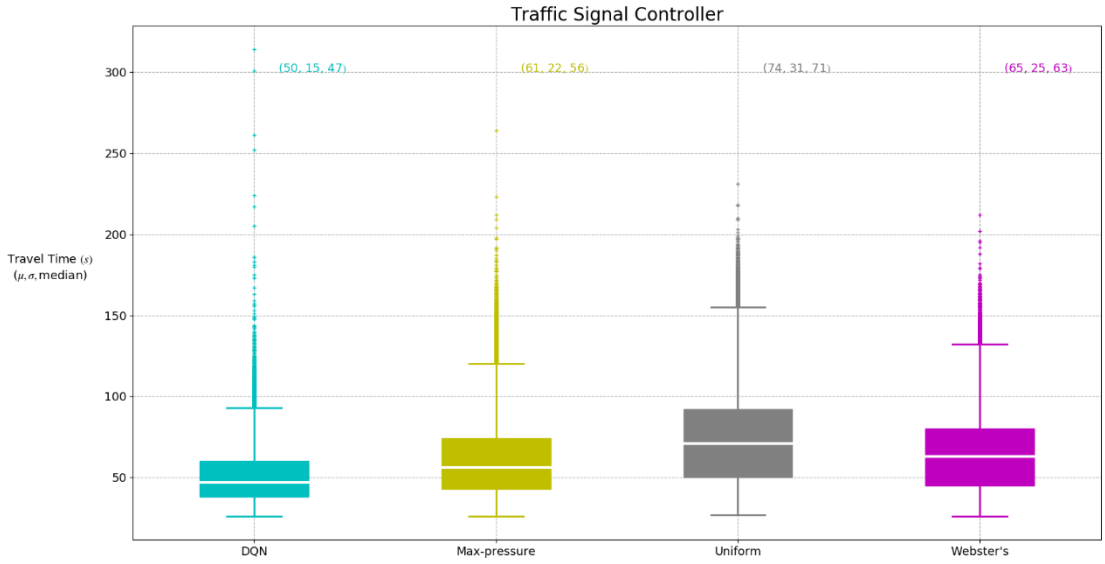

Figure 6.7-17. System Level Performance Comparison of DQN and other Non-learning Controllers with 4,000 Demand and No Incident in Corridor Network

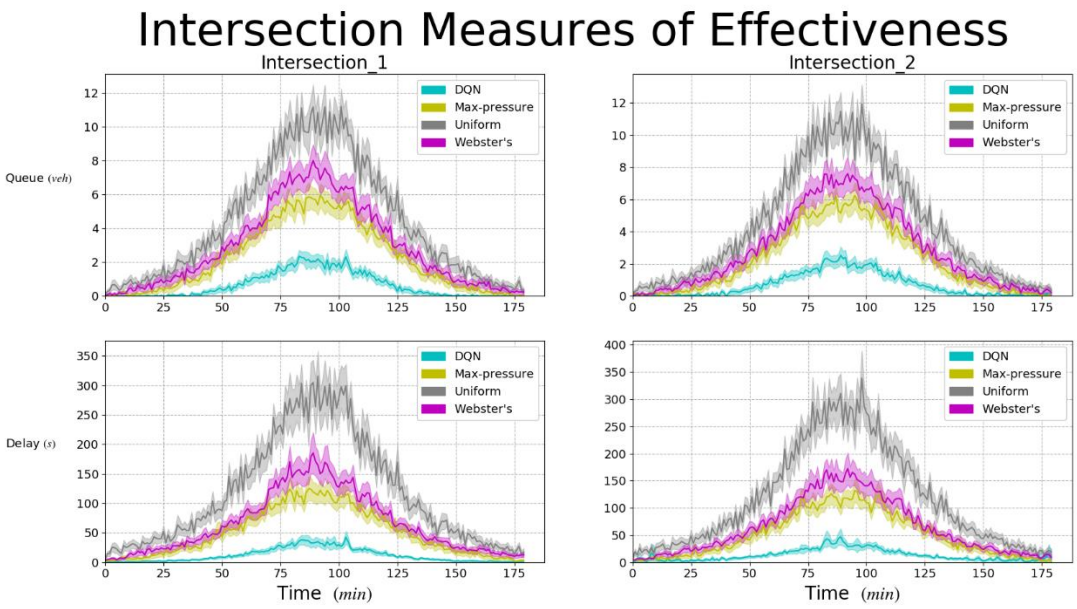

Figure 6.7-18. Intersection Level Performance Comparison of DQN and other Non-learning Controllers with 4,000 Demand and No Incident in Corridor Network

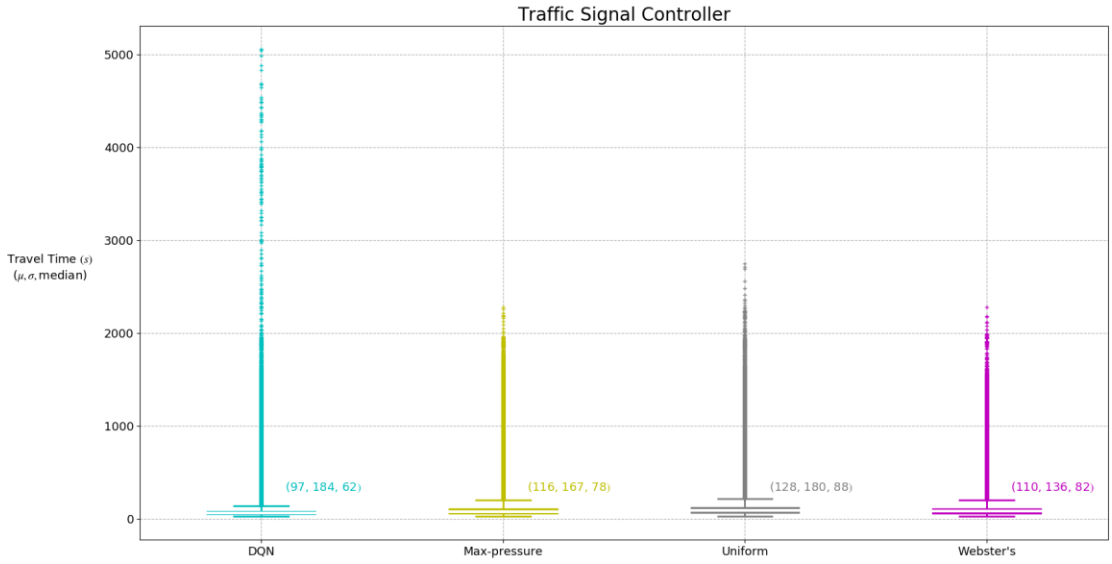

Figure 6.7-19. System Level Performance Comparison of DQN and other Non-learning Controllers with 4,000 Demand and Incident in 2x2 Grid Network

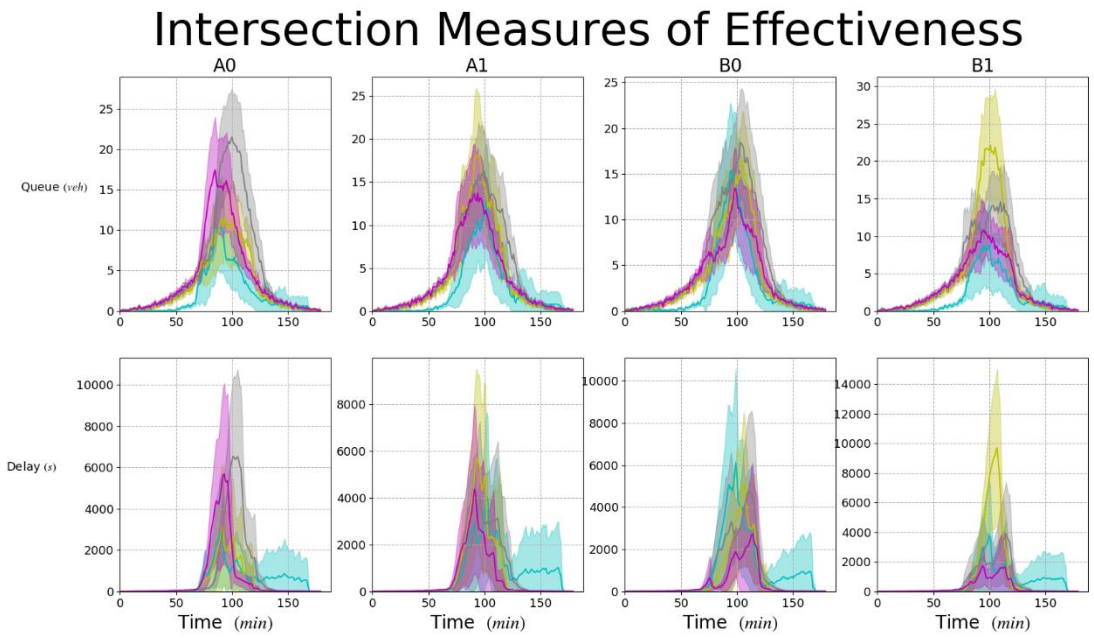

Figure 6.7-20. Intersection Level Performance Comparison of DQN and other Non-learning Controllers with 4,000 Demand and Incident in 2x2 Grid Network

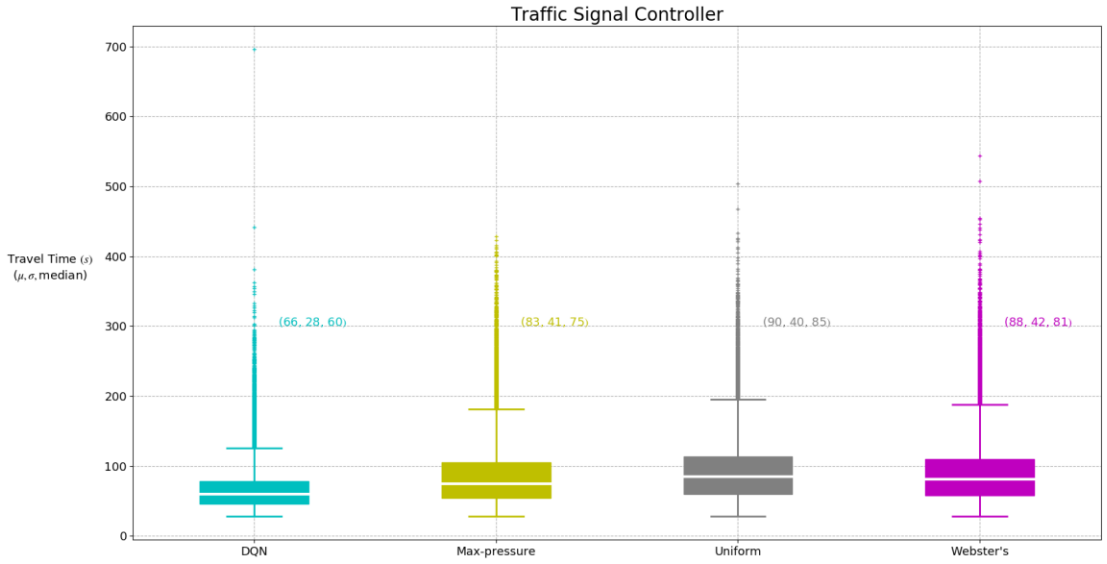

Figure 6.7-21. System Level Performance Comparison of DQN and other Non-learning Controllers with 4,000 Demand and No Incident in 2x2 Grid Network

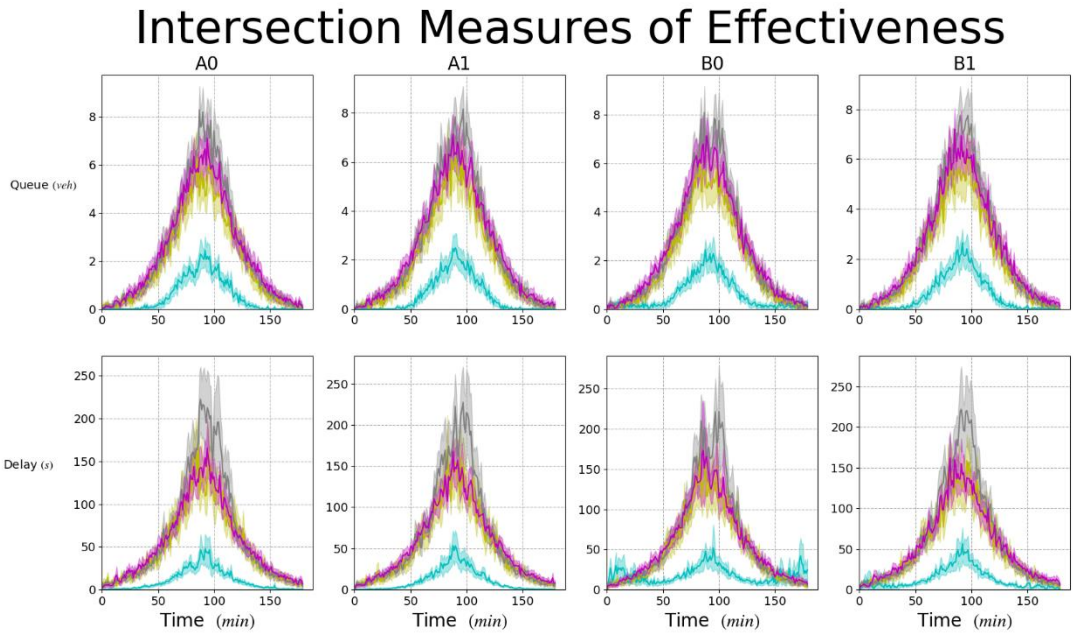

Figure 6.7-22. Intersection Level Performance Comparison of DQN and other Non-learning Controllers with 4,000 Demand and No Incident in 2x2 Grid Network

In general, the application of the DQN model to scenarios with lower traffic volumes, both with and without incidents, demonstrates its notable performance advantage over non-learning models. A peculiar observation from the above figures emerges in the 4,000-demand incident grid network scenario, where direct application of the model encounters some issues toward the end of the simulation. This is likely due to the DQN model's limited exposure to scenarios where vehicles are cleared from the network following an incident, as experienced in the original 6,000-demand grid network with an incident. It should be noted that we use a temporal difference with a step of 2 to calculate the long-term rewards of the DQN model.

To address this limitation, we conduct further training of the original model in a scenario with a traffic demand of 4,000 and an incident. Figure 6.7-23 and Figure 6.7-24 showcase the results of this refined approach. The results shows that the DQN can be improved further by exposing it to enough training time.

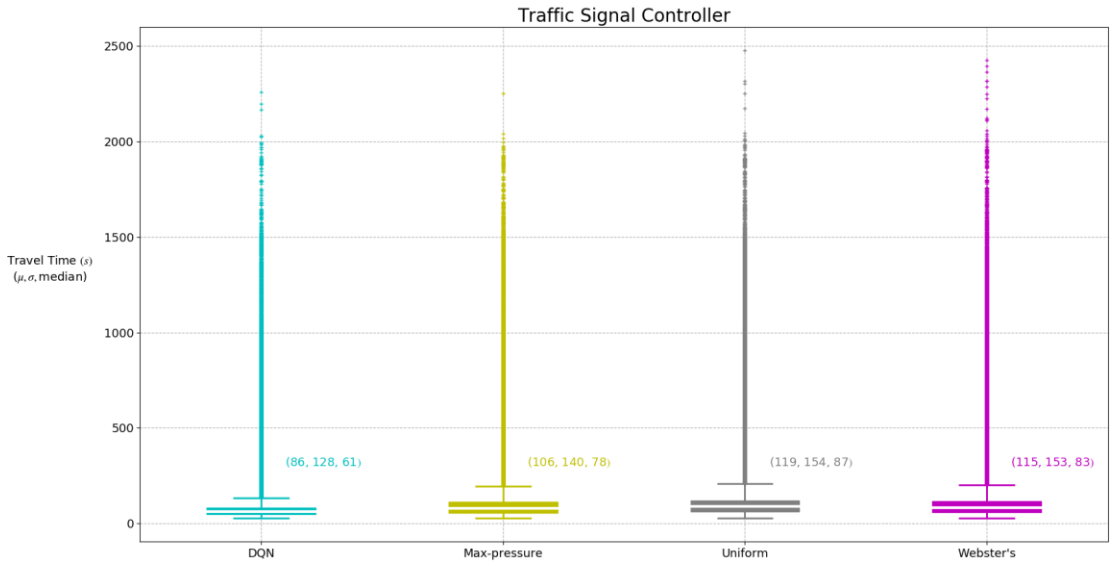

Figure 6.7-23. System Level Performance Comparison of DQN and other Non-learning Controllers with 4,000 Demand and No Incident in 2x2 Grid Network (with further training)

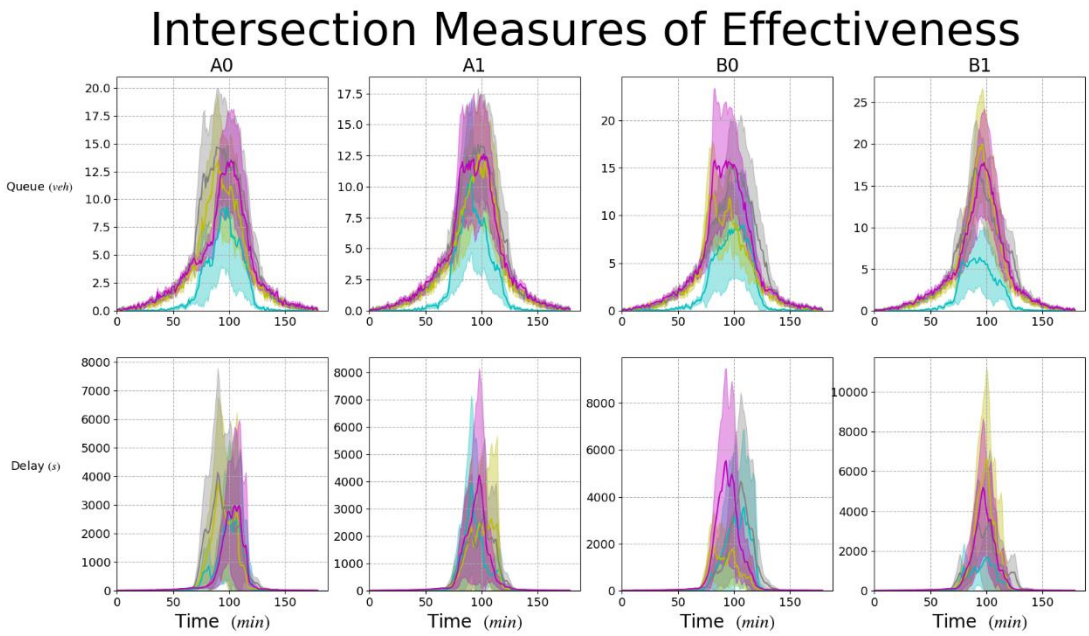

Figure 6.7-24. Intersection Level Performance Comparison of DQN and other Non-learning Controllers with 4,000 Demand and No Incident in 2x2 Grid Network(with further training)

6.8 Conclusion

In conclusion, this chapter has presented a comprehensive comparative analysis between the DQN traffic signal controller and the traditional non-learning traffic signal controller

techniques under the influence of traffic incidents in the network. Through rigorous evaluation, it has been demonstrated that the DQN traffic signal controller significantly outperforms its non-learning counterparts. The ability of the DQN controller to adapt and learn from its environment, coupled with its capacity to handle unpredictable traffic situations, enables it to provide more efficient and effective traffic signal timings.

We employ two distinct network configurations, a 2-intersection corridor and a 2x2 grid network, to assess the performance of these controllers when confronted with traffic incidents. After fine-tuning hyperparameters and further training the DQN controller, we generate results for comparison. Additionally, we apply the model to a scenario without incidents to obtain similar comparative results, highlighting the superior performance of the DQN model. We also investigate lower demand scenarios both with and without traffic incidents in the network to evaluate the robustness of the DQN controller's performance. The findings indicate that, once properly trained, the DQN controller delivers consistent performance across various situations.

The implementation of the DQN traffic signal controller has shown great promise in minimizing congestion, reducing travel time, and enhancing overall traffic flow in the presence of traffic incidents. By incorporating state-of-the-art machine learning techniques, the DQN traffic signal controller effectively manages traffic demands, mitigating the impact of incidents on urban mobility. As a result, this innovative approach offers substantial benefits to cities and urban planners by paving the way for a more sustainable and intelligent transportation system. While further research and refinements are necessary to optimize the performance and scalability of the DQN traffic signal controller, the findings presented in this chapter solidify its potential as a critical component in the future of smart traffic management.

Chapter 7. Summary and Conclusions

7.1 Summary

In recent years, reinforcement learning (RL) has emerged as a promising approach to optimizing traffic signal control. This technique involves enabling traffic signals to learn and adapt to real-time traffic conditions autonomously, resulting in reduced congestion, improved traffic flow, and enhanced road safety. Traditional traffic signal control methods, such as fixed-time and actuated systems, have shown limitations in handling dynamic traffic conditions. Reinforcement learning overcomes these limitations by allowing traffic signals to learn from their environment and make decisions based on current traffic conditions. This adaptive behavior results in more efficient traffic management and minimized delays for commuters. Various reinforcement learning algorithms, including Q-learning, deep Q-networks (DQNs), and proximal policy optimization (PPO), have been explored to address traffic signal control problems. These methods have demonstrated the potential to reduce waiting times, vehicle emissions, and fuel consumption by learning optimal traffic signal policies.

This dissertation examines the efficacy of the simplest reinforcement learning framework, Q-learning, integrated with deep neural networks for optimizing traffic signal control in various network configurations, both with and without traffic incidents. Chapter 2 presents an extensive literature review to assess the current state of research and implementation of reinforcement learning in traffic signal control optimization. Various reinforcement learning approaches have been investigated to enhance intersection performance in transportation networks by modifying traffic signal plans. These methods have yielded promising results. However, there is limited research that presents a robust workflow, including hyperparameter tuning – a crucial and fundamental step in developing machine learning algorithms. Furthermore, the integration of reinforcement learning with traffic incident management for optimizing traffic signals and minimizing the impact of network disruptions due to incidents has not been adequately addressed, despite a significant practical demand for such solutions.

In Chapter 3, the main focus is on addressing these two gaps by elucidating the concept of reinforcement learning, with an emphasis on Q-learning, which when combined with deep neural networks, results in the formation of a deep Q-network (DQN). The chapter not only highlights the benefits of DQN but also discusses its drawbacks and various modifications, such as the incorporation of target networks and experience replay, which can be employed to improve DQN performance.

Chapter 4 outlines the creation of an incident generation module within an open-source microsimulation platform, SUMO. This module assists in generating experiences for the DQN agent, enabling it to gather crucial information from simulations involving traffic incidents. Consequently, the agent learns to modify the traffic signal controller to minimize the network's incident impact. The developed module simplifies the effects of single or multiple vehicle occurrences into a single vehicle with varying lengths to accurately represent its real-world impact. Furthermore, the chapter introduces a depiction of the emergency vehicle rescue process to enhance the realism of the simulations for the reinforcement learning agent. The random generation of incidents eliminates the need for model developers to gather difficult-to-obtain real-world data for use in simulations.

Chapter 5 carries out an in-depth hyperparameter tuning of the DQN within a single intersection simulation scenario. This chapter identifies the most significant hyperparameters in the DQN model, such as the learning rate and reward discount factor. An extensive computational process is undertaken to determine the optimal combination of hyperparameters for both learning (DQN) and non-learning traffic signal controllers (Max-pressure, Uniform, and Websters) within the single intersection scenario. Upon completion of the DQN model training based on the hyperparameter tuning, it is concluded that the DQN outperforms the other non-learning traffic signal controllers.

In Chapter 6, the DQN agent is introduced to a more complex environment, incorporating various network configurations (corridor and 2x2 grid network) and randomly generated incidents within the network. Utilizing the hyperparameter tuning results from the single intersection scenario, the range of potential values for the learning rate and discount factor is narrowed when tuning the corridor and 2x2 grid network DQN models. Experimental results reveal that the DQN outperforms non-learning controllers in both incident and non-incident networks using a single model. This suggests that manual switching of traffic signal plans is unnecessary during implementation, as the DQN can adapt to fluctuating demand patterns. This finding is further corroborated by testing the performance of all controllers under lower traffic demand settings. The results indicate that the DQN can be trained further to handle various traffic demands, demonstrating its promising advantages over non-learning traffic signal controllers.

7.2 Directions for Future Research

In future work, we plan to explore the application of more advanced reinforcement learning (RL) frameworks to optimize traffic signal control performance. By leveraging cutting-edge algorithms and techniques such as multi-agent RL, hierarchical RL, and deep RL, we aim to create a more efficient and adaptive traffic signal control system that can better handle complex urban environments. This will involve designing reward functions that capture various objectives, such as reducing congestion, minimizing travel time, and improving fuel efficiency, while considering the diverse needs of different road users, such as pedestrians, cyclists, and public transportation. Additionally, incorporating real-time data from connected vehicles, traffic sensors, and IoT devices will enable our RL-based traffic control system to be more responsive to dynamic traffic patterns and emerging conditions. Ultimately, our goal is to develop a scalable and robust traffic signal control system that can significantly improve traffic flow and contribute to the development of smarter and more sustainable cities.

In addition, we aim to focus on the practical implementation of reinforcement learning-based traffic signal control systems, bridging the gap between theoretical advancements and real-world applications. This will involve addressing challenges such as system integration, computational efficiency, and robustness to uncertainties, while ensuring that the system can be seamlessly integrated into existing traffic management infrastructures. Additionally, we plan to collaborate with local authorities, transportation agencies, and stakeholders to conduct pilot tests in various urban settings, allowing us to gather critical insights and feedback on the performance, scalability, and adaptability of our proposed system. To foster public acceptance and engagement, we will also emphasize the importance of transparent and interpretable decision-making processes within the RL framework. By considering the complex interplay between

technical feasibility, regulatory compliance, and public acceptance, we strive to deploy an effective reinforcement learning-based traffic signal control system that can contribute to more efficient, safe, and sustainable urban transportation networks.

Machine learning, as the driving force behind the future of technology, holds immense potential for revolutionizing traffic signal control systems. As urban centers continue to expand, the optimization of traffic flow has become increasingly critical to reduce congestion, fuel consumption, and emissions. Studying and implementing machine learning techniques in traffic signal control can lead to adaptive and intelligent systems that dynamically respond to real-time traffic conditions, enhancing overall efficiency and safety. Therefore, it is imperative that researchers, engineers, and policymakers closely examine and collaborate on the development and integration of machine learning methods in traffic management to effectively address the growing complexities of modern transportation networks.

Chapter 8. Glossary

AI	Artificial Intelligence
CFP	Cyclic Flow Profiles
DP	Dynamic Programming
DTA	Dynamic Traffic Assignment
DQN	Deep Q Network
DNN	Deep Neural Network
MDP	Markov Decision Process
MC	Monte Carlo
MOEs	Measurement of Effectiveness
OPAC	Optimized Policies for Adaptive Control
RHODES	Real-Time Hierarchical Optimized Distributed and Effective System
RIMS	Rutgers Incident Management System
SCATS	Sydney Coordinated Adaptive Traffic System
SCOOT	Split Cycle Offset Optimization Technique
SOTL	Self-Organizing Traffic Light
TRRL	Transport and Road Research Laboratory
TIM	Traffic Incident Management
TD	Temporal Difference
VISTA	Visual Interactive System for Transport Algorithms
SUMO	Simulation of Urban Mobility

Chapter 9. References

1. Aboudolas, K. M. A. E., Papageorgiou, M., Kouvelas, A., & Kosmatopoulos, E. (2010). 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.
2. Abdoos, M., Mozayani, N., & Bazzan, A. L. (2011, October). Traffic light control in non-stationary environments based on multi agent Q-learning. In 2011 14th International IEEE conference on intelligent transportation systems (ITSC) (pp. 1580-1585). IEEE.
3. Abdoos, M., Mozayani, N., & Bazzan, A. L. (2014). Hierarchical control of traffic signals using Q-learning with tile coding. Applied intelligence, 40(2), 201-213.
4. Abdulhai, B., Pringle, R., & Karakoulas, G. J. (2003). Reinforcement learning for true adaptive traffic signal control. Journal of Transportation Engineering, 129(3), 278-285.
5. Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A., & Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. Heliyon, 4(11), e00938.
6. Akcelik, R., Besley, M., & Chung, E. (1998). An evaluation of SCATS Master Isolated control. In ARRB TRANSPORT RESEARCH LTD CONFERENCE, 19TH, 1998, SYDNEY, NEW SOUTH WALES, AUSTRALIA.
7. Arel, I., Liu, C., Urbanik, T., & Kohls, A. G. (2010). Reinforcement learning-based multi-agent system for network traffic signal control. IET Intelligent Transport Systems, 4(2), 128-135.
8. Balaji, P. G., German, X., & Srinivasan, D. (2010). Urban traffic signal control using reinforcement learning agents. IET Intelligent Transport Systems, 4(3), 177-188.
9. Ban, X. J., Kamga, C., Wang, X., Wojtowicz, J., Klepadlo, E., Sun, Z., & Mouskos, K. (2014). Adaptive Traffic Signal Control System (ACS-Lite) for Wolf Road, Albany, New York (No. C-10-13). New York (State). Dept. of Transportation.
10. Ban, X., Wojtowicz, J. M., & Li, W. (2016). Decision-making tool for applying adaptive traffic control systems (No. C-13-04). New York State Energy Research and Development Authority.
11. Bell, M. G. (1992). Future directions in traffic signal control. Transportation Research Part A: Policy and Practice, 26(4), 303-313.
12. Carson, J. L. (2010). Best practices in traffic incident management (No. FHWA-HOP-10-050). United States. Federal Highway Administration. Office of Transportation Operations.
13. Chin, Y. K., Bolong, N., Kiring, A., Yang, S. S., & Teo, K. T. K. (2011). Q-learning based traffic optimization in management of signal timing plan. International Journal of Simulation, Systems, Science & Technology, 12(3), 29-35.
14. Cools, S. B., Gershenson, C., & D'Hooghe, B. (2013). Self-organizing traffic lights: A realistic simulation. In Advances in applied self-organizing systems (pp. 45-55). Springer, London.
15. Dell, P. A. O. L. O., & Mirchandani, B. (1995). REALBAND: An approach for real-time coordination of traffic flows on networks. Transp. Res. Rec, 1494, 106-116.
16. Dougald, L. E., Venkatanarayana, R., & Goodall, N. J. (2016). Traffic incident management quick clearance guidance and implications (No. FHWA/VTRC 16-R9, VTRC 16-R9). Virginia Transportation Research Council.
17. Dutta, U., Lynch, J., Dara, B., & Bodke, S. (2010). Safety Evaluation of the SCATS Control System (No. RC-1545K).

18. El-Tantawy, S., Abdulhai, B., & Abdelgawad, H. (2014). Design of reinforcement learning parameters for seamless application of adaptive traffic signal control. Journal of Intelligent Transportation Systems, 18(3), 227-245.
19. Fellendorf, M. (1994, October). VISSIM: A microscopic simulation tool to evaluate actuated signal control including bus priority. In 64th Institute of Transportation Engineers Annual Meeting (Vol. 32, pp. 1-9). Springer.
20. Gartner, N. H., Pooran, F. J., & Andrews, C. M. (2001, August). Implementation of the OPAC adaptive control strategy in a traffic signal network. In ITSC 2001. 2001 IEEE Intelligent Transportation Systems. Proceedings (Cat. No. 01TH8585) (pp. 195-200). IEEE.
21. Gartner, N. H., Pooran, F. J., & Andrews, C. M. (2002). Optimized policies for adaptive control strategy in real-time traffic adaptive control systems: Implementation and field testing. Transportation Research Record, 1811(1), 148-156.
22. Gartner, N. H. (2005, October). Development of demand-responsive strategies for urban traffic control. In System Modelling and Optimization: Proceedings of the 11th IFIP Conference Copenhagen, Denmark, July 25–29, 1983 (pp. 166-174). Berlin, Heidelberg: Springer Berlin Heidelberg.
23. Gayah, V. V., Gao, X. S., & Nagle, A. S. (2014). On the impacts of locally adaptive signal control on urban network stability and the macroscopic fundamental diagram. Transportation Research Part B: Methodological, 70, 255-268.
24. Ge, H., Song, Y., Wu, C., Ren, J., & Tan, G. (2019). Cooperative deep Q-learning with Q-value transfer for multi-intersection signal control. IEEE Access, 7, 40797-40809.
25. Genders, W., & Razavi, S. (2016). Using a deep reinforcement learning agent for traffic signal control. arXiv preprint arXiv:1611.01142.
26. Gershenson, C. (2005). A general methodology for designing self-organizing systems. arXiv preprint nlin/0505009.
27. Ghaman, R. S. (2007). ACS Lite: A Signal Timing Strategy for Closed Loop Systems. In ITE 2007 Annual Meeting and ExhibitInstitute of Transportation Engineers (ITE).
28. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.
29. Goodfellow, I., McDaniel, P., & Papernot, N. (2018). Making machine learning robust against adversarial inputs. Communications of the ACM, 61(7), 56-66.
30. Hadiuzzaman, M., Qiu, T. Z., & Lin, Y. (2012). Real-time Traffic State Estimation and Prediction for Active Traffic and Demand Management: The Application of DynaTAM. In CICTP 2012: Multimodal Transportation Systems—Convenient, Safe, Cost-Effective, Efficient (pp. 3335-3351).
31. Head, K. L., Mirchandani, P. B., & Sheppard, D. (1992). Hierarchical framework for real-time traffic control (No. 1360).
32. Howard, R. A. (1960). Dynamic programming and markov processes.
33. Hunt, P. B., Robertson, D. I., Bretherton, R. D., & Winton, R. I. (1981). SCOOT-a traffic responsive method of coordinating signals (No. LR 1014 Monograph).
34. Kamilaris, A., & Prenafeta-Boldú, F. X. (2018). Deep learning in agriculture: A survey. Computers and electronics in agriculture, 147, 70-90
35. Kell, J. H., & Fullerton, I. J. (1991). Manual of traffic signal design.
36. Klein, L. A. (2001). Sensor technologies and data requirements for ITS.

37. Krajzewicz, D., Erdmann, J., Behrisch, M., & Bieker, L. (2012). Recent development and applications of SUMO-Simulation of Urban MObility. International journal on advances in systems and measurements, 5(3&4).
38. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444
39. Lin, F. B. (1985). Optimal timing settings and detector lengths of presence mode full-actuated control (No. 1010).
40. Lin, L. J. (1992). Reinforcement learning for robots using neural networks. Carnegie Mellon University.
41. Liu, H., & Hall, R. (2000). INCISM: Users Manual.
42. Logi, F., & Ritchie, S. G. (2001). Development and evaluation of a knowledge-based system for traffic congestion management and control. Transportation Research Part C: Emerging Technologies, 9(6), 433-459.
43. Lowrie, P. R. (1990). Scats, sydney co-ordinated adaptive traffic system: A traffic responsive method of controlling urban traffic.
44. Mao, T., Mihaita, A. S., & Cai, C. (2019). Traffic signal control optimization under severe incident conditions using Genetic Algorithm. arXiv preprint arXiv:1906.05356.
45. Markov, A. A. (1954). The theory of algorithms. Trudy Matematicheskogo Instituta Imeni VA Steklova, 42, 3-375.
46. McClelland, J. L., McNaughton, B. L., & O'Reilly, R. C. (1995). Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. Psychological review, 102(3), 419.
47. Mirchandani, P., & Head, L. (2001). A real-time traffic signal control system: architecture, algorithms, and analysis. Transportation Research Part C: Emerging Technologies, 9(6), 415-432.
48. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. nature, 518(7540), 529-533.
49. Shoufeng, L., Ximin, L., & Shiqiang, D. (2008, April). Q-learning for adaptive traffic signal control based on delay minimization strategy. In 2008 IEEE International Conference on Networking, Sensing and Control (pp. 687-691). IEEE.
50. Slinn, M., Matthews, P., & Guest, P. (1998). Traffic engineering design. Principles and practice.
51. Stevanovic, A., Kergaye, C., & Martin, P. T. (2009, November). Scoot and scats: A closer look into their operations. In 88th Annual Meeting of the Transportation Research Board. Washington DC.
52. Ozbay, K., & Bartin, B. (2003). Incident management simulation. Simulation, 79(2), 69-82.
53. Ozbay, K. M., Xiao, W., Jaiswal, G., Bartin, B., Kachroo, P., & Baykal-Gursoy, M. (2009). Evaluation of incident management strategies and technologies using an integrated traffic/incident management simulation. World Review of Intermodal Transportation Research, 2(2-3), 155-186.
54. Prashanth, L. A., & Bhatnagar, S. (2010). Reinforcement learning with function approximation for traffic signal control. IEEE Transactions on Intelligent Transportation Systems, 12(2), 412-421.

55. Robertson, D. I. (1969). TRANSYT: a traffic network study tool.
56. Robertson, D. I. (1986). Research on the TRANSYT and SCOOT Methods of Signal Coordination. ITE journal, 56(1), 36-40.
57. Roess, R. P., Prassas, E. S., & McShane, W. R. (2004). Traffic engineering. Pearson/Prentice Hall.
58. Russell, S., & Norvig, P. (2002). Artificial intelligence: a modern approach.
59. Shelby, S. G., Bullock, D. M., Gettman, D., Ghaman, R. S., Sabra, Z. A., & Soyke, N. (2008, January). An overview and performance evaluation of ACS Lite–a low cost adaptive signal control system. In Transportation Research Board Annual Meeting (Vol. 190, pp. 130-137).
60. Skabardonis, A., Bertini, R. L., & Gallagher, B. R. (1998). Development and application of control strategies for signalized intersections in coordinated systems. Transportation research record, 1634(1), 110-117.
61. Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.
62. Syarif, I., Prugel-Bennett, A., & Wills, G. (2016). SVM parameter optimization using grid search and genetic algorithm to improve classification performance. TELKOMNIKA (Telecommunication Computing Electronics and Control), 14(4), 1502-1509.
63. Van Hasselt, H., Guez, A., & Silver, D. (2016, March). Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence (Vol. 30, No. 1).
64. Varaiya, P. (2013). Max pressure control of a network of signalized intersections. Transportation Research Part C: Emerging Technologies, 36, 177-195.
65. Wang, S. C. (2003). Artificial neural network. In Interdisciplinary computing in java programming (pp. 81-100). Springer, Boston, MA.
66. Watkins, C. J. C. H. (1989). Learning from delayed rewards.
67. Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning, 8(3), 279-292.
68. Webster, F. V. (1958). Traffic signal settings, road research technical paper no. 39. Road Research Laboratory.
69. Winston, P. H. (1992). Artificial intelligence. Addison-Wesley Longman Publishing Co., Inc..
70. Wirtz, J. J., Schofer, J. L., & Schulz, D. F. (2005). Using simulation to test traffic incident management strategies: The benefits of preplanning. Transportation research record, 1923(1), 82-90.
71. Yin, Y., Li, M., & Skabardonis, A. (2007). Offline offset refiner for coordinated actuated signal control systems. Journal of transportation engineering, 133(7), 423-432.

Appendix 1: SUMO Network Generation Script

The command below was run to generate the 4x4 grid network illustrated in this paper.

netgenerate --grid --grid.number=4 --grid.length=200 --default.lanenumber=2 --default.speed=20 --no-turnarounds=true --turn-lanes=1 --turn-lanes.length=100 --default-junction-type=traffic_light --grid.attach-length=200 --tls.yellow.time=3 --tls.left-green.time=12 --tls.allred.time = 2 --output-file=net.net.xml

Network generating parameters and their meanings:

--grid: grid network will be generated. SUMO also provides for other types of networks to be generated automatically, including spider and random networks.

--grid.length defines the length of each intersection leg in meters

--default.lanenumber defines the number of lanes for each approach

--default.speed defines the edge design speed in meters/second

--no-turnarounds defines whether to allow turn around for the left turn lane

--turn-lanes defines the number of left turn lanes

--turn-lanes.length defines the length of left turn lanes

--default-junction-type defines the intersections in the network are controlled by the pretimed traffic signals

--grid.attach-length defines the length of road attached to the fringe of intersections in the network

--tls.yellow.time defines the duration of yellow phase in seconds

--tls.left-green.time defines the protected left turn movement green time in seconds

--tls.allred.time defines the duration of all red phase in seconds

More options of calling NETGENERATE could be found in https://sumo.dlr.de/docs/netgenerate.html.

Appendix 2: Developed Traffic Demand Generating Script

Traffic demand was prepared by calling python randomTrips.py -n net.net.xml -r random.rou.xml --fringe-factor=100000000 --period=0.5 -e 3600.

Where randomTrips.py is a Python script tool provided by SUMO.

-n net.net.xml defines the location of the network file.

-r defines the name of the output route file.

--fringe-factor defines the ratios of through and internal traffic demand in the network. An extremely large number is used here to eliminate the internal traffic demand in the network.

--period defines the 1/number of vehicles generated per second. 0.5 used here means two vehicles will be generated per second in the network.

-e defines the end simulation step of generating trips so here one hour traffic demand is generated.

Appendix 3: DQN Hyperparameter Tuning Results for Single Intersection Network

ID	Discount Factor	Green Duration (seconds)	Learnig Rate	Number of Hidden Layers	Temporal Difference Steps	Update Frequency	Mean (seconds)	Standard Deviation (seconds)
1	0.5	6	1.00E-05	3	2	64	54	29
2	0.5	6	1.00E-05	3	1	64	55	31
3	0.5	6	0.001	6	1	128	56	37
4	0.5	6	0.001	6	2	64	56	34
5	0.5	6	0.0001	3	2	128	57	40
6	0.5	6	0.0001	6	2	128	57	36
7	0.5	6	1.00E-05	3	2	128	57	32
8	0.5	6	1.00E-05	6	1	128	57	37
9	0.5	6	1.00E-05	6	2	64	57	33
10	0.5	6	0.001	3	2	128	57	36
11	0.9	6	1.00E-05	6	2	64	57	35
12	0.5	6	0.0001	3	2	64	58	40
13	0.5	6	1.00E-05	6	2	128	58	39
14	0.5	6	0.001	6	1	64	58	39
15	0.5	6	0.001	6	2	128	58	40
16	0.9	6	1.00E-05	3	2	128	58	35
17	0.5	6	0.0001	3	1	64	59	40
18	0.5	6	0.0001	3	1	128	59	43
19	0.5	6	1.00E-05	3	1	128	59	32
20	0.5	6	0.001	3	1	128	59	37
21	0.5	6	1.00E-05	6	1	64	60	42
22	0.5	12	0.0001	3	1	64	60	33
23	0.5	12	0.0001	3	1	128	60	39
24	0.5	6	0.0001	6	1	64	61	46

25	0.5	6	0.0001	6	2	64	61	46
26	0.5	6	0.001	3	1	64	61	46
27	0.5	6	0.001	3	2	64	61	48
28	0.9	6	0.001	3	1	128	61	43
29	0.5	6	0.0001	6	1	128	63	52
30	0.9	6	0.001	3	1	64	64	38
31	0.9	6	1.00E-05	6	2	128	65	51
32	0.9	12	0.0001	3	1	64	66	35
33	0.9	12	0.0001	3	2	64	66	43
34	0.9	12	0.001	3	2	128	67	61
35	0.99	12	0.0001	3	2	128	69	41
36	0.9	6	0.001	6	2	64	71	57
37	0.5	12	0.0001	3	2	128	72	69
38	0.5	12	0.0001	6	1	64	72	71
39	0.5	12	0.001	3	2	64	72	69
40	0.99	12	0.0001	3	2	64	72	45
41	0.5	12	0.0001	6	1	128	73	70
42	0.5	12	0.001	3	1	64	73	72
43	0.9	12	0.0001	3	1	128	73	56
44	0.9	6	0.001	3	2	128	73	67
45	0.9	6	0.001	6	2	128	73	73
46	0.5	12	0.0001	3	2	64	74	71
47	0.5	12	0.001	3	1	128	74	75
48	0.9	6	1.00E-05	6	1	128	75	56
49	0.5	12	0.0001	6	2	128	76	79
50	0.5	12	1.00E-05	6	2	64	76	78
51	0.5	12	0.001	6	1	128	76	78
52	0.9	6	1.00E-05	6	1	64	76	48
53	0.9	12	0.0001	6	2	128	76	79
54	0.9	12	0.001	6	2	128	76	80
55	0.5	12	1.00E-05	3	1	128	77	79

56	0.5	12	0.001	6	1	64	77	81
57	0.9	12	1.00E-05	6	2	128	77	81
58	0.5	12	0.0001	6	2	64	78	82
59	0.5	12	1.00E-05	6	1	128	78	82
60	0.5	12	1.00E-05	6	2	128	78	81
61	0.9	12	0.001	3	2	64	78	80
62	0.9	6	0.0001	6	2	64	78	81
63	0.9	6	0.001	3	2	64	78	81
64	0.9	12	0.0001	6	2	64	78	81
65	0.9	12	0.0001	6	1	64	78	82
66	0.9	12	0.001	6	1	128	78	82
67	0.5	12	1.00E-05	3	1	64	79	83
68	0.5	12	0.001	3	2	128	79	82
69	0.9	6	0.0001	3	2	64	79	74
70	0.5	12	0.001	6	2	64	80	84
71	0.5	12	0.001	6	2	128	80	86
72	0.9	12	0.0001	3	2	128	80	82
73	0.9	12	1.00E-05	6	2	64	80	84
74	0.5	12	1.00E-05	6	1	64	81	87
75	0.9	6	1.00E-05	3	2	64	81	74
76	0.9	12	0.001	3	1	64	81	79
77	0.9	12	0.001	3	1	128	81	81
78	0.9	12	1.00E-05	3	2	128	81	85
79	0.5	12	1.00E-05	3	2	64	82	88
80	0.9	12	0.001	6	1	64	82	86
81	0.9	12	0.0001	6	1	128	82	88
82	0.5	12	1.00E-05	3	2	128	84	87
83	0.9	6	0.0001	3	2	128	85	87
84	0.9	12	1.00E-05	3	2	64	85	92

85	0.9	12	0.001	6	2	64	87	90
86	0.99	12	0.001	6	2	128	90	138
87	0.9	6	0.0001	3	1	64	92	81
88	0.9	12	1.00E-05	6	1	64	93	106
89	0.9	6	0.001	6	1	128	94	96
90	0.99	12	0.001	3	2	128	98	204
91	0.9	6	0.0001	3	1	128	99	117
92	0.99	12	1.00E-05	6	2	128	101	100
93	0.99	12	1.00E-05	6	2	64	103	112
94	0.9	6	0.0001	6	2	128	104	133
95	0.9	12	1.00E-05	6	1	128	105	128
96	0.9	6	0.001	6	1	64	107	141
97	0.99	6	0.0001	3	2	128	115	166
98	0.9	6	0.0001	6	1	64	129	219
99	0.9	6	0.0001	6	1	128	131	181
100	0.99	12	0.0001	6	2	128	131	366
101	0.9	12	1.00E-05	3	1	128	137	115
102	0.99	12	0.0001	6	1	128	144	270
103	0.99	12	0.0001	3	1	128	150	282
104	0.99	12	0.001	6	1	128	152	444
105	0.9	12	1.00E-05	3	1	64	156	141
106	0.99	6	1.00E-05	6	2	128	156	471
107	0.99	6	0.0001	6	1	128	160	180
108	0.99	6	0.0001	6	1	64	173	177
109	0.99	12	0.0001	6	2	64	190	566
110	0.9	6	1.00E-05	3	1	128	195	139
111	0.99	6	0.0001	3	2	64	207	351
112	0.99	12	0.001	3	1	128	213	696
113	0.99	6	0.0001	6	2	64	221	623
114	0.99	12	0.0001	3	1	64	224	434

115	0.9	6	1.00E-05	3	1	64	234	165
116	0.99	6	1.00E-05	3	2	128	268	185
117	0.99	6	0.0001	3	1	128	274	353
118	0.99	12	1.00E-05	3	1	64	284	227
119	0.99	12	1.00E-05	3	1	128	284	232
120	0.99	6	0.0001	6	2	128	294	800
121	0.99	6	1.00E-05	3	1	128	305	206
122	0.99	12	1.00E-05	3	2	128	307	260
123	0.99	6	0.0001	3	1	64	308	568
124	0.99	6	0.001	3	2	64	341	862
125	0.99	12	1.00E-05	6	1	128	341	1165
126	0.99	6	0.001	3	1	128	344	1054
127	0.99	12	0.001	6	1	64	347	1110
128	0.99	6	0.001	6	1	128	354	1174
129	0.99	6	1.00E-05	3	2	64	361	289
130	0.99	12	0.001	3	2	64	364	1156
131	0.99	12	1.00E-05	6	1	64	369	1188
132	0.99	6	0.001	6	2	128	370	1191
133	0.99	12	0.001	3	1	64	391	1017
134	0.99	6	0.001	6	2	64	400	1076
135	0.99	6	0.001	3	1	64	403	1243
136	0.99	6	0.001	3	2	128	424	1029
137	0.99	6	1.00E-05	3	1	64	496	931
138	0.99	12	0.0001	6	1	64	504	1216
139	0.99	12	1.00E-05	3	2	64	655	1117
140	0.99	6	1.00E-05	6	1	128	728	1536
141	0.99	6	1.00E-05	6	2	64	1104	1554
142	0.99	6	0.001	6	1	64	1129	2038

143	0.99	12	0.001	6	2	64	1184	1985
144	0.99	6	1.00E-05	6	1	64	1263	2072

Appendix 4: Max-pressure Hyperparameter Tuning Results for Single Intersection Network

ID	Green Duration	Mean (seconds)	Standard Deviation (seconds)
1	7	56	22
2	9	58	22
3	8	58	24
4	12	59	25
5	11	60	25
6	10	60	26
7	6	59	29
8	5	61	34
9	13	61	35
10	14	61	36
11	16	63	39
12	15	66	49
13	18	70	58
14	17	71	58
15	20	75	69
16	19	76	71
17	21	79	77
18	22	81	81
19	24	82	82
20	23	84	85
21	25	84	87

Appendix 5: Uniform Hyperparameter Tuning Results for Single Intersection Network

ID	Green Duration (seconds)	Mean (seconds)	Standard Deviation (seconds)
1	21	81	39
2	22	81	39
3	23	82	39
4	25	84	40
5	24	84	42
6	16	82	50
7	19	84	48
8	17	84	52
9	18	85	53
10	20	86	52
11	14	84	55
12	15	89	62
13	12	95	75
14	13	96	74
15	11	106	87
16	10	123	106
17	9	125	105
18	8	164	141
19	7	170	144
20	6	202	168
21	5	250	203

Appendix 6: Webster's Hyperparameter Tuning Results for Single Intersection Network

ID	Max Cycle Length (seconds)	Min Cycle Length (seconds)	Time Interval (seconds)	Satuation Flow Rate	Mean (seconds)	Standard Deviation (seconds)
1	200	60	600	0.38	74	42
2	180	40	600	0.38	75	44
3	160	60	600	0.38	75	45
4	160	40	600	0.38	75	48
5	200	40	600	0.44	74	49
6	200	40	1800	0.44	75	48
7	180	60	600	0.38	76	49
8	160	60	600	0.44	75	51
9	180	40	1800	0.44	76	51
10	200	40	600	0.38	76	51
11	160	40	600	0.44	76	52
12	180	40	600	0.44	75	53
13	160	40	600	0.3	80	49
14	180	40	600	0.3	80	49
15	160	40	1800	0.44	77	53
16	160	80	600	0.38	78	52
17	180	60	600	0.44	76	54
18	160	40	900	0.44	76	55
19	160	60	900	0.3	80	51
20	160	80	600	0.3	81	50
21	200	60	900	0.38	77	54
22	160	40	900	0.38	78	54
23	160	60	600	0.3	81	52
24	180	40	900	0.44	77	56
25	180	60	900	0.38	78	55
26	200	60	600	0.3	83	50

27	160	40	900	0.3	80	54
28	180	40	900	0.38	79	55
29	180	60	600	0.3	83	51
30	200	40	600	0.3	83	52
31	200	40	900	0.3	82	53
32	200	40	900	0.38	79	56
33	200	60	600	0.44	78	58
34	200	80	600	0.3	84	52
35	160	40	1800	0.38	79	58
36	180	40	900	0.3	82	55
37	180	40	1800	0.38	79	58
38	180	60	900	0.44	78	59
39	180	80	600	0.3	84	53
40	200	40	1800	0.38	80	58
41	160	60	900	0.38	81	58
42	180	80	600	0.38	82	57
43	200	60	900	0.3	84	55
44	180	80	900	0.3	84	56
45	200	40	900	0.44	78	62
46	160	40	1800	0.3	83	59
47	160	80	600	0.44	81	62
48	200	80	600	0.38	83	60
49	180	80	600	0.44	81	63
50	180	60	900	0.3	85	60
51	200	80	600	0.44	81	64
52	160	80	900	0.3	85	61
53	180	40	1800	0.3	84	62
54	160	60	1800	0.3	85	62
55	200	60	900	0.44	81	66
56	200	60	1800	0.3	86	63
57	200	80	900	0.3	87	62
58	180	60	1800	0.3	86	64
59	200	60	1800	0.38	84	68

60	160	60	900	0.44	83	70
61	200	80	900	0.38	85	69
62	160	80	900	0.38	85	70
63	200	40	1800	0.3	88	67
64	180	60	1800	0.44	85	72
65	160	60	1800	0.44	85	73
66	180	80	900	0.38	87	72
67	160	60	1800	0.38	87	74
68	180	60	1800	0.38	88	75
69	200	60	1800	0.44	87	76
70	200	80	1800	0.3	90	74
71	160	80	900	0.44	89	80
72	180	80	1800	0.3	92	77
73	200	80	900	0.44	90	81
74	180	80	900	0.44	91	82
75	160	80	1800	0.3	94	81
76	200	80	1800	0.38	92	86
77	160	80	1800	0.38	94	88
78	180	80	1800	0.38	95	88
79	180	80	1800	0.44	94	90
80	160	80	1800	0.44	95	92
81	200	80	1800	0.44	97	94

Appendix 7: Hyperparameter Tuning Results: DQN in Corridor Network with 6,000 Traffic Demand and Incident

ID	-batch	Discount Factor	Green Duration	Learning Rate	-lre	Number of Hidden Layers	-nreplay	Temporal Difference Steps	Update Frequency	-updates	Mean (seconds)	Standard Deviation (seconds)
1	128	0.1	6	1.00E-04	1.00E-07	6	800000	2	128	5000	68	55
2	128	0.1	6	1.00E-05	1.00E-07	6	800000	2	128	5000	67	86
3	128	0.5	6	1.00E-04	1.00E-07	6	800000	2	128	5000	66	90
4	128	0.9	6	1.00E-04	1.00E-07	6	800000	2	128	5000	73	90
5	128	0.9	6	1.00E-05	1.00E-07	6	800000	2	128	5000	77	96
6	128	0.7	6	1.00E-03	1.00E-07	6	800000	2	128	5000	70	107
7	128	0.1	6	1.00E-03	1.00E-07	6	800000	2	128	5000	71	111
8	128	0.7	6	1.00E-04	1.00E-07	6	800000	2	128	5000	72	116
9	128	0.5	6	1.00E-05	1.00E-07	6	800000	2	128	5000	72	117
10	128	0.9	6	0.001	1.00E-07	6	800000	2	128	5000	80	118
11	128	0.5	6	1.00E-03	1.00E-07	6	800000	2	128	5000	76	129
12	128	0.7	6	1.00E-05	1.00E-07	6	800000	2	128	5000	79	139

Appendix 8: Hyperparameter Tuning Results: Max-pressure in Corridor Network with 6,000 Traffic Demand and Incident

ID	Green Duration	Mean (seconds)	Standard Deviation (seconds)
1	5	66	50
2	23	72	73
3	18	71	87
4	17	73	93
5	9	76	93
6	22	74	97
7	21	75	97
8	11	76	102
9	19	78	108
10	14	78	112
11	20	80	110
12	16	77	116
13	7	78	117
14	15	79	116
15	12	80	116
16	13	81	118
17	25	82	118
18	10	82	122
19	8	81	126
20	6	82	135
21	24	85	135

Appendix 9: Hyperparameter Tuning Results: Uniform in Corridor Network with 6,000 Traffic Demand and Incident

ID	Green Duration (seconds)	Mean (seconds)	Standard Deviation (seconds)
1	14	91	92
2	24	99	84
3	17	98	106
4	5	109	96
5	16	96	110
6	13	98	124
7	18	103	120
8	9	99	125
9	11	98	127
10	23	106	119
11	10	99	127
12	15	104	134
13	20	107	134
14	22	112	146
15	25	115	143
16	12	105	154
17	19	112	152
18	7	113	157
19	8	112	158
20	21	116	159
21	6	125	161

Appendix 10: Hyperparameter Tuning Results: Webster's in Corridor Network with 6,000 Traffic Demand and Incident

ID	Max Cycle Length (seconds)	Min Cycle Length (seconds)	Time Interval (seconds)	Satuation Flow Rate	Mean (seconds)	Standard Deviation (seconds)
1	180	40	1800	0.44	77	71
2	160	40	600	0.3	87	66
3	160	40	600	0.38	78	83
4	180	40	600	0.38	81	88
5	180	80	1800	0.3	83	87
6	160	60	600	0.38	83	91
7	180	80	900	0.3	85	94
8	200	40	900	0.3	81	100
9	180	60	900	0.3	82	100
10	200	60	900	0.38	82	101
11	160	60	1800	0.3	85	101
12	180	40	900	0.38	83	103
13	160	80	1800	0.38	86	102
14	160	60	1800	0.38	84	105
15	180	80	900	0.38	87	103
16	200	40	900	0.38	83	107
17	200	80	1800	0.3	86	104
18	200	80	1800	0.44	87	103
19	200	80	1800	0.38	86	105
20	160	60	900	0.3	86	107
21	200	60	600	0.3	86	107
22	200	40	600	0.3	88	106
23	200	60	1800	0.3	88	110
24	180	40	1800	0.3	86	113
25	200	60	900	0.44	86	115
26	200	80	900	0.44	87	114
27	160	60	600	0.3	88	114
28	180	60	1800	0.44	86	116

29	200	60	600	0.44	86	116
30	160	40	1800	0.38	86	117
31	160	80	1800	0.44	89	114
32	180	80	900	0.44	88	115
33	160	80	900	0.3	91	114
34	180	60	600	0.38	89	116
35	180	60	1800	0.3	87	119
36	180	40	900	0.44	88	120
37	160	40	900	0.38	88	121
38	180	80	600	0.44	90	120
39	200	60	1800	0.38	87	123
40	180	60	600	0.44	90	121
41	200	80	600	0.44	89	122
42	160	40	1800	0.3	89	123
43	160	40	900	0.44	89	124
44	200	40	1800	0.3	89	124
45	160	40	1800	0.44	88	126
46	180	80	1800	0.38	93	121
47	160	60	900	0.38	89	126
48	160	80	1800	0.3	90	125
49	200	60	600	0.38	90	125
50	180	40	600	0.3	90	126
51	180	60	600	0.3	90	126
52	200	40	1800	0.38	89	127
53	200	60	900	0.3	90	126
54	160	40	900	0.3	92	126
55	200	80	600	0.38	91	127
56	200	80	900	0.38	91	127
57	160	60	900	0.44	91	129
58	180	40	900	0.3	91	129
59	200	40	600	0.38	89	131
60	180	60	1800	0.38	89	132
61	160	60	1800	0.44	89	133

62	180	80	600	0.3	92	130
63	160	60	600	0.44	91	132
64	200	40	1800	0.44	90	133
65	180	60	900	0.38	91	133
66	200	80	600	0.3	95	130
67	160	80	900	0.38	93	133
68	160	80	600	0.3	95	134
69	200	40	600	0.44	91	138
70	160	80	600	0.38	94	137
71	180	80	1800	0.44	94	140
72	200	60	1800	0.44	94	142
73	160	80	900	0.44	96	142
74	160	40	600	0.44	94	145
75	160	80	600	0.44	97	142
76	200	40	900	0.44	96	143
77	200	80	900	0.3	100	145
78	180	40	600	0.44	97	149
79	180	80	600	0.38	100	149
80	180	40	1800	0.38	97	155
81	180	60	900	0.44	100	158

Appendix 11: Hyperparameter Tuning Results: DQN in 2x2 Grid with 6,000 Traffic Demand and Incident

ID	-batch	Discount Factor	Green Duration	Learnig Rate	-lre	Number of Hidden Layers	-nreplay	Temporal Difference Steps	Update Frequency	-updates	Mean (seconds)	Standard Deviation (seconds)
1	128	0.5	6	1.00E-05	1.00E-07	6	1600000	2	128	5000	99	169
2	128	0.5	6	1.00E-03	1.00E-07	6	1600000	2	128	5000	107	203
3	128	0.9	6	1.00E-04	1.00E-07	6	1600000	2	128	5000	118	231
4	128	0.5	6	1.00E-04	1.00E-07	6	1600000	2	128	5000	114	238
5	128	0.9	6	1.00E-05	1.00E-07	6	1600000	2	128	5000	138	216
6	128	0.7	6	1.00E-05	1.00E-07	6	1600000	2	128	5000	116	251
7	128	0.7	6	1.00E-04	1.00E-07	6	1600000	2	128	5000	131	290
8	128	0.9	6	1.00E-03	1.00E-07	6	1600000	2	128	5000	148	283
9	128	0.7	6	1.00E-03	1.00E-07	6	1600000	2	128	5000	148	307

Appendix 12: Hyperparameter Tuning Results: Max-pressure in 2x2 Grid Network with 6,000 Traffic Demand and Incident

ID	Green Duration	Mean (seconds)	Standard Deviation (seconds)
1	24	97	101
2	23	105	132
3	6	104	135
4	9	106	135
5	16	109	144
6	5	108	155
7	17	111	152
8	19	109	169
9	15	110	169
10	12	113	172
11	8	118	193
12	7	120	209
13	11	125	220
14	18	126	234
15	25	129	231
16	21	133	235
17	14	130	240
18	13	130	244
19	20	140	283
20	10	155	327
21	22	155	336

Appendix 13: Hyperparameter Tuning Results: Uniform in 2x2 Grid Network with 6,000 Traffic Demand and Incident

ID	Green Duration (seconds)	Mean (seconds)	Standard Deviation (seconds)
1	8	110	110
2	23	129	116
3	10	118	143
4	16	129	137
5	20	130	138
6	13	127	146
7	15	129	144
8	12	125	152
9	24	137	140
10	18	133	145
11	25	138	140
12	19	133	146
13	21	138	157
14	14	139	180
15	22	146	178
16	9	137	199
17	17	152	245
18	7	141	271
19	11	150	263
20	5	158	272
21	6	163	270

Appendix 14: Hyperparameter Tuning Results: Webster's in 2x2 Grid Network with 6,000 Traffic Demand and Incident

ID	Max Cycle Length (seconds)	Min Cycle Length (seconds)	Time Interval (seconds)	Saturation Flow Rate	Mean (seconds)	Standard Deviation (seconds)
1	180	60	600	0.38	105	110
2	200	80	600	0.38	109	111
3	160	60	900	0.38	106	120
4	160	40	1800	0.3	106	125
5	160	80	600	0.38	111	125
6	180	80	600	0.44	111	126
7	160	80	900	0.44	111	127
8	180	80	1800	0.3	110	129
9	160	40	900	0.38	109	135
10	200	80	900	0.44	114	131
11	160	40	600	0.3	111	137
12	180	40	900	0.44	110	139
13	160	60	900	0.3	111	139
14	200	80	1800	0.44	113	137
15	180	40	600	0.38	112	143
16	160	40	600	0.44	113	143
17	160	40	1800	0.44	113	143
18	180	80	900	0.44	118	142
19	200	40	600	0.44	113	149
20	200	80	600	0.44	114	151
21	200	40	900	0.3	112	155
22	160	60	1800	0.44	111	158
23	160	40	600	0.38	116	154
24	180	40	600	0.3	119	160
25	180	60	900	0.38	116	167
26	200	60	600	0.38	116	169
27	160	40	1800	0.38	120	166
28	200	80	1800	0.38	118	168

29	180	60	1800	0.44	113	175
30	160	60	1800	0.3	122	168
31	180	80	900	0.3	123	168
32	160	40	900	0.3	119	180
33	180	60	1800	0.3	123	177
34	180	60	1800	0.38	119	181
35	200	60	900	0.38	119	181
36	180	60	600	0.44	119	184
37	160	60	900	0.44	118	187
38	160	40	900	0.44	119	187
39	180	80	600	0.3	128	179
40	200	80	600	0.3	125	182
41	180	40	900	0.38	120	188
42	180	40	1800	0.3	120	191
43	200	60	1800	0.44	124	192
44	160	80	900	0.38	126	192
45	200	60	1800	0.3	124	196
46	160	60	600	0.3	125	196
47	200	60	1800	0.38	125	197
48	180	80	600	0.38	124	201
49	200	40	600	0.3	122	204
50	200	60	600	0.3	126	203
51	180	40	600	0.44	128	204
52	160	60	600	0.44	123	210
53	180	80	1800	0.44	127	207
54	180	40	900	0.3	131	207
55	200	80	1800	0.3	127	211
56	180	60	900	0.44	127	212
57	180	60	900	0.3	133	213
58	200	60	600	0.44	130	216
59	160	60	1800	0.38	128	220
60	160	80	600	0.3	129	221
61	200	60	900	0.3	130	221

62	200	60	900	0.44	135	219
63	200	80	900	0.3	130	225
64	180	40	1800	0.38	129	231
65	180	80	900	0.38	135	226
66	180	60	600	0.3	133	234
67	160	80	1800	0.44	136	234
68	160	80	900	0.3	136	238
69	200	40	600	0.38	136	242
70	160	80	600	0.44	142	239
71	180	40	1800	0.44	139	248
72	200	40	900	0.44	138	253
73	200	40	1800	0.44	140	252
74	200	40	900	0.38	139	257
75	200	40	1800	0.38	141	256
76	160	80	1800	0.38	150	255
77	200	80	900	0.38	144	281
78	200	40	1800	0.3	149	277
79	180	80	1800	0.38	151	279
80	160	80	1800	0.3	153	290
81	160	60	600	0.38	151	300

Appendix 15: Hyperparameter Tuning Results For DQN in 2x2 Grid Network with 6,000 Traffic Demand and No Incident

ID	-batch	Discount Factor	Green Duration	Learning Rate	-lre	Number of Hidden Layers	-nreplay	Temporal Difference Steps	Update Frequency	-updates	Mean (seconds)	Standard Deviation (seconds)
1	128	0.5	6	1.00E-04	1.00E-07	6	1600000	2	128	5000	68	30
2	128	0.5	6	1.00E-05	1.00E-07	6	1600000	2	128	5000	69	30
3	128	0.7	6	1.00E-04	1.00E-07	6	1600000	2	128	5000	69	30
4	128	0.5	6	1.00E-03	1.00E-07	6	1600000	2	128	5000	70	31
5	128	0.7	6	1.00E-05	1.00E-07	6	1600000	2	128	5000	70	31
6	128	0.7	6	1.00E-03	1.00E-07	6	1600000	2	128	5000	71	32
7	128	0.9	6	1.00E-04	1.00E-07	6	1600000	2	128	5000	71	32
8	128	0.9	6	1.00E-03	1.00E-07	6	1600000	2	128	5000	81	42
9	128	0.9	6	1.00E-05	1.00E-07	6	1600000	2	128	5000	89	53

Appendix 16: Hyperparameter Tuning Results For Max-pressure in 2x2 Grid Network with 6,000 Traffic Demand and No Incident

ID	Green Duration	Mean (seconds)	Standard Deviation (seconds)
1	5	80	36
2	6	81	37
3	7	81	37
4	8	83	38
5	9	85	39
6	17	84	40
7	20	84	40
8	16	85	41
9	18	85	41
10	19	85	41
11	21	85	41
12	23	85	41
13	10	86	41
14	11	86	41
15	22	85	42
16	24	85	42
17	25	85	42
18	12	86	42
19	14	86	42
20	15	86	42
21	13	87	42

Appendix 17: Hyperparameter Tuning Results For Uniform in 2x2 Grid Network with 6,000 Traffic Demand and No Incident

ID	Green Duration (seconds)	Mean (seconds)	Standard Deviation (seconds)
1	7	90	40
2	9	91	41
3	10	93	41
4	8	93	42
5	11	96	44
6	12	97	45
7	6	96	48
8	13	100	47
9	14	103	49
10	5	103	52
11	15	105	51
12	16	107	53
13	17	108	54
14	18	109	55
15	19	109	55
16	20	110	57
17	21	112	58
18	22	113	59
19	23	114	60
20	24	115	62
21	25	117	63

Appendix 18: Hyperparameter Tuning Results For Websters in 2x2 Grid Network with 6,000 Traffic Demand and No Incident

ID	Max Cycle Length (seconds)	Min Cycle Length (seconds)	Time Interval (seconds)	Satuation Flow Rate	Mean (seconds)	Standard Deviation (seconds)
1	180	40	1800	0.38	87	39
2	180	40	1800	0.44	87	39
3	160	40	600	0.38	87	40
4	160	40	600	0.44	87	40
5	160	40	900	0.38	87	40
6	180	40	600	0.44	87	40
7	180	40	900	0.38	87	40
8	180	40	900	0.44	87	40
9	180	40	1800	0.3	87	40
10	200	40	600	0.38	87	40
11	200	40	900	0.44	87	40
12	200	40	1800	0.38	87	40
13	200	40	1800	0.44	87	40
14	160	40	1800	0.3	88	40
15	160	40	1800	0.38	88	40
16	160	40	1800	0.44	88	40
17	200	40	1800	0.3	88	40
18	160	40	900	0.3	88	41
19	160	40	900	0.44	88	41
20	160	60	1800	0.44	88	41
21	180	40	600	0.3	88	41
22	180	40	600	0.38	88	41
23	180	40	900	0.3	88	41
24	200	40	600	0.3	88	41
25	200	40	600	0.44	88	41
26	200	40	900	0.3	88	41
27	200	40	900	0.38	88	41
28	200	60	900	0.38	89	41

29	200	60	1800	0.3	88	42
30	200	60	1800	0.44	88	42
31	160	40	600	0.3	89	42
32	160	60	600	0.38	89	42
33	160	60	600	0.44	89	42
34	160	60	900	0.44	89	42
35	160	60	1800	0.3	89	42
36	180	60	600	0.38	89	42
37	180	60	600	0.44	89	42
38	180	60	900	0.3	89	42
39	180	60	900	0.44	89	42
40	180	60	1800	0.3	89	42
41	180	60	1800	0.38	89	42
42	180	60	1800	0.44	89	42
43	200	60	600	0.38	89	42
44	200	60	900	0.3	89	42
45	200	60	1800	0.38	89	42
46	160	60	900	0.3	89	43
47	160	60	900	0.38	89	43
48	160	60	1800	0.38	89	43
49	180	60	900	0.38	89	43
50	200	60	600	0.3	89	43
51	200	60	900	0.44	89	43
52	160	60	600	0.3	90	43
53	180	60	600	0.3	90	43
54	200	60	600	0.44	90	43
55	160	80	600	0.3	93	46
56	160	80	900	0.3	93	46
57	160	80	1800	0.3	93	46
58	160	80	1800	0.38	93	46
59	160	80	1800	0.44	93	46
60	180	80	600	0.3	93	46
61	180	80	1800	0.3	93	46

62	180	80	1800	0.38	93	46
63	180	80	1800	0.44	93	46
64	200	80	900	0.38	93	46
65	200	80	900	0.44	93	46
66	200	80	1800	0.3	93	46
67	200	80	1800	0.38	93	46
68	200	80	1800	0.44	93	46
69	180	80	900	0.38	93	47
70	160	80	600	0.38	94	47
71	160	80	600	0.44	94	47
72	160	80	900	0.38	94	47
73	160	80	900	0.44	94	47
74	180	80	600	0.38	94	47
75	180	80	600	0.44	94	47
76	180	80	900	0.3	94	47
77	180	80	900	0.44	94	47
78	200	80	600	0.3	94	47
79	200	80	600	0.38	94	47
80	200	80	600	0.44	94	47
81	200	80	900	0.3	94	47