Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Reinforcement Learning for Traffic Light Optimization

Sriom Chakrabarti

April 30, 2022

A dissertation submitted in partial fulfilment
of the requirements for the degree of
MAI (Computer Engineering)

Declaration

I hereby declare that this dissertation is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: Sriom Chakrabarti Date: 30/04/2022

Acknowledgements

To my supervisor, Prof. Ivana Dusparic, for her continuous guidance and support. To my family and friends for all the support, for believing in me and being there for me.

Sriom Chakrabarti

Trinity College Dublin April 2022

Abstract

70% of the world's population is expected to live in cities by the year 2050. In addition, to the number of people, there will be a further strain on the city infrastructure and the transportation networks. With increasing vehicles, traffic congestion is becoming a severe problem. Dublin is the 6th most congested city in Europe and the 17th most congested city globally, with an average commuter spending almost 250 hours stuck in cars travelling at less than 10 km per hour. According to research conducted by the Department of Transportation and Sport's Economic and Finance, the estimate of the cost of time lost due to traffic congestion is €358 million in the year 2012, and it is forecasted to rise to €2.08 billion per year in 2033. With the increase in traffic congestion, there will also be an increase in air pollution. According to the latest estimates from the European Environment Agency (EEA), there is an excess of 1300 premature deaths that occur in Ireland each year because of poor air quality. Transportation and congestion need severe improvements and the need for a National Clean Air Strategy supported by WHO standards are more pressing than ever.

One of the most critical considerations when designing an intelligent traffic management system is developing a smart traffic management system which is flexible and can change with the traffic flow. The main goal of an intelligent traffic management system is to reduce traffic congestion. In recent years many Reinforcement Learning techniques have been implemented in order to solve to improve the traffic light control system because of its ability to understand and learn from different complex situations.

This paper discussed two general and three reinforcement learning algorithms and compared them to solve single-agent and multi-agent traffic simulation cases. We take the average accumulated waiting time for each intersection, plot it with the number of steps for different agents, and compare them to find the best solution. We find out that Proximal Policy Optimisation (PPO) gives the best results in each cases followed by Actor Critic Method (A2C), fixed agent, random agent and Deep-Q Networks.

Contents

List of Figures

List of Tables

1 Introduction

With the exponential expansion in the number of vehicle, traffic control in urban areas is becoming increasingly complex. Development of the road network to meet the increased vehicle count is not a socially viable option; instead, proper traffic flow regulation is required to maximize the use of existing infrastructure. Traffic signals were introduced to improve the safety of road users by controlling the flow of traffic. On the other hand, traffic signals create a bottleneck for traffic flow in lanes that do not have the right of way during a given phase, necessitating signal timing optimization to reduce the overall delay experienced by all cars at the intersection.

We will first address what reinforcement learning is, its different types, components and all the related work in this field. We will then discuss Q-Learning, different deep learning and neural network algorithms used in traffic lights. We will then discuss Deep Q-Networks, Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO) algorithms in detail, which we have addressed in this thesis. We will then go through the design section where we have discussed traffic light control problems, different traffic simulations, how SUMO is used, and the traffic light control problems using state representation, action space, and reward function. Then we have discussed a single agent case in a single intersection, a multi-agent case in a double intersection and a more realistic traffic scenario and used different techniques to compare them.

1.1 Background and Motivation

Traffic management is one of the most important topics that need to be addressed in the 21st century. Traffic congestion is a major concern in most parts of the world. Efficient traffic management, traffic monitoring and controlling system are one of the most trending topics for research. (1). Congestion also causes difficulty for emergency vehicles to pass as well as it increases violations, increases red light jumps, and vehicle breakdowns, and it may also lead to accidents which can cause loss of life or property.(2)
Traffic congestion can lead to serious mental and physical problems as well as it is one of the

leading causes of air pollution.

This paper focuses on reducing traffic congestion by traffic light optimization with the help of state-of-the-art Reinforcement Learning techniques and comparing them to give detailed results.

1.2 Related Work

In the last few years, a large amount of work has been proposed around effective traffic light management systems in order to create smart cities. These can be classified into three main groups such as:

1. **Pre-timed signal control** - This is considered the most fundamental type of signal control. In this system, a fixed time is determined for both the cycle length and split phase, which is calculated based on historical traffic demands without considering potential fluctuations in real traffic demand. The duration of each phase is kept constant. (3)
2. **Vehicle-actuated signal control** - To determine traffic signal changes, traffic demand information is used. There is an inductive loop detector which is placed at an intersection which decides the signal control. Gaps that are generated by the traffic flow are calculated by whether the green period should be extended or not. The green time of the signal can vary according to the time of the day. (4)
3. **Adaptive signal control** - This uses real-time sensor traffic data to predict future traffic conditions, and the signal timing control is managed and updated automatically according to the state of the intersection. E.g. The queue, the length of vehicles in each lane and traffic flow fluctuation. (5)

This study focuses on the third approach, intending to propose novel traffic signal control approaches that take advantage of current breakthroughs in machine learning and artificial intelligence.

1.3 Thesis Aims and Objective

The primary goal of this paper is to explore different Reinforcement Learning algorithms that can be used for traffic light optimization and compare those algorithms on the basis of the average accumulated waiting time. We are comparing different algorithms so that we can

have an insight into how they perform under different traffic conditions.

We will also test the algorithms in different single-agent and multi-agent traffic light scenarios with varying traffic flow to imitate real-world scenarios. We will also look at how agents communicate and perform in these various cases.

1.4 Problem Statement

Cities' populations are growing at an alarming rate due to urbanization. Most people move to larger cities in search of better jobs, better opportunities, and a better way of life. However, as opportunities and population in cities grow, so does the number of vehicles. Today, traffic congestion is at an all-time high. Congestion is becoming a significant issue as the number of vehicles grows, the current traffic system's limits are being tested. The most serious flaw is the static nature of the traffic lights, which causes lanes with higher traffic density to receive the same green light as lanes with lower traffic density.

We live in the twenty-first century when everything is constantly changing, and new technologies rapidly replace the old, but we still use the same old traffic management system. In terms of traffic light optimization, there hasn't been much progress. In this paper, we used Reinforcement Learning to approximate policies defining states to simulate the scenario in the virtual environment so that our agent can function appropriately in the real world.

We want an agent to perceive real-world traffic scenarios and use traffic simulation tools for Simulation of Urban MObility (SUMO), which provides an artificial, but realistic environment where potential traffic management actions could be carried out. We demonstrate and compare various Reinforcement Learning methods capable of optimizing traffic light situations through their use. We show, using simulations, that our practices can reduce travel time during high traffic periods on small road networks with sufficiently robust traffic conditions.

1.5 Thesis Assumptions

The agent makes several assumptions about its behaviour while designing different SUMO environments and evaluating different Reinforcement Learning algorithms. We only consider a single agent case with a single intersection and a multiagent environment with two junctions, where each agent controls its own junction when designing the environments.

However, a real-life scenario is very different from our assumptions, and there are multiple lanes with various intersections with pedestrians making it more complicated. Our model cannot solve these real-world scenarios. For that, we need a combination of various agents to work simultaneously with each other to produce a good result.

In this paper, we assumed that the agent is stationary, that their locations are fixed, and that they do not move through the environment. The agents are attached to the static traffic lights. Agents are also presumed to be error-free. As a result, we do not consider scenarios in which they are given inaccurate or incomplete information.

1.6 Thesis Contribution

The main contribution of the thesis is the design, implementation and the evaluation of different reinforcement learning algorithms in different SUMO environments. It also compares different algorithms on the basis of the average accumulated waiting time with the number of steps and gives a detailed result.

1.7 Document Structure

In this paper, Chapter 2 gives the background material about Reinforcement Learning techniques proposed in different papers. It talks about reinforcement learning its types and components. Then we go through Q-Learning, Deep Learning (Neural Network, Backpropagation, Convolutional Neural Network and Optimization Algorithm). We also discussed Deep Q Networks, Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO). In chapter 3, we design different traffic light control problems, traffic simulators, SUMO and go through traffic light control problems such as state representation, action space, and reward function. Chapter 4 discusses single-agent traffic simulation cases using random, fixed, Deep Q-Networks, Actor-Critic Method (A2C) and Proximal Policy Optimisation (PPO) algorithms. Chapter 5 compares single agents in a single intersection using fixed, random, A2C and PPO agents. Multi-agents use fixed, random, A2C and PPO techniques and single-agents use the more realistic scenario using fixed, random, A2C and PPO algorithms.

2 Background and Related Work

To address the issue of intelligent traffic management systems, a lot of work has been proposed, and they can mainly be classified into three groups:

1. **Pre-timed signal control** - It is a system in which a fixed time for all green phases is determined based on the historical traffic demands without considering possible fluctuations in actual traffic demands.
2. **Vehicle-actuated signal control** - It is a system in which inductive loop detectors provide traffic demand information on an equipped intersection and is used to determine signal control. For example, you are extending or terminating a green phase.
3. **Adaptive signal control** - It is a system in which signal timing control is managed and updated automatically based on the current state of the intersection. For example, traffic demand, vehicle queue length in each lane of the intersection, and traffic flow fluctuation.

We are interested in the third approach in this paper and hope to propose novel methodologies for traffic signal control by leveraging recent advances in machine learning and artificial intelligence.

2.1 Reinforcement Learning

Reinforcement is one of the most exciting fields in Artificial Intelligence. Reinforcement learning uses Neural Networks to learn and form the representation based on which the actions are made. Deep neural networks can represent and comprehend the world to act on that representation. Trial and error is the fundamental process by which reinforcement learning agents learn. (6)

RL is the branch of machine learning, studying approximately optimal decision-making in natural and artificial systems. RL has many great applications, and it has already beat the human champion in the game of Chess, Shogi and Go. It means the RL agents can now

defeat the human champions, and artificial intelligence is over empowering the world. The cycle of how RL works is shown in Fig 2.1.

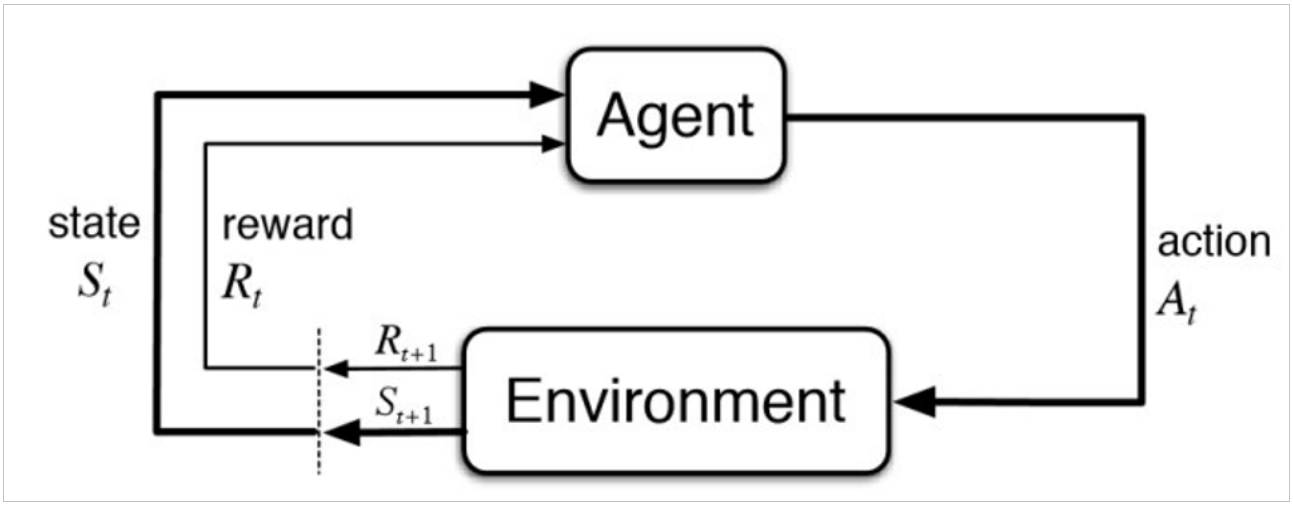


Figure 2.1: Reinforcement Learning.

This figure shows that we have a learning agent that takes action in the environment. The environment responds with the new state and a numerical reward. So, the agent observes the new state and again performs an action. It again revives a reward and the next state. This reward tells the agent how good or bad the action was. For instance, if the agent is playing a game, and if after performing an action, agent scores increase, then the reward is in favour of the agent, and it encourages the agent to exploit this action again in this kind of state. If the agent's score gets lower, the agent is penalised for not taking that action again. The cycle goes on, and after so many trials and errors, eventually, the agent learns the dynamics of the environment and always tries to get higher and higher scores which leads the agent to have a nearly optimal policy for that environment. Note that we mean a mapping from state to action with the policy.

Types of Learning :

1. Supervised Learning
2. Semi-Supervised Learning
3. Unsupervised Learning
4. Reinforcement Learning

Every type of Machine Learning is Supervised Learning, i.e. it is supervised by a loss

function or a function that tells what is good or what's wrong. The main difference between supervised and unsupervised reinforcement learning is the source of the supervision. Unsupervised is that the cost of human labour required for that supervision is low, but it is never zero. At some point or another, there has to be human intervention and human input to provide what is good or what is not. The main aim of Reinforcement Learning is to get that intervention as efficiently as possible.

Supervised Learning takes a lot of examples of data and learns from those examples where the ground truth provides the compressed semantic meaning of what's in that data and from those examples one by one as to which ones are sequences or single samples. We learn to take future models and interpret them.

Reinforcement Learning teaches an agent through experience, not by showing them any dataset but by putting them into the world. The essential design step is to provide the world we want to experience. The agent learns from the world. It learns the rewards, what's good and evil. The designers of the agent not only have to do the algorithm, but we also have to design the world in which that agent is trying to solve the task. The design of the world is a process of reinforcement learning. The method of examples and annotations of samples is called Supervised Learning. The most challenging element of Reinforcement Learning is to set the reward.

In the Reinforcement Learning framework at each step the agent:

1. Executes action
2. Observe new state
3. Receive reward

There is an environment, and an agent acts in that environment. The agent senses the environment by some observation. It gives the environment an action that performs in that environment, and through the action, the environment somewhat changes and then a new observation occurs. While we make observations, we also receive a reward. The things that need to be concerned about the state from which it came or the state in which it arrived, and the reward received.

2.2 Major components of an RL agent

An RL agent may be directly or indirectly trying to learn:

1. **Policy** - Agent's behaviour function.
2. **Value function** - how good is each state and/or action.
3. **Model** - agent's representation of the environment.

As state and action spaces get more significant, the challenge of utilizing reinforcement learning in real-world applications like traffic signal management climbs tremendously. Function approximation techniques and hierarchical reinforcement learning approaches can solve this challenge. Deep learning has recently received a lot of attention, and it's been paired with reinforcement learning approaches to solve complex optimization problems.

2.3 Types of RL algorithms

RL algorithms can be divided into two types:

1. Model-free RL algorithms
2. Model-based RL algorithms

Model-free RL can successfully solve various tasks, including video games and robotics, but it requires a considerable number of inputs to achieve good performance. It does not attempt to comprehend the environment. It only takes action in response to changes in the environment and feedback. According to the learning object, model-free RL can be divided into Q-learning and policy optimisation.

Model-based RL is learning the model in a class of dynamics that can quickly obtain near-optimal control. It allows agents to plan ahead of time and see what will happen due to a series of possible choices before making a decision. The agent takes action on the outcomes of planning into the learning strategy.(7)

This thesis will focus on model-free RL algorithms to control traffic lights. Model-free algorithms can further be classified based on:

1. Value based algorithms
2. Policy gradient algorithms

This thesis will focus on both value-based and policy gradient algorithms. The goal of value-based RL methods is to find the best Q-value function. The Q-learning algorithm is an example of a value-based RL algorithm. In policy gradient, on the other hand, we directly optimize the desired objective function. Policy gradient methods are inefficient in terms of sample size, but they can learn complex tasks more robustly than value-based RL algorithms.

2.4 Markov Decision Process

A Markov Decision Process(MDP) is a foundation(8) that is used to model decision-making problems that can have both random and controllable outcomes:

1. **Agent** - An agent is any entity that is trained to make correct decisions.
2. **Environment** - It is the surrounding of the agent where it can interact. The agent can't control its surroundings, it can only control its own actions.
3. **State** - The exact situation of the agent is defined as the state.
4. **Action** - The decisions made by the agent at a current time step are called actions. We can determine the actions that the agent is going to perform or is capable of performing.
5. **Policy** - It is the thought of the agent in deciding which actions it is going to take. It is a probability of the set of actions that it might take. Actions which have the highest award are most likely to occur.

A state S_t is considered to be a Markov if and only if it satisfies:

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, ..., S_t]$$

According to the Markov Property the state of the system only depends on the previous state, and it has no influence on all the previous steps.

The state transition property of a Markov process is:

$$P_{ss0} = P[S_{t+1} = s0 \mid S_t = s]$$

A process is called as a Markov Process(S, P), where S is the number of states and P are state-transition probabilities. It consists of random states such as S_1, S_2, ...

The state transition probability and the reward in a Markov Process is defined as:

$$P_{ss0} = P[S_{t+1} = s0 \mid S_t = s]$$
$$R_s = E[R_{t+1} \mid S_t = s]$$

A policy π is defined as the distribution of actions given its states

$$\pi(a|s) = P[A_(t) = a \mid S_(t) = s]$$

A policy(π) in a Markov Process is given over by a distribution of the actions given states:

$$\pi(a|s) = P[A_t = a \mid S_t = s]$$

The state value function for a Markov Response Process is given as:

$$v(s) = E[G_t \mid S_t = s]$$

The Bellman Equation represents value functions in different standard ways. It divides the value function into two parts:

1. Immediate reward R_(t+1)
2. Discounted value of the state in future γ v(S_(t+1))

$$v(s) = E[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s]$$

In this thesis, we used a Markov Decision process so that the RL agent could solve this problem. The state is a 21-dimensional vector that contains different traffic light phases(four phases represented by one-hot encoding), and for each lane, its total waiting time and density of cars stopped at the intersection. Action corresponds to a phase in the

traffic light. The reward corresponds to the negative waiting time. So, the agent's primary goal is to try to minimize the waiting time as much as possible.

2.5 Q-Learning

Q-learning is a reinforcement learning algorithm that does not require a model. It is a learning algorithm that is only based on values. Value-based algorithms use equations which update the value function. In the case of policy-based algorithms, it estimates the value function with a greedy policy obtained from the most recent policy improvements.(9)

Q-learning seeks the best action to take given in the current scenario. It also tries to learn a policy that maximizes total reward. It learns from actions outside the scope of the current policy, such as performing random activities, so there is no need for a policy.

In Q-Learning, the letter 'Q' stands for quality. It represents how useful action would be in gaining future rewards.

1. Q*(s, a) is represented as the expected value, i.e., the cumulative discounted reward of doing an action in state s and then implementing the optimum policy.
2. Q-Learning can estimate the value of Q* by using a Temporal difference (s, a). The temporal difference is defined as an agent which learns from an environment with no prior knowledge of the environment.
3. The agent keeps a Q[S, A] table, where S is the set of actions, and A is the action state.
4. Q[S, A] is the current state of Q*(s, a).

2.5.1 Q-Learning Algorithm

1. Set up the Q-table.
2. Select an Action.
3. Carry out an Action.

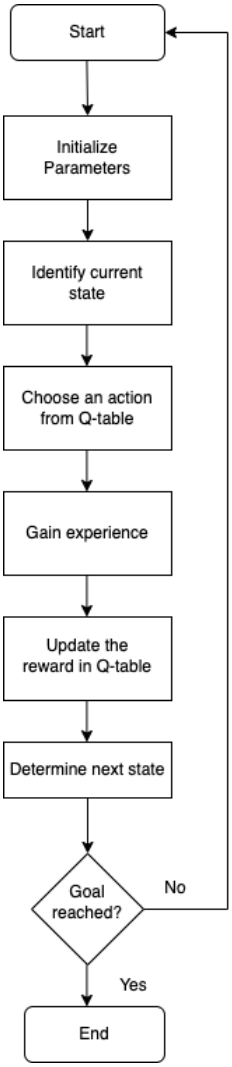


Figure 2.2: Q-Learning algorithm

4. Calculate Reward.
5. Evaluate

2.6 Deep Learning

Artificial intelligence (AI) is a technique that allows humans to mimic human behaviour. Machine Learning is a technique for achieving AI through data-trained algorithms. Deep Learning is a type of Machine Learning that is influenced by the structure of the human brain. Deep Learning is a subset of Machine Learning, a subset of AI. The system is referred to as an artificial neural network. Without human intervention, distinguishing features are identified using a neural network.

It learns from experiences, interprets the information, and interprets it in a hierarchy of

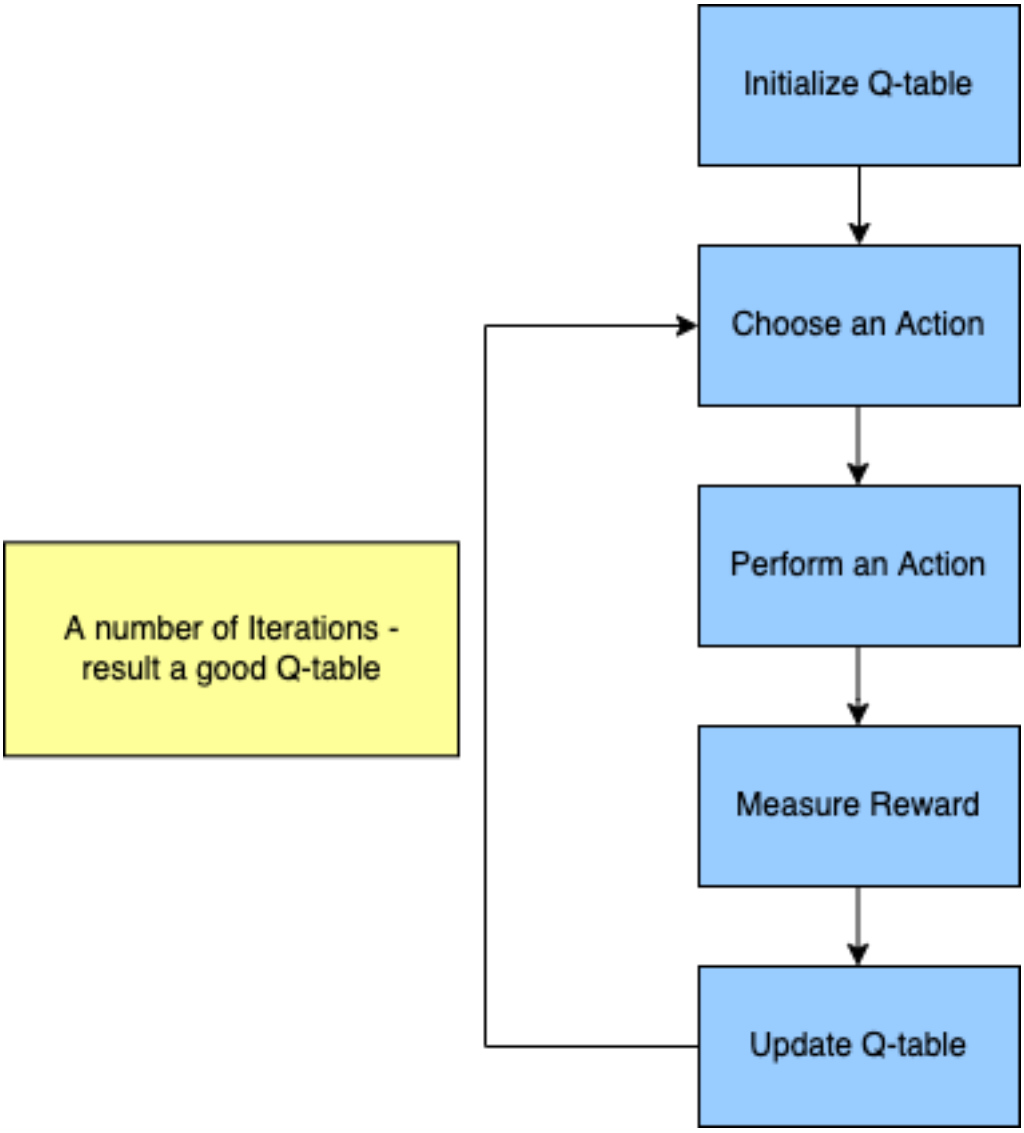


Figure 2.3: Q-Learning Algorithm

concepts. So it enables it to remember complex concepts by constructing them from simpler terms layer by layer. As it learns through experiences, human intervention is not always required.(10)

Due to its ability to learn complex structures using backpropagation, deep learning can be used for face recognition, image classification, speech detection, text-to-speech generation, handwriting transcription, machine translation, medical diagnosis, self-driving cars, traffic lights, digital assistants, recommendation systems etc.

2.6.1 Neural Network

Neural Networks are the base of Deep Learning. They take in data, train themselves, recognise patterns in this data and then predict outputs for a new set of similar data.

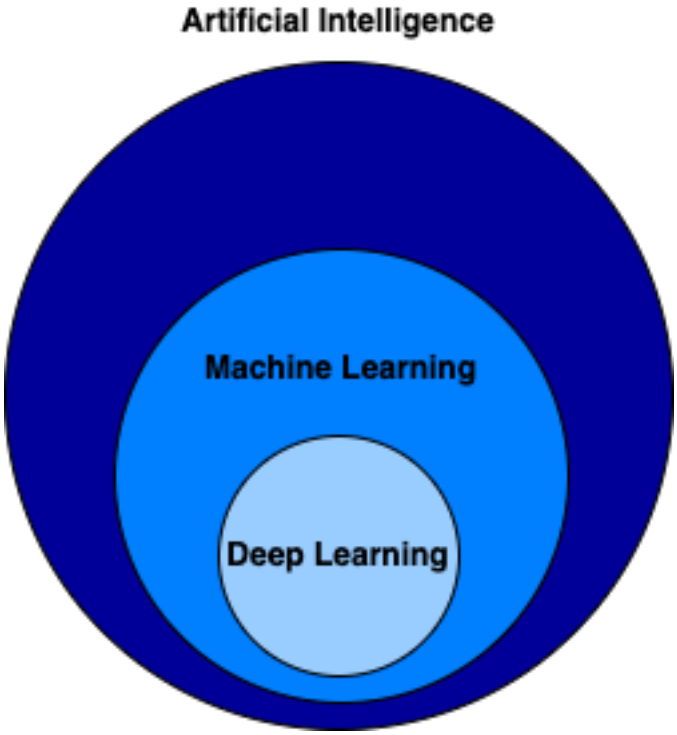


Figure 2.4: AI vs ML vs DL

Input layer - These are represented by purple. It is used to input data into the network. These can be text, vectors or multi-dimensional matrices.

$$x_i = a_i^1,\ i \in 1,2,3,4$$

Hidden layer - There are two hidden layers in this network coloured in green. These layers allow the neural network to learn complex patterns in data.

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

Output layer -

The ouput layer is the final layer in this network coloured in blue which gives the predictions.

$$s = W^{(3)}a^{(3)}$$

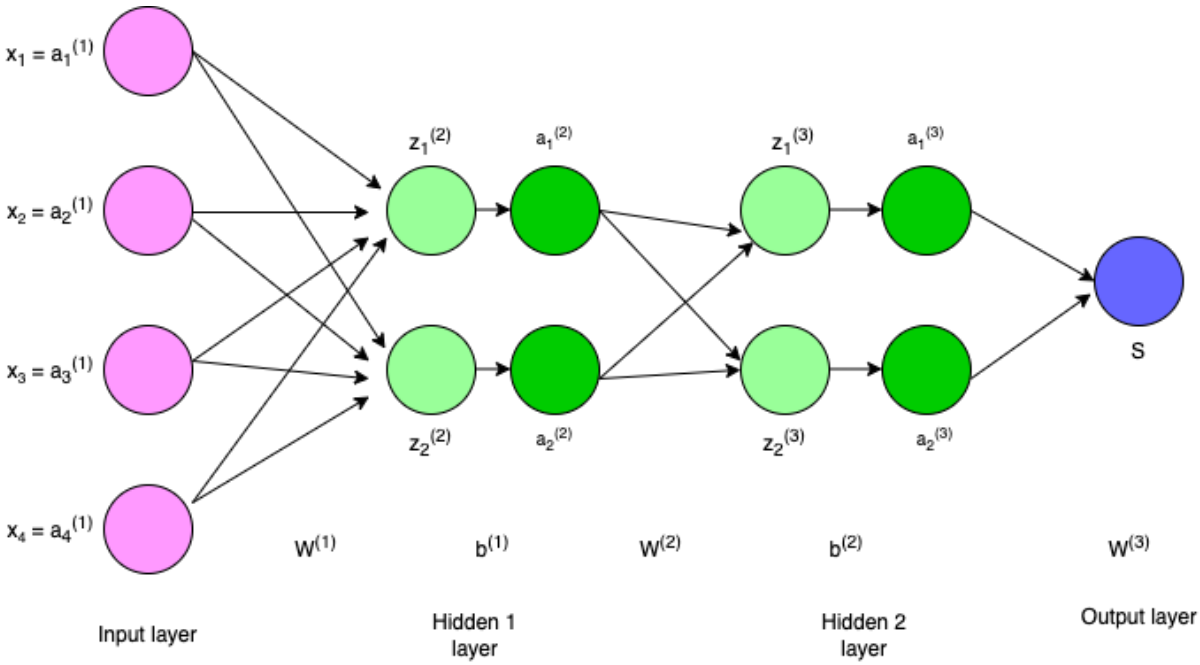


Figure 2.5: Simple 4-layer neural network

2.6.2 Backpropagation

Backpropagation is a technique for fine-tuning the weights in a neural network using errors obtained in the previous iteration. By fine-tuning weights, we can make the model errorless and reliable by increasing generalisation. It is a common technique for training artificial neural networks. This method aids in calculating the gradient of a loss function concerning all of the network's weights.

It uses the chain rule in neural networks to compute the gradient of the loss function for a single weight. It generalizes the delta rule computation. Unlike a native direct analysis, it efficiently computes one layer at a time.(11)

The benefits of backpropagation are:

1. It is straightforward, quick and easy to program.
2. The only parameter we can tune is the number of inputs.
3. It does not require the prior knowledge of the network.
4. It generally works very well.

2.6.3 Convolutional Neural Networks

CNNs were designed to mimic the human nervous system. Convolutional Neural networks combine both deep neural networks and kernel convolutions. It specialises in pattern recognition. It can take an image, assign some weights, and distinguish one from the other. While filters in primitive methods are hand-engineered, ConvNets can learn these filters/characteristics with enough training. Compared to different classification algorithms, the amount of pre-processing required by a ConvNet is significantly less. (12)

ConvNets do not have to be limited to a single Convolutional Layer. Traditionally, the first ConvLayer is in charge of capturing Low-Level features such as edges, colour, gradient orientation, etc. These aim to extract high-level features from the input image, such as edges. With more layers, the architecture adapts to the High-Level features, giving us a network that understands the images in the dataset in the same way that we do.

2.6.4 Optimization Algorithms

The optimisation technique is a way to change the characteristics of the neural network, such as learning rates and weights, to reduce losses. These strategies help decrease losses and give the most accurate results possible.
The most used optimisation algorithms are gradient descent and stochastic gradient descent.

Gradient descent is the most widely used optimisation algorithm. It is primarily used in linear regression and classification algorithms. It can also be implemented in the neural network backpropagation.
The main objective of the gradient descent is to minimise the cost function by updating the learning rate until we reach a minimum. The learning rate is chosen in such a way that it matches the global minimum.
The most used Deep Learning algorithms are:

1. Adagrad
2. RMSProp

3. ADAM
4. AdaDelta

2.7 Deep Reinforcement Learning

Deep Reinforcement Learning is a form of unsupervised learning that uses internal rules to extract information from a massive amount of data. Data is sent to the hidden layers to learn pieces of information without any need for human interference. It has been very helpful in processing a tremendous amount of data, such as image and speech recognition.(6)

Deep Reinforcement Learning combines the ability of Deep Learning with high-dimensional data. It improves the stability of Deep reinforcement learning by using deep neural networks(DNN). It consistently updates the policy function or value functions by updating function approximation or gradient ascending and then assists the agent in making decisions. It produces amazing results in Atari games and Alpha Go. (13)

Due to its excellent results, people have transferred this model to traffic signal control (TSC) since deep reinforcement learning has demonstrated its superior ability to play games. The model derives its environmental information about traffic conditions from sensors or cameras. The key to DRL application on TSC, according to this model, is how to design State, Action, and Reward.(13)

2.8 Urban traffic control

Urban Traffic Control (UTC) plays a crucial part in the Intelligent Transportation System (ITS), yet it is challenging to implement because of its complexities in dynamics. Model-based UTC systems are not good at explaining the nature of traffic dynamics in all cases. Due to this reason, model-free data-driven UTC methods, specifically reinforcement learning-based UTC methods, have been a hot topic recently. (14)

Reinforcement Learning has been very effective in many areas other than traffic management systems, and it has shown positive results. RL allows a system to learn how to choose its behaviour depending on the feedback it receives from its surroundings, i.e., environment. RL-based approaches to traffic management problems typically use the traffic flow states surrounding intersections as observable states, signal timing plan changes as actions, and traffic control performance as feedback. (15)

Deep Q-Networks(16) were used to control a single intersection in one of the first attempts to tackle traffic control problems using DRL methods (17). This has been used by applying it to various other settings such as traffic light coordination (18). New traffic state encoding methods (19) and numerous other multiple models have been proposed such as Deep Deterministic Policy Gradient to improve such systems.

Deep neural networks proposed to simulate the link between states and actions does not fit well for large scale complex UTC problems with many intersections. When the correlation between crossings are interwoven the reward functions which were designed for RL does not accurately define the state of the traffic systems. There are certain training methods for DRL based UTC models which fails to find balance, and they are also too slow to solve large scale UTC problems satisfactorily.

To tackle these problems numerous techniques are introduced such as Residual Networks(ResNet) to learn relationships between different states and actions. Advantage Actor-Critic (A2C) is a type of architecture which does not strictly follow the direction indicated by the gradient ascent $E[R_t]$. Instead of considering cumulative returns R_t policy gradients are evaluated and scaled by advantage $A_t(a_t,s_t)$

The advantage $A_t(a_t,s_t)$ can be calculated as the return subtracted from a learned baseline function $b_t(s_t)$.
We have discussed some RL algorithms that are used for traffic light optimisation:

2.8.1 Deep Q Networks

Deep Q-Network (DQN) can combine reinforcement learning with deep neural networks. Recent advances in deep neural networks, which use multiple layers of nodes to build progressively more abstract data representations, have enabled artificial neural networks to learn concepts such as object categories directly from raw sensory data. The tasks in which the agent interacts with its surroundings through observations, actions, and rewards. The agent's goal is to choose activities that maximize cumulative future rewards.(16)

In Deep Reinforcement Learning, the agent interacts with the environment and learns the best policy. By observing the environment at time t, the agent obtains a state s. The agent then takes appropriate action in that state s under the policy. Finally, the environment is changed to a new state s under agent a with scaler reward r. By evaluating r, the agent learns an action-value function Q(s, a) or the policy. This step is repeated again and again

until the agent masters the ability to formulate the policy. (20)

Reinforcement Learning is very unstable so DQN addressed these instabilities by applying two insights: experience replay and the target network.

Experience replay has three significant benefits. It improves data efficiency because each step of the process can be potentially used for weight updates. Because the samples are correlated, learning directly from the consecutive examples is inefficient. So this can be corrected by randomizing the instances, these correlations can be broken, and the variance of the updates can be reduced. Then, the correct parameters dictate the data sample on which the parameters are trained when learning about policy. Using the DQN, the behaviour distribution is averaged over different prior states, which stabilizes the learning and thus avoids fluctuations.

Q-Learning algorithm solves a Markov decision process (MDP) by determining the best state action value pair or Q-function. An MDP is represented as (S, A, T, R, (γ)). The state and action pair is represented by A. T(s0|s,a) and R(s,a) represents dynamics i.e, transition distribution and reward function with (γ) $\in$ (0,1) representing discount factor. The goal of an RL is to find the policy (π)(a|s) that is the maximum of the cumulative rewards.

$$\pi^* = \text{argmax E s0}\sim \rho 0, st+1 \sim T, at \sim \pi \ [\ \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$$

2.8.2 Multi-Agent Reinforcement Learning

The study of how multiple agents interact in a familiar environment is known as multi-agent reinforcement learning. We can observe different agents collaborating, coordinating, competing or learning collectively to accomplish a common task by interacting with the environment and with one another. It is further divided into three categories:

1. **Fully Cooperative**: Different agents working together to achieve a common long term goal.
2. **Fully Competitive**: Different agents competing against one another in order to

achieve a single goal. The return of agents sums up to zero.

3. **Mix of two**: In this, some agents are coordinating with one another and some against each other.

Multi-Agent Reinforcement Learning (MARL) can be used in many fields such as traffic light control, network routing, economic modelling and analysing of social dilemmas.
The actions taken by one agent can result in different rewards depending on the actions taken by the other agent. After achieving convergence, MARL can typically maintain acceptable performance in terms of the quality of the policies derived and the convergence speed only when a limited number of agents are involved.(21)

The main challenge of MARL is that multiple agents learn concurrently, which results in a non-stationary environment for each other. The action of one agent in the environment influences the reward of the opponent agent and the evolution of the state. The individual compensation and current state are dependent only on the previous state and actions were taken. Therefore, the learning agent may take into account how other agents behave and adapt to the environment as a whole. This renders the assumption of stationarity and the convergence of the single-agent RL algorithm.

Standard Q learning algorithm is yielded through consensus and innovation giving the QD-learning algorithm.

$Q^{i}_{t+1}(s,a) \leftarrow Q^{i}_{t+1}(s,a) + \alpha_{t,s,a}[R^{i}(s,a) + \gamma max Q^{i}_{t}(s',a') - Q^{i}_{t}(s,a)] - \beta_{t,s,a} \sum_{j \in N^{i}_{t}}^{\infty} [Q^{i}_{t}(s,a) - Q^{j}_{t}(s,a)]$

Here, $\alpha_{t,s,a}$, $\beta_{t,s,a} > 0$ are the stepsizes, N^{i}_{t} indicates the number of agent neighbouring to the given agent i at time t. QD-learning captures the difference between the Q-value estimates. The algorithm is guaranteed to converge to the optimum Q function for the tabular setting if certain conditions on the stepsizes are met.(11) (22) (23)

2.8.3 Advantage Actor-Critic (A2C)

A novel system that achieves complete traffic light decision making combines both reinforcement learning with adaptive learning, which allows the system to dynamically select the adaptive phase and the duration based on the current traffic situation. The phase

decision is made using Advantage Actor-Critic (A2C), which can choose the best phase at each step. Then the concept of adequate green time is examined and found that effective green time utilization is highly correlated with time loss minimization. As a result, a new adaptive timing algorithm is devised which can adjust the optimal phase length to minimize green time loss.

A2C algorithm is a state of the art learning algorithm which has higher learning ability and is very robust when compared to Q-Learning algorithm when experimented on traffic light scenaio.

A2C algorithm is a state of the art learning algorithm which has higher learning ability and is very robust when compared to the Q-Learning algorithm when experimented on traffic light scenarios.

A2C algorithm is different from Q-Learning as it is based on the policy gradient method. It directly learns the optimal policy by interacting with the environment. A2C algorithm can be divided into two parts: actor and critic. The actor interacts with the environment by observing it and choosing an action to perform. The actor understands what the optimal action is in different states with the help of the critic. The critic then assesses the actor's choice of action, i.e. how good or bad it is. The critic's evaluation of action becomes increasingly accurate during the synchronous learning process. In the case of traffic light control A2C model is used to make a phase decision.(24)

The formula for critic is represented as:

$$A^{\pi}(s_t,a_t) \approx r(a_t, s_t)+\gamma\ V^{\pi}_{\emptyset}(s_{t+1})\ -\ V^{\pi}_{\emptyset}(s_t)$$

Some of the limitations of the current approach are that first A2C may sometimes make lousy phase decisions which can be mitigated by using other better reinforcement learning models. Second, this experiment can be conducted in various intersections to validate its performance. We also hope to apply this method to more complex situations, which include multiple lanes and junctions with pedestrians, in order to make this process more applicable to the real world.

Algorithm 1 Adaptive Timing Algorithm

The current phase begins after A2C model makes phase decision
Set $t_{remaining} = T_{max}$
while $t_{remaining} > 0$ **do**
    Continue a second for current phase and then collect traffic information
    $t_{remaining} = t_{remaining}$ - 1
    if $h_{min,g} > threshold_1$ and no vehicle within D of the intersection then **then**
        if $h_{min,g}$ - $h_{min,r} > threshold_2$ **then**
            Terminate current phase
        end if
    end if
    if $h_{rear} > t_{remaining} > threshold_1$ **then**
        Terminate current phase
    end if
end while=0

2.8.4 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is a reinforcement learning technique that employs a policy gradient that alternates between sampling data via environment interactions and optimising an objective function using a stochastic gradient ascent. Using trust region policy optimisation for PPO offers certain advantages, and it is also relatively simple to apply.(25)

The PPO algorithm employs fixed-length trajectory segments. Each iteration of N parallel actors collects T time steps of data. The loss is then built on these NT time steps of data and optimised with minibatch SGD for K epochs.(26)

Algorithm 2 PPO, Actor Critic Style

for *iteration* $= 1, 2, \ldots$ **do**
    for *actor* $= 1, 2, \ldots, N$ **do**
        Run policy $\pi_{\theta_{old}}$ in environment for T time steps
        Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
    end for
    Optimize surrogate L wrt. θ, with K epochs and minibatch size $M \leq NT$
    $\theta_{old} \leftarrow \theta$
end for=0

The PPO algorithm comprises two threads: one that collects data, calculates advantage

estimations, and samples mini-batches for the other thread to employ. N parallel actors accomplish these tasks, each performing their responsibilities separately.

2.9 Summary

In this section we discussed backgrounds in reinforcement learning, its types and major components. Then we discussed Markov Decision Process, Q-Learning and its algorithm, Deep Learning, Neural Network, Backpropagation, Convolutional Neural Networks (CNNs), Optimization algorithms, Deep Reinforcement Learning. We then talk about Urban traffic control and discuss some other state-of-the-art RL techniques to that are used such as Deep Q-Networks, Multi-Agent Reinforcement Learning, Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO) algorithms.

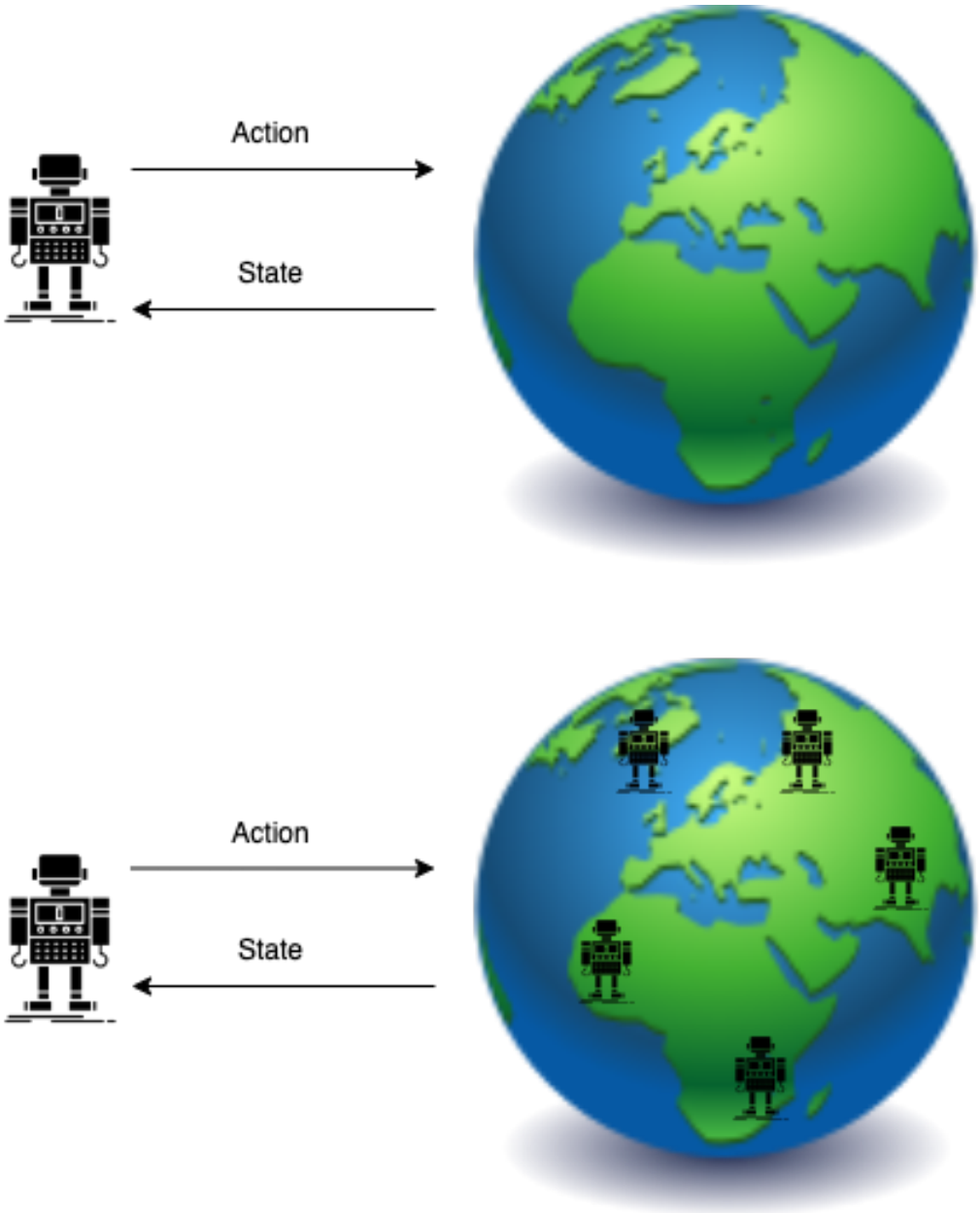


Figure 2.6: (a) Single-agent RL agent interacts with the environmentby performing action and receiving a reward.(b) In MARL algorithm the agent's point of view and other agents can be considered to be a part of the environment, whcih changes due to the actions by other agents.

3 Design

This section discusses how different components of the traffic light optimisation RL algorithm were designed. This section describes how to solve the problem of single and many intersections using applied algorithms.

3.1 Traffic Light Control Problem

At discrete time step t, the Traffic Light Control problem is stated as an RL problem. An agent interacts with the environment, a partial segment of the road network containing the intersection with the traffic light that it is supposed to manage. Specifically, the agent observes a state $s_t \in S$ at the start of time-step t, then selects and executes an action $a_t \in A$ corresponding to one of the traffic signal configurations. The agent switches to the following state $s_{t+1} \in S$ after performing the action $a_t \in A$ and obtains a reward r_t.

3.2 Traffic Simulator

Simulation of Urban MObility(SUMO) is a traffic simulation platform that is free and open source. It was created in 2001, and it allows different simulations of intermodal traffic systems, which includes road vehicles, public transportation, pedestrians and comes with a comprehensive set of scenario-creation tools. SUMO contains numerous supporting tools which can automate some core tasks of creating, running and then evaluating the traffic simulations, which includes network import, route calculation, visualization, and emission calculation. SUMO can be customised with custom models, and it offers a variety of APIs which can be used to control the simulations remotely (27).

3.3 SUMO Uses

SUMO has been used in numerous projects (28) (29) to help answer a very wide range of research and development questions which includes:

1. Evaluating the performance of the traffic lights which includes everything from evaluating the modern algorithms to evaluating weekly timing plans.
2. Vehicle route choice was studied, which includes the development of new methods, the evaluation of eco-aware routing based on pollutant emission, and research into the network-wide influences of autonomous route choice.
3. Training AI algorithms on traffic light plans.
4. Used to simulate parking traffics.
5. Simulations of different traffic effects of self-driving cars.
6. Simulate and validate autonomous driving functions.
7. Analysis of traffic safety and the risks involved.
8. To calculate emissions produced such as noise and pollutants.

3.4 Figures

Single agent case: In this scenario we created a traffic simulation using SUMO with only one intersection. Each side of the road is named as North, East, West and South as traffic is passing from all the sides. Here each street has three lanes. The first lane is for straight driving, while the other two lanes are assigned for driving left and right.

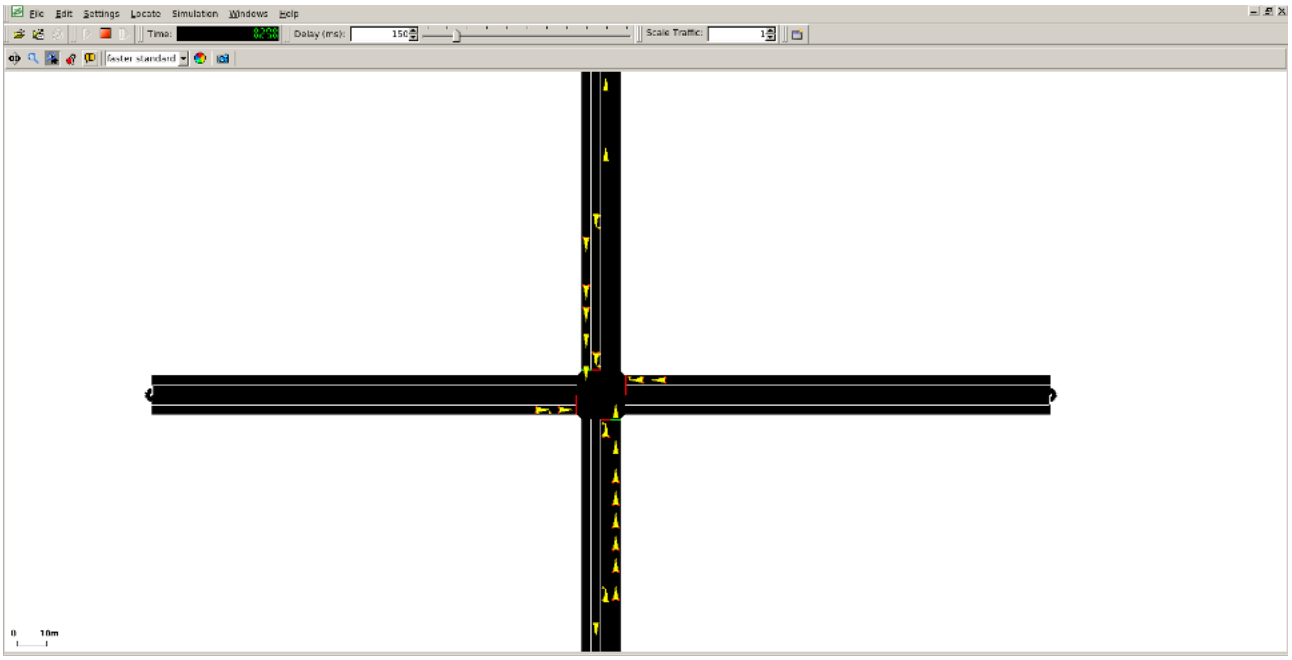


Figure 3.1: Traffic simulation in SUMO using single agent case.

Multi-agent case: In this scenario we have used double intersection where each agent controls its own intersection. Here we have first constructed a wrapper class which can be used to embed single-agent reinforcement learning on multi-agent domains using pettingzoo library. (16)

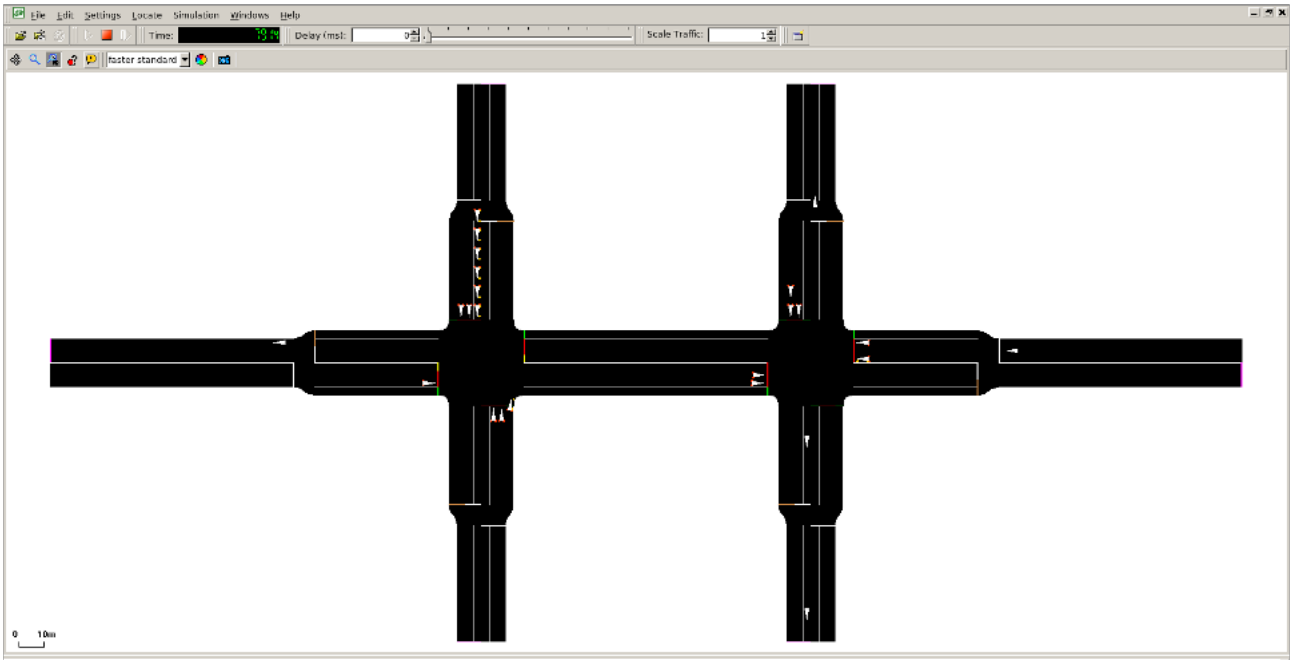


Figure 3.2: Traffic simulation in SUMO using multi-agent case.

3.5 Traffic Light Control Problem

We use different reinforcement learning algorithms to interact with the environment. Although the algorithms differ in characteristics and policies, their interactions with the simulation are very similar. TraCI has been employed to establish a connection between SUMO with the agents. The acronym TraCI stands for "Traffic Control Interface." It allows you to retrieve values of simulated objects and alter their behaviour "on-line" by giving you access to a running road traffic simulation.

TraCI is the API that receives and processes the data and commands. TraCI is used to start the agent and initialise SUMO. The agent can command SUMO to decide what to do when heavy traffic or when to open which road. TraCI uses SUMO to receive information about how many vehicles are currently on the street. Feedback is only required in the case of model-free algorithms.

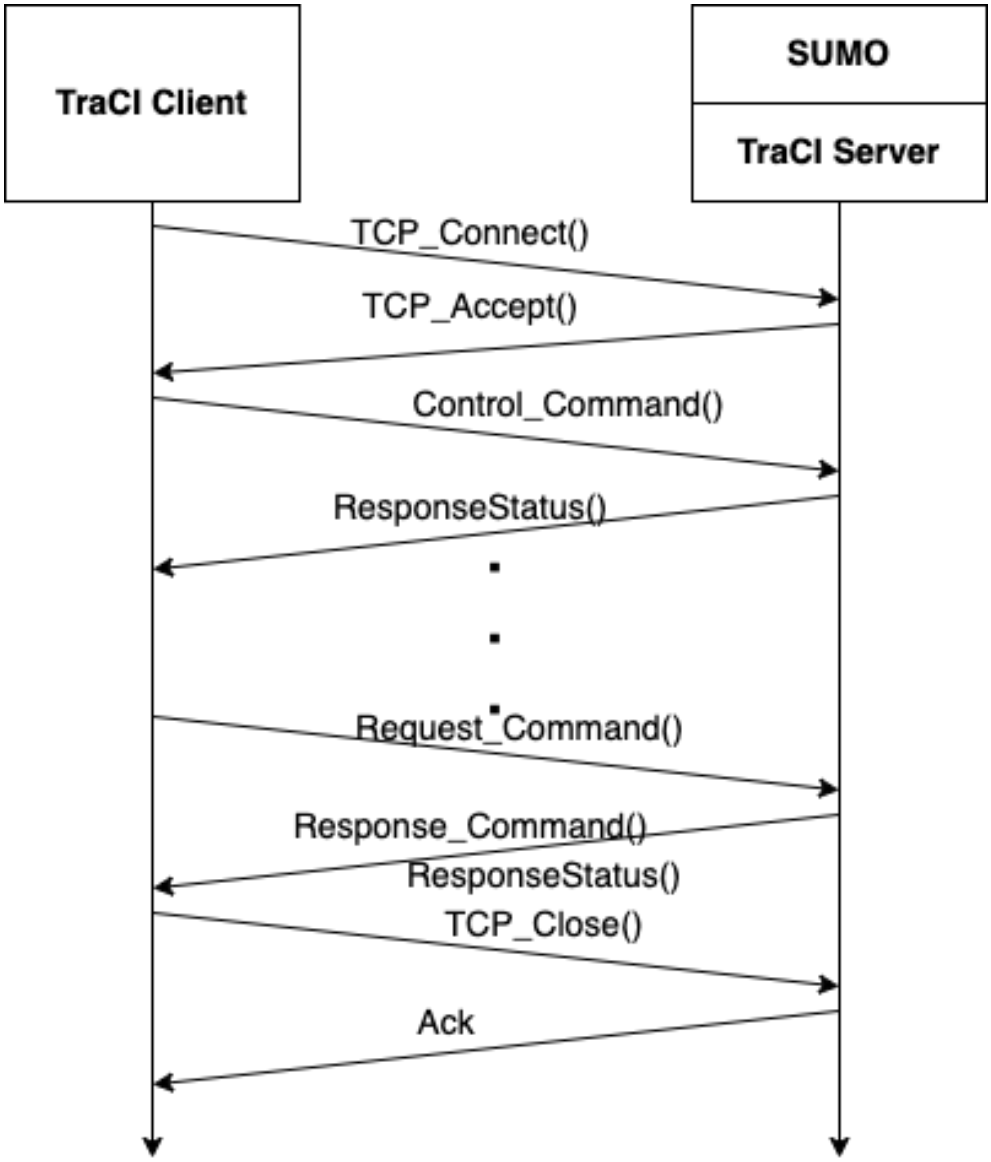


Figure 3.3: Implementation of communication between SUMO and TraCI client

The configurations slightly changes in-case of conventional simulations and those that does not require external methods. As there are no agents involved in the situation so the communication and execution only occurs directly between SUMO and TraCI. The majority of the command is handled by SUMO, which employs its default methodology.

3.5.1 State Representation

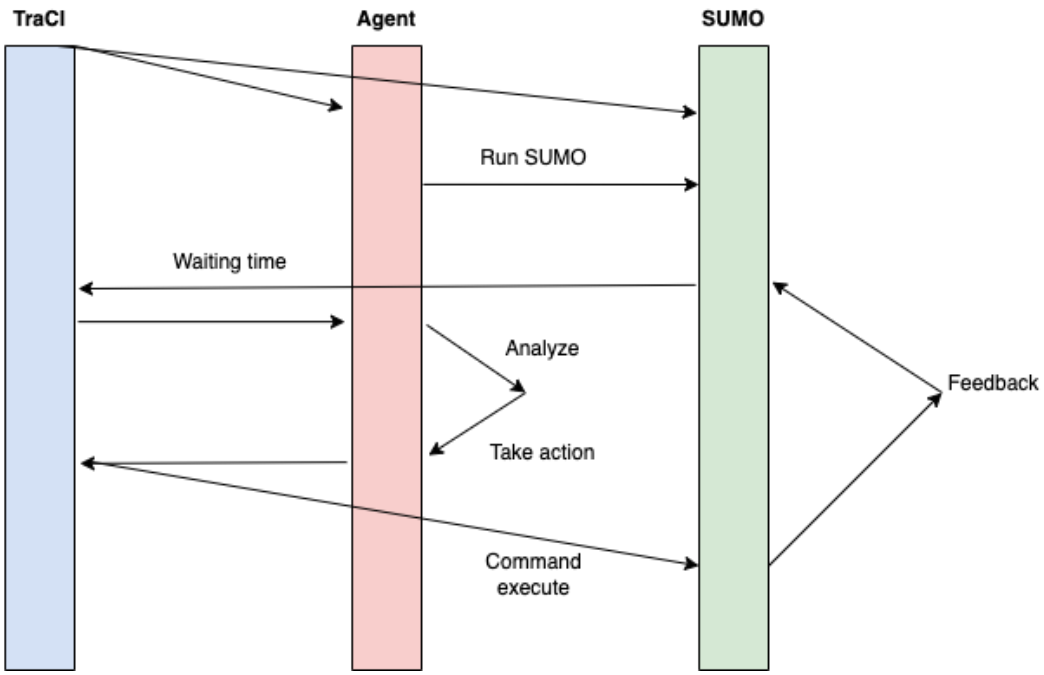


Figure 3.4: Representation of simulations with environment

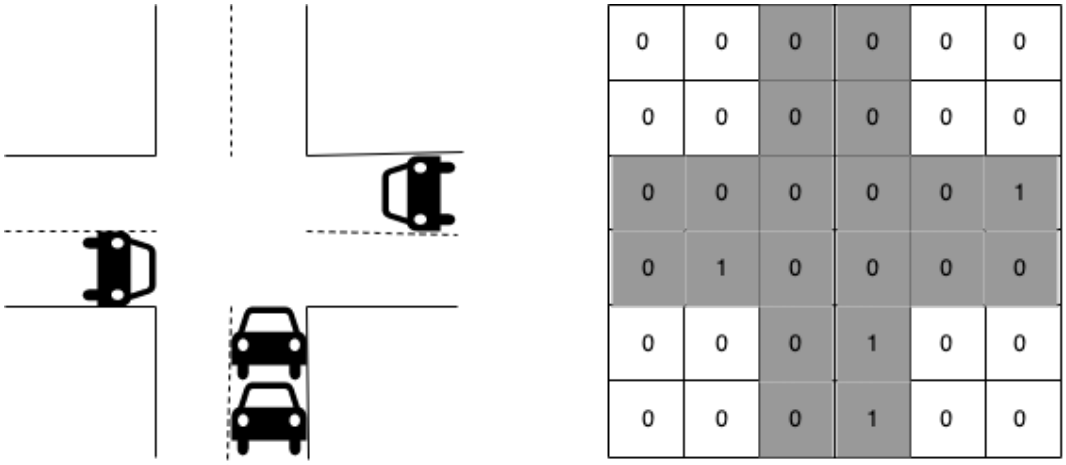
The state in SUMO is an image like representation of the simulator's environment's current state. The state is made of two identical sized matrices: (30) (18)

1. A binary matrix representing the positioning of vehicles.
2. A binary matrix representing the speed of vehicles.

The location is estimated by mapping the continuous space of the simulated environments into a discretised environment by building grids. The matrix represented in Fig 3.5 represents the positioning of vehicles in a particular area. The matrix illustrated in Fig 3.6 represents the vehicle's speed in that exact position. The speed is calculated as the vehicle's current speed to that of the maximum speed that can be achieved. The state of the system is represented by the number of cars, average waiting time of cars and the queue of cars in different directions.

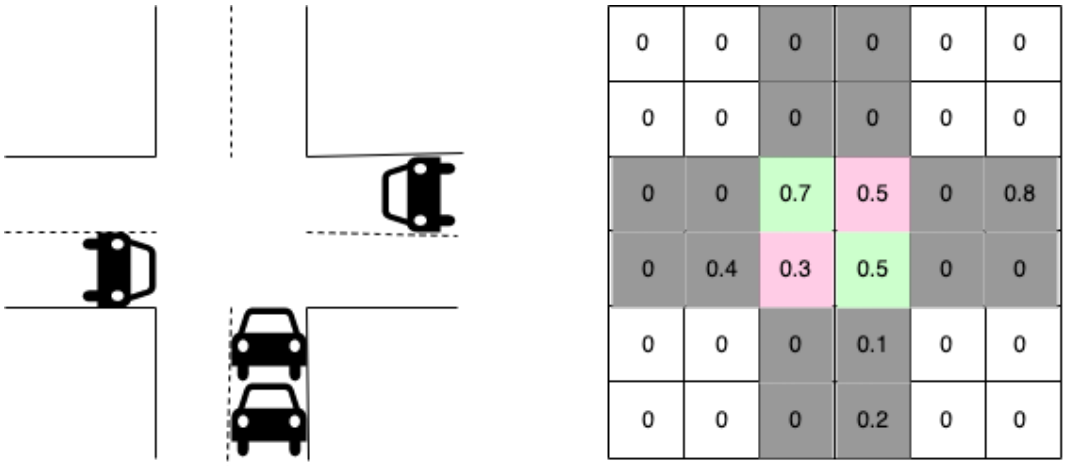
3.5.2 Action Space

After an agent observes a state S_t at the start of each time-step t, the agent choses an action A. Each side of the road is named North, East, West and South.
In the single agent case there are three lanes. The first lane is for going straight and the other two are for going left and right. The representation is shown in Fig 3.7. The action space is defined as



0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	1
0	1	0	0	0	0
0	0	0	1	0	0
0	0	0	1	0	0

Figure 3.5: Classic positional image-alike matrix



0	0	0	0	0	0
0	0	0	0	0	0
0	0	0.7	0.5	0	0.8
0	0.4	0.3	0.5	0	0
0	0	0	0.1	0	0
0	0	0	0.2	0	0

Figure 3.6: Normalized speed image-alike matrix with coded signal plan

In the case of multi-agent each intersection is controlled by a single agent, and we have two intersections.

The agent can only choose one of the actions for each time step t. Each phase has a duration of a one-time step. However, the agent implicitly selects the length of each phase, which might range from 1-time step to the simulation's final time step. Because the agent can only change the phase when it chooses to, there is no limit on how long a phase can last. In this problem, we've only looked at the red and green phases.

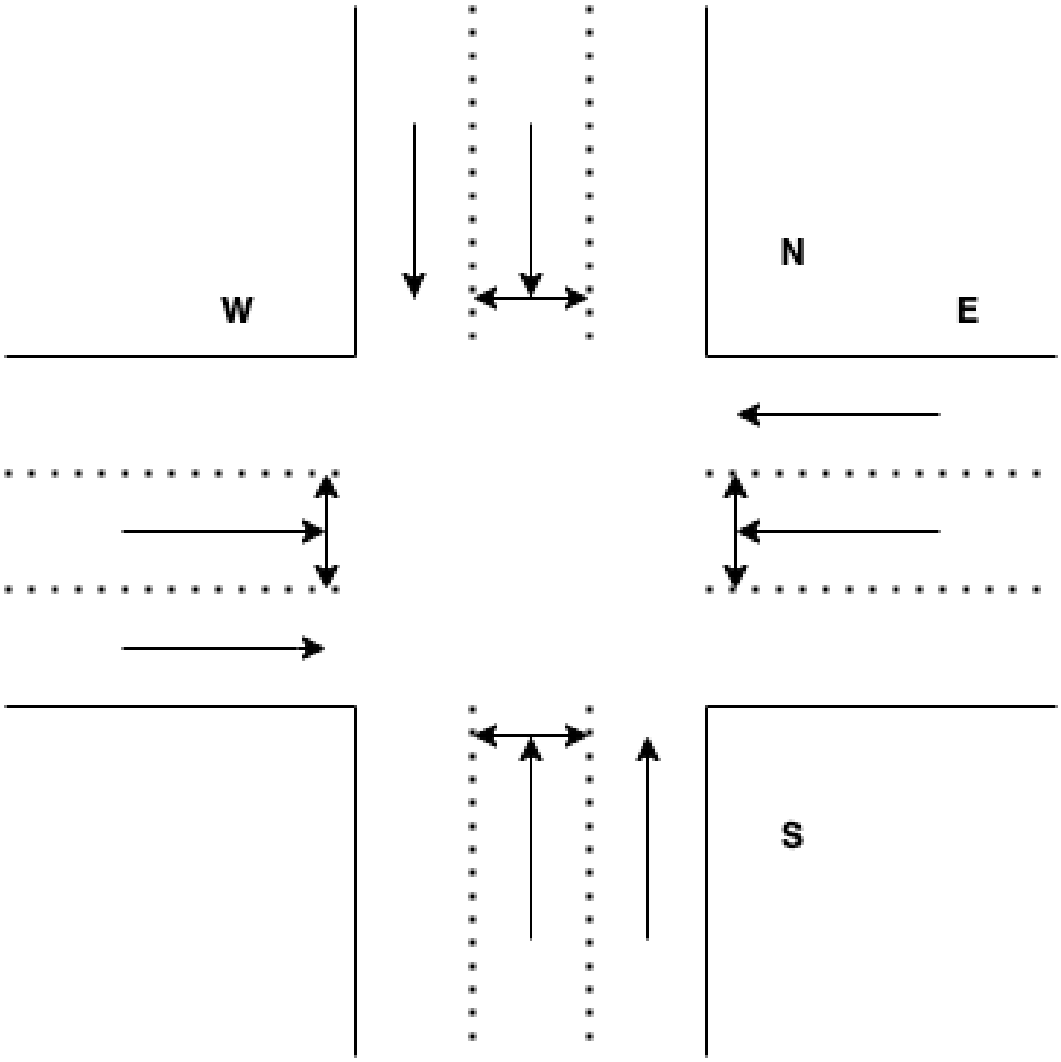


Figure 3.7: Intersection scenario in a single agent

3.5.3 Reward Function

A reward in a reinforcement learning is a scalar value $r_t \in R$ which is chosen in such a way that it converges with the optimal policy. It is given when an agent executes an action a_t in the environment after every time-step t.

Let's assume that $w_{i,t}$ be the total waiting time for the ith vehicle in this environment and W_t be the total waiting time for all the vehicles in this environment at time step t. The reward function is the negative of the total waiting time at time step t. The goal of the agent is to reduce the waiting time as much as possible inorder to receive maximum reward. In the real world scenario our goal is to reduce the waiting time of vehicles so this reward function corresponds to that goal.

$$W_t = \sum_i w_{i,t}$$

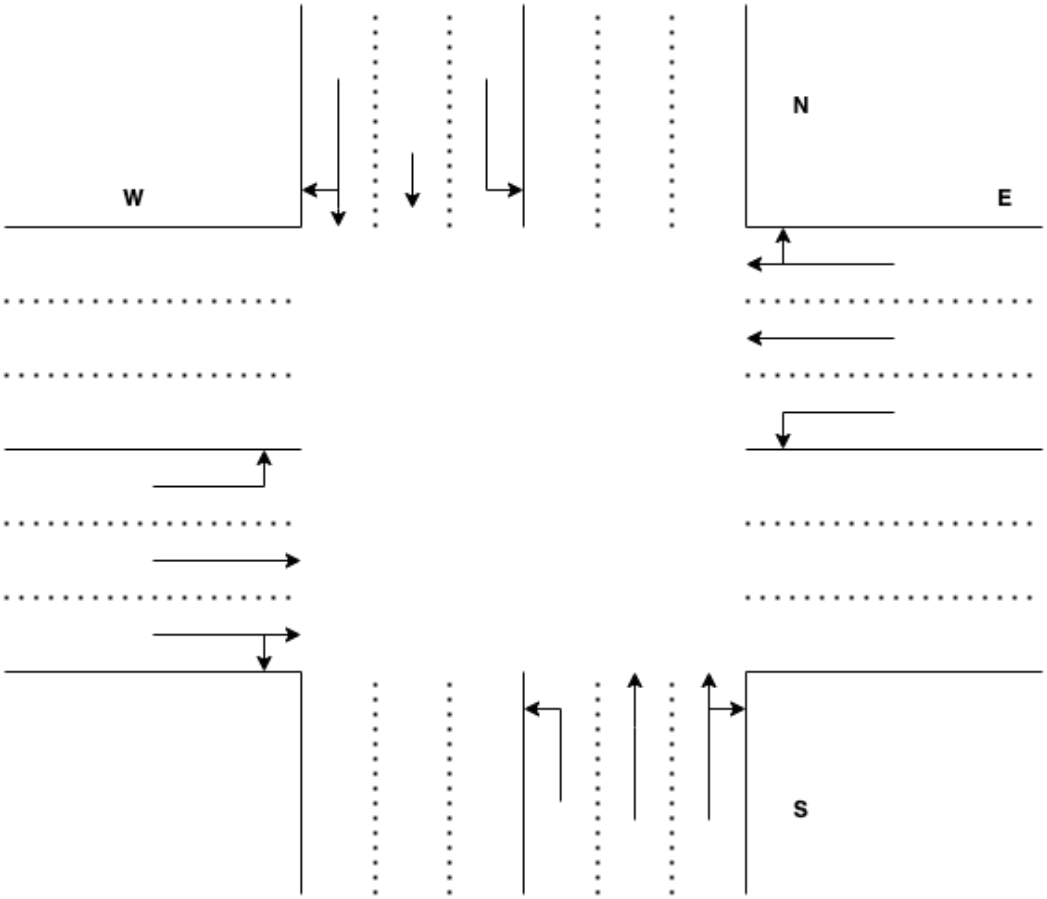


Figure 3.8: Intersection scenario in a multi agent

$$r_t = - W_t$$

3.6 Reinforcement Learning Techniques

We will look at the various parameters and hyperparameters of Reinforcement Learning approaches that are utilized in all the experiments in this part.
The experiments that were carried out in order to choose the best hyperparameters for all the studies are presented here.

Table 3.1: DQN techniques evaluation hyper-parameters

The reward in DQN is the negative of the waiting time. So from Fig 5.4, we can see that DQN has a very high waiting time in the range of 100000. DQN is very unstable. It does not show convergence even after a very large number of steps.

- **Action Space:** The action space remains constant.

Parameter	Value
Max steps	10000
Total time steps	100000
Current Step	0
Gamma	0.99
Learning Rate	0.01
Learning Starts	0
Train frequency	1
Target update interval	100
Exploration initial episodes	0.05
Exploration final episodes	0.01
Verbose	0

Table 3.1: DQN evaluation parameters.

- **Reward function:** The reward function remains the same throughout the number of steps.
- **DQN technique:** The DQN function was called from the stable baseline module and was given the following hyperparameters.

Table 3.2: A2C techniques evaluation hyper-parameters

Parameter	Value
Max steps	10000
Time Steps	800000
Depart Delay	0
Number of steps	5
Step Length	0.4
Current Step	0
Gamma	0.99
Learning Rate	0.0005
Verbose	0

Table 3.2: A2C evaluation parameters.

The reward in A2C is the negative of the waiting time. From Fig 5.5, we can see that the waiting time keeps on decreasing, and it nearly touches zero. It starts from 1000 and touches zero within 60000 steps. It converges and learns from its surroundings after a number of steps.

- **Action Space:** The action space remains constant.
- **Reward function:** The reward function remains the same throughout the number of steps.

- **A2C technique:** The A2C function was called from the stable baseline module and was given the following hyperparameters.

Table 3.3: PPO techniques evaluation hyper-parameters

Parameter	Value
Max steps	10000
Time Steps	800000
Depart Delay	0
Number of steps	128
Step Length	0.4
Current Step	0
Gamma	0.99
Learning Rate	0.0005
Number of epochs	20
Batch size	256
Clip range	0.2
Verbose	0

Table 3.3: PPO evaluation parameters.

The reward in PPO is the negative of the waiting time. From Fig 5.6 we can see that the waiting time decreases, nearly touching zero. It starts from 1400 and touches zero within 60000 steps. It converges and learns from its surroundings after some number of steps. Comparing A2C and PPO we find that A2C converges quickly even though PPO is the state-of-the-art RL algorithm. The environment considered in this situation is very simple with a single intersection this is why both A2C and PPO give good results.

- **Action Space:** The action space remains constant.
- **Reward function:** The reward function remains the same throughout the number of steps.
- **PPO technique:** The PPO function was called from the stable baseline module and was given the following hyperparameters.

3.7 Summary

The judgments made for the primary components of the single and multi-agent RL for the UTC challenge are presented in this section. The significant features of the RL were described, including state and action spaces and the reward function. It also included some hyperparameters for the RL algorithm.

4 Implementation

This section discusses different single-agent and multi-agent UTC simulation environments. We will explain the working of varying RL algorithms. We also present the OpenAI basic framework (31) which have been used to create the agents.

4.1 Reinforcement Learning Algorithms

OpenAI Baseline is a collection of high-quality Python implementations of reinforcement learning algorithms that use the TensorFlow framework (32). Thanks to the Open AI's RL implementation algorithms code [39], the system may use DQN, DDQN, Prioritized Experience Replay, and Dueling Network. Their DQN implementation and variants are roughly in line with published paper scores. We have used pettingzoo library to create a wrapper that can be used to incorporate single agent RL algorithms on multi-agent domains.

4.1.1 TrafficSignal Environment

TrafficSignal class represents a Traffic Signal of an intersection. It is responsible for retrieving information and changing the traffic phase using Traci API.
The methods provided by TrafficSignal class are as follows:

- build_phase(self): This generates the logic for setting the green, yellow and the red signal.
- set_next_phase(self, new_phase): Sets what will be the next green phase and sets yellow if the next phase is different from the current phase.

- compute_observations(self): It will get observations for the next step from the density and queue.
- compute_reward(self): It will get the waiting time reward.
- waiting_time_reward(self): It will get accumulated waiting time.

4.1.2 SUMO Environment

It is the class for gym environment in Traffic Signal Control.

- step(self, action): It will compute the observations and rewards.
- apply_actions(self, actions): It will set the next green phase for the traffic signals.
- encode(self, state, ts_id): It encodes different phases and density of traffic vehicles.

4.1.3 DQNAgent

DQNAgent interacts with the environment using the SUMO Environment class. The DQNAgent requires the following parameters:

- Policy
- Environment
- discount factor γ
- learning rate α
- simulation time st
- minibatch size B.
- replay buffer size M.
- learning frequency λ

4.1.4 A2CAgent

A2CAgent interacts with the environment using the SUMO Environment class. The A2CAgent requires the following parameters:

- Policy
- Environment
- discount factor γ
- learning rate α
- learning frequency λ
- step size

4.1.5 PPOAgent

PPOAgent interacts with the environment using the SUMO Environment class. The PPOAgent requires the following parameters:

- Policy
- Environment
- discount factor γ
- learning rate α
- learning frequency λ
- step size
- number of epochs
- batch size
- clip range

4.2 Summary

In this section we have discussed the OpenAI Baseline code that we utilized to create RL agents. We developed a framework to enable traffic light agents capable of communicating with the traffic simulator SUMO.

5 Evaluation

This section evaluates different RL algorithms for single-agent and multi-agent environments for traffic light control. We describe the metrics for the evaluation as average accumulation waiting time with the number of steps and analyse their outcomes.

5.1 Single Agent Case

In the single-agent case there is a single intersection where we have placed our agent, and we allow traffic to flow from different direction, and then we compare different algorithms on the basis of their waiting time.

5.1.1 Random Agent

We first run random agent on the single-agent case. Random agent take random actions without knowing anything about the traffic. From the figure 5.1 we can see that the random agent does not learn anything at all.

5.1.2 Fixed Agent

Fixed agent allocates fixed amount of time for all the sides of the road. Most traffic lights used in the world work on this principle. Fixed agent is shown in Fig 5.2.

5.1.3 Single agent (Fixed vs Random)

In the case of a single agent in a single intersection, when we compare random with a fixed agent, we find that the fixed agent has much better performance than a random agent as shown in Fig 5.3.

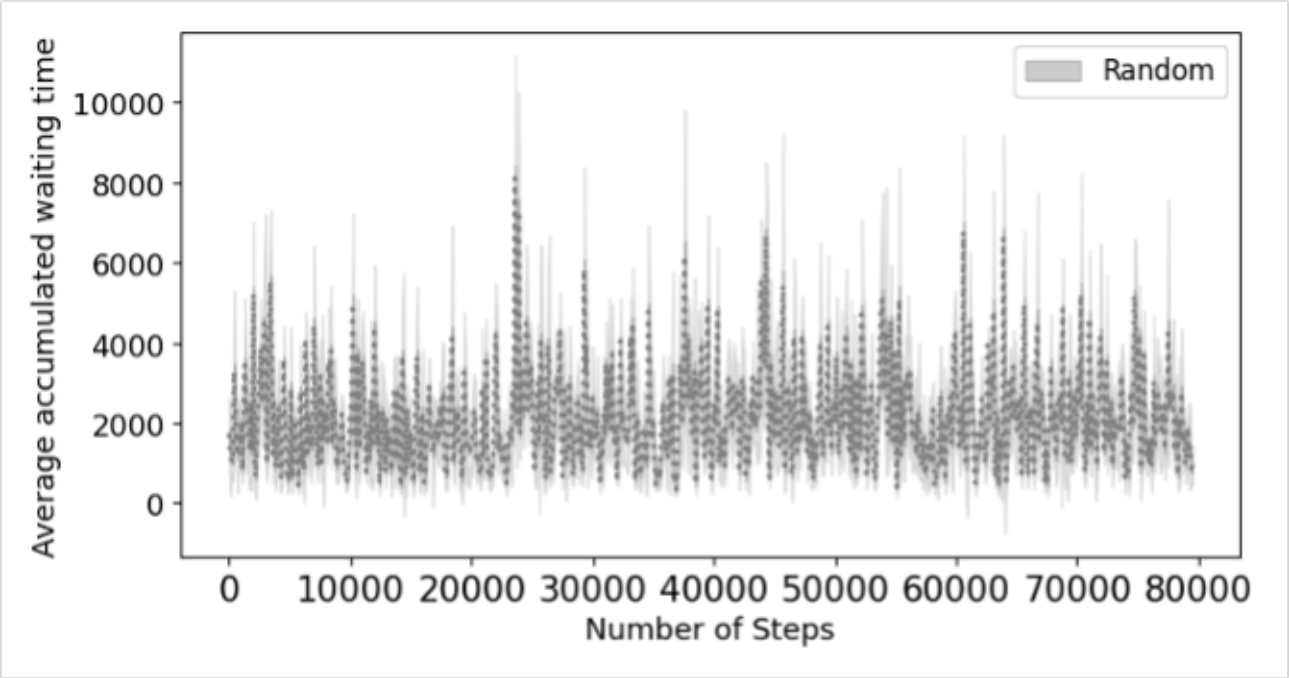


Figure 5.1: Random agent case

5.1.4 Deep Q-Networks

From the Fig 4.3 we can see that DQN is struggling to make an AI agent to solve this task. In an idle case the learning curve which is the waiting time should decrease with time i.e, the agent should learn from the environment and make improvements, but this is not the case in DQN. Since, we know that DQN learns deterministic policies which may not perform in all cases as shown in Fig 5.4.

5.1.5 Actor Critic Method (A2C)

Actor Critic Method (A2C) is a state-of-the art RL algorithm. From the Fig 4.4 we can see that it almost reaches zero waiting time which means there are no vehicles in the queue that are waiting, and the agent actually learns how to control the flow of traffic. We can see that initially the waiting time was very high, but then it decreased after some steps which shows that the agent learns from the environment as shown in Fig 5.5.

5.1.6 Proximal Policy Optimisation (PPO)

Proximal Policy Optimisation (PPO) is a state-of-the art RL algorithm. From Fig 4.5 we can see that it almost reaches zero waiting time which means there are no vehicles in the queue that are waiting, and the agent actually learns how to control the flow of traffic. We can see

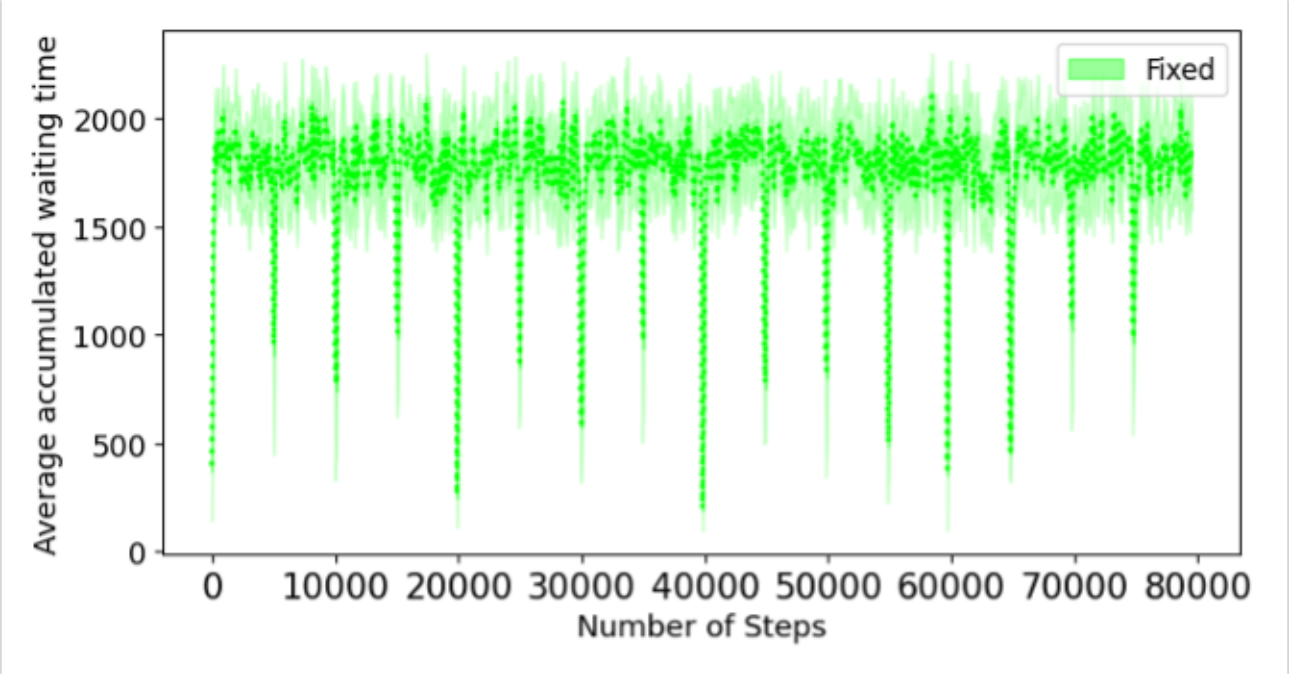


Figure 5.2: Fixed time case

that initially the waiting time was very high, but then it decreased after some steps which shows that the agent learns from the environment as shown in Fig 5.6.
PPO shows nearly identical results like A2C because the SUMO environment that we have considered is very simple with a single agent and intersection.

5.1.7 Single agent (Fixed vs Random vs A2C vs PPO)

In the case of a single-agent in a single intersection, when we compare random, fixed, A2C and PPO, as shown in Fig 5.7. We find that the curves of A2C and PPO nearly coincide. Fixed agent gives average results, and random agent performs the worst among all. Even though PPO is a better algorithm, the performance of A2C and PPO is nearly similar because the environment is very simple.

5.2 Multi-Agent Case

In the multi-agent case there is a double intersection where we have placed two agents and each agent is controlling its own intersection. We allow traffic to flow from different direction, and then we compare different algorithms on the basis of their waiting time. For multi-agent we use pettingzoo library to first create a wrapper that can be used to

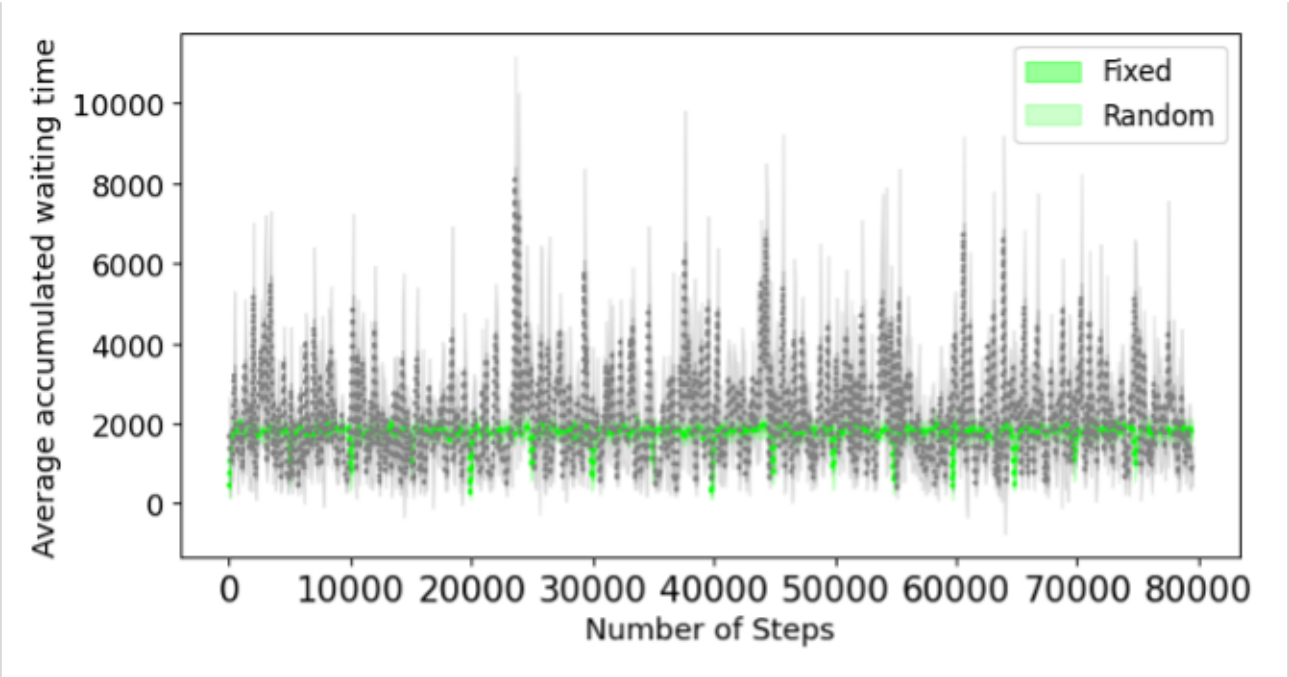


Figure 5.3: Fixed vs Random agent

incorporate single agent RL algorithms on multi-agent domains.

5.2.1 Multi-Agent Random (MADQN)

In the multi-agent random case we see from the Fig 5.8 that it does not learn anything and the waiting time is very poor even after many iterations.

5.2.2 Multi-Agent Fixed (MAFixed)

In the multi-agent fixed time case we see from the Fig 5.9 that it performs better than the random agent, DQN but worse than A2C and PPO. The average accumulated waiting time in this case is better than random but worse than A2C and PPO.

5.2.3 Multi-Agent Deep Q-Networks (MADQN)

In the multi-agent DQN case we see from Fig 5.10 that it fails to learn from the environment. Even after a significant number of steps the average accumulated waiting time is very high. Among all the algorithms used it performs the worst.

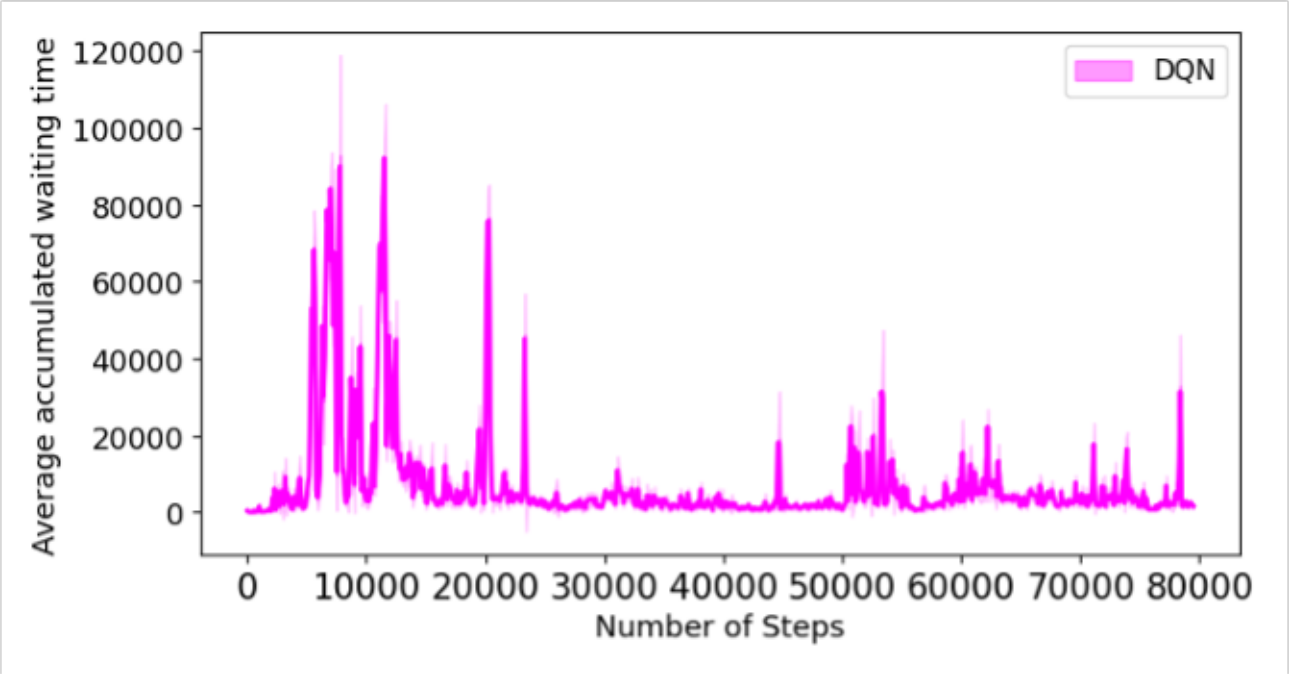


Figure 5.4: Deep Q-Networks

5.2.4 Multi-Agent Actor Critic Method (MAA2C)

In the multi-agent Actor Critic Method (MAA2C) we find that it performs better than all the algorithms discussed before. Initially the average accumulated waiting time is more but then as the number of steps is increased it learns from the environment and the waiting time decreases significantly as shown in Fig 5.11.

5.2.5 Multi-Agent Proximal Policy Optimisation (MAPPO)

In the Multi-Agent Proximal Policy Optimisation (MAPPO) we find that it performs better than any other algorithm discussed. Initially the average accumulated waiting time is more but then as the number of steps is increased it learns from the environment and the waiting time decreases significantly as shown in Fig 5.12. It even outperforms A2C in complex environments.

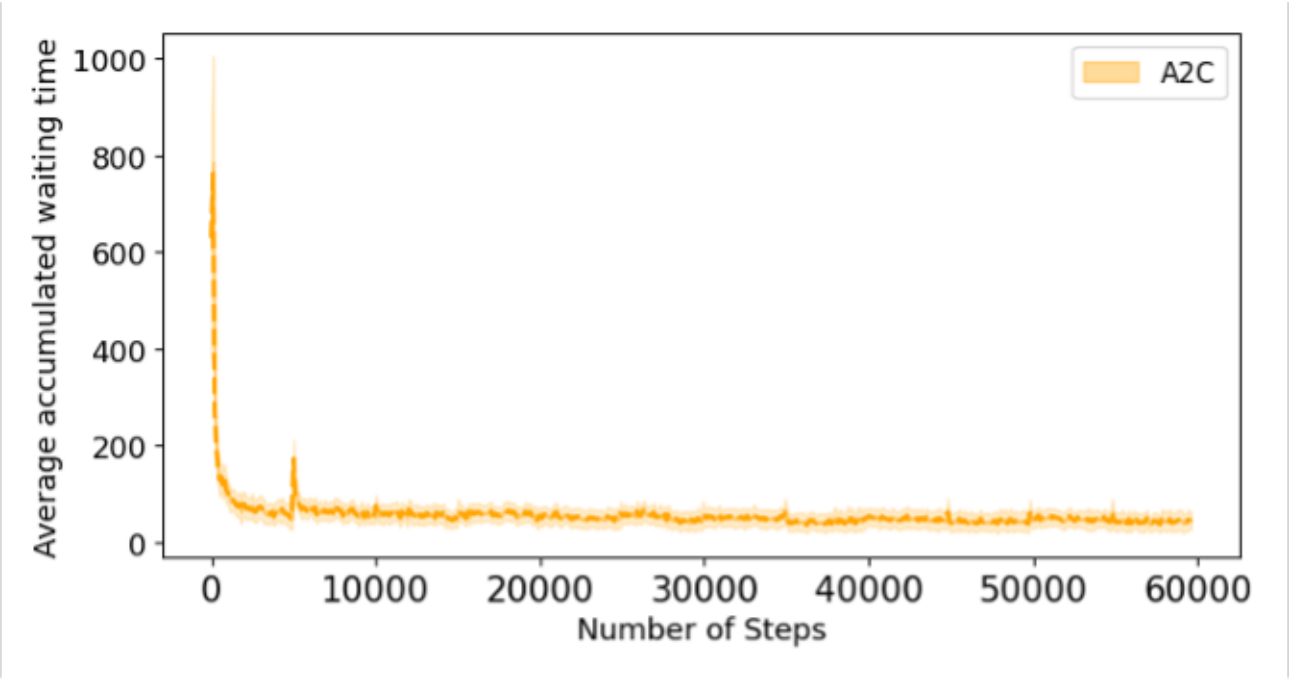


Figure 5.5: A2C algorithm

5.2.6 Multi agent (MAFixed vs MARandom vs MAA2C vs MAPPO)

In the case of a multi-agent in a double intersection, when we compare closely the curves of random, fixed, A2C and PPO, as shown in Fig 5.13, we find that PPO performs the best followed by A2C, fixed and random agent. PPO and A2C can both solve this task with sufficient training they are able to control these two signals.

5.3 Realistic Case

The environments discussed till now consists only of a single intersection and double intersection with uniform traffic. But now we will consider a more realistic situation where in a single intersection there is varying traffic in different time of the day shown in Fig 5.14. The probability of traffic is 0.8 during mornings then it decreases to 0.2 in the afternoon, it is the maximum during the evening at 0.9 and then it drops to the lowest at night to 0.1.

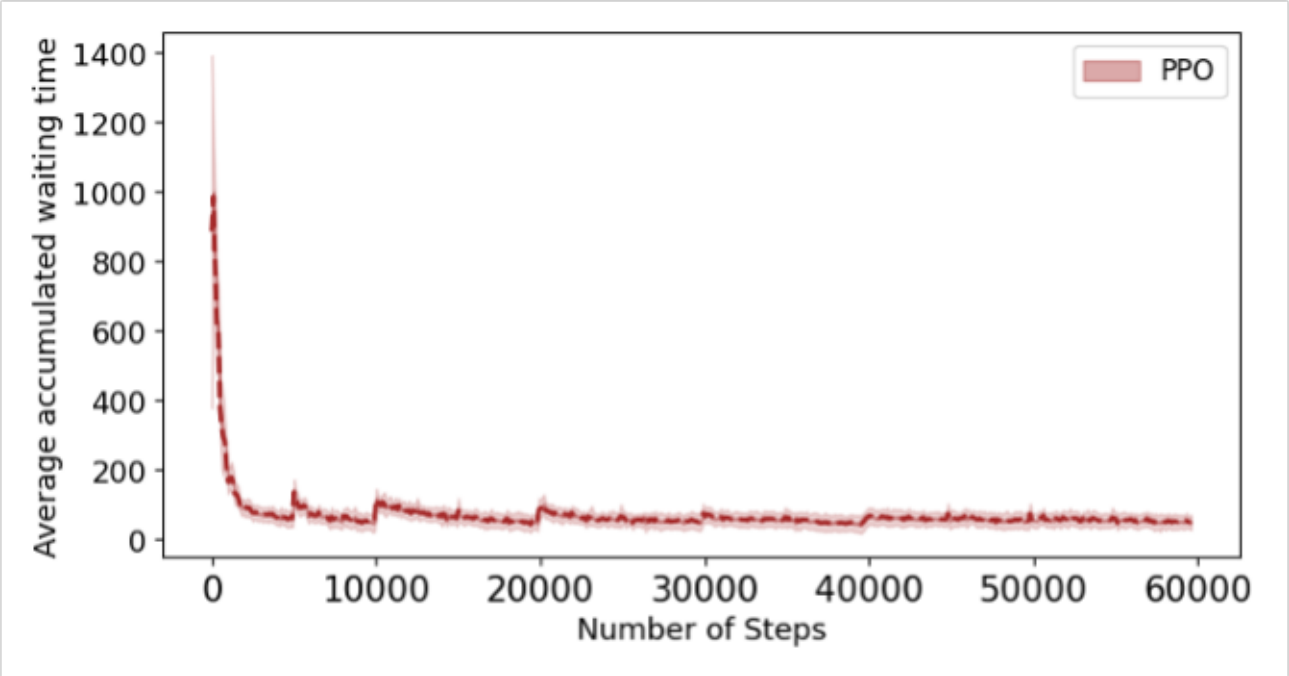


Figure 5.6: PPO algorithm

5.3.1 Random Agent

When we use the random agent in this more realistic case the average accumulated waiting time is very high as expected and the agent does not learn anything even after number of steps as shown in Fig 5.15.

5.3.2 Fixed Agent

When we use fixed agent in this more realistic case the average accumulated waiting time comes out to be better than that of random and DQN, but the agent does not learn anything even after number of steps as shown in Fig 5.16.

5.3.3 Deep Q-Network Agent

When we use DQN in this more realistic case we find the average accumulated waiting time to be the worst. The agent does not learn anything even after a number of steps as shown in Fig 5.17.

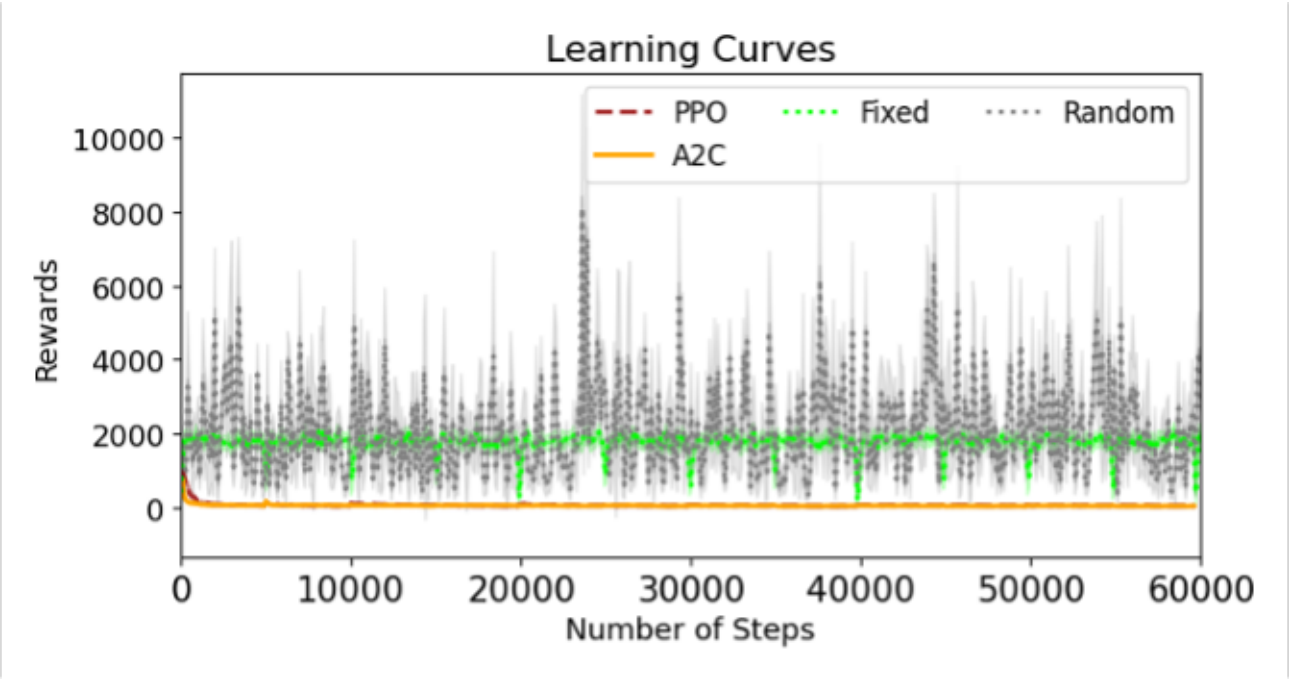


Figure 5.7: Fixed vs Random agent

5.3.4 A2C Agent

When we use A2C in this more realistic case we find that the average accumulated waiting time to be better than all the other algorithms. The waiting time decreases significantly after a number of steps which shows that the agent is learning from the environment. The slope in the graph is decreasing which shows that the agent is learning, and it is getting better in taking decisions as shown in Fig 5.18.

5.3.5 PPO Agent

When we use PPO in this more realistic case we find that the average accumulated waiting time to be the best among all the other algorithms. The waiting time decreases significantly after a number of steps which shows that the agent is learning from the environment. The slope in the graph keeps on decreasing which shows that the agent is learning, and it is getting better in taking decisions as shown in Fig 5.19.

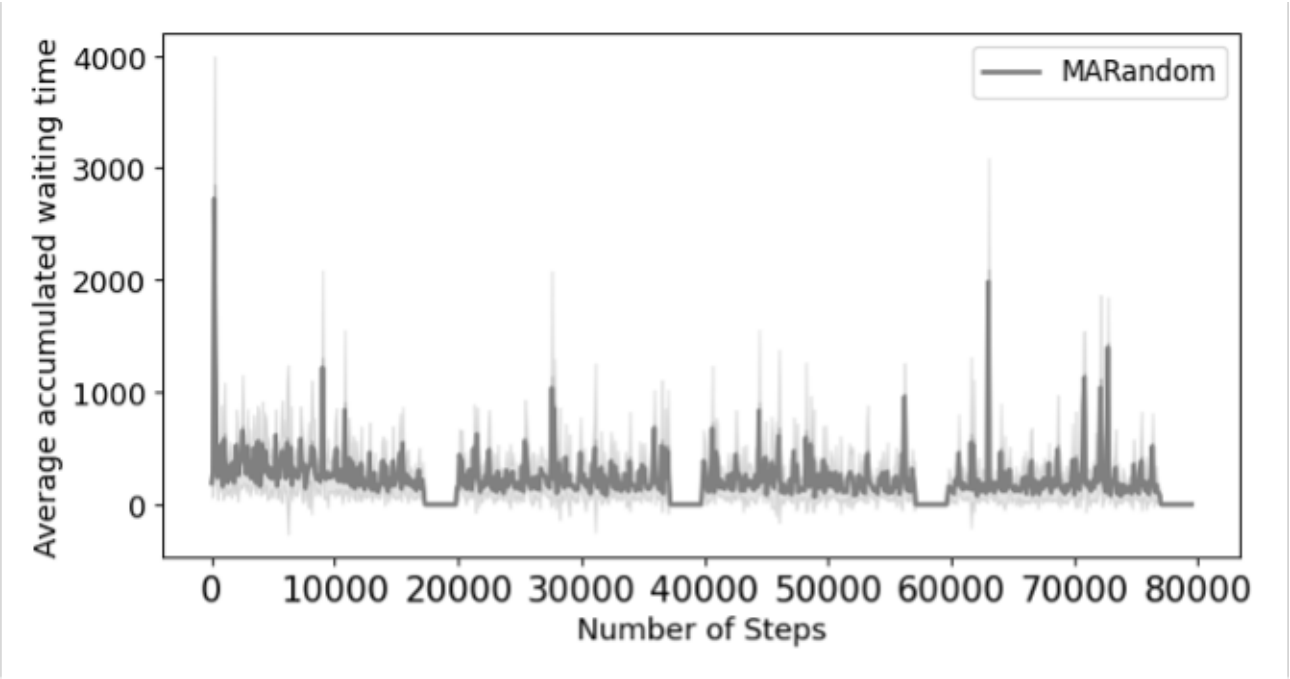


Figure 5.8: Random agent

5.3.6 Single agent realistic case (Fixed vs Random vs A2C vs PPO)

In the case of a single agent with a single intersection, in a more realistic case, we can clearly find out that PPO outperforms every other algorithm followed by A2C, fixed and random agents as shown in Fig 5.20.

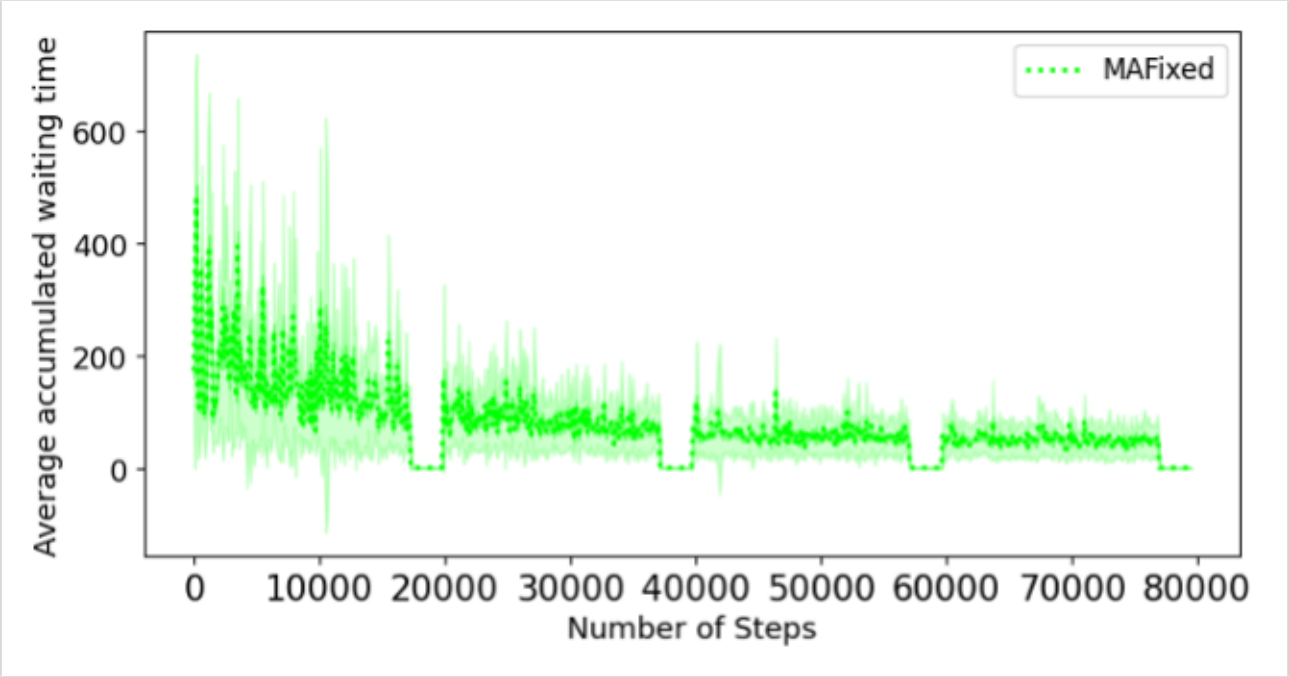


Figure 5.9: Fixed agent

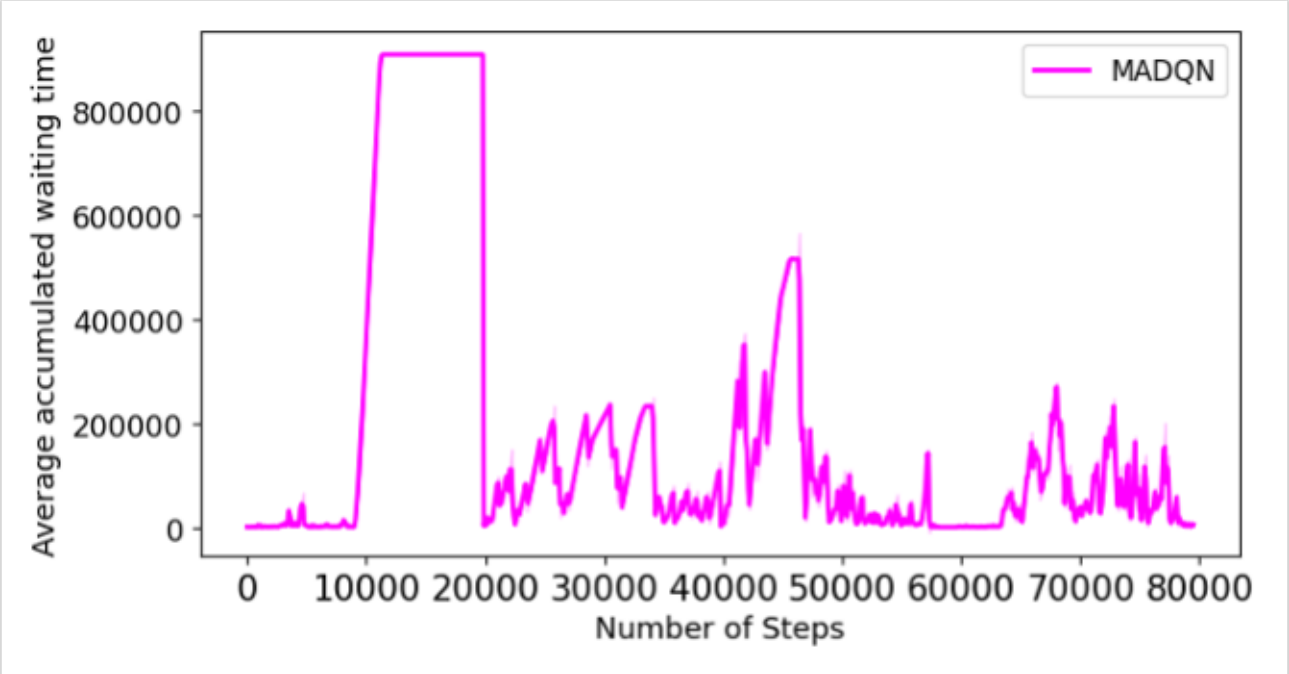


Figure 5.10: DQN algorithm

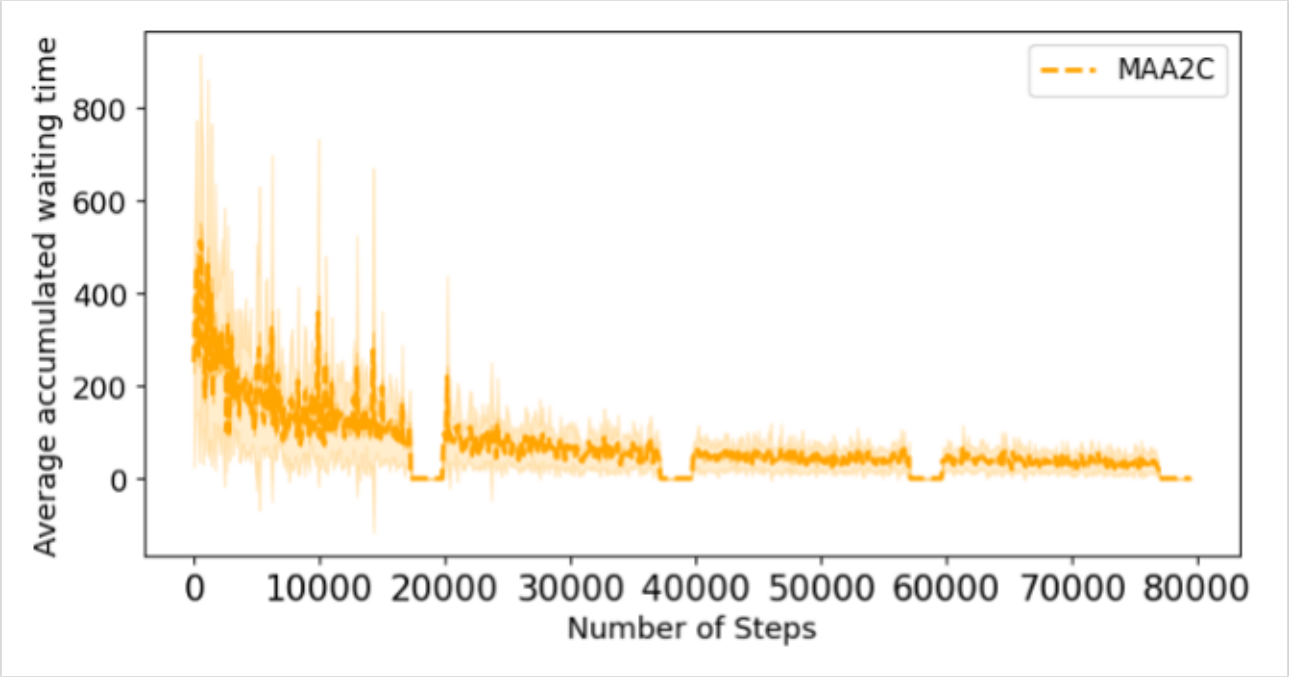


Figure 5.11: A2C algorithm

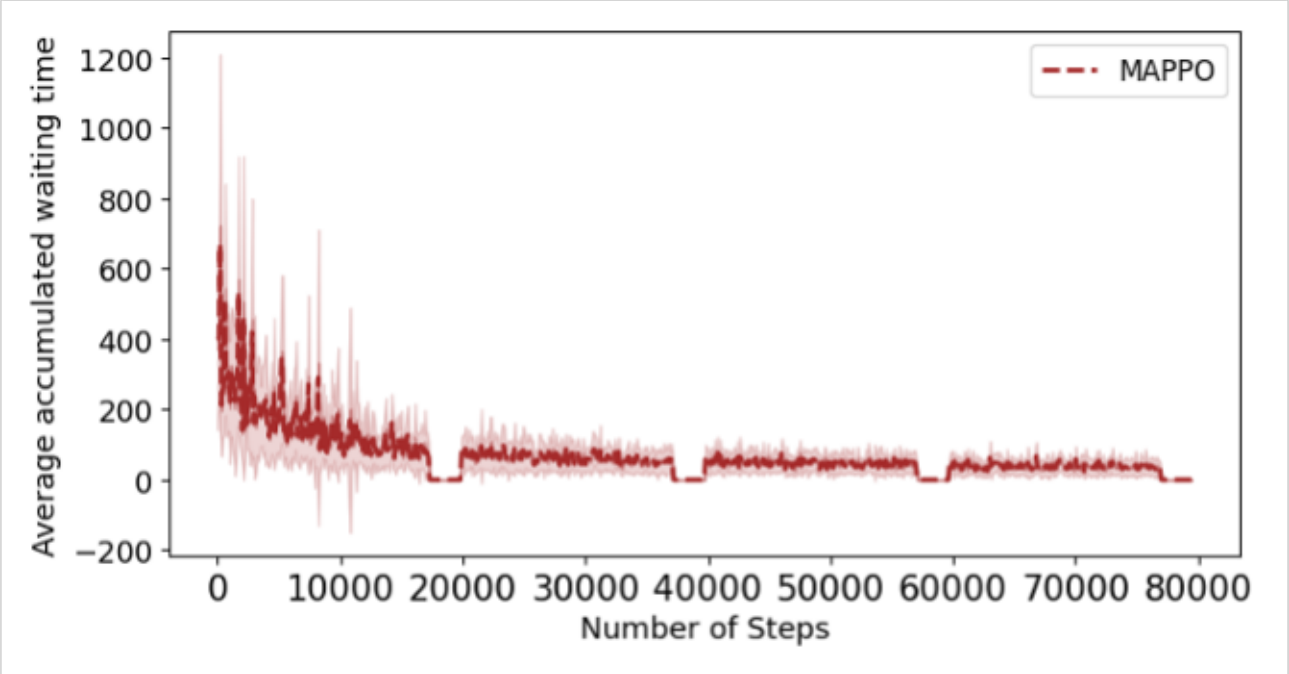


Figure 5.12: PPO algorithm

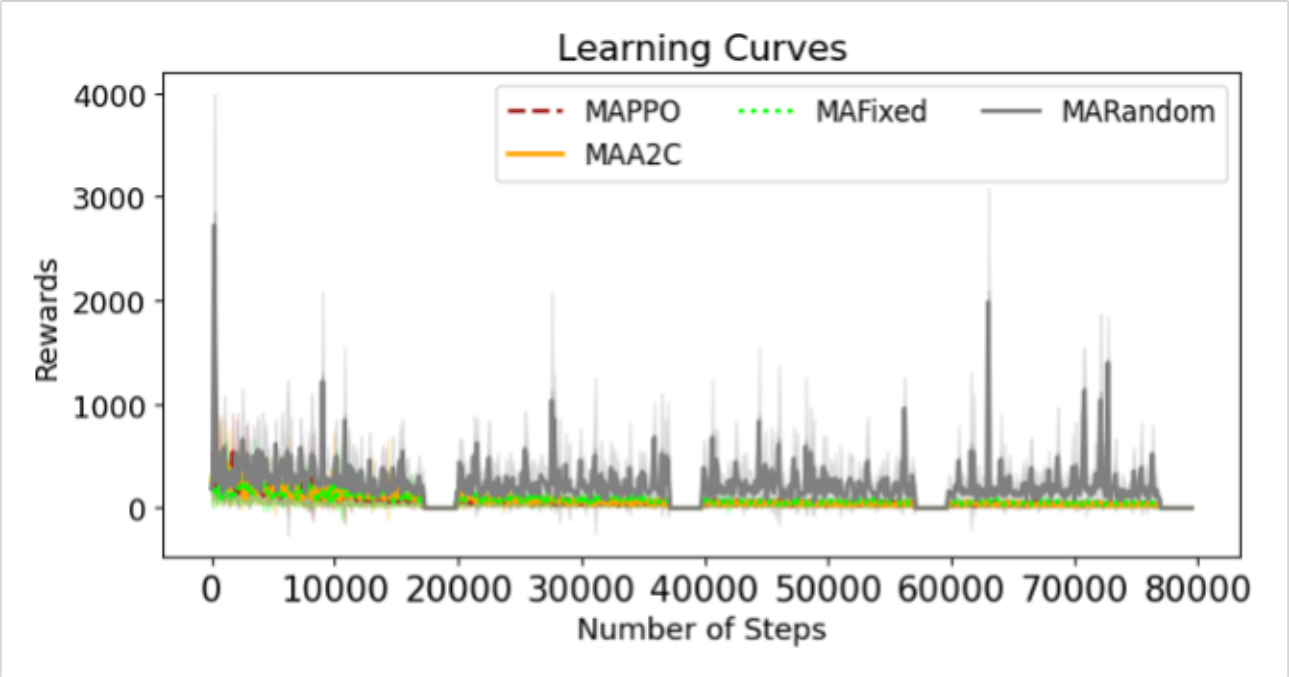


Figure 5.13: Learning curves

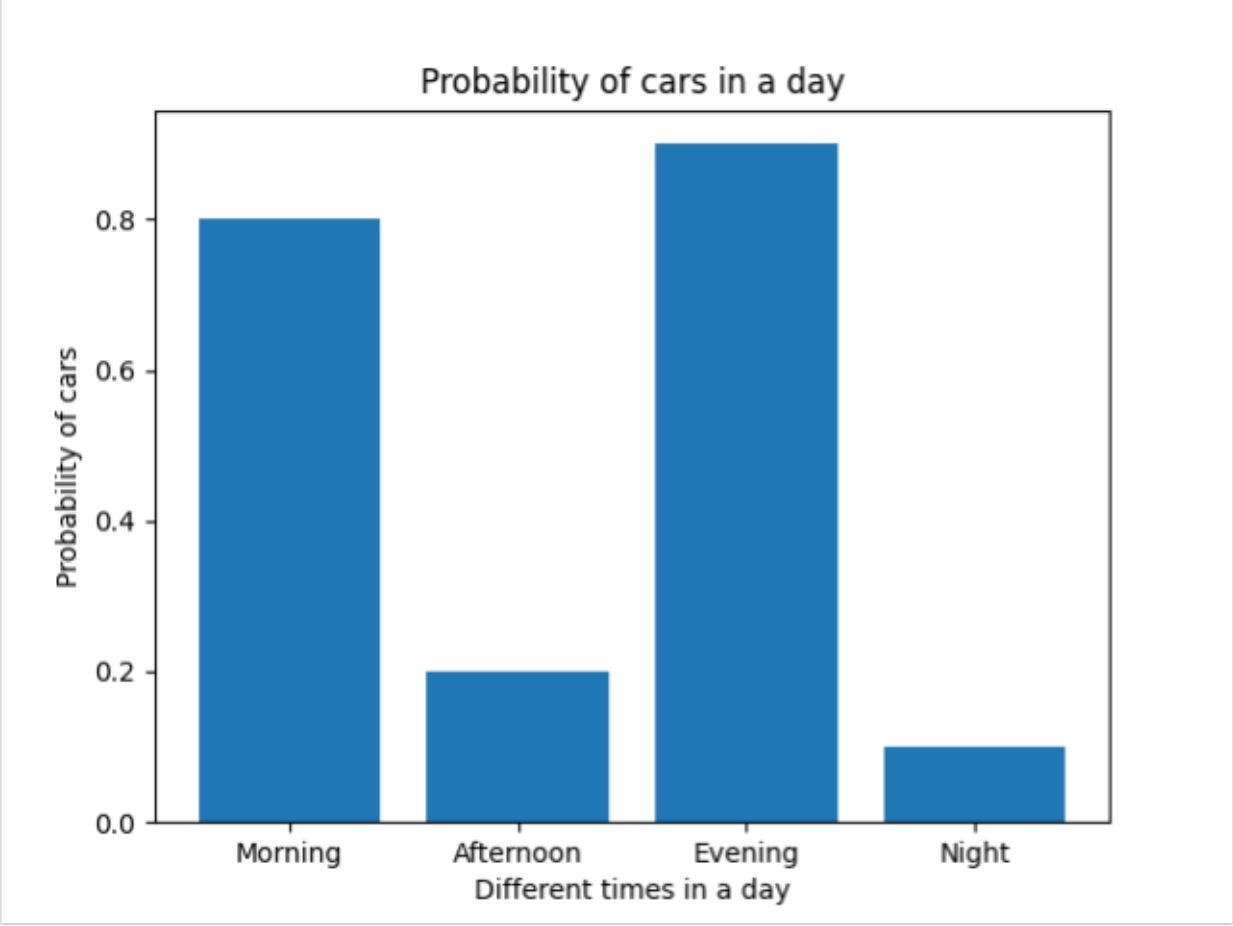


Figure 5.14: Probability of cars in a day

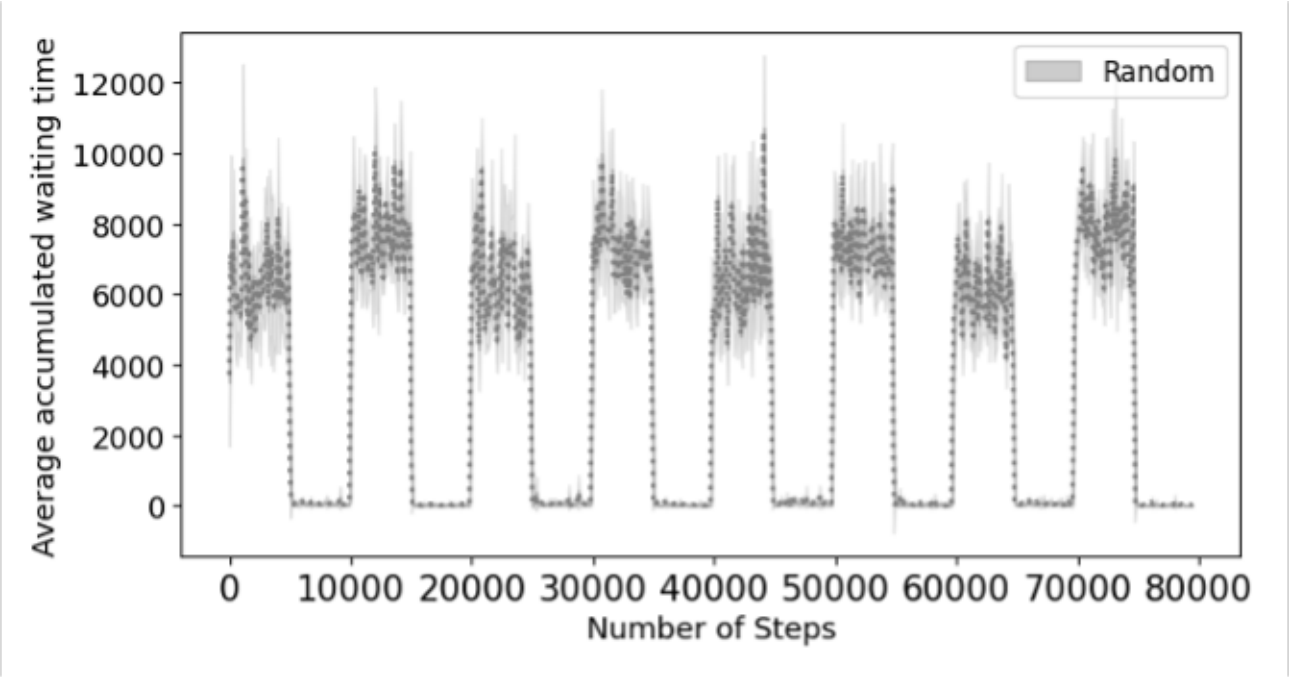


Figure 5.15: Random agent

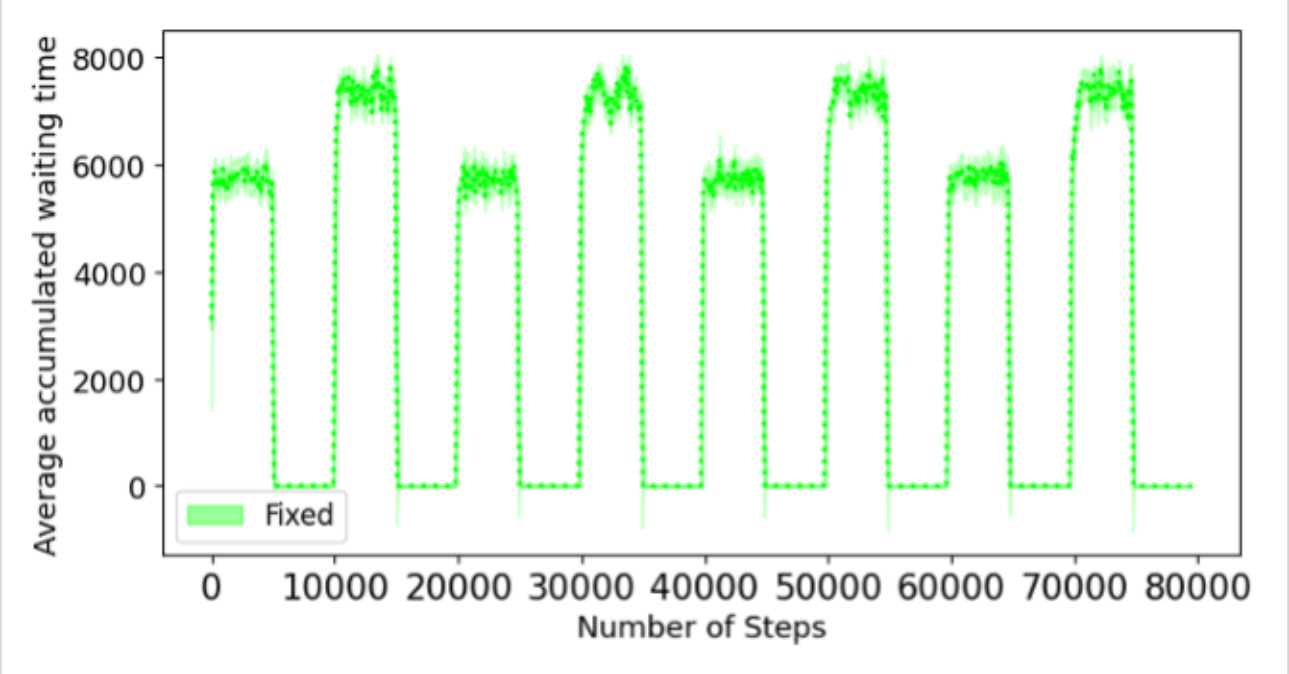


Figure 5.16: Fixed agent

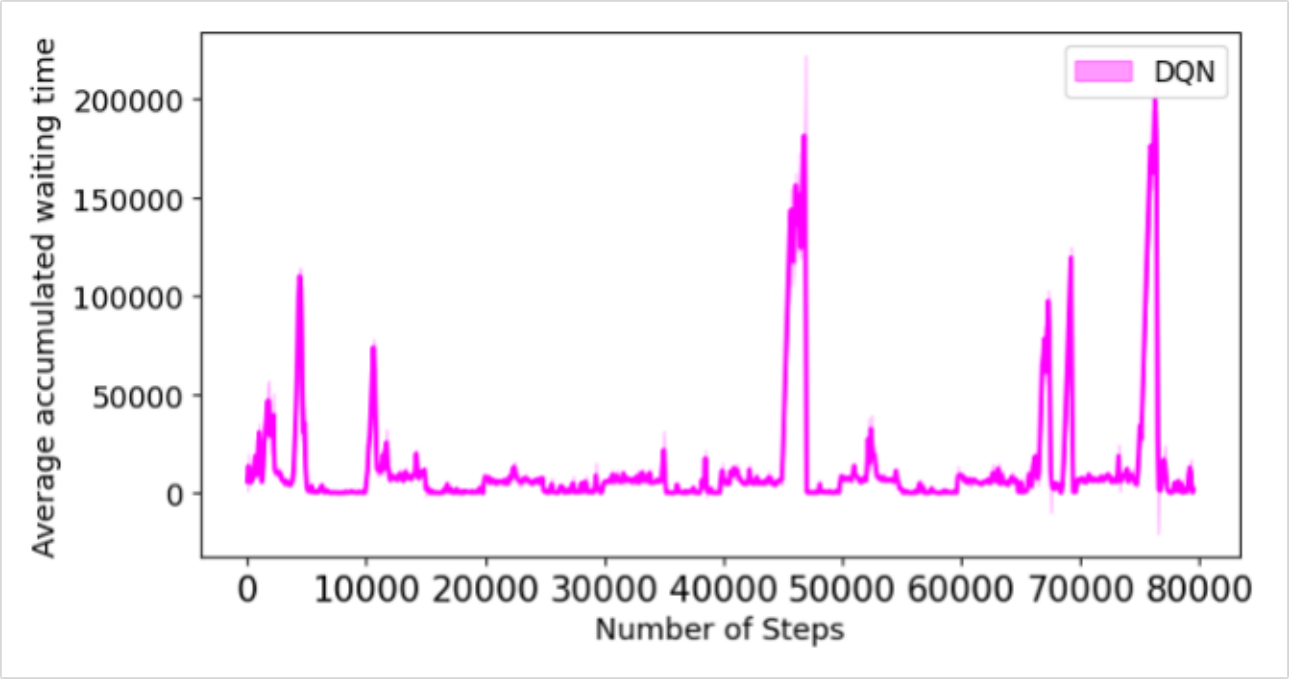


Figure 5.17: DQN agent

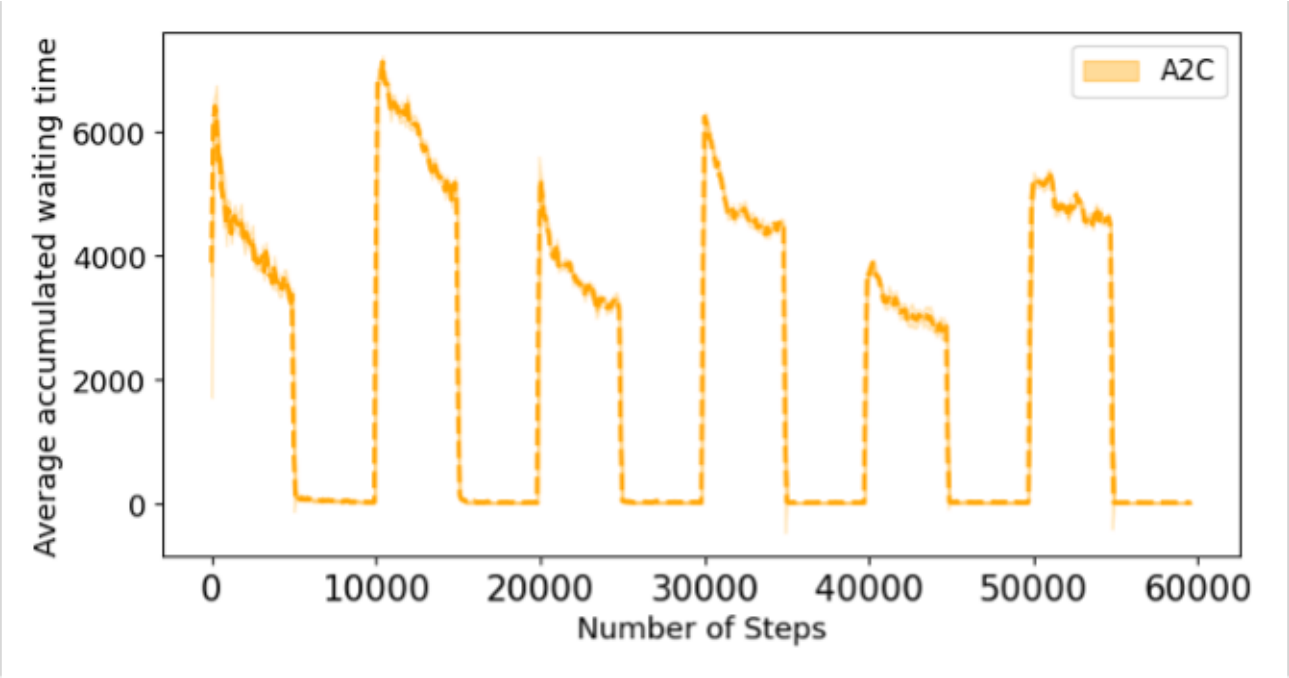


Figure 5.18: A2C agent

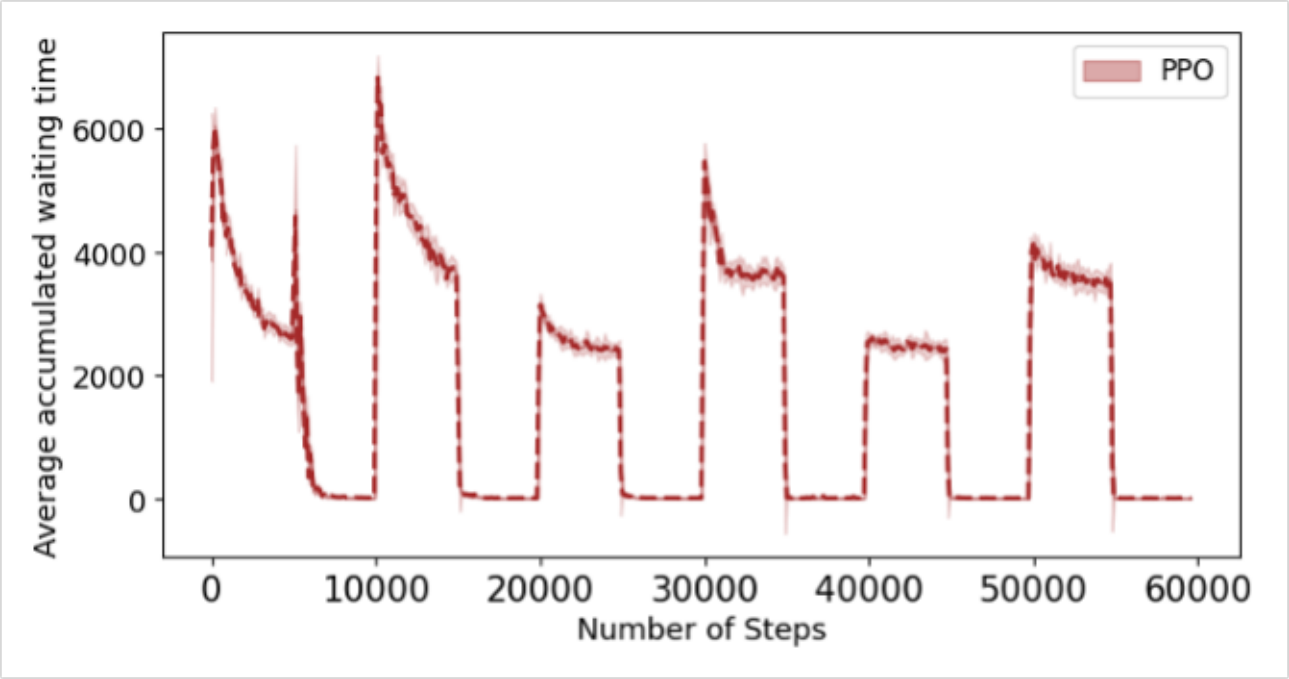


Figure 5.19: PPO agent

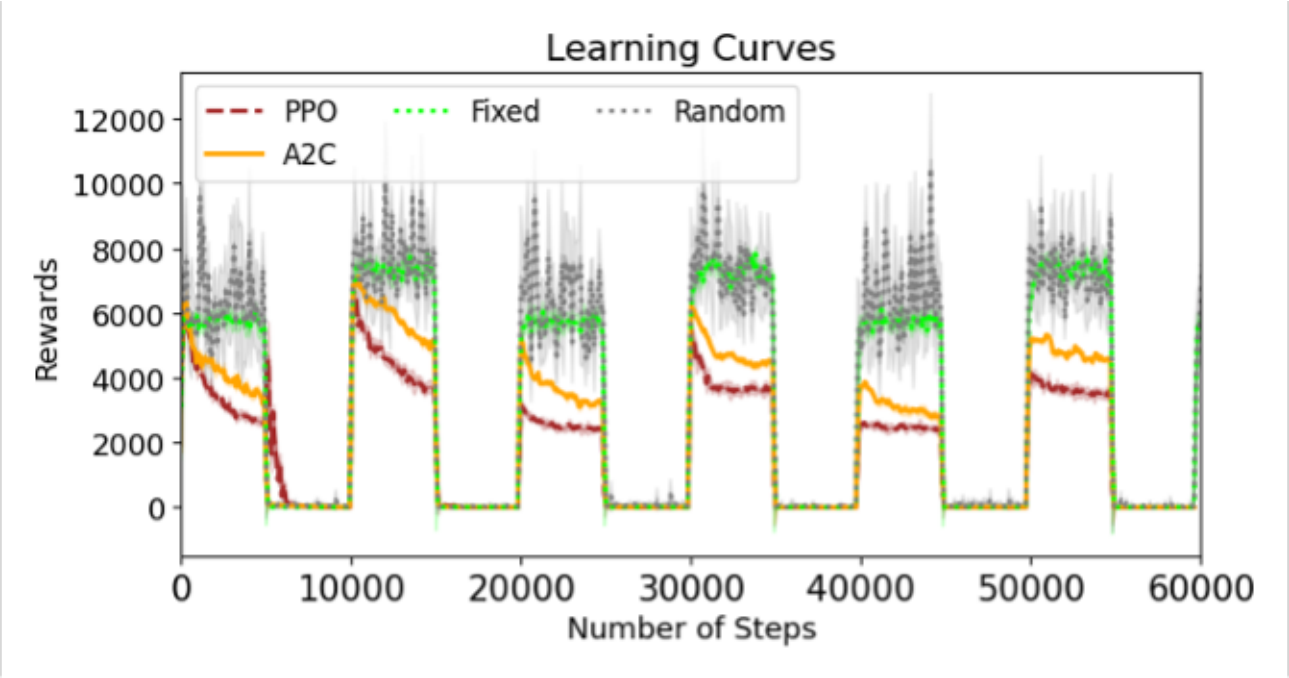


Figure 5.20: Learning curves

6 Conclusion

The proposed model is a sketch of the main issue of transportation congestion in urban cities, but there are numerous other factors which are involved, too, which are ignored in the scope of this thesis. The thesis focuses on a few traffic congestion parameters. However, there are many more that should be included in the model and simulation for a better result. So far, this thesis has focused on reinforcement learning, but accuracy can be improved by using hierarchical reinforcement learning, which is breaking down the final reward in an RL in sub rewards and achieving them one at a time.

6.1 Scope and Limitation

This study has a few disadvantages in addition to its benefits. The most challenging task is to perform the principal component analysis with accurate data. Instead, this research will be analysed with fictional data from the SUMO environment via the TraCI API. The most common issue is the lack of/difficulty obtaining accurate data, which results in an inability to conduct meaningful analysis and make informed decisions. Also, the simulations generated were fundamental. Pedestrians would be challenging to implement in the real world with multiple agents and many junctions with taking bicycles. Also, generating simulations and results takes a lot of processing time. So, it is a very great difficulty.

6.2 Future Implementations

We could have used different reinforcement learning algorithm in this project such as hierarchical reinforcement learning. It is a state of the earth algorithm which decomposes the reward in smaller steps. So the goal of the agent is to achieve different sub-goals rather than focusing on a single goal. Hierarchical RL divide the clustered higher level environment into smaller spaces and making easier the optimization problems that use a divide-and-conquer strategy to deal with it.
We intend to simulate the possibility of road accidents in the future, as traffic congestion which can also regulated by traffic signal controllers utilizing existing simulation methods.

Now, our existing simulation technology can govern traffic lights by simulating an environment artificially, allowing it to react to unexpected situations.
The final goal is to implement this RL approach in a real-world traffic system, test it with the appropriate sensor capabilities (through road loops, cameras, and automobile communication), and set it up with the most dynamic output.

Bibliography

[1] R. R. Mouly, P. R. Rini, A. H. Ethic, and M. I. Ayon, "Traffic congestion reduction in sumo using reinforcement learning method," Ph.D. dissertation, Brac University, 2021.

[2] B. S. Meghana, S. Kumari, and T. P. Pushphavathi, "Comprehensive traffic management system: Real-time traffic data analysis using rfid," in *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, vol. 2, 2017, pp. 168–171.

[3] N. Y. Hamisi, N. H. Mvungi, D. A. Mfinanga, and B. M. M. Mwinyiwiwa, "Prospects of pre-timed arterial traffic control systems in city roads of a typical developing country: a case study of dar es salaam city," in *2009 2nd International Conference on Adaptive Science Technology (ICAST)*, 2009, pp. 339–346.

[4] H. Taale, "Comparing methods to optimise vehicle actuated signal control," in *Eleventh International Conference on Road Transport Information and Control, 2002. (Conf. Publ. No. 486)*, 2002, pp. 114–119.

[5] Y. Feng, K. L. Head, S. Khoshmagham, and M. Zamanipour, "A real-time adaptive signal control in a connected vehicle environment," *Transportation Research Part C: Emerging Technologies*, vol. 55, pp. 460–473, 2015.

[6] S. Zhancheng, "Research on application of deep reinforcement learning in traffic signal control," in *2021 6th International Conference on Frontiers of Signal Processing (ICFSP)*, 2021, pp. 17–21.

[7] Q. Huang, "Model-based or model-free, a review of approaches in reinforcement learning," in *2020 International Conference on Computing and Data Science (CDS)*, 2020, pp. 219–221.

[8] M. N. Moghadasi, A. T. Haghighat, and S. S. Ghidary, "Evaluating markov decision process as a model for decision making under uncertainty environment," in *2007 International Conference on Machine Learning and Cybernetics*, vol. 5, 2007, pp. 2446–2450.

[9] C. J. Watkins and P. Dayan, "\cal q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[12] G. Lou and H. Shi, "Face image recognition based on convolutional neural network," *China Communications*, vol. 17, no. 2, pp. 117–124, 2020.

[13] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[14] Y. Lin, X. Dai, L. Li, and F.-Y. Wang, "An efficient deep reinforcement learning model for urban traffic control," *arXiv preprint arXiv:1808.01876*, 2018.

[15] M. Guo, P. Wang, C.-Y. Chan, and S. Askary, "A reinforcement learning approach for intelligent traffic signal control at urban intersections," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 4242–4247.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[17] L. Li, Y. Lv, and F.-Y. Wang, "Traffic signal timing via deep reinforcement learning," *IEEE/CAA Journal of Automatica Sinica*, vol. 3, no. 3, pp. 247–254, 2016.

[18] E. Van der Pol and F. A. Oliehoek, "Coordinated deep reinforcement learners for traffic light control," *Proceedings of learning, inference and control of multi-agent systems (at NIPS 2016)*, 2016.

[19] W. Genders and S. Razavi, "Using a deep reinforcement learning agent for traffic signal control," *arXiv preprint arXiv:1611.01142*, 2016.

[20] J. Liu, S. Qin, Y. Luo, Y. Wang, and S. Yang, "Intelligent traffic light control by exploring strategies in an optimised space of deep q-learning," *IEEE Transactions on Vehicular Technology*, pp. 1–1, 2022.

[21] K. Zhang, Z. Yang, and T. Basar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *CoRR*, vol. abs/1911.10635, 2019. [Online]. Available: http://arxiv.org/abs/1911.10635

[22] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–359, 10 2017.

[23] L. Canese, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Re, and S. Spanò, "Multi-agent reinforcement learning: A review of challenges and applications," *Applied Sciences*, vol. 11, no. 11, p. 4948, 2021.

[24] P. Wu, B. Song, X. Chen, and B. Liu, "A traffic light control system based on reinforcement learning and adaptive timing," in *International Conference on Neural Computing for Advanced Applications*. Springer, 2021, pp. 545–559.

[25] Y. Zhu, M. Cai, C. Schwarz, J. Li, and S. Xiao, "Intelligent traffic light via policy-based deep reinforcement learning," *arXiv preprint arXiv:2112.13817*, 2021.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[27] D. Krajzewicz, G. Hertkorn, C. Rössel, and P. Wagner, "Sumo (simulation of urban mobility)-an open-source traffic simulation," in *Proceedings of the 4th middle East Symposium on Simulation and Modelling (MESM20002)*, 2002, pp. 183–187.

[28] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. [Online]. Available: https://elib.dlr.de/124092/

[29] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wiessner, "Microscopic traffic simulation using sumo," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2575–2582.

[30] M. Gregurić, M. Vujić, C. Alexopoulos, and M. Miletić, "Application of deep reinforcement learning in traffic signal control: An overview and impact of open traffic data," *Applied Sciences*, vol. 10, no. 11, p. 4011, 2020.

[31] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," 2017.

[32] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "{TensorFlow}: A system for {Large-Scale} machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.