SecureTLC: Secure and Two-level Traffic Signal Control for Intelligent Transportation System

By

Chia-Cheng Yen

Dissertation

Submitted in partial satisfaction of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Office of Graduate Studies

of the

University of California

Davis

Approved:

Dipak Ghosal, Chair

Michael Zhang

Chen-Nee Chuah

Committee in Charge

2022

To my family and people in Davis:

Ph.D. programs are an enduring and learning process from which we unceasingly reappraise ourselves and surpass limits. In the past, in Taiwan, what I thought is unachievable task which can be overcome in this challenging environment by continually attempting and breakthrough ourselves. Through the process, I understand methodology of research and how to effectively approach problems. The beginning of my Ph.D. journey started from a preschooler's commitment to his father. I still vividly remember the time I committed you to complete my Ph.D. program. My father is the first person who inspired me to study abroad. Thank you for sharing a series of interesting stories and amazing experiences of studying in the states, which broaden my horizon in terms of living my life and developing my career abroad. Thank my family for being supportive of my decision and tolerant of my willfulness.

For people who have never stayed abroad, no doubt living in a different country is challenging. We have to acquire knowledge and developed new skills for our research while adapting ourselves to new environments. I want to particularly thank my advisor, Dipak, who is an amiable professor. He is very supportive of my imaginative ideas and willing to provide me enough space to explore. I always enjoy the unchained atmosphere in our lab when doing research. I also want to thank my co-advisors, professor Michael and Chen-Nee. I always learn interdisciplinary knowledge which I would have never had a chance to touch and punctilious attitude toward research from you.

Moreover, I want to thank Janet who is a speaking and writing lecturer in UWP. I appreciate your lectures on and practice in speaking and writing with me. My English proficiency has been significantly improved since I took your classes. This is attributed to the training which I have received from you.

Finally, along the Ph.D. journey, there were unexpected encounters and unforgettable memories. For people who have ever help me, I am very thankful because you make me believe in the good side of human nature.

Abstract

With the advanced wireless technologies and emerging machine learning techniques, there is an increasing trend of using both wireless communication and machine learning along with new platooning control as well as traffic signal control (TSC) algorithms to leverage and accommodate connected and autonomous vehicles (CAVs) for achieving high throughput and low latency traffic flow, lowering accidents, and reducing emissions in urban transportation. However, this development leads to highly connected wireless environments that would increase the potential for cyber-attacks on TSC. These attacks can undermine the benefits of new TSC algorithms. For example, an adversary can first perform reconnaissance to gain understanding of how the TSC operates under different traffic arrival rates, and then, choose a fixed number of vehicles to be attack vehicles to get scheduling priority and/or to create traffic congestion in one intersection which can spread to the entire network. These attacks can compromise TSC systems and significantly increase traffic delay and make TSC completely ineffective. Furthermore, machine learning based TSC approaches also lead to several issues: complexity, dimensionality, and convergence. They require a long period of time and a significant amount of training data to gain insights into improvement of traffic throughput and latency. They are not applicable to problems with a large state size. In addition, some of the approaches adopting neural networks (NNs) as a function approximator leads to a convergence issue due to the non-linearity of NNs. In this work, we have proposed the SecureTLC to address the security and machine learning issues caused by the novel technologies for modern intelligent transportation systems (ITS).

As our first work, we have investigated security vulnerabilities in four different backpressure-based (BP-based) traffic control algorithms: 1) Delay-based BackPressure Control (DBPC), 2) Queue-based BackPressure Control (QBPC), 3) Sum-of-delay-based BackPressure Control (SBPC), and 4) Hybrid-based BackPressure Control (HBPC) which is a combination of the delay-based and queue-based. Their performance has been compared when they are under two misinformation attacks: 1) time spoofing attack and 2) ghost vehicle attack. The time spoofing attack is a type of falsified data attack in which attack vehicles arriving at an intersection alter their arrival times. In the ghost vehicle attack, attack vehicles intentionally disconnect the wireless communication and thereby hide from the TSC. We have shown that the misinformation sent by attack vehicles can influence the signal phases determined by BP-based traffic control algorithms. We have considered an adversary that determines a set of arriving vehicles to be attack vehicles from many candidate sets (attack strategies) in order to maximize the number of disrupted signal phases. We have shown that by formulating the problem as a 0/1 Knapsack problem, the adversary can explore the space of attack strategies and determine the optimal strategy that maximally compromises the performance in terms of average delay and fairness. Through detailed simulation analyses we have also shown that while the DBPC has better fairness, it is more vulnerable to time spoofing attacks than the other schemes. On the other hand, the QBPC, which determines traffic signal phases using the aggregate queue information, suffers from a higher intensity of ghost vehicle attacks. We have examined the drawbacks of both the DBPC and QBPC under different attack scenarios for different traffic patterns including homogeneous and non-homogeneous vehicle arrivals at an isolated intersection. We have proposed two protection mechanisms against the attacks, namely, an auction-based protection algorithm (APA) and a hybrid-based protection algorithm (HPA). Using simulation analysis, we have shown that both protection schemes are able to mitigate the impacts of the time spoofing and ghost vehicle attacks.

In our second work, we have studied the impact of AI-enabled platooning control on fuel consumption. Nowadays, vehicular emissions and traffic congestion have been deteriorated by highly urbanization. The worsen traffic burdens drivers with a higher cost and longer time on driving, and exposures pedestrians to unhealthy emissions such as PM, NO_x, SO_2 and greenhouse gases. In response to these issues, CAV which enables information sharing between drivers and infrastructure was proposed. With advanced wireless technologies offering extremely low latency and CAV, platooning control can be realized to improve traffic efficiency and safety. However, conventional platooning control algorithms require complex computations and hence, are not a perfect candidate when applying to real-time operations. To overcome this issue, this chapter focuses on designing an innovative learning framework for platooning control capable of reducing the fuel consumption by the four basic platoon manipulations, e.g., split, acceleration, deceleration, and no-op. We integrated reinforcement learning (RL) with NNs to be able to model non-linear relationships between inputs and outputs for a complex application. The experimental results reveal a decreasing trend of the fuel usage and a growing trend of the reward. They demonstrate that the proposed DRL platooning control optimizes the fuel consumption by fine-tuning speeds and sizes of platoons.

In our third work, we have applied advanced machine learning techniques to TSC. A recent report has illustrated that an urban road network equipped with an advanced TSC system is capable of reducing average sojourn time of vehicles, traffic collisions, and traffic delay while at the same time decreasing energy consumption and improving parking. Modern TSC systems which leverage advanced machine learning techniques and scheduling algorithms have been a promising topic in recent years. RL has been broadly adopted in many areas, owing to its flexibility in combining an assortment of architectures and neural networks that assists RL agents in learning from a complex environment. Due to its compatibility with numerous existing deep learning techniques, RL can be integrated into a complicated deep learning framework which is extendable and powerful for many deterministic and nondeterministic applications. We proposed a deep RL (DRL) framework which has high extendability and flexibility by incorporating various RL-based algorithms and techniques. We shown that learning performance can be significantly improved by applying an on-policy temporal-difference (TD) learning method, SARSA, to the traffic signal control problem for a traffic network consisting of multiple intersections. The proposed traffic flow maps (TFMs) as input states are formulated by traffic flows at every time slot in the network. These states are fed into a non-linear function approximator to output the optimal action. In order to process TFMs, we combined a convolutional neural network, a dueling network architecture which enhances evaluations of action values when there is a huge action space, and a memory replay that breaks the strong temporal correlations among states. Moreover, we compared learning performance among DQN, 3DQN, DSARSA, and 2DSARSA as well as provide a thorough analysis on their weaknesses and strengths. To the best of our knowledge, this is the first work discussing and evaluating learning effects of different DRL-based agents.

In our fourth work, as an extended work, we have proposed a novel architecture exploiting both the global and local information to further improve the performance of TSC in the third work. We use the proposed TFMs as input states to represent the global information for a DRL agent and pressure-of-the-lanes (POL) to represent the local information as pressure metric for a number of backpressure (BP) controllers. We define a combinatorial reward function using the power metric which maximizes the overall throughput and minimizes the average end-to-end delay of a network. We proposed a hybrid-based two-level control (TLC) architecture which has high extendability and flexibility by enabling a collaboration between the centralized DRL agent using TFM and the decentralized BP controllers using POL in a hierarchical architecture to determine traffic signal phases that optimizes the network throughput and latency. We show that learning performance can be significantly improved by the collaboration between the two levels for a traffic network of multiple intersections. Moreover, we compare learning performance of the proposed TLC with BP, DQN, 3DQN, DSARSA, and 2DSARSA as well as provide a thorough analysis on their weaknesses and strengths.

As our future work, first, we plan to analyze the impacts of malicious attacks on TSC equipped with the DRL-based agent. From our experiments, we have shown that DRL performs well for a traffic network with multiple intersections. We expect to explore the tolerance of DRL-based agent when it is under adversarial attacks. How many attacks can it tolerate? We want to examine the resilience of DRL-based agent as well. How well can it recover after encountering attacks? Second, as we start to consider TSC problems for larger areas, increases in dimensions of states and actions are foreseeable. High dimensional states and actions causing difficulty in learning will be a worthy research topic. We plan to design a two-level control architecture which is capable of reducing the dimension of actions by transferring some workload from a centralized agent to local controllers.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Human population has exponentially increased in this century. Due to an unbalanced development among regions, resources are extremely concentrated on urban cities into which must support increasingly larger number of commutes move everyday. Nowadays, 55% of the human population in our world lives in urban cities. The population in urban areas is predicted to increase to 68% by 2025 [2]. The website called *Our World in Data* [82] presented a series of data visualization related urbanization across the world and predictions of future trends with respect to urban shares, density, and population sizes. According to their predictions of urban and rural populations Fig. 1.3, our world is expected to become more urbanized in 2050. Fig. 1.2 presents the share of the population living in urbanized areas for highly developed and high-population countries from 1700 to 2000. Fig. 1.1 shows a current overview of urbanization across the world. As can be observed, most countries are predominantly urban, especially for high-income areas such as Europe, the U.S., and Japan. For example, over 82% of population in the U.S. lives in urban cities. 91.7% of population in Japan lives in urban cities. This phenomenon is resulting into more traffic congestion, a longer average delay, and high emissions in urban cities. It is also pushing for modern transportation systems. The annual congestion cost for each driver reported by INRIX [22] was 97 hours and $\$1,348$ in 2018 in the United States (US). Greenhouse gas emissions [1] contributed by transportation were 27% in 2015 and 29% in 2017. Medium-duty (including light-duty) and heavy-duty vehicles accounted for 59% and 23% of transportation greenhouse gas emissions in 2017, respectively.

In response to ever-growing traffic demands, the emergence of innovative traffic management systems also known as intelligent transportation system (ITS) is inevitable. ITS has been extensively

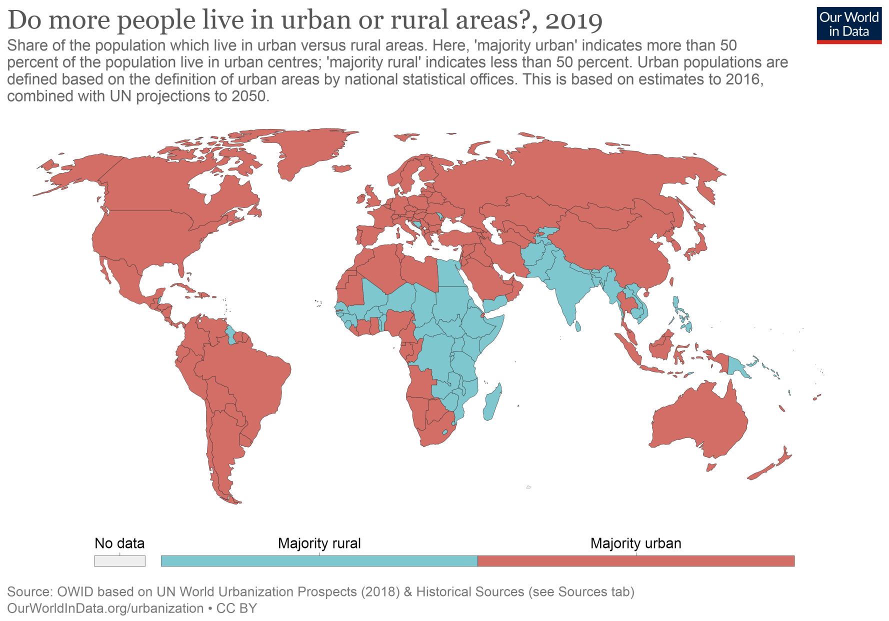

Do more people live in urban or rural areas?, 2019
Share of the population which live in urban versus rural areas. Here, 'majority urban' indicates more than 50 percent of the population live in urban centres; 'majority rural' indicates less than 50 percent. Urban populations are defined based on the definition of urban areas by national statistical offices. This is based on estimates to 2016, combined with UN projections to 2050.
Our World in Data
No data
Majority rural
Majority urban
Source: OWID based on UN World Urbanization Prospects (2018) & Historical Sources (see Sources tab)
OurWorldInData.org/urbanization • CC BY

Figure 1.1: The human population lives in urban versus rural areas in 2019 where 'majority urban' indicates more than 50% of the population lives in urban centres and 'majority rural' indicates less than 50%.

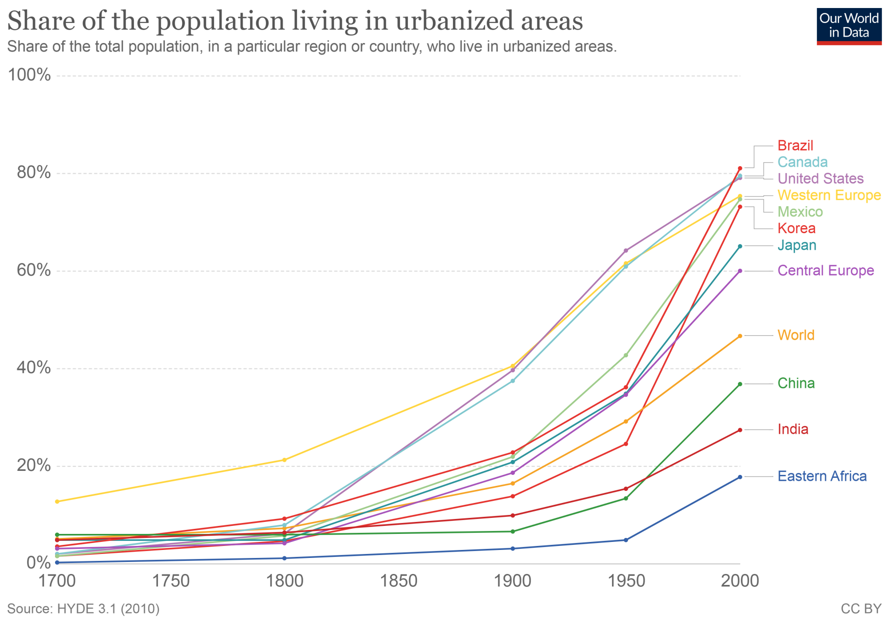


Figure 1.2: Urbanization from 1700s to 2000 over developed and high-population countries

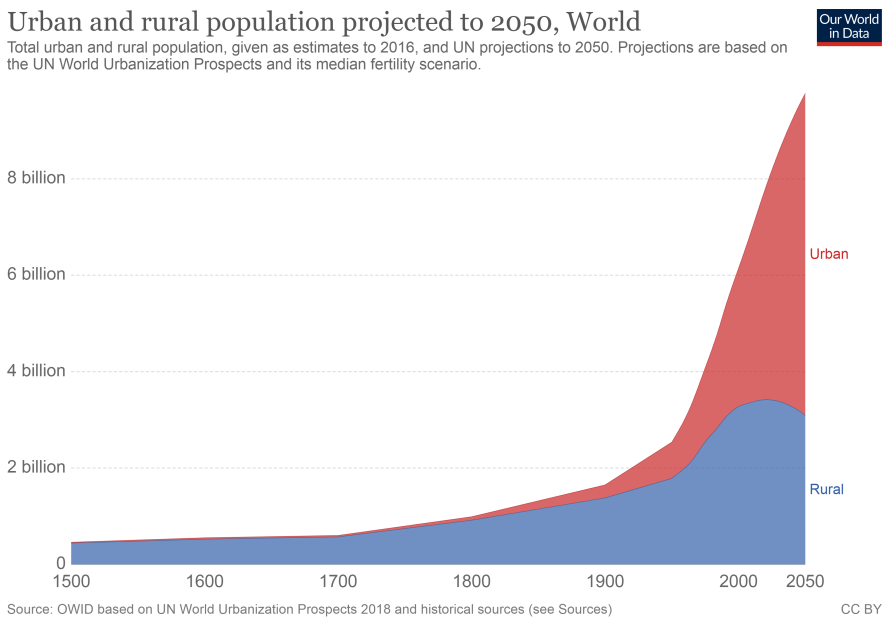


Figure 1.3: Trends of urban and rural populations projected to 2050

developed, especially in Japan, the US, and European Union (EU), as a popular research topic since it was first proposed by the US. ITS is a future technology that is utilized to develop transportation systems with reliability, safety, and efficiency for the next generation. The underlying idea of ITS is to form a highly connected environment where information about pedestrians, vehicles as well as facilities by sensors can be shared unhindered, and the information can be integrated with high speed communication technologies to provide comparatively robust services to users. In recent years, with highly evolved wireless technologies, communications now are much more stable and adaptable to heterogeneous devices such that a variety of applications related to ITS can be realized for efficiency and safety. Since service-based and data integrating systems have become vital in smart cities, Barba *et al.* [11] proposed a framework for a smart city where intelligent traffic lights (ITLs) set up in crossroads are capable of collecting flow density, sending warning messages, and updating statistics. ITLs themselves formulate a network which enables information sharing and estimating statistics for an entire city. This framework aims at providing services based on data collected by the ITL network. Younes *et al.* [123] emphasized more on the intelligent traffic light control (ITLC) and arterial traffic light (ATL) algorithms. ITLC is designed for an isolated intersection and ATL is designed for a network of intersections. Their proposed algorithms are able to schedule phases effectively based on the traffic characteristics. In addition to conventional ITS that is driven by multiple state-of-the-art technologies, Zhang *et al.* [125] proposed a data-driven ITS (D^2ITS) which is capable of fully extracting valuable information from heterogeneous resources such as video, inductive-loop detectors, laser radar, as well as Global Positioning System (GPS), and applying learning algorithms to these data to improve the performance of traffic signal control. Moreover, the recent research on ITS [37] unveiled that proactive control could be a potential development over conventional reactive control.

To develop a securer and smarter system for the next generation, security vulnerabilities and machine learning issues need to be addressed in the design of traffic signal control (TSC) and platooning control (PC) to improve the overall ITS performance. TSC has been a well-known and challenging research topic for complicated traffic networks in the real world [27,64,76,116–118,124,127]. A considerable research work has been dedicated to this area to improve the performance of the signal control. Pre-timed systems which schedule each phase by an equivalent time slot performed well under static traffic flows. However, the fixed timing pattern becomes a limitation to satisfy highly dynamic traffic flows. Bell and Bretherton [12] demonstrated that the obsolete timing plan degrades performance by 3% per year. Hence, the ability to adapt to non-static traffic flows needs to be considered while designing TSC algorithms. Updating timing pattern accordingly and making time slots adjustable enable ITS to be more flexible and reliable. The idea of adaptive TSC (ATSC) was proposed and has been widely applied to traffic scheduling for achieving throughput optimality [16, 64, 76]. Pandit *et al.* [76] formulated the traffic scheduling problem as a job scheduling problem. The arriving vehicles are divided into several platoons which can be regarded as jobs with conflicts. They proposed an algorithm that schedules platoons in a conflict-free manner to minimize delays at the intersection based on the collected speed and position data of vehicles. Cai *et al.* [16] proposed a learning framework that incorporates dynamic programming which is solved using the Bellman equation to achieve adaptability for traffic scheduling. On the other hand, connected and autonomous vehicles (CAVs) [98] enable wireless communications among drivers, road side units (RSUs) and controllers to promote connectivity, automation, throughput, efficiency, and safety for public transportation [55]. With a fully connected traffic environment, PC can be fulfilled by sharing all information of vehicles. PC is a technique that organizes a traffic flow into multiple groups, convoys, or platoons with close-following vehicles also known as road trains in order to increase the overall capacity of roads and reduce fuel consumption and emissions [68] in the traffic network. By forming vehicles into road trains, the overall throughput can be improved because gaps among CAVs are minimized. In addition, fuel consumption and emissions can be reduced due to lower air drag and speed variation for each CAV in a platoon [68]. Platooning can be considered as a effective control strategy for heavy-duty vehicles (HDVs) as well as a promising solution to reduce fuel consumption in HDVs [5, 14, 97].

In this dissertation, we focus on the TSC and PC in terms of security and performance. First, owing to some weaknesses existing in the technologies mentioned above, the integration of them in a modern ITS would not be seamlessly combined and might have vulnerabilities caused by ill-defined data structures, poor-designed architectures, poorly maintained functionalities as well as insecure communication channels. Modern ITS with TSC and CAVs will be exposed to cyber-attacks [26, 77] if designers fail to consider security issues when designing systems. These security vulnerabilities will provide great opportunities to attackers who attempt to compromise TSC systems to disrupt traffic flow and achieve their own benefit. We study two types of misinformation attacks which aim at disrupting signal phases scheduled by four different BP-based traffic control algorithms and our contributions are as follows. We propose a threat model [121] in which an advanced persistent adversary can formulate attack strategies as a 0/1 Knapsack problem and determine the optimal strategy using a branch-and-bound algorithm. By the model, an adversary knows when and where to deploy attack vehicles in arrivals to formulate the optimal attack strategy that maximizes the number of total disrupted phases. For the attack strategies formulated by the model, we analyze the hidden relationship between the number of disrupted phases and delay/fairness by both offline and online simulations. We propose protection mechanisms against these attacks. Second, due to the traffic congestion and emissions, performance improvement in TSC and PC is another imperative topic because their performance can directly influence traffic throughput and fuel consumption which are contributing factors to the issues. Toward this end, we consider deep reinforcement learning (DRL) and adopt DRL techniques such as the dueling neural architecture and experience memory replay to improve TSC and PC regarding traffic throughput, traffic latency, and fuel consumption. The contributions to DRL-based TSC and PC can be summarized as follows. We propose a DRL framework [120] that integrates the deep learning, dueling architecture, and experience replay memory for both TSC and PC to optimize the traffic throughput, traffic latency, and fuel consumption. We propose traffic flow map (TFM) and arrival timing vector (ATV) as states for TSC and PC, respectively to enhance learning performance of DRL-based agents. Based on the DRL framework, we propose a hybrid-based TLC architecture which enables a collaboration between decentralized BP controllers and a centralized DRL agent. To the best of our knowledge, this is the first research combining BP with DRL on TSC. Our results unveil that the collaboration between centralized and decentralized controllers in the two levels substantially improves the network throughput and latency by exploiting the insights of using the local information. The proposed TLC performs better and converges faster than DQN [72], 3DQN [58], DSARSA [128], and 2DSARSA [120].

The rest of this proposal is organized as follows.

1. In Chapter 2, we include background knowledge of backpressure based scheduling algorithms, reinforcement learning, and deep reinforcement learning.
2. In Chapter 3, we address security vulnerabilities of TSC using BP-based algorithms. We examined their reliability when under attack.
3. In Chapter 4, we propose a DRL-based PC to optimize fuel consumption by fine-tuning speeds and sizes of platoons.
4. In Chapter 5, we propose a DRL framework for integrating RL-based TSC, deep learning, and several existing techniques which enhance learning performance.
5. In Chapter 6, we proposed a two-level optimization framework integrating the DRL-based TSC with backpressure algorithms for improving the performance by pure DRL-based TSC.
6. In Chapter 7, we propose the future research work.

Chapter 2

Related Work

2.1 Backpressure-based (BP-based) Traffic Signal Control

The backpressure (BP) routing algorithm [93] was originally developed to optimize the throughput of a queuing network over a multi-hop radio network with random arrival rates. The study in [93] determined the stability region that defines the boundaries of arrival and service rates within which feasible policies exists such that the network is stable. The study proposed an optimal policy that achieves maximum throughput. The original work on backpressure control has been extended in the domain of wireless communication networks [47, 73, 74, 90].

The concept of the backpressure was first applied to traffic signal control in [116]. They summarized some important definitions for network stability to aid in the modeling of the traffic network. In their algorithm, they considered the difference in the queue length between any two adjacent nodes at each time slot and then determined a feasible schedule by giving priority to the phase with the maximum difference. In this algorithm, at each intersection at each time slot, the decision regarding which phase of the traffic movement is scheduled is independently determined based on the queue length. The results showed that the algorithm can achieve maximum network throughput without any prior knowledge about arrival rates.

However, in practice, an algorithm considering only queue length information has an inherent weakness. Specifically, queue-based scheme suffers from larger delay when encountering the last packet (or vehicle) problem in wireless networks [47]. In a delay-based scheme, at each intersection, the decision regarding which phase of the traffic movement is scheduled is determined by the head-of-the-line (HOL) delay. They proved that there is a linear relationship between queue lengths and delays in the fluid limit model. That is, for each link-flow-pair (s, k), $q_{s,k}(t) = \lambda_s w_{s,k}(t)$, they showed that the queue length grows when the delay becomes longer. This implies that the delay-based BP scheme is quite similar to the queue-based BP scheme. However, the delay-based scheme outperforms the queue-based scheme for the last packet problem.

The delay-based BP traffic signal control scheme at an isolated intersection was studied in [117] for solving the last vehicle problem. Besides, a hybrid scheme was proposed that considers both queue and delay information and achieves the advantages of both the schemes. It can switch between the two schemes using a tuning parameter. Depending on different traffic situations, the parameter determines whether the scheme adopts a queue-based scheme or a delay-based scheme. We describe the schemes in detail in Section 3.3.1.

2.2 Connected Autonomous Vehicle (CAV)

Recently, CAV has been well studied and there is a plethora of CAV-based research that has been conducted to optimize vehicular maneuvers and fuel usage as well as reduce traffic latency and pollutant emissions. Lee *et al.* [54] proposed a cooperative vehicle intersection control (CVIC) algorithm that enables seamless cooperation among vehicles and infrastructure to manipulate vehicular maneuvers for a traffic environment without any traffic signal. They conducted experiments and showed improvements of air quality and fuel consumption under an assumption that all vehicles are fully connected and automated. With CAV technology and received signal phase and timing (SPaT) information, drivers is able to increase energy efficiency and reduce pollutant emissions generated by unnecessary stops, idles, accelerations, and decelerations [56]. In [28], Feng *et al.* developed an algorithm that exploits CAV data to determine the optimal signal phase allocation to minimize the total delay and the queue length. Cooperative adaptive cruise control (CACC), which enables information sharing (e.g., speed and acceleration) among connected vehicles to reduce inter-vehicle distance, is one of famous CAV use-cases for achieving mobility and safety [67, 69, 106, 109]. Yang *et al.* [119] developed an Eco-CACC algorithm that calculates a fuel-optimum trajectory for each vehicle arriving an isolated signalized intersection. Wang *et al.* [108] proposed a cluster-wise cooperative algorithm that clusters CAVs into groups and perform eco-driving in a collaborative manner. Furthermore, with platooning control and signal flexibility, Guler *et al.* [38] proposed a CAV-based algorithm which considers the location and speed information shared by connected vehicles to decide which sequence of vehicles need to be discharged from an intersection to reduce the number of stops and the total delay.

2.3 Platooning Control Methods

Platooning control is a well-known technique applied to form vehicular groups (platoons) where vehicles in the same platoon drive close to each other to decrease gaps. This technique can be realized in an environment where most of vehicles are connected and their information such as position and speed is shared. Much research on platooning control has been conducted to study impact on traffic in terms of road capacity, travel latency, and fuel consumption. They demonstrated that 1) road capacity can be substantially increased [40, 66, 95] as well as fuel usage can be decreased due to reduction of air drag and speed variation by platooning [5, 68, 97]. Talebpour *et al.* [92] studied the impact of platooning control on an traffic environment with both CAVs and non-CAVs. Lioris *et al.* [60] investigated potential improvement of intersection capacity when vehicles communicated with each other to form a platoon; thus, they passed an intersection as a whole and increased the saturation flow rate by a factor of up to three. Fernandes *et al.* [30] conducted simulations with different platooning profiles to maximize road capacity. They proposed a new approach that separates the control of the leading vehicle the other member vehicles. The former is controlled by the pre-defined TraCI package and the latter is controlled by a novel module they implemented for SUMO. Santini *et al.* [85] proposed a consensus-based algorithm that regulates gaps among vehicles in the same platoon with respect to not only their predecessor and but also the leading vehicle. The proposed platoon high-order equation was solved by a decentralized control variable embedding the gap information and the communication delays. Zhang *et al.* [126] highlighted work related to fuel savings for truck platooning and summarized existing methods, their contributions, and platooning strategies that computes fuel-efficient speed profiles for vehicles in the same platoon. Moreover, PERMIT [65], a platooning simulator based on SUMO [53] and its extension Plexe, was implemented to simulate platooning maneuvers such as merge, leave, split by using finite state machine. VENTOS [6], an integrated simulation platform based on SUMO [53] and OMNET++, was developed to provide platooning management protocol to perform platoon operations using the three basic maneuvers merge, split and lane-change.

On the other hand, the emergence of machine learning (ML) offers an efficient way to tackle optimization problems by using non-linear approximation functions or an unsupervised learning framework. Many ML-based methods have been well-studied and efforts have been devoted to design platooning control that combines well-known ML models to achieve energy efficiency and improve road capacity. The first application of using RL to CACC was proposed by Desjardins *et al.* [23]. They proposed a model-free RL-based CACC algorithm to perform basic cruise controls (e.g., acceleration and brake) for a platoon of vehicles. They conducted experiments on two vehicles in a simple traffic environment where the state is depicted by headway information, the actions are the cruise controls, and the reward is defined by the gap between the two vehicles. The proposed algorithm updates neural parameters by using gradient descent in order to maximize the expected cumulative reward. Buechel *et al.* [15] introduced an idea of the state augmentation by including advance knowledge and proposed an actor-critic RL algorithm to learn longitudinal control with a predicted leading vehicle trajectory of desired velocities adaptive to road grade changes over time. Chu *et al.* [21] designed a model-based deep RL (DRL) CACC algorithm for a heterogeneous platoon with both human-driven and autonomous vehicles. The control logic is to determine desired headway for vehicles and their acceleration are separately modeled by the optimal velocity model (OVM). Lu *et al.* [61] proposed a deep deterministic policy gradient (DDGP) based platooning control to address inefficiency in exploring continuous action space. A shared model was trained for members in the same platoon to solve the convergence problem. All the platoon members was jointly participating in generating training samples to diversify samples and enhance the exploration. Chen *et al.* [19] proposed a DRL-based algorithm to handle platoon maneuvers, aiming at reducing fuel consumption. The DRL model learns to determine the optimal entry point for vehicles which wants to be merged into a platoon. Li *et al.* [57] discussed how to formulate the drift counteraction optimal control (DCOC) problem in a problem to which RL can be applied. They developed a RL-based adaptive cruise control (ACC) algorithm under the DCOC framework to automatically change a speed and maintain a safe headway for each vehicle, aiming at maximizing the expected time before safety and fuel efficiency are violated. Inappropriate driving behaviors such as frequently deceleration caused by heavy traffic or traffic light usually lead to a higher fuel consumption and accidental risks. Zhou *et* *al.* [129] proposed a DDPG-based algorithm to address well-known traffic oscillation issue triggered by human drivers. The trajectories of CAVs were controlled by acceleration determined the proposed model Their results showed significant improvements including higher travel efficiency and safety as well as lower fuel consumption and traffic oscillation.

2.4 Reinforcement Learning

Reinforcement learning (RL) [91] is an unsupervised learning algorithm where an agent learns from experiences $\mathcal{E}$ receiving from an environment. Experiences contain the states, actions, and rewards where $\mathcal{E} = \{s_0, a_0, r_0, \dots, s_t, a_t, r_t\}$. The agent interacts with an environment by observing states, taking actions accordingly, and receiving rewards such that maximizes the total reward without any prior knowledge. First, the current state s_t is an observation receiving from the environment at every time step t. Second, the current action a_t is a response which could be possibly taken according to the state s_t by the agent. The final one is reward r_t, which is the feedback from the environment after taking action a_t. The reward r_t is utilized to iteratively update the current state-action pair so as to achieve the best policy. RL is a trial-and-error framework which is capable of searching an optimal policy by fine-tuning exploration and exploitation.

2.4.1 Markov Decision Process

Markov Decision Process (MDP), a mathematical framework, is a discrete time control process for modeling decision making problem. Formally, MDP describes an environment for RL and contains the following elements:

- A set of states $\mathcal{S}$;
- A set of actions $\mathcal{A}$;
- A transition function $\mathcal{T}(s_{t+1}|s_t, a_t)$ that maps each state-action pair to a distribution of next state;
- A reward function $\mathcal{R}(s_t, a_t, s_{t+1})$ that measures how well the agent performed in state s_t; and
- A discount parameter $\gamma \in [0, 1]$ for future reward.

RL problems can be formalized as MDPs. The important MDP property is that given the current state s_t, the next state s_{t+1} is independent from all previous states $(s_0, s_1, s_2, \dots, s_{t-1})$. For TSC systems, the learning trajectory modeled by MDP is *episodic* where the control system will be terminated at a given training time T. The objective of a MDP agent is to find an optimal policy π which maximizes the expected cumulative reward $\mathbb{E}[R_t|s,\pi]$ where R_t is given by

$$R_t = \sum_{t=0}^{T} \gamma^t r_t \tag{2.1}$$

where the discount parameter γ indicates how critical the future rewards will be. γ close 0 represents myopic evaluation, whereas γ close 1 denotes far-sighted evaluation. There are two types of RL agents. One fully understands the transition probability $\mathcal{T}$ from the current state s_t to the next state s_{t+1}, which is also known as the model-based RL. The other is not aware of the transition function and needs to explore the environment. That is called model-free RL. Model-free RL can be further divided into 1) value-based and 2) policy-based methods. In a value-based RL agent, a value function which maps each state-action pair to a value will be updated at every iteration. In policy-based RL agent, policy will be updated by using policy gradient at each iteration.

2.4.2 Value-based RL Agent

Given the current state s under policy π, a value function $V^\pi(s)$ estimates the value of state s in which the agent is by the expected cumulative reward and is given by

$$V^\pi(s) = \mathbb{E}[R_t|s,\pi] \tag{2.2}$$

The optimal value function $V^*(s)$ is defined as the maximized state-value function over the policy π for all states and hence given by

$$V^*(s) = \max_\pi V^\pi(s), \forall s \in \mathcal{S} \tag{2.3}$$

Furthermore, the state-action value function $Q^\pi(s,a)$ also known as the quality function (Q-function) is determined by the expected reward of taking action a in the current state s and is given by

$$Q^\pi(s,a) = \mathbb{E}[R_t|s,a,\pi] \tag{2.4}$$

Similarly to the optimal state-value function V^*, optimal action value function Q^* can be calculated by maximizing its expected return over all states and hence given by

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a), \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \tag{2.5}$$

In addition to state-value, actions taken by the agent will affect the return as well. The optimal expected return can be captured by Q^*. Then, the optimal state and action value functions can be given by

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s,a), \forall s \in \mathcal{S} \tag{2.6}$$

Q-function $Q^*(s,a)$ provides the optimal policy π^* by selecting the action a that maximizes the Q-value for the state s and is given by

$$\pi^* = \operatorname*{argmax}_{a \in \mathcal{A}} Q^*(s,a), \forall s \in \mathcal{S} \tag{2.7}$$

2.4.2.1 Temporal-difference Learning Method

Temporal-difference (TD) learning method, which is a model-free RL method, sample experiences from incomplete episodes and update Q-values depending on estimated returns (TD target). Compared to Monte Carlo (MC) learning methods, which update Q-values by the *empirical* mean returned at the end, TD learning methods directly learn by bootstrapping from the current estimate of the value function.

Based on definitions of the value function and TD learning methods, there are two types of RL algorithms: off-policy and on-policy RL algorithms. The classic off-policy and on-policy learning algorithms are Q-learning [111] and SARSA [83], respectively. These TD learning methods are conventional tabular RL. In other words, the values of state-action pairs (Q-values) are stored in a Q-table, and are learned via the recursive nature of Bellman equations utilizing the Markov property given by

$$Q^{\pi}(s_t,a_t) = \mathbb{E}_{\pi}[r_t + \gamma Q^{\pi}(s_{t+1}, \pi(s_{t+1}))] \tag{2.8}$$

For each iteration, the Q^π estimate is updated with a learning rate α to improve the estimation as follows

$$Q^\pi(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha(y_t - Q^\pi(s_t, a_t)) \tag{2.9}$$

where y_t is the TD target for $Q^\pi(s_t, a_t)$ and $y_t - Q^\pi(s_t, a_t)$ is the TD error.

1. **Q-learning** is an off-policy learning model in which the selection of actions based on maximizing Q-values over all actions. It updates Q-value estimates by acting greedily.

$$y_t^{Q-learning} = r_t + \gamma \max_{a_t} Q^\pi(s_{t+1}, a_{t+1}) \tag{2.10}$$

2. **SARSA** is an on-policy learning model in which the selection of actions depends on the policy π derived from the Q-function. It updates Q-values and takes actions by following the same policy.

$$y_t^{SARSA} = r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) \tag{2.11}$$

ϵ-greedy policy is the most common and intuitive approach for exploration and exploitation in both Q-learning and SARSA. In the ϵ-greedy policy, an action is randomly selected with probability ϵ, and the best action with respect to the current policy defined by $Q(s, a)$ is chosen with probability $1 - \epsilon$.

2.5 Deep Reinforcement Learning

Conventional RL algorithms perform well for simple scenarios such as an isolated intersection with static traffic demands and regular traffic patterns. Complicated traffic problems usually lead to huge state space in which the state size grows exponentially. The tabular RL approaches are no longer suitable for learning informative features from complex environments because they will soon run out of memory while constructing Q-table for all state-action pairs and will fail to update all of them. However, the huge state space problem can be solved by deep learning. Deep RL (DRL) applies neural networks as function approximators to approximate Q-values instead of building a Q-table. Various neural networks like artificial neural network (ANN), convolutional neural network (CNN), and recurrent neural network (RNN) have been applied to train RL algorithms for high-dimensional state spaces.

2.5.1 Deep Q-Network

Value-based RL algorithms update Q-value by the tabular approach. This approach is not feasible to visit all state-action pairs especially for complex systems because state space grows exponentially and the action space can also be large. Mnih *et al.* [72] proposed Deep Q-Network (DQN) applying deep neural networks to approximate Q-values where the network parameter set is θ, and the Q-function approximation can be rewritten as $Q(s, a; \theta)$. The goal of the neural network is to update the network parameter set θ that optimizes the loss function $L(\theta_t)$ given by

$$L(\theta_t) = \mathbb{E}_\pi[(y_t - Q^\pi(s_t, a_t; \theta_t))^2] \tag{2.12}$$

where s_t is the current state, a_t is the current action, s_{t+1} is the next state which is dependent on a_t taken by the agent in s_t, a_{t+1} is the next action selected by ϵ-greedy, and y_t is the TD target given by Eq. 2.14 and 2.15.

First, the partial derivative of $L(\theta_t)$ is calculated by differentiating Eq. 2.12. The gradient of the loss function can be obtained as follows:

$$\triangledown_{\theta_t} L(\theta_t) = \mathbb{E}_\pi[(y_t - Q^\pi(s_t, a_t; \theta_t)) \triangledown_{\theta_t} Q^\pi(s_t, a_t; \theta_t)] \tag{2.13}$$

where $\triangledown_{\theta_t} Q^\pi(s_t, a_t; \theta_t)$ denotes the gradient of the current state-action value. We optimize the loss function $L(\theta_t)$ by Eq. 2.13 and the network parameter set θ is updated by using stochastic gradient descent (SGD).

The output of neural network is the best action selected from a discrete set of approximate action values based on Eq. 2.7. DQN takes images as input states and processes them to estimate Q-values by CNNs. The proposed DQN outperforms the expert human performance on a variety of Atari games.

Mnih *et al.* [72] contributed two state-of-the-art techniques for stabilizing the learning process with deep neural networks: 1) target network and 2) experience replay. The target network is a critical part of DQN and plays an important role to stabilize the learning process. Two separate neural networks are implemented in DQN: the main network that approximates the Q-function and the target network that provides the TD target for correcting the TD error (updating the main network). The parameters θ of the main network are updated after every action during the training process. In contrast, the parameters θ^- of the target network are only updated after specific time steps τ. The target network cannot be updated after every iteration because it corrects the TD error for the main network in order to control the value estimations. If two networks were updated simultaneously, the change in the main network would become very exaggerated because of the feedback loop caused by the target network. This will lead to an extremely unstable network. The TD target for DQN can be written as follows:

$$y_t^{DQN} = r_t + \gamma \max_{a_{t+1}} Q^\pi(s_{t+1}, a_{t+1}; \theta_t^-) \tag{2.14}$$

where $Q^\pi(s_{t+1}, a_{t+1}; \theta_t^-)$ is the target network.

2.5.2 Double Deep Q-Network

DQN is the improved version of the conventional Q-learning algorithm by applying neural networks to Q-learning. But, DQN and Q-learning, which both use only one estimator, could lead to overoptimistic value estimates in actions because of having single Q function estimations. Hasselt *et al.* [102] first came out with an idea that applies two estimators separately to DQN. One estimator with the main network is for action selection, and the other estimator with the target network is for action evaluation. Double DQN (DDQN) separates the action selection, and action evaluation for better Q-value estimation. Instead of selecting the Q-value that maximizes future reward using the target network (see Eq. 2.14), DDQN selects the action by the main network and evaluates it by the target network. The TD target for DDQN can be written as follows:

$$y_t^{DDQN} = r_t + \gamma Q^\pi(s_{t+1}, \max_{a_{t+1}} Q^\pi(s_{t+1}, a_{t+1}; \theta_t); \theta_t^-) \tag{2.15}$$

2.5.3 Dueling Network Architecture

Many papers have integrated RL and existing neural networks (NN) together because this combination has revealed success in learning Atari games in recent years. The idea of dueling was proposed

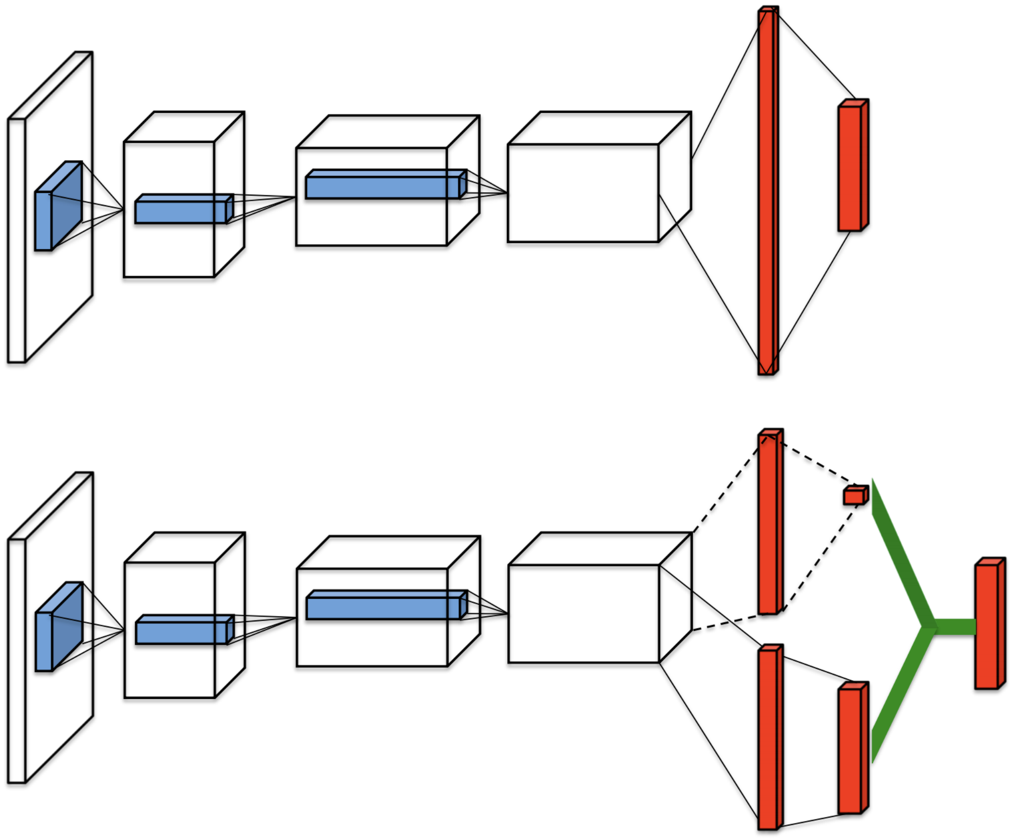


Figure 2.1: The conventional one sequence network is shown at the top. The dueling network with two sequences is shown at the bottom. State-value and the advantages for each action are separately estimated by the dueling network; the output of the dueling network (green) implements Eq. 2.16 to combine state-value and advantages.

by [110]. They first came up with such an idea for the model-free reinforcement learning. Instead of using one single sequence for value estimations, they implemented the dueling architecture where a single Q Network has two separated sequences, one for value and the other for action advantage as shown in Fig. 2.1 (borrowed from [110]). They proposed this architecture because they found that in some states, it matters which action needs to be taken, but most states, the selection of action has no influence on what happens. Hence, they separated the estimations of state-value and action advantage in order to learn which state-values are higher without going through and learn every single state-action pair. Even if they are separated, the state-value and action advantage are still sharing the same convolutional feature.

The goal of the dueling architecture is to estimate Q by separately estimating V and A. It can be formulated as follows:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + [A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)] \tag{2.16}$$

where Q represents the value of selecting a particular action a in a given state s. By following a policy π, the Q-value of state-action pair can be computed recursively. The goal is to maximize the summation of expected reward. V denotes how well it is in the given state s. A indicates a relative measurement of importance of each action a. However, there is a problem where the values are not unique. For example, if A is increased by 5 and V is decreased by 5, it cancels out the result in Q. The strategy here is to subtract the mean in order to solve this non-unique problem and increase stability.

Figures 2.2 and 2.3 are borrowed from [110]. They clearly show the performance of the dueling architecture. In this corridor environment, there are vertical and horizontal directions, each of which has 8 grids and 50 grids, respectively. The goal for the agent is to learn how to reach the destination, the red zone. The actions which could be taken by this agent are going up, down, left, right, and nop. Nop stands for no operation which means it is an useless action and it does not matter whether the agent take it or not because it will not affect the environment and will not be helpful for reaching the goal.

The dueling architecture helps the agent if the action space is large. As mentioned above, there are five actions could be taken. Fig. 2.3 shows different action space consisted of up, down, left, right, and an arbitrary number of nop. For 5 actions, there are up, down, left, right, and one nop. For 10 actions, there are up, down, left, right, and six nops. As can be observed in 2.3, it does not converge faster for a less number of actions. Instead, it achieves the lower squared error (green line) when comparing to a single sequence architecture (red line).

The larger the action space would be, the more the useless nops could be. The dueling architecture trains much better when the number of actions is high because it is able to exclude useless actions by separating the estimations of V and A. Therefore, it is able to outperform the single sequence architecture while increasing the number of actions.

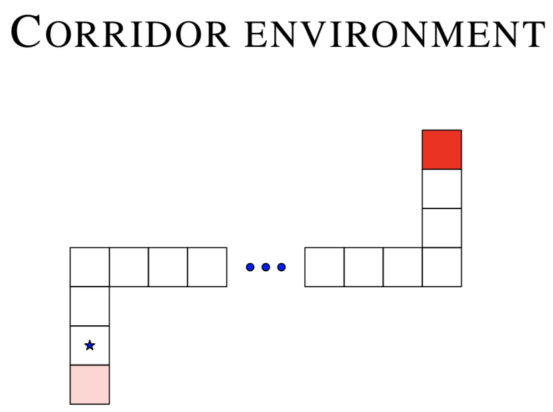


Figure 2.2: A sample environment with 10 grids in vertical direction and 50 grids in horizontal direction

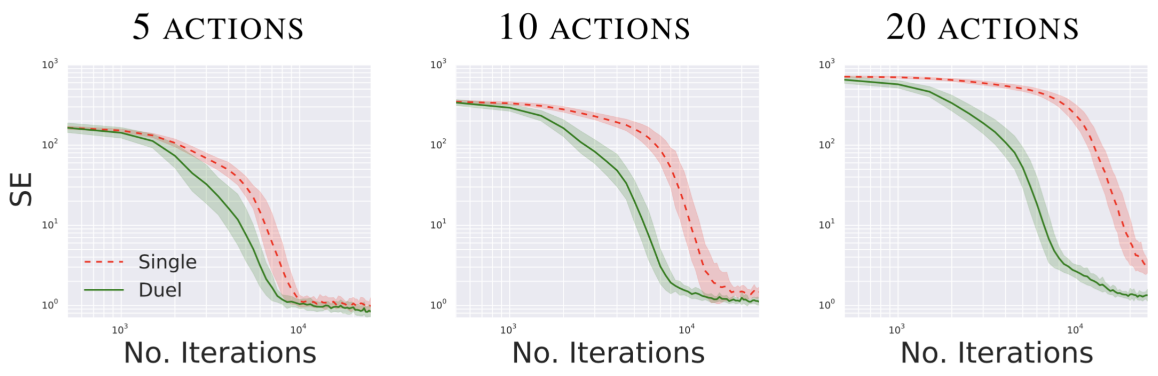


Figure 2.3: Different number of actions and their squared errors

2.5.4 Experience Memory Replay

For RL, in a non-terminating environment, quickly getting rid of experiences after only one time update is probably not a wise decision. First, due to such an environment, all updates are strongly correlated where the temporal correlation will destroy i.i.d assumption of many popular stochastic gradient-based algorithms. Second, those experiences, which have been discarded, could be used later on. Experience replay [86] is able to break the strong temporal correlations by first combining old and new experiences and then, training on them together. Also, it can speed up the learning process by avoiding fully sweep over the entire replay memory. In [86], they used the prioritized experience replay to select replay samples based on priorities. Given p_i is the priority of experience i, a rank-based method is used to calculate the priority of an experience sample. The experiences will be ranked by the TD errors and then the priority p_i of experience i is its rank. The key idea is to increase the replay probability P_i of the samples with a high TD error.

Chapter 3

Security Vulnerabilities in Traffic Signal Control

3.1 Introduction

With the ever-increasing traffic demands, the transportation system faces challenges such as traffic congestion, traffic accidents, and high energy consumption and the related emissions. In order to deal with these critical issues in the transportation systems, a significant plethora of research in the recent decades has investigated algorithms for traffic signal control (TSC) [70, 112]. According to data reported in [127], an urban road network equipped with an advanced TSC system is capable of not only reducing average travel time of vehicles by 11.4% but also decreasing traffic collisions by 6.7%, traffic delay by 24.9%, parking by 27%, while at the same time, decreasing energy consumption. The data revealed the impacts of TSC on our future urban planning.

With connected and autonomous vehicles (CAVs), the transportation system can be viewed as a cyber-physical system (CPS) [10] that is able to combine several multi-disciplinary technologies from sensing, computation, communication, and control to achieve optimal operation. In such a transportation-based CPS, CAVs, roadside units and sensors are seamlessly integrated with TSC using wireless communication. In the foreseeable future, pedestrians, bicyclists, vehicles, and traffic signals, can communicate using Vehicle-to-Vehicle (V2V), and Vehicle-to-Infrastructure (V2I) communication [99]. This enables new traffic control algorithms that can enhance safety and improve the throughput and delay characteristics of the traffic in the road network.

However, cyber-attacks are potential threats to such a fully connected environment [36]. A recent study [80] found that even if attackers are currently not able to compromise the TSC algorithms, they can still send falsified data by hacking sensors to degrade the performance of the TSC algorithms and create traffic congestion. This situation can be compared to the current problem with fake news in social media. According to surveys about fake news [88, 103], fabricated content in news has increased rapidly in recent years. This fake news content is able to influence agenda-setting and distract media attention from real news topics. Furthermore, in [13, 87], it has been demonstrated that popular social media like Twitter creates a breeding ground for widely spreading misinformation which is extremely difficult to be distinguished from real news topics. Misinformation attacks are becoming an intractable problem and need to be considered into the system level design because it is able to not only occupy the time for real news topics but also intentionally misleads the public to create chaos and cause damages in terms of health and finance [31, 44].

Similar to fake news in social networks [13, 31, 44, 87, 88, 103], TSCs are also susceptible to misinformation attacks [80]. In particular, misinformation attacks can be launched by falsifying data on the trajectory and/or the arrival time of a vehicle. Using such attacks, adversaries can disrupt the signal phases to either benefit from getting served earlier than other vehicles or block a platoon of vehicles from passing an intersection for a while. Such disruption at one intersection can lead to delays that can spread to adjacent intersections, and eventually create traffic congestion and gridlock in the entire network. Vulnerabilities of navigation systems such as Google Navigation and Waze, which both use position data of smartphones (a.k.a floating car data (FCD)), have been analyzed specifically with respect to the impact of sending counterfeit GPS data [46]. The study showed that even though the Google protocol uses Transport Layer Security (TLS), an adversary can replay collected packets with modified cookies, time stamps and platform keys to masquerade as multiple vehicles and create traffic congestion. The evaluation focused on user privacy and data authenticity. For example, in Google protocol, the TLS guarantees data integrity; however, the TLS tunnel becomes completely useless once adversaries control the beginning of it. Adversaries are able to randomly select user ID and cookie in the protocol header and then, fabricate counterfeit location data to the navigation system without being detected. That is to say, If adversaries drive around and collect the data packets sent to the navigation system, they can reuse those collected packets and intensify their attacks by transmitting many delayed packets with modified cookies, time stamps and platform keys to masquerade multiple vehicles. If adversaries are smart enough, they can further utilize different IP addresses as well as additional noise which make the navigation system even more difficult to distinguish between real and counterfeit data. Experiments have shown that it is possible for adversaries to control the navigation system and create traffic congestion.

Based on our previous work [117], three BP-based traffic control algorithms were analyzed with respect to throughput and fairness. They are, namely, Delay-based BackPressure Control (DBPC), Queue-based BackPressure Control (QBPC), and Hybrid-based BackPressure Control (HBPC). The result showed that DBPC can outperform QBPC and provide a better fairness while facing the last vehicle problem which is analogous to the the last packet problem [47]. Later, we conducted analyses of the time spoofing attack on the above three BP-based traffic control algorithms as well as Sum-of-delay-based BackPressure Control (SBPC) in [122]. The results showed that BP-based traffic control algorithms are highly susceptible to falsified data attacks particularly when the TSC is not able to easily identify and/or filter out malicious and falsified data. Our experiments showed that the DBPC is more vulnerable to time spoofing attacks than other BP-based traffic control algorithms. In contrast, the HBPC that combines the queue-based and delay-based can be used to switch to queue-based traffic control when the time spoofing attack is detected. The study also considered the coordinated attack strategy in which the knowledge of the heterogeneity in the arrivals can be exploited to amplify the effect of the time spoofing attack in the case of DBPC in a network of intersections.

In this chapter, we study the impacts of two misinformation attacks, namely, time spoofing attack and ghost vehicle attack on the four different BP-based traffic control algorithms [117, 122]. The study considers an isolated intersection in a fully connected system where vehicles and the TSC are able to send and receive data without any degradation even when under attack. Our security study builds on and is contemporary to recent studies [20, 29] and [35] which conducted security analyses on CAV-based transportation systems and found possible attack strategies which could significantly impact the effectiveness of a TSC. The study showed that there would be a 68% increase in the total delay even with only one attack. Other types of attacks that could also significantly reduce the effectiveness of TSCs are outlined in [41, 81, 84]. We consider an advanced persistent threat model (Section 3.4) in which the adversary has time and resources to launch complex attack on the TSC system. Specifically, the adversary using reconnaissance can learn both the traffic pattern and the behavior of the TSC algorithm and has a budget to insert multiple attack vehicles in the traffic stream and launch a misinformation attack. The adversary utilizes an optimization framework in an offline setting to learn the optimal attack strategy that results in the maximum number of disrupted signal phases in a BP-based TSC system. This provides the adversary a method to launch attack in an online setting that can achieve higher number of disrupted signal phases than achieved by a random attack. Furthermore, by analyzing the impact of these attacks, we design protection algorithms and evaluate their performance against these attacks. We show that our proposed algorithms are able to withstand a certain level of attacks and mitigate the damage from misinformation attacks.

3.1.1 Organization

The rest of this chapter is organized as follows. We list our contributions in Section 3.2. In Section 3.3, we introduce the model of the TSC for an isolated intersection and describe the four different BP-based traffic control algorithms considered in this study. In Section 3.4, we elaborate on the misinformation attack and describe the proposed threat model. For an offline setting, we first formulate an optimization problem relating the number of disrupted signal phases and the attack strategy, and then outline a solution based on branch-and-bound algorithm to determine the optimal attack strategy. In Section 3.5, we describe two protection algorithms for misinformation attack on BP-based TSC algorithms. In Section 3.6, we discuss the results of the offline and online simulations of misinformation attack on BP-based traffic control algorithms. Finally, we conclude and discuss future work in Section 3.7.

3.2 Contribution

The main contributions of this chapter are summarized below:

- We analyze the impact of time spoofing attack and ghost vehicle attack on four BP-based traffic control algorithms at an isolated intersection.
- We propose a threat model in which an advanced persistent adversary can formulate the attack strategy as a 0/1 Knapsack problem and determine the optimal strategy using a

branch-and-bound algorithm.

- We show that the ghost vehicle attack degrades the performance of QBPC and SBPC. On the other hand, time spoofing attack degrades the performance of DBPC.
- We propose an auction-based and a hybrid-based protection algorithms against misinformation attacks and show that they can mitigate the impact of the attacks.

3.3 System Model

Fig. 3.1 shows an isolated traffic intersection. We consider a fully connected system in which every vehicle is required to send a message to indicate its arrival when approaching the intersection. From these messages, the TSC knows the number of vehicles at each lane and the arrival time of each individual vehicle in each lane. In the subsequent discussion, the number of vehicles in each lane will be referred to as its queue length and the waiting time of the head vehicle will be considered as the Head-Of-Line (HOL) waiting time. The TSC operates in a time slotted manner. At the beginning of each time slot, it applies a traffic control algorithm to determine a feasible signal phase, i.e., which lane(s) are allowed to release vehicles through the intersection during the time slot. The goal of this study is to analyze the impact of the time spoofing attack and ghost vehicle attack. In time spoofing attacks, vehicles (shown as red vehicles in Fig. 3.1) intentionally falsify their arrival time to make their waiting time appear to be longer. In ghost vehicle attacks, vehicles (shown as green vehicles in Fig. 3.1) disconnect themselves by not sending the required message when arriving at the intersection. Consequently, the TSC is unaware of their presence. We also study our proposed protection algorithms against these attacks. The summary of notations for this paper is listed in Table 3.1.

We model the TSC as a queuing network with vertices and edges. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a directed graph where $\mathcal{V}$ indicates a set of vertices corresponding to different lanes and $\mathcal{E}$ denotes a set of edges corresponding to traffic movements between any connected lanes. Fig. 3.2 shows an example of 4 signal phases at an isolated intersection. Each signal phase consists of two non-conflicting traffic movements. In our simulations, we assume that each vehicle has a fixed routing path and this routing information is known *a priori*.

Let $\mathcal{A} = \{a_{i,j,l} \mid (i,j) \in \mathcal{E}, l \in \mathbb{N}\}$ be the set of all arrivals where $a_{i,j,l}$ denotes the l-th vehicle

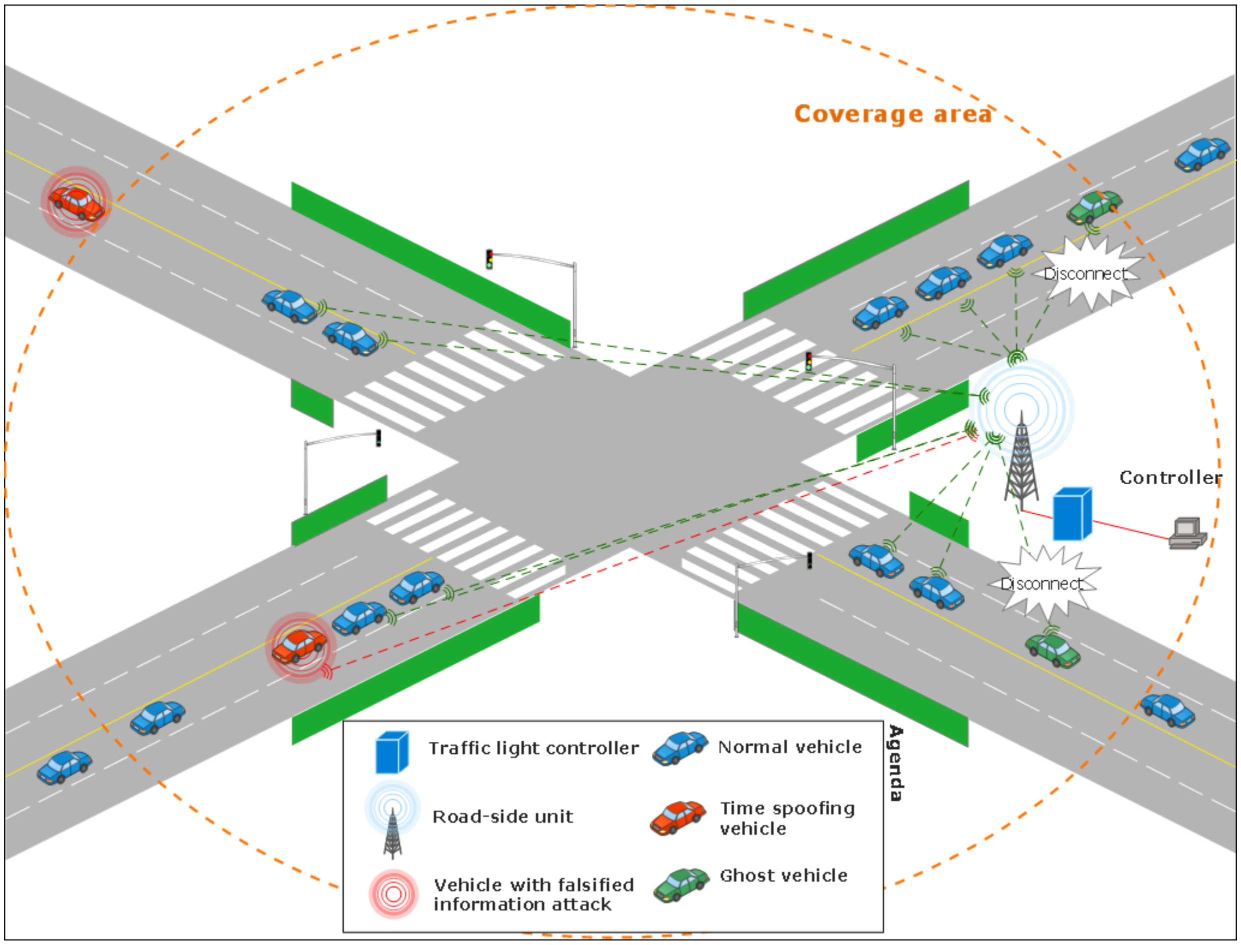


Figure 3.1: The illustration shows the system model at an isolated intersection. The attack vehicles are in red and green. Waiting and approaching vehicles are in blue. The red vehicles carry out time spoofing attacks and the green vehicles carry out ghost vehicle attacks.

arriving at the intersection, and it is in lane i destined for j. We let $N_i(t)$ be the number of vehicles that arrive to the intersection in lane i during time slot t, and τ be the total number of time slots. Then the average arrival rate in lane i, denoted by λ_i is given by

$$\lambda_i = \lim_{\tau \to \infty} \frac{1}{\tau} \sum_{t=0}^{\tau-1} \mathbb{E}(N_i(t)) \tag{3.1}$$

Let $\vec{\lambda} = \{\lambda_1, \lambda_2, \dots, \lambda_{|\mathcal{V}|}\}$ be an arrival rate vector of the intersection. Let $Q_{i,j}(t)$ be the number of vehicles in lane i at the beginning of time slot t that will transfer to lane j. We denote $Q_{i,j}$ to represent the queue of vehicles in lane i that will transfer to lane j and $\vec{Q}(t) \triangleq [Q_{i,j}(t), (i,j) \in \mathcal{E}]$ is the queue length vector at time slot t. Let $F_{i,j}(t)$ be the number of vehicles arriving in lane i for lane j until time slot $t \geq 0$, and $\hat{F}_{i,j}$ be the number of vehicles served at $Q_{i,j}$ until time slot $t \geq 0$.

Based on the above, the queue length $Q_{i,j}(t)$ is given by

$$Q_{i,j}(t) = F_{i,j}(t) - \hat{F}_{i,j}(t) \tag{3.2}$$

Let $T_{i,j,k}(t)$ represent the waiting time of the k-th vehicle in lane i for lane j at time slot t where this time is measured from the time the vehicle arrives at the intersection. Let $W_{i,j}(t)$ be the waiting time of the HOL vehicle in lane i for lane j at time slot t. Therefore, $W_{i,j}(t) \triangleq T_{i,j,1}(t)$ and if $Q_{i,j}(t) = 0$, it implies $W_{i,j}(t) = 0$. Likewise, $\vec{W}(t) \triangleq [W_{i,j}, (i,j) \in \mathcal{E}]$ indicates the HOL waiting time vector at time slot t.

We define $U_{i,j}$ to be the time when the first vehicle (HOL) arrived in lane i for lane j at the intersection. Then

$$U_{i,j}(t) \triangleq t - W_{i,j}(t). \tag{3.3}$$

A feasible signal phase is a set of movements that can be scheduled concurrently. This is denoted as $\vec{p} \in \{0,1\}^{|\mathcal{E}|}$. Let $\mathcal{S}_{\mathcal{P}}$ be a set of all feasible signal phases. Note that if $p_{i,j}(t) = 1$ then vehicles are allowed to pass from lane i to lane j at time slot t. On the other hand, if $p_{i,j}(t) = 0$, vehicles cannot move from lane i to lane j. We use $\mu_{i,j}(\vec{p})$ to represent the saturation flow rate at which vehicles move from lane i to lane j under a feasible signal phase $\vec{p}$.

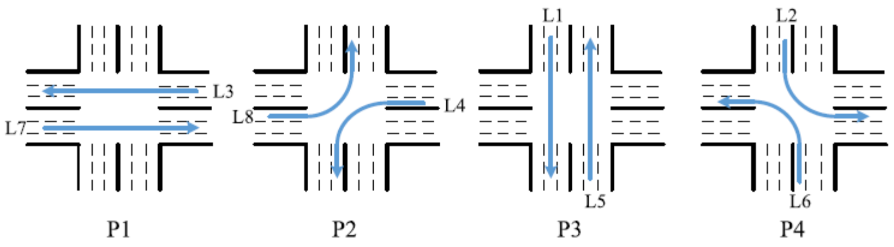


Figure 3.2: An isolated intersection activates the 4 signal phases (P1, P2, P3, P4), each of which consists of two non-conflicting traffic movements in our simulation.

3.3.1 Backpressure Based Traffic Control Algorithms

In this section, we consider four different BP-based traffic control algorithms, namely, Queue-based BackPressure Control (QBPC), Delay-based BackPressure Control (DBPC), Hybrid-based BackPressure Control (HBPC), and Sum-of-delay-based BackPressure Control (SBPC) [117, 122]. In these algorithms at each time slot, the signal phase is assigned to lane(s) with the maximum

Table 3.1: Summary of Notations

Symbol	Description
$\mathcal{V}$	set of vertices (different lanes)
$\mathcal{E}$	set of edges (traffic movements between any two lanes)
$\vec{p}$	a signal phase (movements that can be scheduled concurrently)
$\mathcal{S}_{\mathcal{P}}$	set of feasible signal phases
$\gamma_{i,j}$	positive constant that emphasizes particular movements for the vehicles in lane i for lane j
$\mu_{i,j}(\vec{p})$	saturation flow rate (vehicles per second per lane) from lane i to lane j under a feasible signal phase $\vec{p}$
$N_i(t)$	number of vehicles arriving for lane i at time slot t
λ_i	average arrival rate (vehicles per second per lane) for lane i
$Q_{i,j}(t)$	queue length of lane i going to lane j at time slot t
$F_{i,j}(t)$	number of vehicles arriving in lane i for lane j at time slot t
$\hat{F}_{i,j}(t)$	number of vehicles served in lane i for lane j until time slot t
$T_{i,j,k}(t)$	the waiting time of the k-th vehicle in lane i for lane j at time slot t
$W_{i,j}(t)$	the waiting time of the HOL vehicle in lane i for lane j at time slot t
$\eta_{i,j}$	parameter that gives different relative importance to queue length and HOL waiting time in lane i for lane j, $\eta_{i,j} \in [0,1]$
$\overline{W}_{i,j}(t)$	the sum of the waiting time of all vehicles in lane i for lane j at time slot t.
$\mathcal{M}_{i,j}(t)$	the misinformation sent by the l-th arrival as an attack vehicle in lane i for lane j at time slot t
$U_{i,j}(t)$	time when the HOL vehicle arrived in lane i for lane j at the intersection
$\mathcal{A}$	set of all arrivals
$\mathcal{L}$	attack strategy formulated by the advanced adversary
$\mathcal{L}_{\mathcal{P}}$	set of all possible attack strategies
$\mathcal{B}$	total cost of assigning arrivals to be attack vehicles
ρ	the probability that an arrival at the intersection is an attack vehicle
c_l	the cost of assigning the l-th arrival to be an attack vehicle
$y_l(t)$	variable indicating whether the decision at time slot t is disrupted by misinformation provided by the l-th arrival as an attack vehicle
h_l	total number of decisions disrupted by misinformation provided by the l-th arrival as an attack vehicle
x_l	variable indicating the l-th arrival is an attack vehicle, $x_l \in \{0,1\}$

pressure. As discussed below, the algorithms differ in the manner in which the maximum pressure is determined.

Queue-based Backpressure Control (QBPC) Algorithm In this algorithm, at each time slot t, QBPC selects the signal phase that maximizes the total pressure released where the pressure is measured in terms of the queue length. Specifically, the signal phase is determined by

$$\vec{p}^{\,*}(t) \in \underset{\vec{p} \in \mathcal{S}_{\mathcal{P}}}{\operatorname{argmax}} \sum_{p_{i,j}=1} \gamma_{i,j} \cdot Q_{i,j}(t) \cdot \mu_{i,j}(\vec{p}) \tag{3.4}$$

In the above equation, the right hand side calculates the queue length $Q_{i,j}(t)$ weighted by two parameters $\mu_{i,j}(\vec{p})$ and $\gamma_{i,j}$. The parameter $\mu_{i,j}(\vec{p})$ denotes the number of vehicles that can be transferred through the connected edge when signal phase $\vec{p}$ is activated. The parameter $\gamma_{i,j}$ in which the subscripts i and j refer vehicles in lane i destined to lane j, is a positive constant that can be used to give priorities to particular movements when there are multiple types of traffic. In Section 3.6, $\gamma_{i,j}$ is set to 1 for all movements. The impact of multiple types of traffic with different priorities is left for future work.

Delay-based Backpressure Control (DBPC) Algorithm In this algorithm, DBPC is similar to QBPC except that the signal phase is determined by the maximum waiting time of the HOL vehicle, $W_{i,j}(t)$. Consequently, the signal phase is determined by

$$\vec{p}^{\,*}(t) \in \underset{\vec{p} \in \mathcal{S}_{\mathcal{P}}}{\operatorname{argmax}} \sum_{p_{i,j}=1} \gamma_{i,j} \cdot W_{i,j}(t) \cdot \mu_{i,j}(\vec{p}) \tag{3.5}$$

The definitions of the two parameters $\mu_{i,j}(\vec{p})$ and $\gamma_{i,j}$ are the same as in QBPC.

Hybrid Backpressure Control (HBPC) Algorithm HBPC combines QBPC and DBPC using two parameters, $\eta_{i,j}^{(W)}$, $\eta_{i,j}^{(Q)} \in [0,1]$. These parameter values can be adjusted dynamically to give different relative importance to queue length and HOL waiting time in determining the signal phase. For example, when a traffic arrival rate from lane i to lane j is low, $\eta_{i,j}^{(W)}$ can be set to a value larger than $\eta_{i,j}^{(Q)}$ to guarantee the fairness of the delay performance. Conversely, by setting $\eta_{i,j}^{(Q)}$ to a value larger than $\eta_{i,j}^{(W)}$, we can bound the queue length if traffic arrival rate from lane i to lane j is high. The signal phase is determined by

$$\vec{p}^{\,*}(t) \in \underset{\vec{p} \in \mathcal{S}_{\mathcal{P}}}{\operatorname{argmax}} \sum_{p_{i,j}=1} \gamma_{i,j} \cdot [\eta_{i,j}^{(W)} W_{i,j}(t) + \eta_{i,j}^{(Q)} Q_{i,j}(t)] \cdot \mu_{i,j}(\vec{p}) \tag{3.6}$$

where the definitions of $\mu_{i,j}(\vec{p})$ and $\gamma_{i,j}$ are the same as in QBPC and DBPC.

Sum-of-delay-based Backpressure Control (SBPC) Algorithm Let $\overline{W}_{i,j}(t)$ be the sum of the waiting time of all vehicles in lane i for lane j at time t where $T_{i,j,k}(t)$ is the waiting time of the k-th vehicle in lane i for lane j at time slot t. Then $\overline{W}_{i,j}(t)$ is given by

$$\overline{W}_{i,j}(t) = \sum_{k=1}^{n} T_{i,j,k}(t) \tag{3.7}$$

Compared to DBPC, SBPC considers not only HOL waiting time but also waiting times of all vehicles in lane i for lane j. The signal phase at each time slot t is determined by the maximum sum of the waiting time of all vehicles, $\overline{W}_{i,j}(t)$, and is given by

$$\vec{p}^{\,*}(t) \in \underset{\vec{p}\in\mathcal{S}_{\mathcal{P}}}{\operatorname{argmax}} \sum_{p_{i,j}=1} \gamma_{i,j} \cdot \overline{W}_{i,j}(t) \cdot \mu_{i,j}(\vec{p}) \tag{3.8}$$

The definitions of $\mu_{i,j}(\vec{p})$ and $\gamma_{i,j}$ are the same as in the other three algorithms.

3.4 Threat Model

In our prior work [122] we showed that the sum of the delays over all the vehicles increases when there are more number of phases disrupted by adversaries. Instead of using delay as in [29] or considering flow and service rate as in [35], we consider an advanced persistent threat model in which the adversary has resources and determines the time and the nature of the attack that maximizes the disruption of the traffic flow at the intersection. Towards this end, the adversary first performs reconnaissance to gain understanding of how the TSC operates under different traffic arrival rates. The adversary has a fixed budget with which it can choose a fixed number of vehicles to be attack vehicles. An attack strategy denoted by $\mathcal{L}$, is selecting a subset of the arriving vehicles (denoted by $\mathcal{A}$) to be attack vehicles as shown in Fig. 3.3. Note that the arriving vehicles being selected to be attack vehicles can be in any lane rather than a fixed one. Mathematically, $\mathcal{L} = \{x_l \mid x_l \in \{0,1\}, l \in \mathbb{N}\}$ where x_l is a binary variable that indicates whether the l-th arrival is an attack vehicle. The goal of the adversary is to determine the optimal attack strategy $\mathcal{L}^*$ that maximizes the number of disrupted signal phases as defined below. Our hypothesis is practical because the study in [18] has shown that TSC can be manipulated by sending falsified data to sensors. Note that it will not be realistic if attack vehicles accounts for the majority of all vehicles. Hence, we limit the number of attack vehicles to a reasonable number.

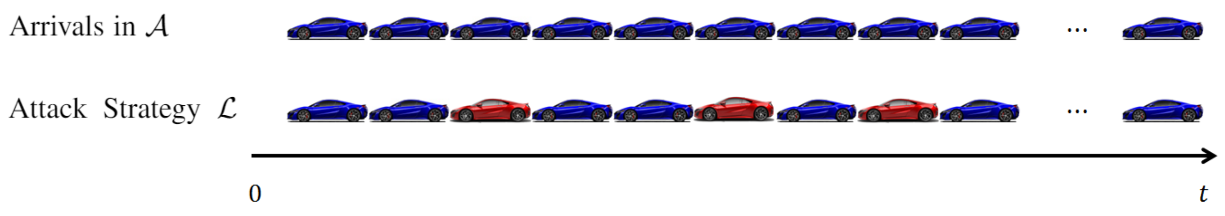


Figure 3.3: An attack strategy $\mathcal{L}$ and arrivals in $\mathcal{A}$ along with arrival time slot t. Each strategy $\mathcal{L}$ will be converted into a binary set where attack vehicles are marked as 1s, e.g., $\mathcal{L} = \{0,0,1,0,0,1,0,1,0,\ldots,0\}$

3.4.1 Misinformation Attack

The study in [18] has shown that TSC can be manipulated by sending fake data to sensors. In this study, the adversary attempts to influence the signal phases by having the attack vehicles provide misinformation to the TSC. We denote the misinformation by $\mathcal{M}$ which can be either not connecting to the TSC and hence becoming a ghost vehicle or falsifying the arrival time. The choice of the misinformation depends on which BP-based traffic control algorithm is adopted by the TSC. The goal is to alter the scheduling decision determined by the TSC to be different from the scheduling decisions without the misinformation. For instance, an attack vehicle could spoof its arrival time to be 50 sec earlier than its actual arrival time resulting in an increase in its waiting time. Under DBPC based scheduling policy given in Eq. 3.5, this may result in a wrong selection of signal phase by the TSC.

Let $\mathcal{M}_{i,j}(t)$ denote the misinformation for the traffic movement from lane i to lane j collected by the TSC system during time slot t, $(i,j) \in \mathcal{E}$. The scheduling decision $\vec{p}\,'$ at time slot t is then given by

$$\vec{p}\,'(t) \in \underset{\vec{p} \in \mathcal{S}_{\mathcal{P}}}{\operatorname{argmax}} \sum_{p_{i,j}=1} \gamma_{i,j} \cdot \mathcal{M}_{i,j}(t) \cdot \mu_{i,j}(\vec{p}) \tag{3.9}$$

where $\gamma_{i,j}$ is a positive constant as explained in Section 3.3.1.

3.4.1.1 Ghost Vehicle Attack

In the ghost vehicle attack, the attack vehicle does not broadcast any message (turns off the network connection) and hence cannot be detected by the TSC. As a result, the TSC system will have misinformation on the number of vehicles in the queue. This will specifically impact QBPC and SBPC algorithms which use aggregate information from all the vehicles in the queue in determining the signal phase. The goal of the adversary is, given a number of ghost vehicles, to determine the optimal attack strategy over time and lanes that maximizes the number of disrupted signal phases denoted by Eq. 3.12.

3.4.1.2 Time Spoofing Attack

We adopt the same definition of "spoofing" as in [20, 122]. Specifically, the attack vehicle sends a falsified arrival time to increase its waiting time. The DBPC algorithm will be misled by this falsified waiting time if it is larger than HOL waiting time in the other lanes. Hence, instead of assigning green time to the lane with longest waiting time, DBPC will activate the lane with the attack vehicle. The goal of the adversary is to determine the optimal attack strategy with a given number of attack vehicles each with a spoofed time that maximizes the number of disrupted signal phases as defined below.

3.4.2 Disrupted Signal Phases

The goal of the adversary is to determine the optimal attack strategy $\mathcal{L}^*$ that maximizes the number of disrupted signal phases. Whether the assigned signal phase at time slot t is disrupted by the l-th arrival can be mathematically represented by an indicator variable $y_l(t)$ which is given by

$$y_l(t) = \begin{cases} 1, & \text{if } \vec{p}\,'(t) \neq \vec{p}\,^*(t) \\ 0, & \text{otherwise} \end{cases} \tag{3.10}$$

where $\vec{p}\,'(t)$ indicates the wrong decision made by the TSC system due to the misinformation it has received and $\vec{p}\,^*(t)$ is the correct decision determined by the TSC system without the misinformation. Note that an attack vehicle keeps sending misinformation until it leaves the intersection. Hence, misinformation sent by one attack vehicle can influence not only one signal phase but also multiple signal phases. Let τ be the total number of time slots. The total number of signal phases disrupted by the l-th arrival as an attack vehicle is given by

$$h_l = \sum_{t=0}^{\tau} y_l(t) \tag{3.11}$$

3.4.3 The Objective Function

The threat model is formulated as the 0/1 Knapsack problem where the profit to the adversary which is the sum of all disrupted signal phases needs to be maximized with a fixed number of attack vehicles and a budget. The threat model is given by

$$\mathcal{L}^* \in \underset{\mathcal{L} \in \mathcal{L}_{\mathcal{P}}}{\operatorname{argmax}} \sum_{x_l=1} h_l \cdot x_l \tag{3.12}$$

subject to

$$\sum_{x_l=1} x_l \leq \rho \cdot |\mathcal{A}| \tag{3.13}$$

$$\sum_{x_l=1} c_l \cdot x_l \leq \mathcal{B} \tag{3.14}$$

$$x_l \in \{0, 1\}, l \in \mathbb{N} \tag{3.15}$$

where h_l is the total number of disrupted signal phases by the l-th arrival as an attack vehicle and ρ denotes the probability that a vehicle arriving at the intersection is an attack vehicle (i.e., a ghost vehicle or time spoofing vehicle). Eq. 3.13 gives the total number of attack vehicles constrained by $\rho \cdot |\mathcal{A}|$ which is cast into an integer number. Notice that it will not be realistic if majority of the vehicles are attack vehicles as that can easily be detected. Hence, the number of attack vehicles is limited to a fraction of all arrivals. In Eq.3.14, c_l is the cost of assigning the l-th arrival to be an attack vehicle and $\mathcal{B}$ denotes the adversary's finite budget. In this study, the cost of assigning any vehicle to be an attack vehicle can be different depending on distinct positions.

Given a budget $\mathcal{B}$, the cost of assigning a vehicle to be an attack vehicle c, and the probability of assigning a vehicle to be an attack vehicle ρ, the goal of the adversary is to determine the attack strategy $\mathcal{L}^*$ that maximizes the number of disrupted signal phases. For example, in a case of a time spoofing attack, $\mathcal{M}_{i,j}(t)$ will be greater than $W_{i,j}(t)$ in Eq. 3.5 due to a falsified arrival time sent out by the time spoofing vehicle in lane i destined to lane j at time slot t. Similarly, in the case of a ghost vehicle attack, $\mathcal{M}_{i,j}(t)$ will be less than $Q_{i,j}(t)$ in Eq. 3.4 because the TSC will not be aware of the ghost vehicle in lane i for lane j at time slot t. DBPC and QBPC algorithms will be misled by $\mathcal{M}_{i,j}(t)$ (i.e., false waiting time or false queue length), and not assign green time to the signal phase with the maximal pressure. In these cases, $y_l(t) = 1$ to indicate that the attack vehicle succeeded in disrupting the signal phase at time slot t. Consequently, victim vehicles, which are supposed to get green time to pass through the intersection at time slot t, may have to wait for several time slots to get scheduled. As a result, the delay of each vehicle in the victim lane will increase.

3.4.4 Branch-and-Bound Solution

Even for an offline case where the number of vehicles and arrival times are known, the proposed threat model formulated as the 0/1 Knapsack problem is an NP-hard problem [33] and can be solved by using a branch-and-bound (BnB) algorithm [43, 63]. A BnB algorithm is designed to effectively search optimal solutions for optimization problems, especially those that involve combinatorial optimization. The algorithm performs a state space search to systematically enumerate all candidate solutions on a rooted tree that is formed by the set of candidate solutions. It is able to terminate branches with poor solutions early in average cases and exclude nodes that are over a given estimated bound. Branches of the tree are subsets of the solution set. Each branch is explored and checked against an upper bound (UB) and a lower bound (LB) on the optimal solution. The branches will be discarded if they are unable to generate better solutions than the best one found so far.

An example of applying BnB to solve the threat model formulated in Eq. 3.12 is illustrated in Fig. 3.4. Since the BnB can only solve minimization problems, we converted the maximization problem to a minimization problem. For each node in the tree, an estimated UB (without fraction), sum of all the disrupted signal phases h (with fraction), and sum of all the cost c is calculated. Given a set of arrivals $\mathcal{A}$, the best attack strategy $\mathcal{L}^*$ can be determined by assigning attack vehicles among the arrivals. In the *node 1*, initially, UB $= -(h_2 + h_3 + h_4) = -6$, $h = -(h_2 + h_3 + h_4 + h_1 \times \frac{7}{10}) = -8.8$, and $c = 0$. Then, the assignment starts out with x_1 where x_l represents the l-th position in the arrivals. If x_1 is selected to be an attack vehicle, then move to the *node 2* in which UB as well as h will not be changed and by including x_1, $c = c_1 = 10$. Otherwise, move to the *node 3* where UB and h need to be updated since x_1 is not considered as an attack vehicle. After re-calculation, UB $= -(h_2 + h_3 + h_4) = -6$, $h = -(h_2 + h_3 + h_4) = -6$. The same steps are repeated to check the rest of x_l until the optimal strategy is reached. As the adversary searches through this state space, nodes with the least h are explored to accelerate the search process by using the least cost branch-and-bound (LC-BnB) algorithm. The optimal solution for the above example is found to be $\mathcal{L}^* = \{1, 0, 1, 1\}$

$\rho \cdot |\mathcal{A}| = 4$

$\mathcal{B} = 17$

l	1	2	3	4
h	4	2	3	1
c	10	3	5	2
h/c	0.4	0.667	0.6	0.5

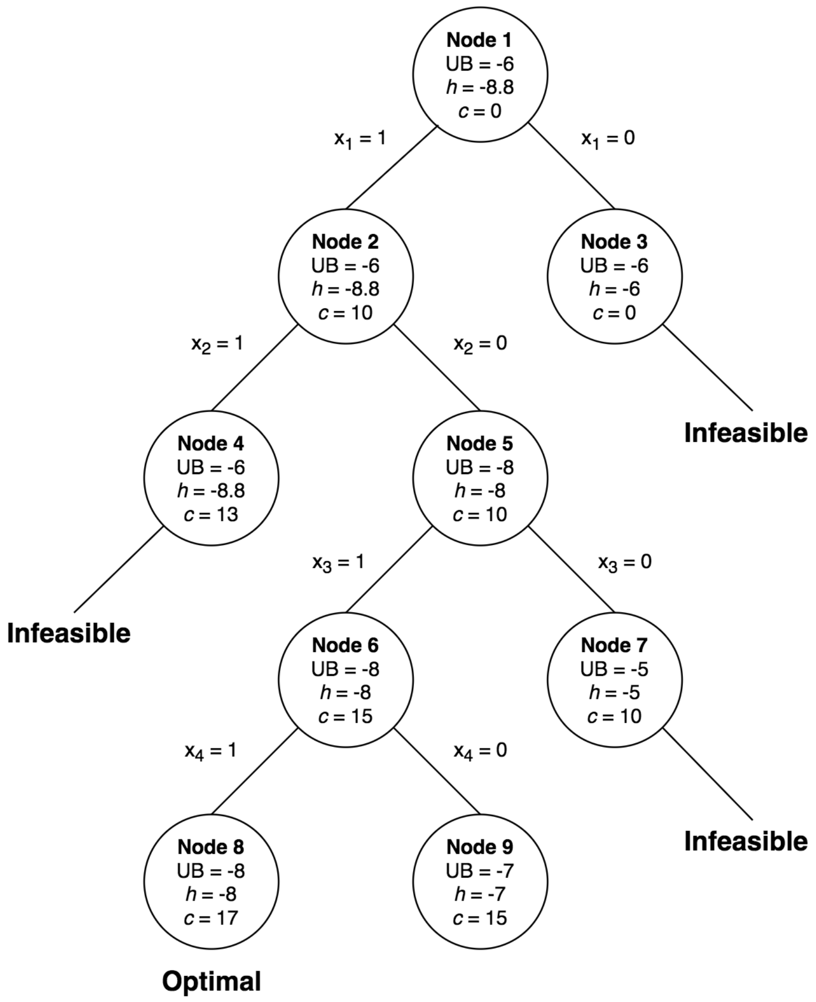

Node 1 UB = -6 h = -8.8 c = 0
x1 = 1
x1 = 0
Node 2 UB = -6 h = -8.8 c = 10
Node 3 UB = -6 h = -6 c = 0
x2 = 1
x2 = 0
Infeasible
Node 4 UB = -6 h = -8.8 c = 13
Node 5 UB = -8 h = -8 c = 10
Infeasible
x3 = 1
x3 = 0
Node 6 UB = -8 h = -8 c = 15
Node 7 UB = -5 h = -5 c = 10
x4 = 1
x4 = 0
Infeasible
Node 8 UB = -8 h = -8 c = 17
Node 9 UB = -7 h = -7 c = 15
Optimal

Figure 3.4: An illustration of the branch-and-bound (BnB) algorithm for solving the threat model.

3.5 Protection Algorithms

Two proposed protection algorithms are discussed in this section. Based on our knowledge of different types of attacks, we propose an auction-based protection algorithm (APA) for DBPC algorithm which determines phases by using HOL sojourn time, and a hybrid-based protection algorithm (HPA) for QBPC algorithm which determines phases by utilizing aggregate information. They are described in the following subsections.

3.5.1 The Auction-based Protection Algorithm (APA)

In [17], a traffic control algorithm was proposed employing a market-based pricing strategy in which for conflicting traffic movements, the most valued trip is given the highest priority. We borrow the idea from Vickrey sealed bid auction [9] in which the highest bidder wins but the winner only pays the second-highest price. Vickrey auction push bidders to report their values truthfully. In a time spoofing attack, attack vehicles falsify their arrival time to make their waiting time high. We propose a protection mechanism based on a modified Vickrey auction system in which the second-highest bidder wins. Considering the second-highest bidder to win the auction round is an effective strategy to disincentivize the adversaries to randomly falsifying their arrival times and make their waiting time arbitrarily high. This is particularly the case when it is difficult to infer what the second-highest bid is. The protection algorithm following this idea is outlined in **Algorithm 1**.

As mentioned in Section 3.3, each signal phase consists of two non-conflicting traffic movements. Each signal phase acts as a bidder and at the beginning of each time slot places a bid which is the pressures from its own non-conflicting traffic movements (line 4). This pressure could be the HOL delay in the case of the DBPC algorithm or the queue length in the case of the QBPC algorithm or the sum of the delays of the vehicles in the queue in the case of the SBPC algorithm. Signal phases (bidders) compete with each other in order to win the bid. From line $3 \sim 13$, the TSC selects the bidder (signal phase) $B_{\vec{p}}^{\dagger}$ with the second highest bid (pressure) to schedule at the time slot. For example in the case of the DBPC algorithm, APA assigns green time to the bidder with the second-highest HOL waiting time instead of a bidder who exaggeratedly spoofs its HOL waiting time far higher than other bidders.

3.5.2 The Hybrid-based Protection Algorithm (HPA)

As mentioned in Section 3.4.1.1 and 3.4.1.2, misinformation sent by ghost vehicles and time spoofing vehicles impact QBPC and DBPC algorithms differently. The basic idea of HPA is to dynamically switch between QBPC and DBPC algorithms depending on the type of attack vehicles (ghost vehicles or time spoofing vehicles). We consider the hybrid algorithm proposed in [117], in **Algorithm 2** and adopt $\eta_{i,j}^{(W)}$ and $\eta_{i,j}^{(Q)}$ to give different weights to the delay information $W_{i,j}$ and queue length $Q_{i,j}$ by adjusting the parameter r in line 5.

Algorithm 1 Auction-based Protection Algorithm (APA)

```
    Input:
    Set of feasible signal phases $\mathcal{S}_{\mathcal{P}}$, highest bidder $B^*$, second-highest bidder $B^\dagger$, and HOL vehicle waiting time $W_{i,j}(t)$ for all $(i,j) \in \mathcal{E}$ during time slot $t$.
    Output:
 1: Signal Phase $\vec{p}^{\,*}, \vec{p}^{\,\dagger} \in \mathcal{S}_{\mathcal{P}}$ to be activated during time slot $t$.
 2: Set $B^*_{\vec{p}} = -\infty, B^\dagger_{\vec{p}} = -\infty, \vec{p}^{\,*} = \emptyset, \vec{p}^{\,\dagger} = \emptyset$;
 3: for each signal phase $\vec{p} \in \mathcal{S}_{\mathcal{P}}$ do
 4:     $B_{\vec{p}} = \sum_{p_{i,j}=1} \gamma_{i,j} \cdot W_{i,j}(t) \cdot \mu_{i,j}(\vec{p})$;
 5:     if $B_{\vec{p}} > B^*_{\vec{p}}$ then
 6:         $B^\dagger_{\vec{p}} = B^*_{\vec{p}}$;
 7:         $\vec{p}^{\,\dagger} = \vec{p}^{\,*}$;
 8:         $B^*_{\vec{p}} = B_{\vec{p}}$;
 9:         $\vec{p}^{\,*} = \vec{p}$;
10:     else if $B_{\vec{p}} > B^\dagger_{\vec{p}}$ then
11:         $B^\dagger_{\vec{p}} = B_{\vec{p}}$;
12:         $\vec{p}^{\,\dagger} = \vec{p}$;
13: end for
14: return $\vec{p}^{\,\dagger}$
```

Algorithm 2 Hybrid-based Protection Algorithm (HPA)

```
    Input:
    Set of feasible signal phases $\mathcal{S}_{\mathcal{P}}$, weighted parameter $r$, queue length $Q_{i,j}(t)$, and HOL vehicle waiting time $W_{i,j}(t)$ for all $(i,j) \in \mathcal{E}$ during time slot $t$.
    Output:
 1: Signal Phase $\vec{p}^{\,*} \in \mathcal{S}_{\mathcal{P}}$ to be activated during time slot $t$.
 2: Set $\mathcal{O}^*_{\vec{p}} = -\infty, \vec{p}^{\,*} = \emptyset$;
 3: Set $\eta^{(W)}_{i,j} = \frac{1}{1+r}, \eta^{(Q)}_{i,j} = \frac{r}{1+r}$;
 4: for each signal phase $\vec{p} \in \mathcal{S}_{\mathcal{P}}$ do
 5:     $\mathcal{O}_{\vec{p}} = \sum_{p_{i,j}=1} \gamma_{i,j} \cdot [\eta^{(W)}_{i,j} W_{i,j}(t) + \eta^{(Q)}_{i,j} Q_{i,j}(t)] \cdot \mu_{i,j}(\vec{p})$;
 6:     if $\mathcal{O}_{\vec{p}} > \mathcal{O}^*_{\vec{p}}$ then
 7:         $\mathcal{O}^*_{\vec{p}} = \mathcal{O}_{\vec{p}}$;
 8:         $\vec{p}^{\,*} = \vec{p}$;
 9: end for
10: return $\vec{p}^{\,*}$
```

3.6 Experimental Results

To evaluate performance of BP-based traffic control algorithms under attack, we consider the distribution of delays, numbers of scheduled signal phases, and fairness as our metrics. We use Jain's Fairness Index [45] which is denoted by f and is given by

$$f(\vec{d} = [d_1, d_2, \ldots, d_M]) = \frac{(\sum_{i=1}^{M} d_i)^2}{M \sum_{i=1}^{M} (d_i)^2} \tag{3.16}$$

where d_i indicates the delay of vehicle i and M represents the total number of vehicles. The main idea of Jain Fairness Index is to compare the total delay to the delays of the individual vehicles. As the denominator becomes smaller, which means the delay of individual vehicle becomes smaller, f will increase. The system achieves a better fairness if Jain Fairness Index is higher.

3.6.1 Analysis of Offline Attack

We consider an offline scenario with multiple vehicles in each lane. We conduct simulations to examine how different attack strategies impact the performance. The adversary can perform these simulations to examine how different attack strategies compromise the TSC system and use the insight to improve upon the random attack in the online case.

Given a set of arrivals $\mathcal{A}$ and ratio of attack vehicles ρ, an adversary generates all permutations of $\mathbf{int}(\rho \times |\mathcal{A}|)$ number of attack vehicles among all the vehicles. Each of these permutations (candidates) is an attack strategy (e.g., $\mathcal{L} = \{0, 0, 0, 1, 0, 0, 0, 0, 1, \ldots, 0\}$). The adversary explores these attack strategies to determine the permutation of the attack vehicles that maximizes the number of disrupted signal phases. The attack strategy that has the maximum impact on the average delay and fairness is the *worst case* for the BP-based traffic control algorithm.

Fig. 3.5a $\sim$ 3.5c show average delay and number of disrupted signal phases (shown along the y-axes) under ghost vehicle attacks for different attack strategies (shown along the x-axis). The attack strategies are sorted in the increasing order of the number of disrupted phases. Although the average delay is not perfectly proportional to the number of disrupted phases, the results demonstrate that disruptions impact the delay performance of the QBPC algorithm. Fig. 3.5d $\sim$ 3.5f show average delays and the number of disrupted signal phases under time spoofing attacks for different attack strategies. In this case as well, signal phases disrupted by time spoofing attack degrade the delay performance of the DBPC algorithm.

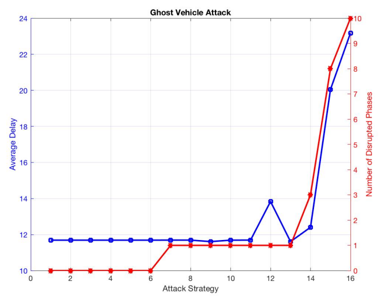
Fig. 3.6a $\sim$ 3.6c show fairness and number of disrupted signal phases for different attack strategies under ghost vehicle attacks. Fig. 3.6d $\sim$ 3.6f show fairness and number of disrupted signal phases for different attack strategies under time spoofing attacks. Similar to the average delay, fairness can be affected by disruptions. As shown in Fig. 3.6a and 3.6d both ghost vehicle and time spoofing attacks decrease fairness by one attack strategy that maximizes the number of disrupted signal phases. As the number of arrivals increases in Fig. 3.6c and 3.6f or if there is a higher ratio of attack vehicles as in Fig. 3.6b and 3.6e, the fairness decreases.

Based on the simulations, we found that most of the attack strategies do not have significant impact on the BP-based traffic control algorithms. However, some of them (specific permutations) can indeed cripple the TSC system. Although examining all possible permutations to discover the one that makes the largest impact is time-consuming and computational inefficient, the adversary can effectively reduce complexity by using BnB method mentioned in Section 3.4.4. In addition, as shown in Fig. 3.5b $\sim$ 3.5c and 3.6b $\sim$ 3.6c, ghost vehicle attacks using different attack strategies increase the number of disrupted signal phases as well as cause fluctuations in the average delay and fairness. The reason for these fluctuations is that in some cases, ghost vehicle attacks may improve the traffic control efficiency. For example, if vehicles are assigned to be attack vehicles in lanes that have a higher arrival rate, then ghost vehicle attacks can enhance the performance of the QBPC algorithm.

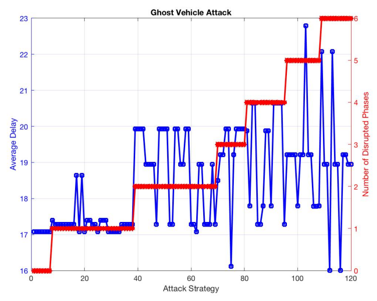
3.6.2 Analysis of Online Attack

For realistic traffic scenarios, a series of simulations were conducted to examine the impact of random time spoofing and ghost vehicle attacks under both homogeneous and heterogeneous arrivals. This section shows general cases by randomly assigning vehicles to be attack vehicles within a given set of arrivals. An attack strategy, which could be any of many candidates but not the optimal one, is a *general case* to BP-based traffic control algorithms when under attack. Based on the offline analysis, the adversary can understand how to improve random attacks.

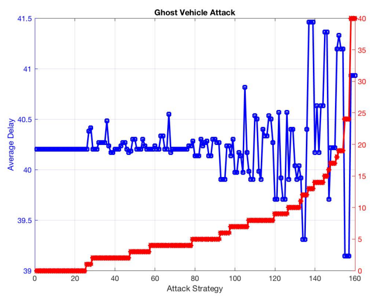
Simulation results presented in this section are based on one isolated intersection with four signal phases as shown in Fig 3.2. Based on [116, 117], the number of vehicles that pass an intersection in each lane during time slot t is $R_m(1 - e^{-\frac{Q(t)+I^a(t)}{R_m}})$ where $R_m = \mu_s T_s$ denotes the maximum number



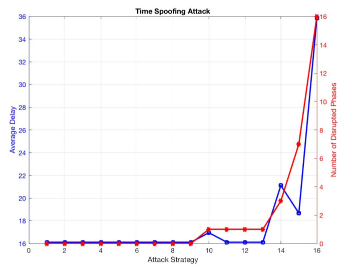
(a) $\rho = 0.05$, $|\mathcal{A}| = 16$



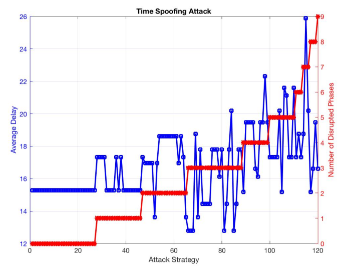
(b) $\rho = 0.1$, $|\mathcal{A}| = 16$



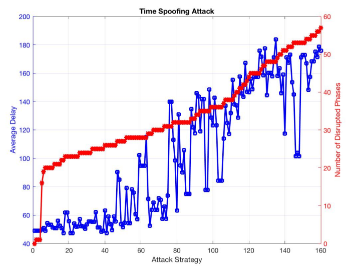
(c) $\rho = 0.00625$, $|\mathcal{A}| = 160$



(d) $\rho = 0.05$, $|\mathcal{A}| = 16$

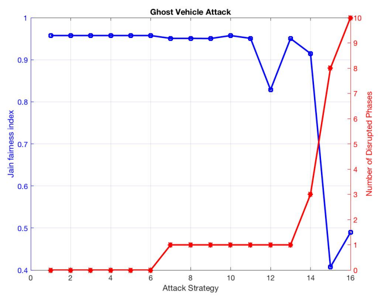


(e) $\rho = 0.1$, $|\mathcal{A}| = 16$

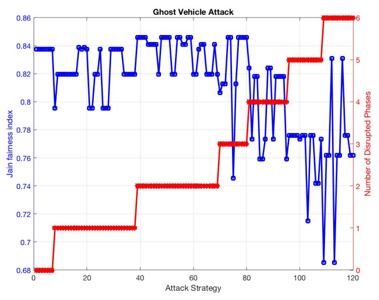


(f) $\rho = 0.00625$, $|\mathcal{A}| = 160$

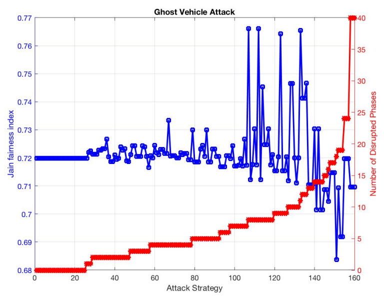
Figure 3.5: Impact of different attack strategies on average delay for BP-based traffic control algorithms. Top row shows QBPC algorithm with ghost vehicle attack and bottom row shows DBPC algorithm with time spoofing attack.



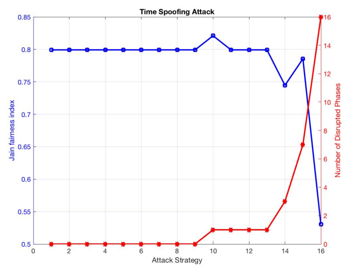
(a) $\rho = 0.05$, $|\mathcal{A}| = 16$



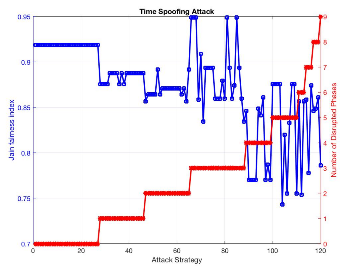
(b) $\rho = 0.1$, $|\mathcal{A}| = 16$



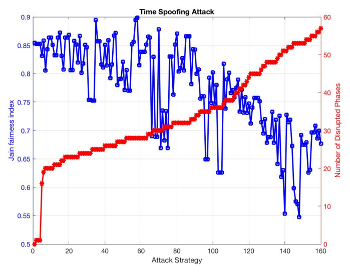
(c) $\rho = 0.00625$, $|\mathcal{A}| = 160$



(d) $\rho = 0.05$, $|\mathcal{A}| = 16$



(e) $\rho = 0.1$, $|\mathcal{A}| = 16$



(f) $\rho = 0.00625$, $|\mathcal{A}| = 160$

Figure 3.6: Impact of different attack strategies on fairness for BP-based traffic control algorithms. Top row shows QBPC algorithm with ghost vehicle attack and bottom row shows DBPC algorithm with time spoofing attack.

of passing vehicles per slot, μ_s is the saturation flow rate, $Q(t)$ represents the queue length of the lane at the beginning of the time slot t, and $I^a(t)$ indicates the number of vehicles that arrive in the lane during time slot t. μ_s is set to 0.5 and the length of each time slot, T_s, is set to 5 seconds for 8 lanes; thus, we obtain $\vec{\mu_s} = [1, 1, 1, 1, 1, 1, 1, 1] * 0.5$ v/s/l (vehicles per second per lane). We assume that the arrivals follow a Poisson process. For homogeneous arrivals, the arrival rate parameters for each of the 8 lanes of $\vec{\lambda} = [1, 1, 1, 1, 1, 1, 1, 1] * 0.125$ v/s/l. For heterogeneous arrivals, the arrival rate parameters for each of the 8 lanes of $\vec{\lambda} = [0.2, 1, 1, 0.5, 0.2, 1, 1, 0.5] * 0.125$ v/s/l. All the results are based on simulation time of 10 hours.

3.6.2.1 Time Spoofing Attack on An Isolated Intersection with Homogeneous Arrival Rates

We first examine the impact of time spoofing attacks on an isolated intersection. We consider attack ratio $\rho = 0.001$ (one attack vehicle per thousand vehicles) and spoofed time $\delta = 500$ seconds. Attack vehicles are independent and they are fixed in some lanes to clearly show the impact of the attack. Based on a 10-hours simulation, there are thousands of vehicles approaching, $\rho = 0.001$ is sufficient to create traffic jams and compromise the system and each attack vehicle is able to spoof its arrival time once in one lane at any time whenever it is approaching to this lane. Consequently, the performance results of the four BP-based traffic control algorithms without attack and under attack are shown in Fig. 3.7.

Fig. 3.7a shows the delay performance of the four BP-based traffic control algorithms. We observe that they perform almost the same and the DBPC is slightly better than others because it can solve the last vehicle problem effectively. Fig. 3.7b shows the number of times each phase is selected by the four different traffic control algorithms as given in equations (3.4), (3.5), (3.6), and (3.8). The reason why the number of times each phase is scheduled by the QBPC becomes unbalanced is because the queue length is based on the number of vehicles in the lane which is an integer. Hence, under the same arrival rate, the probability that queue lengths of different lanes are the same is high. When encountering this problem, we do not deal with balancing the four phases since optimizing the performance is not our first priority in this chapter.

However, after being attacked, in Fig. 3.7e, we note that phases scheduled by the DBPC become much more unbalanced than phases scheduled by the DBPC without attack. The number of times that the DBPC schedules *phase 2* is increased from 1794 to 1838 and *phase 4* is increased from 1799 to 1822, which means attack vehicles indeed compromised the TSC system by spoofing their arrival times. When priority should be given for other phases, e.g., *phases 1 and 3* is now taken away by the lanes with attack vehicles. By spoofing the system, an adversary can acquire more scheduling times from victimized lanes. As a result, it can get scheduled quicker than other vehicles. We call this phenomenon **priority plundering**. The delay performance in Fig. 3.7d shows that the DBPC is vulnerable to time spoofing attacks.

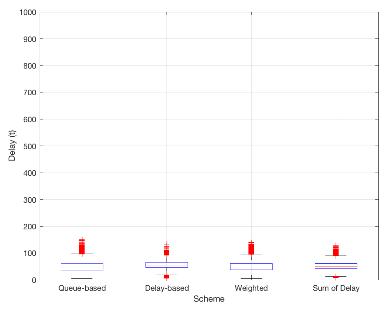
In addition, the priority plundering is also observed for the SBPC and HBPC. The variations are not as much as the DBPC because they not only take the sojourn time into account but also other factors. For example, the SBPC considers all sojourn times of all vehicles and the HBPC considers both the delay and the queue length. As a result, only a few attack vehicles do not impact their performance, and hence, they can tolerate higher attack rate than the DBPC.

Jain's fairness index shows the fairness without attack and under attack in Fig. 3.7c and 3.7f, respectively, where α represent the traffic load. For the DBPC, after being attacked, we can observe that the fairness drops more than others. The fairness of SBPC and HBPC do drop slightly, but the influence is not as large as for the DBPC since the sojourn time is just one of factors they consider. We note that the QBPC is not affected since it only takes the queue length into account and not the sojourn time.

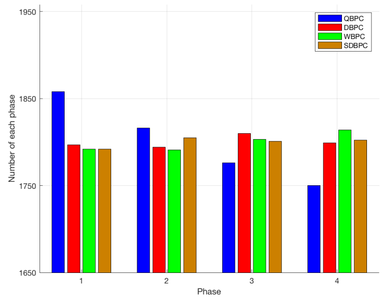
3.6.2.2 Time Spoofing Attack on An Isolated Intersection with Heterogeneous Arrival Rates

Next, we study the case of heterogeneous traffic arrival rates in 8 lanes at an isolated intersection. Specifically, we consider higher arrival rates in horizontal direction (lanes 3, 4, 7, 8) and lower arrival rates in vertical direction (lanes 1, 2, 5, 6). The parameters we use are $\vec{\lambda} = [0.2, 0.2, 1, 1, 0.2, 0.2, 1, 1] * 0.125$ v/s/l, the number of attack vehicles is 5, and spoofed time is 500 seconds.

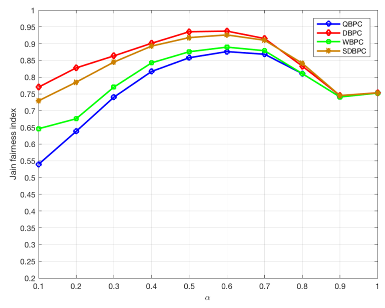
In this scenario, instead of using random attack pattern, we investigate the impact of coordinated attack to our system. This means the attack vehicles could possibly be sent as a team, trying to figure out weaknesses of the TSC system. Under this assumption, an adversary could formulate an attack strategy by sending attack vehicles in the particular lanes. Hence, we first fix the number of attack vehicles in lanes with lower arrival rates and limit attack vehicles to only appear in these lanes. Then, we fix the number of attack vehicles in lanes with higher arrival rates and limit attack vehicles to only appear in the corresponding lanes. We compare the attacks from these two directions



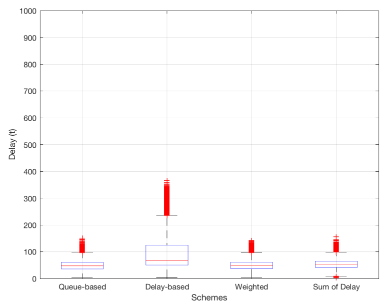
(a) Distributions of delays without attack



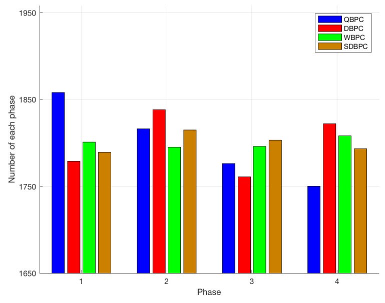
(b) Number of times four signal phases are scheduled without attack



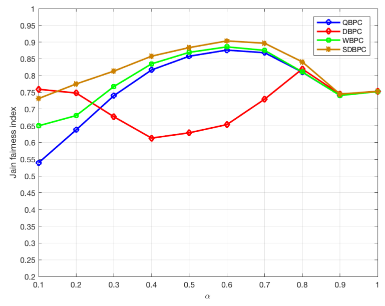
(c) Fairness without attack



(d) Distributions of delays under attack



(e) Number of times four signal phases are scheduled after being attacked



(f) Fairness after being attacked

Figure 3.7: Performance comparison of four BP-based traffic control algorithms with homogeneous traffic flows under time spoofing attack.

to see whether the adversary can benefit from such attack strategies.

Before discussing the results for the different attack strategies, we first look at Fig. 3.8a and 3.8d that show the performance of each traffic control algorithm without attack. Compared to homogeneous arrivals, the DBPC reveals it's advantage when confronting heterogeneous arrivals. Because non-homogeneous arrivals enhances the last vehicle problem from which the QBPC suffers; hence, the DBPC can outperform QBPC very much. In SBPC and HBPC, the former works similar to the DBPC because SBPC considers all delays of all vehicles. The later utilizes the queue length as well as delay and we can decide to make it closer to the QBPC or the DBPC by adjusting the parameter η. However, they both take multiple factors into account whenever making decision. As a result, heterogeneous arrivals do not affect them very much.

Results for the case under attack are shown in Fig. 3.8c, 3.8e, and 3.8f. It is observed that attacks from lanes with lower arrival rates can lengthen the total delay of DBPC. The number of times *phase 3* scheduled by the DBPC increases from 1005 to 1152 and the number of times *phase*

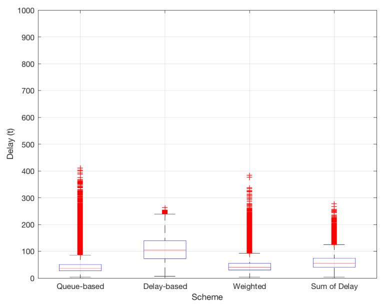
1 scheduled by the DBPC reduces from 2602 to 2596 after being attacked in Fig. 3.8e compared to 3.8b. The lanes with higher arrival rates (lanes 3, 4, 7, 8) should have more scheduling times originally, but the priority is plundered by attack vehicles in lanes with lower arrival rates. We observe that the adversary can benefit from attacking in lanes with lower arrival rates. For *phase 3* scheduled by the DBPC (red bar in Fig. 3.8e), the increasing amount of times gained by spoofing in lower arrival lanes is 147 which is higher than 111 if they had spoofed from the higher arrival lanes. The SBPC and HBPC do not increase like the DBPC. Hence, even if they have coordinated attacks from the lower arrival lanes, it will not affect SBPC and HBPC significantly.

Furthermore, we compare the fairness of being attacked in lanes with higher arrival rates and lanes with lower arrival rates in Fig. 3.8c and 3.8f, respectively. The fairness of the SBPC and HBPC does not drop very much. But for the DBPC, the results show that the fairness of being attacked in lanes with lower arrival rates (lanes 1 and 5) drops more than when the attack is launched from lanes with higher arrival rates (lanes 3 and 7). We also study the case of heterogeneous traffic arrival rates in which the horizontal direction is low and vertical direction is high. Our results show that the same behavior that the adversary can gain more scheduling times by spoofing in lanes with lower arrival rates than in lanes with higher arrival rates.

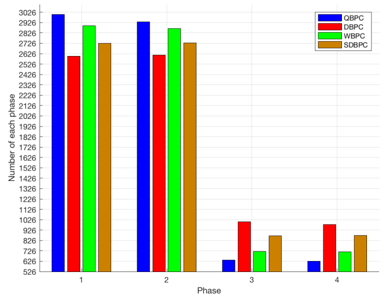
3.6.3 Ghost Vehicle Attack on An Isolated Intersection with Homogeneous Arrival Rates

We consider three different ghost vehicle attack ratios $\rho = 0.0, 0.1$, and 0.3. In order to clearly observe the impact of the ghost vehicle attack, we limit the attacks to *phase 2*, which includes traffic lanes 4 and 8.

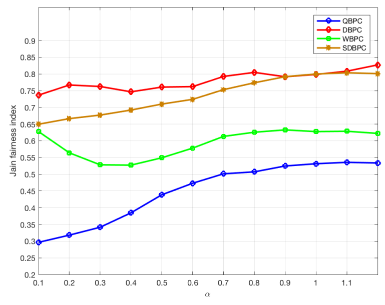
The results for different ghost vehicle attack ratios $0.0, 0.1$, and 0.3 are shown in the 1st, 2nd, and 3rd columns, respectively, in Fig. 3.9. Fig. 3.9a, 3.9d, and 3.9g show complementary cumulative distribution function (CCDF) of the delay, the number of times each phase is assigned, and Jain fairness index, for each of the four BP-based traffic control algorithms with $\rho = 0$, i.e., no attack. In general, all the BP-based traffic control algorithms perform well, and in Fig. 3.9g, performance levels of these algorithms, which are shown by the Jain's fairness index, are almost equivalent to each other. QBPC has a higher delay when compared to the other BP-based traffic control algorithms. The reason why DBPC (red line) achieves a better fairness than QBPC was discussed in [122] and [117].



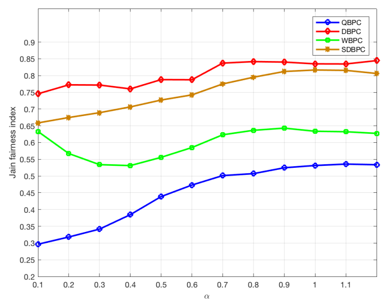
(a) Distributions of delays no attack



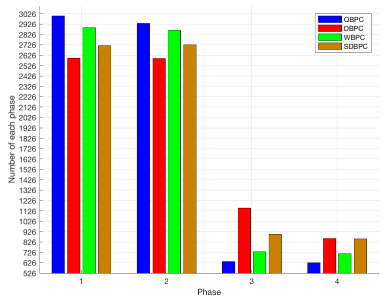
(b) Number of times four signal phases are scheduled no attack



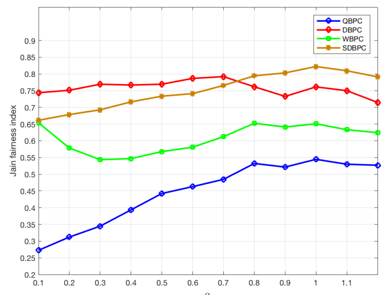
(c) Fairness after being attacked by lane 3 and 7



(d) Fairness without attack



(e) Number of times four signal phases are scheduled under attack



(f) Fairness after being attacked by lane 1 and 5

Figure 3.8: Performance comparison of four BP-based traffic control algorithms with heterogeneous traffic flows under time spoofing attack.

On the other hand, 2nd, and 3rd columns show their performance for different ghost vehicle attack ratios. First, in Fig. 3.9e, we observe the reduction in the number of times *phase 2* (lanes 4 and 8) is scheduled. Second, when comparing Fig. 3.9b to Fig. 3.9a, we can see that ghost vehicle attacks change the distribution of delays for traffic control algorithms where QBPC (blue), HBPC (green), and SBPC (brown) are increased from 151.56 to 177.67, 139.12 to 160.57, and 137.04 to 142.55, respectively. As we increase ρ to 0.3 in Fig. 3.9c, the maximum delay for QBPC exceeds 249 sec and the maximum delays for HBPC and SBPC are also increased to 191.83 sec, and 178.89 sec, respectively. Third, their fairness dropped in Fig. 3.9h when ghost attack ratio is 0.1; however, it is not very obvious. The reduction in fairness becomes worse when ρ is set to 0.3 (as shown in Fig. 3.9i).

We found that ghost attacks are quite detrimental to BP-based traffic control algorithms that determine signal phases based on aggregate information. Assigning 10% of all vehicles to be attack vehicles (i.e., $\rho = 0.1$) can disrupt the traffic control decisions determined by the QBPC algorithm.

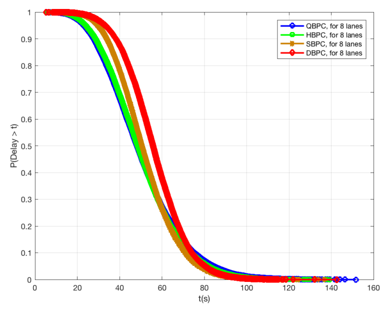
As can be observed in Fig. 3.9e and 3.9f, by increasing ρ from 0.1 to 0.3, ghost vehicle attack can result in imbalance in the allocation of signal phases among the four different phases. Note that this random attack can be improved upon by using a more planned attack based on solving the optimization problem in Eq. 3.12.

In general, ghost vehicle attacks degrade the performance of QBPC, HBPC, and SBPC algorithms and fully exploit the vulnerability of these traffic control algorithms when some of the vehicles are disconnected so that the data sent to the TSC system become totally inaccurate or useless. The QBPC algorithm requires the number of vehicles while the SBPC algorithm requires the sum of all individual waiting times in the different traffic lanes. With ghost vehicles, QBPC and SBPC algorithms would have incorrect data on the number of vehicles and the sum-of-delays. Depending on the intensity of the attack, these algorithms will produce incorrect scheduling decision resulting in disrupted signal phases. However, the DBPC algorithm is less affected by ghost vehicle attacks because it determines signal phases by HOL waiting time instead of aggregate queue information. The DBPC algorithm can be vulnerable only when the HOL vehicle is a disconnected vehicle during time slot t.

3.6.3.1 Ghost Vehicle Attack on An Isolated Intersection with Heterogeneous Arrival Rates

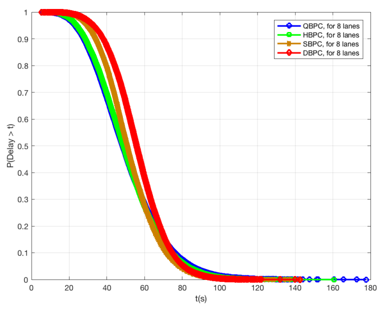
In this simulation, we limit attacks to only lanes 3 and 7, which are two non-conflicting traffic movements in *phase 1*. We originally assumed that the performance of QBPC was supposed to drop. However, the result will go opposite if lanes 3 and 7 are attacked. Fig. 3.10b shows the distribution of the delays when ghost vehicle attacks are launched in lanes 3 and 7 with a ghost vehicle ratio $\rho = 0.3$. Compared to the tail of delay distribution with a maximum delay of approximately 900 sec without ghost attacks (shown in blue line in Fig. 3.10a), the QBPC algorithm has a shorter tail of the delay distribution with a maximum delay of approximately 800 sec when under attack (blue line in Fig. 3.10b). Furthermore, the performance of the QBPC algorithm further improves if ghost vehicle attack ratio is increased to 0.5.

The reason of above improvement in performance after being attacked is because the lanes in which we deploy attack vehicles. The number of vehicles each lane receives is proportional to its arrival rate. For example, lanes 3 and 7 have high arrivals with $\lambda = 1$ compared to lanes 1 and 5 with $\lambda = 0.2$ and as a result, receive more vehicles. Under ghost vehicle attacks, QBPC and SBPC



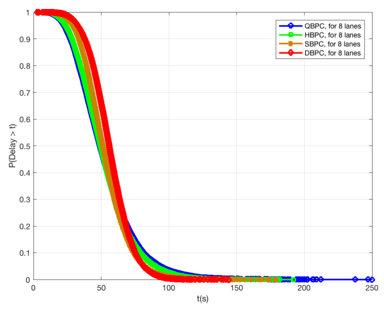
$\rho = 0$

(a) Distributions of delays without attack



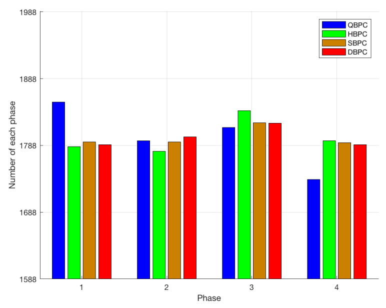
$\rho = 0.1$, Lanes 4 and 8

(b) Distributions of delays under attack



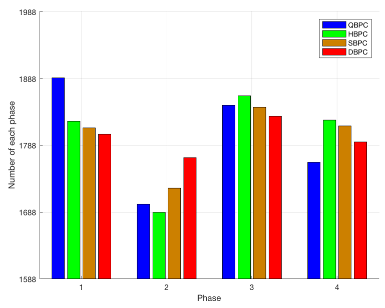
$\rho = 0.3$, Lanes 4 and 8

(c) Distributions of delays under attack



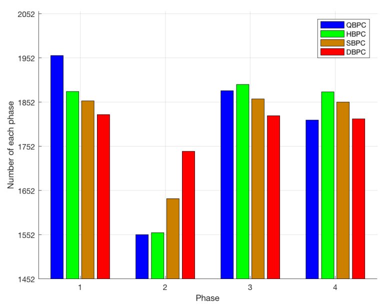
$\rho = 0$

(d) Number of times four signal phases are scheduled without attack



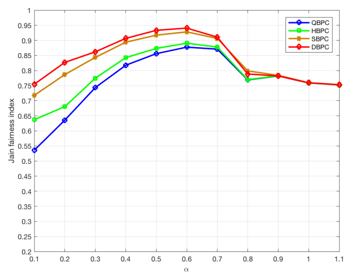
$\rho = 0.1$, Lanes 4 and 8

(e) Number of times four signal phases are scheduled after being attacked



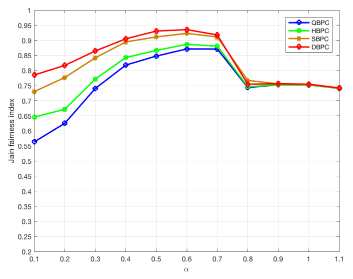
$\rho = 0.3$, Lanes 4 and 8

(f) Number of times four signal phases are scheduled after being attacked



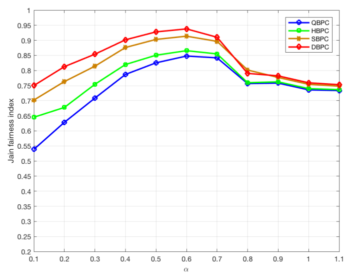
$\rho = 0$

(g) Fairness of each algorithm without attack



$\rho = 0.1$, Lanes 4 and 8

(h) Fairness of each algorithm after being attacked

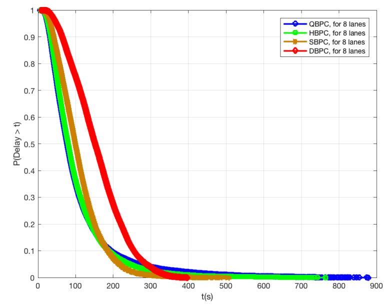


$\rho = 0.3$, Lanes 4 and 8

(i) Fairness of each algorithm after being attacked

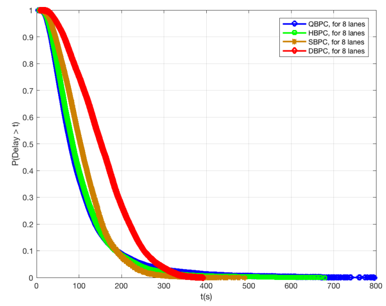
Figure 3.9: Performance comparison of four BP-based traffic control algorithms with homogeneous traffic flows under ghost vehicle attack for different ghost vehicle ratios (ρ).

algorithms operate on incorrect information of queue length and total delay. As a result *phase 1* is scheduled fewer number of times than what is due and the difference is assigned to the other phases. Consequently, even if lanes 3 and 7 in *phase 1* cannot acquire enough times, those time slots are distributed to other signal phases; hence, the overall performance of QBPC algorithm eventually gets improved.



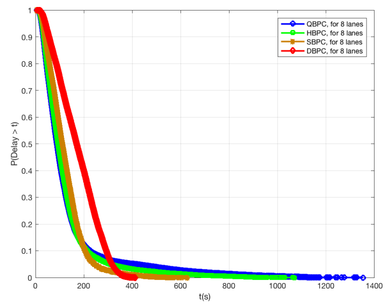
$\rho = 0$

(a) Distributions of delays without attack



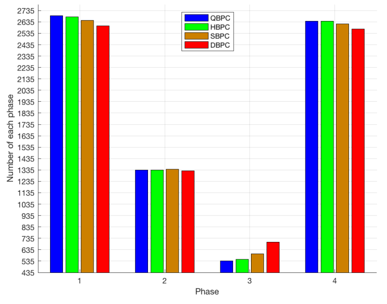
$\rho = 0.3$, Lanes 3 and 7

(b) Distributions of delays under attack



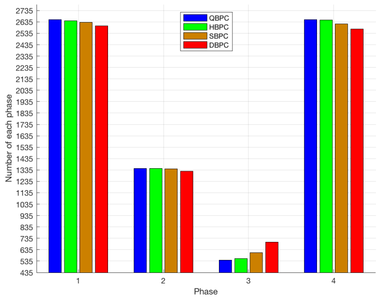
$\rho = 0.3$, Lanes 1 and 5

(c) Distributions of delays under attack



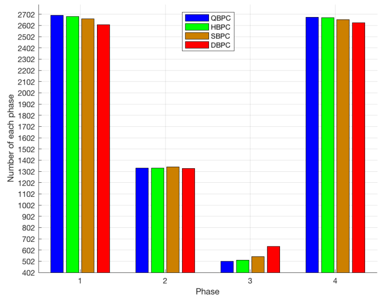
$\rho = 0$

(d) Number of times four signal phases are scheduled without attack



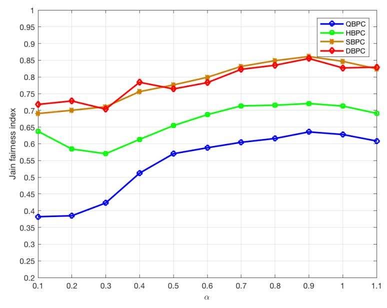
$\rho = 0.3$, Lanes 3 and 7

(e) Number of times four signal phases are scheduled after being attacked



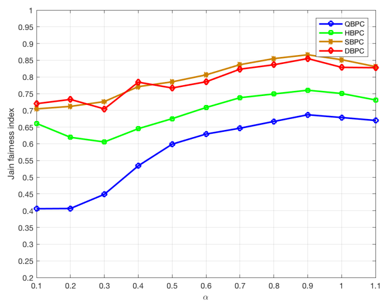
$\rho = 0.3$, Lanes 1 and 5

(f) Number of times four signal phases are scheduled after being attacked



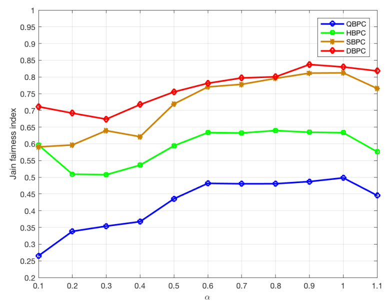
$\rho = 0$

(g) Fairness of each algorithm without attack



$\rho = 0.3$, Lanes 3 and 7

(h) Fairness of each algorithm after being attacked



$\rho = 0.3$, Lanes 1 and 5

(i) Fairness of each algorithm after being attacked

Figure 3.10: Performance comparison of four BP-based traffic control algorithms with heterogeneous traffic flows under ghost vehicle attack in different traffic lanes.

In order to prove our hypothesis that a lower arrival rate benefits ghost vehicle attacks, we limit the attacks to only appear in lanes 1 and 5 which are two non-conflicting traffic movements in *phase 3*. Fig. 3.10c shows the distributions of the delays for traffic control algorithms in which attacks are launched in lanes 1 and 5 (lower arrivals). Fig. 3.10i shows the fairness for traffic control algorithms in which attacks are launched in lanes 1 and 5 (lower arrivals). It reveals that the adversary is able to degrade the TSC if attacks are launched in lanes with lower arrivals (lanes 1 and 5). In Fig. 3.10c, the tail of distribution of delay for QBPC with attack ratio $\rho = 0.3$ is longer than the tail of distribution of delay for QBPC with attack ratio $\rho = 0$. In Fig. 3.10i, the fairness of QBPC with attack ratio $\rho = 0.3$ is worse than the fairness of QBPC with attack ratio $\rho = 0$. Moreover, the performance will become even worse if ghost attack ratio is increased to 0.5. Compared to Fig. 3.10b where ghost attacks appear in lanes 3 and 7 (higher arrivals), the tail of distribution of delay for QBPC in Fig. 3.10c increases to about 1400 sec. Compared to Fig. 3.10h where ghost attacks appear in lanes 3 and 7 (higher arrivals), the fairness in Fig. 3.10i drops more than the fairness for each algorithm in Fig. 3.10h. In short, launching ghost vehicle attacks in lanes with higher arrivals can improve the performance. On the other hand, launching ghost vehicle attacks in lanes with lower arrivals can degrade the performance.

3.6.4 Coordinated Attack using Time Spoofing and Ghost Vehicles

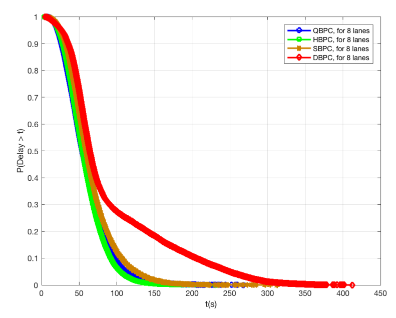
A detailed analysis on the time spoofing attack has been done in our previous work [122]. In the following scenario, we consider a coordinated attack strategy which could happen in a highly developed city in which many workers and commuters from countrysides cause homogeneous arrivals and a well-designed infrastructure contributes a lower number of disconnected vehicles. The coordinated attack is a combination of different types of attacks, many of which would be launched together during a specific range of time in the same intersection. We choose the time spoofing attack and ghost attack as the coordinated attack strategy where the time spoofing attack ratio is $\rho_t = 0.001$, the ghost attack ratio is $\rho_g = 0.45$ and homogeneous arrival rates are $\vec{\lambda} = [1, 1, 1, 1, 1, 1, 1, 1] * 0.125$ v/s/l in 8 lanes.

Fig. 3.11a shows delays of all the BP-based traffic control algorithms which undergo a coordinated attack. As can be observed, when comparing to Fig. 3.9a (without attacks), the distribution of delays become various among algorithms. Various levels of increments in delay imply that a coordinated attack has a different impact on each individual algorithm. The pressure an algorithm adopts decides whether the algorithm is able to tolerate this attack. For instance, although DBPC is able to withstand ghost attacks well, it suffers from time spoofing attacks caused by a small number of attack vehicles. In Fig. 3.11b, the number of times phases scheduled by QBPC and DBPC (blue bar and red bar) are unequally distributed than the other BP-based traffic control algorithms when comparing to Fig. 3.9d. This uneven number of times among phases causes delay increments and fairness downgrades, especially for QBPC and DBPC. The performance of all the BP-based traffic control algorithms are dropped, especially for DBPC. It only requires a tiny number of attack vehicles spoofing arrival time to sabotage the performance of the DBPC. In addition, ghost ratio needs to be very high to observe the difference.

However, there is one compelling phenomenon in which we call it the **attack elimination**, that is, even if the unfair distribution happens in QBPC, the performance measurement shows that QBPC does not decrease as much as DBPC in Fig. 3.11c. We infer that is because ghost vehicles are randomly distributed in 8 lanes; therefore, impacts caused by ghost vehicles could be eliminated by conflicting directions. Imagine that there are total 12 vehicles in the *south-north* direction and 11 vehicles in the *east-west* direction, each of which has 4 ghost vehicles. The priority is supposed to give to the *south-north* direction originally due to the highest pressure in it. But, from the system point of view, the controller is only aware of 8 vehicles in the *south-north* direction and 7 vehicles in the *east-west* direction. It determines the vehicles in the *south-north* direction have higher priority to pass through the intersection. The attacks in different directions eliminate each other and eventually the priority goes back to the *south-north*. Hence, this could be the reason why it acquires particularly high ghost attack ratio to observe the difference. In contrast, the case of elimination is not applicable to DBPC because it always determines the phase by the longest HOL sojourn time no matter whether there are the same number of attack vehicles in the conflicting direction. Adversaries plunder the priority by the spoofed arrival time instead of the number of vehicles. Hence, even though there is equal number of attack vehicles appear in both two directions, the priority will be taken away by the attack vehicles which spoofed an exaggerated HOL sojourn time. For brevity, QBPC determines a phase by depending on the number of vehicles which can be eliminated. However, DBPC determines a phase by relying on HOL sojourn time which cannot be eliminated.

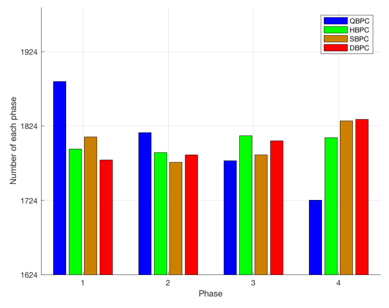
3.6.5 Performance Evaluations of Protection Algorithms

To easily demonstrate the effectiveness of the protection algorithms, for APA, we limit attack vehicles to only lanes 2 and 6 which represent *phase 4*. Fig. 3.12e reveals the distribution signal phases determined by the DBPC algorithm with and without the protection algorithm (purple and red bars)



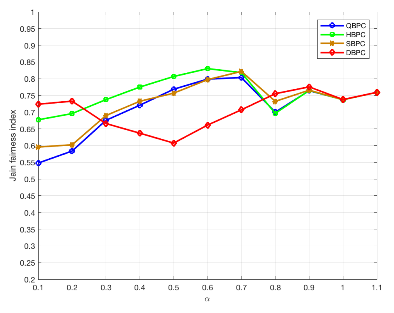
$\rho_t = 0.001,\ \rho_g = 0.45$
Random Attack

(a) Distributions of delays without attack



$\rho_t = 0.001,\ \rho_g = 0.45$
Random Attack

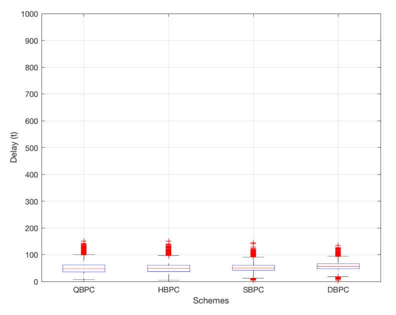
(b) Number of times four signal phases are scheduled after being attacked



$\rho_g = 0.001,\ \rho_g = 0.45$
Random Attack

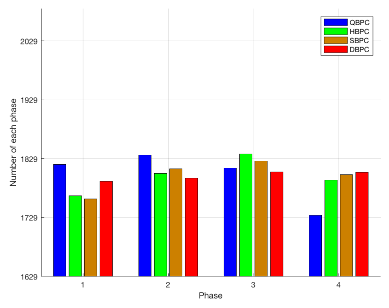
(c) Fairness of each algorithm after being attacked

Figure 3.11: Performance comparison among four BP-based traffic control algorithms for homogeneous traffic flows at an isolated intersection when being attacked by multiple types of attacks.



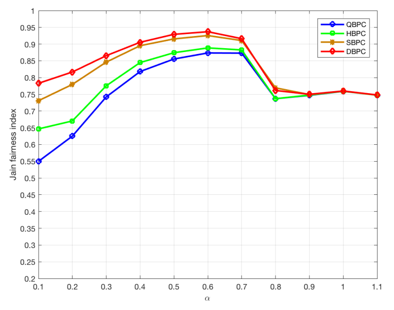
$\rho = 0$

(a) Distributions of delays without attack



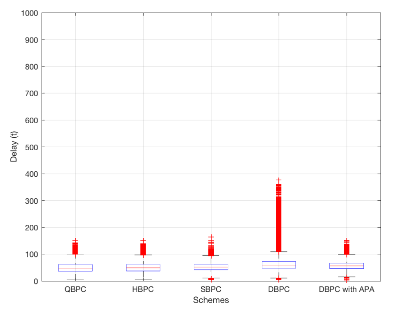
$\rho = 0$

(b) Number of times four signal phases are scheduled without attack



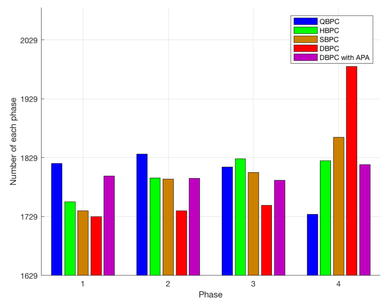
$\rho = 0$

(c) Fairness of each algorithm without attack



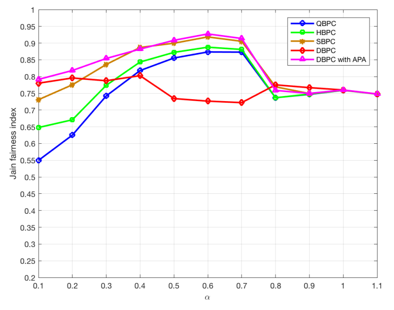
$\rho = 0.001$, Lanes 2 and 6

(d) Distributions of delays under attack



$\rho = 0.001$, Lanes 2 and 6

(e) Number of times four signal phases are scheduled after being attacked



$\rho = 0.001$, Lanes 2 and 6

(f) Fairness of each algorithm after being attacked

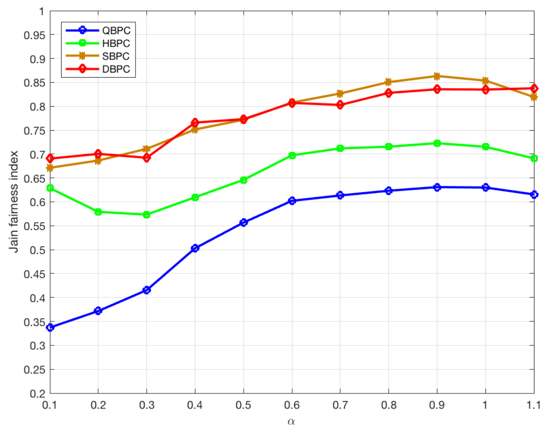
Figure 3.12: Performance of DBPC with APA with homogeneous traffic flows when under time spoofing attacks.

when under time spoofing attack. The DBPC algorithm is vulnerable to time spoofing attack and the tail of the delay distribution increases under attack. Fig. 3.12e shows that the number of times the four phases are scheduled by the DBPC algorithm is very unbalanced (red bars). With homogeneous arrivals and without attack, the number of times different phases are scheduled should be balanced. However, after applying APA, which determines signal phases by selecting the second-highest waiting time rather than the HOL waiting time, to the DBPC algorithm, we observed that the tail of the delay distribution is shorter. Furthermore, as shown in Fig. 3.12e, the number of times different phases are scheduled by DBPC with APA (purple bars) is balanced. Under time spoofing attack, the DBPC with APA (purple) achieves both performance and fairness similar to the case without attacks as shown in Fig. 3.12d and 3.12f.

For HPA, we limit attack vehicles to only lanes 1 and 5 which represent *phase 3*. Fig. 3.13 presents the results of HPA for the QBPC algorithm under ghost vehicle attack. Our strategy is to force the QBPC algorithm to act similarly to DBPC if there is a ghost vehicle attack. This is because the DBPC algorithm is not impacted by ghost vehicle attacks. The value of the weight parameter r can be tuned such that QBPC with HPA performs close to QBPC when not under attack and close to that of the DBPC when under attack. In Fig. 3.13b, the light blue curve (QBPC with HPA) with $r = 50$ achieves good fairness compared to the blue curve (QBPC without HPA). The performance of SBPC becomes unstable when the load α is low (0.1 to 0.4). At such low loads, there are only few vehicles. Consequently, under ghost vehicle attack, there is relatively higher fluctuations in the number of connected vehicles. As a result, the SBPC algorithm makes incorrect scheduling which gets amplified with lower traffic demand. Therefore, the performance of the SBPC algorithm fluctuates with α between 0.1 and 0.4.

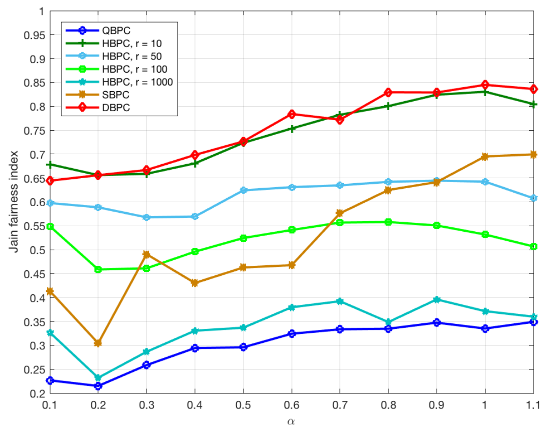
3.7 Conclusion

In this chapter, we have demonstrated how an advanced persistent adversary can perform an offline analyses on a traffic signal control to find the optimal attack strategy that maximizes the number of disruptions in the traffic phases and consequently impact on the average delay and fairness. In our simulations, by launching the time spoofing attack, the adversary can benefit from getting scheduled earlier than other vehicles. In contrast, using ghost vehicle attack, the adversary can block a platoon



$\rho = 0$

(a) Fairness of each algorithm without attack



$\rho = 0.5$, Lanes 1 and 5

(b) Fairness of each algorithm after being attacked

Figure 3.13: Performance of HBPC with HPA adopting $r = 10, 50, 100, 1000$ with heterogeneous traffic flows when under ghost vehicles attacks.

of vehicles from passing through an intersection for a while. Based our simulations we found that the two types of attacks are able to dramatically exploit the vulnerabilities of BP-based traffic control algorithms. We demonstrated that the time spoofing attack is able to reduce the performance of DBPC but has negligible impact on other traffic control algorithms that do not take HOL waiting time into account. The ghost vehicle attack has adverse impact on traffic control algorithms that determine phases by using aggregate information such as the queue length and sum-of-delays. We also showed that the proposed APA and HPA protection algorithms are able to significantly reduce the impact of time spoofing attack and ghost vehicle attack, respectively. As future work, we will extend the current environment to a network of intersections and conduct simulations to show how misinformation attacks spread through the traffic network.

Chapter 4

Deep Reinforcement Learning Based Platooning Control

4.1 Introduction

With excessively dense population in urban cities, traffic congestion has been part of our daily life and impacted on our emotion as well as the environment. Highly congested traffic lengthen the average sojourn time for people who need to commute everyday as well as increase the average fuel consumption (cost) for each driver and the average volume of noxious emissions to the environment. According to the report by INRIX [22], the annual congestion cost for each driver was 99 hours and $\$1,377$ in 2019 in the United States (US). Greenhouse gas emissions [1] contributed by transportation were 29% in 2019. Light-duty and Medium-duty (including heavy-duty) vehicles accounted for 58% and 24% of transportation greenhouse gas emissions in 2019, respectively. Undoubtedly, the ever-worsening congestion, fuel consumption, and emission issues attract public attention to address energy consumption and pollutant emissions generated by transportation systems. Thereby, many solutions with novel technologies have sprung up to address these issues.

Connected autonomous vehicle (CAV) [98] is an advanced technology which enables vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications among drivers, road side units (RSUs) and controllers. Research on CAV to improve mobility, safety and sustainability has been thriving in recent years. With an environment that most of vehicles are connected and all information is shared, platooning control can be fulfilled to improve traffic efficiency and safety. Platooning control is a technique that organizes a traffic flow into multiple groups, convoys, or platoons with close-following vehicles also known as road trains in order to increase the overall capacity of roads and reduce fuel consumption and emissions [68] in the traffic network. Platoons adopt cooperative adaptive cruise control (CACC) [39] with wireless communications to manipulate platooning formations and maneuvers. By forming vehicles into road trains, the overall throughput can be improved because gaps among CAVs are minimized. In addition, fuel consumption and emissions can be reduced due to lower air drag and speed variation for each CAV in a platoon [68]. Platooning can be considered as a effective control strategy for heavy-duty vehicles (HDVs) as well as a promising solution to reduce fuel consumption in HDVs [5, 14, 97].

However, platooning control methods requiring complicated computations are generally time-consuming and not amenable for real-time operations. Such intensive computation problems, on the other hand, are perfect applications for machine learning (ML). Deep reinforcement learning (DRL), one branch of many ML techniques, is an unsupervised learning framework that integrates reinforcement learning (RL) [91] with neural networks. DRL can be applied to an episodic traffic environment where a DRL agent takes pre-defined actions based on observations, and then receives rewards from the environment. The DRL agent learns to determine the optimal speed and size of AV platoons that maximize the expected cumulative reward in order to reduce fuel consumption. In this research, we propose a DRL-based approach that selects the optimal speed and size for each AV platoon to manipulate platoon maneuvers so that each AV platoon can either cross the intersection as a whole without stopping or be split into two sub-platoons that the former sub-platoon can cross without stopping. The goal is to minimize the number of stopping vehicles to reduce fuel use and emissions.

4.1.1 Organization

The rest of this chapter is organized as follows. We list our contributions in Section 4.2. In Section 4.3, we introduce the problem and system model for the platooning control. In Section 4.4, we propose a DRL framework for platooning control to minimize fuel consumption of platoons approaching and leaving an intersection. In Section 4.5, we discuss the performance of the DRL-based agent regarding fuel consumption and delay. Finally, we conclude and discuss future work in Section 4.6.

4.2 Contribution

- We propose a DRL framework that integrates the deep learning, dueling architecture, and experience replay memory for platooning control to minimize fuel consumption of platoons.
- We consider different arrival times of vehicles in a platoon and adopt a vector of arrival timings of vehicles as a state to emphasize differences among platoons.
- We apply appropriate actions to individual platoons based on their vectors of arrival timings (states) instead of using the same action for all platoons.

4.3 Problem Statement and System Model

4.3.1 Problem Statement

Autonomous vehicles provide great controllability that can lead to lower fuel use and traffic delays in transportation networks. Prior work has demonstrated that properly optimized AV platoons can reduce both travel time and emissions by as much as 40% when traveling through an isolated intersection. But the method used to achieve this results, optimal trajectory control, are complex and time consuming to solve, hence not amenable for real-time operations. Such complex problems, on the other hand, can be perfect applications for machine learning (ML). In this research, we plan to 1) using ML to optimize the AV platoon trajectories to reduce fuel use and travel delay, and 2) to consider a mixture of personal cars and buses in the AV platoon, so that the system can be optimized for personal-centric rather than vehicle-centric mobility and footprint.

4.3.2 System Model

We conduct experiments on an environment of a traffic network where several intersections are signalized and traffic movements with fixed routes, e.g., $W-E$ and $N-S$ are controlled by traffic lights with fixed-time signals as shown in Fig. 4.1. The assumption of a high penetration ratio of connected autonomous vehicles (CAVs) is made; thus, information such as velocity, acceleration/deceleration, and headway can be shared with extremely low latency by 5G wireless technologies. Platoons with close-following vehicles arrive periodically, and their platoon maneuvers are controlled by a ML-based platooning controller aiming at reducing the average fuel consumption. The design of a row of intersections instead of a grid network is to study the impact of applying platooning control on fuel savings and examine if the proposed ML-based platooning controller is capable of cleverly dividing a platoon into two when the remaining green time is not enough for all vehicles in a platoon to pass.

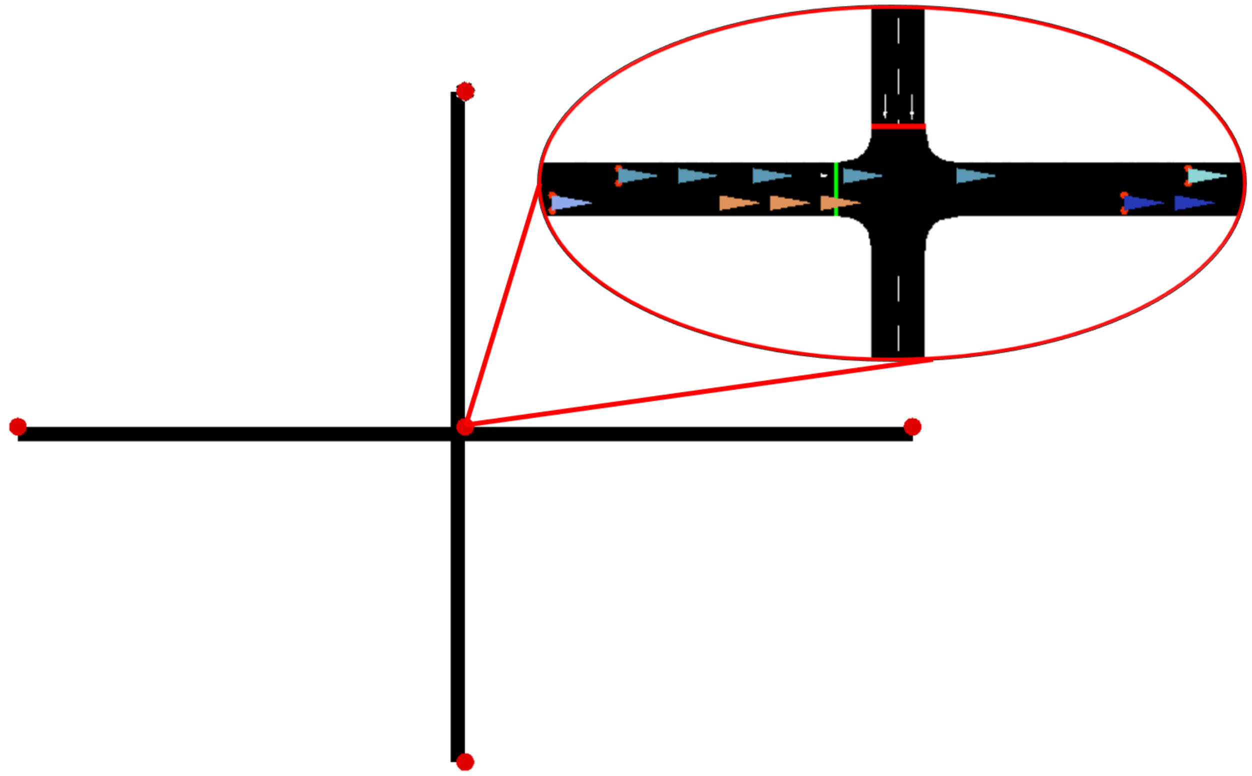


Figure 4.1: An illustration shows a row of signalized intersections where traffic flows move towards fixed routs.

4.4 The Proposed Deep Reinforcement Learning based Platooning Control

To deal with the dynamic of traffic flow and properly select the optimal platooning manipulations adaptive to various remaining green time that maximize the average fuel consumption, we adopt an unsupervised learning approach that is able to learn without human intervention. Deep reinforcement learning (DRL) is an unsupervised learning framework that integrates reinforcement learning with neural networks as shown in Fig. 4.2. We implement two control modules in the proposed framework: traffic light control (TLC) and platooning control (PC). TLC is responsible for controlling traffic lights using fixed-time signals; for example, 20s of green interval plus 3s of yellow change interval for $N - S$ 20s of green interval plus 3s of yellow change interval for $E - W$. PC is responsible for managing the speeds, sizes of the platoons in the environment based on the selected actions from the DRL agent. A DRL agent learns to select actions that maximize the expected cumulative reward by trial and error manner. In this project, we propose a deep reinforcement learning based (DRL-based) algorithm which trains a DRL agent to determine the optimal action every 2s (a time step) to optimize the average fuel consumption. For each time step, the DRL agent interacts with the traffic environment where it first chooses an action (one of the four platooning manipulations), observes some changes from the environment (the next state), and receives a reward. Given the observations and reward, the DRL agent computes a difference value (i.e., loss) from the Q-network and target network and updates the neural networks by the gradient of the loss value.

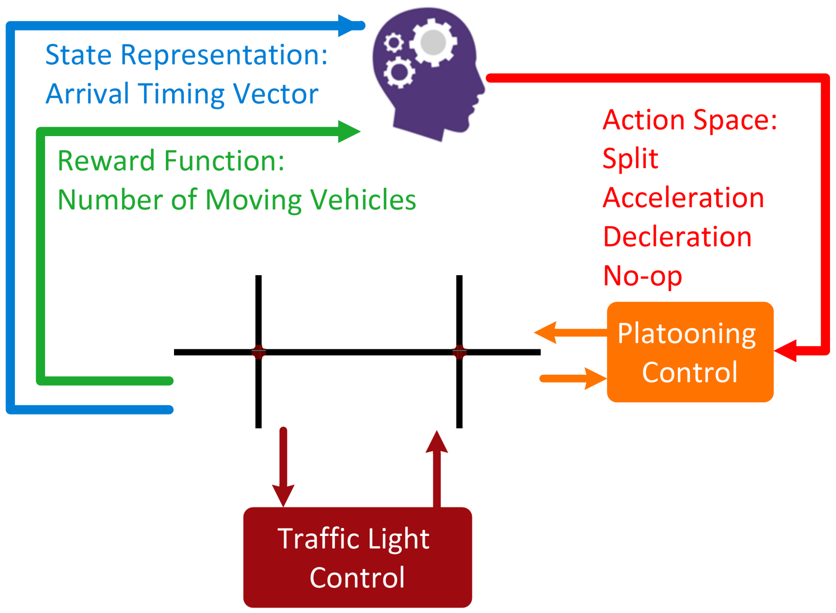


Figure 4.2: The proposed DRL platooning control framework from which The DRL agent learns to select the actions that return the maximum expected cumulative reward. There are two control modules in the proposed framework: traffic light control (TLC) and platooning control (PC). PC is responsible for managing platoons and TLC is responsible for controlling traffic lights using fixed-time signals; for example, 20s of green interval plus 3s of yellow change interval for $N - S$ 30s of green interval plus 3s of yellow change interval for $E - W$

4.4.1 State Representation

Recent research [62, 120] has demonstrated that RL can be trained by using images as states and the others [32, 34, 58, 100, 115] had formulated a matrix for an intersection by marking 0 and 1 based on coordinates of vehicles to describe a state. In these previous papers, one characteristic they all have in common is that one state is mapped to one action. Namely, they considered a fully observable environment as a state and apply an action to an environment. However, in our case, it is unrealistic to map a state to a action and then, apply the same action to all platoons in the environment because their profiles such as speeds, sizes are different; thus, different actions are required for platoons with different profiles.

In response to this issue, we treat platoons distinctively by generating different actions according to their profiles. We formulate a state as a vector of arrival timings for an individual platoon. The arrival timing vector (ATV) includes arrival timings for all the vehicles in a platoon. An arrival timing of a vehicle is determined by three arrival timing variables proposed in [107]. The three arrival timing variables are 1) cruising time-to-arrival t_c, 2) earliest time-to-arrival t_e, and 3) latest time-to-arrival t_l. They can be calculated by

$$t_c = \frac{d_1}{v_1} \tag{4.1}$$

$$t_e = \frac{d_1 - v_1 \cdot \frac{\pi}{2\alpha}}{v_{lim}} + \frac{\pi}{2\alpha} \tag{4.2}$$

$$t_l = \frac{d_1 - v_1 \cdot \frac{\pi}{2\beta}}{v_{coast}} + \frac{\pi}{2\beta} \tag{4.3}$$

$$\alpha = min\left\{\frac{2 \cdot a_{max}}{v_{lim} - v_1}, \sqrt{\frac{2 \cdot jerk_{max}}{v_{lim} - v_1}}\right\} \tag{4.4}$$

$$\beta = min\left\{\frac{2 \cdot a_{max}}{v_1 - v_{coast}}, \sqrt{\frac{2 \cdot jerk_{max}}{v_1 - v_{coast}}}\right\} \tag{4.5}$$

where d_1 is the distance from the current position to the intersection, v_1 is the current speed of the vehicle, α and β are coefficients to calculate t_e and t_l, $jerk_{max}$ is a pre-defined constant denoting the maximum changing rate of acceleration or deceleration, v_{lim} is a pre-defined constant denoting the speed restriction of the current roadway, and v_{coast} is a pre-defined constant denoting the coasting speed.

For each vehicle in a platoon, its optimal arrival timing t_{arr} can be determined by the following rules: 1) assign t_c to be its t_{arr} if t_c is within a green interval T, 2) assign $\min\{t_e, t_c\}$ to be its t_{arr} if the intersection of $[t_e, t_c]$ and a green interval T is not empty, 3) assign $\min\{t_c, t_l\}$ to be its t_{arr} if the intersection of $[t_c, t_l]$ and a green interval T is not empty, and 4) assign the beginning time of the next green interval to be its t_{arr} if the vehicle has no choice and must stop and wait for the next green interval. Note that the rules are ranked by priority; that is to say, no need to move to rule 2, 3, and 4 if rule 1 is applicable. If rule 1 is not applicable and rule 2 is applicable, then no need to go to rule 3, and 4 and so forth. An example of different arrival timings assigned to vehicles in the same platoon is shown in Fig. 4.3. t_e^n is earliest time-to-arrival for the n-th vehicle, t_c^n denotes the cruising time-to-arrival for the n-th vehicle, and t_l^n means latest time-to-arrival for the n-th vehicle. Based on the four rules, ATV can be represented by $[t_{arr}^1, t_{arr}^2, t_{arr}^3, \dots, t_{arr}^n]$. ATV includes arrival timings of all the member vehicles can be meaningful state representation that encodes information of whether a platoon has to be split or not. For example, in Fig. 4.3, there are 6 vehicles in a given platoon. As can be observed, the former three vehicles can pass the intersection within the current green interval if they keep the current speed but the latter three vehicles must stop and wait for the next green interval. Hence, the ATV of the platoon has distinct arrival timings among member vehicles and this pattern helps the DRL agent to recognize if a platoon needs to be split to avoid fully stops and unnecessary accelerations/decelerations. Fully stops and unnecessary accelerations/decelerations are platooning manipulations which lead to additional fuel usage. The number of these misbehaviors has to be minimized to achieve optimal fuel consumption. The trajectories as shown in Fig. 4.3 demonstrates that the whole platoon can avoid fully stops if it can be split into two subplatoons at t_0. The former can keep the current speed and pass the intersection at t_1 while the latter can first decelerate and then accelerate to pass the intersection later at t_2 without any stop.

4.4.2 Action Space

Since the proposed state representation is ATV that simply encodes meaningful information for the DRL agent, a large action space would impede convergence of the learning process. Thus, we only consider four actions, namely, 1) split, 2) acceleration, 3) deceleration, and 4) no-op. These manipulations are enough to accomplish all platoon maneuvers in our simulations.

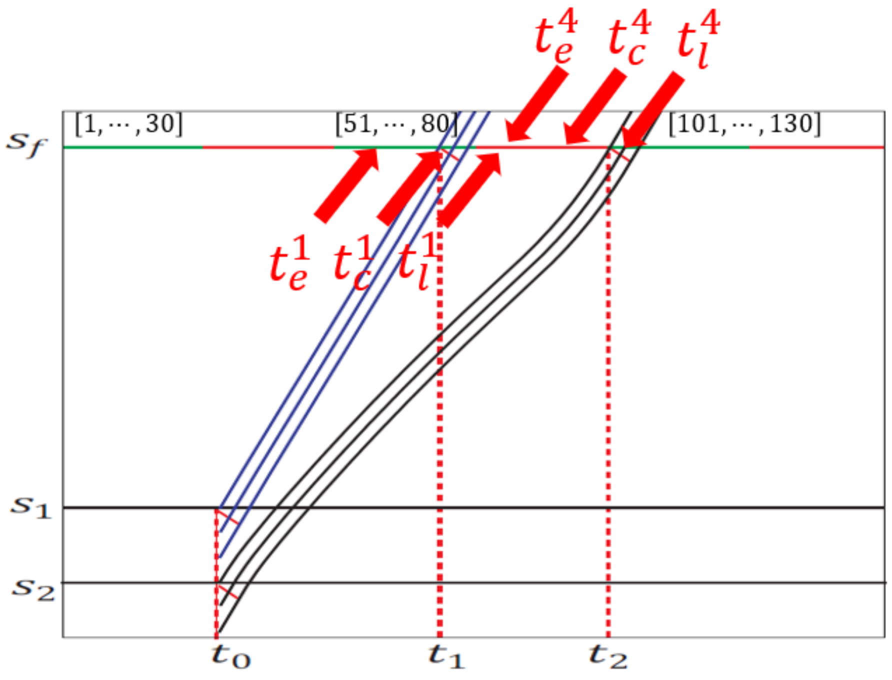


Figure 4.3: An example of different arrival timings for vehicles in the same platoon

4.4.3 Reward Function

As mentioned, the goal is to minimize the number of fully stops and unnecessary accelerations/decelerations in order to reduce fuel consumption. In other words, the number of moving vehicles has to be maximized. Thus, we choose the number of moving vehicles as the reward function.

4.4.4 The Overall Architecture

In this section, we elaborate more on the underlying architecture of the framework and introduce detailed deep learning techniques utilized to train a DRL model. The DRL architecture integrating several components that facilitate the learning process is shown in Fig. 4.4. First, usually, there is strong correlations among several consecutive states especially in time-series data. The experience memory replay component proposed in [86] is applied to break the strong temporal correlations and speeds up the learning process for the DRL agent. The second component included in the architecture is the two separated neural networks, Q-network and target network [72]. In DRL, neural networks are applied as non-linear approximation function to map a state to an action that returns the maximum Q-value. Q-network and target network share exactly the same number of neurons and architecture. The set of neural parameters θ is copied over to the target network every

τ steps. The current state and action are first fed into the Q-network for approximating a Q-value. Previous experiences sampled from the replay memory buffer are inputs to the target network for approximating a target Q-value. The loss value, L, calculated by the Q-value and target Q-value will be partially differentiated to get the gradient in which the agent will have a sense and know how to update the Q-network by backwardpropagation to fine-tune the set of neural parameters θ. The dueling architecture [110] is the third component that helps the selection of action by separating the estimations of state-value and action advantage. Instead of using one single sequence for value estimations, a Q-Network with two separated sequences was implemented, one for value (blue cycle) and the other for action advantage (brown cycle). We include this component because in some states, it matters which action needs to be taken, but most states, the selection of action has no influence on what happens. Hence, we separate the estimations of state-value and action advantage in order to learn which state-values are higher without going through and learn every single state-action pair.

4.5 Results and Discussions

In this chapter, we will elaborate on the hyperparameters in DRL framework and experimental results regarding the learning performance of the DRL agent, average delay and average fuel consumption.

4.5.1 Hyperparameters

As shown in Table 4.1, a set of hyperparameters used in the proposed DRL framework is pre-defined. Traffic arrival ratio for each intersection is 900 vehs/h/lane. The ϵ-greedy function determines the probability of doing exploration or exploitation where ϵ_i as the initializing value is 1.0, $\epsilon_f = 0.01$ is the finalizing value, and the decay ratio is 300. Learning ratio α determines the weight of newly learned knowledge. α approaching 0 makes the agent exploit prior knowledge instead of learning something new while α approaching 1 makes it explore new potential rather than using prior knowledge. α is set to 0.0001 in the simulations. Discount factor γ determines the importance of future rewards. γ closer to 0 makes the agent behave myopically by only considering immediate rewards while γ closer to 1 makes it seek a long-term outcome by weighting future rewards with a higher value. γ is set to 0.99 in the simulations. The replay memory size M used to store previous experiences is set to 30000, and mini-batch size B used to sample experiences from the memory is set to 1024. The

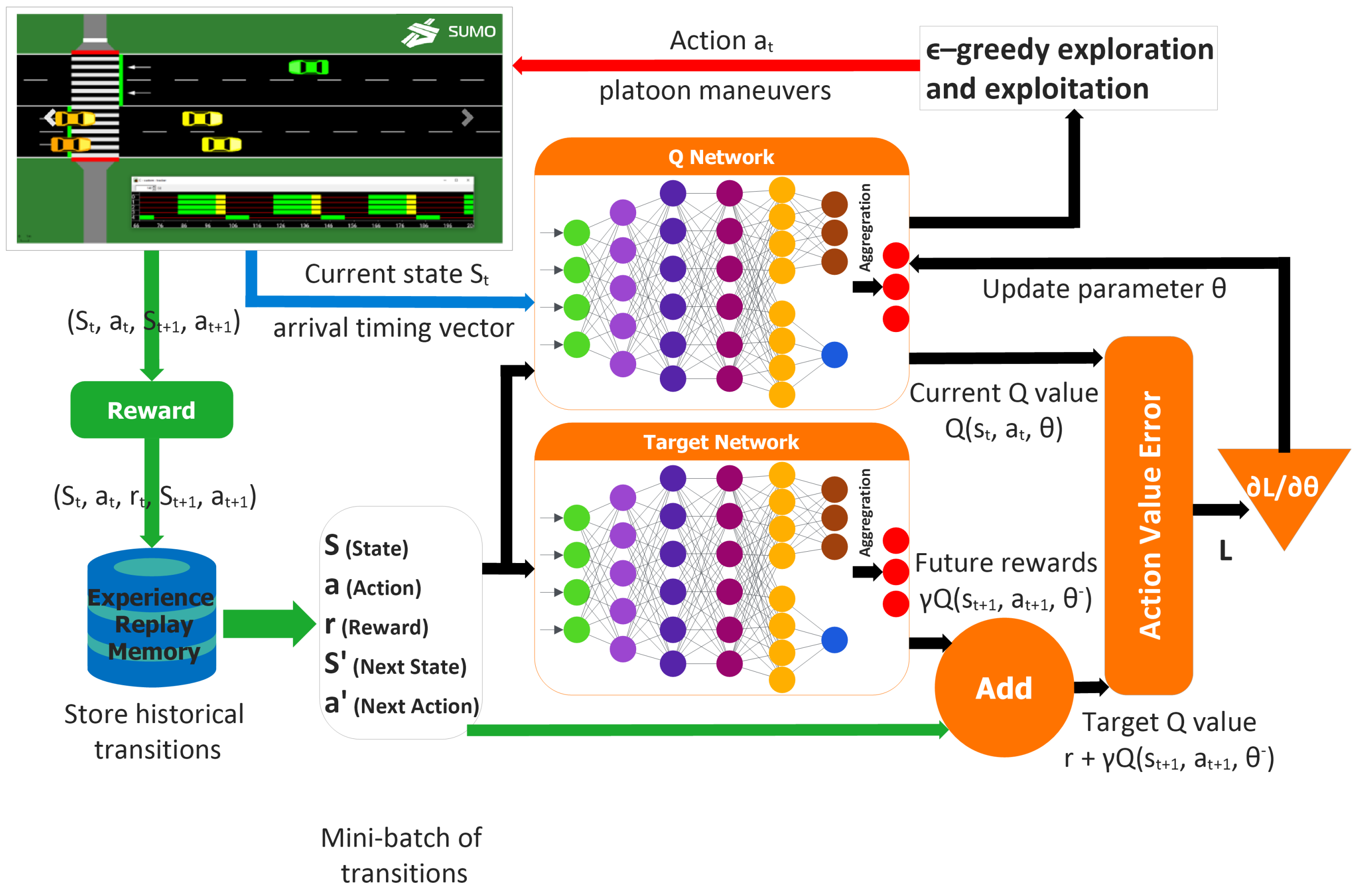

SUMO
Action a_t
platoon maneuvers
ϵ–greedy exploration and exploitation
Q Network
Aggregration
Target Network
Aggregration
Current state S_t
arrival timing vector
(S_t, a_t, S_{t+1}, a_{t+1})
Reward
(S_t, a_t, r_t, S_{t+1}, a_{t+1})
Experience Replay Memory
Store historical transitions
S (State)
a (Action)
r (Reward)
S' (Next State)
a' (Next Action)
Mini-batch of transitions
Update parameter θ
Current Q value $Q(s_t, a_t, \theta)$
Action Value Error
∂L/∂θ
L
Future rewards $\gamma Q(s_{t+1}, a_{t+1}, \theta^-)$
Add
Target Q value $r + \gamma Q(s_{t+1}, a_{t+1}, \theta^-)$

Figure 4.4: An overview of the proposed architecture for DRL agent.

number of training episode $E = 700$ is selected by trial and error.

Table 4.1: Summary of hyperparameters in the proposed DRL framework

Parameter	Description
Traffic arrival ratio:	900 vehs/h/lane
ϵ-greedy decay function:	calculate the probability of exploration
$\epsilon_f + (\epsilon_i - \epsilon_f) \times e^{-\frac{episode}{decay\ ratio}}$	and exploitation where decay ratio is 300
Initializing $\epsilon_i = 1.0$	the beginning value of ϵ
Finalizing $\epsilon_f = 0.01$	the final value of ϵ
Learning ratio $\alpha = 0.0001$	learning ratio for updating the neural network
Discount ratio $\gamma = 0.99$	discount for future rewards
Memory size $M = 30000$	the size of entire replay memory buffer
Mini-batch size $B = 1024$	the size of samples that will be reused
Maximum number of episodes $E = 1000$	the total number of training episodes

4.5.2 Learning Performance

We discuss the impact of platooning on the average fuel consumption by each platoon and the average delay of each platoon. In the simulations, the agent is allowed to explore more than exploiting at a pre-training stage. The pre-training stage is set to go off in 100 episodes. During the pre-training stage, the agent randomly selects an action instead of using the optimal one in order to discover potential actions that would return a better reward. In Fig. 4.5, the trend of the reward shows that the DRL platooning agent keeps obtaining better rewards after the pre-training stage. Namely, it had learned how to either split platoons or control the cruise speeds of the platoons to reduce fuel usage by platoons during the last stage (pre-training) and applies the knowledge to manipulate AV platoons in the current stage (training). As more and more training episodes had been learned (close to the 700-th episode), the agent learns to maximize rewards by controlling AV platoons to achieve lower fuel consumption.

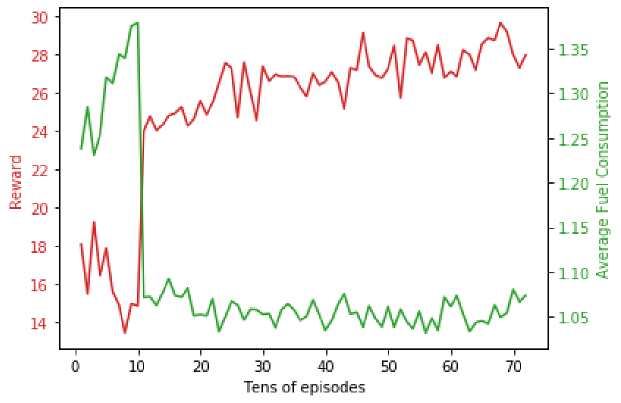


Figure 4.5: The episode reward and average fuel consumption by platoons.

On the other hand, as shown in Fig 4.6, the average delay reveals a growing trend when the reward gets higher. In the simulation, we observe two reasons for this phenomenon. First, we observe that the average delay increases because platoons which can not pass an intersection as a whole reduce their cruise speeds to avoid stopping in front of an intersection. By cruising with a lower speed, the platoons are able to pass an intersection by the next green interval without fully stops. Second, platoons which can not pass an intersection as a whole could be split into two where the first half of the platoons keep the same speed and pass but the second half of them have to fully stop and wait; thus, the second half of them could be a contributing factor to an increasing trend of the delay.

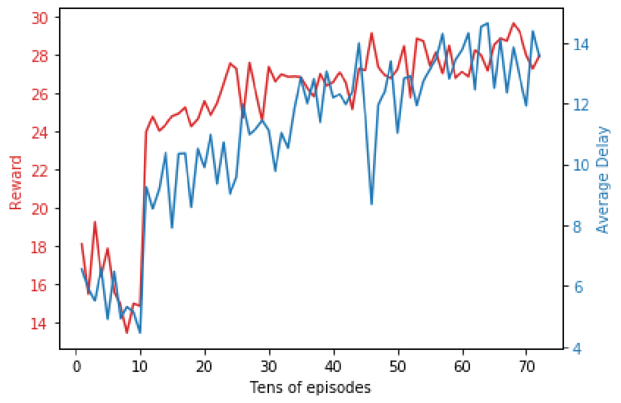


Figure 4.6: The episode reward and average delay by platoons.

4.6 Conclusion and Future Work

With the extremely low latency guaranteed by 5G NR, information sharing among CAVs within a few milliseconds becomes possible, which facilitates research on platooning control to move further. Previous work has demonstrated feasibility of improving latency and fuel usage by optimizing platooning control but tremendous mathematical efforts are required to complete the computation and hence, the computation complexity is high and cannot be applicable to real-world applications. With the help of deep reinforcement learning (DRL), an agent can be trained in advance and learn how to deal with highly changeable traffic flow and select the optimal platooning manipulation to reduce the fuel consumption. In this project, we have demonstrated that the reduction of the fuel usage is feasible by using DRL. The DRL agent learns how to select the optimal action e.g., change the current speed or re-size a platoon) to minimize the fuel consumption. The comparison of learning performance, delay, and fuel consumption shows a promising result if DRL-based platooning control can be applied to CAVs. In our future work, we plan to conduct more complicated scenarios such as considering multi-models traffic and a larger traffic network.

Chapter 5

Deep Reinforcement Learning Based Traffic Signal Control

5.1 Introduction

Recently, great successes in playing Atari games [71] by reinforcement learning (RL) agents have revealed the feasibility of using artificial intelligence (AI) techniques in solving complex problems. In two different strategy games, astonishing milestones indicated higher or at least equivalent to a human-level performance in which machines outperformed human players. For example, AlphaGo [89], which is implemented using a deep reinforcement learning (DRL) algorithm, played an excellent Go game and eventually defeated the most competitive human player, Ke Jie, in 2017. More recently in 2019, AlphaStar [104] defeated players of a real-time strategy game StarCraft II. These developments show that a machine with a self-learning skill is able to perform at the human level after it has been trained by tens of millions of episodes. This has spurred the development of RL-based algorithms for other complex real world applications.

A considerable number of papers using RL have been dedicated to traffic control problems. Thorpe, and Anderson published the earliest paper [96] that applied on-policy learning, SARSA, to the TSC problem. Their simulations were not realistic since complicated traffic situations such as avoiding oncoming traffic flows while making turns, and passing through intersections were not considered; instead, only 4 arrivals at an isolated intersection and $2 \times 10 \times 10$ states are involved.

The study in [3,7] started to model a comparatively complex traffic environment with larger state space and more actions. Abdoos *et al.* [3] applied off-policy learning, Q-learning, to a non-regular traffic network with 50 intersections each with 4 phases. To address the large state space problem, the network was divided into multiple sections each with a subset of intersections. Each partition was governed by a RL agent such that each agent was responsible for only 24 states. Their proposed multi-agent Q-learning method outperformed a fixed-time signal controller.

In a traffic network with multiple intersections, the state space grows exponentially with the number of intersections. The limitation of RL when dealing with a traffic network with complicated flows and numerous intersections becomes evident. A single RL agent is insufficient in such cases because it suffers from curse of dimensionality [25]. In order to solve this issue, an idea of a hierarchical architecture by using a multi-agent RL [4,8,50,78] has been widely applied to the traffic light control problem in multiple intersections. By decomposing a traffic network into several regions, each of which is governed by one agent, they are able to lessen the workload for each agent. Furthermore, each region will consist of many individual intersections for which one agent is responsible for determining traffic light schedule. This evolves into a multi-level control for a traffic network. On the other hand, instead of exactly recording all Q-values for all state-action pairs (s, a), some papers came up with the idea of using an approximator which is able to learn without the tabular RL. The approximation-based RL which utilizes deep neural networks such as convolutional neural network (CNN) has been developed for an isolated intersection [32,34,79,100].

The above studies all addressed the same fundamental problem of RL where a large number of states degrades the learning performance of the RL agent. They can be categorized into two branches, each of which applied a specific approach to solve the problem. In one branch the issue is addressed by using the architecture with a multi-agent RL. In the other branch a state and action, the Q-value can be approximated by using a neural network as an approximator. However, neither of them are perfect solutions for applying to TSC. First of all, although the hierarchical architecture with a multi-agent RL is able to handle a great geographical region, we still need to recursively divide a bigger region into several sub-regions, each of which is assigned to one RL agent. This architecture causes another workload issue, which is sharing of the state information among the multiple RL agents. This requires a reliable communication channel between adjacent intersections and between sub-regions. Otherwise, only local optimality can be guaranteed rather than global optimality. On the other hand, the approximation-based approach is quite promising as it does not have the issue of additional workload and achieving only local optimality. But, it takes a relatively long time to train the model because the action selection will be learned by several neural layers rather than updating values in a table. For example, the deep neural network has to go through many episodes in order to fine-tune the weights in neurons by backpropagation. Then, the agent is able to select the best action with the optimal Q-value. This reveals that an approximation function like CNN or DNN may require a very large number of training episodes to learn and obtain a good result. Moreover, the mathematical guarantees of convergence can no longer be guaranteed if non-linear approximation functions, neural networks are applied. Even worse, sometimes, RL-based algorithms will not converge if the proposed hyperparameters, architecture, and the selection of the initial policy are inappropriate. While many papers have shown good prospects on deep Q-Network (DQN) [72], based on our preliminary experiments, we found that DQN is actually difficult to converge and yield optimal results.

However, there are two aspects, which have not been investigated very well by previous papers, that can be used to improve the performance of deep RL (DRL). First is the impact of applying different temporal-difference (TD) learning algorithms (on-policy and off-policy) with deep learning to a traffic network. Second is the impact of making decisions using different metric rather than a queue length. Previous work has mainly focused on developing Q-learning with neural networks instead of SARSA. According to our preliminary experiments shown in Fig. 5.1, for an isolated intersection, on-policy learning methods, SARSA and SARSA with eligibility traces (SARSA λ), are capable of stabilizing a training process and achieving a better performance than off-policy learning methods based on Q-learning. There are only a few research that has used SARSA dedicated in TSC. Jin and Ma [48–51] implemented a framework utilizing TD methods and a group-based control method which has a better flexibility in green time allocation. They conducted a series of experiments to compare the performance of Q-learning and SARSA. To the best of our knowledge, there has not been any work analyzing the strengths of DQN and deep SARSA (DSARSA) nor research evaluating their learning effects on TSC problems in a large-scale traffic network. Second, all previous research was used to extract information from queue lengths of vehicles. Based on our previous work [117, 122], we have shown that the fairness of using head-of-the-line (HOL) sojourn time is better than that of using queue length. We consider HOL sojourn time as input states for the

DRL-based agent. HOL sojourn times of lanes at every intersection are utilized to formulate traffic flow maps (TFMs). A detail analysis of above mentioned aspects are discussed in this chapter. We expect that this chapter will answer essential questions about DQN and DSARSA and fill the gap in understanding these benefits and limitations. We also believe that our contribution will provide a good reference of applying DRL in ITS for many experienced researchers to review the existing methods and open challenges.

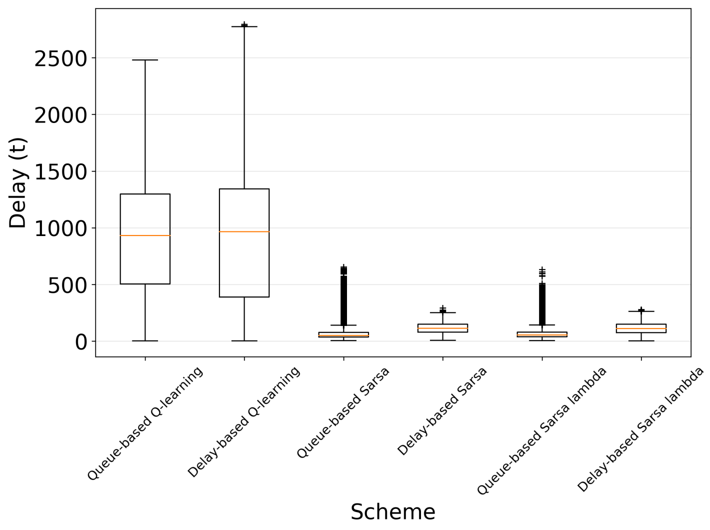


Figure 5.1: Comparison among RL-based methods for an isolated intersection

5.1.1 Organization

The rest of this chapter is organized as follows. Based on our preliminary study, we list our contributions in Section 5.2. In Section 5.3, we introduce our system model and problem for the traffic network. In Section 5.4, we propose the DRL framework for solving TSC problems in a traffic network. In Section 5.5, we discuss the performance of different DRL-based agents, namely, DQN, 3DQN, DSARSA, and 2DSARSA in terms of the training and testing datasets. We also describe baseline algorithms for the comparison. Finally, we conclude and discuss future work in Section 5.6.

5.2 Contribution

- We propose the deep dueling SARSA referred to as 2DSARSA, which combines the deep learning, dueling architecture, and experience replay memory in this chapter. To the best of our knowledge, this is the first framework using 2DSARSA to TSC involving multiple intersections. Preliminary result shows that deep SARSA (DSARSA) outperforms and converges faster than DQN. Furthermore, the dueling architecture and experience replay memory in 2DSARSA can accelerate the convergence rate of the learning for a large action space.

- We propose traffic flow maps (TFMs) for capturing flow dynamics of a traffic network with several intersections. DRL-based agents learn from the proposed TFMs as traffic observations (states) over time. To the best of our knowledge, this is the first work proposing TFMs.
- We provide a thorough analysis on the learning performance of DQN, 3DQN, DSARSA, and the proposed 2DSARSA. To the best of our knowledge, this is the first work discussing DQN and DSARSA in detail in the context of TSC.
- We compare the performance of DQN, 3DQN, DSARSA, and the proposed 2DSARSA. At present, many existing RL techniques such as DQN [72], double deep dueling Q-Network (3DQN) [58], and DSARSA [128] have been developed and tested. In order to prove the performance of 2DSARSA, different agents based on above DRL techniques are implemented and trained separately for comparisons.

5.3 System Model and Problem Statement

We begin our study with a 3 by 3 grid network shown in Fig. 5.2. There are 9 intersections, each of which activates 4 traffic lanes (1, 3, 5, 7) in the traffic system. For each time slot t, every intersection has to determine which phase should be activated. As shown in Fig. 5.3, there are two phases which represent two directions: vertical and horizontal. Vehicles passing through the current intersection i will enter in the next intersection j. The conjunction of two adjacent intersections i and j will cause a traffic bottleneck if the traffic flows from the upstream or the traffic flow to the downstream are not taken into account. The proposed 2DSARSA is designed for solving this problem and achieving optimal throughput and/or end-to-end delay performance for the traffic network.

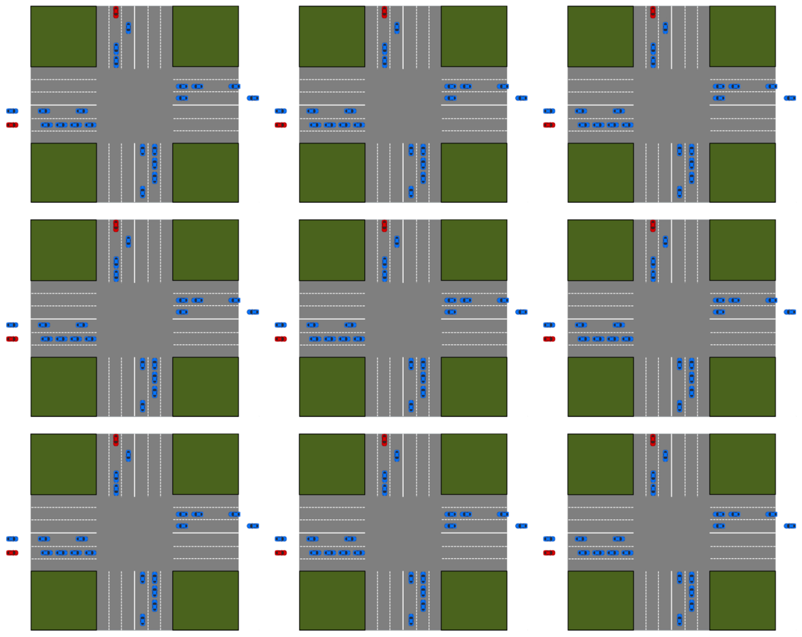
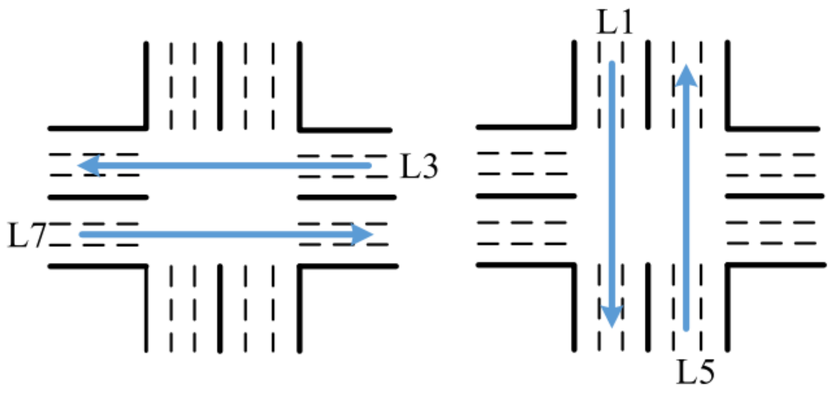


Figure 5.2: A 3 by 3 grid traffic network topology consisting of 9 intersections



(a) P0 with lanes 3 and 7 (b) P1 with lanes 1 and 5

Figure 5.3: Each intersection has two phases, P0 for horizontal flow and P1 for vertical flow.

5.4 The Proposed Learning-based Framework for Multiple Intersections

We propose a DRL framework which enables the incorporation of various RL-based algorithms and exploitation of global information for learning how to interact with a traffic network. Given a specific RL-based algorithm and TFMs of the entire city over time, our framework adopts the given RL-based algorithm with deep learning and takes TFMs as input states to train a DRL-based agent at higher level to determine the best policy that reduces traffic latency (average end-to-end delay) and increase overall throughput. The DRL-based agent of centralized controller with the global information (TFMs) is responsible for managing incoming and outgoing traffic flows, operating at a fine-grained time scale, and learning an appropriate traffic phase for each intersection at each time slot; meanwhile coordinating an entire traffic network consisted of multiple intersections. Local intersections follow phases decided by the DRL-based agent, which makes decisions accordingly to the globally determined performance objectives.

The global information composed of time series TFMs (states) over time must include all essential traffic information which assists the DRL-based agent in learning from the traffic environment. The proposed TFMs are synthesized to represent the traffic environment over time slots. They are formulated by using Poisson arrival data from multiple intersections. The underlying algorithm of the DRL framework is a RL-based algorithm, namely, on-policy or off-policy learning with an approximator implemented by the dueling architecture, CNN, and memory replay buffer. Based on our preliminary experiments, SARSA has revealed a more stable learning curve than Q-learning. The dueling architecture helps the selection of action by separating the estimations of state-value and action advantage. It will benefit the DRL-based agent if the action space is large. The experience memory replay breaks the strong temporal correlations and speeds up the learning process. Hence, these strategies are applied to solve the huge state space and convergence problems. First, a state is fed into the CNN at every time slot. Then, given the state, the network parameter set θ are trained and updated by optimizing the loss function given by Eq. 5.3, and the optimal action is generated according to the learned policy. The proposed DRL framework, which extracts features from TFMs by CNN, could eliminate the workload of exchanging state information in a multi-agent architecture and achieve maximal performance of the agent.

5.4.1 Deep Reinforcement Learning Model

In any DRL model, we need to carefully define the state, the action and the reward function.

5.4.1.1 State Space

The state contains well-rounded information of different intersections from which the agent can learn the traffic flows in a network. Recently, research has shown that RL agents can be well-trained by images [62, 71]. However, most research papers [32, 34, 58, 100, 115] formulate a matrix for an intersection by marking 0 and 1 based on coordinates of vehicles. To the best of our knowledge, we are the first work which proposes TFMs as the global information including information of all traffic flows of the network. State space is based on TFMs. Note that traffic flows refer to HOL sojourn times in this chapter. However, HOL sojourn time is not sufficient for capturing flow dynamics of a traffic network with several intersections, and the delay-based solutions cannot guarantee the linear relation. A new delay metric between links needs to be redesigned such that the linear relation between queue lengths and delays for multi-intersection traffic can be established. Ji *et al.* [47] proposed the idea of utilizing delay differences for wireless networks where $W_i(t)$ denotes the HOL sojourn time of lane i at time slot t in the traffic network. The delay metric $\hat{W}$ and delay difference $\Delta\hat{W}$ are defined as follows:

$$\hat{W_j}(t) = W_j(t) - W_i(t) \tag{5.1}$$

$$\Delta\hat{W_{ij}}(t) = \hat{W_i}(t) - \hat{W_j}(t) \tag{5.2}$$

The queue at an intersection j is roughly the number of vehicles arriving at the traffic network during the time slots between $[U_j(t), U_j(t) + \hat{W_j}(t)]$. When $\hat{W_j}(t)$ is large enough, the queue length of intersection j is on the order of $\lambda \times \hat{W_j}(t)$. Hence, a large delay metric implies a large queue length at j, and a large delay difference implies a large queue difference. HOL differences between any two adjacent intersections can be calculated by Eq. 5.1 and 5.2. The underlying concept is to capture queuing dynamics of a traffic network by using HOL differences. Here, we use the delay metric to substitute the queue metric because delay-based method can achieve better fairness and will not suffer from the last vehicle problem [47, 117].

The traffic flow information of one intersection is stored in a 5×5 matrix as shown in Fig. 5.4. The blue element indicates the average traffic flow of all lanes in one intersection. Orange elements denote traffic flows in different lanes where the figure in each element indicates the number of the traffic lane rather than an actual HOL value. Red and brown elements represent differences of traffic flows from neighboring intersections. The TFMs are formulated by information of traffic flows of all intersections, each of which contains HOL sojourn times corresponding to different lanes, and HOL differences corresponding to different adjacent intersections at each time slot t. The agent learns from these maps and determines the best action according to flow dynamics of a traffic network. All the information that the agent needs to know is encoded in the map shown in Fig. 5.5

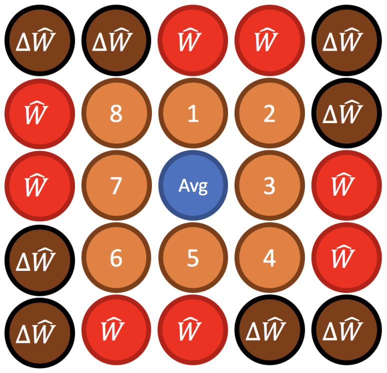


Figure 5.4: An intersection contains traffic flow information where blue element represents the average traffic flow of all lanes at the intersection, orange elements represents traffic flows in different lanes (figures in elements only indicate different traffic lanes rather than actual HOL values), and red and brown elements represent differences of traffic flows from neighboring intersections.

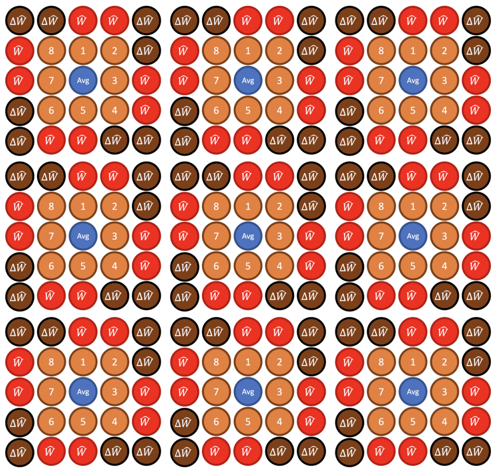
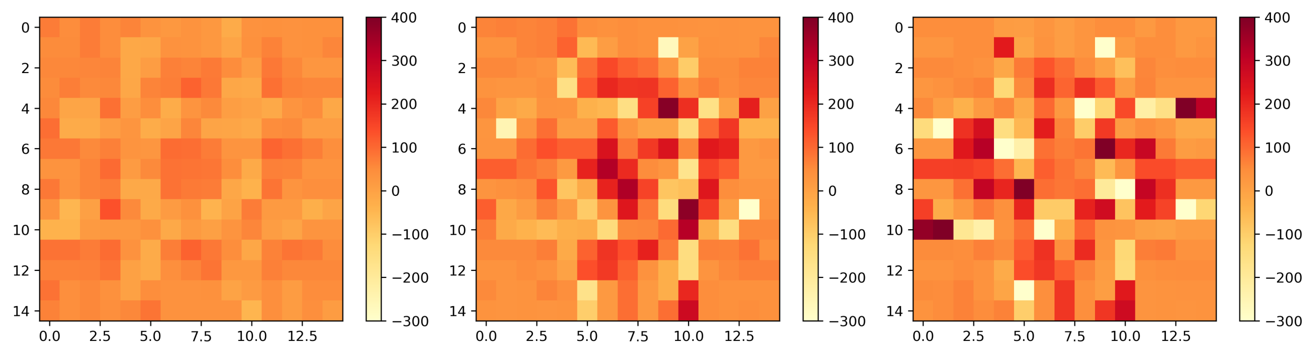


Figure 5.5: An traffic flow map of one traffic network composed of nine intersections

As shown in Fig. 5.6, there are TFMs for the 9 intersections at time slot 220, 5020, and 20020, respectively. These TFMs visualize information of delay-based traffic flows of a traffic network as it evolves over time. The color encodes flow dynamics by HOL sojourn time. As can be observed, they certainly capture flow dynamics of a traffic network. The intensity shows that traffic congestion has become worse over time.



(a) $t = 220$ **(b)** $t = 5020$ **(c)** $t = 20020$

Figure 5.6: TFM for the 9 intersections as it evolves over time

5.4.1.2 Action Space

Each intersection can activate a traffic flow from either vertical or horizontal direction. The action sequence of the traffic network is $a = \{0,1\}^M$ where M is the number of intersections and $0, 1$ denotes which direction should be activated, 0 for the horizontal direction, 1 for the vertical direction. Hence, the action vector can be encoded in M bits and the total action space has a cardinality of 2^M.

5.4.1.3 Reward Function

The throughput and average end-to-end delay of the network are two important factors which indicate the level of traffic congestion in transportation. Higher throughput denotes more vehicles in the network have been served. A lower average end-to-end delay means vehicles in the network do not suffer from waiting. We want to improve the overall throughput of the traffic network while decreasing the average end-to-end delay of vehicles in the network. We consider a power metric $P = \frac{\lambda}{d}$ as our reward function where λ denotes the throughput of the network, d represents the average end-to-end delay of vehicles. The goal of DRL-based agents is to maximize P which is achieved by maximizing λ and minimizing d.

5.4.1.4 Convolutional Neural Network

Research work has shown that RL agents can learn well from images by CNN [62, 71] in recent years. We implement CNN for the proposed framework by specifying the number of layers and size of filters which meets our requirement of training agents for TSC. The agents based on different RL-based

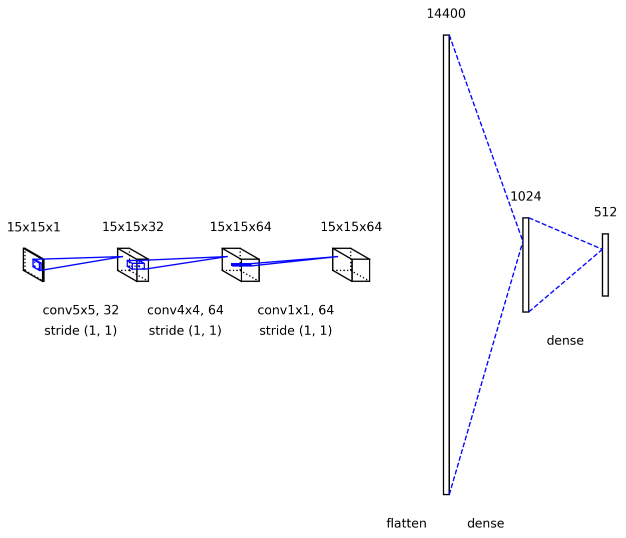


Figure 5.7: The convolutional neural network as the function approximator

algorithms share the same CNN. The deep neural network consists of three convolutional layers and one fully-connected layer as the output in Fig. 5.7. There are 32 5 by 5 filters with 1 strides in the first convolutional layer. The output is a volume with the size of $15 \times 15 \times 32$ which is fed into the second convolutional layer. At the second layer, the number of filters is 64, each of which has the size of 4 by 4 and moves 1 stride each time. After the second layer, the output is a volume with the size of $15 \times 15 \times 64$. The third layer contains 64 filters with the size of 3×3, each of which moves 1 stride every time. This outputs a $15 \times 15 \times 64$ tensor. The final output, a fully-connected layer, transfers the tensor into a 512×1 matrix.

5.4.1.5 The Proposed Deep Dueling SARSA Learning Method

TD learning methods combine nice properties from the Monte Carlo (MC) and dynamic programming (DP). First, TD methods learn state-action function $Q(s, a)$ directly from experience with TD errors and the bootstrapping. TD error denotes the difference between the current estimation and the actual observation of the state-action value. The bootstrapping substitutes the remaining trajectory with one or more estimated rewards in the update step. TD methods constantly track the current TD error and propagate this TD error back in order to update the approximation for every time step. Second, TD methods are capable of learning from incomplete episodes without knowing the dynamic model of the environment. Third, TD methods are suitable for traffic control problems because they perform well in continuous environment.

In contrast, Monte Carlo (MC) methods must wait for the final reward and update the state-action function by using a complete series of rewards, $Q(s_t, a_t) = Q(s_t, a_t) + \alpha \times [G_t - Q(s_t, a_t)]$ where $G_t = R_{t+1} + \gamma \times R_{t+2} + \cdots + \gamma^{T-1} \times R_T$ indicates the total discounted reward at time t. Unlike MC methods, TD methods do not necessarily need to wait until the end of episode. Instead, they update Q-values based on estimated Q-values that have already been learned as given by Eq. 2.9.

SARSA which stands for the current state S, current action A, immediate reward R, next state S' and next action A' is one of TD learning methods. SARSA is also known as an on-policy method which evaluates or improves the behavioural policy, e.g., SARSA fits the action-value function to the current policy, i.e., SARSA evaluates the policy based on samples from the same policy, then refines the policy greedily with respect to action values. On the other hand, off-policy methods, an agent learns an optimal value function/policy, maybe following an unrelated behavioural policy; for example, Q-learning attempts to find action values for the optimal policy directly, not necessarily fitting to the policy generating the data, i.e., the policy which Q-learning obtains is usually different from the policy that generates the samples. In short, on-policy can be understood as learning from same-policy. However, off-policy learns from different-policy.

Deep SARSA refers to the on-policy TD learning algorithm using CNN (mentioned in Section 5.4.1.4) as a function approximator. Similar to [72], a parameterized neural network replaces the conventional tabular approach for updating state-action pairs. DSARSA is incorporated into our proposed DRL framework. The input data of CNN are TFMs and the output is the action with the optimal Q-value over all state-action pairs. θ is a network parameter set of CNN. During the training process, the loss function is defined as follows:

$$L(\theta_t) = (y_t^{DSARSA} - Q^{\pi}(s_t, a_t; \theta_t))^2 \tag{5.3}$$

$$y_t^{DSARSA} = r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1}; \theta_t^{-}) \tag{5.4}$$

where s_t is the current state, a_t is the current action, s_{t+1} is the next state which is dependent on a_t

taken by the agent in s_t, a_{t+1} is the next action selected by ϵ-greedy, and y_t^{DSARSA} is the TD target given by Eq. 5.4. The goal of CNN is to update the network parameter set θ that optimizes the loss function $L(\theta_t)$. First, the partial derivative of $L(\theta_t)$ is calculated by differentiating Eq. 5.3. The gradient of the loss function can be obtained as follows:

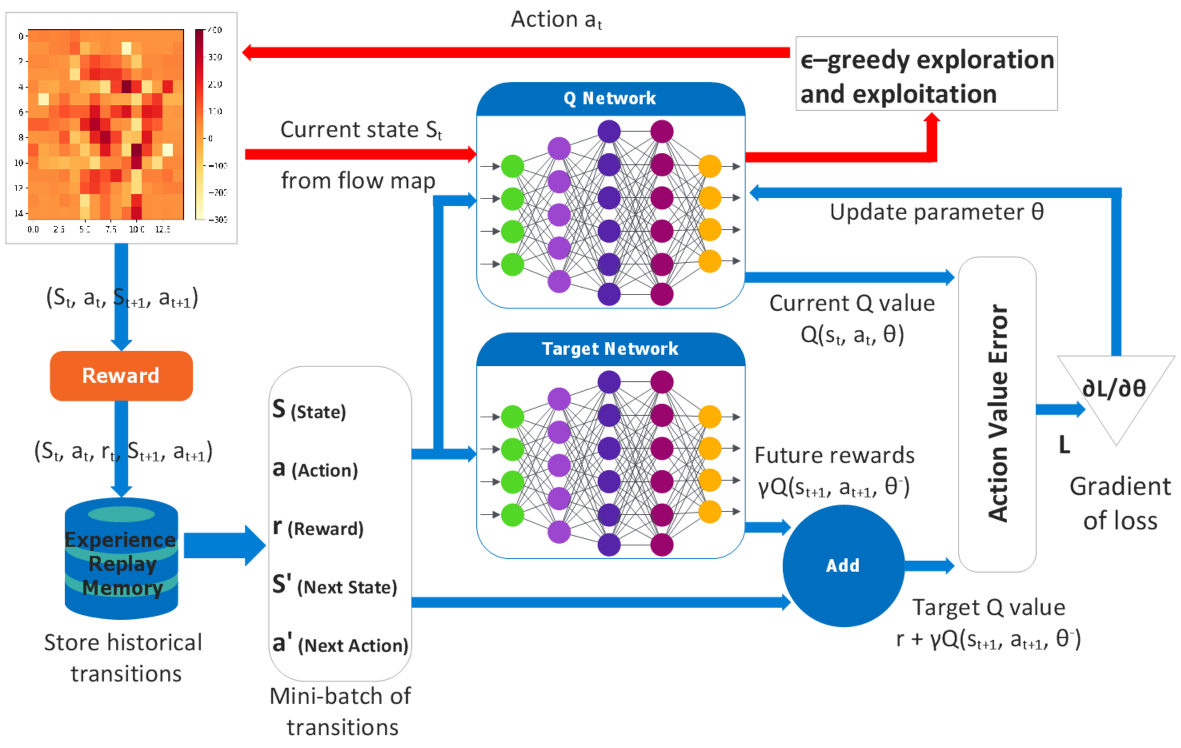
$$\bigtriangledown_{\theta_t} L(\theta_t) = (y_t^{DSARSA} - Q^{\pi}(s_t, a_t; \theta_t)) \bigtriangledown_{\theta_t} Q^{\pi}(s_t, a_t; \theta_t) \tag{5.5}$$

where $\bigtriangledown_{\theta_t} Q^{\pi}(s_t, a_t; \theta_t)$ denotes the gradient of the current state-action value. We optimize the loss function $L(\theta_t)$ by Eq. 5.5 and the network parameter set θ is updated by using stochastic gradient descent (SGD).

The action space considered in this chapter has a cardinality of 2^M since the TSC problem we try to solve is a traffic network involving multiple intersections. The size of our action space would degrade the learning performance of other DRL-agents. As mentioned in Section 2.5.3, the dueling network architecture performs well in a large action space. The benefit of the dueling network architecture is that the estimations of state-value and action advantage are separated for learning which state-values are higher without going through and learn every single state-action pair. We propose the deep dueling SARSA learning using the dueling network architecture referred to as 2DSARSA. In the proposed method, state-value and the advantages for each action are separately estimated by the dueling network. The output of the dueling network is given by Eq. 2.16 in order to combine state-value and advantages.

5.4.2 Overall Architecture of The Proposed DRL Framework

Fig. 5.8 shows the overall architecture of the proposed DRL framework for multi-intersection control. As mentioned, Given a RL-based algorithm, TFMs as global information of the traffic environment is fed to the CNN over time slots. The ϵ-greedy approach followed the function in Table 5.2 is utilized for the exploration and exploitation. The agent is trained depends on the given RL-based algorithm. It learns how to make decisions based on the states and memory replay. Then, the current state, current action, observed reward, next state, and next action are pushed into the memory for reusing them later. There are two separated neural networks, one is a Q-network and the other is a target network, they share exactly the same feature and architecture. The update parameter θ will be



Action a_t
ϵ–greedy exploration and exploitation
Q Network
Current state S_t from flow map
Update parameter θ
Current Q value $Q(s_t, a_t, θ)$
Action Value Error
∂L/∂θ
L
Gradient of loss
$(S_t, a_t, S_{t+1}, a_{t+1})$
Reward
$(S_t, a_t, r_t, S_{t+1}, a_{t+1})$
Experience Replay Memory
Store historical transitions
S (State)
a (Action)
r (Reward)
S' (Next State)
a' (Next Action)
Mini-batch of transitions
Target Network
Future rewards $γQ(s_{t+1}, a_{t+1}, θ^-)$
Add
Target Q value $r + γQ(s_{t+1}, a_{t+1}, θ^-)$

Figure 5.8: The overall architecture of the proposed 2DSARSA

copied over to the target network every τ steps. The current state and action are first fed into the Q-network for approximating a Q-value. Old experiences extracted from the replay memory buffer are used to be inputs to the target network for calculating a target Q-value. The difference, L, between the Q-value and target Q-value will be partially differentiated to get the derivative in which the agent will have a sense and know how to update the Q-network by fine-tuning the parameter θ.

In the proposed DRL framework, various RL-based algorithms mentioned in Section 2.4.2.1 and 5.4.1.5 share this overall architecture. We implement 4 different DRL-based agents, namely, DQN, 3DQN, DSARSA, and 2DSARSA, which derive from the overall architecture with respect to the given RL-based algorithm, neural network, and network architecture as shown in Table 5.1.

5.5 Experimental Results

During the training process, we use 1-hour traffic arrival data as one episode generated by following the Poisson process. All agent run 1000 episodes training data and they pre-train within the first 100

Table 5.1: Properties of Different DRL-based Agents

Methods	TD Learning	Neural Network	Network Architecture	Work
DQN	Q-learning (off-policy)	CNN	Conventional	Mnih *et al.* [72]
3DQN	Q-learning (off-policy)	CNN	Dueling	Liang *et al.* [58]
DSARSA	SARSA (on-policy)	CNN	Conventional	Zhao *et al.* [128]
2DSARSA	SARSA (on-policy)	CNN	Dueling	The Proposed Method

episodes for accumulating replay data. For test process, 10-hour long traffic arrival data which follow the Poisson arrivals as well are used as the test data to verify the robustness of each agent after completing the training step. The difficulty is the training process because all DRL-based agents take a very long time and might not converge at the end. In the experiments, each agent needs to be trained separately; hence, it takes nearly one week to do the training process. The reward, average end-to-end delay (sojourn time), and average queue length are collected as indicators for evaluating the learning performance of each agent after running test data. We want to analyze and discuss comprehensively about their performance through those collected data.

5.5.1 Baseline Scheduling Algorithm

Based on the work [93] for the throughput optimality of a queuing network with random arrival rates, Ji *et al.* [47] further considered a queue difference between any two adjacent nodes at each time slot and then determined a feasible schedule by activating the phase with the highest pressure (maximum queue length). In their algorithm, the controller at each intersection independently determines which phase of the traffic movement is scheduled based on the queue length and differences from neighboring nodes in every time slot. Their results showed that their algorithm can achieve maximum network throughput and global optimality without any prior knowledge about arrival rates.

$$\vec{p}^{*}(t) \in \underset{\vec{p}\in\mathcal{S}_{\mathcal{P}}}{\operatorname{argmax}} \sum_{P_{i,j}=1} \gamma_{i,j} \cdot Q_{i,j}(t) \cdot \mu_{i,j}(\vec{p}) \tag{5.6}$$

Later, this concept was first applied to traffic signal control in [116]. However, the selection of phases does not necessarily depend on only the queue length. The metrics such as the HOL sojourn time, sum-of-delay, and hybrid which combines the queue and HOL delay can be used as the measurement. Based on their work, we applied the delay-based BP scheme using the HOL sojourn time to the network as follows:

$$\vec{p}^*(t) \in \underset{\vec{p}\in\mathcal{S}_{\mathcal{P}}}{\operatorname{argmax}} \sum_{P_{i,j}=1} \gamma_{i,j} \cdot W_{i,j}(t) \cdot \mu_{i,j}(\vec{p}) \tag{5.7}$$

This formula calculates the sum of HOL vehicle sojourn time multiplied by two weighted values $\gamma_{i,j}$ and $\mu_{i,j}(\vec{p})$. As before, for vehicles from lane i to lane j, $(i, j) \in \mathcal{E}$, $\gamma_{i,j}$ is a positive constant that can put more emphasis on certain movements and $\mu_{i,j}(\vec{p})$ denotes a traffic flow of vehicles that can be transferred through the connected edge when phase $\vec{p}$ is activated. The scheme will return a phase that maximizes the total pressure released.

The proposed DRL framework is applied to the TSC problem in order to learn from TFMs of the entire traffic network and then, determine which phases are supposed to be preferentially activated for the entire network with 9 intersections. In the experiment section, the performances of BP-based scheduling schemes as baselines are compared to our proposed DRL framework.

5.5.2 Hyperparameters

Before discussing the experimental results, we want to elaborate more on hyperparameters. The hyperparameters used in the proposed method are listed in Table 5.2. Traffic arrival ratio for each intersection is $\vec{\lambda} = [0.2, 0.5, 0.2, 1] \times 0.125 \times 1.5$ v/s/l (vehicles per second per lane) for generating 1-hour and 10-hour datasets for training and testing, respectively. The ϵ-greedy function for determining whether exploring or exploiting is applied to the proposed framework where ϵ_i as the initializing value is 1.0, $\epsilon_f = 0.01$ is the finalizing value, and the decay ratio is 300. Learning ratio α and discount γ are 0.0001 and 0.99, respectively. The experience replay is applied as well where the total size of replay memory $M = 30000$, and mini-batch size $B = 1024$. The number of training episode $E = 1000$ is selected by trial and error. Note that DRL-based agents depending on different RL-based algorithms share the same hyperparameters.

5.5.3 Learning Performance of Different DRL-based Agents

In Fig. 5.9, 5.12, 5.15, and 5.18, we trained different agents based on DQN, 3DQN, DSARSA, and 2DSARSA, respectively, each of which takes few days to finish 1000 training episodes. Their performance will be very unstable if their training is incomplete. For example, training process is

Table 5.2: Summary of hyperparameters in the proposed DRL framework

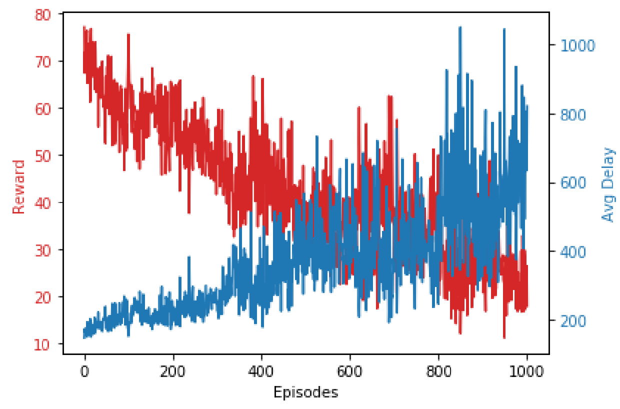
Parameter	Description
Traffic arrival ratio:	$\vec{\lambda} = [0.2, 0.5, 0.2, 1] \times 0.125 \times 1.5$ v/s/l
ϵ-greedy decay function:	calculate the probability of exploration and exploitation where
$\epsilon_f + (\epsilon_i - \epsilon_f) \times e^{-\frac{episode}{decayratio}}$	decay ratio is 300
Initializing $\epsilon_i = 1.0$	the beginning value of ϵ
Finalizing $\epsilon_f = 0.01$	the final value of ϵ
Learning ratio $\alpha = 0.0001$	learning ratio for updating the neural network
Discount ratio $\gamma = 0.99$	discount for future rewards
Memory size $M = 30000$	the size of entire replay memory buffer
Mini-batch size $B = 1024$	the size of samples that will be reused
Maximum number of episodes $E = 1000$	the total number of training episodes

terminated too early, i.e., only 400 or 500 training episodes.

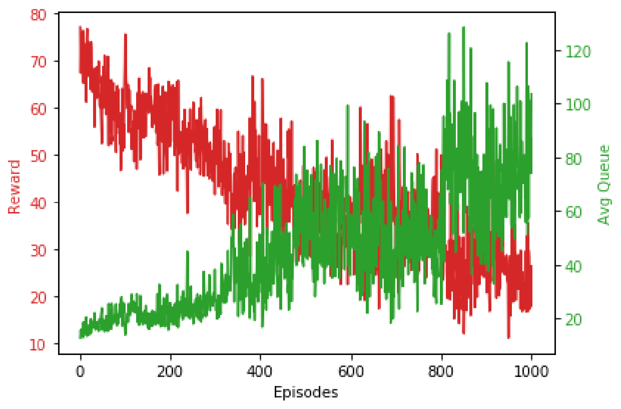
5.5.3.1 Learning Performance of DQN

DQN was proposed by [72] for Atari games which are strong temporal correlated environments. But for traffic networks which are not as simple as Atari games, the effect of the current decision could influence an entire environment in few steps or in even further future. Since the accumulated return caused by a specific decision is not clear, jumping among different policies is not a wise strategy. As shown in Fig. 5.9a and 5.9b, the trend of the reward reflects how well the DQN agent performs over time. Obviously, the agent is not able to increase the reward over episodes. The trend of the reward goes down, which indicates that the agent cannot learn well from the environment. The trends of the average end-to-end delay and average queue length keep increasing, which reflects a very congested traffic caused by poor decisions. Every single decision made by DQN does not necessarily follow the same policy; hence, it does not consider the long-term effect of the current action. Not taking *future* into account will be detrimental to a schedule of traffic movements, which leads to serious congestion and a long delay. Overall fairness of the network in Fig. 5.9d also unravels that vehicles scheduled by the DQN agent suffer from unfair decisions and a longer end-to-end travel time.

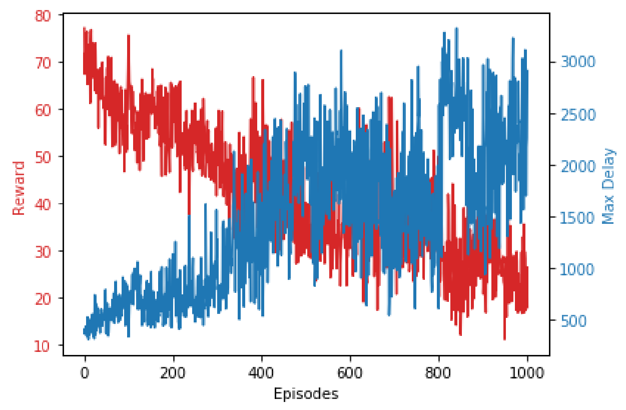
Based on analytics of learning performance we obtain, the agent cannot effectively learn from the environment by DQN. Once the training process has been completed. The DQN agent performs on 10-hour traffic arrival data to exhibit its learning outcomes. Fig. 5.10 shows three delay distributions conducted by QBPC, DBPC, and DQN, each of which represents sojourn times with certain probabilities with which vehicles could encounter. The delay distribution of DQN shows that vehicles would encounter a longer end-to-end travel time with a higher probability compared to two BP schemes. In 5.11, due to an unfair schedule by the DQN agent, a considerable amount of vehicles faces extremely long delay.



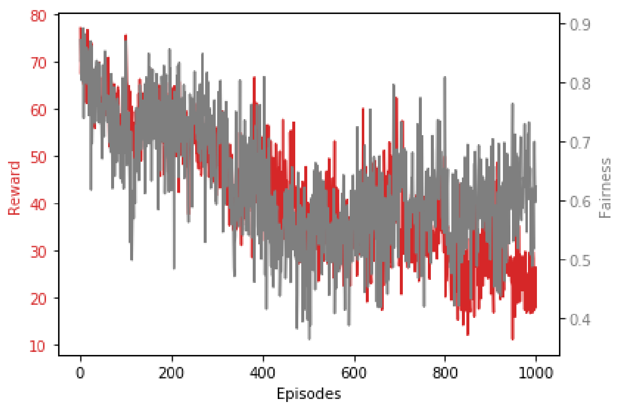
(a) Average delay and reward



(b) Average queue length and reward



(c) Maximum delay and reward



(d) Fairness and reward

Figure 5.9: The trends of delay, queue length, fairness, and reward when training a DQN agent

5.5.3.2 Learning Performance of 3DQN

Liang *et al.* [58] proposed a framework integrating the double deep Q-network with the dueling architecture (3DQN) and memory replay. They applied 3DQN to traffic light control at a single intersection which is comparatively easy. According to their experiments, their framework is able to converge and outperform conventional fixed-time approaches. However, in our experiments of applying 3DQN to the TSC problem in a multi-intersection environment, as shown in Fig. 5.12a and 5.12b, the trend of the reward begins to drop and keep oscillating after 200 episodes, and the average

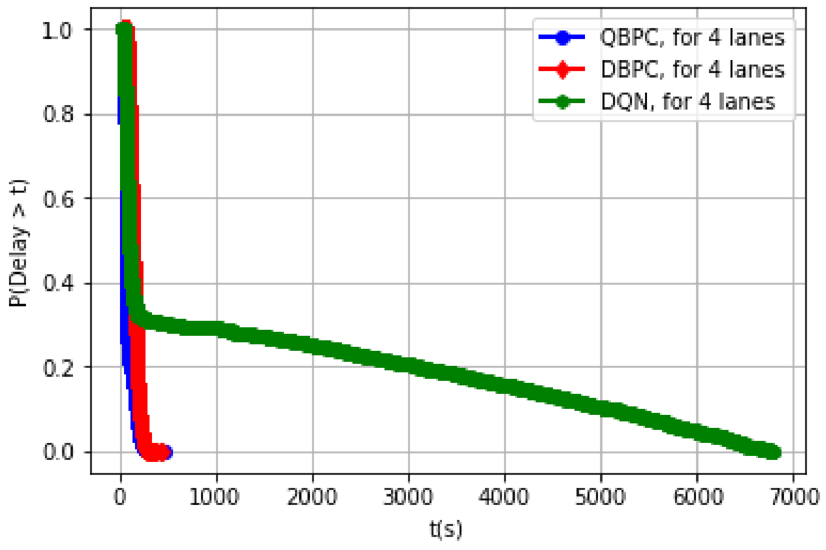


Figure 5.10: Comparison of delay distributions for queue-based BP control (QBPC), delay-based BP control (DBPC), and DQN

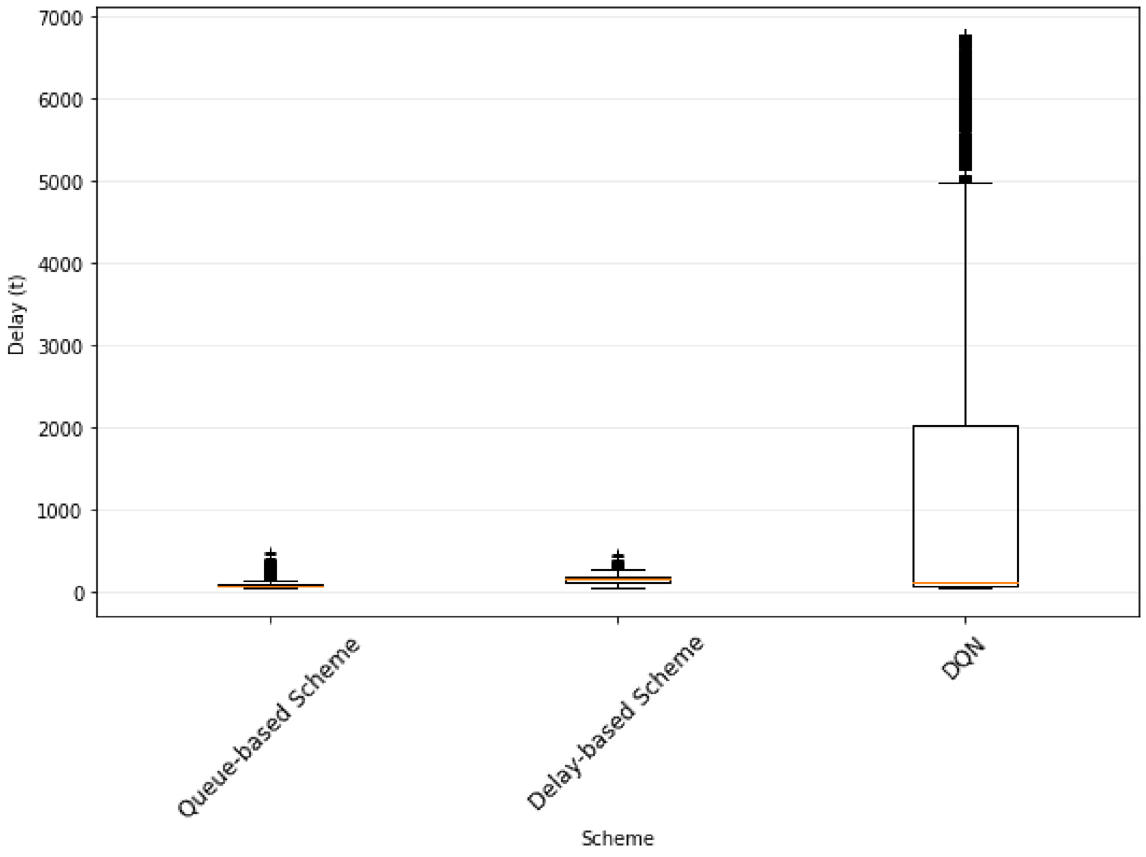


Figure 5.11: Comparison of box plots for queue-based BP control (QBPC), delay-based BP control (DBPC), and DQN

end-to-end delay as well as average queue length of all vehicles increase correspondingly. Similar to DQN, Vehicles scheduled by the 3DQN agent suffer from an unfair scheduling as well in Fig. 5.12d.

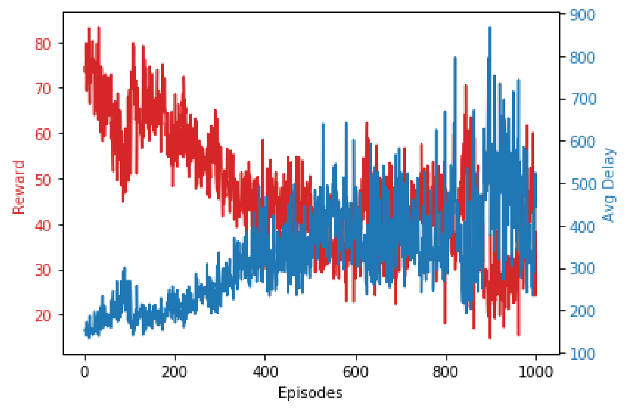
The learning becomes even worse when undergoing more and more training episodes. Based on our experiments, the 3DQN agent ends up performing more poor than we expected. The final results of performing on 10-hour traffic arrival data are shown in Fig. 5.13 5.14. Even if the dueling architecture, which improves the selection of action, has been applied, the 3DQN agent still suffers from high dimensional states and a large action space. Vehicles suffer from a longer end-to-end travel time compared to the DQN agent.

We conclude that DQN and 3DQN both adopting off-policy learning are not suitable for multi-intersection control because the environment considered in this chapter is a traffic network which is far more sophisticated than one intersection. When it comes to a network, the learning performance of using DQN or 3DQN becomes unstable and difficult to converge even though they can deal with one intersection nicely. Both of their trends (reward, average end-to-end delay, and average queue length) gradually become very oscillating when getting close to the end of the training.

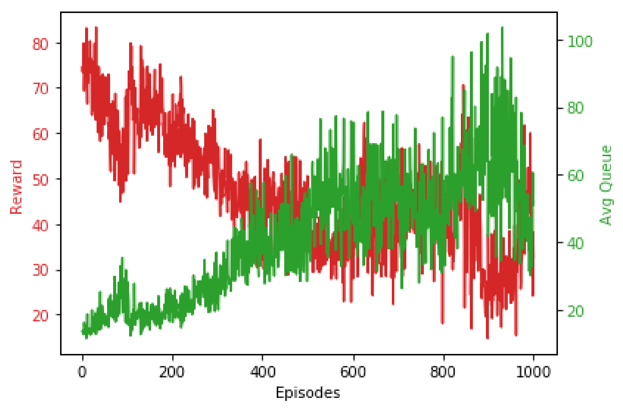
5.5.3.3 Learning Performance of DSARSA

Before discussing the proposed 2DSARSA, we want to first examine the learning performance of DSARSA which uses SARSA with CNN. The difference between SARSA and Q-learning was mentioned in Section 2.4.2.1 and 5.4.1.5. In short, SARSA follows the same policy π and explores optimal state-action values along with this policy. In a highly temporal correlated environment, Q-learning, which selects the optimal action by switching among several policies, would mess up the training process. Switching among multiple policies creates chaos so that the agent will not learn well through Q-learning.

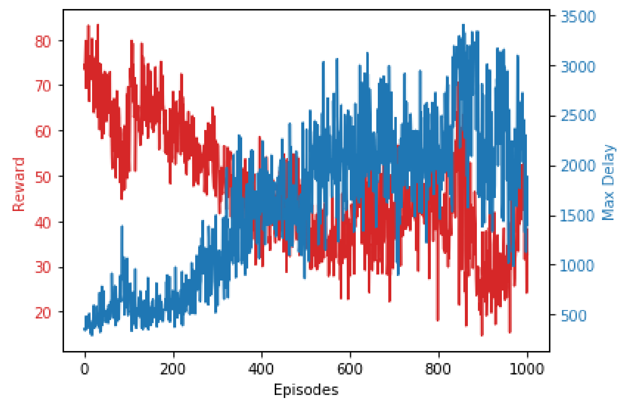
During the training process, the learning performance of the DSARSA agent in terms of reward, delay, queue length, and fairness is shown in Fig. 5.15a, 5.15b, and 5.15d. Before elaborating more on these figures, we want to firstly explain why there are spikes around the 100-th training episode in them. Note that we choose to pre-train within the very first 100 episodes to accumulate tuples of previous states, previous actions, rewards, the current state, and the current action (a.k.a experiences) for the memory replay buffer. Without enough data in the buffer, the number of experiences which can be sampled is insufficient for the agent to compute a loss between a target Q-value and the



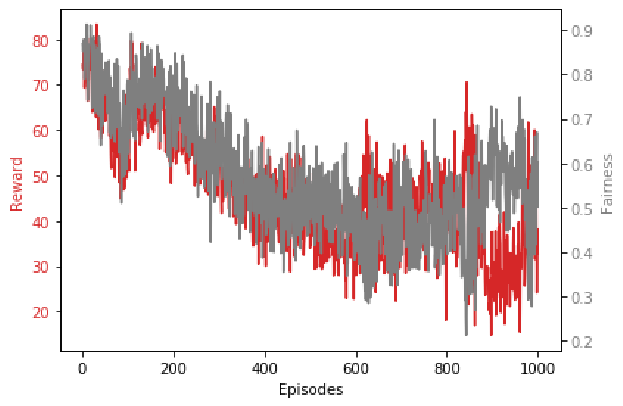
(a) Average delay and reward



(b) Average queue length and reward



(c) Maximum delay and reward



(d) Fairness and reward

Figure 5.12: The trends of delay, queue length, fairness, and reward when training a 3DQN agent

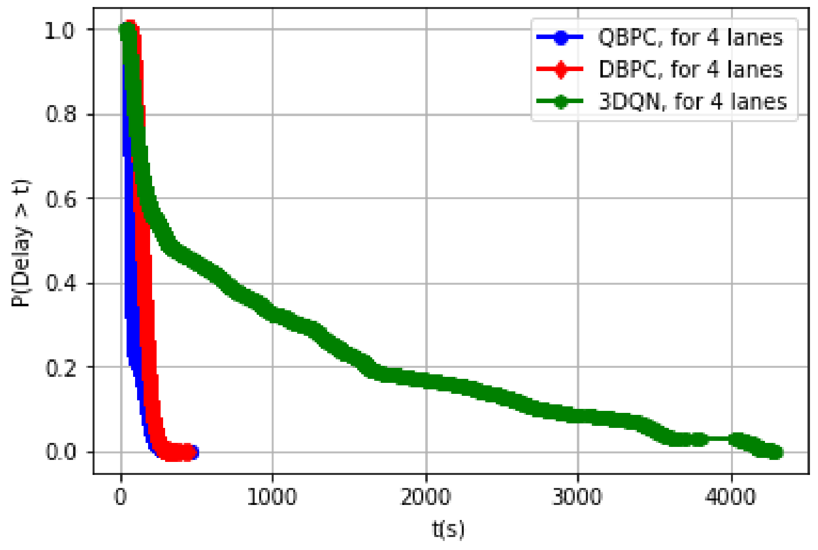


Figure 5.13: Comparison of delay distributions for queue-based BP control (QBPC), delay-based BP control (DBPC), and 3DQN

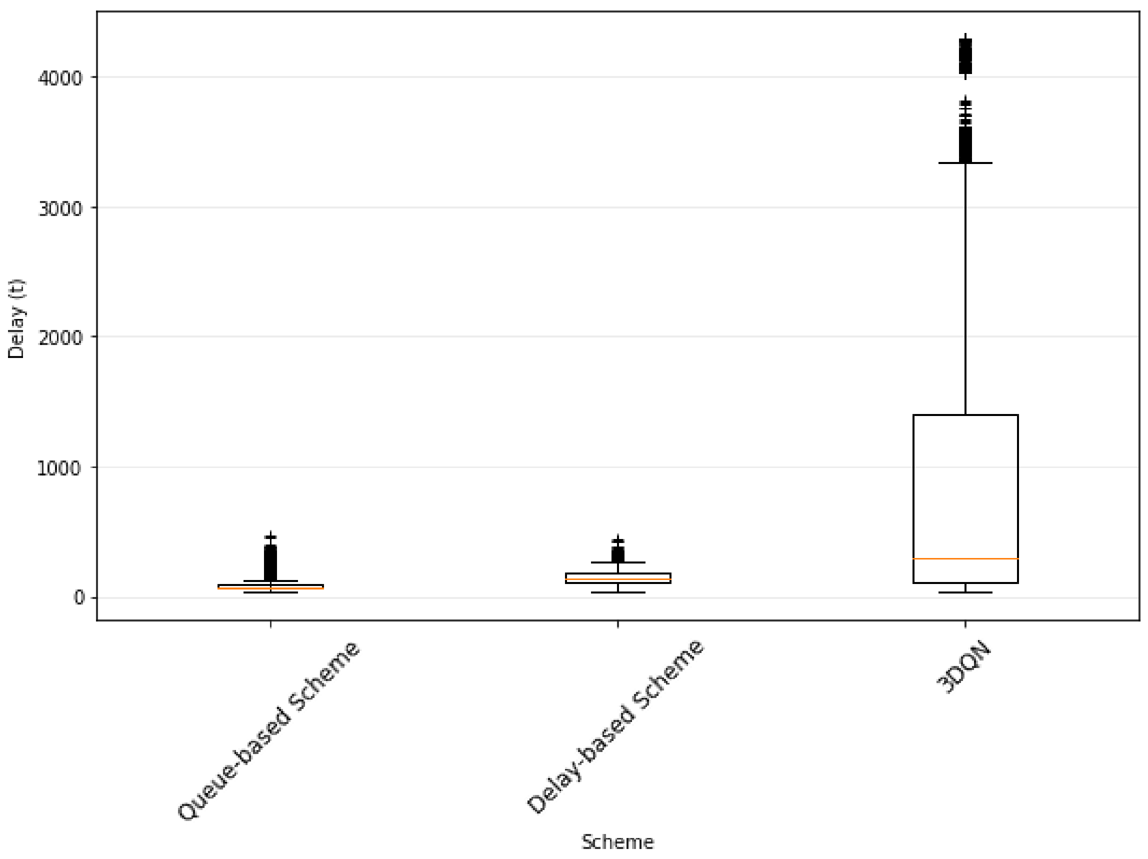
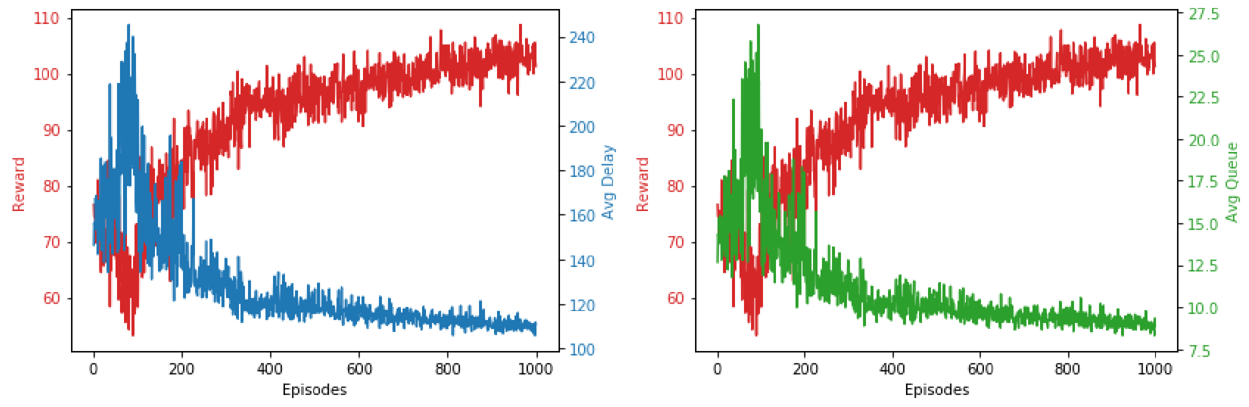


Figure 5.14: Comparison of box plots for queue-based BP control (QBPC), delay-based BP control (DBPC), and 3DQN

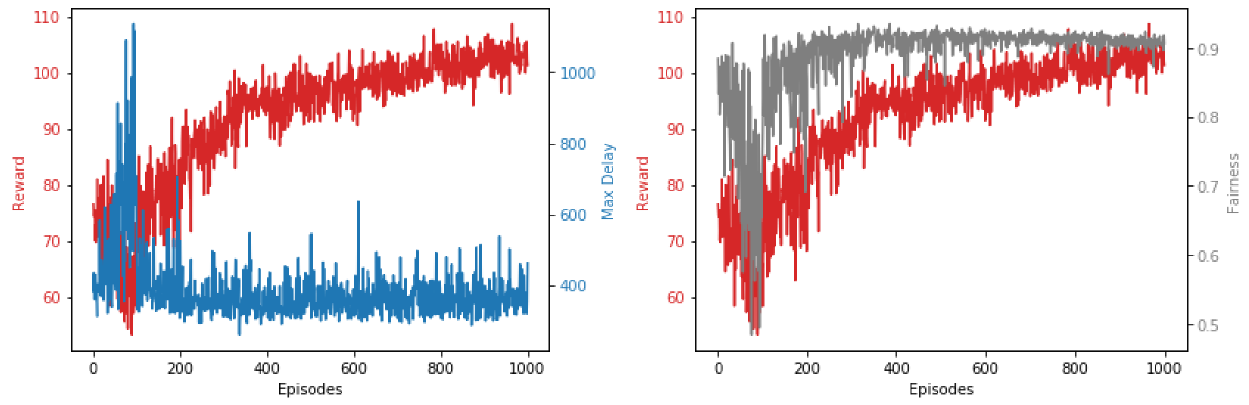
current Q-value, and update the neural network. From the 1-st to the 100-th episode, the DSARSA agent randomly selects actions for exploration while accumulating replay data.

As can be observed, the agent obtains higher rewards after learning from more and more episodes. The average end-to-end delay of vehicles and average queue length of intersections in the network decline quickly after pre-training. The fairness of the network keeps increasing as the reward becomes higher. Overall, the trends of the reward, average end-to-end delay, average queue length, and fairness are converged eventually. Their convergences reveal that the agent with the on-policy learning algorithm is indeed capable of learning from high dimensional states and performing well in a huge action space. Unlike DQN and 3DQN, DSARSA outperforms two BP-based schemes once the training has been done in Fig. 5.16 and 5.17. Fig. 5.16 shows three delay distributions conducted by QBPC, DBPC, and DSARSA, each of which represents sojourn times with certain probabilities with which vehicles could encounter. Fig. 5.17 unveils that there are less outliers (vehicles) which could encounter with a longer sojourn time. In QBPC and DBPC, the sojourn time which outliers encounter are close to 500 sec.

We discussed that DBPC has a shorter sojourn time with which vehicles could encounter, and it achieves a better fairness than QBPC in our previous work. Here, the performance of the DSARSA agent achieves a even shorter sojourn time for outliers and better fairness for all vehicles than DBPC. The curve of DSARSA lying between QBPC and DBPC denoting that it has a comparatively lower probability for vehicles to have the same sojourn time as they do in QBPC and DBPC. For example, the probability of having 100 sec delay in DSARSA is only 0.4; however, the probability is more than 0.8 in DBPC. DSARSA starts to outperform QBPC when delay exceeds 130 sec.



(a) Average delay and reward **(b)** Average queue length and reward



(c) Maximum delay and reward **(d)** Fairness and reward

Figure 5.15: The trends of delay, queue length, fairness, and reward when training a DSARSA agent

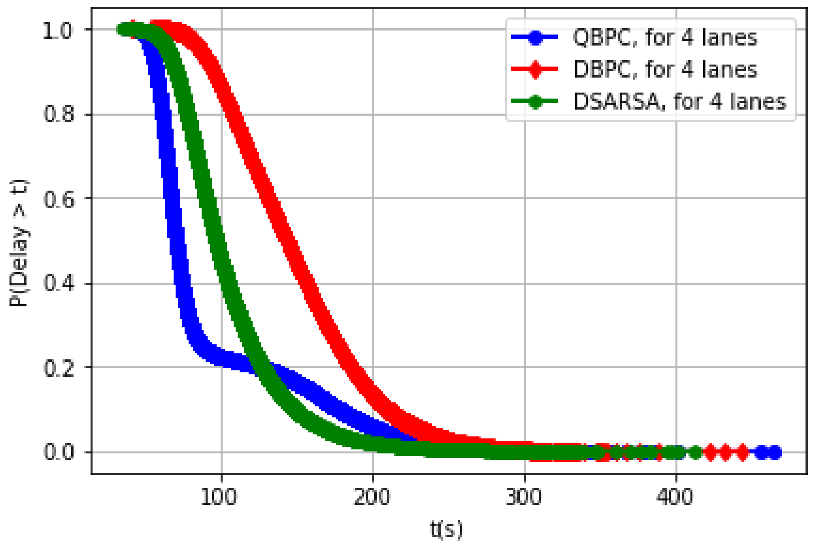


Figure 5.16: Comparison of delay distributions for queue-based BP control (QBPC), delay-based BP control (DBPC), and DSARSA

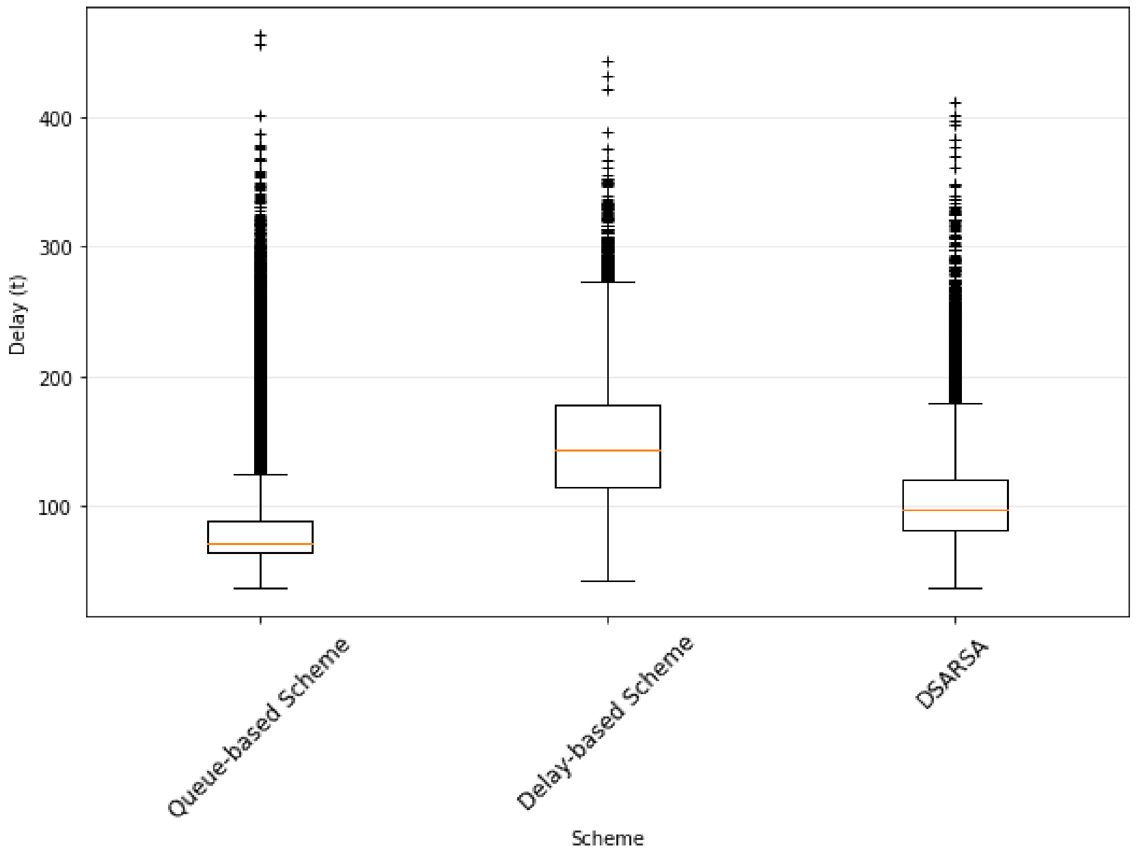


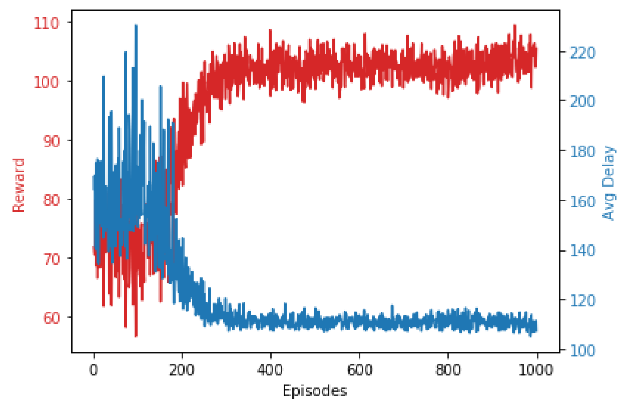
Figure 5.17: Comparison of box plots for queue-based BP control (QBPC), delay-based BP control (DBPC), and DSARSA

5.5.3.4 Learning Performance of 2DSARSA

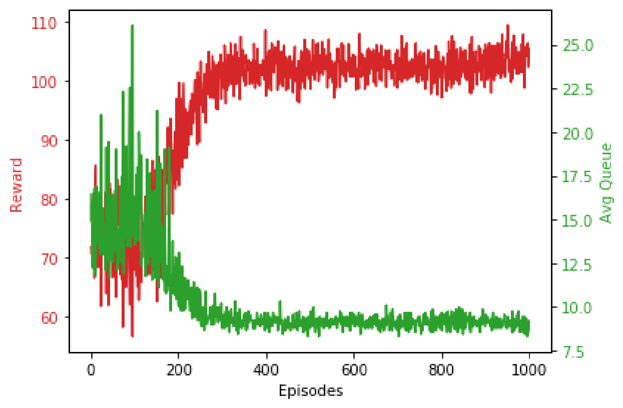
In this chapter, we mentioned that the proposed DRL framework has certain flexibility by adopting various RL-based algorithms with deep learning, and high extendability by incorporating the dueling architecture, memory replay as well as other existing techniques. We propose a new method, 2DSARSA, utilizing SARSA with deep learning, the dueling architecture, and memory replay. The learning performance of the DSARSA agent has already been examined in the last paragraph. The agent can efficiently learn from a complex environment by DSARSA. As shown in Fig. 5.18, the training process of the proposed 2DSARSA, which uses the dueling architecture, converges faster than other DRL-based agents. During the training, similar to DSARSA, we pre-train for 2DSARSA; hence, there are spikes around the 100-th training episode.

As can be seen, the trend of reward keeps raising all the way after the 200-th episode and the reward is getting higher and higher over episodes. With respect to the average end-to-end delay and average queue length, the learning starts to become stable around the 200-th episode in Fig. 5.18a, and 5.18b. In contrast, DSARSA requires much more episodes to be converged where it becomes stable around the 1000-th episode. In Fig. 5.18d, it is clear that the fairness of the 2DSARSA agent reaches a perfect level around the 200-th episode compared to that of the DSARSA agent which achieves the same level around the 400-th episode in Fig. 5.18d. Our analytics shows that the learning performance of the 2DSARSA agent converges by only requiring 300 episodes, which is less than that of the DSARSA agent that converges by using 1000 episodes. The dueling architecture indeed stabilizes the whole training process and speeds up the convergence.

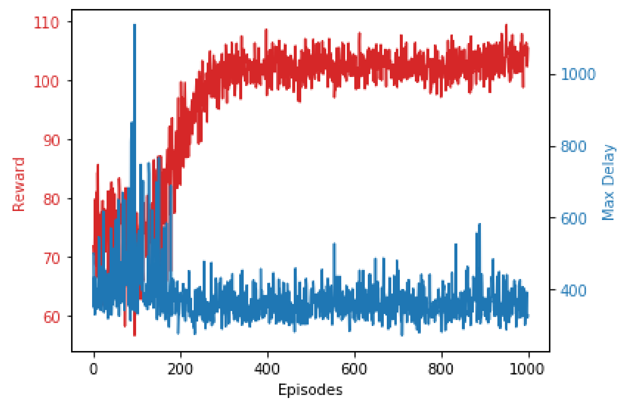
Based on this successful training, the model of the neural network is stored and used to run 10-hour traffic arrival data. The performance of the 2DSARSA agent is shown in Fig. 5.19 and 5.20. It is quite obvious that the 2DSARSA agent outperforms the conventional BP-based algorithm proposed in [47] and [117] for multiple intersections after 1000 training episodes. Regarding the overall fairness, Fig. 5.19 shows the probability of undergoing a long delay for vehicles in the traffic network. The distribution of a extremely long delay (the green tail) for 2DSARSA is much lower than QBPC, DBPC, and other DRL-based agents. On the other hand, as shown in Fig. 5.20, there are less outliers (vehicles) who experience a end-to-end travel delay more than 400 sec comparing to DSARSA.



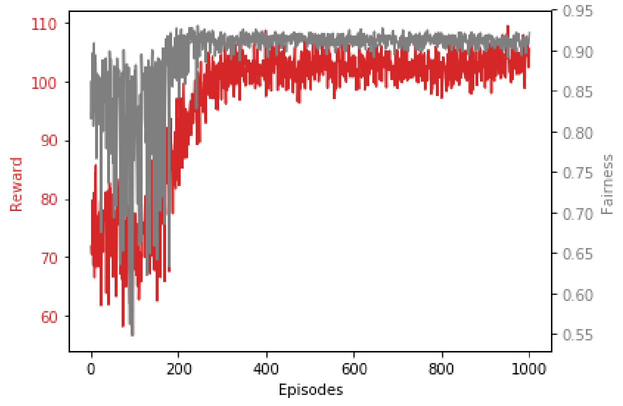
(a) Average delay and reward



(b) Average queue length and reward



(c) Maximum delay and reward



(d) Fairness and reward

Figure 5.18: The trends of delay, queue length, fairness, and reward when training a 2DSARSA agent

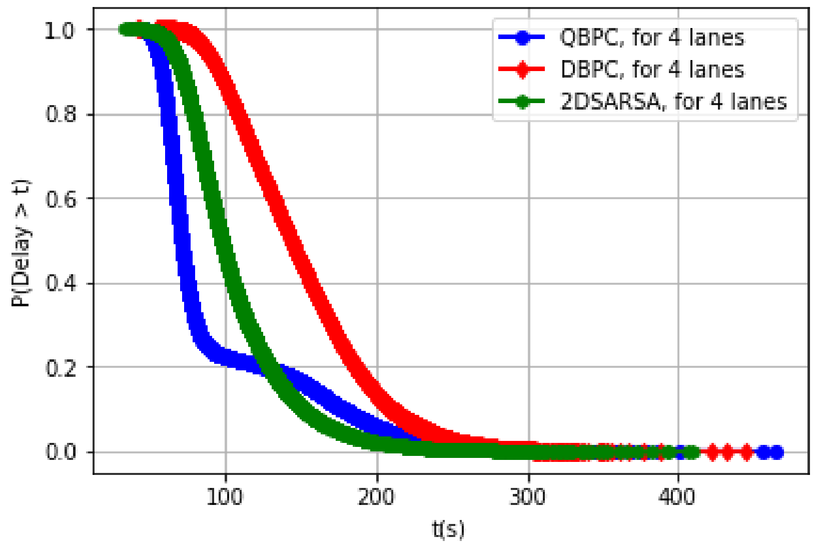


Figure 5.19: Comparison of delay distributions for queue-based BP control (QBPC), delay-based BP control (DBPC), and 2DSARSA

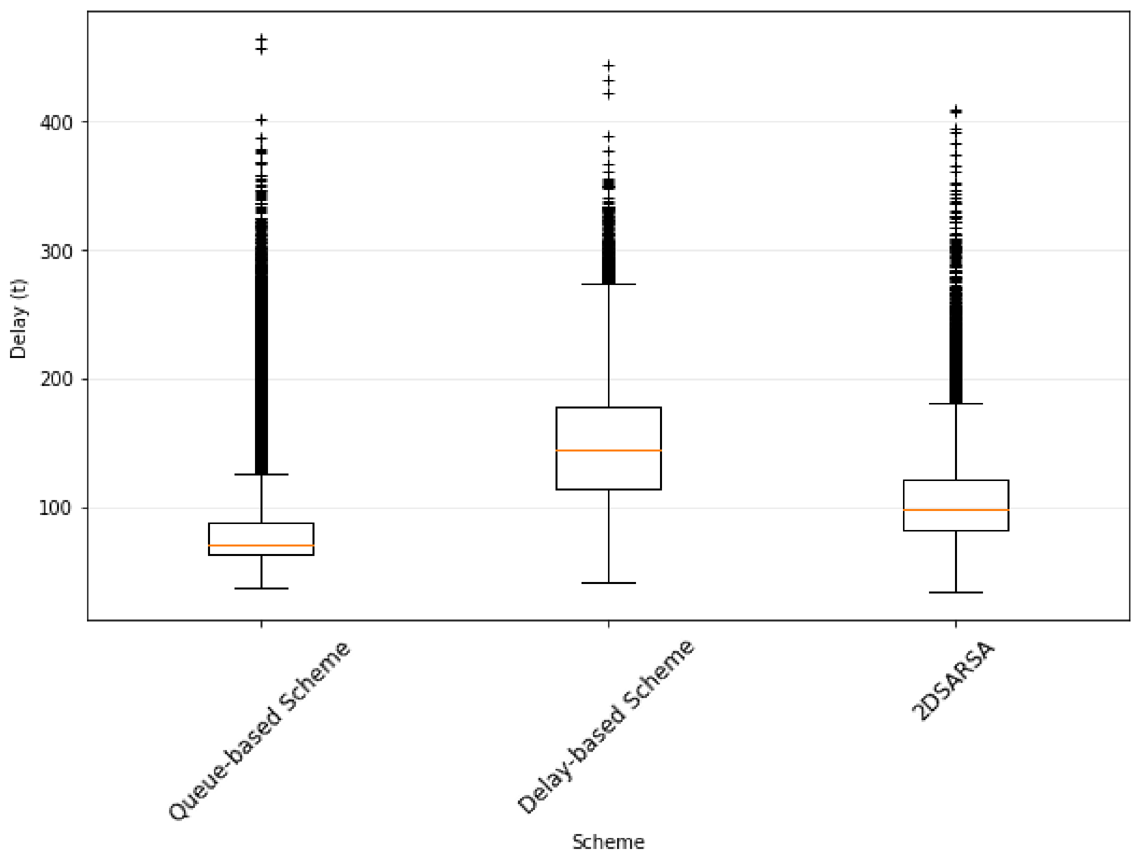


Figure 5.20: Comparison of box plots for queue-based BP control (QBPC), delay-based BP control (DBPC), and 2DSARSA

5.5.4 Critical Evaluations

QBPC serves the lane with the highest queue length, lanes with lower arrivals will get starving. DBPC was proposed for solving the last vehicle problem in [47, 117] where vehicles are scheduled more fairly by the DBPC. Hence, the tail of the delay distribution of DBPC is lower than that of QBPC, which means vehicles will not suffer from a longer delay. In this section, their performance is the baseline compared to DRL-based agents. The learning performance of all DRL-based agents is shown in Fig. 5.21.

First, we want to elaborate more on poor performance of DQN and 3DQN. DQN and 3DQN perform terribly and their learning performance unveils that Q-learning with deep learning and advanced architectures are not appropriate for a complicated environment. The fundamental difference between on-policy and off-policy methods is that off-policy learns by searching the best action from several policies. DQN and 3DQN utilize an off-policy learning method, Q-learning, which

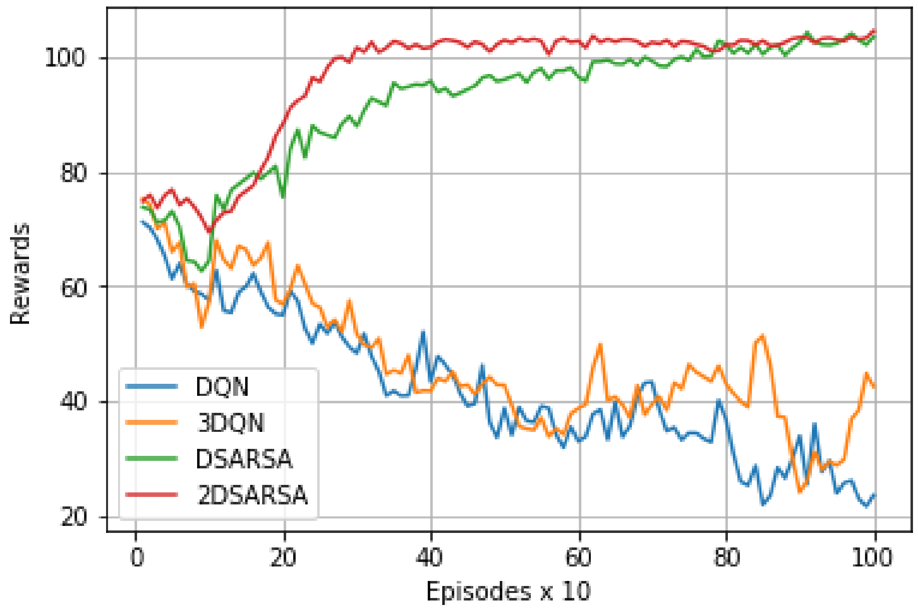


Figure 5.21: Comparison of learning performance among DQN, 3DQN, DSARSA, and 2DSARSA

switches among multiple policies such that maximizes the expected accumulated reward. For many Atari games applying Q-learning, this approach easily get rid of a poorly initial policy and jump to the promising one which returns a better accumulated reward eventually. However, as the state dimension and action space grow exponentially, this approach reveals its weakness while dealing with a sophisticated environment. Unlike the conventional tabular Q-learning, for unlimited (possibility) number of states, it is impossible to visit all state-action pairs; hence, the action selection would be difficult for a Q-learning based agent since it cannot figure out which action is the "best" option to take without going through each state-action pair. The training processes of DQN and 3DQN are oscillating and not converged.

On the other hand, DSARSA and 2DSARSA are capable of learning effectively from the environment during training. Without greedily searching the best policy, SARSA with deep learning learns by following the initial policy and selecting the next action within this policy. Their training processes both work perfectly and are converged at the end. On-policy learning with deep learning shows a certain level of stability while learning from a complicated environment. Because on-policy learns by following the same policy while learning, the whole training process will still be converged even if the policy chosen by SARSA is initially not good enough. The dueling technique, which separates the state-value and action advantage, can effectively filter out useless actions and force the entire training become converged eventually. DSARSA equipped with the dueling architecture can significantly alleviate oscillation during the training process and achieve more optimality than BP-based schemes, DQN, 3DQN, and DSARSA. In Fig. 5.21, the learning performance comparison reveals that 2DSARSA obviously outperforms other DRL-based agents and converges quicker than others.

In a nutshell, 2DSARSA combining the dueling architecture and DSARSA, outperforms other methods and agents. The combination of the dueling architecture and DSARSA indeed stabilizes the throughput optimality of the traffic network. Based on our experiments, purely applying BP-based algorithms to a traffic network does not achieve the global optimality. There is still some room where we can struggle for optimization. Learning through following the same policy provides a good entry point to explore more optimality when doing scheduling in a traffic network.

5.6 Conclusion

The experiments have shown a promising prospective of using DSARSA and the dueling architecture for multi-intersection traffic light control. Due to several success examples in learning Atari games by the deep Q-learning [71], most researchers who have contributed very much on the traffic control problem only concentrated on using off-policy learning methods with deep learning. The DQN and 3DQN would work fine in an environment such as an isolated intersection that is easy to understand. But, when it comes to a complicated environment like a grid network, DQN or 3DQN are no longer enough to solve the problem. The proposed 2DSARSA performs well even if the environment is far more complex than one intersection. Our outcomes show that the 2DSARSA agent is able to learn from the proposed TFMs and even achieve more optimality and fairness than BP-based algorithms and other DRL-based agents.

As our future work, first, more complicated environment including extending the size of the grid network from 3 by 3 to 5 by 5 and increasing the number of traffic lanes will be taken into account. More phases such as turning left or right will be considered as well. Second, a power metric over a fairness ($\frac{\frac{\lambda}{d}}{1-f}$) will be considered as our new reward function to enhance the agent's learning performance where λ denotes the throughput of the network, d represents the average delay of vehicles, and f is the Jain's fairness index [45] of all vehicles in the network.

Third, our experiments already showed that the dueling architecture indeed contributes to select valuable actions as the action space grows larger. However, more robust architecture will need to be designed if the size of the grid network is extended to 5 by 5 or even larger. Instead of the dueling architecture, what else should be included for enhancing the ability of recognizing valuable actions? A novel architecture such as multi-level control which is more suitable for the estimation and action selection will be proposed while the action space is getting huge.

In addition, at present, the depth of the input data is 1, more useful information such as queue length or sum of delay will be involved to formulate a plentiful data volume which would provide more comprehensive knowledge for the agent. Last but not least, the simulations of above improvements will be done by using SUMO for the future work.

Chapter 6

Two-level Control Architecture for Traffic Signal Control

6.1 Introduction

Due to the remarkable achievements made by reinforcement learning (RL) in Atari, Go, and real-time strategy games, simple gaming environments can no longer satisfy scientists' curiosity. In recent years, RL algorithms have started to be applied to more and more real-world applications in a wide variety of fields including traffic signal control (TSC). A detailed survey on applying RL algorithms to TSC is summarized in [42]. However, in these real-world applications, environments which are usually large and complicated require a much larger size of states to describe changes after actions are applied. As RL has been considered in more and more complicated environments, soon, scientists found that conventional RL can no longer competently tackle states with a larger size from complicated environments because the state space grows exponentially with the size of states; namely, much more memory space is needed to store Q-values for all the state-action pairs. Eventually, states with a larger size degrades the learning performance of a RL agent because it fails to update all state-action pairs in an extremely huge Q-table. The curse of dimensionality [25] caused by environments with a huge state space also known as the huge state space problem is the fundamental issue of tabular RL.

Many solutions have been proposed to address this intractable issue, and they can be summarized into two categories: 1) decentralized and 2) centralized approaches. In the decentralized approach, an idea of a hierarchical architecture by organizing multiple RL agents was applied to traffic light control problems in a large geographical area [4, 8, 50, 78, 105]. To effectively reduce workload for each agent, the large geographical area is decomposed into several regions and each region is governed by a RL agent which monitors performance of traffic light control in the region. Each region is further split into many individual intersections, each of which is assigned to a RL agent determining traffic light schedules. This approach evolves into a hierarchy-based RL control for a traffic network. In contrast, the centralized approach developed deep neural networks (DNNs) substituted for the Q-table. The underlying blocks of a DNN are neurons - the data processing units which are responsible for weighting input values and generating intermediate features of the inputs for the next hidden layer. A DNN can be utilized as a function approximator to approximate a Q-value for each state-action pair (s, a) instead of mapping (s, a) to a Q-value from the Q-table, and hence, is not limited by the size of states. The approximation-based RL using DNNs such as convolutional neural network (CNN) has been proposed to address traffic light control problems [32, 34, 58, 79, 100, 114, 115].

The above studies on applying decentralized and centralized approaches have been investigated very well and revealed a significant success in overcoming the fundamental issue of RL. With that being said, drawbacks inherent in these solutions make them imperfect and breed two new problems 1) global optimality and 2) convergence which have not been well studied and addressed yet. First, the decentralized approach organizing multiple RL agents in a hierarchical architecture is capable of tackling a great geographical area, but a larger area needs to be recursively divided into several regions. This hierarchical architecture leads to additional overhead of sharing state information, and requires a reliable communication channel for multi-agent RL. Otherwise, the decentralized approach cannot guarantee global optimality. Second, the centralized approach possessing global information usually requires a significant number of training episodes to train a model because the action selection will be learned by several neural layers rather than simply updating values in the Q-table. A DNN has to go through many episodes in order to fine-tune the weights in neurons by backpropagation. Then, the agent is able to select the action with the optimal Q-value. Moreover, the mathematical guarantees of convergence can no longer be guaranteed if non-linear approximation functions, DNNs are applied. Due to the two inherent problems we concluded, neither the existing approaches can be regarded as an optimal solution to a large scale of network with non-stationary traffic demands and complex traffic patterns; therefore, countermeasures need to be considered when designing a state-of-the-art intelligent transportation systems (ITS) for the next generation.

In this chapter, we propose a hybrid-based two-level control (TLC) architecture that incorporates decentralized backpressure (BP) controllers with a centralized deep reinforcement learning (DRL) agent in a hierarchical architecture. The idea is to exploit insights of traffic signal phases selected by the decentralized BP controllers based on the local information they observed into the improvement of traffic signal phases determined by the DRL agent. Our idea of the collaboration can be forged by multi-level control to further achieve higher learning performance. The centralized DRL agent and a number of decentralized BP controllers are organized into a TLC architecture where the DRL agent with global information is positioned at the higher level and the BP controllers with local information are at the lower level. The DRL agent interacts with the BP controllers to co-determine a joint action composed of traffic signal phases based on both global information (i.e., traffic flow maps (TFMs)) and local information (i.e., pressure-of-the-lanes (POL)). Their goal is to find the optimal policy (i.e., a series of actions based on given states) that maximizes the expected accumulated reward for the network of intersections. We found that the decentralized and centralized approaches can actually compensate each other for their own defects. That is to say, the decentralized BP controllers in a hierarchical architecture can work in close collaboration with the centralized DRL agent for compensating for the lack of global information while the centralized DRL agent with global information can cooperate with decentralized controllers in making the training process converged. So far, research on TSC for a large traffic network concentrate on using either decentralized or centralized approaches. A hybrid-based approach which has not yet been studied is a brand new idea that exploits BP algorithms and DRL. To the best of our knowledge, this is the first research that investigate the feasibility of applying the innovative idea (hybrid-based approach) and evaluate its impact and performance compared with our previous work (2DSARSA), well-known BP algorithms, and famous DRL models (DQN, 3DQN, DSARSA) in detail in the context of TSC.

6.1.1 Organization

The rest of this chapter is organized as follows. Our contributions are listed in Section 6.2. In Section 6.3, we introduce the problem and challenges for the traffic network. In Section 6.4, we introduce the TLC-based optimization framework for the traffic network. In Section 6.5, we propose the hybrid-based TLC architecture to improve the learning performance of our previous work (2DSARSA). In

Section 6.6, we discuss the performance of TLC and compare with that of 2DSARSA in terms of the rewards, the average end-to-end delay, average queue length, and fairness. We also compare with BP algorithms which can be regarded as baseline algorithms. Finally, we conclude and discuss future work in Section 6.7.

6.2 Contribution

- We propose a hybrid-based TLC architecture which integrates decentralized BP controllers with a centralized DRL agent in this chapter. To the best of our knowledge, this is the first research combining BP with DRL on TSC. Our results unveil that the proposed TLC performs better and converges faster than 2DSARSA and the other DRL agents.
- The collaboration between centralized and decentralized controllers in the two levels substantially improves the network throughput and latency by exploiting the insights of using the local information.
- We train the proposed TLC model using two generated datasets, 1-hour and 10-hours traffic arrival data and evaluate the performance with the two schemes of the well-known BP algorithm.
- We compare the learning performance of DQN, 3DQN, DSARSA, 2DSARSA, and the proposed TLC. At present, many existing RL techniques such as DQN [72], double deep dueling Q-Network (3DQN) [58], and DSARSA [128] have been developed and tested. In order to prove the performance of TLC, different agents based on above DRL techniques are implemented and trained separately for comparisons.

6.3 Problem and Challenges

6.3.1 The Curse of Dimensionality

RL and dynamic programming (DP) are two well-known approaches which have been applied to solve optimization problems. They share the same idea of using Bellman equation to discover the optimal policy. In cases that DP can be applied, a large and complex problem is recursively divided into several smaller and simpler sub-problems if these sub-problems can be parts of the problem. In temporal difference (TD) methods, a RL agent learns from high variance rewards based on Bellman equation. The major difference is that DP requires perfect knowledge of the environment (i.e., the state transition function and reward function) whereas model-free RL allows an agent to explore and learn how to take actions by interacting with an environment. Therefore, applications of DP is limited by the curse of modelling. Although RL is not plagued by the curse of modelling, it suffers from environments with large state space due to table look-up. The estimated Q-values must be stored for all state-action pairs. Hence, the space complexity increases exponentially as the number of possible states grows also known as the curse of dimensionality [25, 101] as well as environments with large state space lead to slow learning because mapping a particular state-action pair to its Q-value requires comparatively longer time.

In TSC problems, a RL agent usually needs to tackle traffic light control in a large geographical area consisted of multiple intersections, heterogeneous traffic demands, multi-modal vehicles, and a number of traffic phases. That is to say, a large traffic network described by the above environmental variables is a state to which the agent is applied. A higher resolution of the state description further enhances the agent's understanding of a traffic environment and hence, it is capable of selecting better actions to respond to changes of a traffic environment to boost its performance in traffic light control. However, state space of a traffic environment grows exponentially both with increase in the resolution of the state description and with the number of traffic intersections. For example, in one-dimensional state description, a state is represented by a vector of numerical values, each of which denotes the length of a queue corresponding to a traffic lane. If the maximum length of a queue is 10 vehicles, then there are 10^8 possible states for an intersection with 8 traffic lanes. The possible number of states grows from 10^8 to 10^{16} for a traffic environment with two intersections. The number of states can be exponential growth but the memory space cannot be scaled up as more and more intersections are considered. This is the reason why conventional RL cannot be applicable to problems with a large and complex environment.

6.3.2 Challenges to The Existing Work

In response to the curse of dimensionality, solutions have been proposed in terms of architecture designs and mapping functions. One intuitive solution to address the huge state space issue is to reduce the state size for a RL agent. By decomposing a great traffic environment into several sub-regions or even intersections, the state size which each RL agent needs to process can be reduced to acceptable space complexity to which a Q-table is applicable. Multiple RL agents organized into the same hierarchical architecture [4,8,50,78,105] can be categorized as the decentralized or hierarchy-based approach as shown in Fig. 6.1. The other solution is to design a completely new mapping function to substitute for the Q-table. The Q-table stores estimated Q-values for all state-action pairs during the training stage in order to be able to instantaneously map a specific state-action pair to its Q-value without re-computing. The idea is to reduce time complexity but increase in space complexity is the cost that Q-table must pay in return. To break this restriction, a function approximator is applied to approximate Q-values for all state-action pairs. DNNs are the most popular function approximator which has been considered in much research on TSC [32,34,58,79,100,114,115] and can be categorized as the centralized or approximation-based approach in Fig. 6.1. A DNN is consisted of a number of data processing units, neurons, which are used to weight input values and output intermediate features of the input data for the next hidden layer. DNNs are powerful and flexible to tackle different forms of input data. CNN, a specialized version of DNN, is designed to process the data with Euclidean structures such as two-dimensional images. Compared to the Q-table, DNNs as function approximators do not require large memory space to record estimated Q-values during the training stage; instead, it takes much longer time to go over a considerable amount of training data to be able to select an optimal action given a state by fine-tuning weights in neurons.

However, there are two challenges inherent in the above two approaches. First, in the decentralized approach, due to the decentralized nature, each RL agent observes local state information (i.e., only a part of a traffic network) and determines traffic signal phases for a region based on the local observation. Each agent can only guarantee local optimality for a region instead of achieving global optimality for the entire network. The dissemination of state information can directly influence the quality of decisions determined by RL agents because both upstream flows from and downstream flows to neighbors have to be taken into account; namely, two or more agents need to collaborate together by sharing state information they have. Thus, state sharing is necessary to make sure that all RL agents understand the up-to-date state information (e.g., queue length of lanes and flow dynamics of intersections) from their neighbors before determining traffic signal phases that achieve global optimality and hence, additional overhead of forwarding observed state information to neighboring RL agents is required as well as a reliable communication channel among multiple RL agents needs to be established. Furthermore, to what extent state information need to be spread

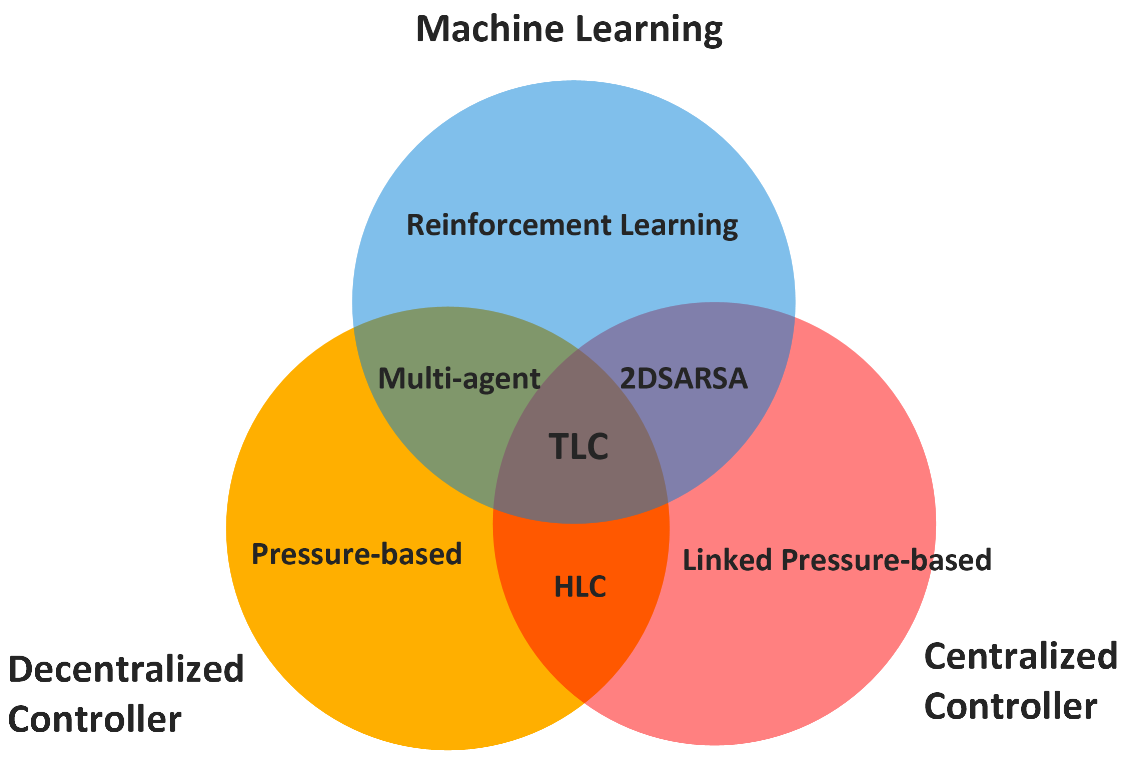


Figure 6.1: Categories for existing approaches to TSC

is an open question to the ITS research community. Second, in the centralized approach, without the participation of the Q-table, the action with the optimal Q-value is approximated by DNNs which requires an extremely long period of time to finalize backpropagation that updates weights in neurons. Due to the non-linear nature of DNNs, even though training is performed on sufficient data, the convergence of the learning is not guaranteed at the end of the training. That is to say, DNNs could be unstable and unable to learn anything from environments with a large size of states.

In this chapter, we have been seeking for an innovative solution capable of combining the advantages of both decentralized and centralized approaches to achieve higher performance on TSC. A hybrid-based architecture forged by multi-level control enables a collaboration between a centralized DRL agent with decentralized BP controllers in a hierarchical architecture and can be the solution which we have been seeking for. We propose a hybrid-based TLC as shown in Fig. 6.1 to co-determine a set of traffic signal phases that optimizes the network throughput and traffic latency for a network of intersections. We assumed that the local state information described by number of vehicles in lanes (i.e., pressure or queue length) in local intersections can be exploited to boost learning performance of a DRL agent. Wei *et al.* [113] proved our idea by using pressure-based states for a DRL agent. They modelled pressure by the arriving and leaving vehicles in intersections and utilized them as local information to describe the states. They showed that improvement on TSC can be achieved for a traffic network of multiple intersections by learning from the pressure-based states and reward function, which is well-supported by BP theory and has been proved to be optimizing the throughput of a traffic environment. We found that the collaboration between a DRL agent and a number of BP controllers achieves better learning performance.

6.4 Problem Formulation and Optimization Framework

We formulate a TSC problem as an optimization problem relating the policy (i.e., a series of actions based on given environmental states) and the reward function. The environment is a 3 by 3 grid traffic network with 9 intersections, each of which is assigned to a BP controller. There are two traffic signal phases: North-South and East-West as shown in Fig. 6.3. The conjunction of any two adjacent intersections where vehicles leave one for the other will become a traffic bottleneck impeding global optimality if the traffic flows from the upstream and to the downstream are not considered. The proposed TLC is designed to solve this issue and achieve optimal throughput and/or end-to-end delay performance for the traffic network. In the proposed TLC architecture, for the intersections, the DRL agent collaborates with a number of BP controllers on deciding which traffic signal phase to be activated. The goal of the DRL agent and the BP controllers is to co-determine the optimal policy that maximizes the network throughput and minimizes the average end-to-end delay for the network of intersections in the optimization framework as shown in Fig. 6.2. All the BP controllers are synchronized and operate per time slot of duration t. For every time slot t, each BP controller independently selects which traffic signal phase to be activated based on POL in an individual intersection. The 9 traffic signal phases selected by the BP controllers are an action. On the other hand, the DRL agent decides whether or not to modify the traffic signal phases based on an observed state s_t described by the TFM and interacts with the BP controllers to co-determine a joint action a_t which is an assignment of traffic signal phases to the 9 intersections. The joint action a_t can be represented by a bit vector of length 9 because only two movements are considered at each intersection. At the end of every time slot t, the DRL agent receives a reward r_t which indicates how well the joint action a_t in state s_t impacts the the network throughput and/or average end-to-end delay.

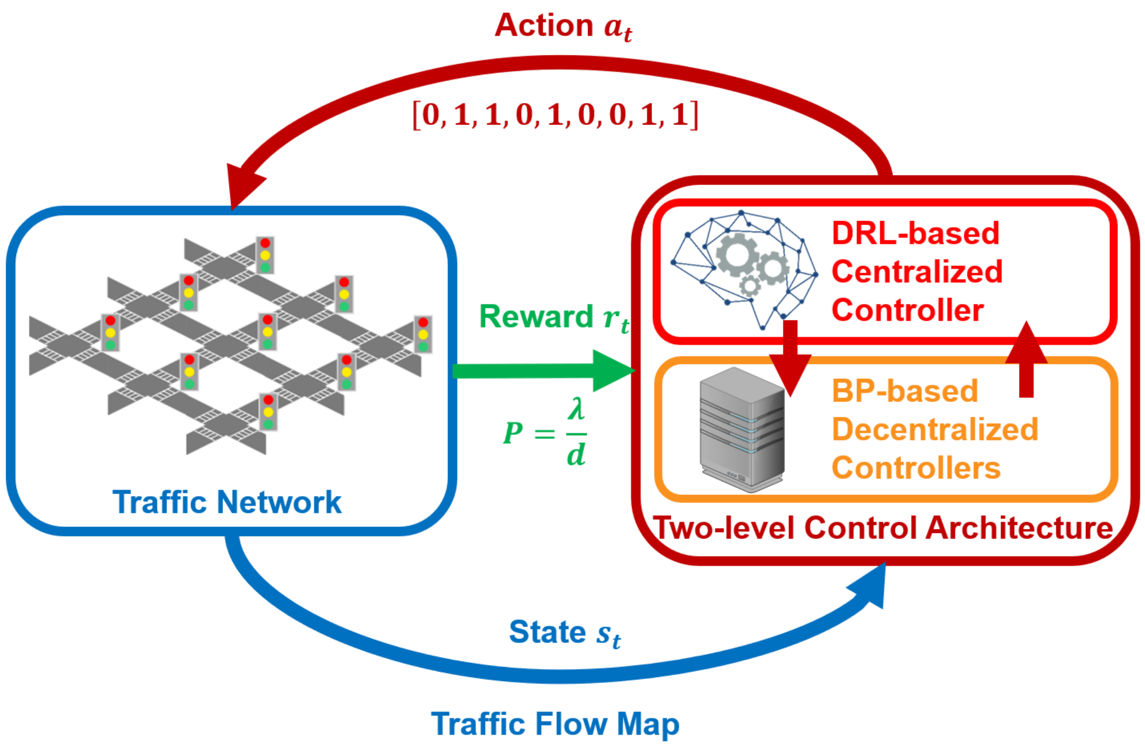
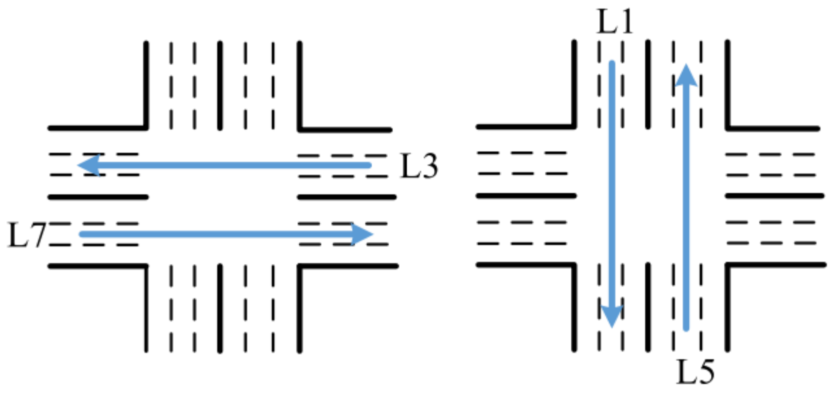


Figure 6.2: The TLC-based optimization framework for a 3 by 3 grid traffic network consisting of 9 intersections



(a) P0 with lanes 3 and 7 **(b)** P1 with lanes 1 and 5

Figure 6.3: Each BP traffic signal controller activates one of the two phases, P0 for horizontal flow and P1 for vertical flow, for every time slot t.

6.5 The Proposed Hybrid-based Two-level Control Architecture for Traffic Signal Control

We propose a hybrid-based TLC architecture that organizes a centralized DRL agent and a number of decentralized BP controllers into the same hierarchical architecture as shown in Fig. 6.4. The centralized DRL agent observing global information (TFMs) is positioned at the higher level, whereas the BP controllers observing local information (POL) are positioned at the lower level. Each BP controller at lower level independently chooses one of the two phases based on the POL it observes at time slot t. Pressure-of-the-lanes referred to as POL denotes the number of vehicles (pressure or queue length) in non-conflict traffic movements which can be released simultaneously at an intersection. The DRL agent trained by TFMs at the higher level possesses a thorough understanding of flow dynamics in the network. Therefore, it is responsible for deciding whether or not to modify the chosen traffic signal phases so as to achieve global optimality. Through modifications of the chosen traffic signal phases, the DRL agent collaborates with BP controllers on co-determining a joint action composed of traffic signal phases to be assigned to the 9 intersections. The idea of the collaboration between the two levels is to gain insight of traffic signal phases chosen by the BP controllers using the POL they observed into the improvement of traffic signal phases determined by the DRL agent. We elaborate on the two levels more in the following sections.

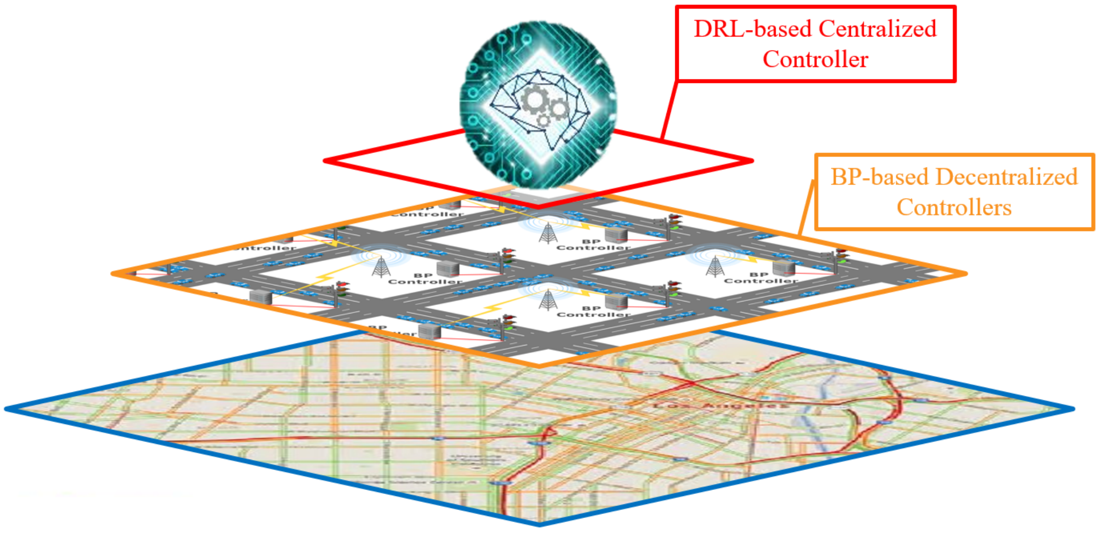


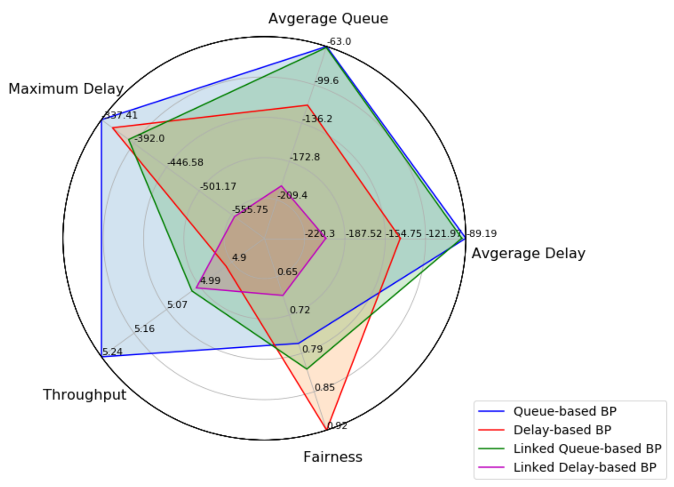
Figure 6.4: The TLC architecture

6.5.1 Backpressure Based Decentralized Controller

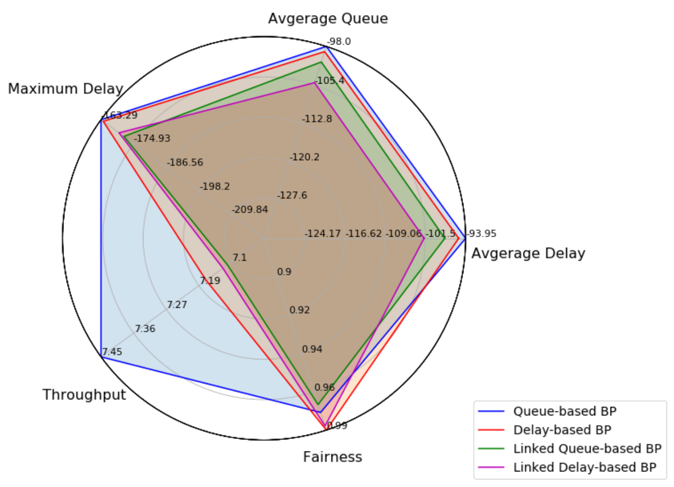
The detailed studies on BP algorithms can be found in Section 2.1. The studies have proved that BP algorithms can achieve maximum network throughput and global optimality without any prior knowledge about arrival rates. Moreover, the concept of *pressure* is not restricted to only queue length but also the other pressure metrics such as head-of-the-line (HOL) delay and sum-of-delay which can be used to develop various BP schemes. The detailed study on the performance of different BP schemes is mentioned in [121, 122].

We select the queue-based BP scheme as the TSC algorithm for the decentralized controllers at the lower level because the queue-based BP scheme reveals better performance regarding the throughput and average end-to-end delay. As shown in Fig. 6.5, we evaluate the four different BP schemes by the five performance metrics; namely, throughput, maximum delay, average queue, average delay, and fairness. Note that *linked* schemes denote that the pressure difference between any two adjacent intersections are considered as the *pressure* when determining traffic signal phases for a network of intersections. In Fig. 6.5a, the four BP schemes are applied to determine traffic signal phases for heterogeneous arrivals. The delay-based BP scheme shows the best fairness. In the linked delay-based BP and linked queue-based BP schemes, we would have expected that at least the throughput and average delay can be improved if pressure difference is considered as the pressure for determining phases in a network. But, the improvements are not very obvious compared to the queue-based BP and delay-based BP schemes. One the other hand, in Fig. 6.5b, they are applied to do traffic signal control for homogeneous arrivals. Under the homogeneous traffic, their performance looks close to each other because the homogeneous traffic is less challenging compared to the heterogeneous one. Among the four schemes using different pressure metrics, the queue-based BP scheme results in highest throughput and lowest average end-to-end delay, which is consistent with the metrics optimized by the proposed power based reward function in Section 6.5.2.3. Hence, adopting queue-based BP algorithm for the controllers substantially assists in co-optimizing system throughput and average end-to-end delay for the DRL agent.

Controllers at the lower level adopt the queue-based BP algorithm to the TSC problem. For every time slot t, without considering any additional information from neighbors, each controller at an intersection independently selects the phase of the traffic movements that maximizes the total



(a) Performance of different BP schemes with heterogeneous arrivals



(b) Performance of different BP schemes with homogeneous arrivals

Figure 6.5: The radar plots show that four different BP schemes are evaluated by the five performance metrics i.e., throughput, maximum delay, average queue, average delay, and fairness.

pressure released where the pressure is measured based on the queue length. The chosen traffic signal phases are forwarded to the higher level for the further modifications by the DRL agent. The phase of the traffic movements is selected by Eq. 6.1 given by

$$\vec{p}^{*}(t) \in \underset{\vec{p}\in\mathcal{S}_{\mathcal{P}}}{\operatorname{argmax}} \sum_{P_{i,j}=1} \gamma_{i,j} \cdot Q_{i,j}(t) \cdot \mu_{i,j}(\vec{p}) \tag{6.1}$$

where the right hand side calculates the queue length $Q_{i,j}(t)$ weighted by two parameters $\mu_{i,j}(\vec{p})$ and $\gamma_{i,j}$. The parameter $\mu_{i,j}(\vec{p})$ denotes the number of vehicles that can be transferred through the connected edge when signal phase $\vec{p}$ is activated. The parameter $\gamma_{i,j}$ in which the subscripts i and j refer vehicles in lane i destined to lane j, is a positive constant that can be used to give priorities to particular movements when there are multiple types of traffic. In our simulation, $\gamma_{i,j}$ is set to 1 for all movements.

6.5.2 Deep Reinforcement Learning Based Centralized Controller

In the design of the higher level controller, we expect that a centralized controller with the global information is responsible for managing incoming and outgoing traffic flows, operating at a fine-grained time scale, and learning an appropriate traffic phase for each intersection at each time slot; meanwhile coordinating an entire traffic network consisted of multiple intersections. BP controllers at the lower level follow the modifications determined by the centralized controller, which makes decisions accordingly to the globally determined performance objectives.

DRL is an ideal unsupervised learning that satisfies our expectation. Through the global information (TFMs), we train a DRL agent which integrates on-policy learning with an approximator implemented by the dueling architecture, CNN, and memory replay buffer to learn how to modify the traffic signal phases selected by the BP controllers at the lower level so as to reduce traffic latency (average end-to-end delay) and increase overall throughput for a traffic network. The dueling architecture helps the selection of action by separating the estimations of state-value and action advantage. It will benefit the DRL agent if the action space is large. The experience memory replay breaks the strong temporal correlations and speeds up the learning process. Hence, these strategies are applied to solve the huge state space and convergence problems. The time series TFMs (states) include informative flow dynamics of a traffic network with several intersections which assists the DRL agent in learning from the traffic environment. The proposed TFMs are synthesized to represent the traffic environment over time slots. They are formulated by using Poisson arrival data from multiple intersections. First, a state is fed into the CNN at every time slot t. Then, given the state, the network parameter set θ are trained and updated by optimizing the loss function given by Eq. 6.5, and the optimal action is generated according to the learned policy.

In any DRL model, the state, action and reward function need to be carefully defined. In the following sections, we elaborate more details on the state, the action and the reward function.

6.5.2.1 State Space

We propose a novel state description using a traffic flow map (TFM) which is an image encoding the HOL sojourn times of lanes at each intersection and differences of HOL sojourn times between adjacent intersections (i.e., new delay metrics). To the best of our knowledge, we are the first work of using TFMs as the states. Previous research has shown that RL agents can be well-trained by images [62, 71]. However, most research [32, 34, 58, 100, 115] either formulated a matrix for an intersection by marking 0 and 1 based on coordinates of vehicles or used queue length of vehicles at the intersection to describe the state.

The idea of utilizing the new delay metrics to establish the linear relation between links for multi-intersection traffic was proposed by Ji *et al.* [47] because they found that a delay is not sufficient for capturing flow dynamics of a traffic network with several intersections, and the delay-based solutions cannot guarantee the linear relation. Our previous studies [117, 121, 122] proved their idea and showed that the delay-based BP scheduling algorithm achieves better fairness than the queue-based BP scheduling algorithm using queue lengths. The delay metrics $\hat{W}_{i,k}$ calculating a difference of HOL sojourn times between adjacent intersections and $\Delta\hat{W}_{i,k}$ calculating a difference of any two $\hat{W}$ are defined by

$$\hat{W_{i,k}}(t) = W_{i,k}(t) - W_{i,k-1}(t) \tag{6.2}$$

$$\Delta\hat{W_{i,k}}(t) = \hat{W_{i,k}}(t) - \hat{W_{i,k+1}}(t) \tag{6.3}$$

where $k-1$ and $k+1$ denote the neighbouring upstream and downstream intersections, respectively, and $W_{i,k}(t)$ denote the HOL sojourn time of lane i at intersection k at time slot t in a traffic network. If an intersection has no neighbors (the edge intersections in our grid network), the HOL sojourn time values are set to 0. These delay metrics guarantee a linear relationship between queue lengths and delays for a traffic network with multiple intersections. Note that similar to the delay metrics one can define queue metrics based on the differences in the queue lengths between the adjacent intersections.

The traffic flow information of each intersection with respect to the delays and the delay metrics is stored in a 5×5 matrix as shown in Fig. 6.6. Note that for each movement there is one lane in each direction; consequently, there are 4 lanes at each intersection. The blue circle in the center indicates the average HOL value of 4 HOL sojourn times in each intersection. Orange circles denote 4 HOL sojourn times for 4 different lanes where the values of these circles will be the corresponding HOL sojourn time values. Red and brown circles represent delay metrics defined in Eq.6.2 and Eq.6.3 with corresponding neighbouring intersection. Note value of each circle is a real number that can be mapped to a color scale in which high values are in red and low values are in yellow.

A TFM of an entire network is created by laying out the traffic state information of each intersection in a two dimensional grid while maintaining the adjacency of the intersections. The structure of a TFM for a 3×3 grid network (consisting of 9 intersections) is shown Fig. 6.8. The agent learns from TFMs and determines the best modifications for the BP controllers according to

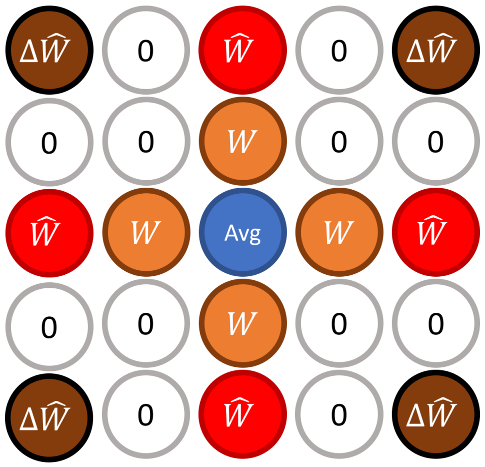


Figure 6.6: An intersection contains traffic flow information where blue element represents the average traffic flow of all lanes at the intersection, orange elements represents traffic flows in different lanes (figures in elements only indicate different traffic lanes rather than actual HOL values), and red and brown elements represent differences of traffic flows from neighboring intersections.

flow dynamics of a traffic network. All the information that the agent needs to know is encoded in the map shown in Fig. 6.7

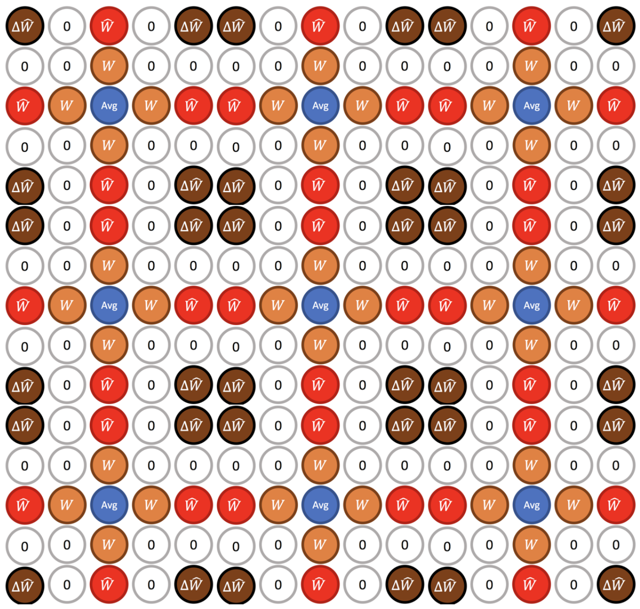
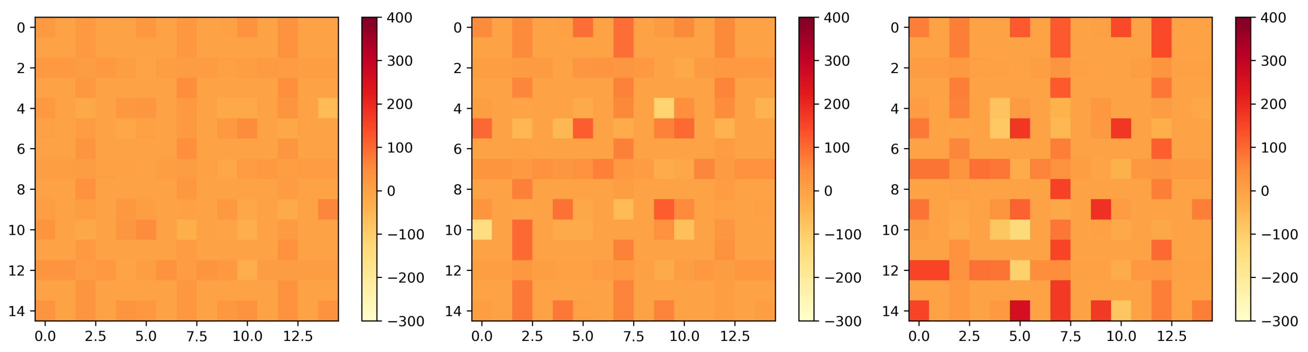


Figure 6.7: An traffic flow map of one traffic network composed of nine intersections

As shown in Fig. 6.8, there are TFMs for the 9 intersections at time slot 455, 1155, and 1755, respectively. Note each TFM consists of 9 tiles each of which is a 5×5 matrix of data values shown in Fig. 6.6. These TFMs visualize the delay information (the HOL sojourn times and the delay metrics) of the traffic flow encoded by varying intensities of color in the network as it evolves over



(a) $t = 455$ **(b)** $t = 1155$ **(c)** $t = 1755$

Figure 6.8: TFM for the 9 intersections as it evolves over time

time. The intensity shows that traffic congestion has become worse over time. As can be observed, they certainly capture flow dynamics of a traffic network.

6.5.2.2 Action Space

For each time slot t, the BP controllers at local intersections select a traffic signal phase from either North-South or East-West directions and form a vector of chosen phases $p_t = \{0, 1\}^M$ where M is the number of intersections and $0, 1$ denotes which direction should be activated, 0 for the East-West direction, 1 for the North-South direction. Based on the optimal modifications learned from the TFMs, the DRL agent improves phases in p_t to generate the action $a = \{0, 1\}^M$.

6.5.2.3 Reward Function

The throughput and average end-to-end delay of the network are two important metrics which indicate the level of traffic congestion in transportation. A higher throughput denotes more vehicles in the network have been served. A lower average end-to-end delay means vehicles in the network do not suffer from waiting. We define the power metric as our reward function given by

$$P = \frac{\lambda}{d} \tag{6.4}$$

where λ denotes the throughput of the network and d represents the average end-to-end delay of vehicles. Note that maximizing P implies that the network throughput is maximized and the average end-to-end delay is minimized. The power based reward function assigns a higher reward to state-action pairs that have high P value. The power metric has been used to design protocols in computer networks including [52]. Operating a packet switched network that maximizes power is referred to as Klienrock's optimal operating point as it achieves the maximum network throughput at the minimum end-to-end delay. Both the network throughput and average end-to-end delay are calculated based on the vehicles which have exited the network during the current time slot. For larger networks it may be more appropriate to calculate the network throughput and the average end-to-end delay over a number of time slots to account for the nominal delay of vehicles going through the network which would result in a delay between the action and response (hence the reward) from the environment. The detailed comparison of the power based reward and the other rewards can be found in [120].

6.5.2.4 Convolutional Neural Network

Research work has shown that RL agents can learn well from images by CNN [62, 71] in recent years. We implement CNN for the proposed TLC by specifying the number of layers and size of filters which meets our requirement of training the TLC agent. CNN consists of three convolutional layers and one fully-connected layer as the output in Fig. 6.9. There are 32 5 by 5 filters with 1 strides in the first convolutional layer. The output is a volume with the size of $15 \times 15 \times 32$ which is fed into the second convolutional layer. At the second layer, the number of filters is 64, each of which has the size of 4 by 4 and moves 1 stride each time. After the second layer, the output is a volume with the size of $15 \times 15 \times 64$. The third layer contains 64 filters with the size of 3×3, each of which moves 1 stride every time. This outputs a $15 \times 15 \times 64$ tensor. The final output, a fully-connected layer, transfers the tensor into a 512×1 matrix. Note that the other DRL agents we implement as the comparisons all share the same CNN as TLC.

6.5.2.5 The Proposed Deep Dueling SARSA Learning Method

TD learning methods combine nice properties from the Monte Carlo (MC) and dynamic programming (DP). First, TD methods learn state-action function $Q(s, a)$ directly from experience with TD errors and the bootstrapping. TD error denotes the difference between the current estimation and the actual observation of the state-action value. The bootstrapping substitutes the remaining trajectory with one or more estimated rewards in the update step. TD methods constantly track the current

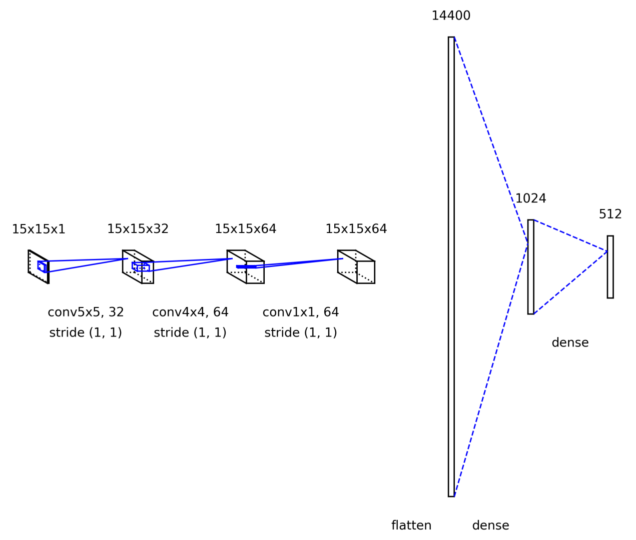


Figure 6.9: The convolutional neural network as the function approximator

TD error and propagate this TD error back in order to update the approximation for every time step. Second, TD methods are capable of learning from incomplete episodes without knowing the dynamic model of the environment. Third, TD methods are suitable for TSC problems because they perform well in continuous environment.

In contrast, Monte Carlo (MC) methods must wait for the final reward and update the state-action function by using a complete series of rewards, $Q(s_t, a_t) = Q(s_t, a_t) + \alpha \times [G_t - Q(s_t, a_t)]$ where $G_t = R_{t+1} + \gamma \times R_{t+2} + \cdots + \gamma^{T-1} \times R_T$ indicates the total discounted reward at time t. Unlike MC methods, TD methods do not necessarily need to wait until the end of an episode. Instead, they update Q-values based on estimated Q-values that have already been learned as given by Eq. 2.9.

SARSA which stands for the current state S, current action A, immediate reward R, next state S' and next action A' is one of TD learning methods. SARSA is also known as an on-policy method which evaluates or improves the behavioural policy, i.e., SARSA fits the action-value function to the current policy, and evaluates the policy based on samples from the same policy, then refines the policy greedily with respect to action values. On the other hand, off-policy methods, an agent learns an optimal value function/policy, maybe following an unrelated behavioural policy; for example, Q-learning attempts to find action values for the optimal policy directly, not necessarily fitting to the policy generating the data, i.e., the policy which Q-learning obtains is usually different from the policy that generates the samples. In short, on-policy can be understood as learning from same-policy. However, off-policy learns from different-policy.

Deep SARSA refers to the on-policy TD learning algorithm using CNN (mentioned in Section 6.5.2.4) as a function approximator. Similar to [72], a parameterized neural network replaces the conventional tabular approach for updating state-action pairs. DSARSA is incorporated into our proposed TLC. The input data of CNN are TFMs and the output is the action (modifications) with the optimal Q-value over all state-action pairs. θ is a network parameter set of CNN. During the training stage, the loss function is defined as follows:

$$L(\theta_t) = (y_t^{DSARSA} - Q^{\pi}(s_t, a_t; \theta_t))^2 \tag{6.5}$$

$$y_t^{DSARSA} = r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1}; \theta_t^{-}) \tag{6.6}$$

where s_t is the current state, a_t is the current action, s_{t+1} is the next state which is dependent on a_t taken by the agent in s_t, a_{t+1} is the next action selected by ϵ-greedy, and y_t^{DSARSA} is the TD target given by Eq. 6.6. The goal of CNN is to update the network parameter set θ that optimizes the loss function $L(\theta_t)$. First, the partial derivative of $L(\theta_t)$ is calculated by differentiating Eq. 6.5. The gradient of the loss function can be obtained as follows:

$$\bigtriangledown_{\theta_t} L(\theta_t) = (y_t^{DSARSA} - Q^{\pi}(s_t, a_t; \theta_t)) \bigtriangledown_{\theta_t} Q^{\pi}(s_t, a_t; \theta_t) \tag{6.7}$$

where $\bigtriangledown_{\theta_t} Q^{\pi}(s_t, a_t; \theta_t)$ denotes the gradient of the current state-action value. We optimize the loss function $L(\theta_t)$ by Eq. 6.7 and the network parameter set θ is updated by using stochastic gradient descent (SGD).

6.5.3 An Illustration of Training The DRL Based Centralized Controller

Fig. 6.10 illustrates how to train the centralized DRL agent for multi-intersection control. As mentioned, the agent is trained based on on-policy learning to learn how to make decisions based on the states and memory replay. TFMs as states of the traffic environment is fed to the CNN over time slots. Then, the current state, current action, observed reward, next state, and next action are

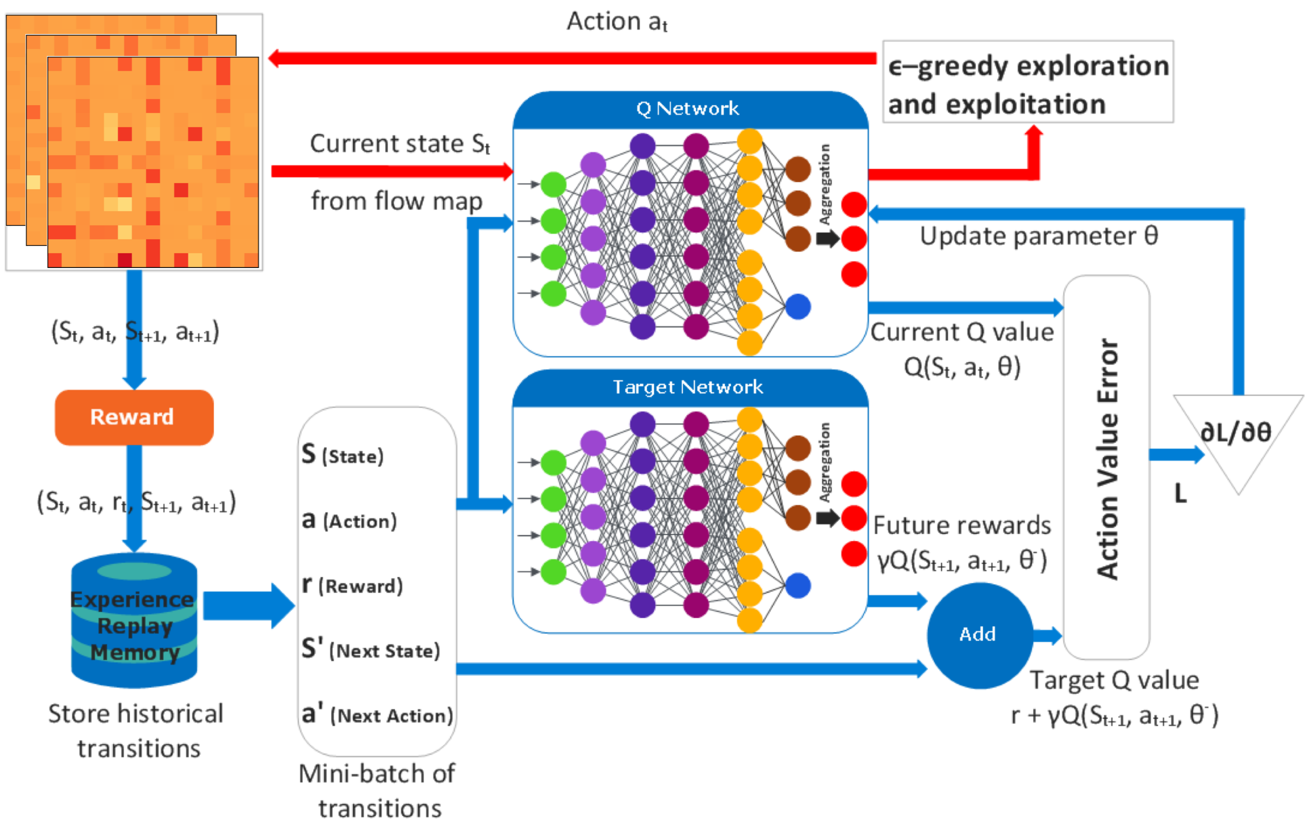

Action a_t
ϵ–greedy exploration and exploitation
Q Network
Current state S_t from flow map
Aggregation
Update parameter θ
Current Q value $Q(S_t, a_t, θ)$
Target Network
Aggregation
Future rewards $γQ(S_{t+1}, a_{t+1}, θ^-)$
Action Value Error
∂L/∂θ
L
$(S_t, a_t, S_{t+1}, a_{t+1})$
Reward
$(S_t, a_t, r_t, S_{t+1}, a_{t+1})$
Experience Replay Memory
Store historical transitions
S (State)
a (Action)
r (Reward)
S' (Next State)
a' (Next Action)
Mini-batch of transitions
Add
Target Q value $r + γQ(S_{t+1}, a_{t+1}, θ^-)$

Figure 6.10: An illustration of training the proposed TLC agent

pushed into the memory for reusing them later. The ϵ-greedy approach followed the function in Table 6.2 is utilized for the exploration and exploitation. There are two separated neural networks, one is a Q-network and the other is a target network, they share exactly the same feature and architecture. The update parameter θ will be copied over to the target network every τ steps. The current state and action are first fed into the Q-network for approximating a Q-value. Old experiences extracted from the replay memory buffer are used to be inputs to the target network for calculating a target Q-value. The difference, L, between the Q-value and target Q-value will be partially differentiated to get the derivative in which the agent will have a sense and know how to update the Q-network by fine-tuning the parameter θ.

As performance comparisons with the proposed TLC, we implement 4 existing DRL agents, namely, DQN, 3DQN, DSARSA, and 2DSARSA, which derive from the illustration with respect to the given RL-based algorithm, neural network, and network architecture as shown in Table 6.1.

Table 6.1: Properties of Different DRL Agents

Methods	TD Learning	Neural Network	Network Architecture	Work
DQN	Q-learning (off-policy)	CNN	Conventional	Mnih *et al.* [72]
3DQN	Q-learning (off-policy)	CNN	Dueling	Liang *et al.* [58]
DSARSA	SARSA (on-policy)	CNN	Conventional	Zhao *et al.* [128]
2DSARSA	SARSA (on-policy)	CNN	Dueling	Yen *et al.* [120]
TLC	SARSA (on-policy)	CNN	Dueling	The Proposed Architecture

6.6 Experimental Results

During the training stage, we used 1-hour traffic arrival data as one episode generated by following the Poisson process. All agents were trained by 1000 episodes of traffic arrival data and pre-trained using the first 100 episodes to collect replay data. For test stage, 10-hour long traffic arrival data which follow the Poisson arrivals as well were used as the test data to verify the robustness of each agent after completing the training stage. The difficulty is the training stage because each DRL agent requires 24 to 27 hours to complete 1000 episodes; hence, it takes nearly one week to finalize all the training stages. The reward, average end-to-end delay (sojourn time), average queue length, maximum delay, and fairness are collected as performance indicators to evaluate all DRL agents after running test data.

6.6.1 Baseline Scheduling Algorithm

Based on the work [93] for the throughput optimality of a queuing network with random arrival rates, Ji *et al.* [47] further considered a queue difference between any two adjacent nodes at each time slot and then determined a feasible schedule by activating the phase with the highest pressure (maximum queue length). In their algorithm, the controller at each intersection independently determines which phase of the traffic movement is scheduled based on the queue length and differences from neighboring nodes in every time slot. Their results showed that their algorithm can achieve maximum network throughput and global optimality without any prior knowledge about arrival rates.

$$\vec{p}^{*}(t) \in \underset{\vec{p} \in \mathcal{S}_{\mathcal{P}}}{\operatorname{argmax}} \sum_{P_{i,j}=1} \gamma_{i,j} \cdot Q_{i,j}(t) \cdot \mu_{i,j}(\vec{p}) \tag{6.8}$$

Later, this concept was first applied to traffic signal control in [116]. However, the selection of phases does not necessarily depend on only the queue length. The metrics such as the HOL sojourn time, sum-of-delay, and hybrid which combines the queue and HOL delay can be used as the measurement. Based on their work, we applied the delay-based BP scheme using the HOL sojourn time to the network as follows:

$$\vec{p}^*(t) \in \underset{\vec{p} \in \mathcal{S}_{\mathcal{P}}}{\operatorname{argmax}} \sum_{P_{i,j}=1} \gamma_{i,j} \cdot W_{i,j}(t) \cdot \mu_{i,j}(\vec{p}) \tag{6.9}$$

This formula calculates the sum of HOL vehicle sojourn time multiplied by two weighted values $\gamma_{i,j}$ and $\mu_{i,j}(\vec{p})$. As before, for vehicles from lane i to lane j, $(i, j) \in \mathcal{E}$, $\gamma_{i,j}$ is a positive constant that can put more emphasis on certain movements and $\mu_{i,j}(\vec{p})$ denotes a traffic flow of vehicles that can be transferred through the connected edge when phase $\vec{p}$ is activated. The scheme will return a phase that maximizes the total pressure released.

The proposed TLC architecture is applied to the TSC problem in order to learn from TFMs of the entire traffic network and then, determine which phases are supposed to be preferentially activated for the entire network with 9 intersections. In the experiment section, the performance of BP scheduling schemes as baselines are compared to the proposed TLC.

6.6.2 Hyperparameters

As listed in Table 6.2, a set of hyperparameters used in the proposed TLC architecture is pre-defined. Traffic arrival ratio for each intersection is $\vec{\lambda} = [0.2, 0.5, 0.2, 1] \times 0.125 \times 1.5$ v/s/l (vehicles per second per lane) for generating 1-hour and 10-hour datasets for training and testing, respectively. The ϵ-greedy function determines the probability of doing exploration or exploitation where ϵ_i as the initializing value is 1.0, $\epsilon_f = 0.01$ is the finalizing value, and the decay ratio is 300. Learning ratio α determines the weight of newly learned knowledge. α approaching 0 makes the agent exploit prior knowledge instead of learning something new while α approaching 1 makes it explore new potential rather than using prior knowledge. α is set to 0.0001 in the simulations. Discount factor γ determines the importance of future rewards. γ closer to 0 makes the agent behave myopically by only considering immediate rewards while γ closer to 1 makes it seek a long-term outcome by weighting future rewards with a higher value. γ is set to 0.99 in the simulations. The replay memory size M used to store previous experiences is set to 30000, and mini-batch size B used to sample experiences from the memory is set to 1024. The number of training episode $E = 1000$ is selected by trial and error. Note that DRL agents depending on different RL-based algorithms share the same hyperparameters.

Table 6.2: Summary of hyperparameters in the proposed TLC architecture

Parameter	Description
Traffic arrival ratio:	$\vec{\lambda} = [0.2, 0.5, 0.2, 1] \times 0.125 \times 1.5$ v/s/l
ϵ-greedy decay function:	calculate the probability of exploration and exploitation where
$\epsilon_f + (\epsilon_i - \epsilon_f) \times e^{-\frac{episode}{decayratio}}$	decay ratio is 300
Initializing $\epsilon_i = 1.0$	the beginning value of ϵ
Finalizing $\epsilon_f = 0.01$	the final value of ϵ
Learning ratio $\alpha = 0.0001$	learning ratio for updating the neural network
Discount ratio $\gamma = 0.99$	discount for future rewards
Memory size $M = 30000$	the size of entire replay memory buffer
Mini-batch size $B = 1024$	the size of samples that will be reused
Maximum number of episodes $E = 1000$	the total number of training episodes

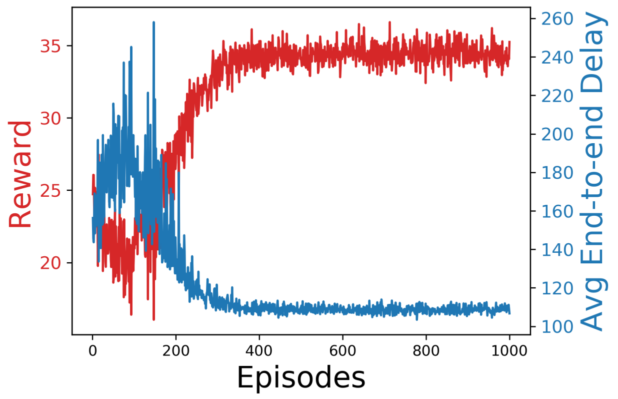
6.6.3 Learning Performance of TLC

In this chapter, we mentioned that the proposed TLC architecture has certain flexibility by adopting DRL with the dueling architecture, memory replay as well as other existing techniques, and high extendability by incorporating a centralized DRL agent with decentralized BP controllers. We elaborate on the learning performance of TLC regarding the four performance metrics: the average queue length, average end-to-end delay, maximum delay, and fairness. As shown in Fig. 6.11, the trend of reward (red curve) kept raising all the way after the pre-training stage and the reward was getting higher and higher over episodes. In Fig. 6.11a, and 6.11b, the average queue length (green curve) and average end-to-end delay (blue curve) were being minimized when the learning curve started to become stable at the 300-th episode. These two metrics reveals less vehicles were waiting in the lanes and experienced long delay. Namely, the TLC agent learns to select the optimal modifications with the highest P value to improve the phases chosen by BP controllers.

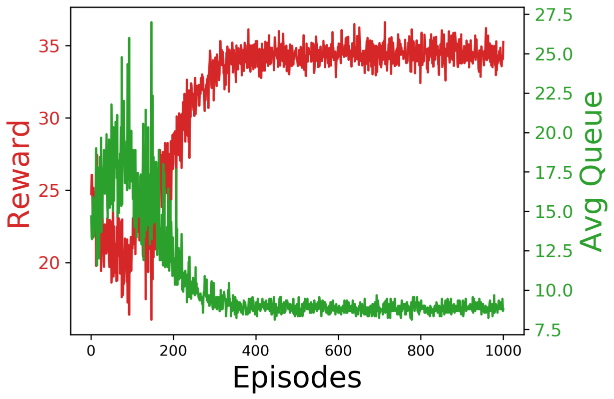
However, the average end-to-end delay averaging out all sojourn times of all vehicles does not reflect individual sojourn times for vehicles; consequently, it is difficult to evaluate whether individual vehicles are fairly scheduled. Therefore, we also consider a fairness metric in addition to the average end-to-end delay. We use Jain's Fairness Index [45] which is denoted by f and is given by

$$f(\vec{d} = [d_1, d_2, \ldots, d_M]) = \frac{(\sum_{i=1}^{M} d_i)^2}{M \sum_{i=1}^{M} (d_i)^2} \tag{6.10}$$

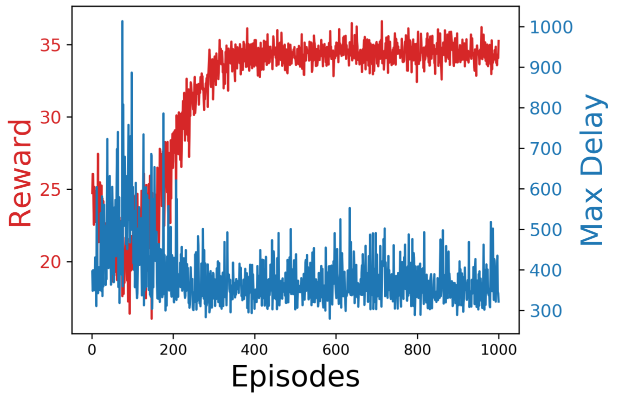
where d_i indicates the delay of vehicle i and M represents the total number of vehicles. The main idea of Jain Fairness Index is to compare the total delay to the delays of the individual vehicles. As the denominator becomes smaller, which means the delay of individual vehicle becomes smaller, f will increase. The system achieves a better fairness if Jain Fairness Index is higher. In Fig. 6.11d, the fairness (gray curve) was maximized and stabilized without many high fluctuations after the 200-th episode. This shows that the modifications to the chosen phases made by the TLC agent did not compromise the fairness and force vehicles undergo a very long sojourn time.



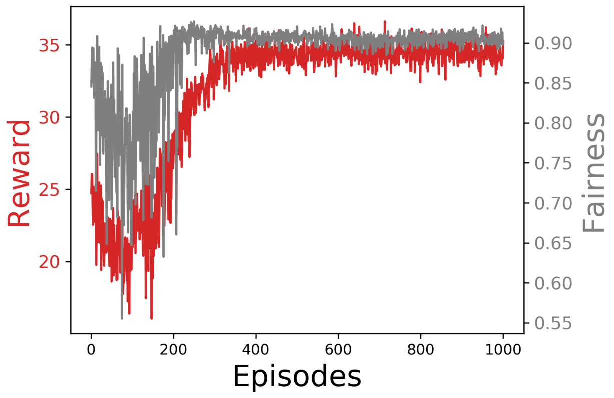
(a) Average delay and reward



(b) Average queue length and reward



(c) Maximum delay and reward



(d) Fairness and reward

Figure 6.11: The trends of delay, queue length, fairness, and reward when training a DRL agent using the proposed TLC architecture

6.6.4 Performance Comparison with Baseline BP Algorithms

Queue-based BP control (QBPC) serves the lane with the highest queue length, but lanes with lower arrivals will get starving. Delay-based BP control (DBPC) was proposed for solving the last vehicle problem in [47, 117] where vehicles are scheduled more fairly by DBPC. Hence, DBPC achieves a better fairness than QBPC in our previous work [117, 121, 122] and the tail of the delay distribution of DBPC (red tail) is shorter than that of QBPC (blue tail), which means less vehicles will suffer from a longer delay in Fig. 6.12. In this section, their performance is the baseline compared to the 2DSARSA and TLC agents. The learning performance of the 2DSARSA agent was discussed in our previous work [120].

Based on the successful training, the model of the neural network was stored and used to run 10-hour traffic arrival data. The performance of the TLC agent in terms of the delay distribution is shown in Fig. 6.12. The delay distribution indicates a probability of experiencing a specific delay. For example, 80% chance of experiencing 100s delay for DBPC, 40% chance of experiencing 100s delay for TLC, and 20% chance of experiencing 100s delay for QBPC. As can be observed, the TLC agent achieves a even shorter tail for outliers and better fairness for all vehicles than QBPC, DBPC, and 2DSARSA. The probability of a extremely long delay (the purple tail) for TLC is much lower than QBPC, DBPC, and 2DSARSA. QBPC and DBPC have probabilities of experiencing delays over 500s and 400s, respectively, whereas the probability of experiencing a delay over 400s is 0 for TLC. The curve of TLC lying between QBPC and DBPC denoting that it has a comparatively lower probability for vehicles to have the same sojourn time as they do in QBPC and DBPC. TLC starts to outperform QBPC when delay exceeds 130 sec.

It is quite obvious that the TLC agent outperforms 2DSARSA and the conventional BP algorithms proposed in [47] and [117] for multiple intersections after completing 1000 episodes. Regarding the overall fairness, Fig. 6.13 shows how many vehicles (circles) which experienced a long delay in the traffic network. The numbers of vehicles suffering a delay over 400s for QBPC, DBPC, and 2DSARSA are 6, 5, and 3, respectively. There is no vehicle suffering a delay over 400s for TLC. On the other hand, the numbers of vehicles suffering a delay between 300s and 400s for QBPC, DBPC, and 2DSARSA are 94, 83, and 55, respectively. There are only 46 vehicles suffering a delay between 300s and 400s for TLC.

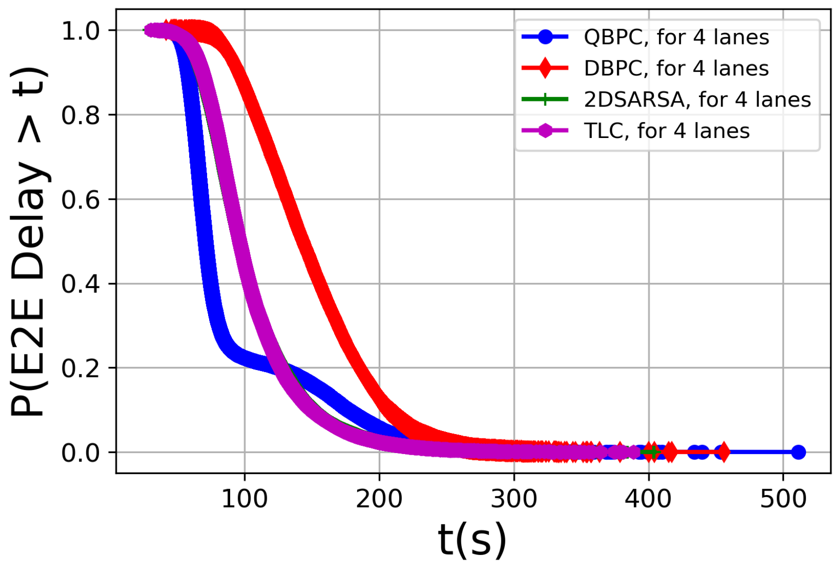


Figure 6.12: Comparison of delay distributions for queue-based BP control (QBPC), delay-based BP control (DBPC), 2DSARSA, and TLC

6.6.5 Performance Comparison with Existing Work

The learning performance of all DRL agents is shown in Fig. 6.14. DQN and 3DQN agents cannot effectively learn from the environment. The learning performance of DQN and 3DQN unveils that Q-learning with deep learning and advanced DRL techniques are not appropriate for a complicated environment. The fundamental difference between on-policy and off-policy methods is that off-policy learns by searching the best action from several policies. DQN and 3DQN utilize an off-policy learning method, Q-learning, which switches among multiple policies such that maximizes the expected accumulated reward. For many Atari games applying Q-learning, this approach easily get rid of a poorly initial policy and jump to the promising one which returns a better accumulated reward eventually. However, as the state dimension and action space grow exponentially, this approach reveals its weakness while dealing with a sophisticated environment. Unlike the conventional tabular Q-learning, for unlimited (possibility) number of states, it is impossible to visit all state-action pairs; hence, the action selection would be difficult for a Q-learning based agent since it cannot figure out which action is the "best" option to take without going through each state-action pair. We conclude that DQN and 3DQN both adopting off-policy learning are not suitable for multi-intersection control because the environment considered in this paper is a traffic network which is far more sophisticated than one intersection. In a traffic network, the learning performance of using DQN or 3DQN becomes

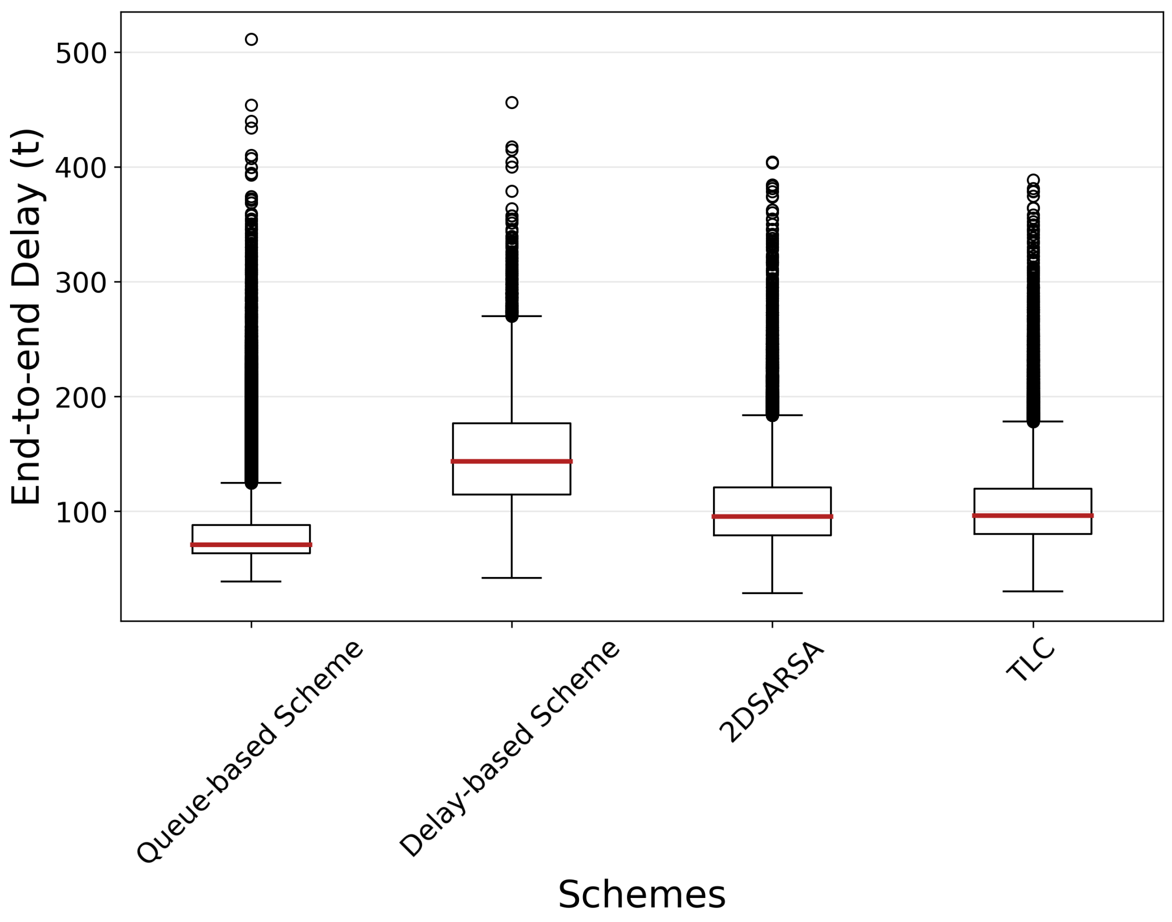


Figure 6.13: Comparison of box plots for queue-based BP control (QBPC), delay-based BP control (DBPC), 2DSARSA, and TLC

unstable and difficult to converge. Both of their trends (reward, average end-to-end delay, and average queue length) gradually became very fluctuating when getting close to the end of the training.

On the other hand, DSARSA, 2DSARSA, and TLC are capable of learning effectively from the environment during training. Without greedily searching the best policy, SARSA with deep learning learns by following the initial policy and selecting the next action within this policy. Their training processes worked perfectly and were converged at the end. On-policy learning with deep learning shows a certain level of stability while learning from a complicated environment. Because on-policy learns by following the same policy while learning, the whole training process will still be converged even if the policy chosen by SARSA is initially not good enough. The dueling technique, which separates the state-value and action advantage, can effectively filter out useless actions and force the entire training become converged eventually. 2DSARSA and TLC equipped with the dueling

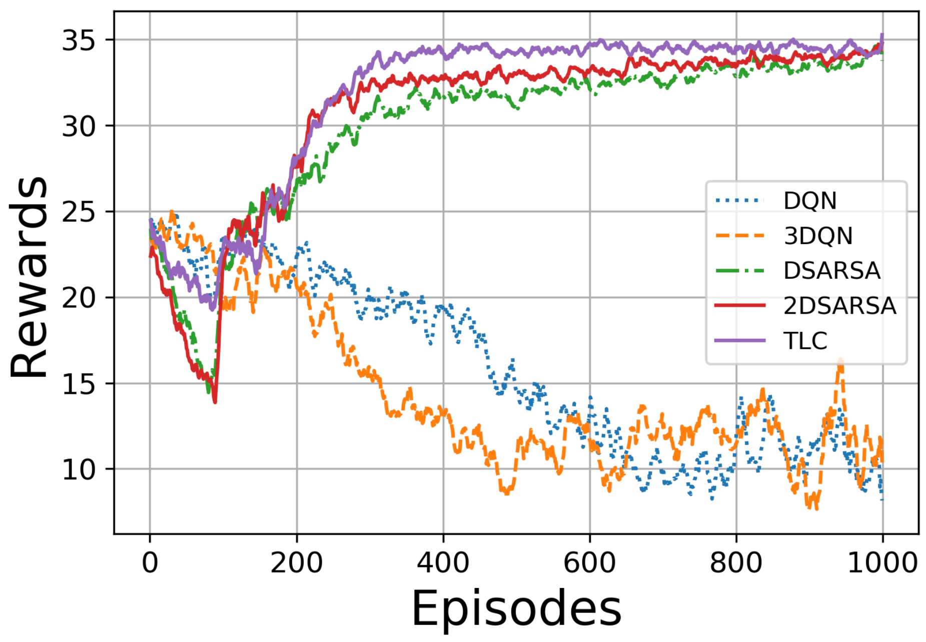


Figure 6.14: Comparison of learning performance among DQN, 3DQN, DSARSA, 2DSARSA, and TLC

architecture can significantly alleviate fluctuations during the training process and achieve more optimality than baseline BP schemes, DQN, 3DQN, and DSARSA.

Moreover, TLC enables the collaboration between a DRL agent and a number of BP controllers where the agent optimizes specific metrics and BP controllers help to select phases that maximizes these metrics. On the top of the chosen phases by BP controllers using POL, the DRL agent modify them to co-determine a series of joint action (modified phases) that maximizes the expected accumulated reward for the network of intersections. In Fig. 6.14, the comparison of learning performance reveals that the TLC agent (purple curve) obviously outperforms the 2DSARSA agent and the other agents. The training process of TLC converged faster than the other DRL agents. Our analytics shows that the learning performance of the TLC agent achieves the highest level and converges by only requiring 300 episodes compared to that of the 2DSARSA agent which needs to complete 1000 episodes to reach the same level as the TLC agent.

In a nutshell, TLC organizing a centralized DRL agent and a number of decentralized BP controllers in the same hierarchical architecture, outperforms other methods and agents. The collaboration between the centralized DRL agent and decentralized BP controllers indeed stabilizes the throughput optimality of the traffic network. Based on our experiments, purely applying

DRL with global information to a traffic network does not achieve the global optimality. Further optimization can still be achieved by enabling the collaboration between the two levels using both POL and TFM.

6.7 Conclusion

The experiments have shown a promising prospective of using TLC for multi-intersection traffic light control. Due to several success examples in learning Atari games by the deep Q-learning [71], most researchers who have contributed very much on the TSC problem only concentrated on using off-policy learning methods with deep learning. The DQN and 3DQN would work fine in an environment such as an isolated intersection that is easy to understand. But, when it comes to a complicated environment like a grid network, DQN or 3DQN are no longer enough to solve the problem. The proposed TLC performs well even if the environment is far more complex than one intersection. Our results show that the TLC agent incorporating a centralized agent and decentralized controllers is able to co-determine traffic signal phases based on both POL and TFMs of the traffic network, and even achieves more optimality and fairness than BP-based algorithms, 2DSARSA, and the other DRL agents.

As our future work, first, more complicated environment including extending the size of the grid network from 3 by 3 to 5 by 5 and increasing the number of traffic lanes will be taken into account. More phases such as turning left or right will be considered as well. Second, a power metric over a fairness ($\frac{\frac{\lambda}{d}}{1-f}$) will be considered as our new reward function to enhance the agent's learning performance where λ denotes the throughput of the network, d represents the average delay of vehicles, and f is the Jain's fairness index [45] of all vehicles in the network. Third, our experiments already showed that the dueling architecture indeed contributes to select valuable actions as the action space grows larger. However, the number of actions has to increase accordingly so as to achieve a certain level of freedom if the size of the traffic network is extended to 5 by 5 or even larger. Instead of the dueling architecture, what else should be included for enhancing the ability of recognizing valuable actions? For problems with higher dimensional action spaces, a multi-branch architecture which is more suitable for the estimation and action selection needs to be considered while the action space is getting huge.

In addition, at present, the depth of the input data is 1, more useful information such as queue length or sum of delay will be involved to formulate a plentiful data volume which would provide more comprehensive knowledge for the agent. Last but not least, the simulations of above improvements will be done by using SUMO for the future work.

Chapter 7

Future Work

7.1 The Impacts of Cyber-attacks on RL-based TSC

Two potential security vulnerabilities, the time spoofing and ghost vehicle attacks, which have been examined their impacts on BP-based TSC and proved that they threat modern TSC systems in our preliminary work. On the other hand, in terms of advanced intelligent traffic controller, we proposed the TLC architecture which leverages the global information as well as local information to determine traffic signal phases. The TLC agent with the global information at the higher level operates at a fine-grained time scale. The performance of the proposed architecture has been analyzed and showed a significant improvement of traffic delay achieved by deep RL-based method. However, to the best of our knowledge, at present, there is no investigation on potential threats to RL-based TSC. We will study the impact of malicious attacks on RL-based TSC schemes and observe whether an intelligent RL-based agent can be applied to TSC for mitigating impacts of cyber-attacks. Our goal is to develop a theoretical model based on RL which are capable of suppressing diffusion of malicious attacks, recovering or even surviving from a serious cyber-attack.

Challenge 1: Representation of A Contaminated Network: Modeling diffusion of malicious attacks over a traffic network composed of multiple intersections is challenging. Extracting knowledge of spreads of attacks from a traffic network is essential. We would formulate diffusion flow maps (DFM) into which all spreads of malicious attacks could be comprehensively involved. Misbehaved attacks causing serious traffic jams could be collected by roadside units. The estimates of spreads could be calculated, and then DFM would be synthesized by these estimates.

Challenge 2: Understanding of Spreading Misinformation: Although we can capture spreads and generate insightful information, the following questions still need to be examined. How well the DRL-based agent can understand the data we provide? Would the DRL-based agent learn from it? Furthermore, would the learning process converge eventually if DFMs were considered?

Challenge 3: RL-based Model for Cyber-attacks: How to come up with a RL-based model that is able to learn from misbehaved attacks, and then properly adjust scheduling for mitigating the impacts or even recovering from these attacks? In this stage, existing neural architectures, RL-based techniques will be examined to figure out their abilities of withstanding attacks. We will propose state-of-the-art framework consisting the novel architecture and RL-based model that is able to handle malicious attacks.

7.2 The Convergence of RL with High Dimensional Spaces

As the traffic demand grows rapidly as well as more and more intersections need to be considered into the TSC system nowadays, RL-based methods suffer from the curse of dimensionality and large action space issue. Recently, a plethora of papers [58, 71, 102, 115] have studied on high dimensional state spaces and well established solutions by utilizing non-linear approximators such as deep neural networks. However, the utilization of non-linear function approximators first does not guarantee the convergence of the learning process, especially, for high dimensional observation spaces, and second it cannot handle high dimensional action spaces very well. Those issues lead to a convergence problem at which we can make more efforts for the future work. The following papers [24, 59, 94] focused on the issue of high dimensional action spaces and proposed solutions to it. The potential research directions at which we can make efforts are two. One is exploration and exploitation and the other is neural network architecture.

Our goal is to design a cutting-edge control architecture which can offload some burdens from the DRL-based agent to local controllers such that decrease the dimension of actions. There are two levels in the proposed architecture. The DRL-based agent as a higher level policy maker operates at a coarse time scale. On the other hand, in a lower level, local controllers work at individual intersections at a fine-grained time scale. The DRL-based agent selects actions which are going to influence decisions determined by local controllers and forces them gradually alter their decisions towards the global policy made by the agent. Each action a consists of a set of control parameters where $a \in \mathbb{R}^{18}$. By cooperating with local controllers, we expect to lower down the dimension of actions for the DRL-based agent by separating the control into two levels. The proposed novel control architecture will be integrated into our existing framework.

Challenge: Action space grows exponentially based on the number of joint intersections and phases. For example, a $4-$phase of signal control system with the coarsest discretization $a \in \{1, 2, 3, 4\}$ for a 3 by 3 traffic network consisting of 9 intersections leads to an action space with dimensionality: $4^9 = 262144$.

7.2.1 Exploration Efficiency in Large Action Space

The simplest exploration and exploitation method, $\epsilon-$greedy, selects action based on a single parameter. This dithering strategy that switches between the best action and random selected actions would fail to deeply explore actions that return a long-term reward. Existing methods like Boltzmann exploration and Thomson sampling significantly outperform $\epsilon-$greedy.

Osband *et al.* [75] proposed a bootstrapped DQN for deep explorations via randomized value functions sampled from an approximate posterior. K different datasets $\tilde{D}$ are initially sampled from original D for K bootstrapped heads, each of which is trained by its bootstrapped sub-sample of the data. They are independent from each other but all share the same network. Randomized value functions have been recently proved that they work similar to Thompson sampling without an intractable exact posterior update. Their bootstrapped DQN achieves computationally efficiency and faster learning in a range of Atari 2600 games by training an ensemble of Q-functions.

7.2.2 Multi-branch for Large Action Space

The dueling architecture [110] mentioned in chapter 2.5.3 can learn states with a higher value without taking every action into account. Their idea is to separate the estimates of state-value and action advantage. Inspired by it, Tavakoli *et al.* [94] proposed a multi-branch architecture for problems with high dimensional action spaces. For a complicated environment, the number of actions has to increase accordingly so as to achieve a certain level of freedom. This proposed architecture divides an action space into several dimensions and implements branches sharing features from the same neural network for these individual action dimensions. Compared to the bootstrapped method [75], their architecture uses the same dataset instead of sampling different datasets for each branches, performing well even when the number of action grows linearly.

Bibliography

[1] Fast facts: U.s. transportation sector greenhouse gas emissions 1990-2017. https://www.epa.gov/greenvehicles/fast-facts-transportation-greenhouse-gas-emissions.

[2] Un estimates: 68% of the world population projected to live in urban areas by 2050. https://www.un.org/development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html, 16 May 2018.

[3] Monireh Abdoos, Nasser Mozayani, and Ana LC Bazzan. Traffic light control in non-stationary environments based on multi agent q-learning. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC 2011)*, pages 1580–1585. IEEE, 2011.

[4] Monireh Abdoos, Nasser Mozayani, and Ana LC Bazzan. Holonic multi-agent system for traffic signals control. *Engineering Applications of Artificial Intelligence*, 26(5-6):1575–1587, 2013.

[5] Assad Al Alam, Ather Gattami, and Karl Henrik Johansson. An experimental study on the fuel reduction potential of heavy duty vehicle platooning. In *13th international IEEE conference on intelligent transportation systems*, pages 306–311. IEEE, 2010.

[6] Mani Amoozadeh, Hui Deng, Chen-Nee Chuah, H Michael Zhang, and Dipak Ghosal. Platoon management with cooperative adaptive cruise control enabled by vanet. *Vehicular communications*, 2(2):110–123, 2015.

[7] Sahar Araghi, Abbas Khosravi, Michael Johnstone, and Doug Creighton. Q-learning method for controlling traffic signal phase time in a single intersection. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 1261–1265. IEEE, 2013.

[8] Itamar Arel, Cong Liu, Tom Urbanik, and Airton G Kohls. Reinforcement learning-based multi-agent system for network traffic signal control. *IET Intelligent Transport Systems*, 4(2):128–135, 2010.

[9] Lawrence M Ausubel, Paul Milgrom, et al. The lovely but lonely vickrey auction. *Combinatorial auctions*, 17:22–26, 2006.

[10] Radhakisan Baheti and Helen Gill. Cyber-physical systems. *The impact of control technology*, 12(1):161–166, 2011.

[11] Carolina Tripp Barba, Miguel Angel Mateos, Pablo Reganas Soto, Ahmad Mohamad Mezher, and Mónica Aguilar Igartua. Smart city for vanets using warning messages, traffic statistics and intelligent traffic lights. In *2012 IEEE Intelligent Vehicles Symposium*, pages 902–907. IEEE, 2012.

[12] Margaret C Bell and RD Bretherton. Ageing of fixed-time traffic signal plans. In *International Conference on Road Traffic Control*, 1986.

[13] Alexandre Bovet and Hernán A Makse. Influence of fake news in twitter during the 2016 us presidential election. *Nature Communications*, 10(1):1–14, 2019.

[14] Fred Browand, John McArthur, and Charles Radovich. Fuel saving achieved in the field test of two tandem trucks. 2004.

[15] Martin Buechel and Alois Knoll. Deep reinforcement learning for predictive longitudinal control of automated vehicles. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2391–2397. IEEE, 2018.

[16] Chen Cai, Chi Kwong Wong, and Benjamin G Heydecker. Adaptive traffic signal control using approximate dynamic programming. *Transportation Research Part C: Emerging Technologies*, 17(5):456–474, 2009.

[17] Dustin Carlino, Stephen D Boyles, and Peter Stone. Auction-based autonomous intersection management. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 529–534. IEEE, 2013.

[18] Cesar Cerrudo. An emerging us (and world) threat: Cities wide open to cyber attacks. *Securing Smart Cities*, 17:137–151, 2015.

[19] Chen Chen, Yuru Zhang, Mohammad R Khosravi, Qingqi Pei, and Shaohua Wan. An intelligent platooning algorithm for sustainable transportation systems in smart cities. *IEEE Sensors Journal*, 2020.

[20] Qi Alfred Chen, Yucheng Yin, Yiheng Feng, Z Morley Mao, and Henry X Liu. Exposing congestion attack on emerging connected vehicle based traffic signal control. In *Network and Distributed Systems Security (NDSS) Symposium*, 2018.

[21] Tianshu Chu and Uroš Kalabić. Model-based deep reinforcement learning for cacc in mixed-autonomy vehicle platoon. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 4079–4084. IEEE, 2019.

[22] G. Cookson. Inrix global traffic scorecard, 2018.

[23] Charles Desjardins and Brahim Chaib-Draa. Cooperative adaptive cruise control: A reinforcement learning approach. *IEEE Transactions on intelligent transportation systems*, 12(4):1248–1260, 2011.

[24] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

[25] Samah El-Tantawy, Baher Abdulhai, and Hossam Abdelgawad. Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (marlin-atsc): methodology and large-scale application on downtown toronto. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1140–1150, 2013.

[26] David Elliott, Walter Keen, and Lei Miao. Recent advances in connected and automated vehicles. *Journal of Traffic and Transportation Engineering (English Edition)*, 6(2):109–131, 2019.

[27] Yiheng Feng, K Larry Head, Shayan Khoshmagham, and Mehdi Zamanipour. A real-time adaptive signal control in a connected vehicle environment. *Transportation Research Part C: Emerging Technologies*, 55:460–473, 2015.

[28] Yiheng Feng, K Larry Head, Shayan Khoshmagham, and Mehdi Zamanipour. A real-time adaptive signal control in a connected vehicle environment. *Transportation Research Part C: Emerging Technologies*, 55:460–473, 2015.

[29] Yiheng Feng, Shihong Huang, Qi Alfred Chen, Henry X Liu, and Z Morley Mao. Vulnerability of traffic control system under cyberattacks with falsified data. *Transportation Research Record*, 2672(1):1–11, 2018.

[30] Pedro Fernandes and Urbano Nunes. Platooning of autonomous vehicles with intervehicle communications in sumo traffic simulator. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 1313–1318. IEEE, 2010.

[31] Emilio Ferrara, Onur Varol, Clayton Davis, Filippo Menczer, and Alessandro Flammini. The rise of social bots. *Communications of the ACM*, 59(7):96–104, 2016.

[32] Juntao Gao, Yulong Shen, Jia Liu, Minoru Ito, and Norio Shiratori. Adaptive traffic signal control: Deep reinforcement learning algorithm with experience replay and target network. *arXiv preprint arXiv:1705.02755*, 2017.

[33] Michael R Gary and David S Johnson. Computers and intractability: A guide to the theory of np-completeness. *Freeman, San Francisco*, 1979.

[34] Wade Genders and Saiedeh Razavi. Using a deep reinforcement learning agent for traffic signal control. *arXiv preprint arXiv:1611.01142*, 2016.

[35] Amin Ghafouri, Waseem Abbas, Yevgeniy Vorobeychik, and Xenofon Koutsoukos. Vulnerability of fixed-time control of signalized intersections to cyber-tampering. In *2016 Resilience Week (RWS)*, pages 130–135. IEEE, 2016.

[36] Branden Ghena, William Beyer, Allen Hillaker, Jonathan Pevarnek, and J Alex Halderman. Green lights forever: Analyzing the security of traffic infrastructure. In *8th {USENIX} Workshop on Offensive Technologies ({WOOT} 14)*, 2014.

[37] Xiaoyan Gong and Xiaoming Liu. A data mining based algorithm for traffic network flow forecasting. In *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems*, volume 1, pages 193–198. IEEE, 2003.

[38] S Ilgin Guler, Monica Menendez, and Linus Meier. Using connected vehicle technology to improve the efficiency of intersections. *Transportation Research Part C: Emerging Technologies*, 46:121–131, 2014.

[39] Levent Güvenç, Ismail Meriç Can Uygan, Kerim Kahraman, Raif Karaahmetoglu, Ilker Altay, Mutlu Sentürk, Mümin Tolga Emirler, Ahu Ece Hartavi Karci, Bilin Aksun Guvenc, Erdinç Altug, et al. Cooperative adaptive cruise control implementation of team mekar at the grand cooperative driving challenge. *IEEE Transactions on Intelligent Transportation Systems*, 13(3):1062–1074, 2012.

[40] Neal Harwood and N Reed. Modelling the impact of platooning on motorway capacity. In *Road Transport Information and Control Conference 2014 (RTIC 2014)*, pages 1–6. IET, 2014.

[41] Hamssa Hasrouny, Abed Ellatif Samhat, Carole Bassil, and Anis Laouiti. Vanet security challenges and solutions: A survey. *Vehicular Communications*, 7:7–20, 2017.

[42] Ammar Haydari and Yasin Yilmaz. Deep reinforcement learning for intelligent transportation systems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[43] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM (JACM)*, 21(2):277–292, 1974.

[44] Peter J Hotez. Texas and its measles epidemics. *PLoS medicine*, 13(10):e1002153, 2016.

[45] Rajendra K Jain, Dah-Ming W Chiu, William R Hawe, et al. A quantitative measure of fairness and discrimination. *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 1984.

[46] Tobias Jeske. Floating car data from smartphones: What google and waze know about you and how hackers can control traffic. *Proc. of the BlackHat Europe*, pages 1–12, 2013.

[47] Bo Ji, Changhee Joo, and Ness B Shroff. Delay-based back-pressure scheduling in multihop wireless networks. *IEEE/ACM Transactions on Networking*, 21(5):1539–1552, 2012.

[48] Junchen Jin and Xiaoliang Ma. Adaptive group-based signal control by reinforcement learning. *Transportation Research Procedia*, 10:207–216, 2015.

[49] Junchen Jin and Xiaoliang Ma. Adaptive group-based signal control using reinforcement learning with eligibility traces. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2412–2417. IEEE, 2015.

[50] Junchen Jin and Xiaoliang Ma. Hierarchical multi-agent control of traffic lights based on collective learning. *Engineering applications of artificial intelligence*, 68:236–248, 2018.

[51] Jin Junchen and Ma Xiaoliang. A learning-based adaptive signal control system with function approximation. *International Federation of Automatic Control-PapersOnLine*, 49(3):5–10, 2016.

[52] L. Kleinrock. Internet congestion control using the power metric: Keep the pipe just full, but no fuller. *Ad Hoc Netw.*, 80:142–157, 2018.

[53] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent development and applications of sumo-simulation of urban mobility. *International journal on advances in systems and measurements*, 5(3&4), 2012.

[54] Joyoung Lee and Byungkyu Park. Development and evaluation of a cooperative vehicle intersection control algorithm under the connected vehicles environment. *IEEE Transactions on Intelligent Transportation Systems*, 13(1):81–90, 2012.

[55] Li Li, Ding Wen, and Danya Yao. A survey of traffic control with vehicular communications. *IEEE Transactions on Intelligent Transportation Systems*, 15(1):425–432, 2013.

[56] Meng Li, Kanok Boriboonsomsin, Guoyuan Wu, Wei-Bin Zhang, and Matthew Barth. Traffic energy and emission reductions at signalized intersections: a study of the benefits of advanced driver information. *International Journal of Intelligent Transportation Systems Research*, 7(1):49–58, 2009.

[57] Zhaojian Li, Tianshu Chu, Ilya V Kolmanovsky, and Xiang Yin. Training drift counteraction optimal control policies using reinforcement learning: An adaptive cruise control example. *IEEE Transactions on Intelligent Transportation Systems*, 19(9):2903–2912, 2017.

[58] Xiaoyuan Liang, Xunsheng Du, Guiling Wang, and Zhu Han. A deep reinforcement learning network for traffic light cycle control. *IEEE Transactions on Vehicular Technology*, 68(2):1243–1253, 2019.

[59] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[60] Jennie Lioris, Ramtin Pedarsani, Fatma Yildiz Tascikaraoglu, and Pravin Varaiya. Doubling throughput in urban roads by platooning. *IFAC-PapersOnLine*, 49(3):49–54, 2016.

[61] Sikai Lu, Yingfeng Cai, Long Chen, Hai Wang, Xiaoqiang Sun, and Yunyi Jia. A sharing deep reinforcement learning method for efficient vehicle platooning control. *IET Intelligent Transport Systems*, 2021.

[62] Xiaolei Ma, Zhuang Dai, Zhengbing He, Jihui Ma, Yong Wang, and Yunpeng Wang. Learning traffic as images: a deep convolutional neural network for large-scale transportation network speed prediction. *Sensors*, 17(4):818, 2017.

[63] Silvano Martello and Paolo Toth. An upper bound for the zero-one knapsack problem and a branch and bound algorithm. *European Journal of Operational Research*, 1(3):169–175, 1977.

[64] Nitin Maslekar, Mounir Boussedjra, Joseph Mouzna, and Houda Labiod. Vanet based adaptive traffic signal control. In *2011 IEEE 73rd Vehicular Technology Conference (VTC Spring)*, pages 1–5. IEEE, 2011.

[65] Jesús Mena-Oreja and Javier Gozalvez. Permit-a sumo simulator for platooning maneuvers in mixed traffic scenarios. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3445–3450. IEEE, 2018.

[66] Jesús Mena-Oreja, Javier Gozalvez, and Miguel Sepulcre. Effect of the configuration of platooning maneuvers on the traffic flow under mixed traffic scenarios. In *2018 IEEE Vehicular Networking Conference (VNC)*, pages 1–4. IEEE, 2018.

[67] Yue Meng, Li Li, Fei-Yue Wang, Keqiang Li, and Zhiheng Li. Analysis of cooperative driving strategies for nonsignalized intersections. *IEEE Transactions on Vehicular Technology*, 67(4):2900–2911, 2017.

[68] Mark Michaelian and Fred Browand. Field experiments demonstrate fuel savings for close-following. 2000.

[69] Vicente Milanés, Steven E Shladover, John Spring, Christopher Nowakowski, Hiroshi Kawazoe, and Masahide Nakamura. Cooperative adaptive cruise control in real traffic situations. *IEEE Transactions on intelligent transportation systems*, 15(1):296–305, 2013.

[70] Alan J Miller. Settings for fixed-cycle traffic signals. *Journal of the Operational Research Society*, 14(4):373–386, 1963.

[71] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[72] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[73] Michael J Neely, Eytan Modiano, and Charles E Rohrs. Dynamic power allocation and routing for time varying wireless networks. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)*, volume 1, pages 745–755. IEEE, 2003.

[74] Michael J Neely and Rahul Urgaonkar. Optimal backpressure routing for wireless networks with multi-receiver diversity. *Ad Hoc Networks*, 7(5):862–881, 2009.

[75] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29:4026–4034, 2016.

[76] Kartik Pandit, Dipak Ghosal, H Michael Zhang, and Chen-Nee Chuah. Adaptive traffic signal control with vehicular ad hoc networks. *IEEE Transactions on Vehicular Technology*, 62(4):1459–1471, 2013.

[77] Simon Parkinson, Paul Ward, Kyle Wilson, and Jonathan Miller. Cyber threats facing autonomous and connected vehicles: Future challenges. *IEEE Transactions on Intelligent Transportation Systems*, 18(11):2898–2915, 2017.

[78] KJ Prabuchandran, Hemanth Kumar AN, and Shalabh Bhatnagar. Multi-agent reinforcement learning for traffic signal control. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC 2014)*, pages 2529–2534. IEEE, 2014.

[79] LA Prashanth and Shalabh Bhatnagar. Reinforcement learning with function approximation for traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 12(2):412–421, 2010.

[80] M. Prigg. Has new york's traffic light system been hacked? In *Daily Mail*. Daily Mail, May 2014.

[81] Maxim Raya and Jean-Pierre Hubaux. The security of vehicular ad hoc networks. In *Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks*, pages 11–21, 2005.

[82] H. Ritchie and M. Roser. Our world in data, urbanization. https://ourworldindata.org/urbanization.

[83] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. Citeseer, 1994.

[84] Fatih Sakiz and Sevil Sen. A survey of attacks and detection mechanisms on intelligent transportation systems: Vanets and iov. *Ad Hoc Networks*, 61:33–50, 2017.

[85] Stefania Santini, Alessandro Salvi, Antonio Saverio Valente, Antonio Pescapè, Michele Segata, and R Lo Cigno. A consensus-based approach for platooning with inter-vehicular communications. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 1158–1166. IEEE, 2015.

[86] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[87] Chengcheng Shao, Giovanni Luca Ciampaglia, Onur Varol, Kai-Cheng Yang, Alessandro Flammini, and Filippo Menczer. The spread of low-credibility content by social bots. *Nature Communications*, 9(1):1–9, 2018.

[88] Jieun Shin, Lian Jian, Kevin Driscoll, and François Bar. The diffusion of misinformation on social media: Temporal pattern, message, and source. *Computers in Human Behavior*, 83:278–287, 2018.

[89] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[90] Abhishek Sinha and Eytan Modiano. Optimal control for generalized network-flow problems. *IEEE/ACM Transactions on Networking*, 26(1):506–519, 2017.

[91] R. S Sutton and A. G Barto. *Reinforcement learning: An Introduction, 2nd edition.* MIT press, Oct. 2018.

[92] Alireza Talebpour and Hani S Mahmassani. Influence of connected and autonomous vehicles on traffic flow stability and throughput. *Transportation Research Part C: Emerging Technologies*, 71:143–163, 2016.

[93] Leandros Tassiulas and Anthony Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. In *29th IEEE Conference on Decision and Control*, pages 2130–2132. IEEE, 1990.

[94] Arash Tavakoli, Fabio Pardo, and Petar Kormushev. Action branching architectures for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[95] Luca Terruzzi, Riccardo Colombo, and Michele Segata. Poster: On the effects of cooperative platooning on traffic shock waves. In *2017 IEEE Vehicular Networking Conference (VNC)*, pages 37–38. IEEE, 2017.

[96] Thomas L Thorpe and Charles W Anderson. Traffic light control using sarsa with three state representations. Technical Report Technical Report, Armonk, NY, USA, 1996.

[97] Sadayuki Tsugawa, Shin Kato, and Keiji Aoki. An automated truck platoon for energy saving. In *2011 IEEE/RSJ international conference on intelligent robots and systems*, pages 4109–4114. IEEE, 2011.

[98] Elisabeth Uhlemann. Introducing connected vehicles [connected vehicles]. *IEEE vehicular technology magazine*, 10(1):23–31, 2015.

[99] Elisabeth Uhlemann. Introducing connected vehicles [connected vehicles]. *IEEE Vehicular Technology Magazine*, 10(1):23–31, 2015.

[100] Elise Van der Pol and Frans A Oliehoek. Coordinated deep reinforcement learners for traffic light control. *Proceedings of Learning, Inference and Control of Multi-Agent Systems (at NIPS 2016)*, 2016.

[101] Nees Jan van Eck and Michiel van Wezel. Application of reinforcement learning to the game of othello. *Computers & Operations Research*, 35(6):1999–2017, 2008.

[102] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[103] Chris J Vargo, Lei Guo, and Michelle A Amazeen. The agenda-setting power of fake news: A big data analysis of the online media landscape from 2014 to 2016. *New media & society*, 20(5):2028–2049, 2018.

[104] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[105] Xiaoqiang Wang, Liangjun Ke, Zhimin Qiao, and Xinghua Chai. Large-scale traffic signal control using a novel multiagent reinforcement learning. *IEEE transactions on cybernetics*, 51(1):174–187, 2020.

[106] Ziran Wang, Guoyuan Wu, and Matthew J Barth. A review on cooperative adaptive cruise control (cacc) systems: Architectures, controls, and applications. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2884–2891. IEEE, 2018.

[107] Ziran Wang, Guoyuan Wu, and Matthew J Barth. Cooperative eco-driving at signalized intersections in a partially connected and automated vehicle environment. *IEEE Transactions on Intelligent Transportation Systems*, 21(5):2029–2038, 2019.

[108] Ziran Wang, Guoyuan Wu, Peng Hao, and Matthew J Barth. Cluster-wise cooperative eco-approach and departure application for connected and automated vehicles along signalized arterials. *IEEE Transactions on Intelligent Vehicles*, 3(4):404–413, 2018.

[109] Ziran Wang, Guoyuan Wu, Peng Hao, Kanok Boriboonsomsin, and Matthew Barth. Developing a platoon-wide eco-cooperative adaptive cruise control (cacc) system. In *2017 ieee intelligent vehicles symposium (iv)*, pages 1256–1261. IEEE, 2017.

[110] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003. PMLR, 2016.

[111] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[112] F. V. Webster. *Traffic signal setting.* Number Technical Paper 39. Road Research Laboratory, HMSO, London, U.K., 1958.

[113] Hua Wei, Chacha Chen, Guanjie Zheng, Kan Wu, Vikash Gayah, Kai Xu, and Zhenhui Li. Presslight: Learning max pressure control to coordinate traffic signals in arterial network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1290–1298, 2019.

[114] Hua Wei, Nan Xu, Huichu Zhang, Guanjie Zheng, Xinshi Zang, Chacha Chen, Weinan Zhang, Yanmin Zhu, Kai Xu, and Zhenhui Li. Colight: Learning network-level cooperation for traffic signal control. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1913–1922, 2019.

[115] Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. Intellilight: A reinforcement learning approach for intelligent traffic light control. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2496–2505, 2018.

[116] Tichakorn Wongpiromsarn, Tawit Uthaicharoenpong, Yu Wang, Emilio Frazzoli, and Danwei Wang. Distributed traffic signal control for maximum network throughput. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pages 588–595. IEEE, 2012.

[117] Jian Wu, Dipak Ghosal, Michael Zhang, and Chen-Nee Chuah. Delay-based traffic signal control for throughput optimality and fairness at an isolated intersection. *IEEE Transactions on Vehicular Technology*, 67(2):896–909, 2017.

[118] Nan Xiao, Emilio Frazzoli, Yiwen Luo, Yitong Li, Yu Wang, and Danwei Wang. Throughput optimality of extended back-pressure traffic signal control algorithm. In *2015 23rd Mediterranean Conference on Control and Automation (MED)*, pages 1059–1064. IEEE, 2015.

[119] Hao Yang, Hesham Rakha, and Mani Venkat Ala. Eco-cooperative adaptive cruise control at signalized intersections considering queue effects. *IEEE Transactions on Intelligent Transportation Systems*, 18(6):1575–1585, 2016.

[120] Chia-Cheng Yen, Dipak Ghosal, Michael Zhang, and Chen-Nee Chuah. A deep on-policy learning agent for traffic signal control of multiple intersections. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6. IEEE, 2020.

[121] Chia-Cheng Yen, Dipak Ghosal, Michael Zhang, and Chen-Nee Chuah. Security vulnerabilities and protection algorithms for backpressure-based traffic signal control at an isolated intersection. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[122] Chia-Cheng Yen, Dipak Ghosal, Michael Zhang, Chen-Nee Chuah, and Hao Chen. Falsified data attack on backpressure-based traffic signal control algorithms. In *2018 IEEE Vehicular Networking Conference (VNC)*, pages 1–8. IEEE, 2018.

[123] Maram Bani Younes and Azzedine Boukerche. Intelligent traffic light controlling algorithms using vehicular networks. *IEEE Transactions on Vehicular Technology*, 65(8):5887–5899, 2015.

[124] Ali A Zaidi, Balázs Kulcsár, and Henk Wymeersch. Back-pressure traffic signal control with fixed and adaptive routing for urban vehicular networks. *IEEE Transactions on Intelligent Transportation Systems*, 17(8):2134–2143, 2016.

[125] Junping Zhang, Fei-Yue Wang, Kunfeng Wang, Wei-Hua Lin, Xin Xu, and Cheng Chen. Data-driven intelligent transportation systems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 12(4):1624–1639, 2011.

[126] Linlin Zhang, Feng Chen, Xiaoxiang Ma, and Xiaodong Pan. Fuel economy in truck platooning: a literature overview and directions for future research. *Journal of Advanced Transportation*, 2020, 2020.

[127] Dongbin Zhao, Yujie Dai, and Zhen Zhang. Computational intelligence in urban traffic signal control: A survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):485–494, 2011.

[128] Dongbin Zhao, Haitao Wang, Kun Shao, and Yuanheng Zhu. Deep reinforcement learning with experience replay based on sarsa. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–6. IEEE, 2016.

[129] Mofan Zhou, Yang Yu, and Xiaobo Qu. Development of an efficient driving strategy for connected and automated vehicles at signalized intersections: A reinforcement learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 21(1):433–443, 2019.