Deep Reinforcement Learning to Minimize Traffic Congestion with Emergency Facilitation

By

Dulmina Renuke Kodagoda

Index No : 15000712

This dissertation is submitted to the University of Colombo School of Computing
In partial fulfillment of the requirements for the
Degree of Bachelor of Science Honours in Computer Science

University of Colombo School of Computing
35, Reid Avenue, Colombo 07,
Sri Lanka
2019

Declaration

I, D.R.Kodagoda hereby certify that this dissertation entitled "Deep Reinforcement Learning to Minimize Traffic Congestion with Emergency Facilitation" is entirely my own work and it has never been submitted nor is currently been submitted for any other degree.

…….. ……..

Date Signature of Student

I, G.P.Senevirathne, certify that I supervised this dissertation entitled "Deep Reinforcement Learning to Minimize Traffic Congestion with Emergency Facilitation" conducted by D.R.Kodagoda in partial fulfillment of the requirements for the degree of Bachelor of Science Honours in Computer Science.

…….. ……..

Date Signature of Supervisor

Abstract

In the domain of intelligent traffic light control, which real-time traffic data to consider has a huge impact on the efficiency and performance of the traffic light control system. The rewards and state representations used in previous studies can mislead the agent in some cases.

This paper examines those problems and proposes a solution using the standard deviation of the vehicle waiting time. Existing studies have not yet provided emergency facilitation. This paper proposes a method that provides emergency facilitation.

The proposed method is self-evaluated with another version of the proposed method under both synthetic and real-world data, and it has proven that consideration of standard deviation has a significant impact on performance. The proposed method is also evaluated with a statistical method and a fixed time and has outperformed both of them. Buy considering vehicle type it was able to approximate the emergency vehicle waiting time to zero which was initialy at 20s when starting the training. With the help of standard deviation of waiting time, It was able to approximate the regular vehicle waiting time to 21s which was initialy at 60s when starting the training. The proposed method was able to record 21.588s of average waiting time of regular vehicles at the testing phase outperforming against methods.

Preface

This research is mainly focused on two novel aims. First one is to evaluate the impact of considering the standard deviation of vehicle waiting times when making a signal change. The concept of considering standard deviation was solely my own intuition and has not been proposed by any other researcher. The second novel idea is providing emergency facilitation while minimizing traffic congestion. As of my knowledge, no previous work has considered emergency facilitation.

Design and implementation in chapter 3 and 4 of the proposed method and other methods which are used to evaluate the proposed method are entirely my work except the statistical method used to evaluate the proposed method is proposed by a previous work.

The results of chapter five rely upon a simulation environment (SUMO) which is implemented in a previous work. The training and evaluation are carried out by myself with my supervisor and co-supervisor.

Acknowledgment

I want to start by expressing my sincere gratitude to my supervisor Mr. G.P.Senevithathne, senior lecturer of the University of Colombo School of Computing and my co-supervisor Mr. K.V.D.J.P.Kumarasinghe, lecturer of University of Colombo School of Computing for guiding me throughout the research and the academic writing. Without their support, this research will not be a success.

I would also like to thank all the senior lecturers, lecturers, Instructors of the University of Colombo School of Computing for giving me their feedback.

None of this would have been possible without my parents who are the two great pillars in my life. I am truly grateful for the moral support and the amazing chances they've given me over the years, and especially believing in me and supporting me constantly during this critical endeavor in my academic life. Both of you have inspired me to take challenges in life and it is this mere inspiration that stimulated me to successfully complete this research.

Last but not least, I would like to thank my best friends Wasura Wattearachchi and Dushani Perera for their feedback and suggestions, especially for double-checking all the documentation of this research.

List of Acronyms

RL: Reinforcement Learning

MDPs: Markov Decision Processes

AKA: Also Known As

LSTM: Long Short-Term Memory

CNN: Convolutional Neural Network

GCNN: Graph Convolutional Neural Network

FCNN: Fully Connected Neural Network

AVG: Average

STD: STandard Deviation

SUMO: Simulation of Urban Mobility

NE: North to East

NS: North to South

NW: North to West

ES: East to South

EW: East to West

EN: East to North

SW: South to West

SN: South to North

SE: South to East

WN: West to North

WE: West to East

WS: West to South

Table of Contents

List of Figures

Chapter 1

Introduction

The inefficient traffic light control causes congestion at the junctions as observed in everyday life. This will lead to a huge wastage of time, money, pollution of air, pollution of sound and even vehicle accidents. Existing traffic light control systems are explicitly programmed based on historical information without considering the real-time traffic.

Also, we often see emergency vehicles (ex: ambulance) are stuck in traffic and the patient's life depending on the traffic lights. Though this is a huge problem no previous work considered the emergency vehicle facilitation with traffic congestion control.

Worst case is when there is low traffic with no vehicle passing through the intersection, all waiting for their turn because of the explicitly programmed traffic light control without considering the real-time traffic.

We often witness policemen directly manage the intersection by hand signals. Most of the time this human operator can see the real-time traffic condition in the intersecting roads and smartly determine the duration of the allowed passing time for each direction using his/her long-term experience. But it is not practical to assign a policeman to every junction all the time. This witness motivates us to propose a smart traffic light control system which can take real-time traffic condition as input and manage the intersection just like the human operator.

1.1 Background Theories

Real-world traffic condition evolves in a complicated way, affected by many factors:

- Vehicle arrival rate
- Road conditions
- Weather
- Driver's preference

All of these things cannot be represented in a user-defined algorithm or from a traffic model. Techniques which can directly learn from the observed data without making unrealistic assumptions is needed.

1.1.1 Reinforcement Learning

Reinforcement learning (RL) [39] is an area of machine learning that focuses on how something might act in an environment to maximize some given reward. It is a trial and error method which will learn from the observations and learn from mistakes. The ultimate goal of a reinforcement learning agent is to learn an optimal policy function [1].

A reinforcement learning algorithm is modeled by a concept called Markov Decision Processes (MDPs) [2].

In MDPs there are 5 main components [39]

- Set of states' $\mathcal{S}$
- Set of actions $\mathcal{A}$
- State transition function $\mathcal{P}$
- Reward function $\mathcal{R}$
- Discount rate γ

$\mathcal{S}$: At time step t, the agent observes the state of the environment $s^t \in \mathcal{S}$.

$\mathcal{A}$, $\mathcal{P}$: At time step t, the agent takes an action $a^t \in \mathcal{A}$, which will introduce a state transition in the environment according to the state transition function.

$$\mathcal{P}\,(s^{t+1} \mid s^t, a^t): \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$$

R: At time step $t+1$, the agent obtains a reward r^{t+1} from a reward function by executing action $a^t \in \mathcal{A}$, in state $s^t \in \mathcal{S}$ and moving to the state $s^{t+1} \in \mathcal{S}$.

$$\mathcal{R}\,(s^t, a^t): \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$$

γ: The goal of an agent is to find an optimal policy that maximizes the expected return G, which is the discounted sum of all the rewards that the agent gets. So the γ will be a value between 0-1 and usually, it is chosen close to 1:

$G = \sum_{i=0}^{z} \gamma^i r^{t+1}$ where z can be infinite or finite.

Figure 1.0.1 illustrates the basic idea of reinforcement learning.

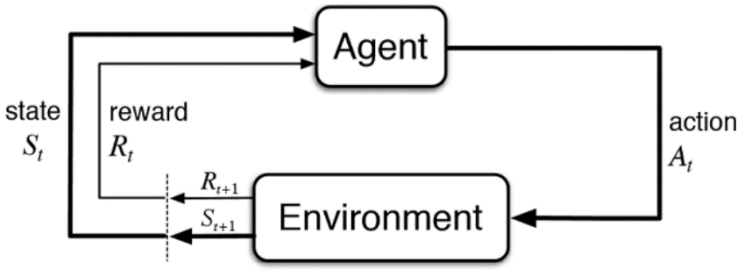


Figure 1.0.1: High-level Idea of RL

1.1.2 Exploration vs Exploitation

There are two ways of agent selecting an action $a^t \in \mathcal{A}$, by observing the state $s^t \in \mathcal{S}$. Agent will either explore the environment by taking random action $a^t \in \mathcal{A}$ or exploit the environment by taking action $a^t \in \mathcal{A}$ according to a learned policy [39].

1.1.3 Policies

When speaking about policies, formally we say that an agent "follows a policy." For example, if an agent follows policy π at time t, then $\pi(a|s)$ is the probability that $a^t = a$ when $s^t = s$. This means that, at time t, under policy π, the probability of taking action a in state s is $\pi(a|s)$. The policy can be represented as a state-value function or an action-value function [39].

1.1.4 State-Value Function

State-value function (V^π) defines how good it is for an agent to be in any given state $s \in \mathcal{S}$. That is the discounted expected return from starting from state s at time $t=k$ and following policy π thereafter [39].

$$V^\pi(s) = E_\pi \left[\sum_{t=k}^{\infty} \gamma^k r^{t+1} \mid s^t = s \right]$$

1.1.5 Action-Value Function

Action-value function (Q^π) defines how good it is for an agent who is in state $s \in S$ to take action $a \in \mathcal{A}$. That is the discounted expected return by taking action a in state s and following policy π thereafter [39].

$$Q^\pi(s,a) = E_\pi \left[\sum_{t=k}^{\infty} \gamma^k r^{t+1} \mid S^t = s, A^t = a \right]$$

1.1.6 Bellman Optimality Equation

Bellman optimality equation is considered as the heart of any reinforcement learning algorithm [2]. It is used to find an optimal policy function. In other words optimal action-value function ($Q^*(s,a)$) and optimal state-value function ($V^*(s)$).

Bellman Optimality Equation For Q^*

$$Q^*(s,a) = E_\pi \left[r^{t+1} + \gamma \max Q^*(s', a') \right]$$

This means that, for any state-action pair (s, a) at time t, the expected return from starting in state $s \in \mathcal{S}$, selecting action $a \in \mathcal{A}$ and following the optimal policy thereafter (aka *the Q-value* of this pair) is going to be the expected reward the agent can get from taking action **a** in state **s**, which is r^{t+1}, plus the *maximum* expected discounted return that can be achieved from any possible next state-action pair (s', a').

1.2 Why Reinforcement Learning?

To apply a traffic model, we need to differentiate between good and bad signal plans given a traffic condition. But the signal plan is always relative and nature of traffic is hard to model (**depends on the weather, road condition, driver's performance, day, time, season**) so we cannot define such signal plans. Also, we do not have such a big amount of training data to train with. Instead, we have to first take action to change the signal plans and then learn from the outcomes. This trial-and-error approach is also the core idea of Reinforcement Learning (RL).

1.3 Research Problem

How do we efficiently control traffic lights to minimize traffic congestion with emergency facilitation using real-time traffic data?

1.4 Research Questions

1) Does the **standard deviation of vehicle waiting time** distribution has a significant impact on traffic congestion control? (This will be further explained in section 1.5)
2) Does consideration of vehicle type can provide emergency facilitation?.

1.5 Justification for the research

Intelligent traffic light control is a vast area of research. Though there are many solutions proposed no one paid attention to emergency facilitation which is a very critical need for intelligent traffic light control. Though this is a less frequent situation, no intelligent traffic light control system is useful if it cannot adapt in a life-critical scenario. There is a previous research to prioritize emergency vehicles but their primary target is only to prioritize emergency vehicles, not to control traffic congestion [4]. So, this research will take the vehicle type as an input and adapt the traffic light in such a scenario and provide traffic congestion control with emergency facilitation.

Learning an optimal policy is a very challenging thing, hence a way of rewarding a reinforcement learning agent is the most critical part. Poor reward functions can lead the agent to learn poor policy functions. For examples consider the following traffic state representation of vehicle waiting time.

State 1 : vehicle 1 - 50s vehicle 2 - 5s vehicle 3 - 5s

State 2: vehicle 1- 20s vehicle 2-20s vehicle 3 - 20s

Most of the previous works have considered only the average waiting time of vehicles when rewarding the agent. Though the state one is better than state two, both the above states have the same reward because the average waiting time of vehicles is 20 in both cases.

Some other works have considered queue length, vehicle speed but queue length and speed will also be the same for both the states.

The simplest solution for this issue is to consider the maximum waiting time of vehicles when rewarding the agent but it will raise other issues. Consider the following example.

State 1 : vehicle 1 - 50s vehicle 2 - 50s vehicle 3 - 50s

State 2: vehicle 1- 50s vehicle 2-20s vehicle 3 - 10s

Though state 2 is better than state 1 the reward is the same for both the states if we consider maximum waiting time which is again not correct.

So to overcome this issue we consider average waiting time and the **standard deviation** of waiting times of the vehicles together to reward the agent. This is very similar to feature engineering in machine learning where a new feature is derived from doing mathematical operations to an existing feature. This will be further discussed in section 3.2.1.

For an agent to make decisions the state representation should contain all the necessary information about the traffic state. But that information should be easily and accurately measurable and should have a well-connected relationship with decision making. So in this research, we will use only the consequential traffic data to consideration.

Finally, innovations in RL such as dueling networks, experience replay, the double network is not applied in the field. According to the literature review, there is only one such research [11]. Therefore, this research will address all of the above mentioned major research gaps.

1.6 Research Goals and Objectives

- **Goals**
    - Evaluate the effectiveness of considering the standard deviation
    - Minimize traffic congestion
    - Facilitating emergency transportation
- **Objectives**
    - Build two versions of the agent with and without standard deviation to evaluate the effectiveness of considering standard deviation.
    - Identifying optimal policy function to reduce traffic.
    - Use the type of vehicles when making a decision to provide emergency facilitation.

1.7 Delimitations of Scope

- **In Scope**
    - Design and development prototype of the proposed method
    - Facilitating emergency transportation
    - Experiments and evaluations using the simulation environment (SUMO)
- **Out Scope**
    - Taking the real environment inputs by using cameras and sensors
    - Multiple junction environment

1.8 Research Assumptions

The Local solution to the problem will achieve the global solution.

1.9 Methodology

Since this domain is a well-explored research area [35] the first step of this research will be doing a deep literature review and compare how previous works handled the problem and identify the research gap. Finding a good simulation environment will be the next step of the methodology. The third step is to design and implement the proposed method and finally evaluate the proposed method with synthetic data as well as the real-world data. The evaluation step will consist of evaluating with normal traffic conditions, rush hours, low traffic and emergencies as well. The evaluation also consists of evaluating the proposed method with fixed time traffic control and with a user-defined traffic model.

- This research will be using a deductive approach
- Quantitative data: (this will be further explained in section 3)
    - Waiting time of a vehicle
    - Route of the vehicle
    - The standard deviation of waiting time
    - Number of vehicles
    - Phase index of the traffic light
- Qualitative data:
    - Type of a vehicle

Figure 1.0.2 represents the proposed research methodology.

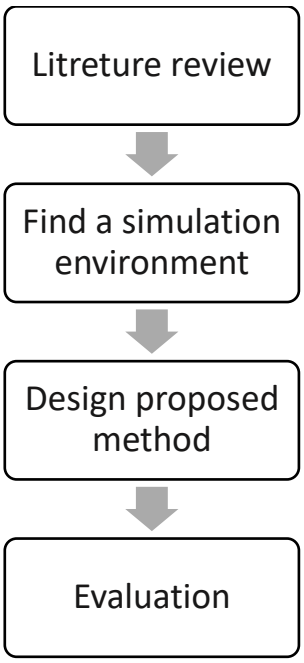


Figure 1.0.2: Proposed research methodology

1.10 Outline of the Dissertation

This dissertation outlined as follows. Chapter one states the introduction and background theories. Chapter two contains in-depth literature review about the domain which contains how previous works have addressed the RL components. The third Chapter states the design of the proposed methods and other methods which are used to evaluate the proposed method. Chapter four contains all the implementation details, hyperparameter settings used to implement the proposed method. Chapter five states the simulation settings, training, and evaluation of the proposed method and result analysis. The Last chapter, demonstrates the conclusion of the research, limitations and future work.

Chapter 2

Literature Review

Early studies on adaptive traffic control is based on simpler algorithms such as linear programming [18], fuzzy logic [17] etc. concerning the computing power at that era. In 2014 Obadah M.A Ayesh*, et al.* [3] have proposed a method by using the queue length and they suggested an equation to find out the estimated time and actual time to determine the estimated time reference for optic Green. But it was found that the queue length approach needs approximately 11.0849 seconds to make a decision.

Another work was done in 2014 for prioritizing emergency vehicles by using Radio Frequency [4]. But they have only targeted on emergency facilitation.

Malik Tubaishat, *et al.* proposed a method using wireless sensor networks in 2007[5] where sensors are deployed on the lanes going in and out the intersection to detect the vehicle 's number, speed, etc. But these methods are very costly and inefficient.

The application of AI to control traffic lights has been an active field of research since 1990. The most popular method of AI is machine learning. Machine learning provides systems the ability to learn and improve from examples without being explicitly programmed.

In 1994 Mikami, *et al.* [6] proposed distributed reinforcement learning using a genetic algorithm to control traffic lights. Due to the limitations of computational power, it could not be implemented at that time.

Until 2010 RL technique is limited to tabular Q learning [27] where they usually make small size state spaces such as the number of waiting vehicles at a given time [7], [8], [9], [10].

A study by applying Deep Reinforcement Learning was in 2018 by Xiaoyuan Liang, *et al.* [11]. They proposed a double dueling deep Q network (3DQN) with prioritized experience replay. They have defined the states like a grid-based on two pieces of information, position and speed of vehicles at an intersection. the action space is defined by selecting every phase's duration in the next cycle. rewards as the change of the cumulative waiting time between two neighboring cycles and they have used Adams optimizer as the optimizer for the deep Q-Network. They conclude that their method can reduce over 20% of the average waiting time compared to the waiting time of a vehicle when training starts. But in this research, they hadn't reflected how long a vehicle had been waiting in the queue at the state definition which will be very important when making a signal change.

Long Short-Term Memory (LSTM), Convolutional Neural Networks (CNN) is also applied in this field [28],[11], Tomoki Nishi *et all.* Proposed a method using Graph Convolutional Neural Networks (GCNN). The authors conclude that the proposed method is able to learn the policy twice as fast as regular Fully Connected Neural Network (FCNN) [29]. Also, there are some previous works based on Swarm Intelligence [13].

Many researchers are using RL to control traffic congestion. Below is the summary of how other researchers have defined the basic components of RL for traffic signal control.

2.1 State Definitions

State definition defines how to represent the traffic state to the RL agent. Table 2.1

Table 2.1: State Definitions

Element	Previous work
Queue length	Wei *et al.* 2018 [14], Aslani *et al*. 2017 [15], Mannion *et al*. 2016 [22]
Waiting time	Wei *et al* 2018 [14], Chu et al. 2019 [20]
Volume	Wei *et al* 2018 [14], Aslani *et al*. 2017 [15], Casas 2017 [17]
Delay	Arel *et al*. 2010 [9]
Speed	Liang *et al* 2018 [11], El-Tantawy *et al*. 2013 [16], Casas 2017 [17]
Phase duration	van der Pol *et al*. 2016 [18], El-Tantawy *et al*. 2013 [16]
Congestion	Bakker *et al*. 2010 [19], Iša et al. 2006 [23]
Position of vehicles	van der Pol *et al*. 2016 [18], Wei *et al*. 2018 [14], Liang *et al* 2018 [11], Bakker *et al*. 2010 [19]
Phase	Aslani *et al*. 2017, 2018b; El-Tantawy*et al*. 2013, Wei *et al.* 2018 [14]

Queue length: Queue length calculated concerning the lane which is the number of waiting vehicles on the lane. There are different definitions of a "waiting" state of a vehicle. In [14], a vehicle with a speed of less than 0.1 m/s. is considered as waiting; in [19], a vehicle without movement is considered as waiting.

Waiting time: How long a vehicle been waiting. As mentioned above there are different definitions of waiting state. The starting point of the waiting time is also having different definitions: in [18],[14], they consider the waiting time starting from the last timestamp

the vehicle moved, while [11] consider the waiting time from the time the vehicle enters the road network.

Volume: number of vehicles on the lane within a given radius.

Delay: Delay of a vehicle is defined as the time taken by a vehicle to pass through the junction minus the normal travel time (which equals the distance divided by the speed limit).

Phase duration: Phase duration of the current phase is defined as how long the current phase has lasted.

Speed: vehicle moving speed is also considered in previous studies. However, it is inefficient and computationally expensive to measure and calculate the speed of each vehicle in practical.

Congestion: Some studies take the congestion of the outgoing lane into account for effective learning for the cases of congestion and no congestion. This is very useful for Srilankan Context. The congestion of a lane can be defined either as an indicator (0 for no congestion and 1 for congestion) or the level of congestion which is equal to the number of vehicles divided by the maximum allowed vehicles on the lane.

Positions of vehicles: Some studies represent the positions of vehicles in a grid-like structure and pass is through a CNN to recognize the traffic pattern. A matrix is used to represent the grid and 0 indicates no vehicle in that grid and 1 indicated the presence of vehicle [11], [14], [18].

Phase: Traffic signal combination index [14], [18].

Some recent studies such as [14], [18]. propose to use images as states. However, these methods will take a longer time to train and because of the higher dimension of the input, there will be a huge number of zero observations.

2.2 Reward Functions

Table 2.2 contains the summary and the comparison of reward functions proposed in previous works

Table 2.2: Reward Functions

Element	Previous work
Queue length	Wei *et al* 2018 [14], Aslani *et al*. 2017 [15] ,van der Pol et al. 2016 [18], Mannion et al. 2016 [22]
Waiting time	Liang *et al* 2018 [11], Bakker et al. 2010 [19], Chu et al. 2019 [20], Mannion et al. 2016 [22], Nishi et al. 2018 [21] ,van der Pol et al. 2016 [18], Wei et al. 2018 [14], Xu et al. 2013 [24]
Speed	Casas 2017 [17], van der Pol et al. 2016 [18], Wei et al. 2018 [14]
Number of stops	van der Pol *et al* 2016 [18]
Throughput	Aslani *et al*. 2017 [15], Wei et al. 2018 [14], Xu et al. 2013 [24]
Frequency of signal change	Wei *et al*. 2018 [14]
Accident avoidance	Van der Pol *et al*. 2016 [18]

Queue length: lower the queue length, higher the reward to the agent.

Speed. A reward that takes the average speed of all vehicles in the road. A higher average speed of vehicles in the road network indicates the vehicles travel to their destinations faster. Higher the speed Higher the reward.

The number of stops. A reward can use the average number of stops of all vehicles in the network. Intuitively, the smaller the number of stops, the more smoothly the traffic moves.

Throughput. The throughput is defined as the total number of vehicles that pass the intersection or leave the network during a certain time interval after the last action.

Frequency of signal change. The frequency of signal change is defined as the number of times the signal changes during a certain time period. Intuitively, the learned policy should not lead to flickering, i.e. changing the traffic signal frequently, as the effective green time for vehicles to pass through the intersection might be reduced.

Accident avoidance. Some studies have special considerations on accident avoidance.

For example, there should not be many emergency stops. Furthermore, jams or would-be collisions should be prevented.

2.3 Action Definitions

Table 2.3 contains the summary and the comparison of action definitions proposed in previous works

Table 2.3: Action Definitions

Element	Previous work
Set the current phase duration	Aslani *et al.* 2017 [15], Xu et al. 2013 [24]
Set cycle-based phase ratio	Liang *et a*l 2018 [11], Casas 2017 [17]
Keep or change the current phase	van der Pol *et al*. 2016 [18] Wei *et al*. 2018 [14], Mannion et al. 2016 [22]

Choose next phase	Chu *et al*. 2019 [20], El-Tantawy *et al*. 2013 [16], Nishi *et al*. 2018 [21], Arel et al. 2010 [9], Bakker et al. 2010 [19]

▪ **Set the current phase duration**. Here, the agent learns to set the duration for the current phase by choosing from pre-defined candidate time periods.

▪ **Set cycle-based phase ratio**. Here, the action is defined as the phase split ratio that the signal will set for the next cycle. In these types of methods, the total cycle length is defined to be fixed and action is to select a phase ratio from a pre-defined set of candidate phase ratios.

▪ **Keep or change the current phase**. Here, an action is represented as a binary number which indicates the agent decides to keep the current phase or change to the next phase in a cycle-based phase sequence.

▪ **Choose the next phase**. Decide which phase to change in a variable phase sequence, in which the phase sequence is not predetermined. Here, the action is the phase index that should be taken next. As a result, this kind of signal timing is more flexible, and the agent is learning to select a phase to change to, without assumptions that the signal would change cyclically.

2.4 Existing Systems

- SCATS(Sydney Coordinated Adaptive Traffic System) installed at about 42,000 intersections in over 40 countries [32].
- SCOOT(Split Cycle Offset Optimisation Technique) [31].

They are using real-time traffic data to make decisions but both of them are developer-defined traffic signal plans.

OPAC [33] and PRODYN [34] are some other similar solutions but due to their complexity, they are less popular.

2.5 Synopsis

This chapter mainly focused on how earlier researchers proposed solutions on this domain. This chapter encompasses two parts, firstly it briefly describes solutions without reinforcement learning and as the next part, it contains the summary and comparisons of other proposed methods on reinforcement learning over more than twenty papers. The comparison compares how the main components of reinforcement learning (States, Actions, Rewards) are defined in previous studies. Finally, it mentions existing systems on the domain.

Chapter 3

Design

This chapter will elaborate on the design of the proposed method and the other three methods (avg only method, statistical method, fixed time method) which are used to compare and contrast the proposed method.

As the first part of the research design, a four-way junction with three incoming traffic tracks (left turn, right turn, direct track) and one outgoing lane for each road is designed. Figure 3.1 shows the overview of the four-way junction.

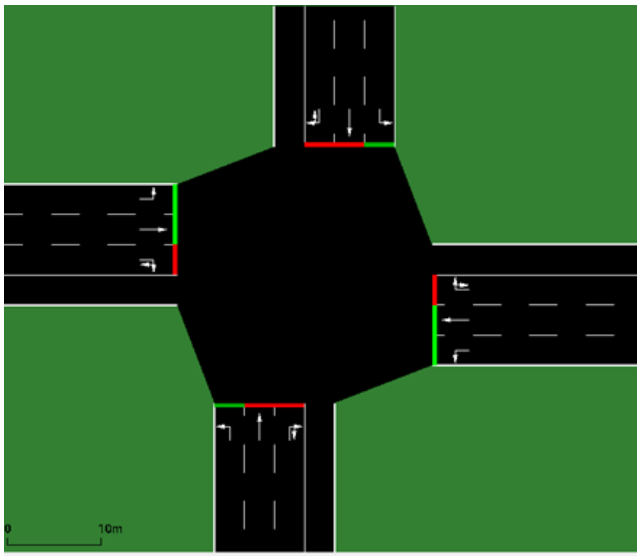


Figure 3.1: Overview of the four-way junction

3.1 Valid States of Traffic Light

Combination of signals which will allow vehicles to pass over intersection without conflict is a valid state.

Traffic lights should always be one of these valid states. To be clear Figure 3.2 represents a hand drawing of an invalid state.

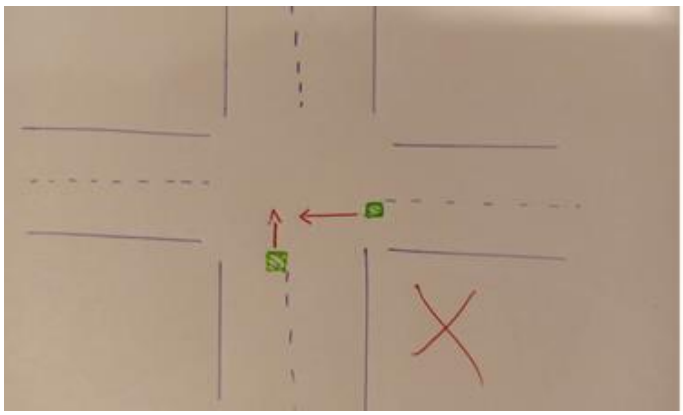
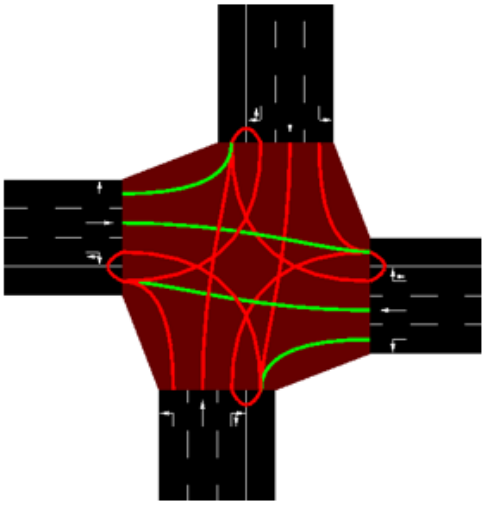
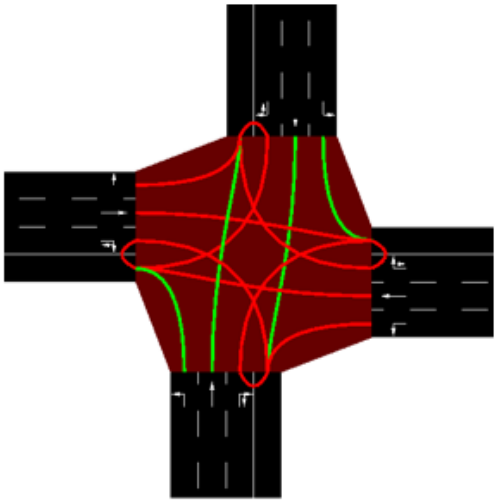


Figure 3.2: Hand drawing of an invalid state

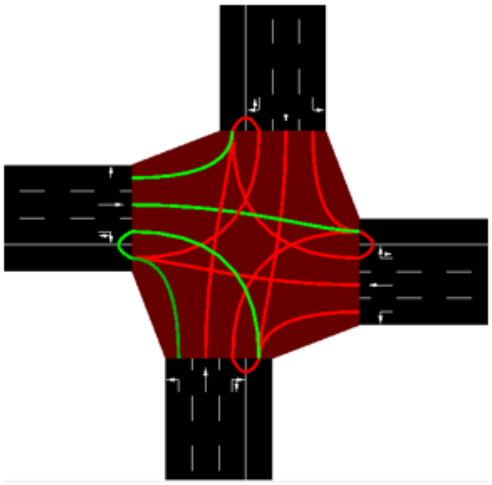
Altogether 8 such valid states are defined as shown in Figure 3.3 Green lines are the directions that the vehicles are **allowed** to pass through the junction in that state. Red lines are the directions that the vehicles are **not allowed** to pass through the junction in that state.



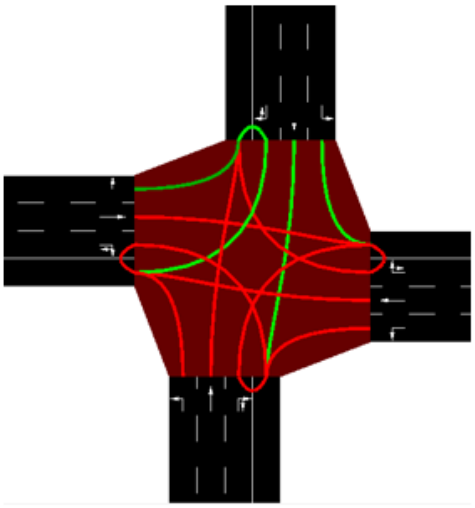
Phase 1 : vehicles are allowed to pass WE, EW, WN, ES directions only.



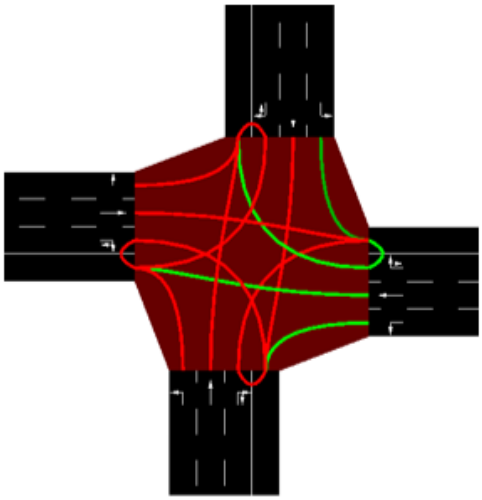
Phase 2 : vehicles are allowed to pass NS, SN, SW, NE directions only.



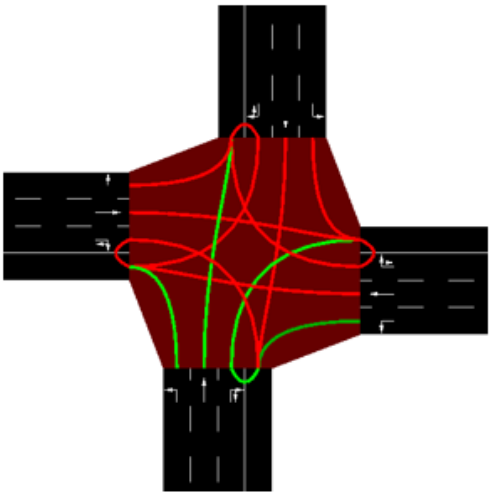
Phase 3 : vehicles are allowed to pass WE, WS, WN, SW directions only.



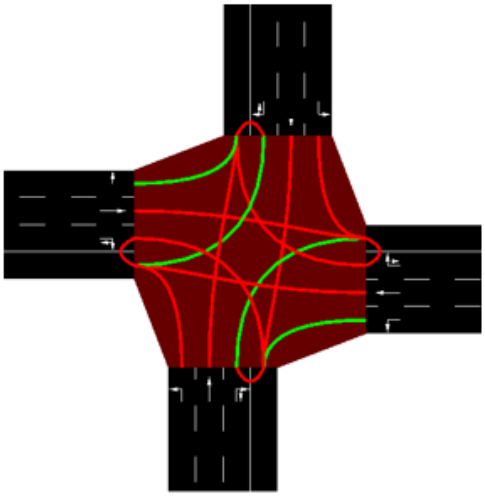
Phase 4 : vehicles are allowed to pass NE, NS, NW, WN directions only.



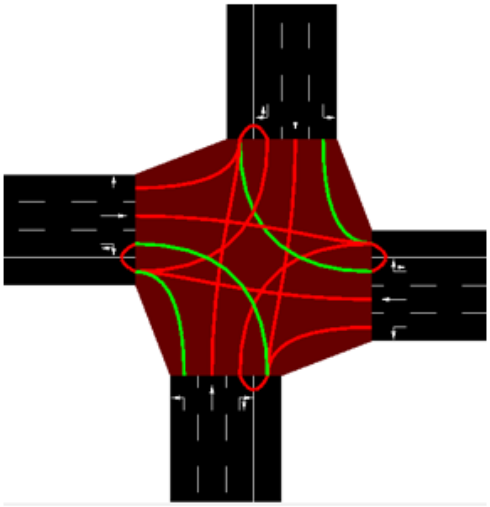
Phase 5 : vehicles are allowed to pass
ES, EW, EN, NE directions only.



Phase 6 : vehicles are allowed to pass
SW, SN, SE, ES directions only.



Phase 7 : vehicles are allowed to pass
WN, NW, SE, ES directions only.



Phase 8 : vehicles are allowed to pass
WS, SW, EN, NE directions only.

Figure 3.3: Eight valid States

There will be a yellow state between every two states to avoid emergency braking and ensure safety. (

Figure 3.4)

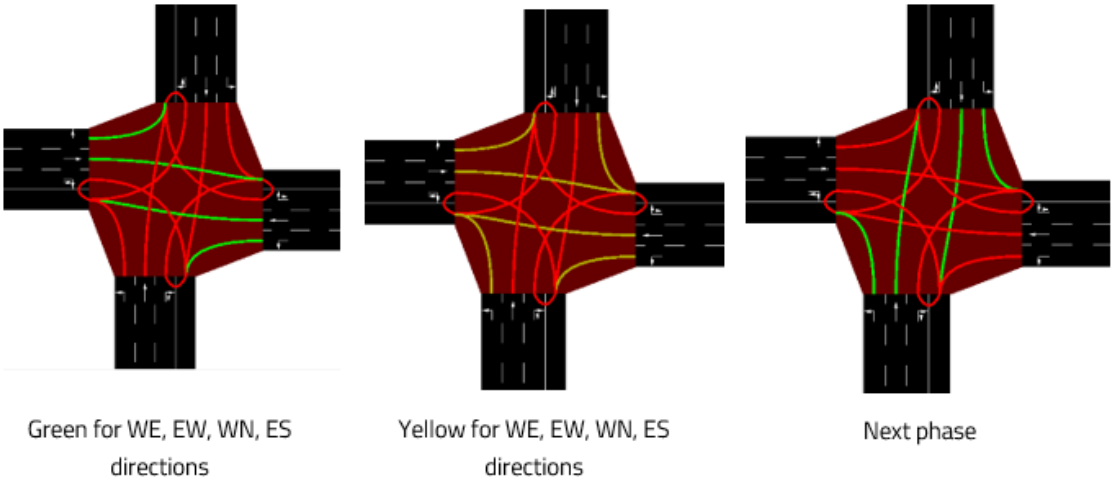


Figure 3.4: Yellow state between each two valid states

3.2 Proposed Method

This section will elaborate on the design of each component of the proposed method. In generic, the agent will observe the current traffic environment(state) and it will take necessary action (signal change) using a deep q-network which is a deep neural network. Then the agent will be rewarded on how it was able to minimize the traffic congestion and how it was able to prioritize emergency vehicles. After that by using the Bellman optimality equation (section 1.1.6) weights of the neural network will be updated to obtain the optimal action-value function (section1.1.5). Figure 3.5 represents the Generic working of the proposed method.

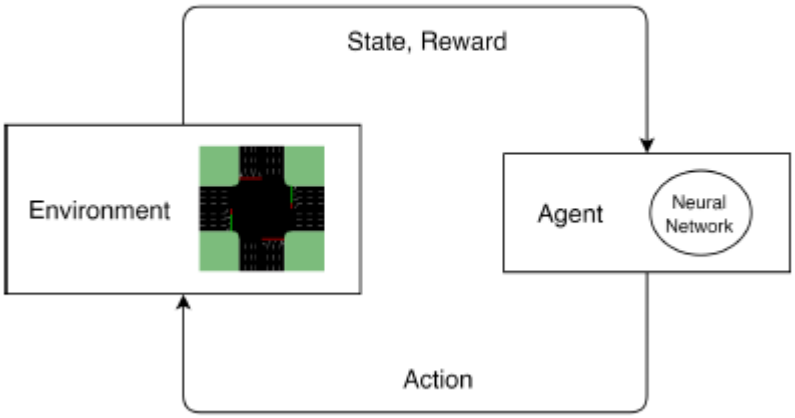


Figure 3.5: Generic working of the proposed method

Throughout this section, state representation, action representation, reward design, DQN design of the proposed method will be discussed.

3.2.1 State Representation

Realtime traffic data around the junction should be represented in a numerical way such that it contains all the traffic information for an agent to learn an optimal policy. According to the literature review, there are many different ways of state representation.

We first employed the same method that chu *et al.* [20] have proposed with a small change that is to represent each lane $\boldsymbol{l}$ by a 3-dimensional vector.

$$l_{i,t} = \{ length_{i,\,t} , wait_{i,t} , emg_{i,t} \}$$

Where i is lane index t is timestamp. $\mathrm{length}_{i,\,t}$ denotes the number of waiting vehicles in that lane, $wait_{i,t}$ measures the waiting time of the front vehicle of that lane which will obviously be the maximum waiting time of that lane, $emg_{i,t}$ denotes the maximum waiting time of emergency vehicles in that lane. But it was found that this representation lacks some important information about the environment.

Consider the following Figure 3.6 which shows two cases of vehicle waiting times in a lane. Waiting times of vehicles are displayed under the vehicles.

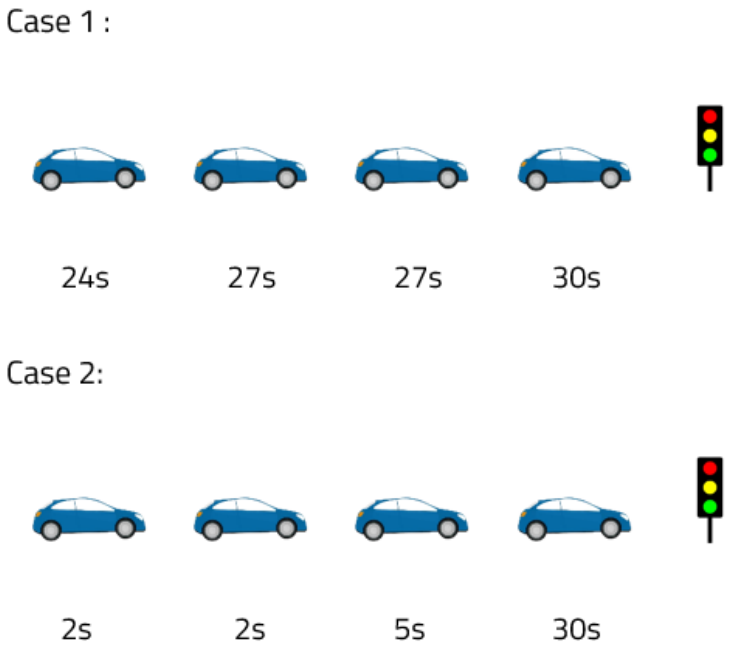


Figure 3.6: Two cases of vehicle waiting times-example 1

For both the cases $l_{i,t}$ will be the same because for both cases the front vehicle's waiting time is 30s which is the maximum waiting time of vehicles on the lane, the number of vehicles in the lane is four and no emergency vehicle presents. But for an adequate state representation $l_{i,t}$ should clearly differentiate these two cases in order for an agent to learn an optimal policy.

To overcome this we defined $wait_{i,t}$ as the average waiting time of vehicles in lane $\boldsymbol{l}$. But it was found that this representation also lacks some important information about the environment. Consider the following Figure 3.7 which shows two cases of vehicle waiting times in a lane. Waiting times of vehicles are displayed under the vehicles.

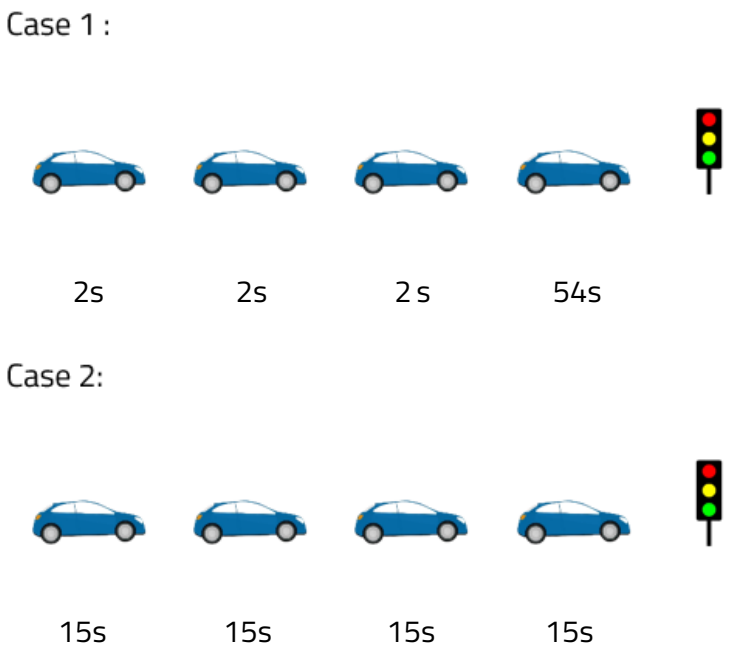


Figure 3.7: Two cases of vehicle waiting times-example 2

For both the cases $l_{i,t}$ will be the same because for both cases the average waiting time is 60s (2 + 2 + 2 + 54 = 60 ; 15 + 15 + 15 + 15 = 60), the number of vehicles in the lane is four and no emergency vehicle presents.

To differentiate those kinds of cases standard deviation of waiting times of vehicles ($std\_wait_{i,t}$) in each lane $\boldsymbol{l}$ along with the average waiting time is considered. That will represent the average waiting time of vehicles and how close each vehicle's waiting time to the average waiting time.

$$l_{i,t} = \{ length_{i,\,t}\,,\, avg\_wait_{i,t},\, std\_wait_{i,t},\; emg_{i,t} \}$$

As proposed in [19], [23] we decided to add congestion level $c_{i\,,\,t}$ of the corresponding outgoing lane $\boldsymbol{l}^{|}$(lane to which vehicles are moving into) for each lane $\boldsymbol{l}$. The final lane representation of the proposed method is.

$$l_{i,t} = \{ length_{i,\,t}\,,\, avg\_wait_{i,t},\, std\_wait_{i,t},\; emg_{i,t},\, c_{i,\,t} \}_{1x5}$$

For the state representation, we also add the current phase index $\boldsymbol{p}$ of the traffic light as proposed in [14] where $1 \leq \boldsymbol{p} \leq 8$. The hypothesis here is by using phase index the agent will get some idea about moving vehicles. (Vehicles in which the lanes are allowed to pass through the junction are moving vehicles).

As the final representation of the environment, we represent a state with traffic matrix $\boldsymbol{\mathcal{M}}$ (12x5 matrix 12 lanes and one lane $l_{i,t}$ is represented by a vector of dimensionality 5) and one hot encoded vector [38] of phase index $\boldsymbol{\mathcal{P}}$ (1x8).

Figure 3.8 represents a sample traffic state. Lane indexes are circled and dark squares represent the emergency vehicles and others are regular vehicles. The numbers inside squares represent the waiting time of the vehicle in seconds. Figure 3.9 represents the corresponding state representation matrices $\boldsymbol{\mathcal{M}}$ and $\boldsymbol{\mathcal{P}}$.

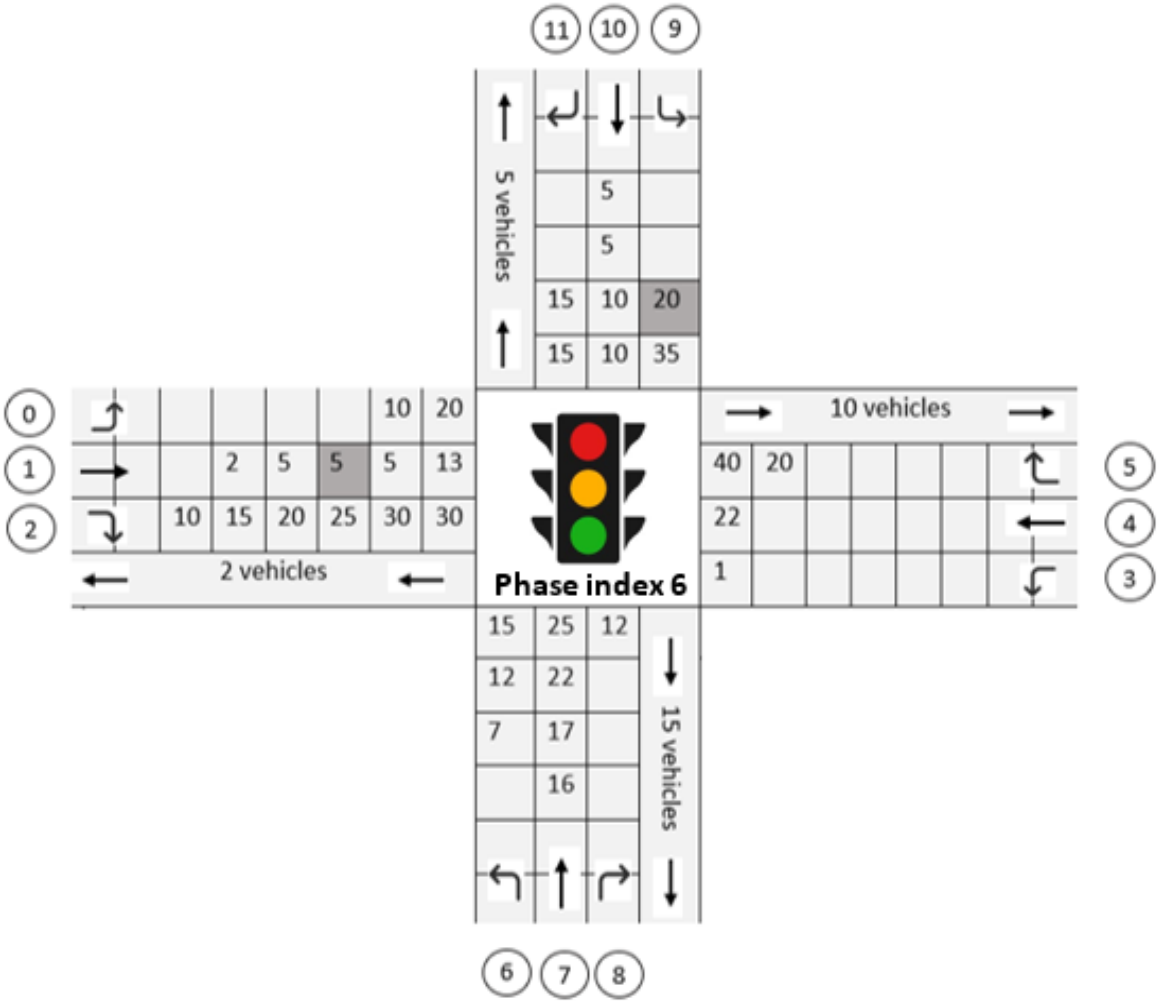


Figure 3.8: Sample traffic state

2	15	5	0	12/17
5	6	3.68	5	7/17
6	21.66	7.45	0	2/17
1	1	0	0	2/17
1	22	0	0	15/17
2	30	10	0	12/17
3	11.33	3.29	0	15/17
4	20	3.67	0	12/17
1	12	0	0	7/17
2	27.5	7.5	20	7/17
4	7.5	2.5	0	2/17
2	15	0	0	15/17

$\mathcal{M}$

0	0	0	0	0	0	1	0

$\mathcal{P}$

Figure 3.9: Corresponding traffic state matrices M and P

Each row of M contains traffic information about the matching traffic lane number with the row index. The first column represents the number of vehicles, second column represents the average waiting time of the lane, third column represents the standard deviation of the lane, fourth column represents the average emergency waiting time of the lane, fifth column represents the congestion level of the corresponding outgoing lane. When calculating the congestion level the negative impact of the congestion is added as the value. The maximum number of vehicles allowed in a lane is 17. For example in the first row $12/17 = 1 - 5/17$.

3.2.2 Action Representation

The agent's action space is the set of phase indexes of the traffic light $\mathcal{A}$ = { p1, p2, p3, p4, p5, p6, p7, p8} [39] which is a similar approach to state spaces proposed in [20],[21], [37]. After every twelve seconds agent will select the next phase from the action space. If the selected next phase is the same as the current phase agent will stay in the same state for another twelve seconds, otherwise, the agent will move to the new phase preceded by three seconds of yellow signal phase.

3.2.3 Reward Design

The agent will be rewarded on the action a_t it makes depending on the new traffic state after changing the traffic signal to the selected phase at time t [39]. The traffic environment is allowed to execute for twelve seconds with the new traffic signal phase and then observe the traffic state as the new traffic state such that vehicles will have enough time to move and form a new traffic state.

The agent should be rewarded based on both how it controls regular vehicles and how it controls emergency vehicles. Also, the agent should be rewarded not only with how it reduces the average waiting time but also how good it was able to reduce standard deviation as well so that we can guarantee that other vehicles will stay close to average waiting time.

Then the reward R_{t+1} is defined concerning the new traffic state as follows.

$$R_{t+1} = 50 - ((\text{reg\_avg\_waiting}_{t+1} + K * \text{reg\_std\_waiting}_{t+1}) + Z * (\text{emg\_avg\_waiting}_{t+1} + K * \text{emg\_std\_waiting}_{t+1}))$$

Where

- K, Z are hyperparameters (refer section 5.3.1 for different value assignments)
- $\text{reg\_avg\_waiting}_{t+1}$: average waiting time of regular vehicles around 100m radius to the junction.
- $\text{reg\_std\_waiting}_{t+1}$: standard deviation of waiting time of regular vehicles around 100m radius to the junction.
- $\text{emg\_avg\_waiting}_{t+1}$: average waiting time of emergency vehicles around 100m radius to the junction.
- $\text{reg\_avg\_waiting}_{t+1}$: average waiting time of regular vehicles around 100m radius to the junction.

We used the negative impact of average waiting times [11] and the negative impact of the standard deviation of waiting times to punish the agent with a negative reward if the waiting time and standard deviation are considerably higher. Technically if

$((\text{reg\_avg\_waiting}_{t+1} + \text{K} * \text{reg\_std\_waiting}_{t+1}) + \text{Z} * (\text{emg\_avg\_waiting}_{t+1} + \text{K} * \text{emg\_std\_waiting}_{t+1})) > 50$ the agent will receive negative reward otherwise it will receive positive rewards.

3.2.4 Deep Q-Network Design

Since the state space is infinite, simple tabular Q-learning is not possible. Instead, a deep neural network [41] to predict the action given the state is needed which is deep reinforcement learning [40].

State representation matrices ($\mathcal{M}$ and $\mathcal{P}$ as discussed in section 3.2.1) are faded into the neural network through two separate input layers and concatenated inside the neural network.

Traffic matrix $\mathcal{M}$ and phase index vector $\mathcal{P}$ first passed through two fully connected layers [41] and then passed through a concatenation layer to concatenate the two outputs. The output of the concatenation layer then passed through another fully connected layer to form the final phase index prediction. Figure 3.10, Figure 3.11 and Figure 3.12 represent the plot and the summary of the proposed deep q-network and the architecture of the DQN respectively.

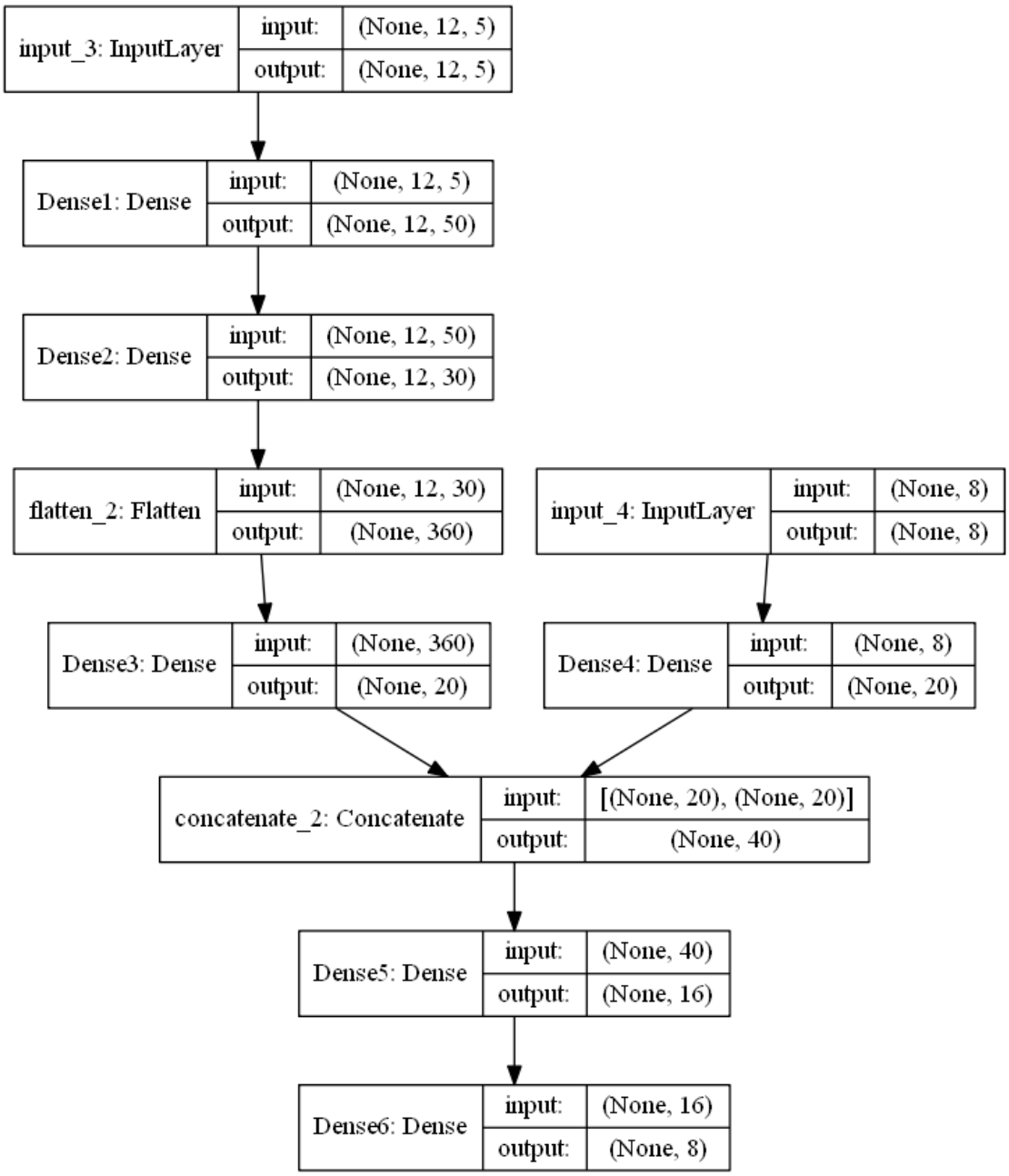


Figure 3.10: Plot of the proposed deep q-network

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 12, 5)	0	
Dense1 (Dense)	(None, 12, 50)	300	input_1[0][0]
Dense2 (Dense)	(None, 12, 30)	1530	Dense1[0][0]
flatten_1 (Flatten)	(None, 360)	0	Dense2[0][0]
input_2 (InputLayer)	(None, 8)	0	
Dense3 (Dense)	(None, 20)	7220	flatten_1[0][0]
Dense4 (Dense)	(None, 20)	180	input_2[0][0]
concatenate_1 (Concatenate)	(None, 40)	0	Dense3[0][0] Dense4[0][0]
Dense5 (Dense)	(None, 16)	656	concatenate_1[0][0]
Dense6 (Dense)	(None, 8)	136	Dense5[0][0]

Total params: 10,022
Trainable params: 10,022
Non-trainable params: 0

Figure 3.11: Summary of the proposed deep q-network

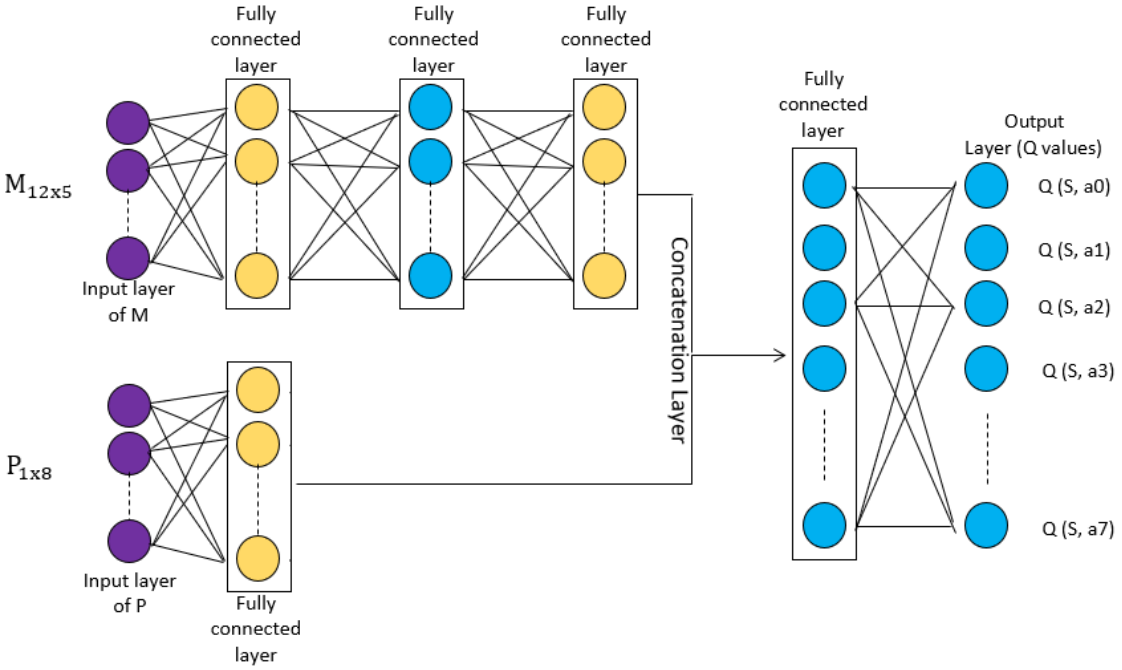


Figure 3.12: DQN architecture

As shown in Figure 3.12 DQN contains eight neurons in the output layer which will output the Q-values for each signal phase when the current traffic state $\mathcal{M}$ and $\mathcal{P}$ are faded into the neural network through input layers. The agent then changes into the signal phase which has the maximum q value.

$$\text{Next phase} = \text{argmax}\ (\ Q(\ \mathbf{s_t}\ ,\ \boldsymbol{a}\ )\ |\ \text{for every}\ \boldsymbol{a} \in A)$$

Weights of the deep q-network are updated according to the Bellman optimality equation as discussed in section 1.1.6 using experience replay.

3.2.5 Experience Replay

When the agent takes an action $\boldsymbol{a_t}$ at state $\mathbf{s_t}$ then it will observe a new state $\mathbf{s_{t+1}}$ and receive a reward at $\mathbf{R_{t+1}}$ which is an experience of taking an action in a given state. The agent should store this experience in a memory called *replay memory* for the learning purpose. Experience $\boldsymbol{e}$ is defined as a tuple of four elements as follows.

$$e = \{a_t\ ,\ s_t\ ,\ R_{t+1}\ ,\ s_{t+1}\}$$

After each signal change agent will store the current experience in the replay memory and take a random sample called *batch* out of the replay memory and train the DQN with it which is called experience replay [42]. The training of the agent is discussed in section 3.2.6.

Replay memory has its own memory size $\mathcal{M}$**.** When the replay memory is full of experience, the oldest experience is removed to store the new experience. Also, the experience replay will not be allowed to run if the number of samples in the replay memory is less than batch size $\boldsymbol{b}$**.** Figure 3.13 represents the process of experience replay after each action.

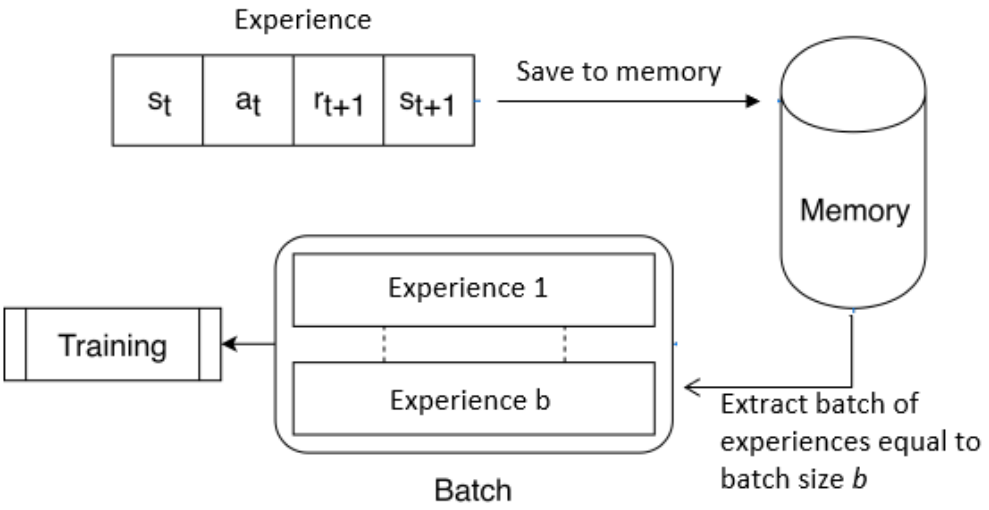


Figure 3.13: Process of experience replay after each action

3.2.5.1 Advantages of Experience Replay

With the use of experience replay, the training phase has two major advantages:

1. It removes correlations in the observation sequence [43].
2. Refresh the experience of the agent [42].

In this environment, two consecutive states are naturally correlated since the state of the environment $\mathbf{s_{t+1}}$ is a direct evolution of the state $\mathbf{s_t}$. Most of the information contained in the state $\mathbf{s_{t+1}}$ does not derive as a consequence of an agent's action, but rather as the spontaneous transformation of the current situation $\mathbf{s_t}$. Therefore experience replay has been implemented to not inducing misleading correlation in the agent's neural network. Secondly, during the training process, there is the possibility that the neural network forgets the knowledge gained about a situation visited in the early stages of the training. By using experience replay, the agent occasionally receives a "refresh" of what it has learned before in an older state.

3.2.6 Learning the Optimal Policy

Learning will undergo after each signal change with an experience batch size of $\boldsymbol{b}$. Target is to learn the optimal Q-value for any given state and action. For every sample in experience batch, the following operations are performed [39].

1. For state $\mathbf{s_t}$ and action $\boldsymbol{a_t}$ in the experience obtain the predicted Q-value $Q(\mathbf{s_t}, \boldsymbol{a_t})$.
2. Then calculate the target Q-value for state $\mathbf{s_t}$ and action $\boldsymbol{a_t}$ using the Belman optimality equation.

$$Q_{target}(\mathbf{s_t}, \boldsymbol{a_t}) = r^{t+1} + \gamma \max(Q(\mathbf{s_{t+1}}, \boldsymbol{a}) \mid \text{for every } \boldsymbol{a} \in A)$$

   - r^{t+1} can be directly extracted from the experience tuple.
   - To calculate $\max(Q(\mathbf{s_{t+1}}, \boldsymbol{a}) \mid \text{for every } \boldsymbol{a} \in A)$ part $\mathbf{s_{t+1}}$ is faded into the DQN and obtain the Q-values for $\mathbf{s_{t+1}}$
   - Take the maximum Q-value for $\mathbf{s_{t+1}}$
   - Calculate $Q_{target}(\mathbf{s_t}, \boldsymbol{a_t})$.
3. Calculate the error E = $Q_{target}(\mathbf{s_t}, \boldsymbol{a_t}) - Q(\mathbf{s_t}, \boldsymbol{a_t})$.
4. $Q_{target}(\mathbf{s_t}, \boldsymbol{a_t}) = Q(\mathbf{s_t}, \boldsymbol{a_t}) + \Delta E$

3.2.7 Fixed Q-Targets

Using the same DQN to obtain the $Q(\mathbf{s_t}, \boldsymbol{a_t})$ and the $Q_{target}(\mathbf{s_t}, \boldsymbol{a_t})$ leads to unstable learning of the agent since the weights of the DQN are updated after replaying every sample (as mentioned in section 3.2.6) it will output a different $Q_{target}(\mathbf{s_t}, \boldsymbol{a_t})$ for the same state and action which leads to unstable learning.

Fixed Q-Targets [43] introduced as a solution for unstable learning. It proposed using a separate version of DQN of the agent as the target network and calculate $Q_{target}(\mathbf{s_t}$

$, a_t$) from it. Then clone the DQN into to target graph after some rounds of experience replay. This will make learning stable for a while.

3.2.8 Double Q-learning

Double Q-learning [44] is a solution for the overestimation of Q-values. max (Q(s_{t+1} , a) | for every $a \in$ A) will depend on how to DQN weights are initialized and what actions and what states are explored by the agent which means we cannot guarantee max (Q(s_{t+1} , a)) is correct. The solution proposed is to use DQN network to select the best action of the next state s_{t+1} and use the target network to calculate the Q-value of that action for state s_{t+1} . Figure 3.14 represents the idea of Double Q-learning

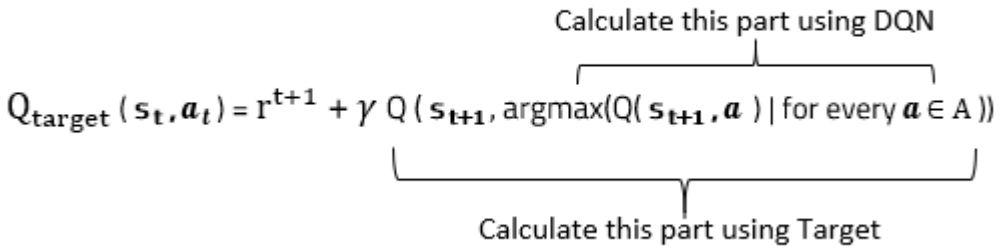


Figure 3.14: Idea of Double Q-learning

3.2.9 Exploration Vs Exploitation Tradeoff

Initially, the traffic control agent knows nothing about controlling traffic so that the agent is allowed to take random actions to explore the environment and get the experience. With the time agent will learn to control traffic efficiently so that it will be allowed to make signal changes with its learning. Initially, the agent is advised to Explore the environment but with the time it is advised to exploit the environment than exploring. For that, a concept of exploration vs exploitation tradeoff introduced [39].

Exploration rate $0 < \epsilon < 1$ is defined as

$$\epsilon = e^{-1 * action\ number * epsilon\ decay\ rate}$$

Epsilon decay rate decides how fast ϵ is decreasing. Then a random number is generated *r.* Then the agent will decide to either explore or exploit as follows.

```
If r < ϵ
    Agent explores the environment
Else
    Agent exploits the environment
```

3.2.10 High-Level Architecture

Figure 3.15 illustrates the high-level architecture of the proposed method. For each 12 seconds agent first observe the current traffic state and build the M and P matrices. If r < ∈ then it selects a random traffic phase from the eight valid phases, else it predicts the best next signal phase using the DQN. The agent will then change the traffic signal to the new traffic phase. The agent will then receive a reward calculated by reward function as discussed in Section 3.2.3. Finally, the agent will store the experience in replay memory and execute experience replay. Experience replay is done with the help of Fixed Q-Targets and Double Q-Networks.

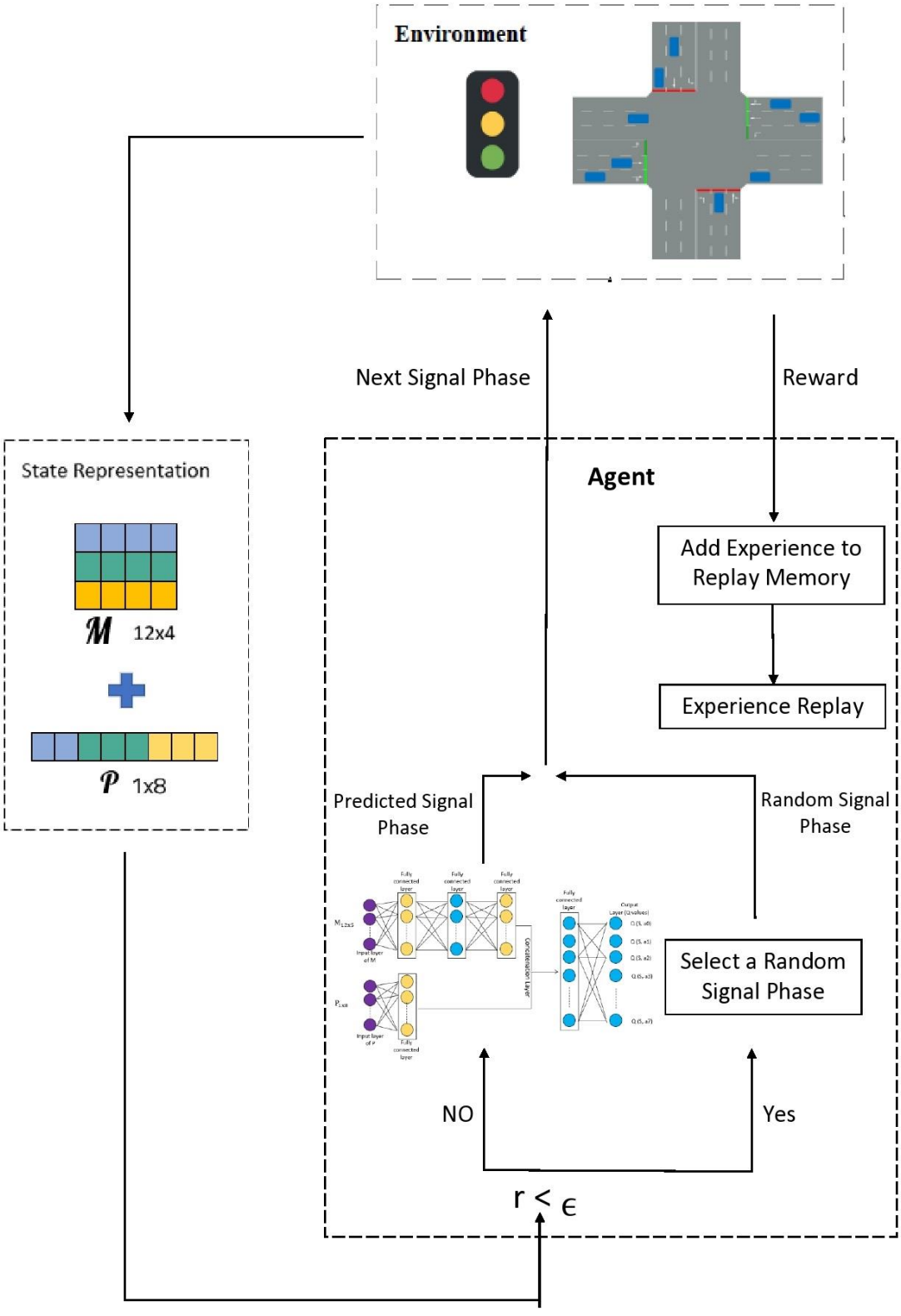


Figure 3.15: High-level Architecture of the Proposed Method

3.3 Average Only Method

To demonstrate that considering average and the standard deviation (proposed method) has a significant improvement over only considering the average, another version of the same agent is designed by only considering the average waiting time of vehicles. Which will be called as the average only method.

Only the state representation, reward design, and DQN is the difference when compared to the proposed method. All other design elements remain the same for both the agents.

3.3.1 State Representation

The only difference with the proposed method state representation is that this representation doesn't contain $std\_wait_{i,t}$ column to represent the traffic condition in each lane. Which means the $l_{i,t}$ is defined as

$$l_{i,t} = \{ length_{i,\,t}\,,\, avg\_wait_{i,t},\ emg_{i,t},\, c_{i,\,t} \}_{1x4}$$

This also means that the dimension of M will be 12x4 matrix and p remains in the same dimension 1x8.

3.3.2 Reward Design

The reward design of the average only method is very similar to the proposed method without considering the standard deviation of vehicles when rewarding the agent. The reward for the average only method is defined as follows.

$$R_{t+1} = 30 - (\text{reg\_avg\_waiting}_{t+1} + Z * \text{emg\_avg\_waiting}_{t+1})$$

3.3.3 Deep Q-Network Design

Since the diamention of the M is changed the only difference is that input layer of M is in dimention 12x3 every other designs of the DQN remains the same.

3.4 Statistical Method

The method proposed by Binbin Zhou et.all [48] is used as the statistical method to evaluate the proposed method.

3.5 Fixed Time Method

The fixed time method allows each phase to stay in 30 seconds and cyclically move to the next phase. This means the fixed time method is not considering real-time traffic data. Figure 3.16 represents the cyclic phase change of the fixed time method.

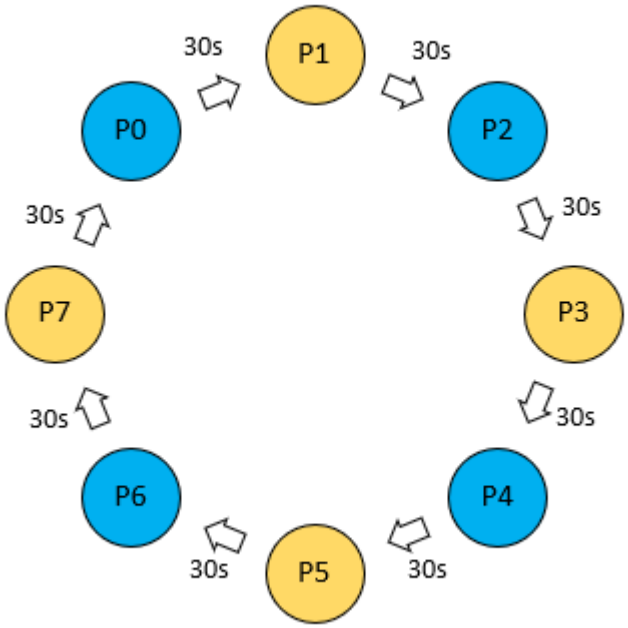


Figure 3.16: Cyclic phase change per 30s in fixed time method

Chapter 4

Implementation

This chapter mainly focuses on the implementation details of the proposed method, average only method, statistical method and fixed time method. Apart from that, configuration of the simulation environment is also discussed in this chapter. Implementation and evaluation were done using ASUS Intel® core™ i5-6200U CPU @ 2.30GHz 2.40GHz with 8GB RAM and 2GB of NVIDIA GeForce 920MX GPU.

4.1 Configuration of SUMO

Sumo (Simulation of Urban MObility)[30] is used as the simulation software for real-time traffic simulation. Sumo is developed using C++ by the Institute of Transportation Systems at the German Aerospace Center.

The simulation environment is originally in the left-hand driving mode so that it has to be converted to the right-hand driving mode to match with the local scenario. A four-way junction with three incoming traffic tracks (left turn, right turn, direct track) and one outgoing lane for each road is designed using net edit which is a sub tool of sumo. Figure 4.1 shows the overview of the four-way junction.

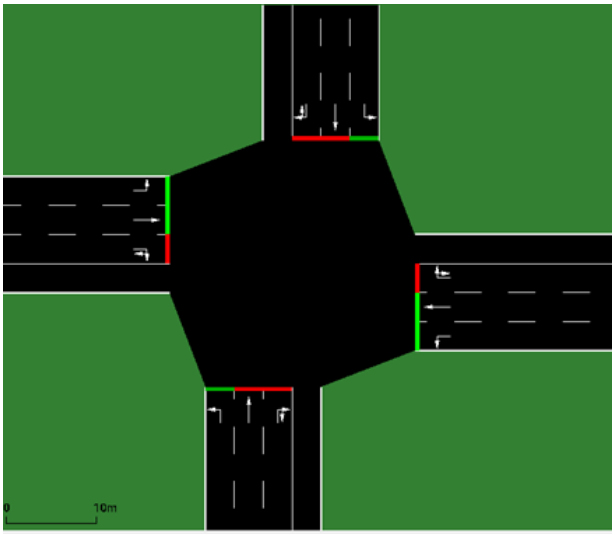


Figure 4.1: 4-way junction design by SUMO

Two types of vehicles are used in the simulation environment.

- Regular

<vType id="SUMO_DEFAULT_TYPE" accel="2.6" decel="4.5" sigma="1" length="4" minGap="2" maxSpeed="50"/>

- Emergency(ambulance,VIP vehicles etc)

<vType id="emergency" guiShape="emergency" accel="2.9" decel="4.5" sigma="1" length="5" minGap="2" maxSpeed="60"/>

Emergency vehicles have higher acceleration speed and max speed than regular vehicles.

4.2 Traci (Traffic Control Interface)

Traci is an API to access Sumo and retrieve values of simulated objects and to manipulate their behavior "on-line". Sumo and Traci are working as client-server architecture. In this case, sumo is working as the server and Traci is working as the client.

The following functions of Traci are used to get the data and manipulate objects.

- *traci.lane.getLastStepVehicleIDs(lane_id)*
  Returns the vehicle id in a given lane id.
- *traci.vehicle.getTypeID(vehicle_id)*
  Returns the type of the vehicle given the vehicle id.
- *traci.vehicle.getLanePosition(vehicle_id)*
  Returns the position of the vehicle given vehicle id.
- *traci.trafficlight.setPhase('0',2)*
  Used to change the phase of the signal light.
- *traci.simulationStep()*
  Proceed simulation by one timestep

4.3 Implementation of the Proposed Method

A popular machine learning library Keras is used as the main library with Tensorflow GPU as the backend using python 3.6.

4.3.1 Vehicle class

Defined a vehicle class to store information about each vehicle which will contain vehicle id, waiting time and type of the vehicle as attributes. Vehicles within 100m area from the junction are considered as the traffic data.

```
class Vehicle:
    def __init__(self,vid,waiting_time,vtype):
        self.vid=vid
        self.waiting_time=waiting_time
        self.vtype=vtype
```

4.3.2 Sumo Class

Class Sumo is defined to contain all the details about the simulation environment. Abstract of the sumo class as follows.

Attributes

- *RedYellowGreenStates* : A list to contain all the phase indexes of the traffic light.
- *Hyper parameters K and Z*: As discussed in section 3.2.3
- *nearby_ vehicles* : list to contain vehicle objects within 100m radius

Methods

- *updateWaitingTime()* : Update waiting time of nearby vehicles. This will add new coming vehicles to nearby_vehicles list concerning the lane index

```
#register the new vehicle
new_vehicle_obj=Vehicle(vehicle_id,0,traci.vehicle.getTypeID(vehicle_id))
nearby_vehicle_on_lane=self.nearby_vehicles[i-1][j]
nearby_vehicle_on_lane.append(new_vehicle_obj)
self.nearby_vehicles[i-1][j]=nearby_vehicle_on_lane
```

remove vehicles from the list which leaves the junction.

```
all_vehicles_on_lane = traci.lane.getLastStepVehicleIDs(str(i)+"si_"+str(j))

for vehicle in self.nearby_vehicles[i-1][j]:
    # removing the vehicle from the memory which has went through the juncti
    if vehicle.vid not in all_vehicles_on_lane:
        nearby_vehicle_on_lane=self.nearby_vehicles[i-1][j]
        nearby_vehicle_on_lane.remove(vehicle)
        self.nearby_vehicles[i-1][j]=nearby_vehicle_on_lane
```

If the vehicle is already in the list and still in the queue, it will increment the waiting time of that vehicle by 1s.

```
if vehicle.vid==vehicle_id:#checking for exsisting vehicles
    vehicle.waiting_time+=1
    vehicle_list=self.nearby_vehicles[i-1][j]
    # replace the old vehicle object with updated one
    vehicle_list[e]=vehicle
    self.nearby_vehicles[i-1][j]=vehicle_list
    flag=0
    break
```

This method is called for every time step.

- *getState()*: Return the current traffic state $\mathcal{M}$ and $\mathcal{P}$

generate $\mathcal{P}$ by one-hot encoding the phase index.

```
#onehot encode the phase index
current_phase = self.RedYellowGreenStates.\
index(traci.trafficlight.getRedYellowGreenState('0'))
num_classes = 8 # since we have 8 phases
target = np.array(current_phase)
phase_index = np.eye(num_classes)[target]#onehot encoding
```

calculating average and standard deviation of waiting time for each lane

```
#prepairing incomming state
emergency_waiting_time=0
waiting_time_distribution=[]
for vehicle in self.nearby_vehicles[i-1][j]:
    if vehicle.vtype == "emergency":
        if vehicle.waiting_time > emergency_waiting_time:
            emergency_waiting_time = vehicle.waiting_time
    else:
        waiting_time_distribution.append(vehicle.waiting_time)
        total_waiting_time_distribution.append(vehicle.waiting_time)

if not waiting_time_distribution:# append 0 to empty list to get rid
    waiting_time_distribution.append(0)

if(emergency_waiting_time):
    emg_waiting_time_distribution.append(emergency_waiting_time) #ap

waiting_time_mean =  np.mean(waiting_time_distribution)
waiting_time_std = np.std(waiting_time_distribution)
```

Prepare matrix $\mathcal{M}$. values are scaled to the range 0-1 to increase the effectiveness. When scaling two constraints are used. The first one is the maximum waiting time of a vehicle is 100s. The second one is the maximum number of vehicles that can be fit into the 100m range is 17.

```
# adding queue length
incomming_state[j+(i-1)*3][0] = len(self.nearby_vehicles[i-1][j])/17

if(waiting_time_mean>100): #adding average waiting time
    incomming_state[j+(i-1)*3][1] = 1
else:
    incomming_state[j+(i-1)*3][1] = waiting_time_mean/100

if(waiting_time_std>100): #adding standard deviation
    incomming_state[j+(i-1)*3][2] = 1
else:
    incomming_state[j+(i-1)*3][2] = waiting_time_std/100

if(emergency_waiting_time>100): # adding emergency waiting time
    incomming_state[j+(i-1)*3][3] = 1
else:
    incomming_state[j+(i-1)*3][3] = emergency_waiting_time/100
```

- *changePhase(next_phase_index)* : change the traffic signal phase to a new traffic phase given the new phase index. This will generate a yellow state given the current traffic phase and new traffic phase and it will first change the traffic signal to the corresponding yellow state for 3 seconds and then it will be changed to the new traffic state.

```
def changePhase(self,next_phase_index):

    current_phase=traci.trafficlight.getRedYellowGreenState('0') # 0-7
    next_phase=self.RedYellowGreenStates[next_phase_index]


    if(next_phase != current_phase):#generate yellow phase
        yellow_phase = ""
        for i in range(0, len(current_phase)):
            if(current_phase[i].lower()=="g" and next_phase[i].lower()=="r"):

                yellow_phase = yellow_phase+"y"
            else:
                yellow_phase = yellow_phase+current_phase[i]
        #set yellow phase
        traci.trafficlight.setRedYellowGreenState('0',yellow_phase)

        for i in range(3): # allow yellow phase for 3 seconds
            traci.simulationStep()
            self.updateWaitingTime()
    traci.trafficlight.setRedYellowGreenState('0',next_phase)
    return
```

- *Reward(current_ state_ distribution)*: Returns the reward for the current waiting time of vehicles.

```
def reward(self,current_state_distribution):

    #calculating the mean and the std of the mean of all the lanes for the current
    reg_waiting_time_mean =  np.mean(current_state_distribution[0])
    reg_waiting_time_std = np.std(current_state_distribution[0])

    emg_waiting_time_mean = np.mean(current_state_distribution[1])
    emg_waiting_time_std = np.std(current_state_distribution[1])


    reward = 50 - ((reg_waiting_time_mean + self.K*reg_waiting_time_std) \
                  + self.Z*(emg_waiting_time_mean + self.K*emg_waiting_time_std))
```

4.3.3 DQN Class

This class contains three methods.

- *load_model()* : Loads the saved model from the disk
- *save_model()* : Save the model to the disk
- *createDQN()* : Creates a new DQN and return the model

Keras functional API is used to implement the DQN model. Two input layers are defined. The first input layer is of dimensionality 12x5 to feed $\mathcal{M}$. Then it is passed through 3 fully connected layers (dense layers) each contains 50, 30, 20 neurons respectively. All three layers have the activation function of ''*relu*'' [47].

The second input layer is of dimensionality 1x8 to feed $\mathcal{P}$. Then it is passed through a fully connected layer which contains 20 neurons with activation ''*relu*''.

Then the two outputs are concatenated with a concatenation layer. After that concatenated output is passed through a fully connected layer of 16 neurons with activation ''relu''. Finally, the output layer contains 8 neurons which correspond to the phases with no activation function because we need to get the exact Q ($\mathbf{s_t}, \boldsymbol{a_t}$).

Adam optimizer[45] is used as the optimizer and mean squared error (mse) [46] is used to calculate the loss of the model when training.

```
def createDQN(self):
    traffic_State = Input(shape = (12,5))
    phase_index = Input(shape = (8,))

    Dense1_out = Dense(50,name="Dense1",activation='relu')(traffic_State)
    Dense2_out = Dense(30,name="Dense2",activation='relu')(Dense1_out)
    Flatten_out = Flatten()(Dense2_out)
    Dense3_out = Dense(20,name="Dense3",activation='relu')(Flatten_out)

    Dense4_out = Dense(20,name="Dense4",activation='relu')(phase_index)

    Concatination_layer = concatenate([Dense3_out,Dense4_out])
    Dense5_out = Dense(16,name="Dense5",activation='relu')(Concatination_layer)
    #no activation function for last layer
    Dense6_out = Dense(8,name="Dense6")(Dense5_out)

    model = Model(inputs=[traffic_State,phase_index], outputs=[Dense6_out])
    model.compile(optimizer="Adam",loss="mse")

    return model
```

4.3.4 Replay Memory Class

This class contains all the details about replay memory and it handles experience replay of the agent. It contains five attributes and two methods.

Attributes

- *capacity*: This defines the maximum number of experience tuples that can be stored in replay memory
- *replay_memory*: A list to contain experience tuples
- *push_count*: Contains the number of insertions were done to replay memory from the beginning.
- *buffer_state*: Used to hold the previous state
- *buffer_action*: Used to hold the previous action

```
class ReplayMemory:
    def __init__(self,capacity,initial_state,initial_action):
        self.capacity=capacity
        self.replay_memory=[]
        self.push_count=0
        self.buffer_state=initial_state
        self.buffer_action=initial_action
```

Methods

- *addToReplayBuffuer(self,current_ state,action,reward)* : This method will add new experience to experience replay memory.

  To record an experience, current state $\mathbf{s}_t$, current action $\mathbf{a}_t$, reward $\mathbf{r}_{t+1}$ and the next state $\mathbf{s}_{t+1}$ is needed.

  The problem with these kinds of environments is that the next state and the reward cannot be directly observed here because it takes some time for vehicles to move and actually become another state after making a decision. Which means that we have to buffer the current state $\mathbf{s}_t$ and current action $\mathbf{a}_t$ until the method is called again and adds that current state as the next state of the previous experience. Similarly, the current state in the second call is used to calculate the reward for the buffered state because the current state is the next state for the buffer state and action.

```
def addToReplayBuffuer(self,current_state,action,reward):#here the reward .

    experience=[self.buffer_state,self.buffer_action,reward,current_state]
    #buffer_state and buffer_action contains the previous state and the ac
    self.buffer_state=current_state
    self.buffer_action=action

    if(len(self.replay_memory)<self.capacity):
        self.replay_memory.append(experience)
    else:
            # replace the oldest  experience with the new one in replay me
        self.replay_memory[self.push_count%self.capacity]=experience
        self.push_count +=1
```

- *Replay (self,replay_ batch_ size):* This will handle the experience replay of the agent. The method will first check if the replay memory contains samples more than the batch size. If not it will simply return without going to experience replay.

```
if len(self.replay_memory) < replay_batch_size:
    return
```

  If true it will extract an experience batch form the replay memory and for each sample, it starts training.

```
replay_samples = random.sample(self.replay_memory, replay_batch_size)
for sample in replay_samples:

    state, action, reward, new_state = sample
```

Current Q-value extracted by passing the state variable to the PolicyNetwork which is the DQN.

```
prediction = PolicyNetwork.predict(state)
```

Then it will identify the best action of the new state by feeding the new state into the PolicyNetwork. After that, the target network is used to get the Q-value for the best action selected by the PolicyNetwork.

```
#with double network concept
best_action_new_state = np.argmax(PolicyNetwork.predict(new_state)[0])
#q value for the best action from the target network
Q_future = TargetNetwork.predict(new_state)[0][best_action_new_state]
```

Then the bellman optimality equation is used to calculate the Q target. After that, the PolicyNetwork will train on the experience.

```
target = prediction
target[0][action] = reward + Q_future * gamma
PolicyNetwork.fit(state, target, epochs=1, verbose=0)
```

4.3.5 Agent Class

Two versions of DQN are maintained (PoliciyNetwork and TargetNetwork) for the concept of Fixed Q-Targets as discussed in Section 3.2.7. This class handles the decision makings of the agent and maintaining the weights of the target DQN. It contains two attributes and two methods.

Attributes

- step_number : Represents the number of actions (signal changes) that the agent made so far.
- tau : This is a hyperparameter that decides the amount of the weight difference of the two DQNs will be added to the weights of the target DQN.

Methods

- getAction(self, state): Predict and output the next best signal phase given the current state using the PolicyNetwork. If first calculates the epsilon value and a random value between zero and one. If epsilon value is greater than the random number it outputs a random phase which is the concept of the exploration, else it predicts the next best signal phase using the Policy network. Epsilon will get decreased when the step_number getting lager. Epsilon has its minimum limit.

```
def getAction(self,state):
    epsilon = math.exp(-1.*self.step_number*epsilon_decay)
    epsilon = max(epsilon_min, epsilon)
    self.step_number+=1
    print(epsilon)
    if np.random.random() < epsilon:
        return np.random.randint(0,action_space)
    else:
        return np.argmax(PolicyNetwork.predict(state)[0])
```

- update_target_graph(self): Updates the target DQN weights with the Policy network by using tau.

```
def update_target_graph(self):
    weights = PolicyNetwork.get_weights()
    target_weights =TargetNetwork.get_weights()
    for i in range(len(target_weights)):
        target_weights[i] = self.tau * \
        (weights[i] - target_weights[i]) + target_weights[i]
    TargetNetwork.set_weights(target_weights)
```

4.3.6 Hyperparameter Settings

Below are the tuned values of the hyperparameters used.

```
#parameaters to be tuned
minimum_phase_duration=12
gamma = 0.999 # discount rate
epsilon_min = 0.01
epsilon_decay = 0.0003 #0.0003
learning_rate = 0.001
tau = 0.125
replay_memory_capacity = 10000
replay_batch_size=256
```

```
sumo_env=Sumo(K=.5,Z=1)
steps_per_episode=5000
```

4.3.7 Main Function

It creates two versions of DQN.

```
DQNet=DQN()
PolicyNetwork = DQNet.load_model()

PolicyNetwork.summary()
TargetNetwork = DQNet.load_model()
```

The main function will run the training of the agent for twenty episodes each which will contain 5000 steps. The simulation environment is reinitialized in each episode and taken into an intermediate traffic state before training.

```
while True:
    if(episode_no>20):
        break
    print("============== episode {} ================".format(episode_no))
    traci.start([sumo_env.getSumoBinary(gui=False), "-c",\
                 "cross3ltl.sumocfg", '--start'])
    initial_steps = 0
    while initial_steps < 100: # take initial environment to intermediat tr
        traci.simulationStep()
        initial_steps +=1
```

For each second it calls the *updateWaitingTime* () method of the *sumo* class to update the waiting times of vehicles and take action between every 12 seconds.

```
while step_count % steps_per_episode != 0:

        step_count += 1
        traci.simulationStep()
        sumo_env.updateWaitingTime()
        if(step_count % minimum_phase_duration == 0):
```

During each decision, it first obtains the current state using the *getState()* method in *sumo* class and then passes it to the *getAction (current_ state)* method in the agent class to get the next phase. It will then invoke the *chngePhase (action)* method of *sumo* class to change the traffic signal.

Then the reward is obtained which corresponding to the action and adds that to the replay memory. Experience replay is performed after each action.

```
if(step_count % minimum_phase_duration == 0):#agent will make a decision

    current_state,current_waiting_distribution=sumo_env.getState()
    action=agent.getAction(current_state)
    sumo_env.changePhase(action)
      print(current_state)
      print(current_state_distribution)
    reward,regular_waiting,regular_std,emergency_waiting,emergency_std= \
    sumo_env.reward(current_waiting_distribution)

    replayMemory.addToReplayBuffuer(current_state,action,reward)
    replayMemory.replay(replay_batch_size)#experience replay is performed
```

updating the target graph is performed 4 times per episode.

```
if(step_count % steps_per_episode/4 == 0):
    agent.update_target_graph()#
```

4.4 Implementation of the Average Only Method

The implementation of the average only method is almost identical to the proposed method. The only difference is it is not using the standard deviation in the state representation and reward function so the agent will be only rewarded by considering the average waiting time and state representation doesn't contain any information about the standard deviation of vehicle waiting times.

```
def reward(self,current_state_distribution):

    #calculating the mean and the std of the mean of all the lanes for the c
    reg_waiting_time_mean =  np.mean(current_state_distribution[0])
    reg_waiting_time_std = np.std(current_state_distribution[0])

    emg_waiting_time_mean = np.mean(current_state_distribution[1])
    emg_waiting_time_std = np.std(current_state_distribution[1])

    reward = 30 - ((reg_waiting_time_mean) + self.Z*(emg_waiting_time_mean))
```

4.5 Implementation of the Statistical Method

The method proposed by Binbin Zhou et.al [48] is implemented with bellow changes to support emergency facilitation and to match with the proposed method environment.

```
def __init__(self):
    self.GreenIndexes_list=[[4,1], [10,7], [2, 1], [11, 10], [5
    #indexes of the green lanes for each state
def getNextPhase(self,current_state):
    # this will output the indexes of lanes sorted by maximum
    # average waining time in decending order
    traffic_index=np.argsort((-current_state[0]), axis=0)[:,1]

    a=current_state[0][:,2]
    candidate_phases=[]

    try:
        # if emergency vehicle arriving on a lane give that
        # lane the priority
        max_traffic_lane=np.where(a==1)[0][0]
        print("emergency vehicle arrived")
        if max_traffic_lane in [0,9,3,6]:
            raise Exception() #if emergency vehicle is in left
```

If an emergency vehicle arriving on a lane this method will give that lane the highest priority. If there is no emergency vehicle it selects the phases which contain green for the lane which has the maximum traffic as candidate phases. Then among those candidate phases select the phase which contains green for the lane which has the second maximum traffic.

```
except:
    k=0
    while True:
        #avoid left turn lane
        if traffic_index[k] in [0,9,3,6]:
            k+=1
        else:
            max_traffic_lane=traffic_index[k]
            break

  print("max_traffic_lane : ",max_traffic_lane)
for i,GreenIndexes in enumerate(self.GreenIndexes_list):

    GreenIndexes_1=GreenIndexes.copy()
    if max_traffic_lane in GreenIndexes:
    # all signal phases which has green for max_traffic_lane
    #is a candidate for next phase
        GreenIndexes_1.remove(max_traffic_lane)
        candidate_phases.append \
        ({'index':i,'GreenIndexes':GreenIndexes_1[0]})
#there will be only 2 candidate phases for any green index
if np.where(traffic_index==candidate_phases[0]['GreenIndexes'])
>np.where(traffic_index==candidate_phases[1]['GreenIndexes']):
    return candidate_phases[1]['index']
else:
    return candidate_phases[0]['index']
```

4.6 Implementation of the Fixed Time Method

The traffic signal will change to the next phase after 30 seconds without considering any realtime traffic data. This will continue cyclically.

```
while step_count % steps_per_episode != 0:
    step_count += 1
    traci.simulationStep()
    sumo_env.updateWaitingTime()
    if(count % 30 == 0):# signal will change every 30 second
        sumo_env.changePhase(phase_index % 8)
        phase_index +=1
```

4.7 Implementation of vehicle detection API

When deploying the proposed method in the real world we need a way to detect vehicles in the images captured through surveillance cameras. It should also be able to

detect lanes and vehicle types. An initial approach to detect this information is implemented using Tensorflow Object Detection API. Refer to Appendix C.

Chapter 5

Training and Evaluation

This chapter evaluates the performance and success level of the proposed solution. The evaluation will undergo according to a proper evaluation model as follows using the simulation environment SUMO [30].

5.1 Evaluation Model

The evaluation model contains 2 major parts that are to evaluate with the synthetic dataset and to evaluate with the real-world dataset. For both the datasets, first, the performance of the proposed method (Avg + Std) Vs Avg only method is evaluated. As the next step, the proposed method will be evaluated with a fixed time method and with a user-defined traffic model (statistical method). All of the above comparisons will be done under low, moderate, high traffic scenarios and dynamic traffic scenarios. Figure 5.1 represents the diagram of the evaluation model.

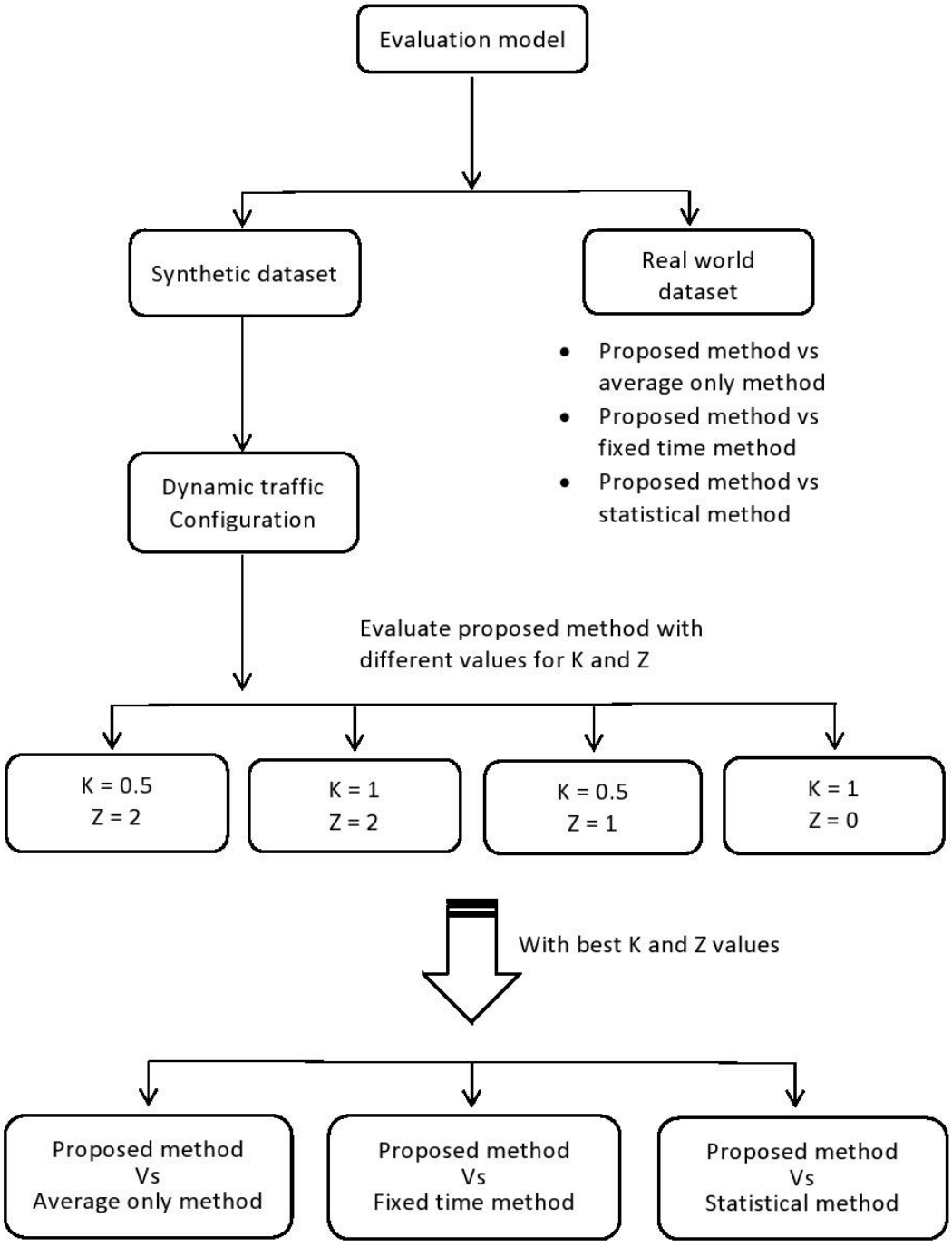


Figure 5.1: Evaluation model diagram

5.2 Evaluation Metrics

The following matrices are used to measure the performance of each method in the evaluation process.

1. **Average waiting time of regular vehicles** (AWT_r)

$$AWT_r = \frac{\sum_{i=0}^{n} W_{i,t}}{n}$$

Average waiting time of regular vehicles around 100m radius where the $\boldsymbol{n}$ is the total number of regular vehicles around 100m radius in a given time $\boldsymbol{t}$.

2. **Standard deviation of regular vehicle waiting time** (STD_r)

$$STD_r = \sqrt{\frac{\sum_{i=0}^{n}(AWT - W_{i,t})^2}{n}}$$

Standard deviation of regular vehicle waiting time around 100m radius where the $\boldsymbol{n}$ is the total number of regular vehicles around 100m radius in a given time $\boldsymbol{t}$.

3. **Average waiting time of emergency vehicles** (AWT_e)

$$AWT_e = \frac{\sum_{i=0}^{n} W_{i,t}}{n}$$

Average waiting time of emergency vehicles around 100m radius where the $\boldsymbol{n}$ is the total number of emergency vehicles around 100m radius in a given time $\boldsymbol{t}$.

4. **Standard deviation of emergency vehicle waiting time** (STD_e)

$$STD_e = \sqrt{\frac{\sum_{i=0}^{n}(AWT - W_{i,t})^2}{n}}$$

Standard deviation of emergency vehicle waiting time around 100m radius where the $\boldsymbol{n}$ is the total number of emergency vehicles around 100m radius in a given time $\boldsymbol{t}$.

5.3 Evaluation With Synthetic Dataset

The proposed method against the other three methods namely average only method statistical method and fixed time method are evaluated with the evaluation matrices mentioned in section 5.2 under dynamic traffic configuration which includes low, moderate, high traffic hours and which is also a combination of configurations used in [14], [37]. Sumo supports vehicle arrival approximation to Poisson Distribution by using the probability attribute of the vehicle flow tag. When the probability attribute is used it generates vehicles randomly with the given probability on each second. Table 5.1 shows the traffic flow settings for dynamic traffic configuration according to Poisson Distribution. In Table 5.1 SE is to mention South to East direction and similarly for N: North and W: West.

Table 5.1: Traffic flow settings according to Poisson Distribution

Vehicle type	Arrival rate (probability/sec)	Start time (s)	End time (s)
Regular	For all directions: 0.05	0	50000
	+		
	• NE, SE: 0.03	5000	9000
	• WE : 0.05	5000	10000
	• WS, ES: 0.03	15000	20000
	• NS: 0.05	15000	21000
	• WN, EN: 0.03	25000	30000
	• SN: 0.05	25000	35000
	• NW, SW: 0.03	40000	45000
	• SN: 0.05	42000	46000
Emergency	• WE: 0.01	0	10000
	• WS: 0.01	2000	10000
	• WN: 0.007	5000	20000
	• EW: 0.006	7000	15000
	• ES: 0.006	9000	50000
	• EN: 0.007	12500	50000
	• SW: 0.01	18000	20000
	• SE: 0.007	20000	50000

	• SN: 0.005	23000	25000
	• NW: 0.08	30500	50000
	• NE: 0.07	30500	50000
	• NS: 0.005	40000	50000

In table Table 5.1 For all directions: 0.05 means vehicles will be generated with a probability of 0.05 per second in each direction from 0s to 5000s of simulation time. At 5000s another vehicle flow of vehicle generation probability 0.03 is added to NE, SE direction so that the total arrival rate of NE, SE direction will be increased while other directions remain in the same single flow of probability 0.05 from 5000s to 9000s.

Distribution of the traffic volume with the 15 minute time intervals is shown in Figure 5.2 for each configuration. Rush hours (high traffic) are marked between red dotted lines.

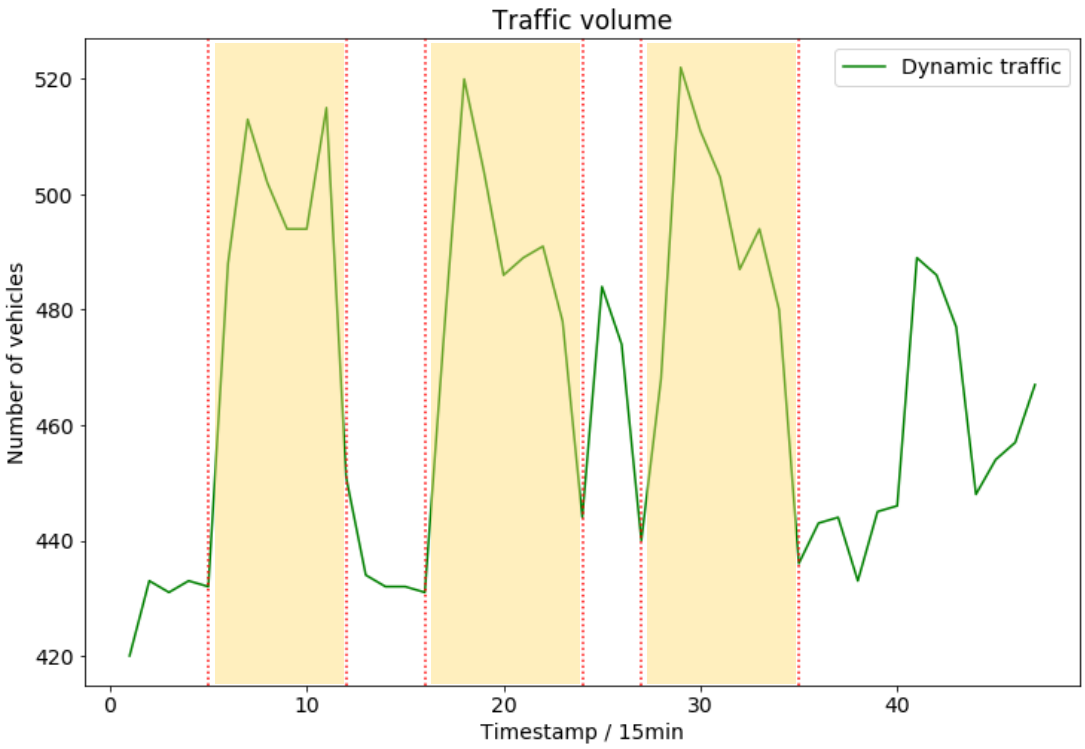


Figure 5.2: Distribution of the traffic volume for each configuration

Apart from traffic flow configurations, regular vehicles and emergency vehicles have different simulation settings as shown in Table 5.2 similar to settings used in [37].

Table 5.2: Vehicle simulation settings

Parameter	Regular Vehicle	Emergency Vehicle
Acceleration	2.6m/s	2.9m/s
Deacceleration	4.5m/s	4.9m/s
length	4m	5m
Minimum gap between vehicles	2m	2m
Maximum speed	50kmph	60kmph

The proposed method and avg only method is trained on 275h of simulated traffic in the SUMO environment under the implementation settings mentioned in section 4.3.

5.3.1 Evaluation For Different K's and Z's

Before evaluating with other methods the proposed method undergoes a self evaluation with different values K's and Z's to pinpoint the best values for K and Z. Below are the four different configurations for K and Z to the self evaluation.

1. K = 0.5, Z = 2 (low priority to std, high priority to emergency vehicles)
2. K = 1, Z = 2 (high priority to std, high priority to emergency vehicles)
3. K = 0.5, Z = 1 (low priority to std, medium priority to emergency vehicles)
4. K = 0.5, Z = 0 (low priority to std, no priority to emergency vehicles)

Figure 5.3, Figure 5.4, Figure 5.5, Figure 5.6 shows how each configuration was able to minimize the traffic congestion at the junction. In these figures, x-axis is the number of actions that the agent takes that means the number of signal changes (either to stay in the same phase or move to another phase). The agent will take action for every

12second intervals. More enlarged versions (starting from 3000 for the x-axis) of these figures can be found in Appendix A to get a clear view.

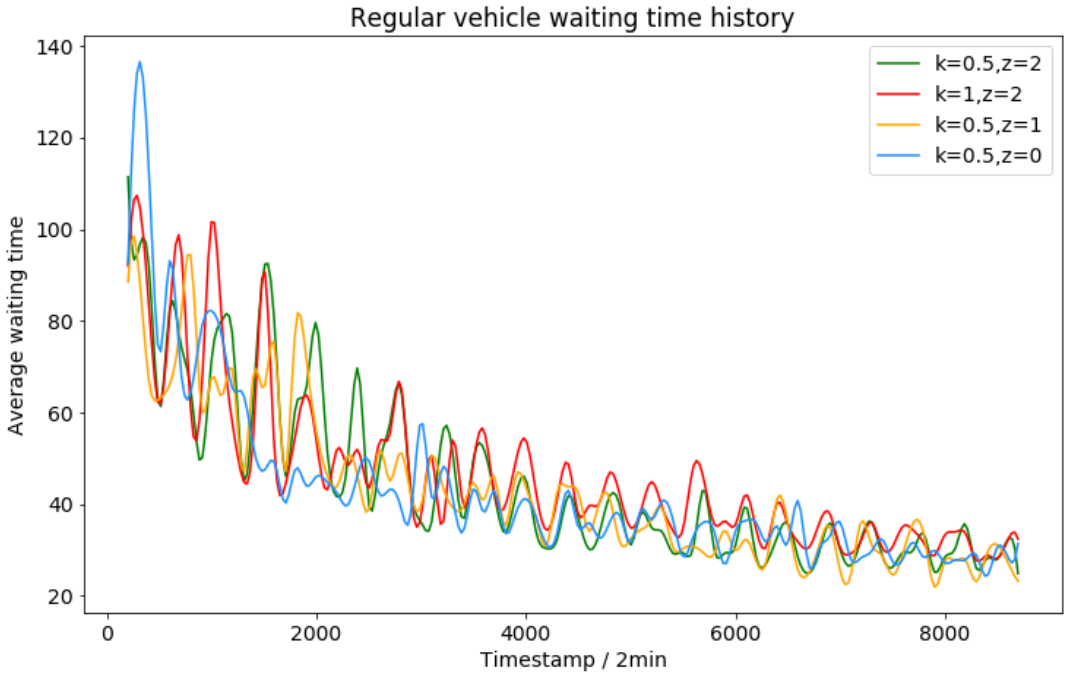


Figure 5.3: Regular vehicle average waiting time for different K and Z values

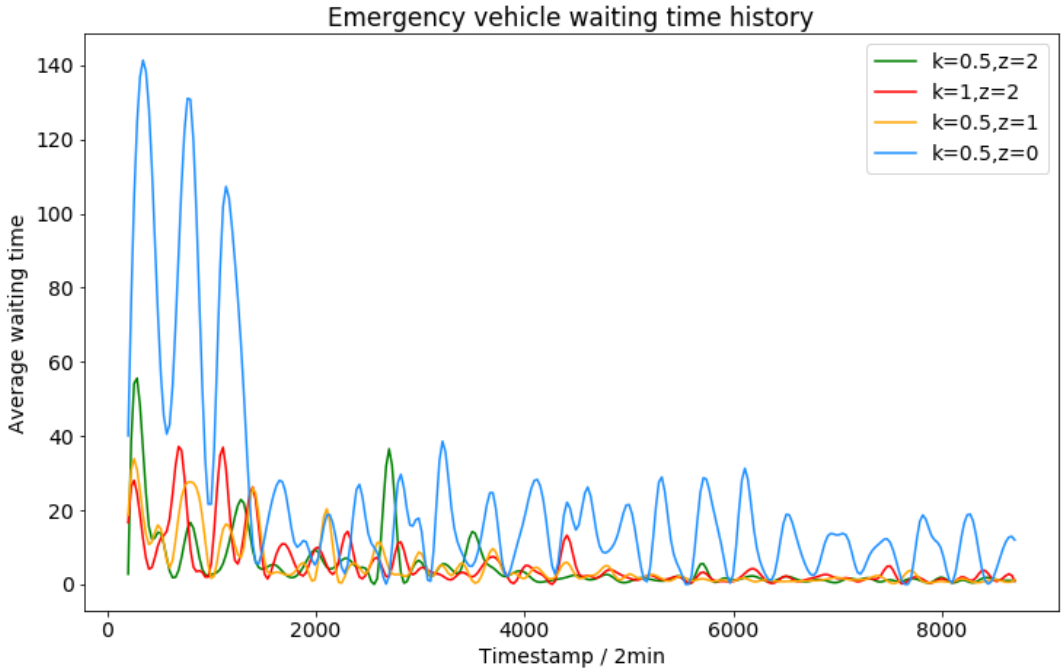


Figure 5.4: Emergency vehicle average waiting time for different K and Z values

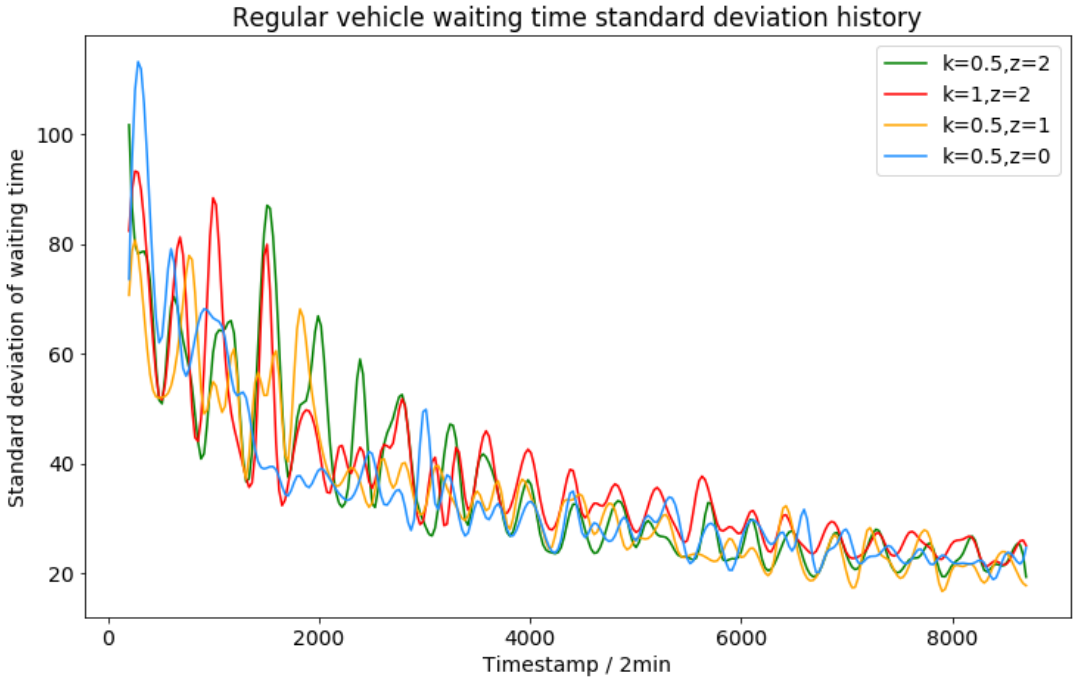


Figure 5.5: Regular vehicle standard deviation of average waiting time for different K and Z values

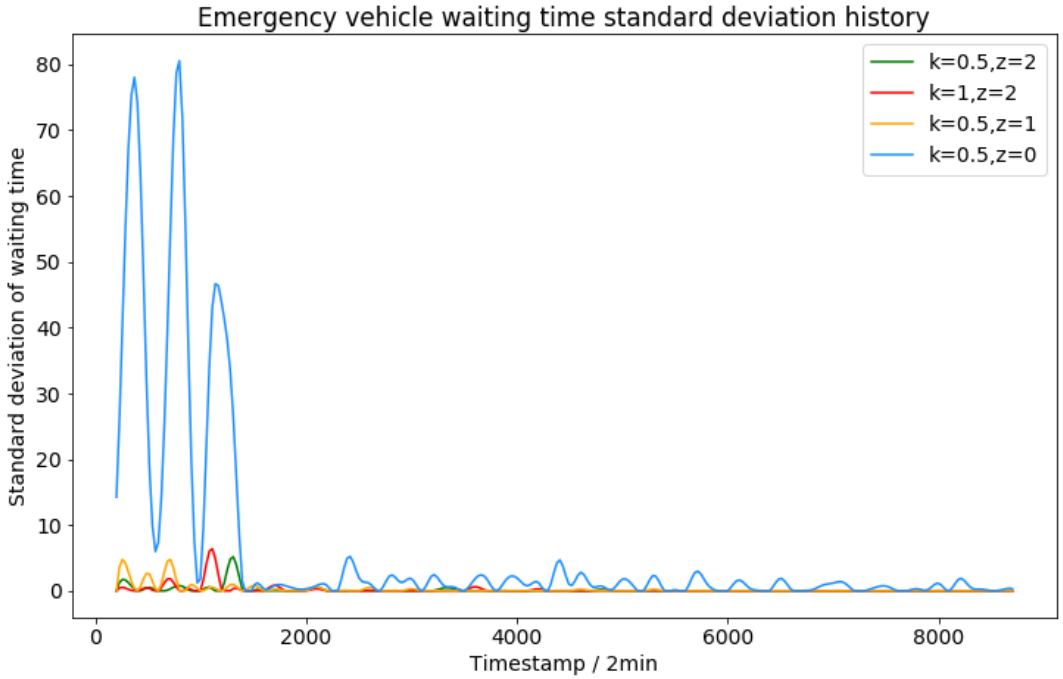


Figure 5.6: Emergency vehicle standard deviation of average waiting time for different K and Z values

According to Figure 5.3, Figure 5.4, Figure 5.5, Figure 5.6 Configuration 1,3,4 have shown almost similar results while configuration 2 shows poor results for the regular vehicles. Among configuration 1,3,4 the configuration 3 has shown slightly better results compared to other configurations. Configuration 4 was not able to minimize emergency vehicle waiting time as other configurations since Z is assigned zero. When Z is zero it does not consider the vehicle type. Table 5.3 shows the maximum, minimum and average of evaluation matrices of all the four configurations for the total training time.

Table 5.3: Performance for different configurations of K and Z

Evaluation matrix	K = 0.5 Z = 2	K = 1 Z = 2	K = 0.5 Z = 1	K = .5 Z = 0
Maximum of regular vehicle average waiting time (s)	111.388	106.427	94.023	136.100
Minimum of regular vehicle average waiting time (s)	24.843	27.733	21.980	24.199
Mean of regular vehicle average waiting time (s)	44.844	46.566	43.511	43.095
Maximum of emergency vehicle average waiting time (s)	52.520	37.450	28.110	131.88
Minimum of emergency vehicle average waiting time (s)	0.390	0.260	0.480	0.35
Mean of emergency vehicle average waiting time (s)	4.886	5.340	5.065	22.21

Maximum of regular vehicle std of waiting time (s)	101.738	91.742	79.503	113.293
Minimum of regular vehicle std of waiting time (s)	19.338	21.487	16.819	18.874
Mean of regular vehicle std of waiting time (s)	36.315	37.456	34.670	34.71
Maximum of emergency vehicle std of waiting time (s)	5.235	6.500	4.850	80.068
Minimum of emergency vehicle std of waiting time (s)	0	0	0	0
Mean of emergency vehicle std of waiting time (s)	0.121	0.142	0.216	6.013

From Table 5.3 it is clear that configuration 3 was able to minimize vehicle traffic congestion than the other three configurations. From here onwards configuration 3 is used to compare the proposed method with the statistical method, average only method, and fixed time method.

5.3.2 Training Evaluation

The proposed method and avg only method is trained on 500h of simulated traffic in the SUMO environment under configuration 3 which was found as the best value assignments for K and Z in section 5.3.1. These reinforcement learning methods are compared with the fixed time method and statistical method while training.

Figure 5.7 represents how each method was able to minimize the average waiting time of vehicles at the junction and Figure 5.8 represents how each method was able to minimize the standard deviation of waiting time at the junction.

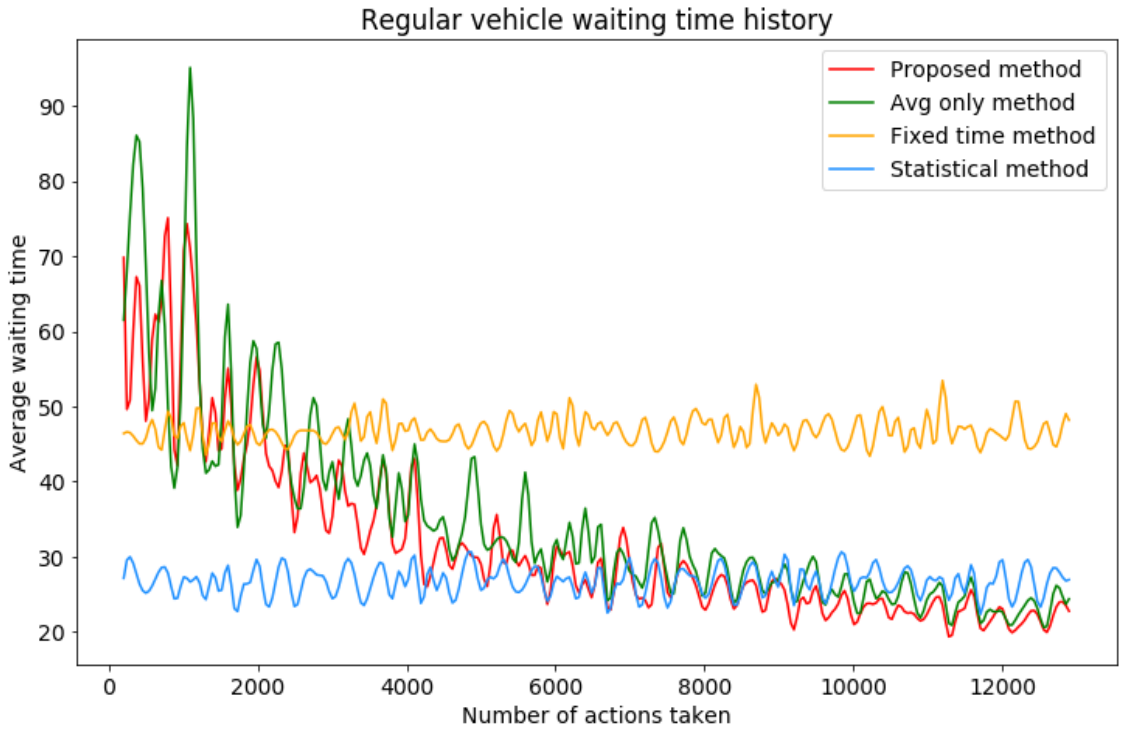
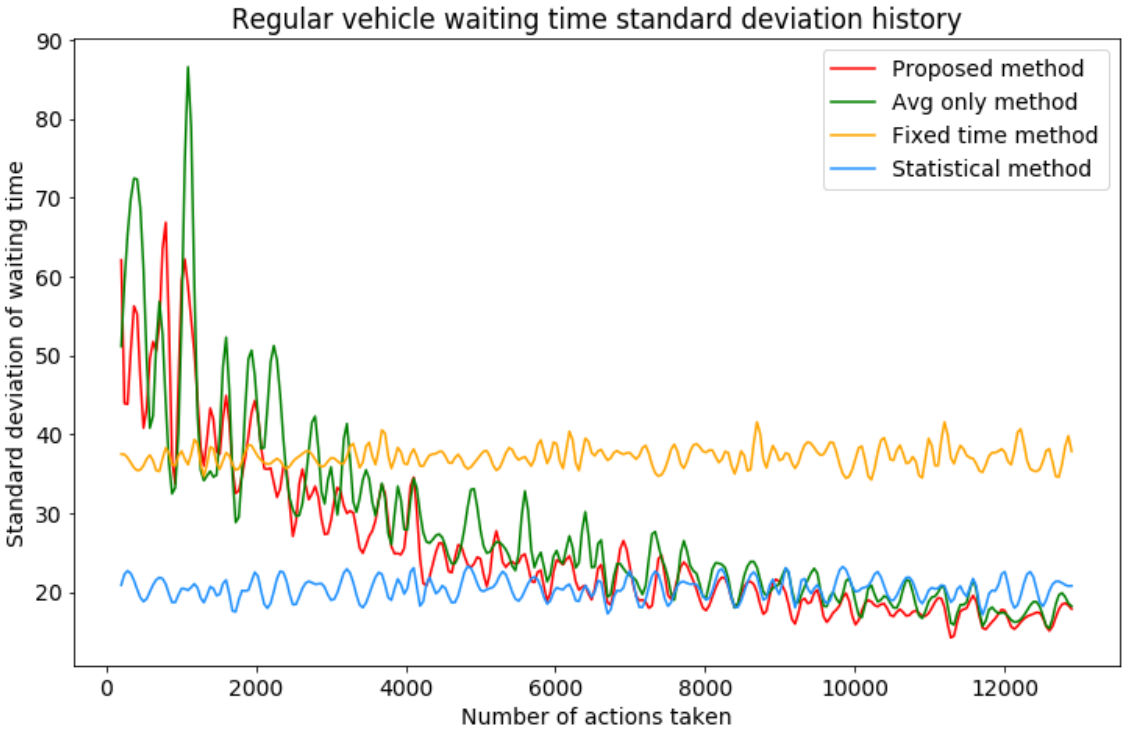


Figure 5.7: Learning to minimize average waiting time



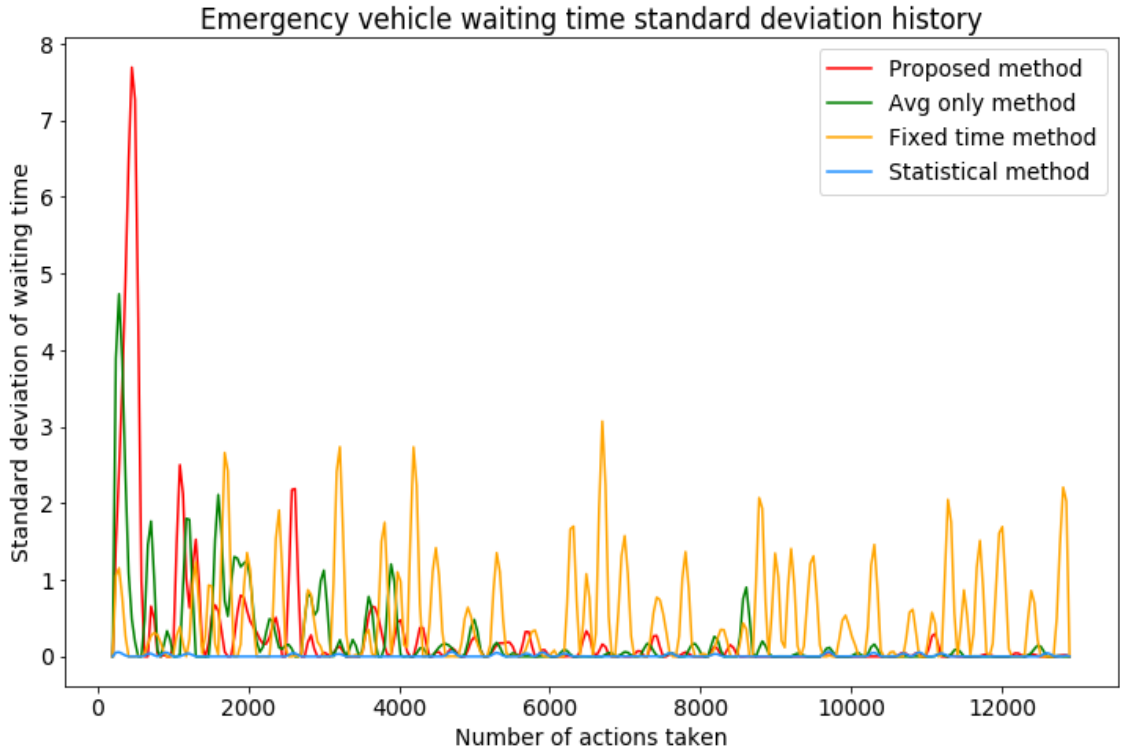


Figure 5.8:: Learning to minimize the standard deviation of waiting time

According to Figure 5.7 and Figure 5.8 initially, the reinforcement learning methods (proposed method and average only method) are not performing well since they are not aware of the environment at the beginning. But with the help of exploration and exploitation (section 1.1.2) reinforcement learning agents were able to learn how to minimize the average waiting time and the standard deviation of waiting times. Reinforcement learning methods perform even worse than the fixed time method at the beginning but at half of the training time, they were easily able to outperform the fixed time method. The statistical method was performing well from the beginning but with the time reinforcement learning methods were able to outperform the statistical method as well.

However, the average only method shows only small improvement over statistical method but the proposed method shows better results than every method at the end buy reducing the average waiting time and the standard deviation.

Figure 5.9 represents how reinforcement learning methods were able to improve the reward with time. Since there is no learning for the fixed time method and statistical method those methods were not able to improve the reward with the time. In Appendix A Figure A.5 shows how all four methods gain the reward with the time while Figure 5.9 only shows the reinforcement learning methods for clear comparison.

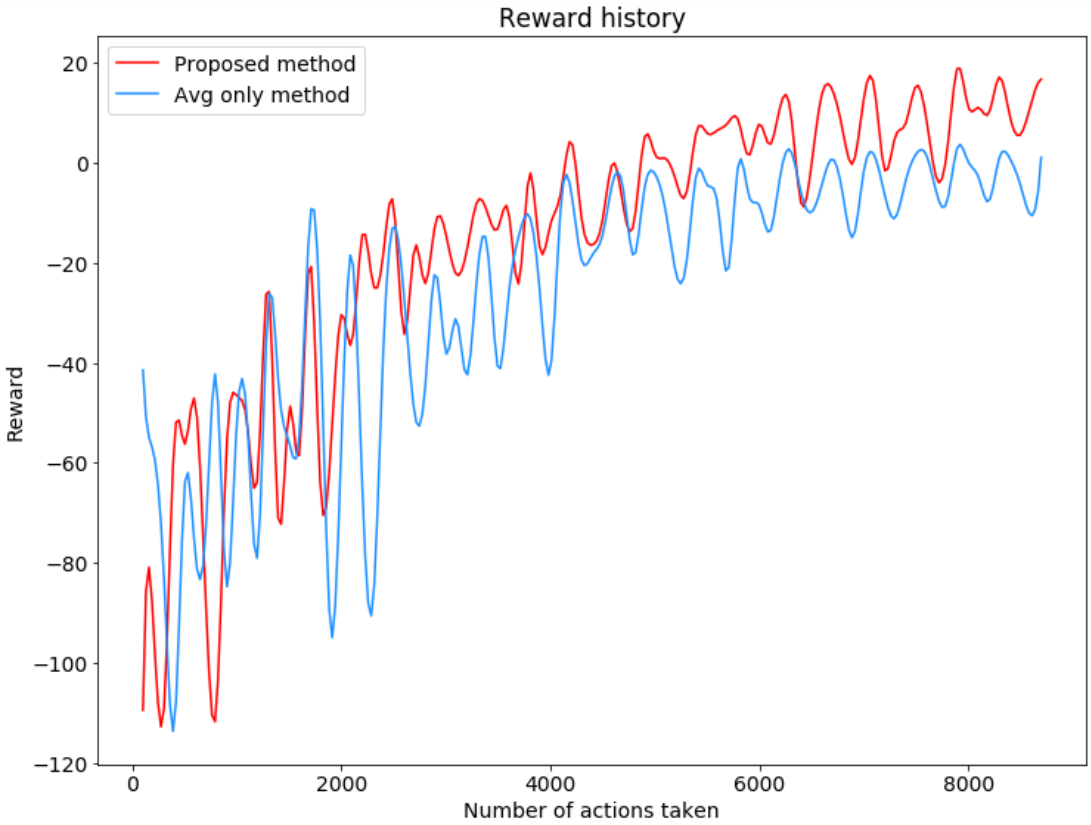


Figure 5.9: Reward improvement of the proposed method and average only method

Both the reinforcement learning methods initially get very lower minus rewards but with the time both of the algorithms learned to receive a higher reward. The proposed method outperforms the average only method in the 2000'th action and there onwards it continues to improve well than the average only method.

5.3.3 Testing Evaluation

5.3.3.1 Average Waiting Time

Figure 5.10 shows how each method controlled regular vehicle traffic for 14hrs. Peaks of the lines show the rush hours and pits show the low traffic time while in between them is the moderate traffic time. The X-axis is the two-minute time stamps which the average waiting time is calculated for. One timestamp is the average waiting time of vehicles for two minutes. And y-axis shows the average waiting time of the vehicles. Graphs are smoothed using the smoothing factor of 300.

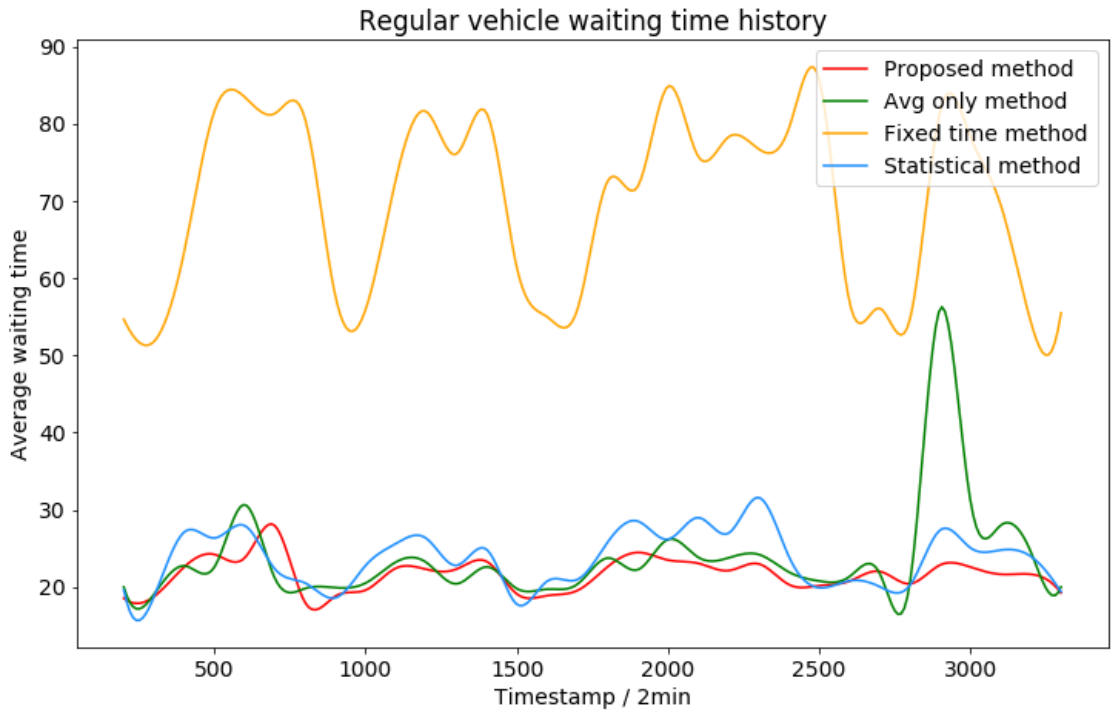


Figure 5.10: Average waiting time control of regular vehicles by each method in the testing environment

According to Figure 5.10 Fixed time method has very poor performance compared to the other three methods. The average only method and proposed method have similar performance but the proposed method outperformed the average only method most of the time. The average only method failed to efficiently control traffic around the 3000th timestamp. There are some cases that the statistical method was able to reduce average waiting time than the proposed method, but most of the time, the proposed method was able to minimize the average waiting time than the other three methods.

Figure 5.11 shows how each method controlled emergency vehicle traffic for 14hrs. Since the fixed time method producing bad results and average only method fails at 3000th timestamp Figure 5.12 shows the performance comparison without the fixed time method and the average only method for the clear comparison.

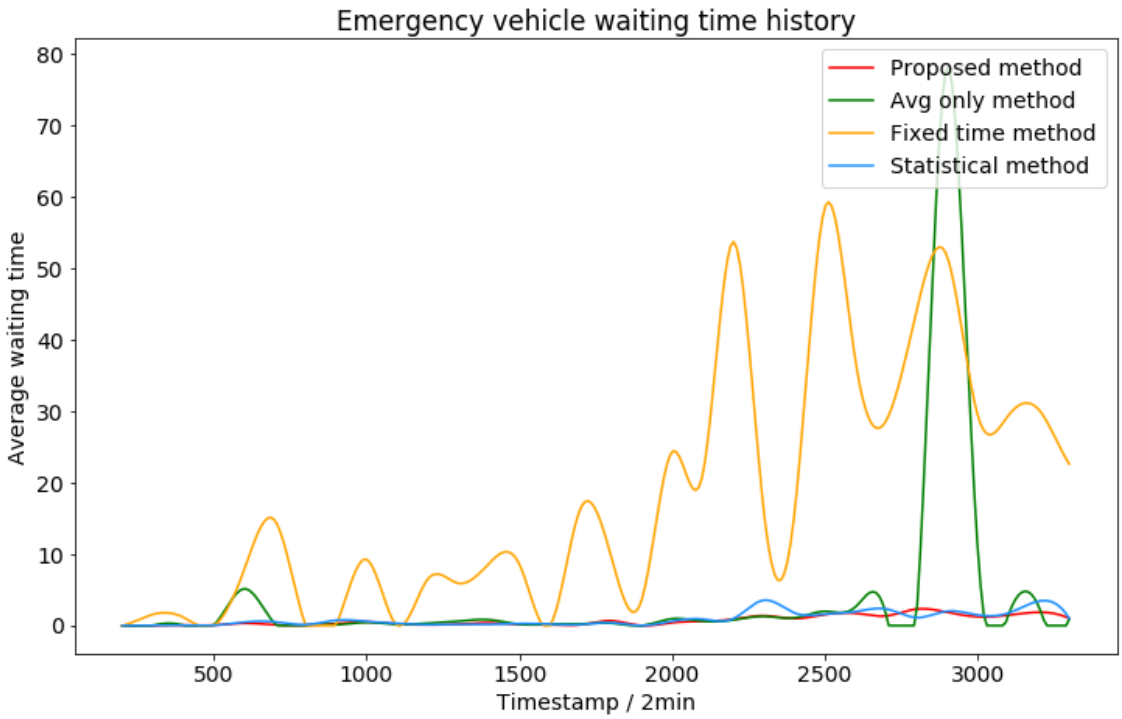


Figure 5.11: Average waiting time control of emergency vehicles by each method in the testing environment

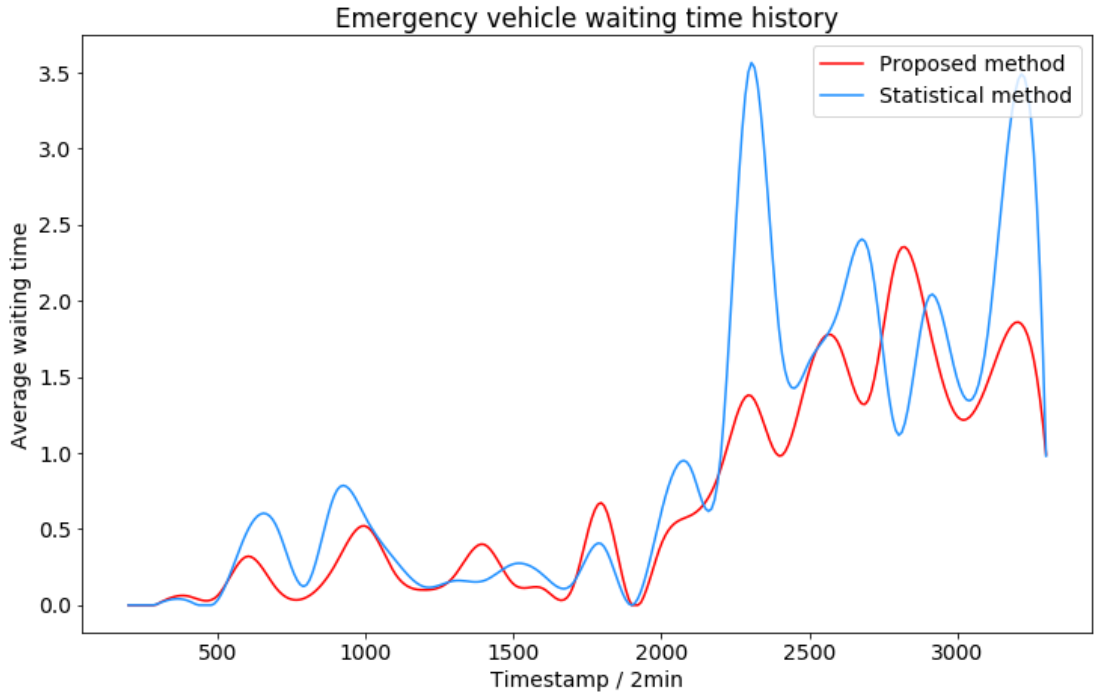


Figure 5.12: Average waiting time control of emergency vehicles by the proposed and the statistical method in the testing environment

According to Figure 5.12, both the methods are performing well on emergency vehicles because all of the methods were able to maintain the average waiting time of emergency vehicles under four. Overall the proposed method performed well than the statistical method.

5.3.3.2 Standard Deviation

Figure 5.13 shows how each method controlled regular vehicle traffic for 14hrs. The X-axis is the two-minute timestamps in which the standard deviation of average waiting time is calculated. One timestamp is the standard deviation of the average waiting time of vehicles for two minutes. And y-axis shows the average waiting time of the vehicles.

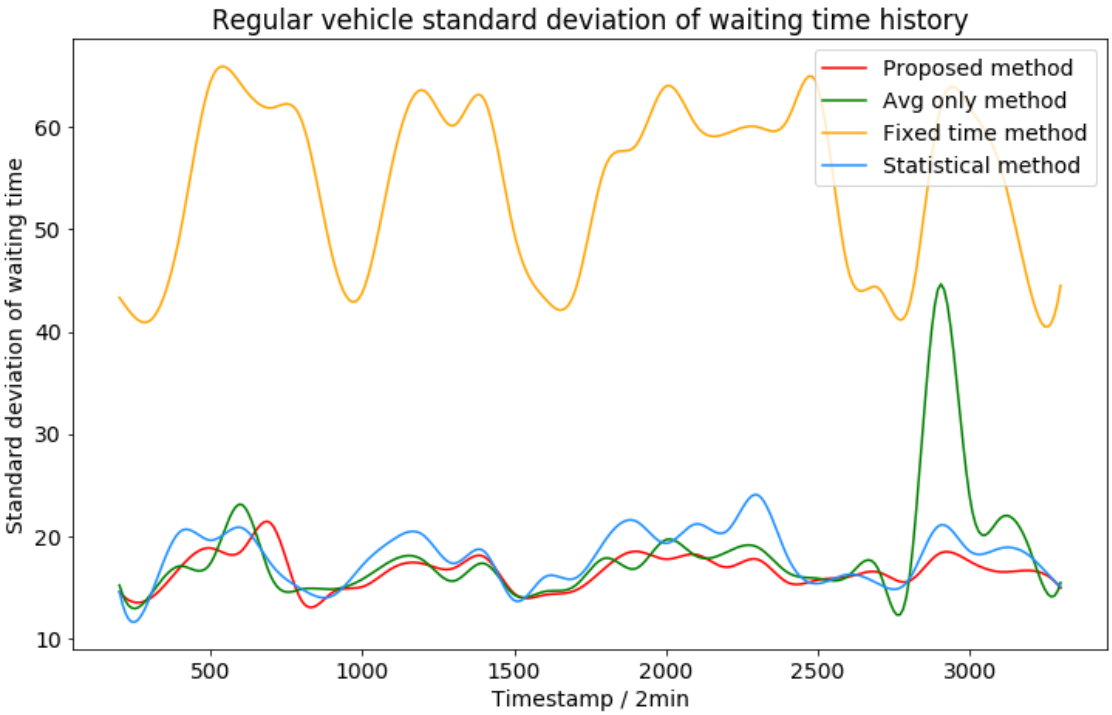


Figure 5.13: Standard deviation control of regular vehicles by each method in the testing environment

Figure 5.13 is very similar to Figure 5.10 that observed for average waiting time. All four methods performed in the exact way that they performed on average waiting time. That demonstrates the proposed method is not only reducing the average waiting time but it also able to reduce the standard deviation of waiting times which guarantees that all the vehicles will have a waiting time which is close to average waiting time.

Figure 5.14 shows how each method controlled emergency vehicle traffic for 14hrs. Since the fixed time method producing bad results average only method fails at 3000th timestamp Figure 5.15 shows the performance comparison without the fixed time method and average only method.

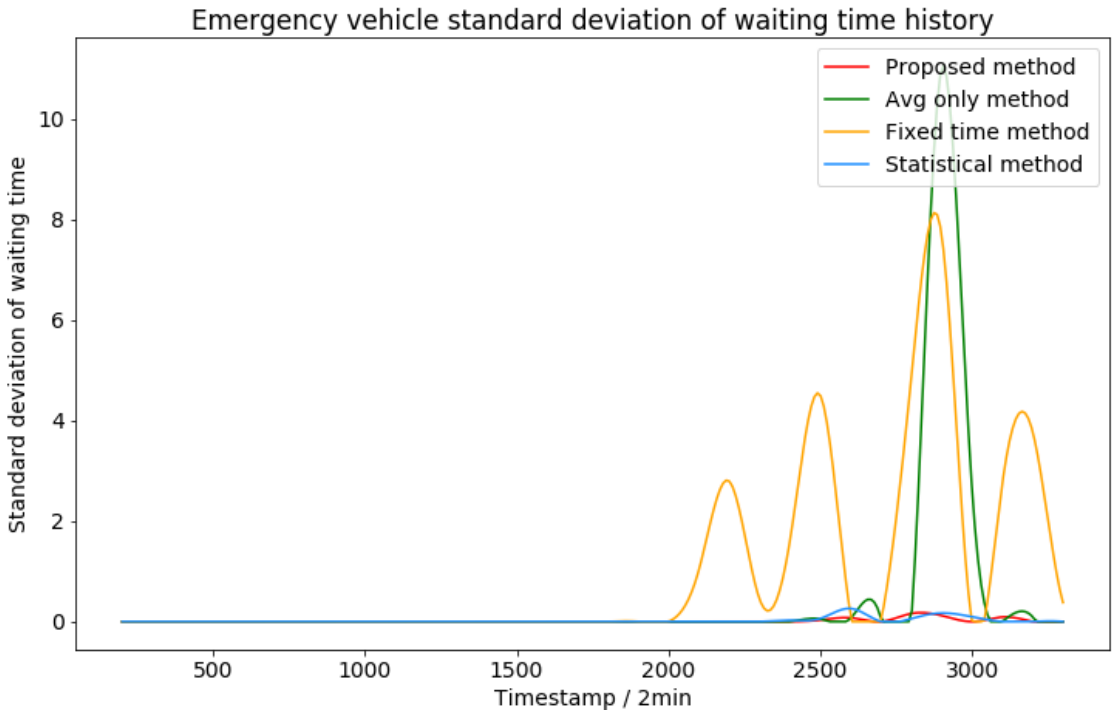


Figure 5.14: Standard deviation control of emergency vehicles by each method in the testing environment

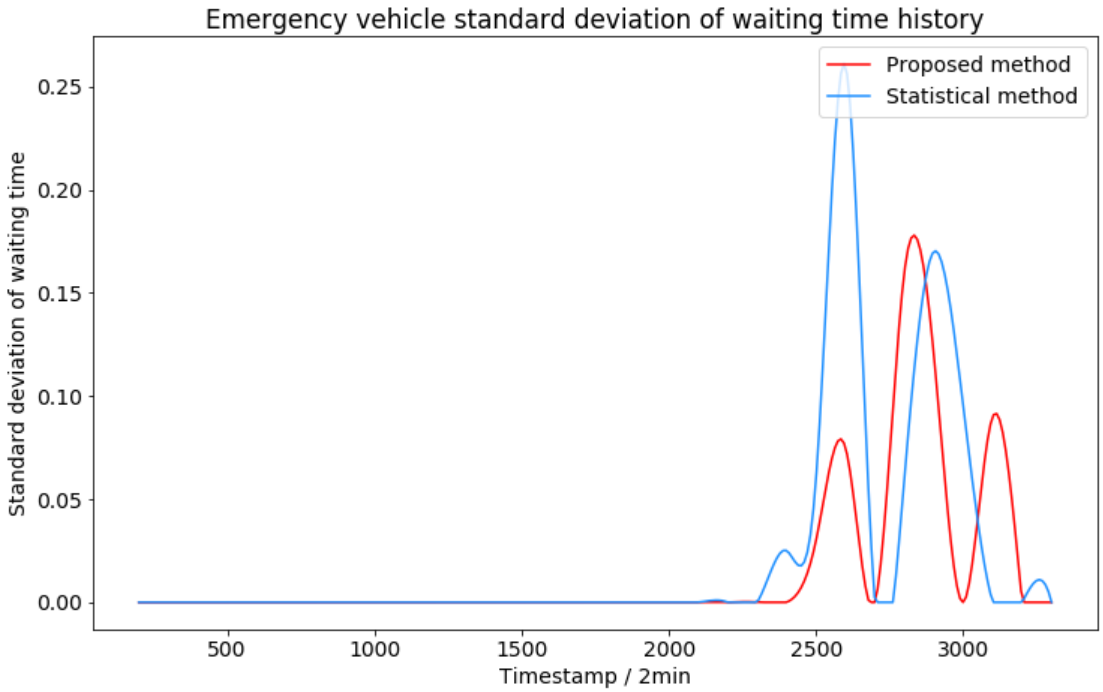


Figure 5.15: Standard deviation control of emergency vehicles by proposed,avg only and statistical method in the testing environment

5.3.3.3 Testing Summary

Table 5.4 shows the maximum, minimum and average of evaluation matrices of all the four methods for the total simulation time (14 hrs).

Table 5.4: Performance summary for evaluation matrices

Evaluation matrix	Proposed method	Average only method	Fixed time method	Statistical method
Maximum of regular vehicle average waiting time (s)	27.941	56.059	85.730	30.594
Minimum of regular vehicle average waiting time (s)	17.989	19.064	51.797	16.781

Mean of regular vehicle average waiting time (s)	21.588	23.631	69.597	22.652
Maximum of emergency vehicle average waiting time (s)	2.31	78.31	58.545	3.55
Minimum of emergency vehicle average waiting time (s)	0	0	0	0
Mean of emergency vehicle average waiting time (s)	0.684	3.609	17.391	0.896
Maximum of regular vehicle std of waiting time (s)	21.270	44.457	64.388	23.092
Minimum of regular vehicle std of waiting time (s)	13.756	14.259	41.049	12.780
Mean of regular vehicle std of waiting time (s)	16.579	18.026	54.326	16.938
Maximum of emergency vehicle std of waiting time (s)	0.16	10.99	7.7	0.26
Minimum of emergency vehicle std of waiting time (s)	0	0	0	0
Mean of emergency vehicle std of waiting time (s)	0.014	0.436	0.935	0.020

As summarized in Table 5.4 the proposed method outperforms all other methods on most of the cases and all the mean cases as highlighted in the table.

5.4 Evaluation on real-world dataset

Realworld traffic volume data set is available for download which contains traffic data around junctions in Canada [36]. Each row of the dataset contains the following attributes.

- Intersection ID
- Intersection name
- Datetime_bin: Traffic volume is calculated for 15 min intervals
- Classification: Vehicle type
- Leg: Vehicles approaching direction
- Dir: Vehicles headed direction
- Volume: Number of vehicles observed within 15 minutes

Unfortunately, there are no emergency type vehicles in the dataset. So the evaluation with real-world data is only possible for regular vehicles. This real-world dataset is embedded to SUMO using a python script which converts the dataset to an XML format that the SUMO supports. The implementation of the python script can be found in Appendix B.

Figure 5.16 and Figure 5.18 show the average waiting time and the standard deviation of average waiting time for each method. Since the fixed time method not performing well Figure 5.17 and Figure 5.19 represent graphs without the fixed time method.

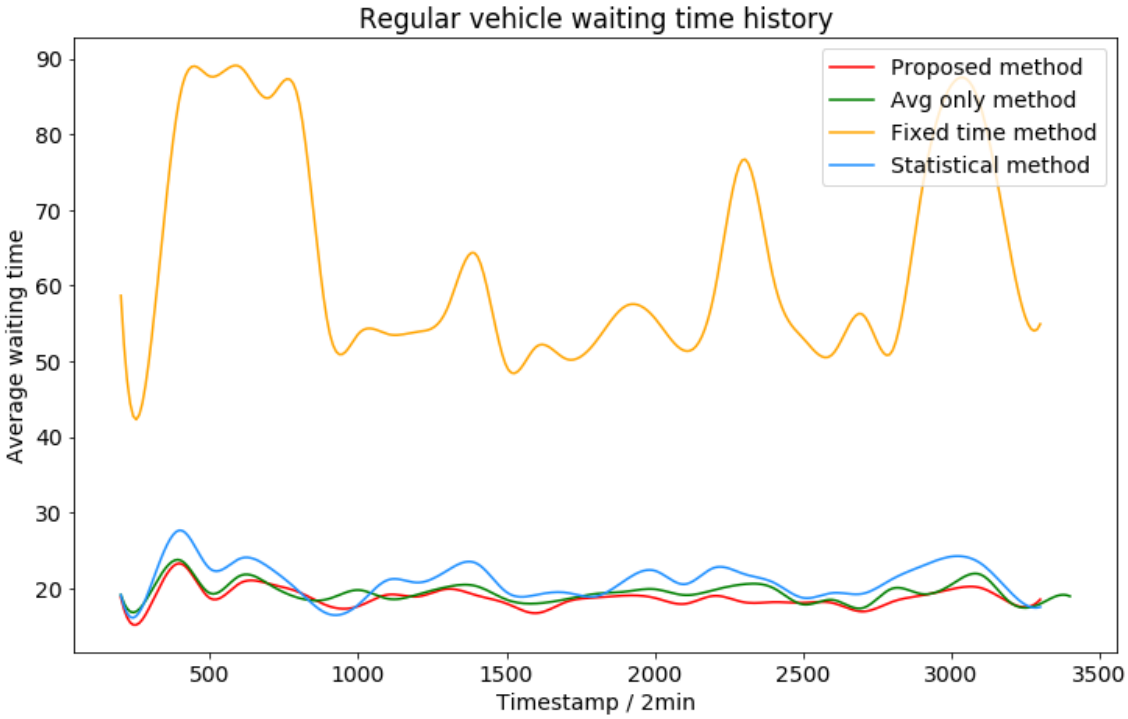


Figure 5.16: Average waiting time control on real-world data

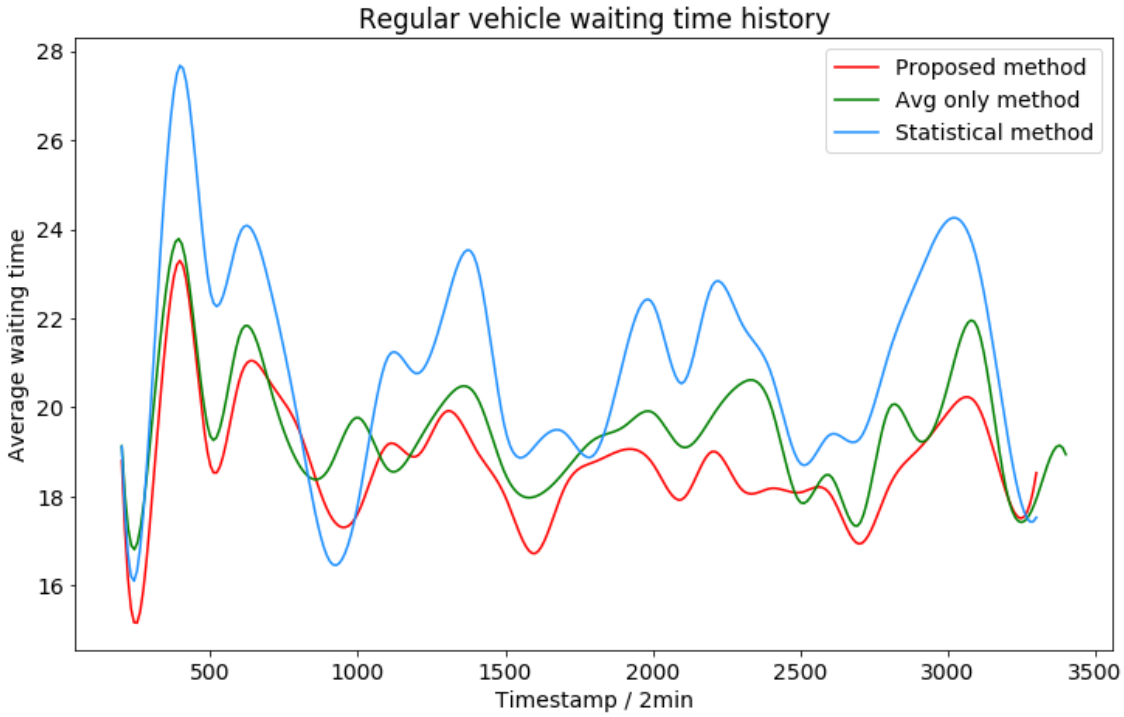


Figure 5.17: Average waiting time control on real-world data without fixed time method

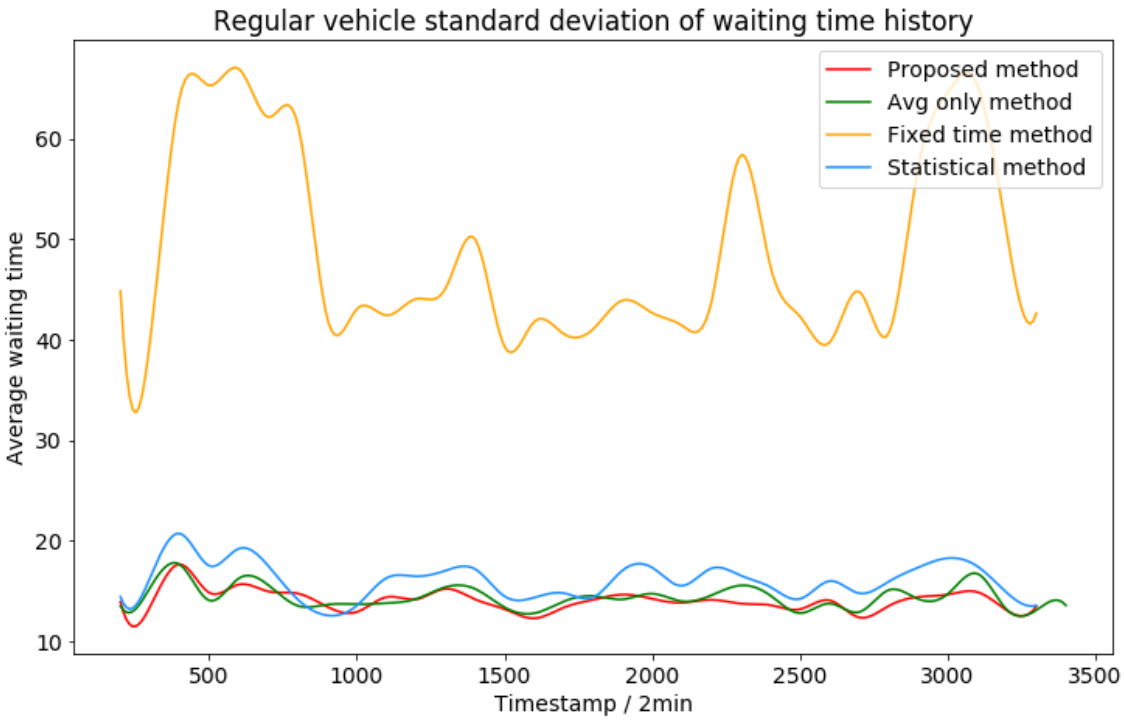


Figure 5.18: Standard deviation of waiting time control on real-world data

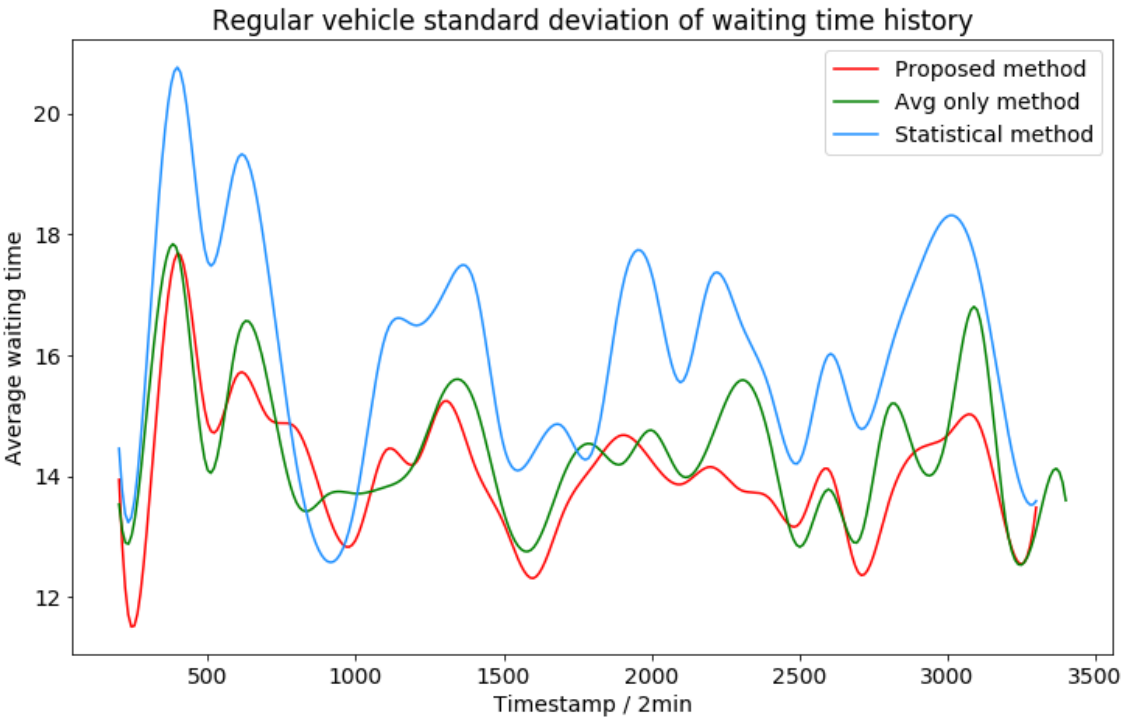


Figure 5.19: Standard deviation of waiting time control on real-world data without fixed time method

Unlike synthetic data, the proposed method and average only method performed better than the statistical method. According to Figure 5.17 and Figure 5.19 it is clear that the proposed method was able to perform well in real-world traffic data with the help of standard deviation of vehicle waiting times than the average only method.

Table 5.5: Performance summary for evaluation matrices on the real-world dataset

Evaluation matrix	Proposed method	Average only method	Fixed time method	Statistical method
Maximum of regular vehicle average waiting time (s)	23.281	23.728	88.974	27.679
Minimum of regular vehicle average waiting time (s)	16.720	17.377	49.337	16.603
Mean of regular vehicle average waiting time (s)	18.787	19.474	63.005	20.942
Maximum of regular vehicle std of waiting time (s)	17.689	17.681	66.957	20.751
Minimum of regular vehicle std of waiting time (s)	12.314	12.815	39.336	12.607
Mean of regular vehicle std of waiting time (s)	14.101	14.394	48.737	16.041

According to Table 5.5 which contains the overall summary of the evaluation metrics, Highlighted values are the matrices in which the proposed method performed better than other methods. The statistical method was able to maintain a lower minimum regular vehicle average waiting time than the proposed method while the average only method was able to maintain a lower maximum of regular vehicle standard deviation of waiting time. Apart from those two instances, the proposed method has the best performance than the other methods including all the mean cases. From all the above facts, it is clear that the proposed method outperforms the statistical method, average only method and fixed time method with the help of standard deviation.

Chapter 6

Discussion and Conclusion

6.1 Conclusion

This research is mainly focused on evaluating the significance of considering vehicle's standard deviation of waiting time around the junction to effectively control the traffic. The proposed RL agent was designed with a state representation that contains the number of vehicles, the average waiting time of vehicles, the waiting time standard deviation of vehicles and vehicle type for each lane with the current signal phase index. The agent's action is to select the next signal phase from the predefined traffic phase pool which ensures the safe green signal combinations. The agent is rewarded on four factors which are the average waiting time of regular vehicles, the average waiting time of emergency vehicles, the standard deviation of regular vehicle waiting time and the standard deviation of the emergency vehicle waiting time.

The proposed method is evaluated on both synthetic and real-world dataset using the SUMO simulation environment. As against method to evaluate the proposed method, a different version of the proposed method is implemented without considering the standard deviation. Evaluation results have proven that by using standard deviation it was able to minimize traffic congestion than average only method. This concludes that considering standard deviation has a positive impact. Furthermore, both RL methods

were able to outperform the statistical method and the fixed time method on the real-world dataset.

The second main research question is a way to provide emergency facilitation. For that, the vehicle type is embedded in state representation and it has shown that the proposed method was capable of prioritizing emergency vehicles and approximate emergency vehicle waiting time to zero while the method which is not considering the vehicle type has not performed well. Which concludes considering the type of vehicle has a significant value.

The third research question is a way to use standard deviation for effectively rewarding the agent. Result evaluation has stated that the proposed agent was able to learn well and outperform the other methods with the help of the proposed reward function which considers both the average waiting time and the standard deviation when rewarding the agent.

6.2 Limitations

The proposed method failed to maintain a lower maximum of the standard deviation of the regular vehicle waiting time when compared to the average only method on the real-world dataset. More training and expanding the DQN hidden layers can be a solution to this limitation. The learning of the agent to prioritize the emergency vehicles is a slow process since the arrival of an emergency vehicle to the junction is a rare case. This can be solved using the prioritized experience replay which pays more attention to the important experiences.

6.3 Future Work

When come to real-world the coordination between traffic junctions will be important for global traffic congestion minimization. Therefore a coordinated multi-agent scenario can be explored considering standard deviation with emergency facilitation. Furthermore, there can be more useful information to represent the traffic state such as vehicle position and velocity. But when considering those types of information one

should pay attention to whether that information is efficiently capturable or not. Treating intelligent traffic light control as a sequence problem is another area of exploration where one can use a sequence of traffic states as the input to the agent. Though the model uses the congestion level of the outgoing lane when making a signal change, it is not tested on those kinds of traffic conditions where the outgoing lane is congested. Training and testing agents on those traffic conditions will add a novel contribution to the domain. The model-free Reinforcement Learning is another type of RL which does not need to model the transition function (non-deterministic state transition) and the reward function. Evaluating the Effectiveness of standard deviation with a model-free RL agent is another area to explore.

References

[1] Richard S Sutton, Andrew G Barto. 2018. Reinforcement learning: An introduction. MIT press.

[2] R. Bellman, "A markovian decision process," tech. rep., DTIC Document, 1957.

[3] Obadah M.A Ayesh, Venus W. Samawi, and Jehad Q. Alnihoud."Traffic light control utilizing queue length".Proceedings of the World Congress on Engineering 2014 Vol I, WCE 2014, July 2 - 4, 2014, London, U.K.

[4] M. H. A. Ilmudin, N. M. Z. Hashim, A. S. Ja'afar, A. Salleh, A. Jaafar, M. F. M. Sam."Traffic Light Control System using 434 MHz Radio Frequency".2014.

[5] Malik Tubaishat, Yi Shang and Hongchi Shi."Adaptive Traffic Light Control with Wireless Sensor Networks".2007.

[6] S. Mikami and Y. Kakazu, "Genetic reinforcement learning for cooperative traffic signal control," in Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on. IEEE, 1994, pp. 223–228.

[7] B. Abdulhai, R. Pringle, and G. J. Karakoulas, "Reinforcement learning for true adaptive traffic signal control," Journal of Transportation Engineering, vol. 129, no. 3, pp. 278–285, May 2003.

[8] Y. K. Chin, N. Bolong, A. Kiring, S. S. Yang, and K. T. K. Teo, "Q-learning based traffic optimization in management of signal timing plan," International Journal of Simulation, Systems, Science & Technology, vol. 12, no. 3, pp. 29–35, June 2011.

[9] I. Arel, C. Liu, T. Urbanik, and A. Kohls, "Reinforcement learning-based multi-agent system for network traffic signal control," IET Intelligent Transport Systems, vol. 4, no. 2, pp. 128–135, June 2010.

[10] P. Balaji, X. German, and D. Srinivasan, "Urban traffic signal control using reinforcement learning agents," IET Intelligent Transport Systems, vol. 4, no. 3, pp. 177–188, September 2010.

[11] Xiaoyuan Liang, Xusheng Du, Guiling Wang, Zhu Han."Deep Reinforcement Learning for Traffic Light Control in Vehicular Networks".2018.

[12] Ying Liu, Lei Liu, Wei-Peng,"Intelligent Traffic Light Control Using Distributed Multi-agent Q Learning" Chen Fujitsu Laboratories of America, Inc., Sunnyvale, CA, USA Electrical and Computer Engineering, Rutgers University, New Brunswick, NJ, USA 2017.

[13] S.T.Rakkesh, A.Ruvan Weerasinghe, R.A.Chaminda Ranasinghe."Traffic Light Optimization Solutions using Multimodal,Distributed and Adaptive Approaches". International Conference on Advances in ICT for Emerging Regions (ICTer) 2015 .

[14] Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. 2018. IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control. In ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD). 2496–2505.

[15] Mohammad Aslani, Mohammad Saadi Mesgari, and Marco Wiering. 2017. Adaptive traffic signal control with actor-critic methods in a real-world traffic network with different traffic disruption events. Transportation Research Part C: Emerging Technologies 85 (2017), 732–752.

[16] Samah El-Tantawy, Baher Abdulhai, and Hossam Abdelgawad. 2013. 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 (2013), 1140–1150.

[17] Noe Casas. 2017. Deep deterministic policy gradient for urban traffic light control. arXiv preprint arXiv:1703.09035 (2017).

[18] Van der Pol et al. 2016. Coordinated Deep Reinforcement Learners for Traffic Light Control. NIPS.

[19] Bram Bakker, Shimon Whiteson, Leon Kester, and Frans CA Groen. 2010. Traffic light control by multiagent reinforcement learning systems. In Interactive Collaborative Information Systems. Springer, 475–510.

[20] Tianshu Chu, Jie Wang, Lara Codecà, and Zhaojian Li. 2019. Multi-Agent Deep Reinforcement Learning for Large-scale Traffic Signal Control.

[21] Tomoki Nishi, Keisuke Otaki, Keiichiro Hayakawa, and Takayoshi Yoshimura. 2018. Traffic Signal Control Based on Reinforcement Learning with Graph Convolutional Neural Nets. In 2018 21st International Conference on Intelligent Transportation Systems (ITSC). IEEE, 877–883.

[22] Patrick Mannion, Jim Duggan, and Enda Howley. 2016. An experimental review of reinforcement learning algorithms for adaptive traffic signal control. In Autonomic Road Transport Support Systems. Springer, 47–66.

[23] Jirı Iša, Julian Kooij, Rogier Koppejan, and Lior Kuijer. 2006. Reinforcement learning of traffic light controllers adapting to accidents. Design and Organisation of Autonomous Systems (2006), 1–14.

[24] Lun-Hui Xu, Xin-Hai Xia, and Qiang Luo. 2013. The study of reinforcement learning for traffic self-adaptive control under multiagent markov game environment. Mathematical Problems in Engineering 2013 (2013).

[25] S. Chiu and S. Chand, "Adaptive traffic signal control using fuzzy logic," in The First IEEE Regional Conference on Aerospace Control Systems, April 1993, pp. 1371–1376.

[26] B. De Schutter, "Optimal traffic light control for a single intersection," in American Control Conference, vol. 3, June 1999, pp. 2195–2199.

[27] C. J. Watkins and P. Dayan, "Q-learning," Machine learning, vol. 8, no. 3-4, pp. 279–292, 1992.

[28] Tianshu Chu, Jie Wang, Lara Codecà, and Zhaojian Li "Multi-Agent Deep Reinforcement Learning for Large-scale Traffic Signal Control" 2019.

[29] Tomoki Nishi, Keisuke Otaki, Keiichiro Hayakawa and Takayoshi Yoshimura, "Traffic Signal Control Based on Reinforcement Learning with Graph Convolutional Neural Nets". 2018 21st International Conference on Intelligent Transportation Systems (ITSC).

[30] D. Krajzewicz, M. Bonert, and P. Wagner, "The open source traffic simulation package SUMO," RoboCup 2006 Infrastructure Simulation Competition, 2006.

[31] P. B. Hunt, D. I. Robertson, R. D. Bretherton, and M. C. Royle, "The SCOOT on-line traffic signal optimization technique," Traffic Engineering & Control, vol. 23, no. 4, 1982.

[32] J. Y. K. Luk, "Two traffic-responsive area traffic control methods: SCAT and SCOOT," Traffic engineering & control, vol. 25, no. 1, pp. 14–22, 1984.

[33] N. H. Gartner, Demand-responsive Decentralized Urban Traffic Control. US Department of Transportation, Research and Special Programs Administration, 1982.

[34] J.-J. Henry, J.-L. Farges, and J. Tuffal, "The PRODYN real time traffic algorithm," in Proceedings of the IFAC/IFIPI/FORS Conference On Control, 1984.

[35] Hua Wei, Guanjie Zheng, Vikash Gayah, Zhenhui Li "A Survey on Traffic Signal Control Methods", arXiv:1904.08117.

[36] Toronto Open Data Team 2018. King St. Transit Pilot–Detailed Traffic & Pedestrian Volumes. https://open.toronto.ca/dataset/king-st-transit-pilot-detailed-traffic-pedestrian-volumes/

[37] Dunhao Zhong and Azzedine Boukerche. (2019). Traffic Signal Control Using Deep Reinforcement Learning with Multiple Resources of Rewards. 23-28. 10.1145/3345860.3361522.

[38] Potdar, Kedar & Pardawala, Taher & Pai, Chinmay. (2017). A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers. International Journal of Computer Applications. 175. 7-9. 10.5120/ijca2017915495.

[39] Richard S Sutton and Andrew G Barto. 2018. Reinforcement learning: An introduction.

[40] Yuxi Li, Deep Reinforcement Learning: An Overview, arXiv:1701.07274v6 [cs.LG] 26 Nov 2018.

[41] Arnold, Ludovic & Rebecchi, Sébastien & Chevallier, Sylvain & Paugam-Moisy, Hélène. (2011). An Introduction to Deep Learning. Proceedings of the European Symposium of Artificial Neural Network, ESANN2011. 1. 477-488.

[42] L-J LIN. "Reinforcement Learning for Robots Using Neural Networks". In: Ph.D. thesis, Carnegie Mellon University (1993).

[43] Volodymyr Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. Nature, 518 (7540):529–533, 2015.

[44] Hado van Hasselt and Arthur Guez and David Silver Google DeepMind. Deep Reinforcement Learning with Double Q-learning 2015.

[45] Diederik P. Kingma, Jimmy Ba. Adam: A Method for Stochastic Optimization.

[46] Kalyan Das1, Jiming Jiang2and J. N. K. Rao. MEAN SQUARED ERROR OF EMPIRICAL PREDICTOR.

[47] Abien Fred M. Agarap. Deep Learning using Rectified Linear Units (ReLU). arXiv:1803.08375v2.

[48] B. Zhou, J. Cao, X. Zeng and H. Wu, "Adaptive Traffic Light Control in Wireless Sensor Network-Based Intelligent Transportation System," *2010 IEEE 72nd Vehicular Technology Conference - Fall*, Ottawa, ON, 2010, pp. 1-5.

Appendix A: Figures

Figure A.1-A.4 represents the enlarged versions of Figure 5.3, Figure 5.4, Figure 5.5, Figure 5.6.

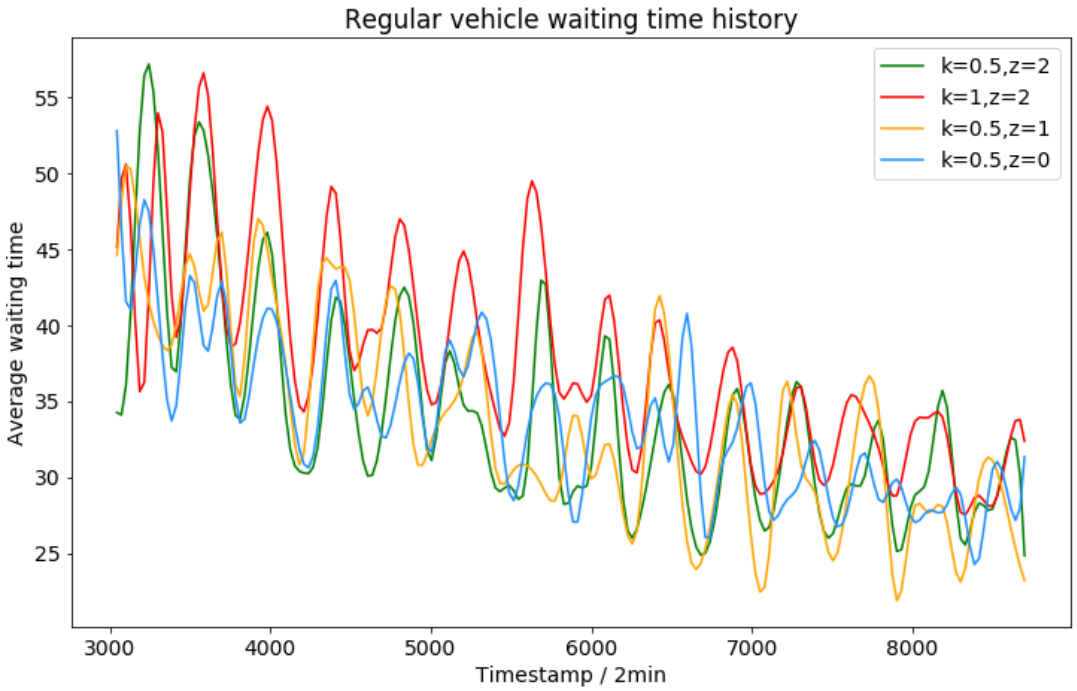


Figure A.1: Regular vehicle average waiting time for different K and Z values starting from 3000 for the x-axis

Figure A.2: Emergency vehicle average waiting time for different K and Z values starting from 3000 for the x-axis

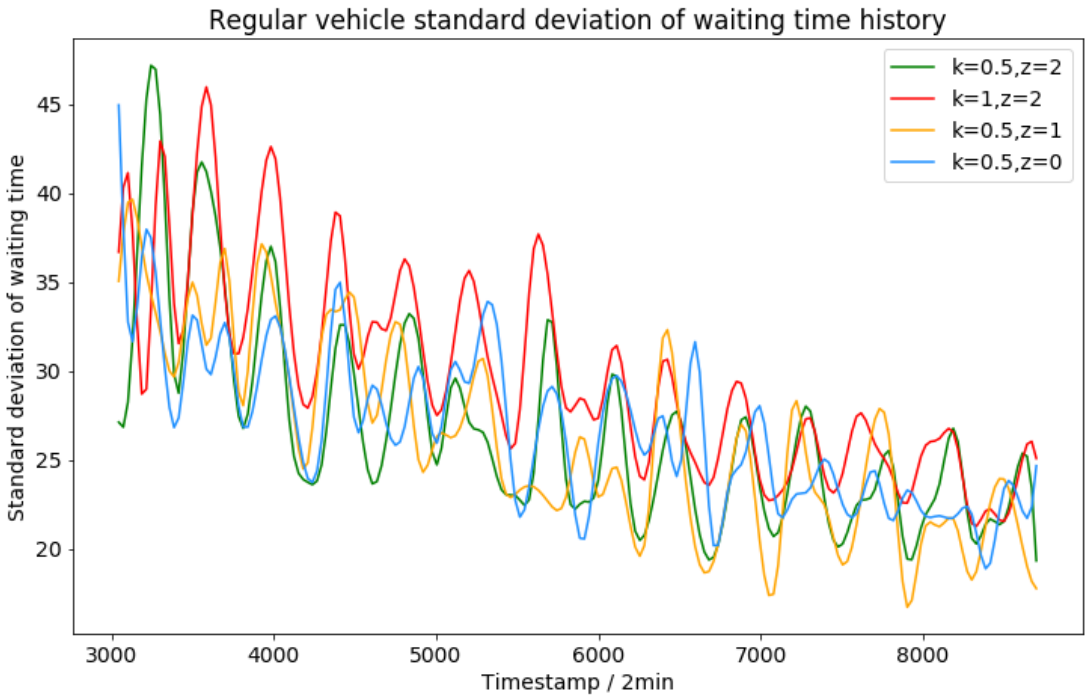


Figure A.3: Regular vehicle standard deviation of average waiting time for different K and Z values starting from 3000 for the x-axis

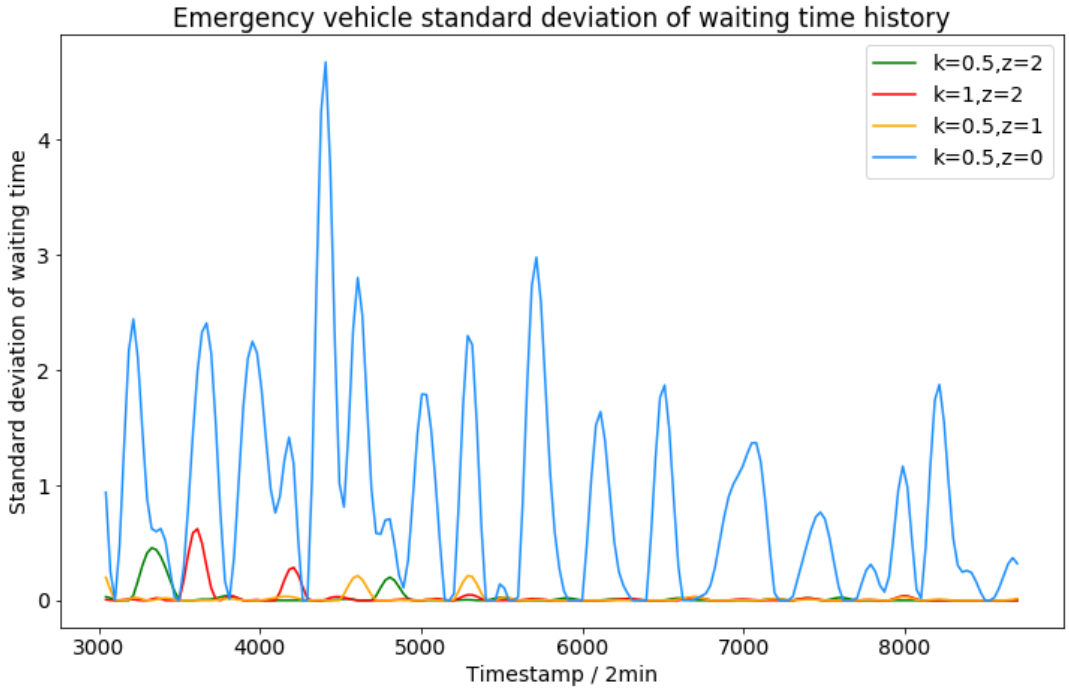


Figure A.4 Emergency vehicle standard deviation of average waiting time for different K and Z values starting from 3000 for the x-axis

Figure A.5 represents how all the four methods were able to gain the reward with the learning. Since the is no learning for statistical method and fixed time method they were not able to improve the reward while the reinforcement learning method does. The proposed method was able to get a higher reward than the average only method with time.

Figure A.5: Reward gain for each method with the training

Appendix B: RealWorld Data Plugging to SUMO

Used pandas and LXML etree libraries to build the XML rout file with the real-world data file. Since the data file contains a large amount of data, only the first 510 rows are taken to build the XML file.

```
import pandas as pd
from lxml import etree
```

```
df=pd.read_csv('RealWorld_data/traffic_volumes_2018.csv')
```

```
df=df.loc[df['classification'] == 'Lights']
```

```
traffic_data = df[:510]
traffic_data.head(25)
```

intersection_name	px	datetime_bin	classification	leg	dir	volume
Queen St E / Jarvis St	6	2018-01-15 06:00:00	Lights	E	EB	12
Queen St E / Jarvis St	6	2018-01-15 06:00:00	Lights	E	WB	30
Queen St E / Jarvis St	6	2018-01-15 06:00:00	Lights	N	NB	105
Queen St E / Jarvis St	6	2018-01-15 06:00:00	Lights	N	SB	90
Queen St E / Jarvis St	6	2018-01-15 06:00:00	Lights	S	NB	108
Queen St E / Jarvis St	6	2018-01-15 06:00:00	Lights	S	SB	96
Queen St E / Jarvis St	6	2018-01-15 06:00:00	Lights	W	EB	19
Queen St E / Jarvis St	6	2018-01-15 06:00:00	Lights	W	WB	34
Queen St E / Jarvis St	6	2018-01-15 06:15:00	Lights	E	EB	26
Queen St E / Jarvis St	6	2018-01-15 06:15:00	Lights	E	WB	38
Queen St E / Jarvis St	6	2018-01-15 06:15:00	Lights	N	NB	83
Queen St E / Jarvis St	6	2018-01-15 06:15:00	Lights	N	SB	104
Queen St E / Jarvis St	6	2018-01-15 06:15:00	Lights	S	NB	94
Queen St E / Jarvis St	6	2018-01-15 06:15:00	Lights	S	SB	102
Queen St E / Jarvis St	6	2018-01-15 06:15:00	Lights	W	EB	22
Queen St E / Jarvis St	6	2018-01-15 06:15:00	Lights	W	WB	47
Queen St E / Jarvis St	6	2018-01-15 06:30:00	Lights	E	EB	28
Queen St E / Jarvis St	6	2018-01-15 06:30:00	Lights	E	WB	34
Queen St E / Jarvis St	6	2018-01-15 06:30:00	Lights	N	NB	122
Queen St E / Jarvis St	6	2018-01-15 06:30:00	Lights	N	SB	116
Queen St E / Jarvis St	6	2018-01-15 06:30:00	Lights	S	NB	125
Queen St E / Jarvis St	6	2018-01-15 06:30:00	Lights	S	SB	114
Queen St E / Jarvis St	6	2018-01-15 06:30:00	Lights	W	EB	33

```
begin = 0
root = etree.Element("root")
routes={"eeb":"2si_to_4o","ewb":"2si_to_1o","nnb":"4si_to_1o",\
        "nsb":"4si_to_3o","snb":"3si_to_4o","ssb":"3si_to_2o",\
        "web":"1si_to_2o","wwb":"1si_to_3o","eeb":"2si_to_3o",\
        "ewb":"2si_to_1o"}
for index, row in traffic_data.iterrows():
    direction = row['leg'].lower()+row['dir'].lower()

    try:
        #check whether previous_datetime_bin defined
        previous_datetime_bin
    except:
        previous_datetime_bin = row['datetime_bin']

    if(row['datetime_bin'] != previous_datetime_bin):
        begin += 900#incease the new flow start time by 15 min
        previous_datetime_bin = row['datetime_bin']

    flow = etree.Element\
    ("flow", id=str(index),type="SUMO_DEFAULT_TYPE",\
    route=str(routes[direction]), begin=str(begin),\
    end=str(begin+900), probability=str(row['volume']/900))
    root.append(flow)
```

```
print(etree.tostring(root, pretty_print=True))
tree = etree.ElementTree(root)
tree.write("realworld.rou.xml")
```

Appendix C: Implementation of Vehicle Detection API

```
def process_img(original_image):# grayscalling and edg detection
    scale = 1
    delta = 0
    ddepth = cv2.CV_16S
    original_image=np.array(original_image)
    gray_image=cv2.cvtColor(original_image,cv2.COLOR_RGB2GRAY)
    processed_image=cv2.Canny(gray_image,threshold1=200,threshold2=300)
    return processed_image
```

```
def roi(img,vertices):# taking the region of interest
    mask=np.zeros_like(img)
    cv2.fillPoly(mask,vertices,255)
    masked = cv2.bitwise_and(img,mask)
    return masked
```

Load vehicle detection model

```
# name of the vehicle detection model.
MODEL_NAME = 'vehicle_detection_model'

# Path to frozen detection graph. This is the actual model that is use
PATH_TO_FROZEN_GRAPH = MODEL_NAME + '/frozen_inference_graph.pb'

# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = os.path.join('assets', 'label_map.pbtxt')

detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_FROZEN_GRAPH, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name='')
```

```
def detect_roads():

    global left,right,top,bottom,left_edge1,left_edge2,right_edge1,\
    right_edge2,top_edge1,top_edge2,bottom_edge1,bottom_edge2
    original_image=ImageGrab.grab(bbox=(10,155,670,740))
    (im_width, im_height) =original_image.size
    processed_image=process_img(original_image)
    verticies=np.array([[0,230],[230,230],[230,0],[370,0],[370,230],\
                        [670,230],[670,350],[370,350],[370,570],\
                        [230,570],[230,350],[0,350]])
    processed_image=roi(processed_image,[verticies])
    #make this call only onece
    left,right,top,bottom=\
    road_detection.detect_roads(processed_image,im_width,im_height)

    #detecting the boundry line of the road
    left_edge1=max(x[0] for x in left)
    left_edge2=min(x[0] for x in left)

    right_edge1=max(x[0] for x in right)
    right_edge2=min(x[0] for x in right)

    top_edge1=max(x[0] for x in top)
    top_edge2=min(x[0] for x in top)

    bottom_edge1=max(x[0] for x in bottom)
    bottom_edge2=min(x[0] for x in bottom)
```

```
def getState(sess):

    global num_vehicles_detected,current_vehicles,left,right,top,\
    bottom,left_edge1,left_edge2,right_edge1,right_edge2,top_edge1,\
    top_edge2,bottom_edge1,bottom_edge2

    original_image=ImageGrab.grab(bbox=(10,155,670,740))
    (im_width, im_height) =original_image.size
    # this will return x,y,vehicle_class
    vehicle_centroids=vehicle_detection.get_objects(original_image,sess)

    #assigning vehicles to tracks

    for vehicle_centroid in vehicle_centroids:
        section=0 #this is used to fill the coresponding element of
        #lane_matrix depending on vehicle road and track
        #detecting the current road of a vehicle
        if vehicle_centroid[0]<=left[0][1]:
            road="left"
        elif vehicle_centroid[0]>=right[0][1]:
            road="right"
        elif vehicle_centroid[1]>=top[0][1]:
            road="top"
        elif vehicle_centroid[1]<=bottom[0][1]:
            road="bottom"
        else: #ignoreing the vehicles passing through the intersection
            continue
```

```
#detecting the current track of a vehicle
if road=="left":

    if (1/4)*abs(left_edge1-left_edge2)>=\
    abs(left_edge1-vehicle_centroid[1]):
        track=0
    elif (2/4)*abs(left_edge1-left_edge2)>=\
    abs(left_edge1-vehicle_centroid[1]):
        track=1
    elif (3/4)*abs(left_edge1-left_edge2)>=\
    abs(left_edge1-vehicle_centroid[1]):
        track=2
    else:
        track=3
elif road=="right":
    section=4
    if (1/4)*abs(right_edge1-right_edge2)>=\
    abs(right_edge2-vehicle_centroid[1]):
        track=0
    elif (2/4)*abs(right_edge1-right_edge2)>=\
    abs(right_edge2-vehicle_centroid[1]):
        track=1
    elif (3/4)*abs(right_edge1-right_edge2)>=\
    abs(right_edge2-vehicle_centroid[1]):
        track=2
    else:
        track=3
elif road=="top":
    section=8
    if (1/4)*abs(top_edge1-top_edge2)>=\
    abs(top_edge1-vehicle_centroid[0]):
        track=0
    elif (2/4)*abs(top_edge1-top_edge2)>=\
    abs(top_edge1-vehicle_centroid[0]):
        track=1
    elif (3/4)*abs(top_edge1-top_edge2)>=\
    abs(top_edge1-vehicle_centroid[0]):
        track=2
    else:
        track=3
else:
    section=12
    if (1/4)*abs(bottom_edge1-bottom_edge2)>=\
    abs(bottom_edge2-vehicle_centroid[0]):
        track=0
    elif (2/4)*abs(bottom_edge1-bottom_edge2)>=\
    abs(bottom_edge2-vehicle_centroid[0]):
        track=1
    elif (3/4)*abs(bottom_edge1-bottom_edge2)>=\
    abs(bottom_edge2-vehicle_centroid[0]):
        track=2
    else:
        track=3
```

Figure C.1 shows how the Canny edge detector was able to detect roads.

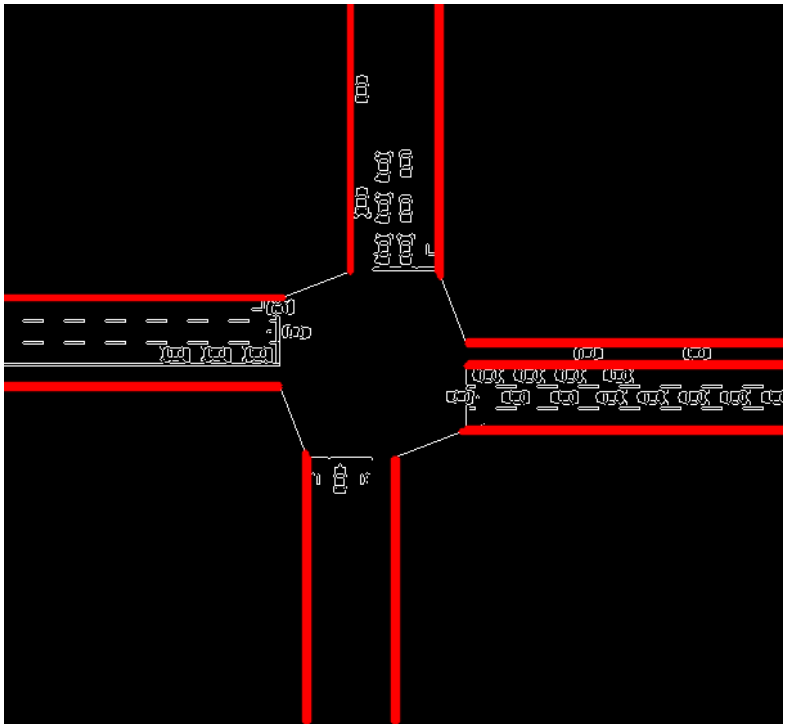


Figure C.1: Road edge detection using canny edge detector

Figure C.2 shows that the identified vehicles are marked using a boundary box. It also contains the type of the vehicle and the confidence. Since this is the first attempt algorithm, some vehicles are not detected.

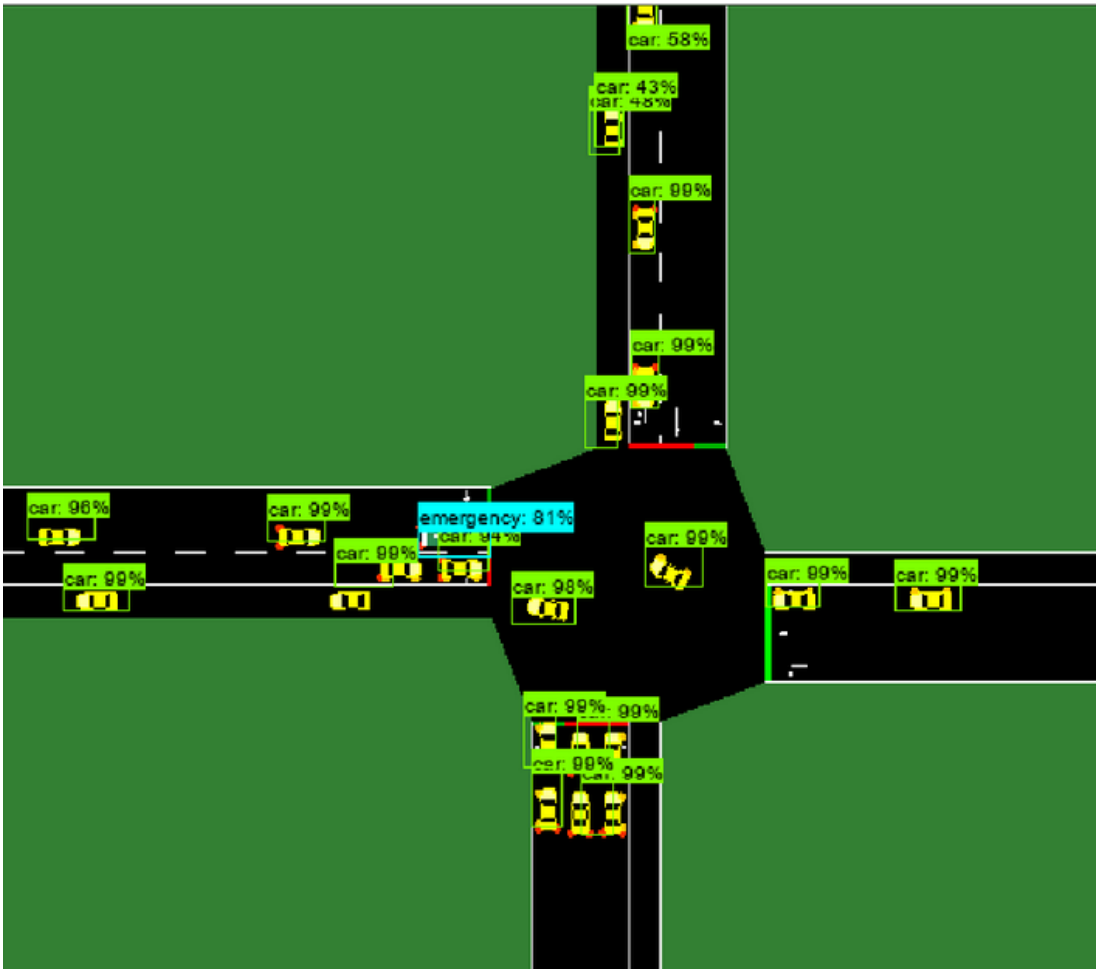


Figure C.2: Identified vehicles are marked using boundary box