

Traffic Signal Control using Reinforcement Learning



By

Qazi Umer Jamil

NUST CMS ID 317920

Supervisor

Dr. Karam Dad Kallu

Department of Robotics & Artificial Intelligence
School of Mechanical & Manufacturing Engineering (SMME) ,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(July 2023)

Traffic Signal Control using Reinforcement Learning

By

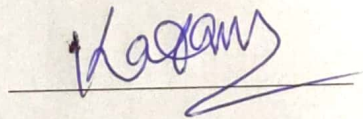
Qazi Umer Jamil

NUST CMS ID 317920

A thesis submitted in conformity with the requirements for
the degree of *Master of Science* in
Robotics and Intelligent Machines Engineering

Supervisor

Dr. Karam Dad Kallu




Department of Robotics and Artificial Intelligence
School of Mechanical and Manufacturing Engineering (SMME)
National University of Sciences and Technology (NUST)
Islamabad, Pakistan

National University of Sciences & Technology
MASTER THESIS WORK


We hereby recommend that the dissertation prepared under our supervision by: (Student Name & Regn No.) Qazi Umer Jamil (Regn No: 317920)
 Titled: Traffic Signal Control using Reinforcement Learning be accepted in partial fulfillment of the requirements for the award of MS RIME degree.

Examination Committee Members

1. Name: Dr. Kashif Javed

Signature: 

2. Name: Dr. Muhammad Jawad Khan

Signature: 

Supervisor's name: Dr. Karam Dad Kallu

Signature: 

Date: 20-7-2023



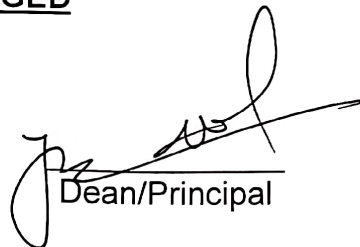
Head of Department

24-7-2023

Date

COUNTERSIGNED

Date: 24-7-23


 Dean/Principal

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS thesis written by Mr. Qazi Umer Jamil (Registration No. 317920), of SMME (School of Mechanical and Manufacturing Engineering) has been vetted by undersigned, found complete in all respects as per NUST Statutes / Regulations, is within the similarity indices limit and is accepted as partial fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature: Karam

Name of Supervisor: Dr. Karam Dad Kallu

Date: 20-7-2023

Signature (HOD): Kallu

Date: 24-7-2023

Signature (Dean/Principal): [Signature]

Date: 24-7-23

Proposed Certificate for Plagiarism

It is certified that PhD/M.Phil/MS Thesis Titled Traffic Signal Control using Reinforcement Learning by Qazi Umer Jamil has been examined by us. We undertake the follows:

- a. Thesis has significant new work/knowledge as compared already published or are under consideration to be published elsewhere. No sentence, equation, diagram, table, paragraph or section has been copied verbatim from previous work unless it is placed under quotation marks and duly referenced.
- b. The work presented is original and own work of the author (i.e. there is no plagiarism). No ideas, processes, results or words of others have been presented as Author own work.
- c. There is no fabrication of data or results which have been compiled/analyzed.
- d. There is no falsification by manipulating research materials, equipment or processes, or changing or omitting data or results such that the research is not accurately represented in the research record.
- e. The thesis has been checked using TURNITIN (copy of originality report attached) and found within limits as per HEC plagiarism Policy and instructions issued from time to time.

Name & Signature of Supervisor

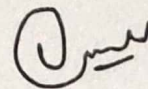
Dr. Karam Dad Kallu

Signature :

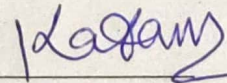


Plagiarism Certificate (Turnitin Report)

This thesis has been checked for Plagiarism. Turnitin report endorsed by Supervisor is attached.



Qazi Umer Jamil,
NUST CMS ID: 317920



Dr. Karam Dad Kallu,
(Supervisor)

Declaration

I, *Qazi Umer Jamil*, declare that this thesis titled “Traffic Signal Control using Reinforcement Learning” and the work presented in it are my own and has been generated by me as a result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a Master of Science degree at NUST
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at NUST or any other institution, this has been clearly stated
3. Where I have consulted the published work of others, this is always clearly attributed
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work
5. I have acknowledged all main sources of help
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself



Qazi Umer Jamil,
NUST CMS ID: 317920

Copyright Statement

- Copyright in text of this thesis rests with the student author. Copies (by any process) either in full, or of extracts, may be made only in accordance with instructions given by the author and lodged in the Library of SMME, NUST. Details may be obtained by the Librarian. This page must form part of any such copies made. Further copies (by any process) may not be made without the permission (in writing) of the author.
- The ownership of any intellectual property rights which may be described in this thesis is vested in SMME, NUST, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of SMME, which will prescribe the terms and conditions of any such agreement.
- Further information on the conditions under which disclosures and exploitation may take place is available from the Library of SMME, NUST, Islamabad.

Dedication

This work is dedicated to my university teachers, the torchbearers of knowledge and wisdom. Their invaluable lessons, unwavering support, and ceaseless encouragement have been instrumental in my academic journey. Their commitment to imparting knowledge and fostering intellectual curiosity has profoundly influenced my educational path. To all my professors who dedicated their time, shared their expertise, and instilled in me the passion for learning, this thesis is a testament to your invaluable contribution to my academic growth. Your teachings extend far beyond the classroom, and for this, I am forever grateful. Thank you for inspiring me and for pushing me to reach new heights.

Acknowledgments

Glory be to Allah (S.W.A), the Creator, the Sustainer of the Universe. Who only has the power to honour whom He please, and to abase whom He please. Verily no one can do anything without His will. From the day, I came to NUST till the day of my departure, He was the only one Who blessed me and opened ways for me, and showed me the path of success. There is nothing which can payback for His bounties throughout my research period to complete it successfully.

Qazi Umer Jamil

Contents

1	Introduction	1
1.1	Background	1
1.2	Problem Statement	2
1.3	Research direction	3
2	Literature review	5
2.1	Agent	5
2.1.1	Simple reflex agent	6
2.1.2	Model-based Reflex Agent	6
2.1.3	Goal-based Agent	7
2.1.4	Utility-based Agent	8
2.1.5	Learning Agent	8
2.1.6	Reinforcement Learning Approach	9
2.1.7	Markov Decision Process (MDP)	11
2.1.8	RL Algorithms	12
2.1.9	RL Algorithm Considered In This Work	14
2.2	Survey of related work	17
2.2.1	State Space Representation	19
2.2.2	Defining Action Spaces	20
2.2.3	Reward Function Formulation	21
2.2.4	Reinforcement Learning Approaches	21

CONTENTS

2.2.5	Convergence in Q-Learning	22
2.2.6	Simulation software	23
3	Design of the Reinforcement Learning Agent	25
3.1	Problem Definition	26
3.2	Environment	27
3.2.1	Single Agent Environment	27
3.3	State Representation	30
3.3.1	Action Space	33
3.3.2	Reward Function	36
3.4	Learning Mechanism	37
3.4.1	Deep neural network architecture	38
3.4.2	Training deep neural network	38
4	Experimental Setup	40
4.1	Urban Mobility Simulation Tools and Techniques	41
4.1.1	Netconvert	42
4.1.2	Polyconvert	42
4.1.3	NetEdit	42
4.1.4	TraCI (Traffic Control Interface)	43
4.1.5	Simulation Options	43
4.2	Experience Replay: A Key Mechanism in Reinforcement Learning	43
4.3	Training Methodology of Reinforcement Learning Agent	45
4.4	Balancing Exploration and Exploitation in Training	46
4.5	Traffic Generation in Simulations	47
5	Results and Discussion	51
5.1	Evaluating Performance	51
5.2	Evaluation Metrics	52

CONTENTS

5.2.1	Total Negative Reward	52
5.2.2	Total Accumulated Wait Time	53
5.2.3	Expected Wait Time per Car	53
5.2.4	Average Queue Length	54
5.3	Results	54
5.3.1	Agent 1: Deep Reinforcement Learning with Q Learning (DQN)	54
5.3.2	Agent 2: Deep Reinforcement Learning with Double Q Learning (DDQN)	59
5.4	Conclusion	64
5.5	Future Work	65

List of Figures

2.1	Type of agents	6
2.2	Simple reflex agent	6
2.3	Model-Based Reflex Agent	7
2.4	Goal-Based Agent	7
2.5	Utility-Based Agent	8
2.6	Learning Agent	9
2.7	Reinforcement Learning Agent	10
3.1	Single Agent Environment	28
3.2	Potential actions for a turn-based agent at a four-way intersection	35
4.1	Operational Cycle of the Reinforcement Learning Agent	40
4.2	Process of Experience Replay	44
4.3	Epsilon Decay in Linear Model	47
4.4	Three Different Traffic Generation Distributions	48
5.1	Total Negative Reward	55
5.2	Total Average Reward	56
5.3	Total Reward	56
5.4	Wait time in Cardinal Direction of North	57
5.5	Wait time in Cardinal Direction of East	58
5.6	Wait time in Cardinal Direction of West	58

LIST OF FIGURES

5.7	Wait time in Cardinal Direction of South	59
5.8	Total Negative Reward	60
5.9	Total Average Reward	60
5.10	Total Reward	61
5.11	Wait time in Cardinal Direction of North	62
5.12	Wait time in Cardinal Direction of East	62
5.13	Wait time in Cardinal Direction of West	63
5.14	Wait time in Cardinal Direction of South	63

List of Tables

3.1	Notations used in Chapter 3	26
3.2	Traffic Light States	29
3.3	Vehicle Attributes	30
3.4	Figure 3-9: An encoding matrix with a 32-bit capacity, capable of managing up to 492 vehicles.	32
3.5	Figure 3-10: An encoding matrix with a 32-bit capacity, capable of managing up to 200 vehicles.	33
3.6	Figure 3-11: An encoding matrix with a 32-bit capacity, capable of managing up to 300 vehicles.	33
3.7	Figure 3-12: An encoding matrix with a 48-bit capacity, capable of managing up to 200 vehicles.	33
3.8	Figure 3-13: An encoding matrix with a 48-bit capacity, capable of managing up to 300 vehicles.	33
4.1	Summary of Simulation Files	42
4.2	Traffic Generation in Different Traffic Scenarios	49
4.3	Traffic Generation Probability for Four-Way Intersection	49
5.1	Chapter 5 Terminology	52

Abstract

This thesis presents an in-depth examination of traffic light control optimization using Reinforcement Learning (RL) techniques. The research focuses on two specific RL algorithms: Deep Q-Learning (DQN) and Double Deep Q-Learning (DDQN), investigating their ability to reduce wait times at a four-way traffic intersection. The RL agents' learning process is driven by a reward function based on waiting times, designed to guide the agents towards minimizing these times. A transition phase is implemented in the model, allowing for flexibility and responsiveness to changing traffic conditions. Deep Neural Networks (DNNs) are used as function approximators, facilitating the understanding of the association between state-action pairs. The architecture comprises five fully connected hidden layers, providing an effective means of approximating the Q-values for the state-action pairs. Training data for the DNN is stored in an Experience Replay Memory, which is effectively a history of state, action, reward, and subsequent state. The study concludes that both DQN and DDQN agents demonstrated an increasing proficiency over time, indicating the successful application of RL techniques in traffic light control systems. This research contributes to the ongoing efforts to employ advanced RL techniques in optimizing traffic flow, with potential applications in intelligent transportation systems, smart cities, and autonomous vehicle navigation.

Introduction

1.1 Background

Increasing world population, particularly within urban areas [8], has increased the demand for road transportation, necessitating the efficient movement of people, goods, and services. The increased numbers of vehicles, however, are contributing to increasingly serious traffic congestion issues, especially in densely populated urban areas and cities. This congestion is not only problematic for individual commuters and the transport industry, but it also leads to increased fuel consumption, elevated levels of CO₂ emissions, and wasted time. This not only affects the environment, but makes travel costly in terms of money and time.

Several factors contribute to these traffic challenges, including the large number of vehicles on the road, outdated or poorly designed traffic infrastructure, and inefficient traffic light control systems. While it's unrealistic to curb the number of vehicles people purchase or embark on large-scale infrastructure overhauls without burdening the economy, it's crucial to explore feasible solutions to these pressing issues.

One promising strategy is focused on optimizing green signal timing at traffic signal at intersections. Traditional traffic light systems often run on a predetermined traffic light sequence program that allocates a fixed duration of green time for each direction at an intersection, regardless of real-time number of vehicles at the intersection. While this static approach works relatively well under low traffic conditions, it tends to break down during periods of high traffic flow, such as during popular events, and peak school or working hours.

An interesting observation has been made when these automated systems fail due to technical or other issues: a human traffic officer often takes over control at the intersection. These officers, based on their experience and real-time assessment of the traffic situation, dynamically adjust the signal timing for each direction of the intersection. This ability to adapt to the situation on the ground makes human traffic controllers remarkably effective. However this approach is not scalable and sustainable in the long run.

A feasible solution, therefore, is to develop and deploy advanced traffic control systems that can mimic this human ability to adapt to real-time traffic conditions. Using machine learning and artificial intelligence algorithms, these systems could analyze traffic flows, anticipate traffic patterns, and dynamically adjust signal timings to optimize traffic flow at intersections. This approach could significantly enhance the efficiency of our road transport systems, mitigating traffic congestion and its associated problems.

1.2 Problem Statement

As emerging economies continue to grow and urbanize, the infrastructure, specifically traffic and road systems, are undergoing significant transformations. The continuous reshaping and upgrading of road networks often result in the alteration of previously established traffic patterns. Such drastic changes can inadvertently increase traffic congestion issues, impacting mobility and efficiency.

Traditionally, efforts to alleviate these congestion issues have relied heavily on the pre-setting of green light durations, established according to historical traffic data. However, this approach often fails to yield the desired results.^[69] The primary reason is its inability to account for fluctuations in traffic demand, which are frequent but unpredictable, given the changing nature of road networks and urban population growth.

An alternative approach has been to employ inductive loop detectors, which are sensors designed to control traffic signals. Despite their usefulness, these sensors come with notable drawbacks. They often lack precision in detecting smaller vehicles, and are prone to damage from regular wear and tear of the roads, leading to reliability and maintenance issues. They also require significant changes to the road's physical infrastructure.

What's clearly needed is a more dynamic, intelligent, and adaptable system that goes

beyond merely managing traffic flow at intersections. Such a system should be able to learn and adjust to new control policies based on evolving traffic behaviors and changing road conditions.

In the last few years, machine learning has emerged as a domain that can address complex tasks such as object detection, recognition and classification. These complex tasks, however, rely heavily on the availability of extensive training examples. Unfortunately, in traffic light control scenario, this kind of training examples is seldom available. Undoubtedly, even though data may be accessible, it's important to recognize that the characteristics of this data will be distinctive for each traffic light controller throughout the city.

One potential solution to this challenge lies in a specific machine learning domain known as Reinforcement Learning (RL).[4] RL operates on the principle of learning through interactions with agent's environment and receiving feedback on its actions in terms of rewards or punishments. Unlike other learning models, RL does not necessitate a comprehensive understanding of physics of the environment beforehand.

Initially, the Reinforcement Learning (RL), the agent navigates traffic environment by executing actions in a seemingly random manner. It then observes the impact of these actions on traffic flow, learning from the improvements or regressions that ensue. This iterative trial and error process gradually enables the RL agent to comprehend and enhance the efficiency of traffic control at intersections.

Given the dynamics traffic flows and the constantly evolving road networks in developing nations, employing reinforcement learning could be a promising way forward. Through continued learning and adaptation, such a system could help manage the complex task of traffic control effectively, making our urban road networks more efficient and less congested.

1.3 Research direction

The fundamental objective of our research dissertation is focused on the applicability and effectiveness of deep reinforcement learning (DRL) algorithms within the field of traffic flow control. It aims to identify and compare various reinforcement learning methodologies, with the goal of enhancing the flow of traffic movement at intersections.

The outcome will involve selecting the most efficient algorithm for potential practical deployment. Additionally, the research will explore various sensor technologies and methods to implement these reinforcement learning algorithms practically.

This work will focus on following Deep Reinforcement Learning algorithms:

- Deep Reinforcement Learning with Q Learning (DQN) [25]
- Deep Reinforcement Learning with Double Q Learning (DDQN) [33]

These particular algorithms have been chosen due to their significant contributions to the field and their potential applicability to the traffic control problem.

Main objectives of this research are outlined below:

- Implement a traffic light control system for a single intersection using above listed DRL algorithms. This objective aims to explore the practical application of these algorithms in a controlled setting, understanding their operations and impacts on traffic control.
- Conduct a comparative analysis of the reinforcement learning algorithms' performance. By doing this, we can evaluate their effectiveness and suitability for the specific demands of traffic control.
- Performance evaluation of reinforcement learning agents in varying traffic flow settings. This objective will test the versatility and adaptability of the algorithms under different conditions, highlighting their robustness or lack thereof.
- Identify the hyperparameters that significantly influence the performance of the reinforcement learning agent. This analysis will help optimize the learning process, allowing the system to operate at its highest potential efficiency.
- Propose a viable method to implement a reinforcement learning agent in a practical setting using existing sensor technologies. This objective will bridge the gap between theoretical and practical application, offering a tangible solution for real-world traffic control challenges.

By fulfilling these objectives, this research aims to contribute significantly to the field of traffic control, potentially offering a viable, intelligent solution to the persistent problem of traffic congestion.

CHAPTER 2

Literature review

This chapter will provide a fundamental understanding of learning agents and reinforcement learning, alongside a review of past research efforts aimed at resolving traffic control problems.

2.1 Agent

Drawing on work of Russell and Norvig [30], a learning agent can be described as any entity capable of perceiving its environment through sensors and interacting with it using actuators. The agent's actions, driven by these actuations, result in changes to current state of environment. An agent's decisions to actions to take can be based either on the current environmental state or a series of past environmental states. Agent's behavior is essentially determined by state action pairs, which correlates a sequence of past states with a particular action.

A rational agent is one that makes the optimal decision at every environmental state. However, the action function does not provide feedback on the correctness of actions. To enable agent to distinguish between beneficial and detrimental actions, a performance measure is necessary. This measure provides feedback to agent by analyzing resultant state of the environment after an action has been taken. In the field of reinforcement leaning, this performance measure is known as "reward".

Based on their complexity, agents can be categorized as follows:

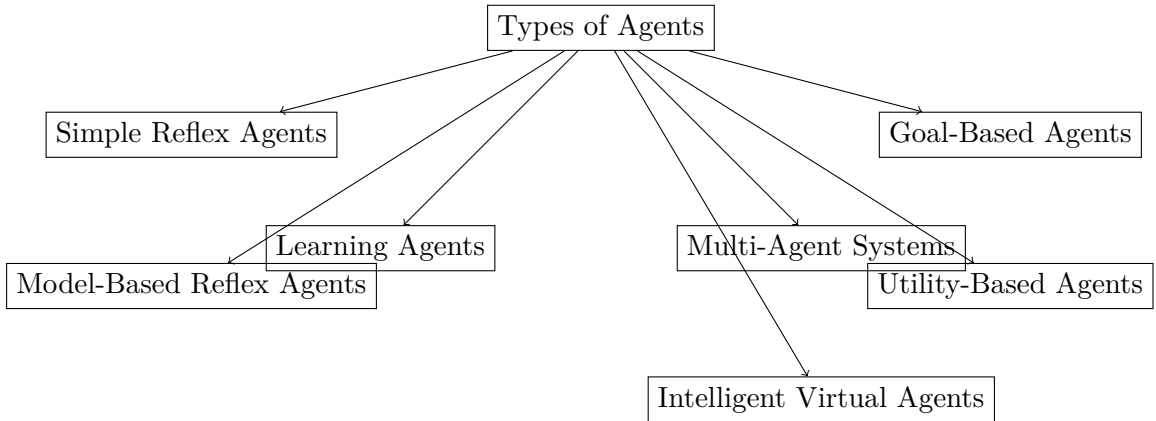


Figure 2.1: Type of agents

2.1.1 Simple reflex agent

A simple reflex agent operates only on given current state of environment, disregarding any historical sequence of states. This agent adheres to a condition-action rule – "if condition, then action." However, its effectiveness is reduced as it operates in a fully observable environment. In other words, this agent may make optimal decisions if the entire state of given environment is partially or fully observable. Moreover, the agent has no ability to adapt to the environment. [66]

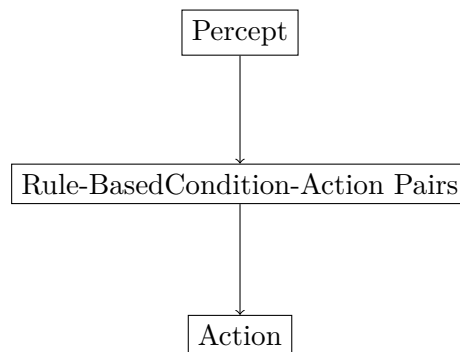


Figure 2.2: Simple reflex agent

2.1.2 Model-based Reflex Agent

A model-based reflex agent increase the capabilities of the simple reflex agent. It possesses an internal structure, referred to as the environment model, which describes physics of the environment which is not fully observable. This model enables the agent

to handle environments that are only partially observable, thus improving its decision-making abilities. [18]

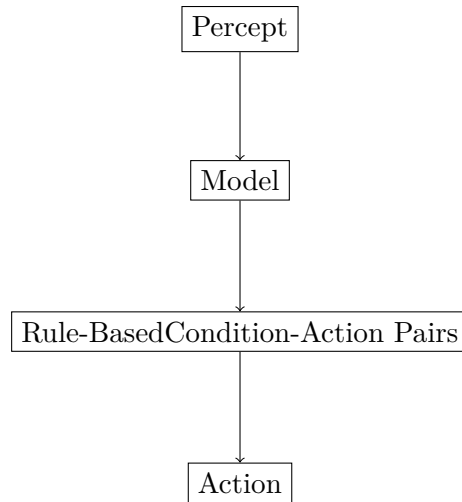


Figure 2.3: Model-Based Reflex Agent

2.1.3 Goal-based Agent

Goal-based agent broadens the limits of a model-based agent by incorporating information about the goals it seeks to achieve. This goal-oriented information allows the agent to make strategic decisions that bring it closer to achieving its set goals. As such, the agent's decisions are not only driven by current conditions but also guided by the desired outcomes.[11]

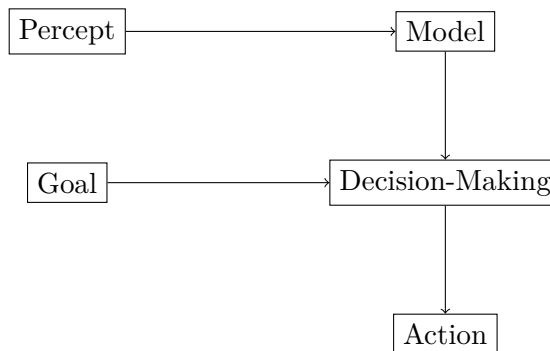


Figure 2.4: Goal-Based Agent

2.1.4 Utility-based Agent

While goal information may enable an agent to accomplish a task successfully, it does not necessarily guide it towards the most efficient way of doing so. For instance, consider a scenario where an agent is presented with two routes to a goal, as shown in Figure 2-1. The blue path represents the optimal route, yet the agent chooses the red path. Despite both paths leading to the same goal, they entail different costs. A utility-based agent, thus, incorporates a performance measure that give preference the selection of the optimal path, balancing the goal with the cost of actions taken to achieve them.[12]

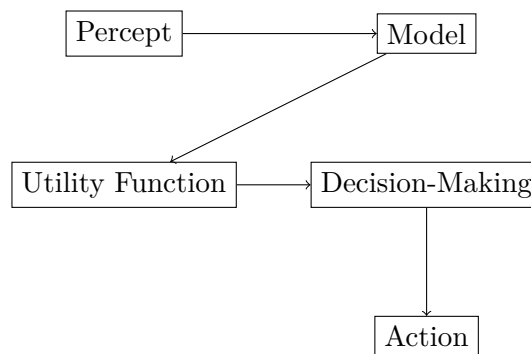


Figure 2.5: Utility-Based Agent

2.1.5 Learning Agent

A learning agent consists of four components, as depicted in Figure 2.6. These include:

- The learning element, responsible for improving the selection of actions.
- The performance element, which determines the desired actions. The performance element is the whole agent discussed in previous sections since it receives the state and performs the action.
- The critic component receives the state, compares the agent's performance against a predefined standard, and provides feedback to the learning element. This feedback aids in enhancing future actions.
- The problem generator suggests the performance element to carry out some random actions, aiming to generate new and informative experiences.

The learning agent is the most complex yet useful type of rational agent. Various

techniques and algorithms have been developed to implement learning agents in reinforcement learning, which will be discussed in upcoming later sections.[7]

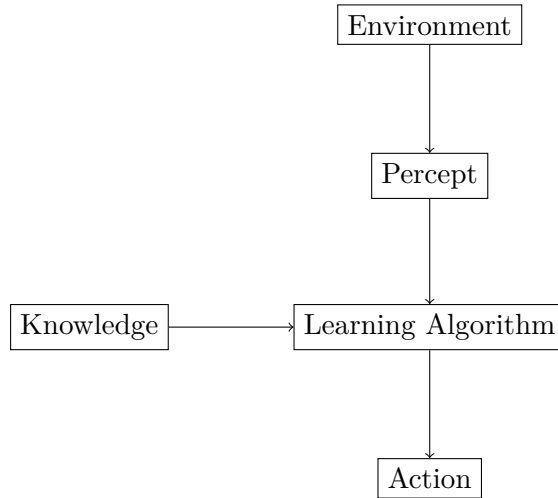


Figure 2.6: Learning Agent

2.1.6 Reinforcement Learning Approach

Reinforcement learning (RL) is a subfield of machine learning, primarily focused on determining the optimal sequence of actions for a software agent in a given environment, all with the aim to maximize the cumulative reward over time. This learning paradigm symbolizes a dynamic and interactive methodology. In this scenario, an agent evolves within its environment, aiming to establish the most effective behavior through active engagement and reciprocated feedback, typically framed as rewards or punishments. Sutton2018.

In the standard RL paradigm, an agent performs actions within an environment, transitioning between various states. For each action taken, the environment provides the agent with a corresponding reward (or penalty). The agent’s objective is to learn an optimal policy, which is essentially a strategy or set of rules, that will maximize the cumulative reward over time.[5]

The RL approach can be broken down into the following components:

- Agent: The learner and decision maker.
- Environment: The physics with which agent interacts with.
- State: A configuration of the agent and environment at a given time.

- Action: Choices that the agent makes.
- Reward: Feedback from the environment following an action.
- Policy: The agent’s strategy or the method of selecting actions.

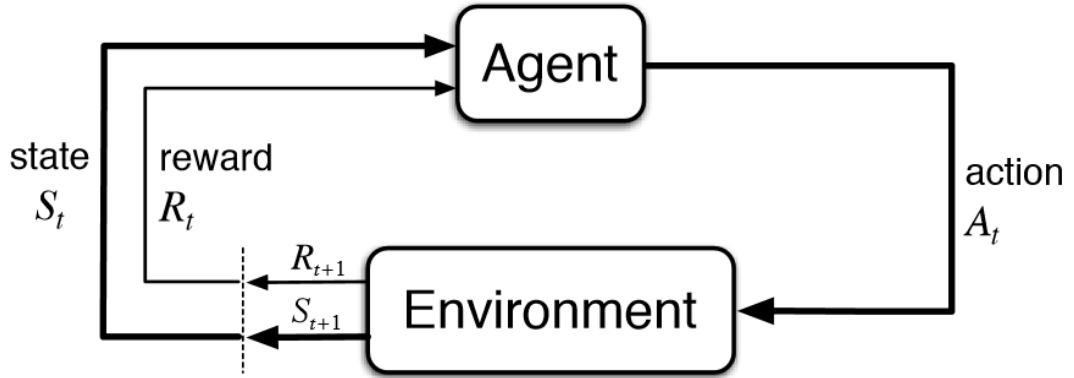


Figure 2.7: Reinforcement Learning Agent

A typical reinforcement learning cycle starts with agent observing current state (s) of environment. Based on this observation, the agent then selects an action (a) to perform. The environment transitions to next state (s') following the action, and the agent gets reward (r) that reflects feedback of action taken. This reward might be immediate or delayed. The agent uses this reward as a signal to update its policy, improving its decision-making process over time.

RL has been effective in addressing various complex problems across numerous domains. DeepMind’s AlphaGo, for instance, used RL and deep learning to defeat the world champion in Go, a complex board game [35] [31]. In robotics, RL algorithms have been used to teach robots to perform intricate tasks that would be challenging to hand-engineer, like manipulating objects or locomotion [40]. In traffic light signal control, RL algorithms are employed to optimize signal timings, leading to improved traffic flow and reduced congestion. [21]

Despite its remarkable achievements, reinforcement learning remains an engaging area for ongoing research. Numerous challenges continue to demand attention, such as managing vast state and action spaces, navigating multi-agent environments, mastering transfer learning, and addressing safety and ethical concerns associated with RL systems.

2.1.7 Markov Decision Process (MDP)

Reinforcement learning employs the methodology of a Markov Decision Process (MDP), an analytic structure used to represent decision-making contexts in which results are partially stochastic and partially dictated by the decision-maker. The defining feature of an MDP is that the likelihood of moving to any specific state depends only on the current state and action, rather than the sequence of preceding events. This characteristic is known as the Markov property.

Mathematical Framework of MDP

In the MDP model, the following elements are typically present: [19]

- States (S): A set of states S , where each state is denoted by s , and s belongs to S . A state represents a condition of the environment at a given time.
- Actions (A): A set of actions A , where each action is denoted by a , and a belongs to A . An action represents the decision made by an agent in a given state.
- Transition Probability ($P(s_{t+1}|s, a)$): This illustrates the likelihood of transitioning from the existing state s to a succeeding state s_{t+1} when an action a is undertaken. Frequently, these transition probabilities are delineated via a transition model.
- Reward Function ($R(s, a, s_{t+1})$): This function gives an immediate reward after transitioning from state s to state s_{t+1} through action a . The reward indicates the goodness of an action taken in a state.
- Discount Factor (γ): The discount factor, ranging from 0 to 1, gauges the significance of upcoming rewards. A steeper discount factor denotes a scenario where imminent rewards hold equivalent importance to those expected in the future.

The agent's goal within an MDP is to find an optimal policy π^* , which is a function defining the best action to take in each state so that the cumulative future reward is maximized. This reward is often discounted over time, giving more importance to immediate rewards over distant future rewards.

RL Problem formulation as MDP

The process of reinforcement learning in an MDP can be described as follows:

- The agent selects an action a according to the current policy $\pi(s)$.
- The environment transitions to a new state s_{t+1} following the transition probabilities $P(s_{t+1}|s, a)$.
- The reward function generates an immediate reward r according to $R(s, a, s_{t+1})$.
- The agent receives the new state s_{t+1} and the reward r .

This cycle goes on until a concluding state is arrived at, signifying the termination of the episode. The aim is to gradually optimize the policy π in a way that the anticipated cumulative discounted reward is maximized. [44]

Over the years, many algorithms have been proposed to solve MDPs, ranging from traditional methods like value iteration and policy iteration to more recent techniques using deep learning like Deep Q-Learning mnh, Advantage Actor-Critic methods [34], and many more.

2.1.8 RL Algorithms

Reinforcement Learning (RL) algorithms are classified into following:

Model-based vs Model-free Algorithms

Model-based Algorithms: These algorithms utilize environment's model, which describes the dynamics (physics) of the environment. This model predicts next state and corresponding reward based on the give current state and action. The model can either be provided a priori or learned from agent's interaction with the environment. This class of algorithms uses this model to plan by considering future possibilities before taking an action. Examples of model-based algorithms include Dyna-Q, Monte-Carlo Tree Search (MCTS), and various forms of Planning by Dynamic Programming (DP).[50]

Model-free Algorithms: These algorithms do not require a model (physics) of the environment. Instead, they learn solely from the experience gained by interacting with the environment. They typically require more samples to learn effectively compared

to model-based methods. Q-learning, State-action-reward-state-action (SARSA), and Deep Q Network (DQN) are well-known examples of model-free algorithms.[47]

Value-based vs Policy-based Algorithms

Value-based Algorithms: Value-based methods learn "value function", which is a measure of how suitable a particular state or action is. The policy, which determines behavior of agent, is derived from the value function. In other words, the agent policy is to choose action that maximizes the value function. Prominent examples include Q-learning and DQN.[62]

Policy-based Algorithms: These algorithms directly parameterize and learn policy without explicitly learning the value function. Policy gradient methods like REINFORCE and actor-critic methods like Advantage Actor Critic (A2C) and Proximal Policy Optimization (PPO) fall under this category.[39]

Actor-Critic Algorithms: These methods have strengths of both value-based and policy-based methods. They have two components: an actor that maintains a policy (how the agent should behave) and a critic that estimates the value function (how good the chosen action is). The critic informs the actor how to update the policy.[41]

On-policy vs Off-policy Algorithms

On-policy Algorithms: On-policy methods learn the policy's value used to choose actions based on give states. In other words, data used to update the policy must be collected using the current policy. Examples of on-policy algorithms include SARSA and A2C.

Off-policy Algorithms: Off-policy methods can find optimal policy regardless of the policy used to gather the data. They can learn from actions taken by an exploratory or even a random policy. This property makes off-policy methods more flexible and widely used in practice. Q-learning and DQN are examples of off-policy methods.

Each category of RL algorithms has its strengths and weaknesses, and the choice between them depends on the specifics of the task at hand, including the complexity of the environment, the availability of a model, dimensionality of the action and state, and computational resources available.[28]

2.1.9 RL Algorithm Considered In This Work

In this work, we consider different variants of RL algorithms such as Deep Q-Learning, and Double Deep Q-Learning for traffic light control signal timing optimization. These algorithms demonstrate their ability to adapt to real-time changes and make more informed decisions, thus enhancing the overall traffic flow.

Q-Learning

Q-Learning is a form of model-free reinforcement learning, introduced by Watkins in 1989. It provides a way to find an optimal policy for an agent interacting with an environment modeled as a Markov Decision Process (MDP). The objective of Q-Learning is to learn a policy that allows the agent to maximize the expected total reward over all successive steps, starting from the current state.[1]

Q-Learning uses a table, the Q-table, to store Q-values. The Q-value, or action-value, $Q(s, a)$, denotes the expected return or future reward for taking action 'a' in state 's' following a given policy π . The Q-value essentially answers the question: "What is my expected reward, given I am in a certain state, and I take a specific action?"

The Q-value is learned through iterative updates, making use of the Bellman equation as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)] \quad (2.1.1)$$

Where:

- α is the learning rate, which determines to what extent the newly acquired information overrides the old information.
- $R(s, a)$ is the immediate reward for taking action 'a' in state 's'.
- γ is the discount factor, determining the importance of future rewards compared to immediate rewards.
- $\max_{a'} Q(s', a')$ represents the estimate of optimal future value, given the next state 's' and all possible actions 'a'.

Algorithm 1 Q-Learning

Set values for learning rate α , discount factor γ , and initialize the reward matrix R .

Initialize $Q(s, a)$ to zeros for all states s and actions a .

for each episode **do**

 Select a starting state s randomly.

for each step of the episode **do**

 Choose an action a from state s using policy (e.g., ϵ -greedy or Boltzmann).

 Execute action a , obtain reward r from R , and observe the next state s' .

 Update $Q(s, a)$ using the Q-learning update rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)].$$

 Update the current state: $s \leftarrow s'$.

end for

 Continue until s is a terminal state.

end for

Q-learning algorithm, in its simplest form, is a table-filling method where the agent fills up a Q-table over a large number of episodes, by taking actions, observing rewards, and updating Q-values. Once the table is reasonably accurate, the agent can start to use it for decision making by simply choosing action with highest Q-value in any given state.

Deep Q-Learning

While Q-Learning is highly effective for environments with lower dimension state and action spaces, it encounters scalability issues for higher dimensional state and action spaces because of difficulty in maintaining and updating a Q-table. Deep Q-Learning (DQN), first introduced by researchers at DeepMind, overcomes this limitation by replacing the Q-table with neural network that approximates the Q-value function. Therefore, it is suitable for environments with high dimensional or even continuous state and action spaces.[\[56\]](#)

In DQN, deep neural network is used to approximate our optimal Q-value function. This network takes environment's state as input and outputs estimated Q-values for all agent's actions. By using a neural network, DQN can generalize Q-values over many states, bypassing the need for a lookup table.

The neural network is trained to lower the difference between the predicted Q-value and

the target Q-value, given by the Bellman equation. This is achieved through backpropagation and an optimization method, for example stochastic gradient descent (SGD), RMSprop or Adam. The loss is given by:

$$\text{Loss} = \frac{1}{N} \sum (R(s, a) + \gamma \max_{a'} Q(s', a', \theta) - Q(s, a, \theta))^2 \quad (2.1.2)$$

Where:

- N : number of training samples.
- $R(s, a)$: immediate reward.
- $\max_{a'} Q(s', a', \theta)$: It represents the estimate of optimal Q value, given next state s' and all possible actions a' , following the current policy parameterized by θ .
- $Q(s, a, \theta)$: Predicted Q-value of the action a in the state s .

Algorithm 2 Deep Q-Learning

Initialize deep Q-network $Q(s, a, \theta)$ with random weights θ .

Initialize target Q-network $Q(s, a, \theta^-)$ with weights $\theta^- = \theta$.

for each episode **do**

 Initialize state s .

 Preprocess initial state to obtain the corresponding input for the network x_1 .

for each step of the episode **do**

 Choose action a from state s using policy derived from Q (e.g., ϵ -greedy).

 Execute action a in emulator and observe reward r and next state s' .

 Preprocess state s' to obtain the corresponding input x_{t+1} .

 Store transition (x_t, a, r, x_{t+1}) in replay memory D .

 Sample random minibatch of transitions (x_j, a_j, r_j, x_{j+1}) from D .

 Set $y_j = \begin{cases} r_j & \text{for terminal } x_{j+1} \\ r_j + \gamma \max_{a'} Q(x_{j+1}, a', \theta) & \text{for non-terminal } x_{j+1} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(x_j, a_j, \theta))^2$ with respect to the weights θ .

 Every C steps reset $Q(s, a, \theta^-) = Q(s, a, \theta)$.

end for

end for

Double Deep Q-Learning

While Deep Q-Network (DQN) significantly improved reinforcement learning agents ability to handle large or even continuous state and action spaces, it has a significant drawback known as the overestimation bias. This bias arises because DQN tends to estimate higher Q-values than the true Q-values due to its use of the maximum operator in the Bellman equation for updating estimates of Q-value.

Double Deep Q-Learning (Double DQN or DDQN), proposed by [33], presents a solution to mitigate the overestimation bias inherent in the standard DQN. The key insight behind Double DQN is to decouple action selection from action evaluation process during estimation of Q-value, thereby reducing overoptimistic Q-value estimates produced by DQN.

In DDQN, one network (usually referred to as the online network) is used for action selection, and another network (typically the target network) is used for evaluation of Q-value of that selected action. This results in new target for the Q-value update:

$$\text{Loss} = \frac{1}{N} \sum (R(s, a) + \gamma Q(s', \arg \max_{a'} Q(s', a', \theta), \theta') - Q(s, a, \theta))^2 \quad (2.1.3)$$

Here:

- $\arg \max_{a'} Q(s', a', \theta)$ chooses the action that maximizes Q-value in next state s' , as estimated by the online network.
- $Q(s', \arg \max_{a'} Q(s', a', \theta), \theta')$ estimates Q-value of the chosen action in next state s' , as estimated by given target network.

By decoupling action selection and evaluation, DDQN reduces overestimation of Q-values and tends to provide more accurate and stable learning than DQN. It's worth mentioning that, similar to DQN, DDQN also utilizes techniques like experience replay and target network for stabilizing learning process.

2.2 Survey of related work

The problem of traffic flow optimization is a longstanding challenge within the field of transportation research. Over the past several decades, numerous methods have been

Algorithm 3 Double Deep Q-Learning

Initialize deep Q-network $Q(s, a, \theta)$ with random weights θ .

Initialize target Q-network $Q(s, a, \theta^-)$ with weights $\theta^- = \theta$.

for each episode **do**

 Initialize state s .

 Preprocess initial state to obtain the corresponding input for the network x_1 .

for each step of the episode **do**

 Choose action a from state s using policy derived from Q (e.g., ϵ -greedy).

 Execute action a in emulator and observe reward r and next state s' .

 Preprocess state s' to obtain the corresponding input x_{t+1} .

 Store transition (x_t, a, r, x_{t+1}) in replay memory D .

 Sample random minibatch of transitions (x_j, a_j, r_j, x_{j+1}) from D .

 Set $y_j = \begin{cases} r_j & \text{for terminal } x_{j+1} \\ r_j + \gamma Q(x_{j+1}, \arg \max_{a'} Q(x_{j+1}, a', \theta), \theta^-) & \text{for non-terminal } x_{j+1} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(x_j, a_j, \theta))^2$ with respect to the weights θ .

 Every C steps reset $Q(s, a, \theta^-) = Q(s, a, \theta)$.

end for

end for

proposed to tackle this issue, with varying degrees of success. Despite the ongoing development of new strategies, the complexities and unpredictable nature of traffic flow patterns presents this a very complex area of study.

Webster [16] was one of the pioneering figures in traffic optimization, employing mathematical methods to calculate optimal phase time, which at that time represented a significant leap forward. However, as traffic volumes increased at intersections, Webster's approach needed adaptations and improvements, prompting Robertson [9] to make modifications to the original methodology.

The advent of advanced sensor and computing technologies ushered in a new era of solutions. Fuzzy control methods [14] and genetic algorithms [60] emerged as promising approaches for traffic control systems. These solutions, reliant on historical data, experienced deteriorating performance when confronted with changing traffic patterns. Hence, the need for a traffic control system capable of learning from changes in traffic behavior became apparent.

Reinforcement learning has become a key area of focus for traffic optimization researchers in recent years [67]. The work by [22] provides a comprehensive overview of reinforcement learning techniques applied to traffic control from 1997 to 2010. These techniques employed Q-tables and linear functions to determine Q-values. Various algorithms, state spaces, action spaces, reward functions, and simulation software have been suggested in context of reinforcement learning for addressing traffic light control challenges.

2.2.1 State Space Representation

One key aspect of reinforcement learning solutions for traffic light control lies in the representation of state space. There are three general categories of state space representations utilized in traffic control research to date.

The first category employs features hand-engineered by human experts. These features could be queue lengths of vehicles on the incoming roads at an intersection [64] or a discretization of incoming lanes into individual cells [54]. Each lane cell is then mapped onto a vector cell that indicates presence of a vehicle (1) or its absence (0).

Second category uses vector encoding for the positions and relative velocities of vehicles. Some researchers have even included the traffic light phase in their state representation

along with the positional and relative velocity data [48].

The third category uses image-based features to represent the state [57]. This approach involves taking a snapshot of the intersection’s current state. In some cases, four sequential images are stacked together to represent the state and to provide a sense of motion. For a more comprehensive state representation, queue length, accumulated waiting time, current and next light phase, and an image of the intersection can all be combined, as demonstrated in the work by [65].

Understanding these categories of state space representations helps to create a roadmap of how different research approaches have been utilized and could be improved upon in future traffic control systems.

2.2.2 Defining Action Spaces

The definition of action space in reinforcement learning varies based on degree of flexibility allowed to the agent. These variations lead to different degrees of responsiveness and adaptability to changing traffic conditions.

In a lower-flexibility action space, the reinforcement learning agent operates within a predefined set of traffic light phases [58]. The agent merely selects from these predefined phases, and each phase is active for a fixed duration. This rigidity limits the system’s adaptability to fluctuating traffic conditions.

Medium-flexibility action spaces offer more adaptability. In this setup, the agent decides whether or not to change the current phase [37]. While the cycle of phases remains fixed, each phase duration can vary based on the current state. This flexibility allows for more responsive traffic management, though the fixed phase cycle still places limits on the system’s adaptability. In this case, agent can also choose sequences of traffic light signal sequences that are not feasible for a real life scenario.

High-flexibility action spaces provide the most dynamic environment. In this case, the agent can select not only which phase to activate but also how long that phase should last [10]. This full control over both phase selection and duration allows the system to respond most accurately to real-time traffic conditions.

2.2.3 Reward Function Formulation

The reinforcement learning agent gets feedback from environment in form of rewards after taking an action in given state. This reward function is typically defined based on performance measures such as vehicle wait time, queue length, and throughput.

Many recent studies have used the cumulative vehicle delay between the two actions as a reward function [52] [27]. The delay is measured as the number of seconds that a vehicle remains stationary at an intersection, effectively capturing the inefficiencies of traffic light phasing.

In contrast, other researchers have used the cumulative waiting time as a reward function [64] [43]. The cumulative wait time can be defined as sum of wait times for all vehicles entering the network, providing a more holistic view of the traffic system's efficiency.

In some cases, researchers employ a reward function that seeks to balance the queue lengths on all sides of an intersection [27]. This balance-focused reward function aims to ensure fair distribution of delays across all traffic directions.

Each approach to defining the reward function provides its unique perspective and emphasis on different aspects of traffic management, influencing the behavior and effectiveness of the reinforcement learning agent.

2.2.4 Reinforcement Learning Approaches

Recent research in the field of traffic control optimization has been based on various reinforcement learning algorithms, each with different artificial neural network architectures. Their common goal is to ensure high-performing traffic control systems.

Deep Q-learning is one of most widely used algorithms used in this domain. It employs a deep neural network for approximation of Q-values for a given state, enabling RL agent to learn optimal behavior [26]. Depending on the type of state representation, different architectures of neural networks can be used. For instance, Artificial Neural Networks (ANNs) have been used for vector representation of states [24], while Convolutional Neural Networks (CNNs) are being utilized for image-based state representations [20] [29].

Researchers like Gao et al. [36] and Gender and [68] implemented CNNs to extract features from a matrix representation of vehicle position and velocity. The CNN output

was fed into a fully connected layer that was also fed with with the current phase information. This layer, in turn, was connected to an output layer yielding Q-values for all potential actions. The performance of these approaches significantly outpaced traditional traffic control systems.

Mousavi, Schukat, and Howley [38] presented two alternative methodologies to address traffic control: value-based and policy-based approach. In the value-based approach, the action values were estimated by minimizing the error of Q-values. In contrast, the policy-based approach updated policy parameters to learn the policy. Both approaches used a CNN as a function approximator to extract features from intersection images. The outcome was either Q-values for actions (value-based approach) or a probability distribution over actions (policy-based approach). Both methods achieved commendable performance without stability issues.

Li Li [42] proposed a novel method using deep stacked autoencoders (SAEs) to determine the appropriate action in the current state. Autoencoders are neural networks that aim to set the target output identical to the inputs. This approach employed a specific loss function to quantify the error between predicted Q-values and the target Q-values, with the aim for minimizing this error using an optimizer. The results suggested that the deep stacked autoencoder approach surpassed traditional traffic light control systems.

2.2.5 Convergence in Q-Learning

One of the main challenges in Q-learning is ensuring that the Q-values converge to stable values. This lack of convergence can occur when agent fails to explore entire state space or when the neural network's weights are not properly updated. Various hyperparameters, such as learning rate, batch size, and exploration rate, also influence this convergence problem. [32]

Several solutions are proposed in the literature to address this issue. Notably, these include (1) experience replay, (2) use of a target network, and (3) epsilon-greedy exploration policy. Experience replay employs random sampling of training examples to eliminate correlation between states. The target network approach uses two neural networks: one for training and the other for predicting Q-values. The epsilon-greedy exploration policy sets an exploration rate that permits random action at the start of training to explore a more extensive state space [53].

Wei et al.[59] suggested a technique that utilized multiple memory containers for storing training examples. Each memory container stored training examples based on a specific phase-action pair. During training, an equal number of training examples are sampled from all containers, thereby maintaining training stability.

2.2.6 Simulation software

Traffic simulation software tools are crucial for developing and testing reinforcement learning models in traffic management. Here are some widely-used ones:

Simulation of Urban MObility (SUMO): SUMO is a free and open source traffic simulation suite which is highly portable and allows for microscopic, mesoscopic, and macroscopic simulations. It is well-regarded in the research community. It supports multiple simulation scenarios and can integrate with other applications using its robust API. It's also frequently updated, thus benefiting from continuous improvements and features. [17]

Green Light District (GLD) simulator: GLD is a simple and open-source software primarily focused on simulating traffic lights. Being open-source, it's free to use and modify. Its simplicity also makes it easy for beginners to use.[13]

CORSIM: CORSIM is a paid microscopic simulation software that simulates surface street and freeway networks. It is known for its detailed simulation and analysis capabilities and includes a broad array of features for traffic analysis.[6]

CityFlow: CityFlow is an open-source traffic simulation framework specifically designed for reinforcement learning scenarios. It is built for large-scale city traffic scenarios and supports multi-agent reinforcement learning. It is also designed to be more efficient than traditional simulators.[55]

PTV Vissim: PTV Vissim is a comprehensive traffic simulation software that allows for multi-modal traffic modeling. It offers detailed and realistic simulations and is suitable for various traffic engineering, planning, and operational analysis tasks.[63]

Paramics Discovery: Paramics Discovery is a paid software used for microscopic traffic and pedestrian simulation. It offers comprehensive and realistic modeling of various transport modes, including cars, cyclists, and pedestrians.[23]

Aimsun: Aimsun is a comprehensive modeling platform for traffic management and planning. It supports multi-resolution modeling (macroscopic, mesoscopic, and microscopic) and is well-suited for large-scale projects.[\[15\]](#)

Design of the Reinforcement Learning Agent

This research employs artificial intelligence techniques to examine potential improvements in traffic flow at intersections. The study will utilize traffic simulation software, in which a reinforcement learning (RL) agent, following specific rules and decision-making abilities, seeks to optimize the flow of traffic. To train this agent to select the most effective action under any traffic conditions, specific learning techniques grounded in reinforcement learning and deep learning theories are utilized.

To properly design a reinforcement learning agent, certain key components must be defined and understood. These components include environment, state space, action space, reward function, and learning mechanism itself. This chapter provides an in-depth discussion of these elements. Table 3.1 outlines all the notations used throughout this chapter for quick reference.

Table 3.1: Notations used in Chapter 3

Notation	Meaning
\mathcal{ITL}	Intersection of Traffic Light
s	Individual State
a	Individual Action
r	Individual Reward
ts	Sampling Time Step
\mathcal{A}	Set of Possible Actions
ql	Queue Length
hl	Hidden Layer

The Intersection Traffic Light (\mathcal{ITL}) represents RL agent in the environment, where RL agent interacts with traffic conditions and controls traffic signals. An individual state (s) captures current state of environment, including such information as vehicle positions and traffic signal statuses. An action (a) denotes a decision made by the agent that can affect the environment, for example, changing a traffic light from green to red. Rewards (r) provide feedback to the agent based on the results of its actions, with better outcomes leading to higher rewards.

The sampling time step (ts) is the interval at which the agent takes actions and receives feedback. The set of possible actions (\mathcal{A}) includes all actions that agent can perform in response to given state. The queue length (ql) represents count of vehicles waiting at traffic signal intersection, and hidden layer (hl) is a part of the neural network used by the agent for learning.

In the following sections, each component will be described in more detail. Each explanation will also be accompanied by relevant references to research papers to provide a comprehensive overview of these components within the context of reinforcement learning.

3.1 Problem Definition

In the given research problem, the agent's environment consists of an intersection with multiple incoming and outgoing lanes. Each incoming road is regulated by traffic lights

(ITL) controlled by an intersection controller that manages the phase sequence of traffic light controller.

For this research work, the term 'reinforcement learning agent' or simply 'agent' signifies that the intersection controller has been replaced by a reinforcement learning algorithm responsible for controlling the traffic lights. However, the term 'Non RL agent' indicates that intersection controller is replaced by a fixed traffic light controller, where all phases have a predetermined sequence and duration.

The agent interacts with its environment through a state ' s ', takes an action ' a ', and gets a reward ' r '. The simulation operates in time steps, each corresponding to a one-second duration. Following a certain number of simulation steps, the agent's sampling timestep ' ts ' occurs. During each sampling timestep, agent interacts with environment, attains the state ' sts ', and calculates the reward ' rts '. Based on the current state ' sts ' and prior knowledge, traffic light

3.2 Environment

The environmental setting in which the reinforcement learning (RL) agent operates is fundamental to understanding its function and performance. As previously explained, in the context of this research, the environment is composed of intersections, with multiple intersections forming a network. Networks can include a single agent or multiple agents, with the latter scenario referred to as a multi-agent environment. For this research, we focus on both these configurations to provide a comprehensive analysis of the RL agent's performance.

3.2.1 Single Agent Environment

Visualized in Figure 3.1, a single-agent environment is shown as a standard four-way intersection with four access points in the four cardinal directions. These intersection arms are central to the simulation, each equipped with four lanes that vehicles can utilize to approach and subsequently leave the intersection.

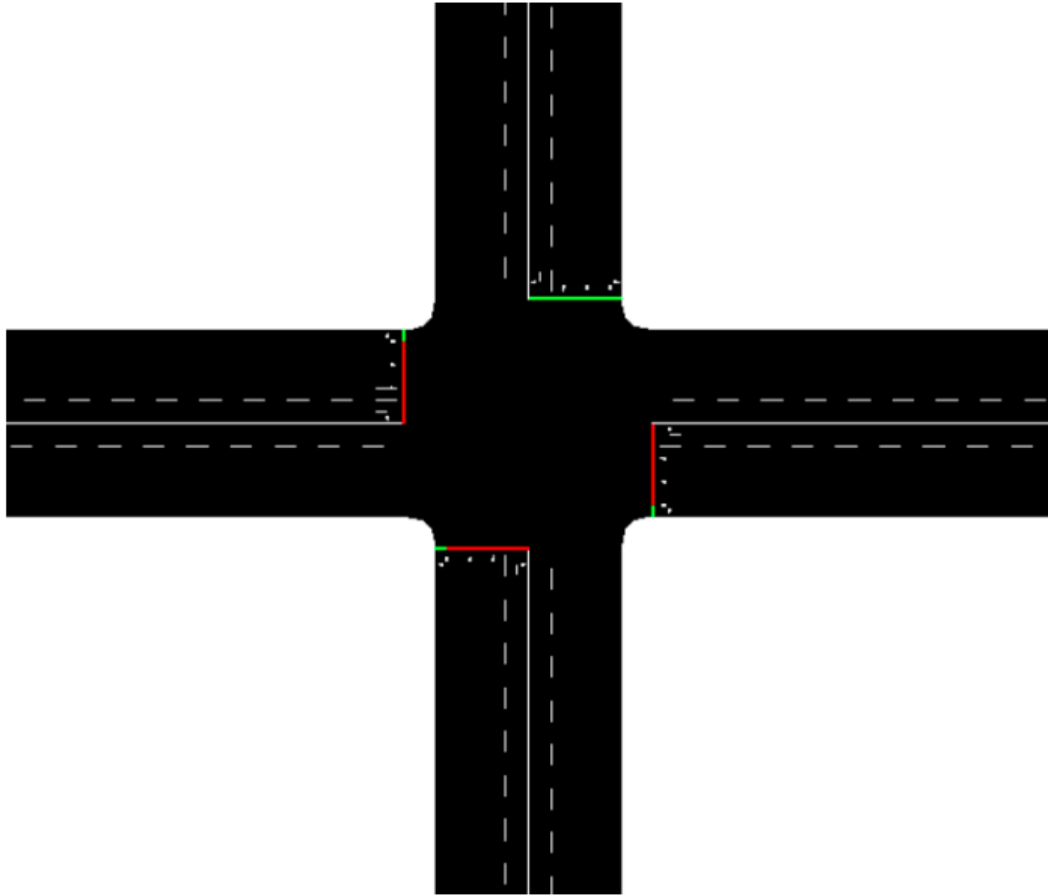


Figure 3.1: Single Agent Environment

The length of each arm is set to 750 meters, measured from the origin point of a vehicle to the intersection’s stop line. This measurement mirrors typical urban road scenarios, thus providing a practical context to our simulations.

The single-agent environment introduces an element of uncertainty with vehicles being randomly introduced in any incoming lanes, each with a standard initial speed of 10 m/s. Upon approaching a given intersection, the vehicle determines the preferred lane based on its pre-programmed destination. Adhering to left-hand driving rules, the possible routes that vehicles can opt for at the intersection include:

1. Right turn: Vehicles intending to make a right turn choose the right-most lane.
2. Go straight: Vehicles moving straight ahead select one of the middle two lanes.
3. Left turn: Vehicles turning left opt for the left-most lane.

Building the single agent environment is made possible by the Simulation of Urban

MObility (SUMO) version 1.2.0. This open-source software is highly valued for its microscopic, multimodal, and continuous traffic simulation capabilities. It provides a realistic portrayal of traffic at intersections and within broader networks, while allowing the user to tweak minute network and traffic generation details.

The intersection encompasses 12 distinct traffic lights, as illustrated in Eq. 3.1. The subscript within the equation symbolizes the position of each traffic light. For instance, t_{lnl} signifies the traffic light situated on the west side of the intersection, regulating traffic intending to make a left turn. In a similar fashion, t_{lns} denotes the traffic light on the east side of the intersection, overseeing traffic aiming to proceed straight. This rule applies uniformly to all traffic lights as delineated in Eq. 3.1.

$$ITL = \{t_{lnl}, t_{lns}, t_{lnr}, t_{lwl}, t_{lws}, t_{lwr}, t_{lel}, t_{les}, t_{ler}, t_{lsl}, t_{lss}, t_{lsr}\} \quad \text{Eq. 3.1}$$

Each traffic light can adopt one of three potential states, as displayed in Table 3-2.

Description	Traffic Signal Character
Green light	g
Yellow light	y
Red light	r

Table 3.2: Traffic Light States

In SUMO, the key component of traffic flow regulation is the traffic light system. Each lane on the road features a separate traffic light. The status of these traffic lights at each sampling timestep 'ts' is visually represented by colors on stop line.

The operational logic of traffic lights follows a set of rules:

- The transition of traffic light colors follows the sequence: green-yellow-red-green.
- There cannot be a scenario where all traffic lights at the intersection are red simultaneously.
- Only one side of the intersection can have all traffic lights in a green or yellow state at any given time.
- The yellow light's duration remains constant at 4 seconds.

- The green light duration varies, fixed at 30 seconds for the turn-based agent and variable for the time-based agent.

These rules help define the concept of a 'phase,' which is the combination of all traffic light states for a fixed interval. This is integral to traffic light schedules and thereby to the overall traffic flow.

Vehicles in this environment are generated randomly, each carrying unique attributes and following distinct driver behavior models. This diversity adds realism to the simulation, further contributing to the reliability of the research outcomes.

The attributes of the vehicles are as follows:

Attribute	Value
Length	5 m
Width	1.8 m
Minimum gap	2.5 m
Maximum speed	25 m/s
Maximum acceleration	1 m/s ²
Maximum deacceleration	4.5 m/s ²

Table 3.3: Vehicle Attributes

The Krauss model and the LC2003 model guide the vehicles' driving behavior.

3.3 State Representation

The agent's state representation is an important component in reinforcement learning, representing the environmental status at a specified sampling timestep t_s , denoted as s_{t_s} . The formulation of the state has a substantial impact on the efficacy of the RL agent. Although the state in traffic control scenarios is continuous, it can be made discrete by sampling at discrete time intervals. This research employs the measure of queue lengths at traffic junctions as a representation of the state space.

The term "queue length" refers to the aggregate count of vehicles that are at a standstill at the intersection. With regards to an agent operating on a turn-based system, the state is defined by gathering queue lengths from each side of the intersection. The number

of queue lengths taken into account is contingent upon the number of sides that the intersection has. For example, in the case of an intersection with four roads converging, the state would be composed of four queue lengths. On the other hand, the state for the time-based agent is the queue length of one intersection side as its role is to alter the phase duration of that specific side based on the queue length.

Various state types are used in the literature to optimize traffic flow at intersections. One such widely used state representation is the image-based state. These states are better as compared to the state representation used in this research because they do not require any human-crafted feature. However, this approach is not chosen due to practicality concerns, as it's unrealistic to obtain top view images of real-world intersections (it is also difficult to simulate this state). Instead, the state used in this thesis can be easily procured using a set of cameras at the intersection but integrating a vehicle counter.

Nonetheless, the scalar state acquired is unrefined and not ideal for training the RL agent. To establish a connection between state-action pairs, the use of a deep neural network becomes necessary. Training this network with scalar inputs is difficult due to the low input dimensions. Therefore, human-crafted features such as binary encoding are used to increase the state's dimension and facilitate training.

Different types of binary encodings are employed to find the optimal state representation that enhances agent performance. Key elements of each encoding include the encoding size and encoding weights. The encoding size refers to the count of cells present in the encoding matrix, while encoding weights signify the least count of vehicles necessary to fill a cell in the encoding matrix with 1's.

The pseudo-code in Algorithm 1 delineates each procedure to populate the encoding matrix. For instance, consider Figure 3-9 (which will be included in the actual thesis). This figure shows a type-1 encoding matrix with specific encoding weights for each cell. This particular matrix is capable of handling up to 492 vehicles on a single side of the intersection. When the queue length on one side reaches 492 vehicles, every cell in the encoding matrix for that side would be filled with 1's. Conversely, when no vehicles are present, all cells within the encoding matrix will be populated with 0's.

Algorithm 4 Finding State Encoding

```

1: procedure GETSTATEENCODING
2:   Initialize the state matrix with zeros:  $M \leftarrow \text{zeros}(4, 12)$ 
3:   Initialize the encoding weights matrix:  $E \leftarrow \text{EncodingWeights}(4, 12)$ 
4:   Find the queue length:  $ql \leftarrow \text{QueueLength}$ 
5:   for  $j = 1$  to  $M.\text{columns}$  do
6:     for  $i = 1$  to  $M.\text{rows}$  do
7:        $C \leftarrow E[i, j]$ 
8:       if  $ql \geq C$  then
9:          $ql \leftarrow ql - C$ 
10:         $M[i, j] \leftarrow 1$ 
11:      end if
12:    end for
13:  end for
14:  return  $M$ 
15: end procedure

```

This discussion implies that the encoding matrix's first columns are populated first, reflecting real-world traffic scenarios where drivers prioritize filling empty spaces at the intersection first. The state representation design also extends to type-2, type-3, type-4, and type-5 encoding matrices, which have their own unique capacities and bit sizes. In the end, the encoding matrices for all directions at the intersection, pertaining to the turn-based agent, are gathered and merged to form a unified vector. This consolidated vector characterizes the intersection's state, serving as a vital input for agent training.

1	2	8	12	16	24	28	32
1	2	8	12	16	24	28	32
1	2	8	12	16	24	28	32
1	2	8	12	16	24	28	32

Table 3.4: Figure 3-9: An encoding matrix with a 32-bit capacity, capable of managing up to 492 vehicles.

1	1	2	4	8	10	12	14
1	1	2	4	8	10	12	14
1	1	2	4	8	10	12	14
1	1	2	4	8	10	12	14

Table 3.5: Figure 3-10: An encoding matrix with a 32-bit capacity, capable of managing up to 200 vehicles.

1	2	4	8	12	14	16	20
1	2	4	8	12	14	16	20
1	2	4	8	12	14	16	20
1	2	4	8	12	14	16	20

Table 3.6: Figure 3-11: An encoding matrix with a 32-bit capacity, capable of managing up to 300 vehicles.

1	1	1	2	2	2	4	4	8	8	8	10
1	1	1	2	2	2	4	4	8	8	8	10
1	1	1	2	2	2	4	4	8	8	8	10
1	1	1	2	2	2	4	4	8	8	8	10

Table 3.7: Figure 3-12: An encoding matrix with a 48-bit capacity, capable of managing up to 200 vehicles.

1	1	2	2	4	4	8	8	10	12	12	12
1	1	2	2	4	4	8	8	10	12	12	12
1	1	2	2	4	4	8	8	10	12	12	12
1	1	2	2	4	4	8	8	10	12	12	12

Table 3.8: Figure 3-13: An encoding matrix with a 48-bit capacity, capable of managing up to 300 vehicles.

3.3.1 Action Space

In reinforcement learning, an action signifies the task performed by the agent in the environment. In the context of the traffic management system, undertaking an action

equates to triggering the green light phase for one direction at the intersection, lasting for a predefined period. Analogous to the state, an action is also executed at a sampling timestep, t_s .

For this thesis research, we are focusing on a turn-based agent. The turn-based agent's action space comprises selecting which side of the intersection receives the green phase. The number of potential actions for the turn-based agent depends on the intersection sides and thus varies accordingly. For instance, the turn-based agent will have three actions available for a three-way intersection, and four actions for a four-way intersection. To illustrate, consider the action space of a turn-based agent operating on a four-way intersection, represented as:

$$A_{turn} = \{N_G, W_G, E_G, S_G\}$$

In the equation above, the actions are defined as follows:

- North-Green (NG): This refers to the activation of the green signal for automobiles positioned at the north end of the crossroads, aiming to continue forward or veer right.
- West-Green (WG): Here, the green phase is engaged for cars at the west end of the intersection with the intention to carry on straight or execute a right turn.
- East-Green (EG): This indicates the operation of the green phase for vehicles located at the east side of the junction, planning to drive straight ahead or take a right turn.
- South-Green (SG): This refers to the illumination of the green light for vehicles at the south portion of the crossroads, intending to move forward or turn right.

The visual representation of these actions for a turn-based agent at a four-way intersection is shown in below diagrams showcasing each potential action (North Green, West Green, East Green, and South Green).

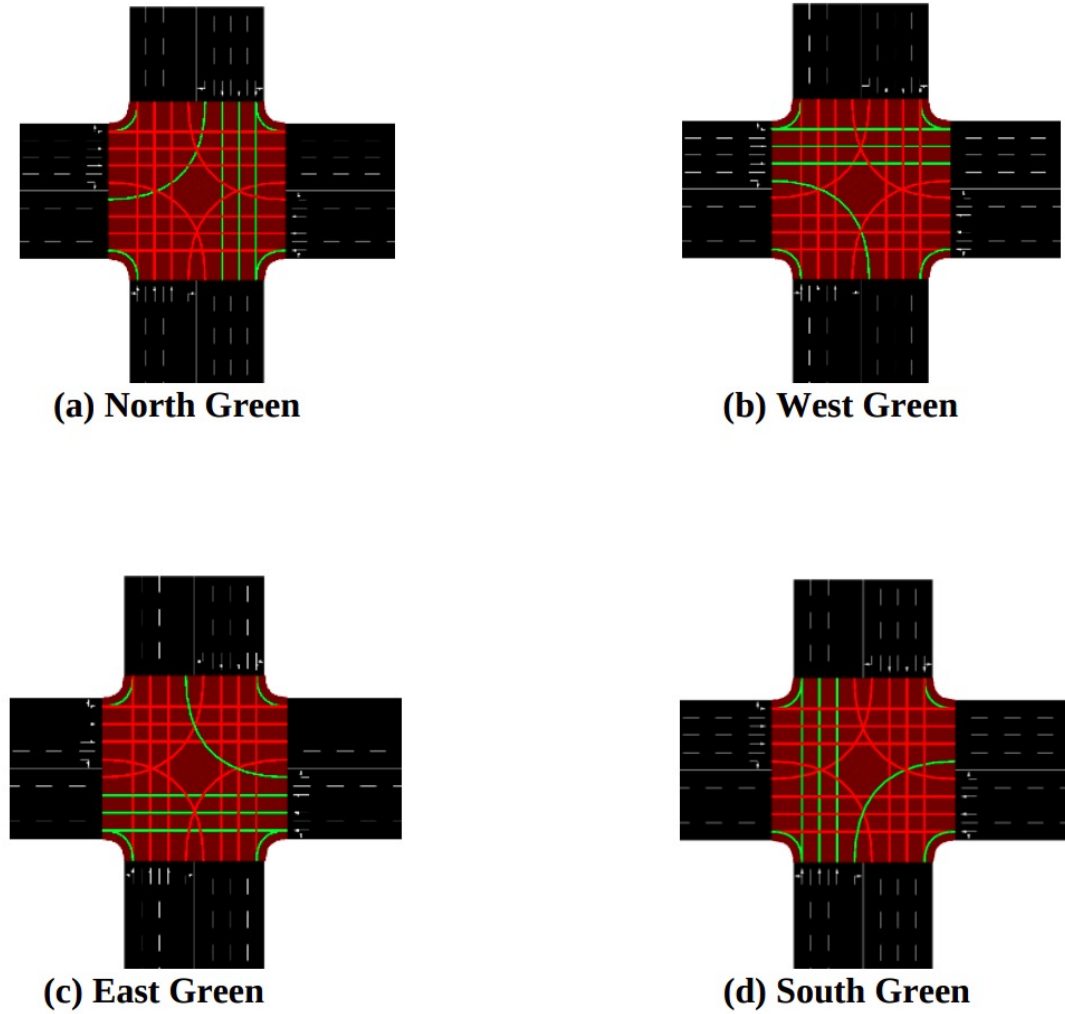


Figure 3.2: Potential actions for a turn-based agent at a four-way intersection

In a non-reinforcement learning environment, a transition phase, represented by the activation of the yellow light at the intersection for a duration of 4 seconds, takes place following every green phase. In the model governed by the turn-based agent, a transition phase is introduced only when the agent's selected action differs from the preceding one. For instance, suppose the action picked by the agent at sampling timestep t_s is a_0 and aligns with the action carried out at the former timestep $t_s - 1$, then no transition phase transpires. However, should the agent select a different action at sampling timestep $t_s + 1$, a transition phase will take place.

3.3.2 Reward Function

In reinforcement learning, the reward function serves as a crucial feedback mechanism, allowing the agent to discern the efficacy of an action executed in a given environment. After an action is taken, the agent acquires a reward, effectively indicating the qualitative aspect of the action, i.e., whether it was beneficial or detrimental. This feedback informs the agent’s model updates for future decision-making, rendering the reward integral to the agent’s training process. Typically, rewards can be either positive or negative, with positive values signaling a favorable action and negative ones indicating an unfavorable action.

This work primarily seeks to enhance traffic flow at intersections. Consequently, the reward function ought to derive from performance measures that can potentially influence traffic flow. Contemporary research has explored various reward functions, and the following are some prominent examples:

Queue Length Reward

This reward is computed from the difference in queue length at two distinct timesteps, ql_{ts-1} (previous timestep) and ql_{ts} (current timestep). A positive reward indicates a decrease in stationary vehicles at the intersection due to effective action by the agent. Conversely, a negative reward signals an increase in stationary vehicles due to a poorly chosen action. The calculation of the reward using queue lengths is represented as follows:

$$r_{ts} = ql_{ts-1} - ql_{ts}$$

Density and Speed

This function uses both vehicle density and mean vehicle speed to compute the reward. Positive rewards reflect a decrease in vehicle density and an increase in mean speed, while negative rewards suggest the opposite. The reward function is given as:

$$r_{ts} = k(\alpha(d_{ts-1} - d_{ts}) + \beta(v_{ts} - v_{ts-1}))$$

where k is a constant, α and β are weights, d denotes vehicle density on all roads, and v signifies the sum of mean speeds on all roads.

Wait Time Reward

This function determines the reward based on the accumulated waiting times of vehicles that are stationary at the intersection. Here, awt signifies the total wait time, computed by adding the wait times of all vehicles halted at all junction sides. The reward r_{ts} is derived by deducting the accumulative wait time of the current timestep from that of the preceding timestep. This function is designed to reward favorable actions with positive values and penalize unfavorable actions with negative ones.

$$r_{ts} = awt_{ts-1} - awt_{ts}$$

This research primarily employs the Wait Time Reward function among the suggested rewards, given that it considers the waiting times of vehicles at junctions. The hypothesis is that an efficient intersection minimizes the wait times for vehicles during the green phase. This aspect is not considered in the other two reward functions, hence Wait Time Reward is used in this work.

3.4 Learning Mechanism

The learning mechanism outlines the methodology by which a reinforcement learning (RL) agent discerns the correlation between a state and an action. The Q-learning methodology, an off-policy and model-free RL approach, enables this learning process by assigning Q-values to various actions. These Q-values, representing the quality or “goodness” of an action, are the linchpins for decision-making by the agent. Essentially, the agent assigns Q-values to every possible action for each state in the environment and subsequently selects actions based on these values [25].

A common approach to mapping the state of the environment to corresponding Q-values of actions is the Q-table. This matrix is a two-dimensional structure, where every row corresponds to a unique state within the state space, every column signifies an action within the action space, and each cell encapsulates the relevant Q-value. The essence of Q-learning lies in discerning the Q-values for all plausible actions in a particular state, and the action leading to the highest Q-value is considered the best choice for that state. However, the Q-table approach suffices only for finite state and action spaces. As the size of these spaces grows, employing Q-tables becomes computationally infeasible. In

such scenarios, function approximators are called upon to map the given state of the environment to Q-values of actions [25].

Deep Neural Networks (DNNs) have demonstrated their capabilities as efficient function approximators, and their application can aptly replace Q-tables. DNNs process the state as input and output the predicted Q-values. Specifically, in traffic control scenarios, the action set is finite but the state space is virtually infinite. This makes it imperative to leverage DNNs to establish a correlation between states and actions.

3.4.1 Deep neural network architecture

The reinforcement learning model in this study utilizes a Deep Neural Network (DNN) as a function approximator to develop the relationship between state-action pairs. This DNN is composed of five fully connected hidden layers, each utilizing the Rectified Linear Unit (ReLU) as the activation function. The hidden layers, labelled as h_l , consist of varying quantities of nodes, as illustrated in Figure 3-17, with h_{l1} , h_{l2} , h_{l3} , h_{l4} , and h_{l5} accommodating 512, 512, 512, 256, and 128 nodes, respectively.

The DNN is structured with an input layer that accepts the environmental state and an output layer that produces the Q-values of actions. The number of nodes within the input and output layers is contingent upon the particular RL agent in use.

For a turn-based agent, the nodes in the input layer equal the product of the encoding matrix size and the number of intersection sides. The nodes in the output layer equate to the size of the turn-based action space, represented as A_{turn} . On the other hand, for a time-based agent, the count of nodes in the input layer matches the size of the encoding matrix, and the nodes in the output layer match the size of the time-based action space, represented as A_{time} .

The output layer, being fully connected, incorporates a linear activation unit. This highlights the principle that the activation function of the output layer should not limit the range of output values, particularly in the context of Q-learning.

3.4.2 Training deep neural network

The training of the deep neural network involves training data, a tuple that includes the state, action, reward, and the subsequent state. This data is preserved in a component

known as Experience Replay Memory, capable of storing up to 50,000 training instances. Once the capacity is fulfilled, older training instances are eliminated to accommodate the new ones.

With the initiation of traffic simulation, new training instances are successively added to memory at every sampling timestep t_s . The raw training examples from memory need to be appropriately formatted to comply with the input and output prerequisites of the deep neural network. The state, serving as the network's input, is directly retrieved from the training instance, while the Q-values, forming the output, are determined using the Bellman equation.

A random batch of training instances is extracted from memory and processed through the Bellman equation to calculate target Q-values. The discrepancy between these target Q-values and the Q-values projected by the deep neural network functions as the error metric, which is then used to update the deep neural network's weights using the Adam optimizer.

The training of the deep neural network is significantly impacted by hyperparameters. Key hyperparameters for this training encompass the learning rate and the batch size, set at 0.001 and 64 training instances, respectively. A comprehensive discussion on the training process will be presented in section 4.3.

Experimental Setup

In the previous chapter, the agent's specifications, including state, action, and reward definition, were detailed. The present chapter delves deeper into the agent's operational flow, elaborating on how the components of the reinforcement learning (RL) agent integrate to process state information, train the underlying deep neural network, and subsequently select an appropriate action. This operational loop is executed at every designated sampling timestep, t_s . Figure 4.1 presents the general workflow of a reinforcement learning agent, which applies both to turn-based and time-based agents.

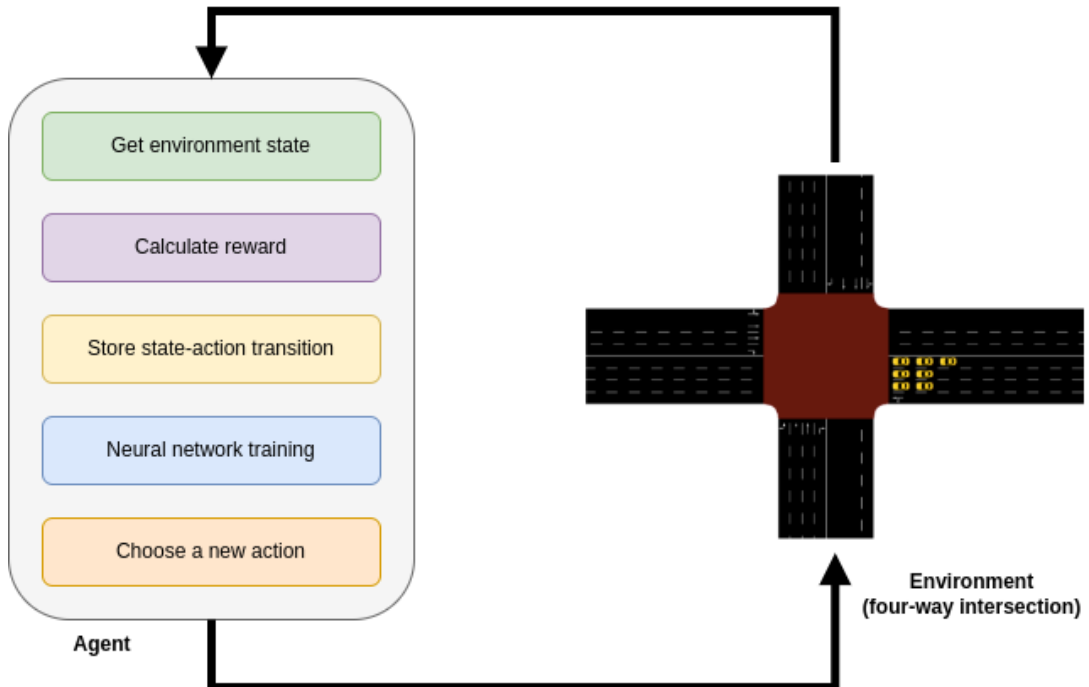


Figure 4.1: Operational Cycle of the Reinforcement Learning Agent

The sampling timestep, t_s , starts after a certain number of simulation steps. At the onset of this timestep, the agent gathers the state of the environment, encoding it following the pre-defined encoding matrix. Subsequently, the agent determines the waiting time for all vehicles halted at the intersection.

The next step involves the calculation of the reward, achieved by deducting the waiting time at the preceding timestep ($t_s - 1$) from the current waiting time at t_s . This approach enables the agent to measure the efficiency of its actions in terms of reducing overall wait time at the intersection.

Now, having the information of the current state, action, reward, and subsequent state, the agent stores this data into its memory. This data serves as the agent’s experience or training examples and is important in neural network training.

At the start of the simulation, the agent’s memory lacks sufficient training examples to initiate learning. Hence, the agent focuses on gathering these training examples in its memory until it reaches a pre-set training threshold. Once this threshold is met, the agent launches the training procedure.

On completion of this cycle, the turn-based agent selects the traffic phase as its action, while the time-based agent decides the duration of the phase to present to the environment.

The remainder of this chapter will elaborate on the salient features of the chosen simulation software, detail the agent’s training procedure, and discuss the generation of various traffic scenarios.

4.1 Urban Mobility Simulation Tools and Techniques

The training process of a reinforcement learning (RL) agent requires a simulation environment. Various tools such as PTV Vissim, Aimsun Live, TSIS-CORSIM, SUMO, PARAMICS, and SimMobility are available for this purpose. For this research, we have chosen to utilize the Simulation of Urban MObility (SUMO) version 1.2.0, given its robust and comprehensive features.

SUMO is a versatile, open-source, microscopic, multimodal, and continuous traffic simulation package that offers a Python interface for enhanced programmability. It necessitates the creation of simulation files in xml format that encapsulate all the relevant

configuration details related to the simulation. An overview of these files and their purposes is tabulated below in Table 4.1.

Table 4.1: Summary of Simulation Files

File Name	Extension
Node	<NAME>.nod.xml
Edge	<NAME>.edg.xml
Type	<NAME>.xml
Network	<NAME>.net.xml
Route	<NAME>.rou.xml
Configuration	<NAME>.sumocfg

While the network, route, and configuration files are essential for executing the simulation, the node, edge, and type files are necessary for constructing the network file. SUMO provides a set of additional tools for editing all aspects of road infrastructure. In the context of this thesis, we have utilized the following:

4.1.1 Netconvert

Netconvert is a command-line utility included with the SUMO package. It converts node, edge, and type files into a network file. This tool can also handle map files from other sources such as OpenStreetMap (OSM) and VISUM-network.

4.1.2 Polyconvert

This command-line tool imports geometrical shapes (polygons or points of interest) from different sources, improving the visual representation of the simulation.

4.1.3 NetEdit

NetEdit is a GUI-based network editor used for modifying existing network attributes. In this research, it was employed for editing road and intersection-related attributes. It proved particularly useful for editing imported networks from other sources such as OSM and VISUM-network.

4.1.4 TraCI (Traffic Control Interface)

TraCI provides access to various simulation parameters, such as vehicle position, velocity, acceleration, and intersection phase. When imported as a module into a Python program, it enables direct access to and manipulation of the simulation environment.

4.1.5 Simulation Options

SUMO offers two types of simulations: terminal-based and GUI-based. The terminal-based version runs in the background and is faster, making it suitable for gathering simulation data and training RL agents. The GUI-based simulation, in contrast, provides a visual representation of the road network and traffic, making it better suited for presentations and demonstrations.

Additional reading for understanding these simulation tools can be found in the SUMO documentation [`sumo`], and for a broader perspective on traffic simulations, we recommend Barceló [46] and the works cited therein.

4.2 Experience Replay: A Key Mechanism in Reinforcement Learning

Experience Replay is an essential technique in reinforcement learning that enhances performance and learning efficiency [25]. Its fundamental principle lies in the storage and subsequent utilization of "experiences" or training instances gathered during the simulation. These experiences, symbolized by \mathcal{H} , form the basis of the learning process.

Each experience, \mathcal{H} , encompasses four elements: the current state (s_t), the action taken in that state (a_t), the subsequent reward received (r_t), and the next state that action leads to (s_{t+1}).

$$\mathcal{H} = \{s_t, a_t, r_t, s_{t+1}\} \quad (\text{Eq. 4.1})$$

An Experience Replay buffer is used to store these instances for future use during training. This storage and retrieval process is depicted in Figure 4.2.

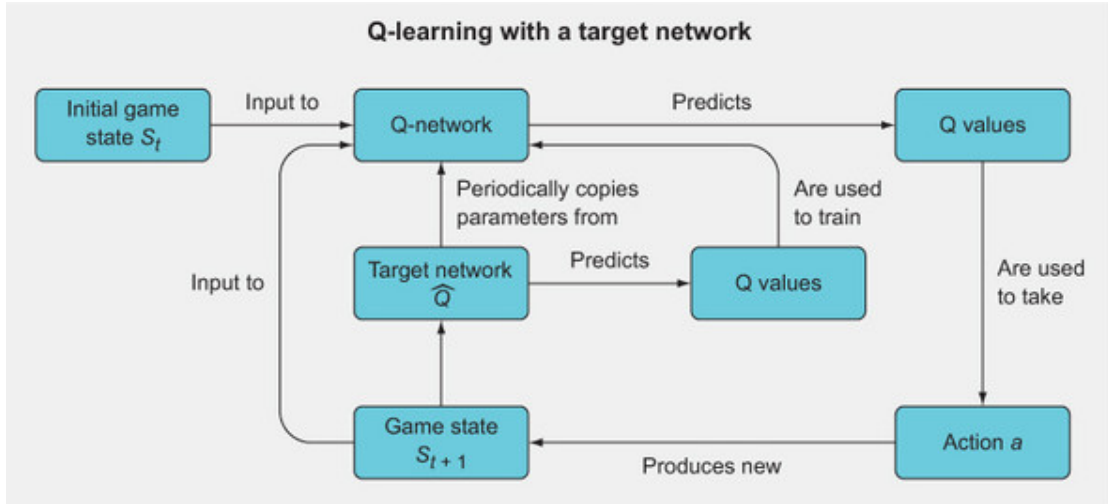


Figure 4.2: Process of Experience Replay

The buffer incorporates two primary parameters: memory size and batch size. Memory size determines the total number of experiences that can be stored, and for the purpose of this study, it is set to 50,000 experiences. Batch size, on the other hand, dictates the number of experiences drawn from the memory for each training step. It's a hyper-parameter, meaning its optimal value isn't predefined but is chosen based on empirical testing. In most recent studies, the batch size falls between 16 and 128 experiences [45]. In our case, we have used a batch size of 64 experiences.

During the simulation, the RL agent starts filling up the memory with experiences. Training commences once the number of stored experiences exceeds the batch size. After this point, the agent retrieves a random batch of experiences from memory for each training step. As the simulation continues, the memory eventually gets filled, triggering a 'first in, first out' strategy, where older experiences are discarded to accommodate new ones. On average, each episode generates about 270 experiences, filling up the memory after approximately 185 episodes.

Despite the availability of other techniques for RL agent training [49] [51], Experience Replay offers several advantages:

- It breaks the correlation between consecutive experiences: In traffic simulations, two states might be correlated, leading to instabilities in training. Experience Replay randomly samples experiences, thereby minimizing such correlations.
- It periodically refreshes the agent's experiences: Over time, the agent might forget

the knowledge gained in earlier stages of training and become biased towards recent states. Random sampling of experiences counters this bias, ensuring a balanced recall of older and newer experiences.

- It allows efficient use of memory: By discarding older experiences, Experience Replay maintains a finite, manageable size of memory, making it an efficient solution for storing and accessing training examples.

Both turn-based and time-based agents employed in this study use the same methodology for storing and retrieving experiences.

4.3 Training Methodology of Reinforcement Learning Agent

This section provides a comprehensive breakdown of the process. During simulation, for every sampling time step, t_s , an instance of training data, denoted by \mathcal{H} , is stored in the Experience Replay buffer, \mathcal{M} . Training begins when the buffer accumulates sufficient examples to form a batch, \mathcal{B} . A batch is a randomly selected subset of examples from the buffer, as indicated by $\mathcal{B} \in \mathcal{B}$, where \mathcal{B} is the collection of all possible combinations of experiences. For this study, the batch size, \mathcal{B}_s , is set to 64, thus requiring at least 64 training examples to commence training.

$$\mathcal{B} = [\mathcal{H}(1), \mathcal{H}(2), \dots, \mathcal{H}(64)] \quad (\text{Eq. 4.2})$$

Each training example is a quadruple, comprising s_t , a_t , r_t , and s_{t+1} . In the next step, all current and next states are isolated from the batch.

Subsequently, the Q-values, $Q(s_t, a_t)$, corresponding to each current state, s_t , are calculated. This involves feeding each state, $s_t(i)$, to an Artificial Neural Network (ANN) to predict the Q-values relative to $a_t(i)$, where i denotes the i th training example in the batch. The number of Q-values derived depends on the action space of the agent.

The process is repeated to calculate the Q-values, $Q(s_{t+1}, a)$, for each next state, s_{t+1} . The Q-value yielding the highest value is selected. The number of potential Q-values derived is contingent on the action space of the agent.

The Q-values are updated using the Bellman equation (Eq. 4.3) [bellman1957dynamic]. This equation consists of two components: the immediate reward and the discounted fu-

ture reward. The latter is computed by multiplying the maximum Q-value (obtained in step 4) with a discount factor, γ . The sum of the immediate reward and the discounted future reward provides the expected Q-values to replace the ones calculated in step 3.

$$Q^{gt}(s_t, a_t) = r_t + \gamma \cdot \max_a Q(s_{t+1}, a) \quad (\text{Eq. 4.3})$$

The Q-values derived in step 5 are deemed as the 'ground-truth' Q-values. The expectation is that feeding state s_t into the deep neural network (DNN) should yield these Q-values. This can only be achieved by updating the weights of the DNN. To accomplish this, we need to calculate the loss, which is the mean squared difference between the predicted Q-values and the ground-truth Q-values.

4.4 Balancing Exploration and Exploitation in Training

One of the key challenges in the training phase of an RL agent involves managing the exploration-exploitation trade-off. Initially, the agent lacks knowledge about the best course of action and focuses on exploration over performance, adopting random actions to learn about the state-action space. As the training process continues, the agent becomes more knowledgeable about the state-action pairs and minimizes exploration. Consequently, the agent increasingly adopts exploitation actions to optimize performance.

The agent's propensity to choose a random action is governed by a variable called epsilon (ϵ), also known as the exploration rate. Initially, during the early training stages, the exploration rate is high, leading to more random actions. As training concludes, the exploration rate diminishes, causing the agent to favor exploitative actions. There exist a number of functions for modeling the exploration rate throughout training. For the purpose of this study, we focus on the linear decay model.

In the linear decay model, also known as the ϵ -greedy strategy [45], the exploration rate epsilon (ϵ) starts at 1 (signifying complete exploration) and linearly decays to 0 (signifying complete exploitation) by the end of training. The model is defined by the equation:

$$\epsilon = 1 - \frac{e}{N} \quad (\text{Eq. 4.5})$$

where e is the current episode and N is the total number of episodes.

This model provides a balanced approach to exploration and exploitation, allowing the agent to fully explore the state-action space in the initial stages of training and gradually transition to exploitation as it becomes more knowledgeable. However, it is essential to maintain this balance as an imbalance could hinder the agent’s learning. Too much exploration might lead to suboptimal decisions, while excessive exploitation could cause the agent to miss potentially more optimal solutions.

Figure 4.3 showcases a plot of the linear decay model (Eq. 4.5), demonstrating how the exploration rate decays over the episodes.

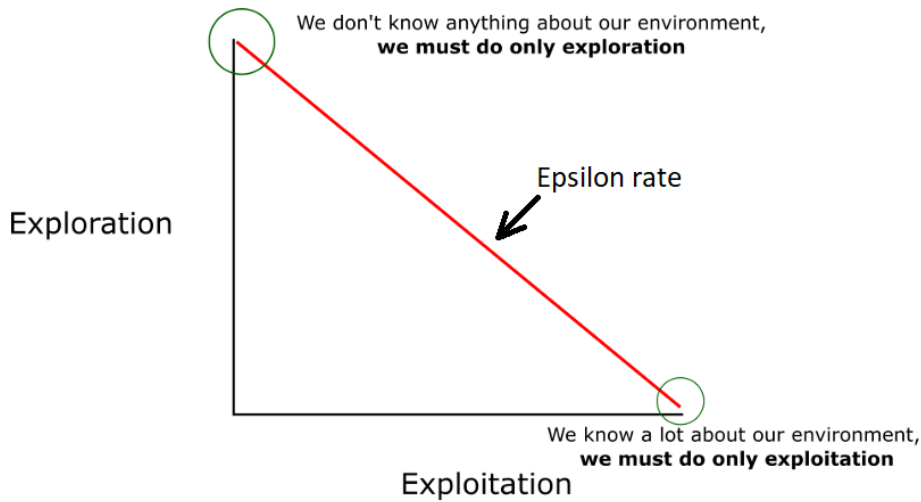


Figure 4.3: Epsilon Decay in Linear Model

Overall, the balance between exploration and exploitation is critical in RL, and the choice of the exploration rate model plays a crucial role in the training and performance of the agent. The ϵ -greedy strategy, in particular, has proven to be effective in many RL applications [2, 45] and is thus employed in this study.

4.5 Traffic Generation in Simulations

The proper simulation of traffic generation has a significant impact on the performance of an agent. In the training phase, it is crucial to generate traffic patterns that resemble

real-world scenarios. For this purpose, various probability distributions can be utilized to simulate different traffic patterns. The most commonly used distributions for this purpose are Uniform, Gaussian, and Weibull, as depicted in Figure 4.4, with the x-axis representing the number of simulation steps and the y-axis indicating the number of vehicles generated in each step.

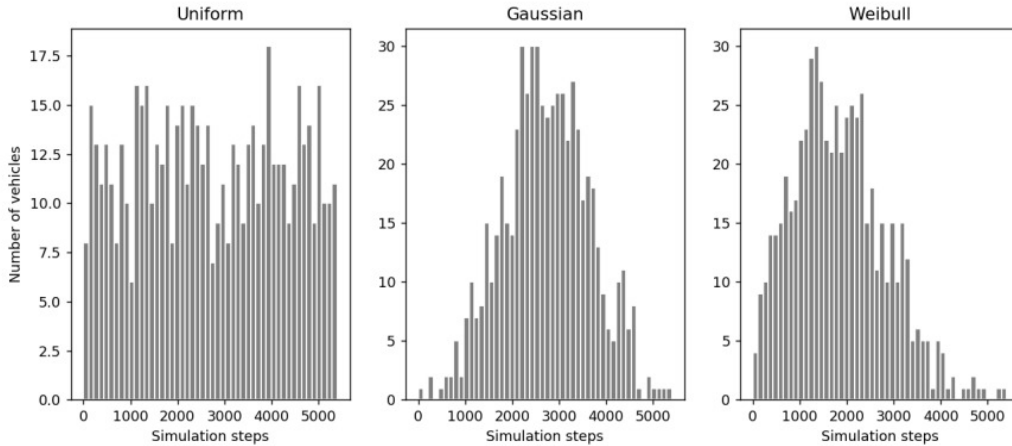


Figure 4.4: Three Different Traffic Generation Distributions

The Uniform distribution is one of the simplest and generates an equal number of vehicles at every simulation timestep. However, this approach fails to accurately capture real-world traffic patterns where vehicle flow varies over time. Similarly, Gaussian distribution falls short of modeling real-world traffic accurately as it increases and decreases gradually, while traffic flow in reality often peaks during certain hours and decreases slowly.

On the other hand, the Weibull distribution closely mimics real-world traffic patterns, demonstrating high traffic flow during peak hours followed by a slow decrease. This behavior aligns with actual traffic patterns, particularly during morning and evening rush hours. Consequently, the Weibull distribution was selected for this study [3].

To optimize the agent’s performance, four different traffic scenarios were created. These scenarios varied the number of cars generated, as summarized in Table 4.2. During the training phase, these traffic scenarios were cycled after each episode.

Table 4.2: Traffic Generation in Different Traffic Scenarios

Traffic Scenario	Traffic Generation
Low	600
High	3000
East-West (EW)	1500
North-South (NS)	1500

The total number of cars generated in each traffic scenario and the timestep at which these cars entered the simulation environment were defined and used to determine the source and destination of the generated cars.

The probability distribution of car generation for each traffic scenario is listed in Table 4.3. For the Low and High traffic scenarios, the car generation probability is the same for all directions.

Table 4.3: Traffic Generation Probability for Four-Way Intersection

Traffic Scenario	East	West	North	South
Low	0.25	0.25	0.25	0.25
High	0.25	0.25	0.25	0.25
East-West (EW)	0.40	0.40	0.10	0.10
North-South (NS)	0.10	0.10	0.40	0.40

In each case, the destination of each car was determined using a pre-defined probability distribution: 60% of cars proceed straight, 20% turn left, and the remaining 20% turn right.

Traffic was simulated in the SUMO environment using a route file that contains all the necessary information about each car’s source, destination, departure time, and route between source and destination. To identify the optimal route, the A* algorithm was used, which is provided by SUMO’s DUAROUTER package [61].

This section provides a comprehensive approach to generating traffic for RL training, aiming to closely mimic real-world scenarios. By doing so, the trained agent can better learn how to handle complex, real-world traffic scenarios, thus making the model more

useful and robust in practical applications.

It's worth mentioning that East-West and North-South traffic scenarios were designed to emulate heavy traffic on main thoroughfares in urban environments. The aim is to train the reinforcement learning agent to avoid developing a preference for one specific direction at intersections.

For the East-West scenario, 80% of the traffic originated from East and West directions, with the remaining 20% coming from North and South. Meanwhile, in the North-South scenario, 80% of the traffic came from North and South, with the remaining 20% from East and West. These specific distributions were intended to accurately represent the varied traffic patterns observed in actual city traffic.

This comprehensive approach to generating traffic for RL training aims to mirror real-world scenarios as closely as possible. The design is meant to enable the agent to effectively learn to navigate through complex traffic patterns, thereby increasing the model's applicability and robustness in real-world situations.

Results and Discussion

In this chapter, the results from different simulation experiments involving reinforcement learning (RL) agents are evaluated and analyzed. These assessments provide a reference point or benchmark to ascertain the performance of the agent. To ensure a comprehensive understanding, seven distinct evaluation metrics are employed.

These metrics are employed to make comparisons between multiple iterations of turn-based and time-based agents. At the end of this chapter, we will reflect on the optimal models from single-agent and multi-agent perspectives, discussing their potential applications in real-world settings. A glossary of terms used throughout this chapter can be found in Table 5-1.

5.1 Evaluating Performance

The performance measure is a yardstick that assesses how effectively the agent interacts within the simulated environment. This measure encapsulates the quality of the agent's performance. Unlike previous studies that have predominantly used single evaluation metrics [russo2020, prashanth2019], this analysis makes use of seven evaluation metrics. This approach acknowledges the complexity of performance in RL and recognises that a single metric may not fully capture the agent's overall performance.

In the evaluation phase, the trained RL agent is simulated over five episodes. During each episode, vehicles are generated using a Weibull distribution with random seeds. Data for the seven evaluation metrics are gathered from the simulation and averaged to obtain mean values.

Table 5.1: Chapter 5 Terminology

Notation	Definition
e	Current episode
$Tnr(e)$	Total negative reward for episode e
ts	Sampling timestep
m	Total number of sampling timesteps in one episode
$r(ts)$	Reward value at timestep ts
$awt(ts)$	Accumulative wait time of all cars at ts
$ar(e)$	Accumulative reward for episode e
c	Individual car
n	Total number of cars in network at ts
rd	Road towards the intersection
$Tawt(e)$	Total accumulative wait time for episode e
$ewpc(e)$	Expected wait time per car for episode e
$aql(e)$	Average queue length for episode e
ql	Queue length of stationary cars at intersection
$p(ts)(c)$	Speed of car c at ts
sgn	Signum mathematical function
RL	Reinforcement Learning

It should be noted that in this research, four different traffic scenarios are utilized. For each traffic scenario, all seven evaluation metrics are calculated and compared, offering a comprehensive assessment of agent performance. These seven evaluation metrics will be discussed in detail in the following sections.

5.2 Evaluation Metrics

5.2.1 Total Negative Reward

In the context of reinforcement learning, rewards can be positive or negative as discussed in section 3.5. A critical assessment of RL agent performance involves aggregating these rewards. The Total Negative Reward (Tnr) refers to the sum of all negative rewards garnered from the reward function during one simulation episode. This metric is

calculated at the conclusion of each episode, and the RL agent’s objective is to maximize it. Equation 5.1 illustrates how Tnr is calculated for a single episode e .

$$Tnr(e) = \sum \min(0, r(ts)) \text{ for } ts \text{ from } 1 \text{ to } m. \quad (5.2.1)$$

5.2.2 Total Accumulated Wait Time

The Wait Time (wt) at a specific sampling timestep ts represents the sum of waiting periods for each car c located on an incoming road rd . Consider the scenario illustrated in Figure 5-2, which represents an individual car’s wait time at various sampling timesteps as it approaches an intersection. The car’s wait time remains zero until it halts at the intersection due to a red light. Notably, the wait time only increases when the car’s speed falls below 1 m/s.

The Accumulated Wait Time (awt) sums the individual wait times of all cars present on the network’s incoming road at a specific sampling timestep ts , as described in Equation 5.2:

$$awt(ts) = \sum [rd(c) \cdot wt(ts)(c)] \quad (5.2.2)$$

where $rd(c)$ is 1 if the car is on the incoming road, and 0 otherwise.

The Total Accumulated Wait Time ($Tawt$) represents the sum of all $awt(ts)$ for a given episode e , as expressed in Equation 5.3:

$$Tawt(e) = \sum awt(ts) \text{ for } ts \text{ from } 1 \text{ to } m. \quad (5.2.3)$$

5.2.3 Expected Wait Time per Car

The Expected Wait Time per Car ($ewpc$) is the average waiting time an individual car c can anticipate when attempting to cross the intersection. This measure is calculated at the conclusion of each episode e , using Equation 5.4:

$$Ewpc(e) = \frac{\sum [rd(c) \cdot wt(ts = m)(c)]}{n} \text{ for } c \text{ from } 1 \text{ to } n. \quad (5.2.4)$$

Here c denotes number of cars. The $Ewpc(e)$ serves as a crucial performance metric for the reinforcement learning agent. Lower values denote superior agent performance, whereas higher values signify poorer performance.

5.2.4 Average Queue Length

Queue Length (ql) signifies the number of cars stationary at the intersection, where cars moving slower than 1 m/s are considered stationary. This metric is calculated at every sampling timestep using Equation 5.5:

$$ql(ts) = \sum rd(c) [1 - \text{floor} \{ \text{sgn} (p(ts)(c) - 1) / 2 + 1 \}] \text{ for } c \text{ from } 1 \text{ to } n. \quad (5.2.5)$$

Here, 'floor' is a mathematical function that rounds down values, while 'sgn' is the signum function. The Average Queue Length (aql) is the mean ql value calculated at the end of each episode e . It is determined for each direction of the intersection using Equation 5.6. Lower $aql(e)$ values denote superior performance, while higher values indicate poorer performance.

$$aql(e) = \frac{\sum ql(ts) \text{ from } ts = 1 \text{ to } m}{m}. \quad (5.2.6)$$

These metrics ($Tawt$, $ewpc$, and aql) serve as crucial tools in assessing the effectiveness of reinforcement learning agents in traffic signal control scenarios. Further studies can be found in the literature that extensively address these issues. For example, "A Deep Reinforcement Learning Network for Traffic Light Cycle Control" (Li, 2019) and "A Survey on Traffic Signal Control Methods" (Yang, 2020) provide deeper insights into these parameters and their applications in real-world traffic management systems.

5.3 Results

5.3.1 Agent 1: Deep Reinforcement Learning with Q Learning (DQN)

The upcoming figures showcase key metrics, namely the total negative reward, total positive reward, total average reward, and total reward across episodes for the DQN agent with experience replay. It is worth noting that initially, the values for these

metrics are relatively high, indicating the agent's initial lack of proficiency. However, as the episodes unfold, a consistent decline in the total negative reward becomes apparent. This downward trend serves as evidence that the RL agent actively learns and refines its behavior over time, leading to improved performance.

Total Negative Reward

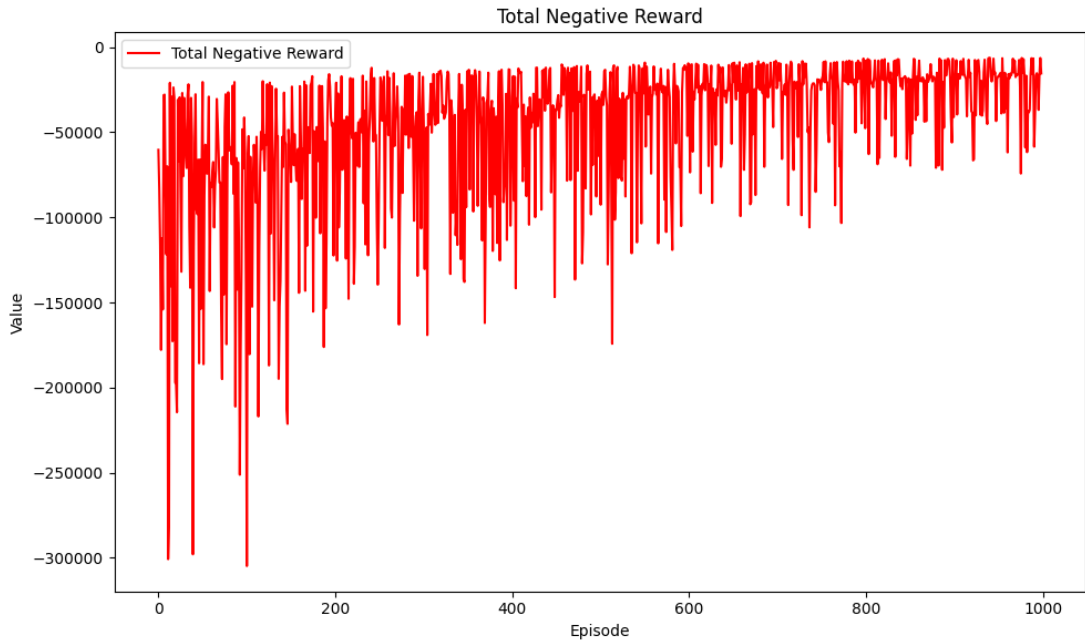


Figure 5.1: Total Negative Reward

Total Average Reward

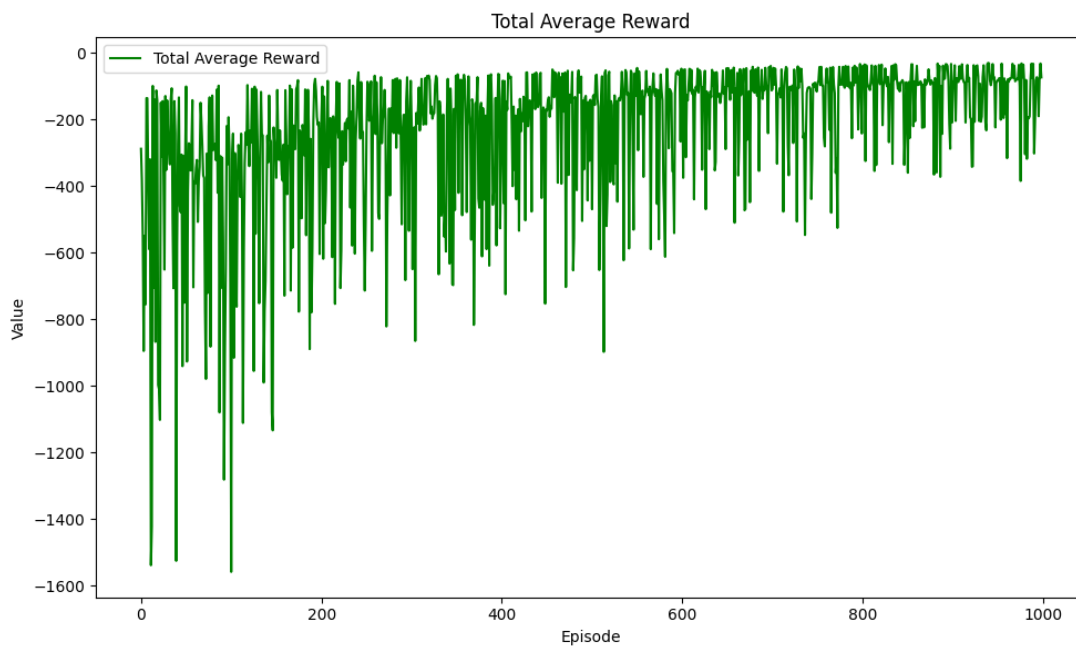


Figure 5.2: Total Average Reward

Total Reward

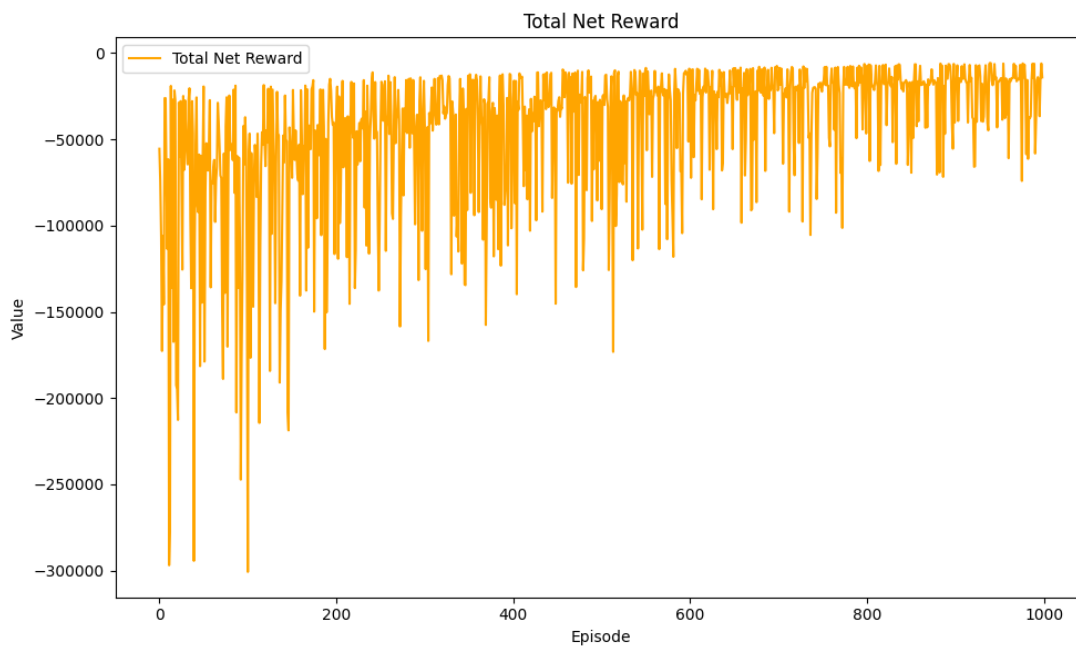


Figure 5.3: Total Reward

The upcoming figures present the trend of vehicle wait times for each cardinal direction at a traffic intersection (North, East, South, West). Initially, the wait times at the intersection were higher, reflecting the inefficiency of the RL agents. However, as the RL agents continue to learn throughout each episode, the wait times consistently decrease. This reduction in wait times indicates that the RL agents are gradually improving their decision-making abilities and optimizing traffic flow at the intersection.

Wait time in Cardinal Direction of North

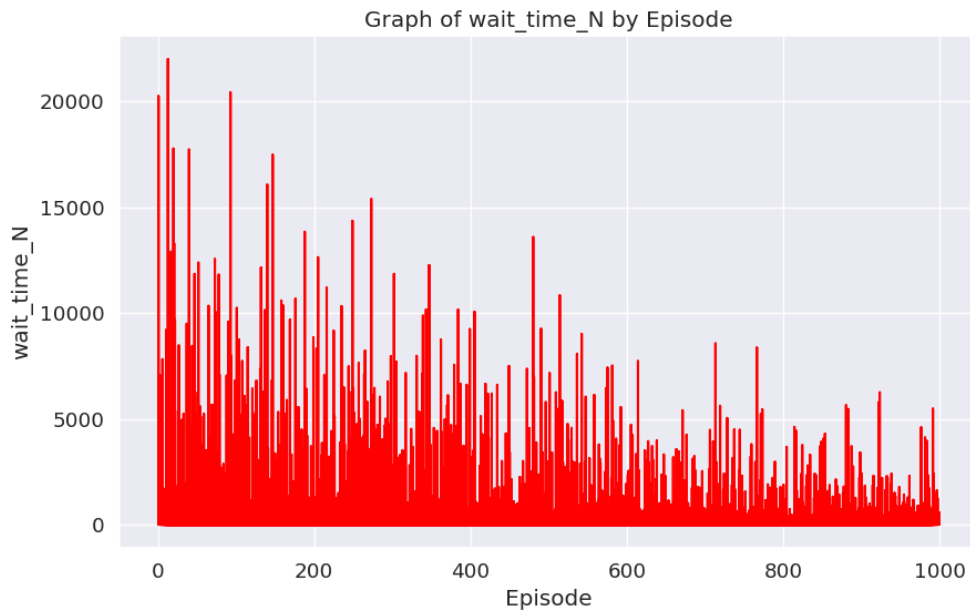


Figure 5.4: Wait time in Cardinal Direction of North

Wait time in Cardinal Direction of East

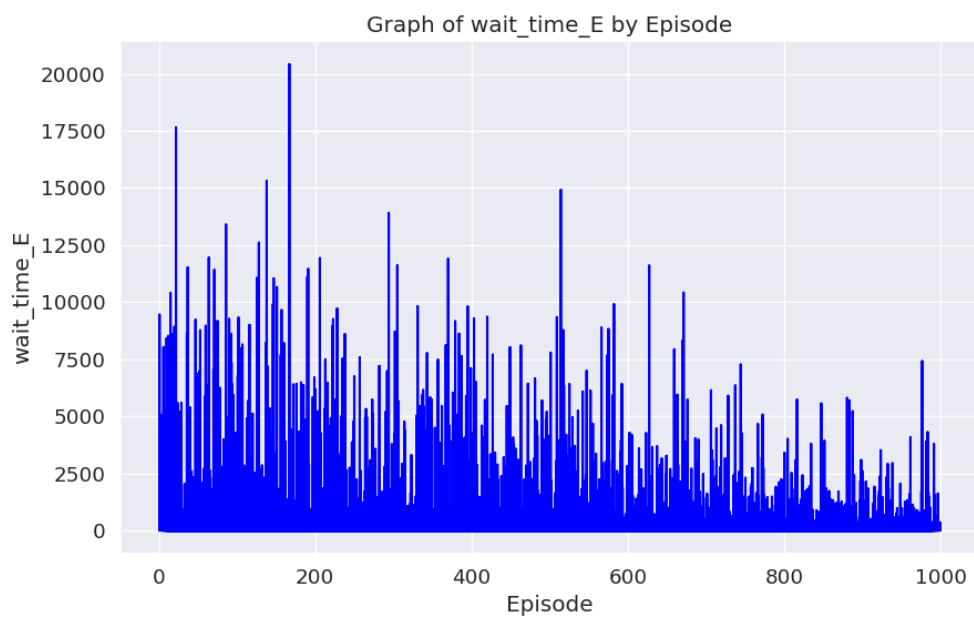


Figure 5.5: Wait time in Cardinal Direction of East

Wait time in Cardinal Direction of West

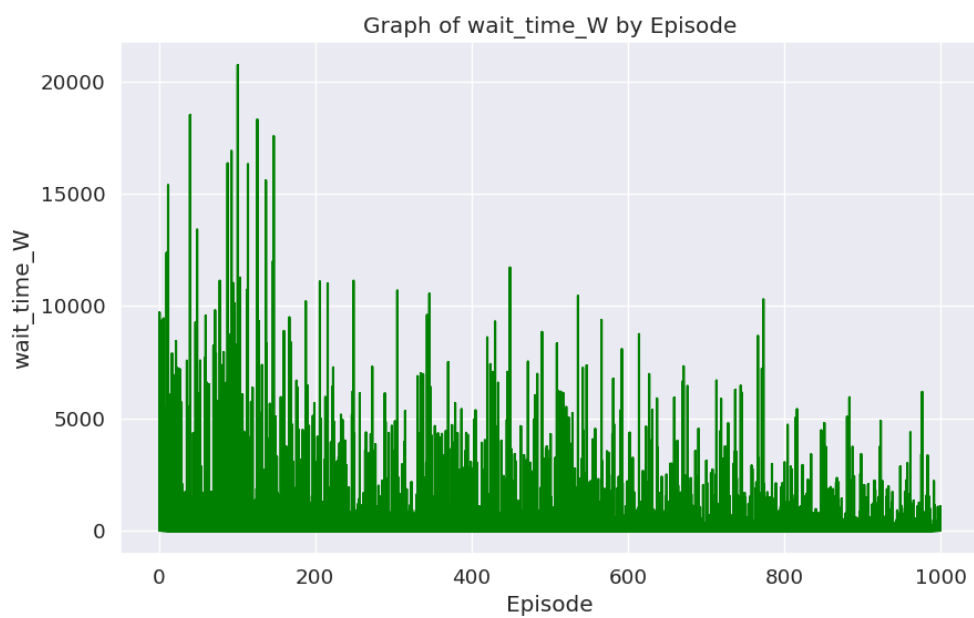


Figure 5.6: Wait time in Cardinal Direction of West

Wait time in Cardinal Direction of South

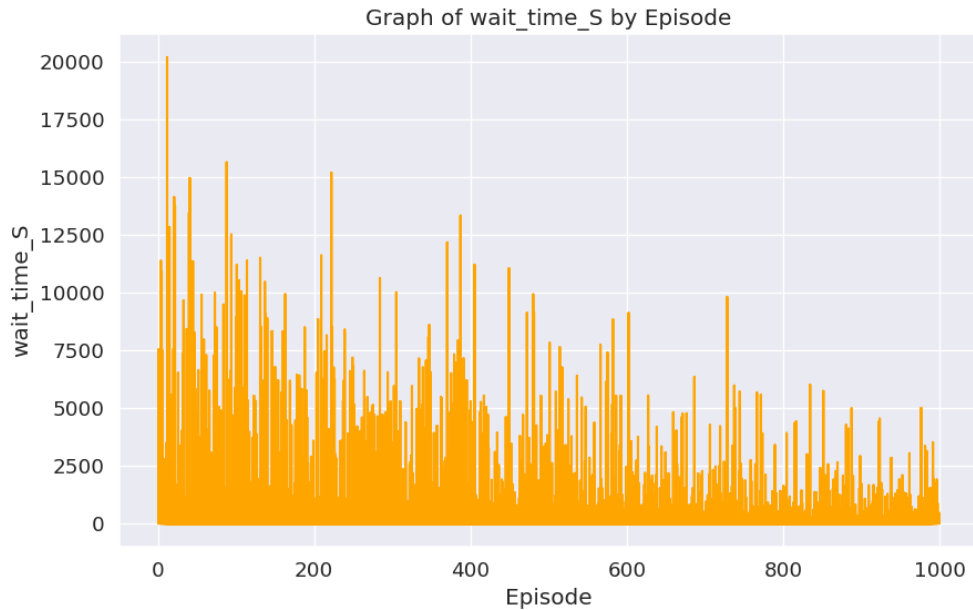


Figure 5.7: Wait time in Cardinal Direction of South

5.3.2 Agent 2: Deep Reinforcement Learning with Double Q Learning (DDQN)

The subsequent figures show performance indicators such as total negative reward, total positive reward, total average reward, and the comprehensive reward over multiple episodes for the DDQN agent coupled with experience replay. It's crucial to highlight that the initial episodes register high metrics, reflecting the agent's preliminary stage of learning and consequent inefficient performance. Nevertheless, as the agent progresses through the episodes, a consistent decrease in the total negative reward is observed. This continuous improvement indicates the RL agent's adaptive learning capability, demonstrating its gradual refinement of strategies leading to superior performance.

Total Negative Reward

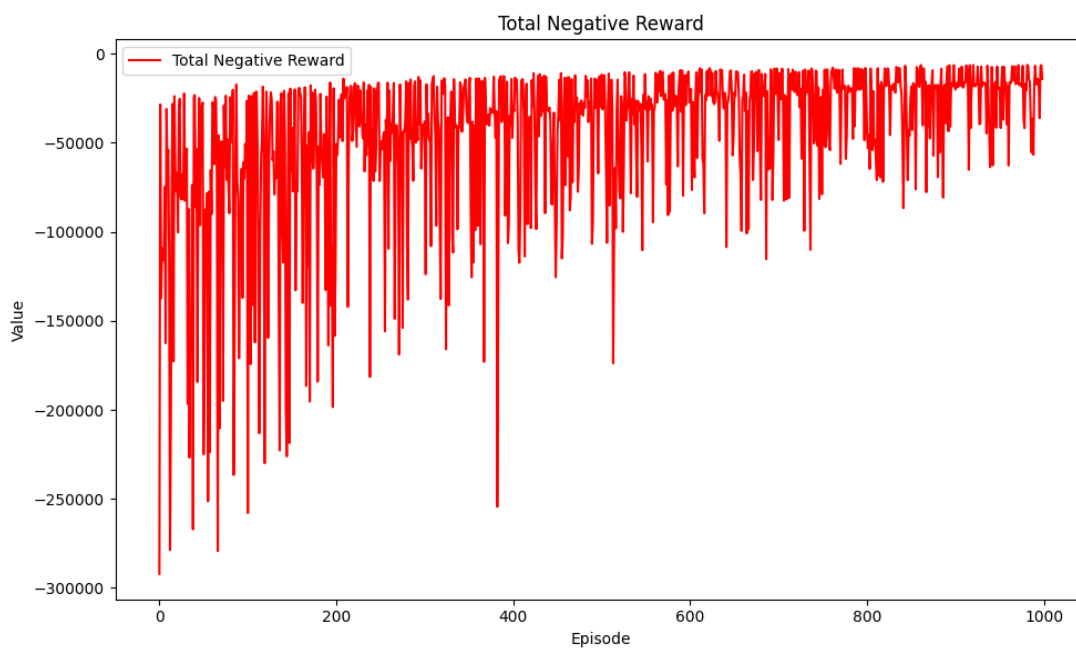


Figure 5.8: Total Negative Reward

Total Average Reward

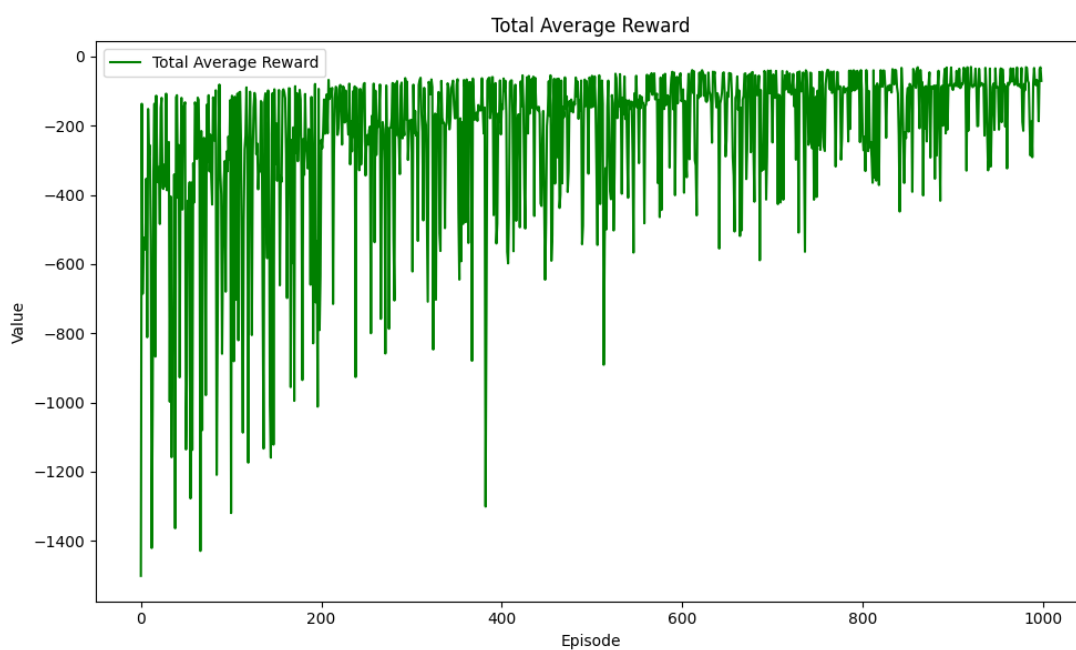


Figure 5.9: Total Average Reward

Total Reward

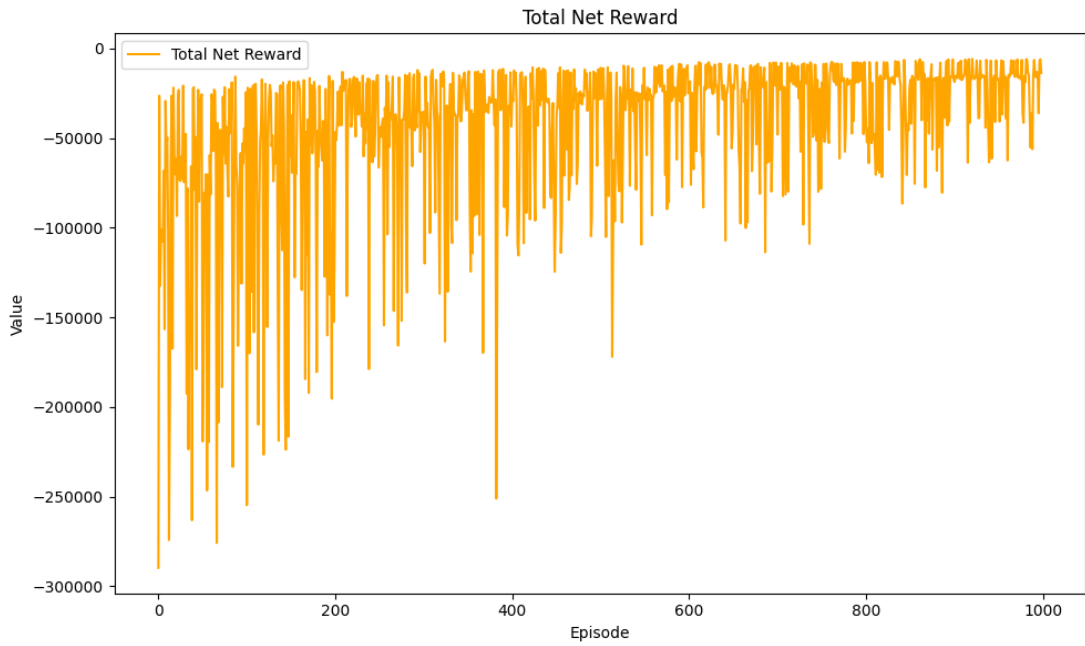


Figure 5.10: Total Reward

The forthcoming illustrations depict the trajectory of vehicle wait times at a traffic intersection, covering each cardinal direction (North, East, South, West). Early episodes showcase elevated wait times, demonstrating the RL agents' initial inefficiencies. As episodes progress, however, these times consistently reduce, highlighting the RL agents' continuous learning and enhanced decision-making capabilities, leading to traffic flow optimization at the intersection.

Wait time in Cardinal Direction of North

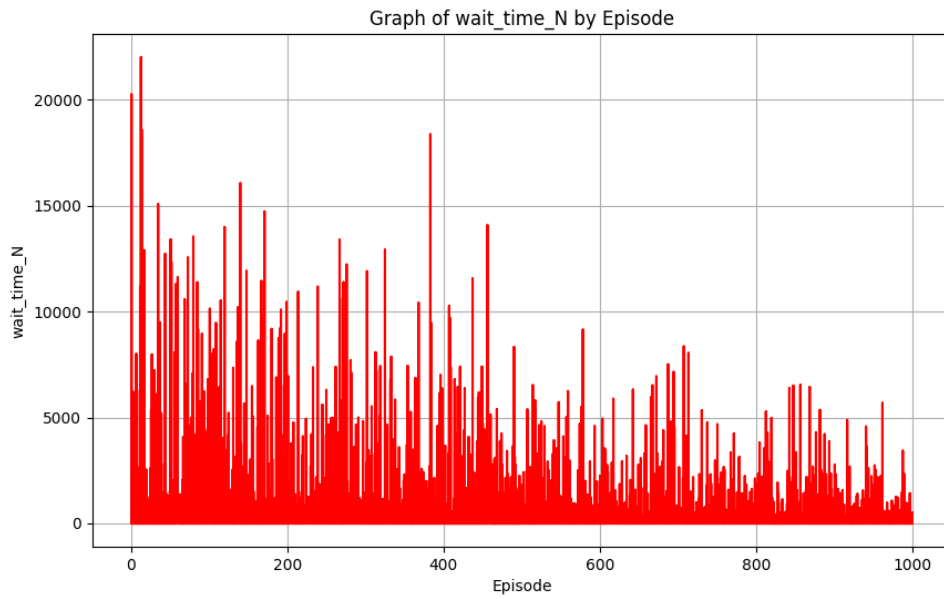


Figure 5.11: Wait time in Cardinal Direction of North

Wait time in Cardinal Direction of East

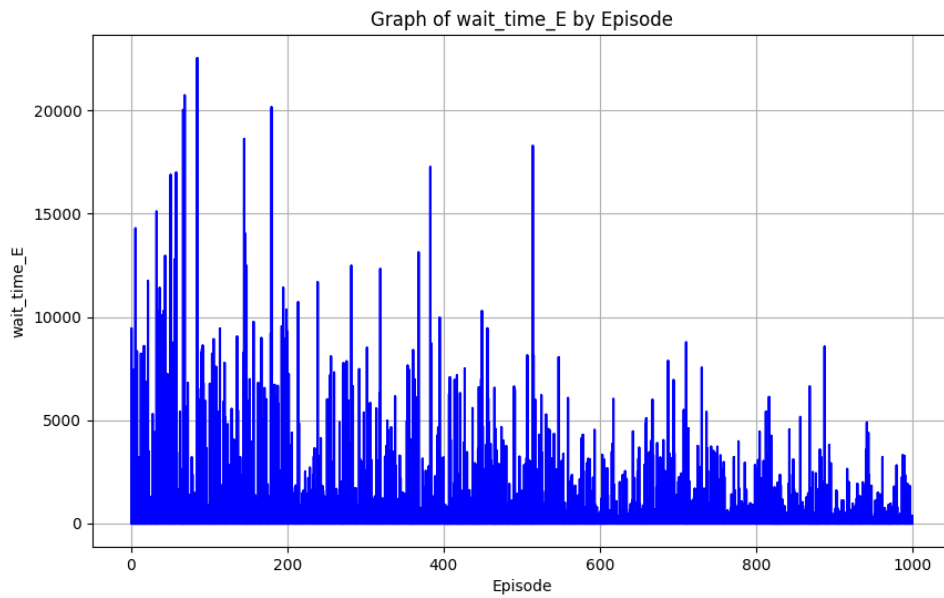


Figure 5.12: Wait time in Cardinal Direction of East

Wait time in Cardinal Direction of West

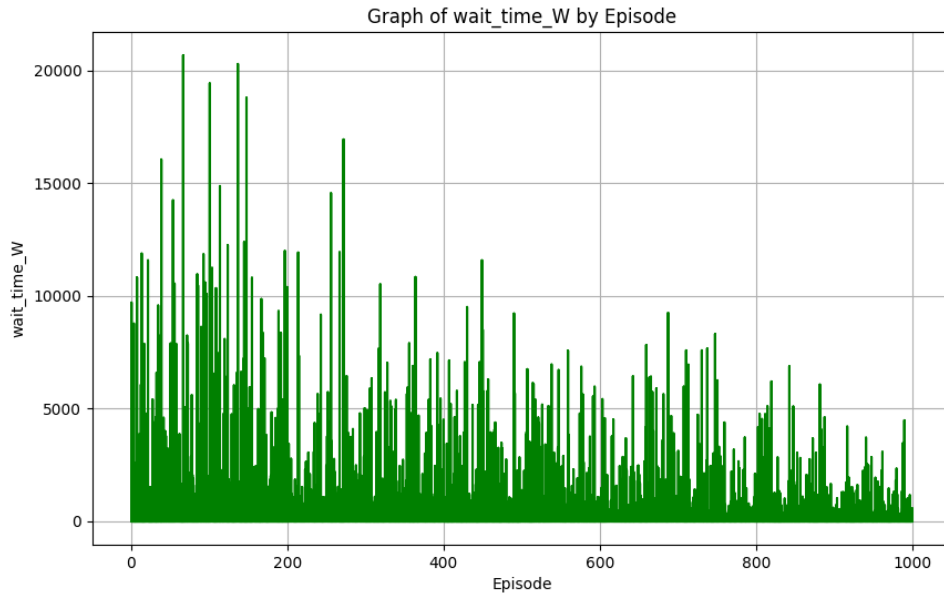


Figure 5.13: Wait time in Cardinal Direction of West

Wait time in Cardinal Direction of South

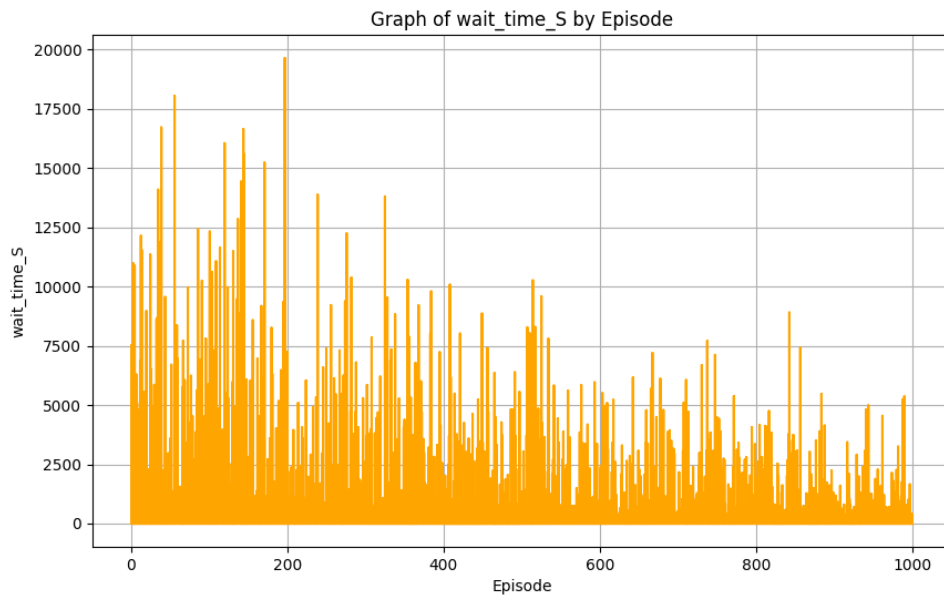


Figure 5.14: Wait time in Cardinal Direction of South

5.4 Conclusion

In summary, the study demonstrates that both of our algorithms – the Deep Q-Learning (DQN) and the Double Deep Q-Learning (DDQN) – exhibit convergence when run for 1000 episodes, indicating a promising trend of continual learning and performance improvement.

This performance trend underscores the significance of reinforcement learning (RL) in optimizing traffic light control systems. The RL agents started with high wait times, signifying an initial lack of proficiency. However, as the agents proceeded through the episodes, they progressively reduced the wait times, exhibiting learning and decision-making enhancement.

The encoding method used to represent the state space of the intersection had a pivotal role in the agent’s performance. It allowed the agents to better understand the traffic scenario and make informed decisions. Furthermore, the reward function, designed to minimize the wait times, provided the necessary feedback for the agents to improve their performance.

The use of Deep Neural Networks (DNN) as function approximators also played a crucial role. They helped to cope with the large state space encountered in traffic scenarios by learning the association between state-action pairs effectively.

Additionally, the training process, which involved experience replay memory and the Bellman equation, allowed the RL agents to effectively learn from their past experiences, which further enhanced their decision-making capabilities.

It should be noted, however, that the learning curve could potentially be further saturated by increasing the number of episodes and the number of games in each episode. Both of these are hyperparameters that can be fine-tuned as per specific requirements. Fine-tuning these parameters could potentially lead to even better performance and more robust learning by the RL agents.

In conclusion, this research presents a compelling case for the application of RL techniques in traffic light control systems. It suggests that RL, coupled with an effective state representation method, a suitable reward function, and a robust training process, can substantially optimize the traffic flow and reduce the wait times at traffic intersections. Future studies could explore different RL algorithms, state representation methods, and

reward functions to continue advancing in this area.

5.5 Future Work

Traffic control at multiple intersections is an intricate problem that is increasingly being tackled using multi-agent deep reinforcement learning. Managing traffic flow across multiple intersections is a complex problem being progressively tackled through multi-agent deep reinforcement learning. Approaches such as Independent Q-Learning, Fully Observable Critic, Value Function Factorization, Consensus algorithms, and learning to communicate each offer their own unique advantages and drawbacks. However, these methods have significantly advanced the field, facilitating cooperative interaction between agents and enabling coordinated control over traffic networks. Despite these advancements, challenges such as optimality versus scalability, non-stationarity of the environment, and the problem of credit assignment persist.

Future research directions should focus on addressing these challenges and further improving the performance of multi-agent RL in managing traffic. The incorporation of other AI techniques, such as attention mechanisms, could help in selecting relevant information and dealing with the curse of dimensionality. Hierarchical reinforcement learning and the use of graph theory also hold potential. Finally, the development of novel communication protocols among agents, including the efficient exchange of policy gradients or weights, could lead to more sophisticated and effective traffic control solutions. As we continue to make progress in these areas, the potential for RL to revolutionize traffic control systems becomes increasingly tangible.

Bibliography

- [1] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine Learning* 8 (1992), pp. 279–292.
- [2] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine Learning* 8 (1992), pp. 279–292.
- [3] Arthur J. Hallinan Jr. “A review of the Weibull distribution”. In: *Journal of Quality Technology* 25.2 (1993), pp. 85–93.
- [4] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. “Reinforcement learning: A survey”. In: *Journal of Artificial Intelligence Research* 4.0 (1996), pp. 237–285.
- [5] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. “Reinforcement learning: A survey”. In: *Journal of Artificial Intelligence Research* 4 (1996), pp. 237–285.
- [6] Abolhassan Halati, Henry Lieu, and Susan Walker. “CORSIM-corridor traffic simulation model”. In: *Traffic Congestion and Traffic Safety in the 21st Century: Challenges, Innovations, and Opportunities*. Urban Transportation Division, ASCE; Highway Division, ASCE; Federal Highway Administration, USDOT; and National Highway Traffic Safety Administration, USDOT. 1997.
- [7] Stuart Russell. “Learning agents for uncertain environments”. In: *Proceedings of the eleventh annual conference on Computational learning theory*. 1998.
- [8] Anthony J. McMichael. “The urban environment and health in a world of increasing globalization: issues for developing countries”. In: *Bulletin of the World Health Organization* 78.0 (2000), pp. 1117–1126.
- [9] M. Behrisch et al. “SUMO – Simulation of Urban MObility”. In: *Proceedings of the 4th Middle East Symposium on Simulation and Modelling*. 2002, pp. 183–187.

- [10] Baher Abdulhai, Rob Pringle, and Grigoris J. Karakoulas. “Reinforcement learning for true adaptive traffic signal control”. In: *Journal of Transportation Engineering* 129.3 (2003), pp. 278–285.
- [11] Z. Q. Shen, Robert Gay, and Xuehong Tao. “Goal-based intelligent agents”. In: *International Journal of Information Technology* 9.1 (2003), pp. 19–30.
- [12] Wenji Mao and Jonathan Gratch. “A utility-based approach to intention recognition”. In: *AAMAS 2004 Workshop on Agent Tracking: Modeling Other Agents from Observations*. Vol. 46. 2004.
- [13] Marco Wiering and et al. “Simulation and optimization of traffic in a city”. In: *IEEE Intelligent Vehicles Symposium*. IEEE. 2004.
- [14] J. Casas et al. *Traffic Simulation with Aimsun*. Springer, 2006.
- [15] Jordi Casas and et al. “Traffic simulation with Aimsun”. In: *Fundamentals of Traffic Simulation*. 2010, pp. 173–232.
- [16] Martin Fellendorf and Peter Vortisch. *Fundamentals of Traffic Simulation*. Springer, 2010.
- [17] Michael Behrisch and et al. “SUMO—simulation of urban mobility: an overview”. In: *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind. 2011.
- [18] Gustavo AL de Campos, Emmanuel SS Freire, and Mariela I. Cortés. “Norm-based behavior modification in reflex agents”. In: *Proceedings on the International Conference on Artificial Intelligence (ICAI)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). 2012.
- [19] Frédéric Garcia and Emmanuel Rachelson. “Markov decision processes”. In: *Markov Decision Processes in Artificial Intelligence*. 2013, pp. 1–38.
- [20] Volodymyr Mnih and et al. “Playing Atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).
- [21] M. El-Tantawy, B. Abdulhai, and H. Abdelgawad. “Design of reinforcement learning parameters for seamless application in operational traffic signal control”. In: *Journal of Intelligent Transportation Systems* 18.2 (2014), pp. 227–245.

BIBLIOGRAPHY

- [22] Samah El-Tantawy, Baher Abdulhai, and Hossam Abdelgawad. “Design of reinforcement learning parameters for seamless application of adaptive traffic signal control”. In: *Journal of Intelligent Transportation Systems* 18.3 (2014), pp. 227–245.
- [23] Yu Li and et al. “Study of the time-collocation of signal lamp at intersection”. In: *Mathematical modelling of engineering problems* 2.2 (2015), pp. 5–10.
- [24] Timothy P. Lillicrap and et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).
- [25] Volodymyr Mnih and et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533.
- [26] Volodymyr Mnih and et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533.
- [27] Wade Genders and Saiedeh Razavi. “Using a deep reinforcement learning agent for traffic signal control”. In: *arXiv preprint arXiv:1611.01142* (2016).
- [28] Matthew Hausknecht and Peter Stone. “On-policy vs. off-policy updates for deep reinforcement learning”. In: *Deep Reinforcement Learning: Frontiers and Challenges, IJCAI 2016 Workshop*. AAAI Press. New York, NY, USA, 2016.
- [29] Volodymyr Mnih and et al. “Asynchronous methods for deep reinforcement learning”. In: *International conference on machine learning*. PMLR. 2016.
- [30] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. Malaysia: Pearson Education Limited, 2016, pp. 67–77.
- [31] D. Silver and et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), pp. 484–489.
- [32] Elise Van Der Pol. “Deep reinforcement learning for coordination in traffic light control”. MA thesis. University of Amsterdam, 2016.
- [33] Hado Van Hasselt, Arthur Guez, and David Silver. “Deep reinforcement learning with double q-learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.
- [34] Ziyu Wang and et al. “Sample efficient actor-critic with experience replay”. In: *arXiv preprint arXiv:1611.01224* (2016).

BIBLIOGRAPHY

- [35] Haofeng Yu. “From Deep Blue to DeepMind: What AlphaGo Tells Us”. In: *Predictive Analytics and Futurism* 13 (2016), pp. 42–45.
- [36] Juntao Gao and et al. “Adaptive traffic signal control: Deep reinforcement learning algorithm with experience replay and target network”. In: *arXiv preprint arXiv:1705.02755* (2017).
- [37] Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. “Traffic light control using deep policy-gradient and value-function-based reinforcement learning”. In: *IET Intelligent Transport Systems* 11.7 (2017), pp. 417–423.
- [38] Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. “Traffic light control using deep policy-gradient and value-function-based reinforcement learning”. In: *IET Intelligent Transport Systems* 11.7 (2017), pp. 417–423.
- [39] Ofir Nachum and et al. “Bridging the gap between value and policy based reinforcement learning”. In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.
- [40] A. A. Rusu and et al. “Sim-to-real robot learning from pixels with progressive nets”. In: *Conference on Robot Learning*. 2017, pp. 262–270.
- [41] Tuomas Haarnoja and et al. “Soft actor-critic algorithms and applications”. In: *arXiv preprint arXiv:1812.05905* (2018).
- [42] Yilun Lin and et al. “An efficient deep reinforcement learning model for urban traffic control”. In: *arXiv preprint arXiv:1808.01876* (2018).
- [43] Soheil Mohamad Alizadeh Shabestary and Baher Abdulhai. “Deep learning vs. discrete reinforcement learning for adaptive traffic signal control”. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2018.
- [44] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT Press, 2018.
- [45] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.
- [46] Vinh An Vu. “Real-Time Traffic Management Strategy Generation: Framework, Application and Performance”. PhD thesis. Singapore: National University of Singapore, 2018.

- [47] Arthur Guez and et al. “An investigation of model-free planning”. In: *International Conference on Machine Learning*. PMLR, 2019.
- [48] Mengyu Guo and et al. “A reinforcement learning approach for intelligent traffic signal control at urban intersections”. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2019.
- [49] Maximilian Igl and et al. “Generalization in reinforcement learning with selective noise injection and information bottleneck”. In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019.
- [50] Lukasz Kaiser and et al. “Model-based reinforcement learning for atari”. In: *arXiv preprint arXiv:1903.00374* (2019).
- [51] Kimin Lee and et al. “Network randomization: A simple technique for generalization in deep reinforcement learning”. In: *arXiv preprint arXiv:1910.05396* (2019).
- [52] Xiaoyuan Liang and et al. “A deep reinforcement learning network for traffic light cycle control”. In: *IEEE Transactions on Vehicular Technology* 68.2 (2019), pp. 1243–1253.
- [53] Matthew Muresan, Liping Fu, and Guangyuan Pan. “Adaptive traffic signal control with deep reinforcement learning: an exploratory investigation”. In: *arXiv preprint arXiv:1901.00960* (2019).
- [54] David Sierra Gonzalez. “Towards human-like prediction and decision-making for automated vehicles in highway scenarios”. PhD thesis. Université Grenoble Alpes (ComUE), 2019.
- [55] Zheng Tang and et al. “Cityflow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019.
- [56] Jianqing Fan and et al. “A theoretical analysis of deep Q-learning”. In: *Learning for Dynamics and Control*. PMLR, 2020.
- [57] Ammar Haydari and Yasin Yilmaz. “Deep reinforcement learning for intelligent transportation systems: A survey”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.1 (2020), pp. 11–32.

- [58] Anssi Kanervisto, Christian Scheller, and Ville Hautamäki. “Action space shaping in deep reinforcement learning”. In: *2020 IEEE Conference on Games (CoG)*. IEEE. 2020.
- [59] Harshal Maske, Tianshu Chu, and Uroš Kalabić. “Control of traffic light timing using decentralized deep reinforcement learning”. In: *IFAC-PapersOnLine* 53.2 (2020), pp. 14936–14941.
- [60] Palwasha W. Shaikh and et al. “A review on swarm intelligence and evolutionary algorithms for solving the traffic signal control problem”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.1 (2020), pp. 48–63.
- [61] Luis Urquiza-Aguiar and et al. “Comparison of SUMO’s vehicular demand generators in vehicular communications via graph-theory metrics”. In: *Ad Hoc Networks* 106 (2020), p. 102217.
- [62] Xinshi Zang and et al. “Metalight: Value-based meta-reinforcement learning for traffic signal control”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 01. 2020.
- [63] Kudrat Kutlimuratov and et al. “Modelling traffic flow emissions at signalized intersection with PTV vissim”. In: *E3S Web of Conferences*. Vol. 264. EDP Sciences. 2021.
- [64] Shen Li and et al. “Planning and decision-making for connected autonomous vehicles at road intersections: A review”. In: *Chinese Journal of Mechanical Engineering* 34.1 (2021), pp. 1–18.
- [65] Anum Mushtaq and et al. “Traffic flow management of autonomous vehicles using deep reinforcement learning and smart rerouting”. In: *IEEE Access* 9 (2021), pp. 51005–51019.
- [66] Sebastian Schmid, Daniel Schraudner, and Andreas Harth. “Performance Comparison of Simple Reflex Agents Using Stigmergy with Model-Based Agents in Self-Organizing Transportation”. In: *2021 IEEE International Conference on Autonomous Computing and Self-Organizing Systems Companion (ACSOS-C)*. IEEE. 2021.
- [67] Yang Xiao and et al. “Leveraging deep reinforcement learning for traffic engineering: A survey”. In: *IEEE Communications Surveys & Tutorials* 23.4 (2021), pp. 2064–2097.

BIBLIOGRAPHY

- [68] Mobin Yazdani, Majid Sarvi, and Saeed Asadi Bagloee. “A Comparison Study of Different Data Resolutions for Deep Reinforcement Learning Based Adaptive Traffic Signal Control System”. In: (2021).
- [69] Wenwei Yue and et al. “What is the root cause of congestion in urban traffic networks: Road infrastructure or signal control?” In: *IEEE Transactions on Intelligent Transportation Systems* 23.7 (2021), pp. 8662–8679.

draft 2

Kaplan
Dr. Karam Dad Kallu
Assistant Professor of
Research and Design in Machine Engineering
School of Mechanical and
Manufacturing Engineering
(SMME) RUST H-12 Islamabad

ORIGINALITY REPORT

10%

SIMILARITY INDEX

6%

INTERNET SOURCES

6%

PUBLICATIONS

5%

STUDENT PAPERS

PRIMARY SOURCES

1

Submitted to Higher Education Commission
Pakistan

Student Paper

2%

2

"Deep Learning for Unmanned Systems",
Springer Science and Business Media LLC,
2021

Publication

1%

3

www.idr.iitkgp.ac.in

Internet Source

<1%

4

Submitted to University of Bedfordshire

Student Paper

<1%

5

Submitted to University of Johannesburg

Student Paper

<1%

6

Submitted to Queen Mary and Westfield
College

Student Paper

<1%

7

"Neural Information Processing", Springer
Science and Business Media LLC, 2020

Publication

<1%

thesis.unipd.it