Multi Agent Deep Recurrent Q-Learning for Different Traffic Congestion

Azlaan Mustafa Samad

Multi Agent Deep Recurrent Q Learning for Different Traffic Congestion Scenarios

by

Azlaan Mustafa Samad

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday January 31, 2020 at 03:00 PM.

Student number: 4962702

Thesis committee:	Dr. ir. F. Oliehoek,	TU Delft, Daily Supervisor
	Prof. dr. ir. C. Vuik,	TU Delft, Chair Professor
	Prof. dr. ir. A. W. Heemink,	TU Delft, Full Professor
	A. Czechowski	TU Delft, Interactive Intelligence Group Member

Contents

Abbreviations and Symbols

Abbreviations

RL:	Reinforcement Learning
MDP:	Markov Decision Process
MAS:	Multi Agent System
TP:	Transfer Planning
DRQN:	Deep Recurrent Q-Network
ReLU:	Rectified Linear Unit
CNN:	Convolution Neural Network
ANN:	Artificial Neural Network
MBGD:	Mini-Batch Gradient Descent
SGD:	Stochastic Gradient Descent
ADAM:	Adaptive Moment Estimation
TLC:	Traffic Light Control
ER:	Experience Replay

Symbols

$v_\pi(s)$:	Value function
$q_\pi(s,a)$:	Q-function or Action-Value function
γ:	Discount factor
α:	Learning Rate
ϵ:	Exploration Rate
τ:	History Size
$\mu_{ij}(a_j)$:	Message passing parameter
$R(\mathbf{a})$:	Global Payoff function
$\vec{a^*}$:	Optimal Joint Action
$f_{ij}(a_i,a_j)$:	Factored Global Payoff Function
Q_{θ^0}:	Primary Q-Network
Q_{θ^-}:	Target Q-Network

1 Introduction

Currently the world is undergrowing rapid urbanisation due to the influx of more and more people into the urban areas from rural hinterlands, especially in developing countries. In order to keep up with the rapid urbanisation, the cities are adopting to new technology in the field of city planning. One of the major affects of this unsustainable rapid urbanisation is the increase in traffic on the roads. Existing traffic light causes numerous issue such as delays, accidents, noise, air pollution and monetary losses. Based on 2012 study by Washington University, it was concluded that long commutes eat up exercise time leading to various health issues like obesity issues, lower fitness levels, higher blood pressure—all strong predictors of heart disease, diabetes [4]. Today's traffic light systems leads to inefficient traffic flow and therefore longer travel times.

In some recent research work [27], Reinforcement Learning (RL) is used in order to control the traffic light by treating it as a sequential decision making problem. In RL, the agent learns from trial and error by interacting with the environment where the goal of the agent is to maximise the reward in the long term. In case of application of RL in traffic flow problems, Markov Decision Process(MDP) is used to model the problem, where the *states* contains the information of the traffic light intersections, *actions* are the different traffic lights configurations, while the *rewards* are formulated by taking into account different factors like waiting time, delays, teleports(specific to SUMO [1] simulator), emergency stops etc.

This thesis is a continuation of previous work [27] [25] [23] [24] wherein Deep Q-Learning is used to find an optimal *policy* which chooses action in order to maximise the total cumulative reward of the state. After learning to optimise the traffic flow for a single intersection(or single agent), the same idea can be extrapolated to Multi Agent System(MAS), where the number of intersection is more than one. This can be done using the idea of transfer planning [17] for training the agent(or traffic intersection) for smaller sub-problems i.e global coordination is decomposed into local coordination using the theory of coordination graphs to find a joint global optimal action using different coordination algorithms.

The aim of the thesis is to extend the previous work for different congestion scenarios for MAS. In order to do this, various congestion scenarios are selected and single and MAS are trained for these scenarios. Finally using the approach of Transfer Planning(TP) along with different Coordination Algorithms various congestion scenarios are studied for MAS.

1.0.1. Contribution

By conducting research as described above, this thesis aims to find answers to the following questions:

- *How does Transfer Planning combined with Maxplus or Brute Coordination algorithm perform when compared to Individual Q-Coordination?*
  This can studied by training agents for single as well as MAS. The MAS can be trained for small source problem which then can be extended to bigger problem by using TP. Finally, comparison can be done on the basis of the reward function and average travel time.

- *How does the agent perform for different traffic congestion for single as well as MAS?*
  Again the above approach can be followed, but first the agents need to be trained for different

traffic congestion scenarios. In order to define congestion, various literature can be studied and conclusion can be drawn on how to choose low, medium or high traffic congestion.

1.0.2. Outline

In the subsequent chapters theoretical concepts necessary to understand MDP, Deep Recurrent Q-Network(DRQN), TP and coordination algorithms is discussed. Further, experimental setup, network architecture, simulation software is discussed for single and MAS. Then experiments results for both the systems. Finally towards the end, results are concluded and discussed along with future work.

2

Background

2.0.1. Reinforcement Learning

Reinforcement Learning is a type of learning in which the states are mapped to actions in order to maximise a numerical reward signal. The action taker is known as the agent. It observes the environment either fully or partially and takes some action in order to land in a different state and receives a reward for taking that particular action. This action not only effects the current reward but also all subsequent rewards. The agent is not told which action to take instead it must learn which action yields most reward through a process of trial and error.

2.0.2. Markov Decision Process

Markov Decision process (MDP) is the mathematical framework of sequential decision making, where actions influence immediate reward, and subsequent future rewards. In this process, the agent observes the environment at each time steps $t = 0, 1, 2, 3\ldots$ in the form of state s_t selects some action a_t. One time step later after taking the action, it receives a numerical reward r_{t+1}, and ends up in state s_{t+1}.

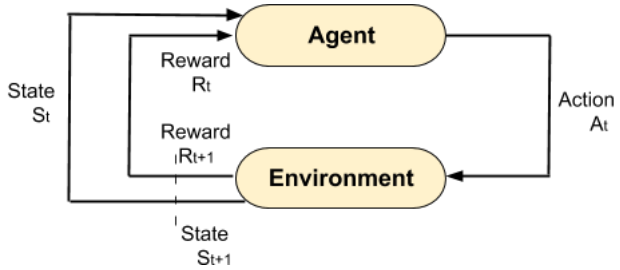

Figure 2.1: The agent–environment interaction in a Markov Decision Process.

MDP is formally represented as:

- $\mathcal{S}$ is the space of possible states;
- $\mathcal{A}$ is the space of possible actions;
- $p(s', r \,|\, s, a) \doteq Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$,
  the transition probability of ending up in state s' and obtaining reward r from previous state s by taking an action a

The agent's goal is to maximize total reward it receives over time, this means maximising not just immediate reward but cumulative reward in the long run. This can be represented as the following:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.1}$$

where γ is the discounted rate such that $0 \leq \gamma \leq 1$.

γ determines the present value of future rewards. If $\gamma = 1$, then every reward is weighed equally, whereas when it equals 0, then the agent is myopic and is concerned only with maximising immediate rewards.

It can also be represented as:

$$G_t \doteq R_{t+1} + \gamma G_{t+1} \tag{2.2}$$

Policy is a mapping from state to probabilities. If an agent follows a policy π at time t, then $\pi(a|s)$ is the probability of taking an action $A_t = a$ when in state $S_t = s$ at time t. In Reinforcement Learning, this policy is updated from experience in order to attain maximum cumulative reward.

Value function is an estimate of how good it is for an agent to be in a particular state, which implies the expected future reward. It is the expected cumulative reward when starting in state s and following a policy π.

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right], \tag{2.3}$$

for all $s \in \mathcal{S}$.

Next we come to action-value function or popularly known as q-function. Action-value function denoted as $q_\pi(s, a)$, is the expected return starting from state s, taking an action a, and thereafter following policy π:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum \gamma^k R_{t+k+1}|S_t = s, A_t = a] \tag{2.4}$$

where $\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \,|s, a) = 1$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$.

In RL, the values of $v_\pi(s)$ and $q_\pi(s, a)$ can be learned from experience. If one maintains the average of return that follow a particular state, for each time the state has been encountered then this value converges to the $V_\pi(s)$, since the number of times the state is encountered is infinity. If average is kept for each action taken when in a particular state, then this will converge to the action-value function $q_\pi(s, a)$.

Bellman Equation: It is a representation of the value of the state, assuming one takes the best possible action now and at each subsequent step. Below, the recursive relationship between the value function of a state with respect to other state is shown. This is also known as the Bellman Equation.

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\,[r + \gamma \mathbb{E}_\pi\,[G_{t+1} \mid S_{t+1} = s']] \\ &= \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a)\,[r + \gamma v_\pi(s')] \end{aligned} \tag{2.5}$$

for all $s \in \mathcal{S}$.

Optimal Policies and Optimal Value Function: Optimal policies are the policies that secure maximum rewards in the long term (cumulative reward). A policy π is better or equal to another policy π' if its expected return is greater than or equal to that of π' for all states $s \in \mathcal{S}$. Therefore, $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$. This policy π is often known as optimal policy, and often written as π^*.

Optimal State-Value Function:

$$v_*(s) \doteq \max_\pi v_\pi(s) \tag{2.6}$$

for all $s \in \mathcal{S}$.

Optimal Action-Value Function:

$$q_*(s,a) \doteq \max_{\pi} q_\pi(s,a) \tag{2.7}$$

for all $s \in \mathcal{S}$ and $A \in \mathcal{A}(s)$.

Its the expected reward when in state s and taking an action a and thereafter, following an optimal policy. Therefore, the optimal action-value function can be represented as a function of the optimal state-value function in the following way:

$$q_*(s,a) = \mathbb{E}[\mathrm{R}_{t+1} + \square * v_*(S_{t+1}) \mid S_t = s, A_t = a] \tag{2.8}$$

Bellman Optimality equation: According to this, the value of a state under an optimal policy must equal the expected return for the best action from that state.

$$\begin{aligned} v_*(s) &\doteq \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)\,[r + \gamma v_*(s')] \end{aligned} \tag{2.9}$$

$$\begin{aligned} q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}), a') \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r \mid s, a)[r + \gamma \max_{a'} q_*(s', a')] \end{aligned} \tag{2.10}$$

The Bellman Equation can be explained better with Figure **??**. This diagram is also called the backup diagram. Here, the open circle represent the state, while the solid circle represent the state-action pair. The agent is in state s, representing the root node at the top and based on a policy, takes a certain action a, then the environment responds and lands the agent in any of the next probable state s' securing a reward r. The Figure **??** represents the backup diagram for the optimality equations.

If the state of the environment is well defined or in other words if the transition probability is known, then dynamic programming methods can be used to find the optimal solution using the recursive definition. One of these method is Value Iteration, in which the value function is updated for all states by updating each q-value and then using the maximum q-value to update the value function.

Explicitly solving the Bellman optimality equation provides one route to finding an optimal policy, and thus to solving the reinforcement learning problem. However, this solution is rarely directly useful. This is so because it is based on the following three assumptions, which are rarely met: the environment is accurately known, Markov Property and sufficient computational power. Therefore, in many cases the transition probability is unknown. Under such scenario, the agent uses RL algorithms, the agent learns a mapping from state to actions from interacting with the environment and receiving feedback.

There are two types of Reinforcement Learning:

- Model-based: Agent samples from the environment to estimate the transition probability, then uses planning algorithm to find an optimal policy.
- Model-Free: Agent directly estimates the state-action value function from experience.

2.0.3. Tabular Q-Learning

Q-learning is a model-free reinforcement learning algorithm. That is, it does not build its own model of the environment's transition functions, but rather directly estimates the Q-value of the state-action pair $q(s,a)$. Specifically, Q-learning is an off-policy algorithm, which is a class of algorithms that uses

a different policy for estimating Q-values than for action-selection. That is, Q-learning updates the Q-values of the current state-action pair using the greedy policy to estimate the Q-value of the optimal policy of the next state-action pair.

In traditional Q-learning, the agent employs a lookup table of state-action pairs and iteratively updates the Q-value estimates using:

$$q_{t+1}(s,a) = q_t(s,a) + \alpha[R_t + \gamma[\max_{a'} q_t(s_{t+1}, a')] - q_t(s,a)] \tag{2.11}$$

In words, the difference between the current estimate of the state-action pair, and the actual value of the (s,a)-p air. However, since the true value of the (s,a)-pair is not known upfront, the agent instead uses the current reward signal and the maximizing Q-value of the next state as a proxy for the true value.This is called tabular Q-learning, and it has the nice property that it converges given infinite samples.

2.0.4. Q learning with Function Approximation

Extending reinforcement learning to function approximation also makes it applicable to partially observable systems, in which the full state is not available to the agent. A solution to the problem of continuous S is function approximation, where supervised machine learning algorithms are used to approximate the Q-function. Q-value is a function parameterised by weight θ. These weights can be updated using gradient descent methods, minimizing the mean squared error between the current estimate of $q(s,a)$ and the target, which is defined as the true Q-value of the (s,a)-pair under policy $q_\pi(s,a)$.

The gradient descent update can be derived by taking the derivative of the mean squared error (MSE):

$$MSE(\theta) = \sum_{s \in \mathcal{S}} P(s) \left[q_\pi(s,a;\theta^*) - q_t(s,a;\theta_t)\right]^2 \tag{2.12}$$

where $P(s)$ is the sampling distribution, or the probability of visiting state s under policy π.

With the q-function approximation represented as a function with learnable parameters, a regular supervised learning method can be used to approximate the true Q-function.

2.0.5. Neural Network

A neural network is a machine learning model parameterised by a set of parameters θ that maps an M-dimensional input vector $\vec{x}$ through a series of hidden layers and activations, to a K-dimensional output vector $\vec{y}$. It is used as a non-linear function approximator. Specifically, a neural network consists of interconnected layers, where each layer computes a linear mapping between the input x and its weights w, adding a bias term b and mapping the result through a non-linear activation function - needed to introduce non-linearity into the model, for example a Rectified Linear Unit(ReLU). For example, mapping input vector $\vec{x}$ through one hidden layer with weights $W_0 \in \theta$, bias term $b_0 \in \theta$ and non-linearity h_0 results in the following equation:

$$\vec{x}' = h_0(W_0\vec{x} + b_0) \tag{2.13}$$

The output $\vec{x}''$ can be used as input to the next hidden layer, for example weights $W_1 \in \theta$, bias $b_1 \in \theta$ and non-linearity h_1:

$$\vec{x}'' = h_1(W_1 h_0(W_0\vec{x} + b_0) + b_1) \tag{2.14}$$

And so on. As the network grows deeper, the model can approximate more complex functions, but it also becomes harder to train. For that reason, much of the field of deep learning is dedicated to solving problems such as finding more reliable and faster methods of training neural networks and escaping local minima.

2.0.6. Convolution Network

Convolution Neural Networks(CNN) are analogous to Artificial Neural Networks(ANN) but with a difference. CNN is primarily used in the field of pattern recognition within images. This allows us to encode image-specific features into the architecture, making the network more suited for image-focused tasks,

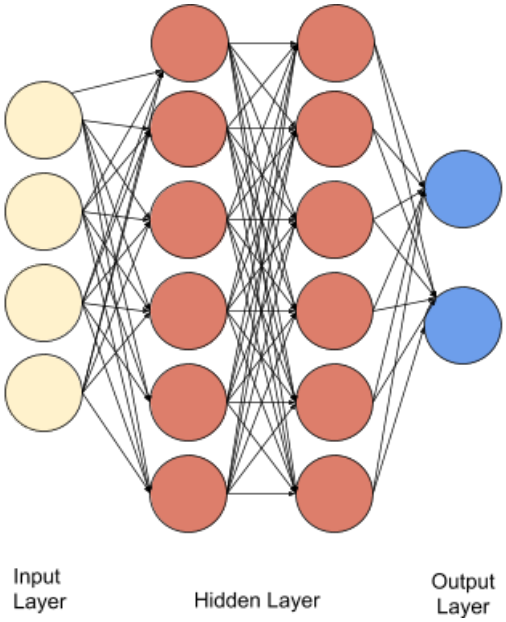

Figure 2.2: Artificial Neural Network [28]

whilst further reducing the parameters required to set up the model [18]. In ANN the input layer is fully connected to a series of hidden layers which ultimately is connected to the output layer as shown in Fig. 2.2. Whereas in CNN, only small regions of the input neurons are connected to the neurons in the hidden layer, these regions of the input layer are known as the local receptive fields. The local receptive field is translated across an image to create a feature map from input layer to the hidden layer. Like a typical ANN, CNN also have neurons with weights and biases. The model learns these values during the training process and it continuously updates them with each new training example. However in CNN, the weights and biases are same for all neurons in a given hidden layer. This means all neurons are detecting the same feature such as an edge or blob in different regions of an image. Then the output of each neuron is transformed using an activation function. This can be further transformed by applying a pooling step for reducing the dimensionality of the feature map. Finally, in the last hidden layer each neuron is fully connected to the output layer, this produces the final output.

The architecture of the CNN can be divided into the following subdivisions :

- As found in other forms of ANN, the input layer will hold the pixel values of the image.
- The convolutional layer will determine the output of neurons of which are connected to local regions of the input through the calculation of the scalar product between their weights and the region connected to the input volume. The rectified linear unit (or ReLu) aims to apply an 'elementwise' activation function such as sigmoid function to the output of the activation produced by the previous layer.
- The pooling layer will then simply perform downsampling along the spatial dimensionality of the given input, further reducing the number of parameters within that activation.
- The fully connected layers will then perform the same duties found in standard ANNs and attempt to produce class scores from the activations, to be used for classification. It is also suggested that ReLu may be used between these layers, as to improve performance by introducing non-linearity into the model.

2.0.7. Training the Neural Network

At the beginning of training of the neural network, we randomise the weights. But in order to produce optimum results we have to updates the weights so that our final result is as close as possible to the target weights. In order to do this, we define a Loss function or an error function and minimise it at

every step of training process. There are various methods used in order to update the weights to reach an optimum result. These are known as the Optimisation Algorithms, which help us minimise the Loss or Objective function and are based on certain learnable parameters of the model.

2.0.8. Gradient Descent

Gradient is a multi variate generalisation of a derivative. It has a direction and points to the direction of steepest increase of the function. Our job is to minimise the loss function, therefore we take a step in the negative direction of the gradient. If we compute the gradient of the loss function with respect to our weights and take a step in the negative gradient and update the new weights, eventually our loss function will decrease and converge to some local minima [2]. The idea is to choose a optimum step in the negative direction. If the step is too big, then the algorithm diverges and we jump over the minima. And if it is too small, we might converge to the local minima. Mathematically, it can be represented as the following:

$$w^{(i+1)} = w^{(i)} - \eta \nabla E(w^{(i)}), \tag{2.15}$$

where,
E : loss function
w : weight
η : learning rate.

In machine learning, the error function is generalised in the form of sums as follows:

$$E(w) = \frac{1}{n} \sum_{i=1}^{n} E_i(w) \tag{2.16}$$

There are various types of Gradient Descent method. The important ones are Mini-Batch Gradient Descent(MBGD) and Stochastic Gradient Descent(SGD).
In Mini-Batch, we approximate the derivative on some small batch of the dataset and use it to update the weights as shown in equation 2.17, whereas in Stochastic approach we update the weights for each training example as shown in equation 2.18.

Mini-Batch GD:

$$w^{(i+1)} = w^{(i)} - \eta \sum_{i=1}^{n} \nabla E_i(w)/n, \tag{2.17}$$

Stochastic GD:

$$w^{(i+1)} = w^{(i)} - \eta \nabla E_i(w), \tag{2.18}$$

2.0.9. ADAM

ADAM (ADaptive Moment Estimation) [9] is a variant of combination of AdaGrad and RMSProp [22]. It makes use of both the average first moment(mean)and the average second moment(variance) of the gradient.

$$m^{t+1} = \beta_1 \cdot m^t + (1 - \beta_1).\nabla E \tag{2.19}$$

$$v^{t+1} = \beta_2 \cdot v^t + (1 - \beta_2) \cdot \nabla E^2 \tag{2.20}$$

where, β_1 and β_2 are hyperparameters.

2.1. Single-Agent System

2.1.1. Single Agent

One of the earliest work done in the application of Deep Reinforcement Learning in the field of traffic flow problem was done by Li et al [14] which used Deep Q-Learning to control a single intersection. Several other researches started exploring this area and came up with different ways to define the states, rewards functions and actions for modelling the Traffic Light Control(TLC) problem. An agent in TLC problem is defined as a single intersection which controls all the traffic lights available at the intersection as shown in the Fig. 2.3.

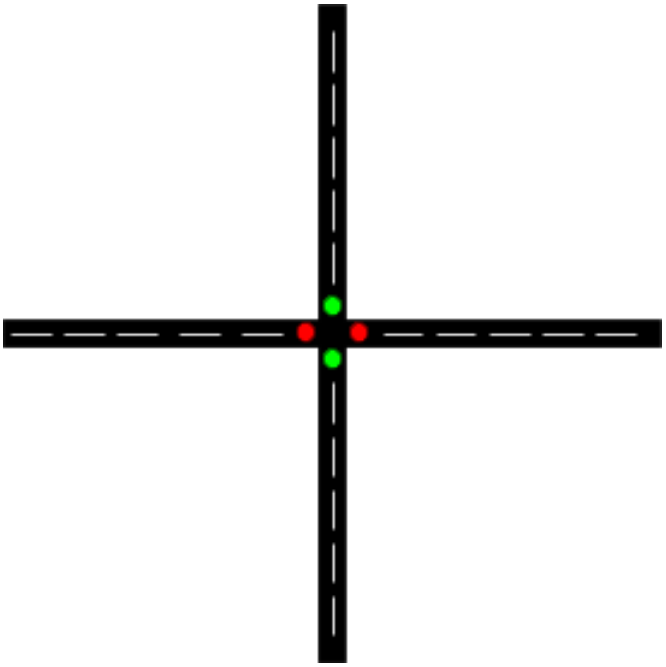

Figure 2.3: A Single Traffic Light Intersection.

2.1.2. States

The states in terms of traffic flow problem is usually defined in terms of vehicle's position and velocity. The definition of the state defined in this thesis is based on previous work [23][24][25][14][15][19]. The idea proposed is to divide the intersection in network of grids such that each element of the grid can contain maximum of one vehicle. The smallest element of the grid are small-sized square shaped. Depending on the presence of vehicle the grid is assigned binary values, either 1 or 0. Value is 1 when the vehicle is present and 0 when there is no vehicle in the grid. This can be quantified in terms of matrices, where corresponding to the cell of the grid, there is an element in the matrix, with binary values as shown in Fig. 2.4 and 2.5. In addition to representation of the cars in the matrix, traffic light might also be important information for the agent [23] [25]. So in order to incorporate the traffic light in the binary matrix, float values are assigned to different traffic light, specifically 0.8 for green, 0.5 for yellow and 0.2 for red.

2.1.3. Rewards

Several factors should be taken into account while formulating the reward function. One way to do is by penalising the agent every time the car stops which is commonly known as the waiting time. Other ways of penalising the agent is when the average speed of the vehicles in a lane is below the maximum allowed speed, well established as delay. Normalised delay is represented as the ratio of difference between maximum allowed speed and the vehicle speed to maximum allowed speed.

$$Normalised\ Delay = 1 - \frac{Vehicle\ Speed}{Maximum\ Allowed\ Speed} \tag{2.21}$$

The reward function used throughout this thesis is based on previous work [25] represented as:

$$Reward = -0.5 \sum_{i=1}^{N} d_i - 0.5 \sum_{i=1}^{N} w_i \tag{2.22}$$

where d_i and w_i represent individual vehicle normalised delay and waiting time.

2.1.4. Actions

Actions in case of traffic control problem are defined as the different combinations of traffic light signal at an intersection. It is usually expressed as different phases and the agent selects one of these phases

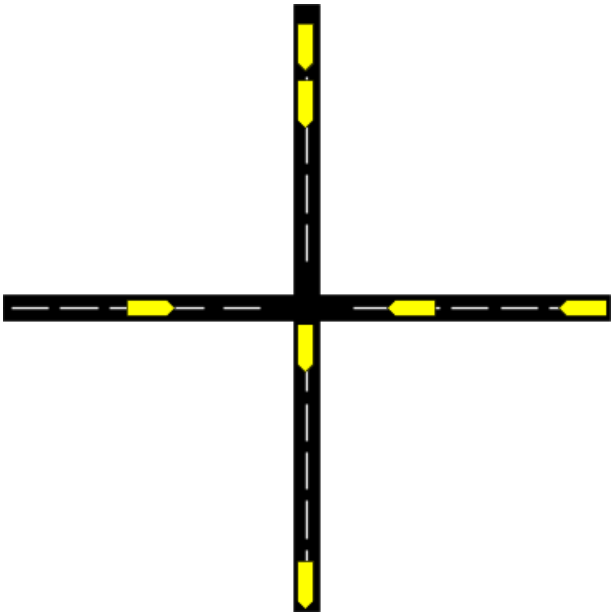

Figure 2.4: Representation of Traffic Environment.

Figure 2.5: Binary Matrix representation of corresponding Traffic Environment.

or actions to maximise the long term cumulative reward. Actions are essentially the options from which the agent choose in order to assign a traffic lane green light. Based on previous work [25], the possible actions made available to the agent are GrGr and rGrG as shown in the Fig. 2.6 and 2.7.

2.1.5. Yellow Light

Yellow signal is important for switching between two phases as it guarantees safety by allowing speeding vehicles to stop by providing them with enough time when the switch is being made between green and red signals. It can be formulated either as a fixed time for yellow light when switching between two actions or it can be selected as an action itself. In this thesis, the yellow light timing is chosen to be fixed and is activated every time there is a change in action. Based on previous work [25], it is assigned to be 4 seconds. This provides enough time for the traffic to come to a halt safely between switching of actions.

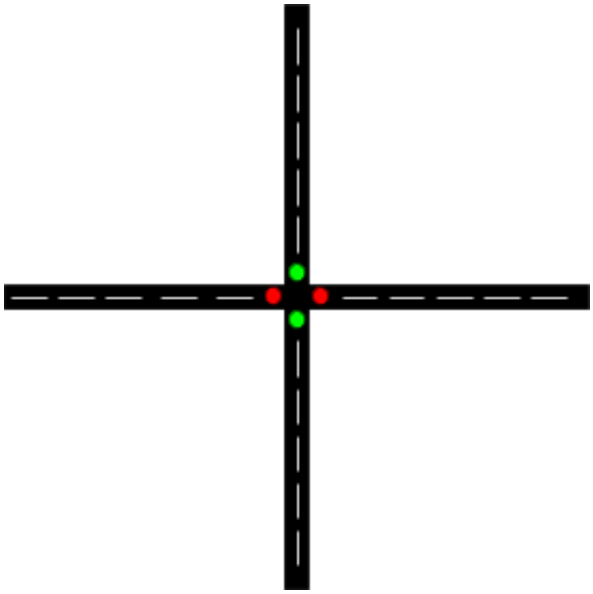

Figure 2.6: Representation of Action GrGr.

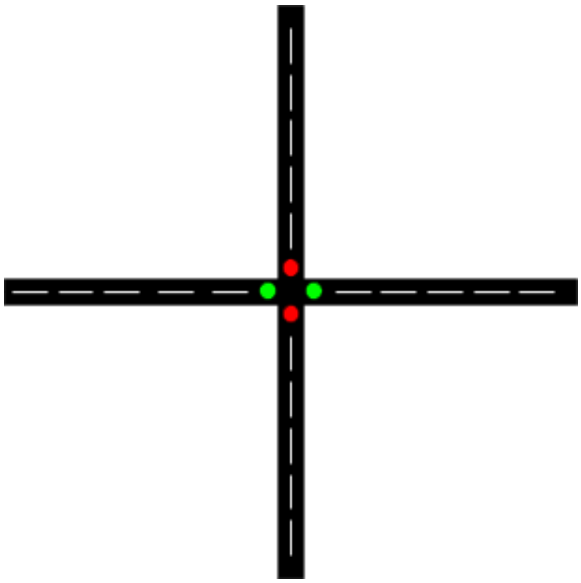

Figure 2.7: Representation of Action rGrG.

2.2. Multi Agent System

Multi Agent systems in Traffic Light control is represented as multiple intersections which coordinate with each other in order to reduce congestion. It is shown in the Figure 2.8.

2.3. Multi-Agent Learning

When the number of agents increases, the common goal of these multiple agents together is to find a joint optimal action such that the global reward is maximised. This joint optimal action is represented as the following:

$$\vec{a^*} = (a_1, a_2, ..., a_N) \tag{2.23}$$

where a_i is the local action taken by the agent i. One way to do is to have central controller that learns joint action for every agent present in the environment in order to maximise the global reward

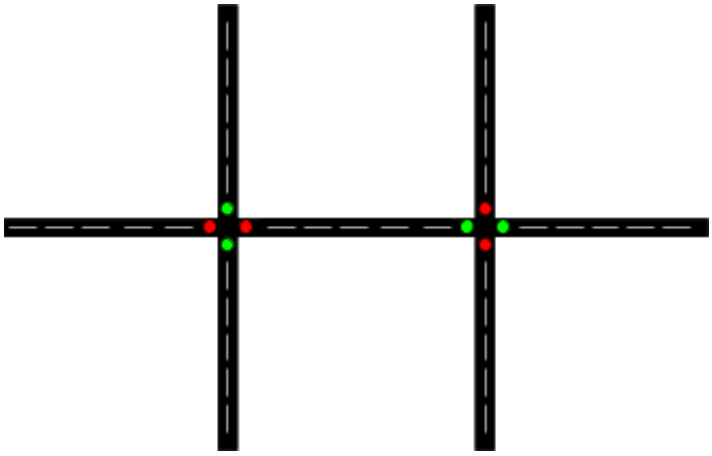

Figure 2.8: Two Agent Traffic Light Scenario.

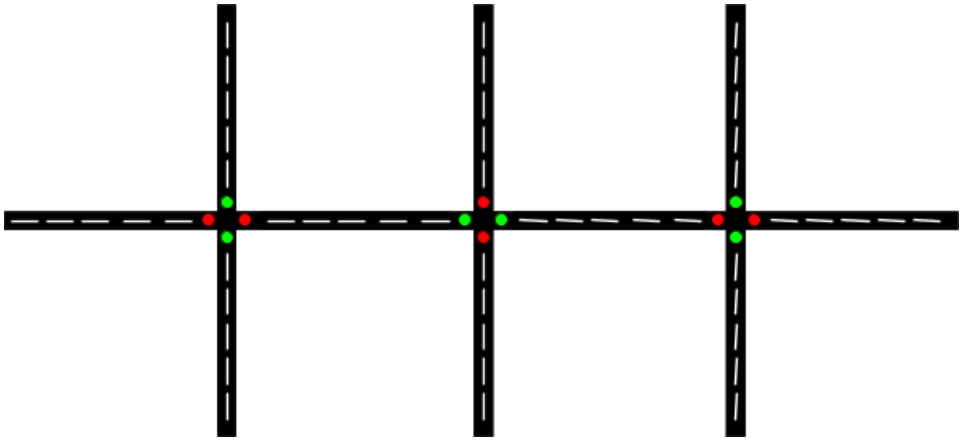

Figure 2.9: Three Agent Traffic Light Scenario.

and communicate to each agent its individual action. However, this approach is not feasible since the joint action space increases exponentially with increase in the number of agents.

Anther way to approach MAS is to factorise global Q-function into local Q-functions and then updation is performed either in coordination with other agents or individually as discussed in subsequent sections.

2.3.1. Independent Q-Learning

Here, at first we discuss the Multi-Agent reinforcement learning techniques, where each agent takes its own action independent of other agents [5]. The global Q-function(state-action value function) is expressed as a linear combination of all individual Q-functions as shown below:

$$Q(\mathbf{s}, \mathbf{a}) = \sum_{i=1}^{n} Q_i(\mathbf{s}, a_i) \tag{2.24}$$

The individual Q-function is updated entirely based on its own local reward as shown below:

$$Q_i(\mathbf{s}, a_i) := Q_i(\mathbf{s}, a_i) + \alpha[R_i(s, a) + \gamma \max_{a_i'} Q_i(\mathbf{s}', a_i') - Q_i(\mathbf{s}, a_i))] \tag{2.25}$$

This technique has storage and computational advantages since the action space for each agent is small. But convergence does not hold anymore since, actions are taken independent of each other and it ignores other agent's action which influences the system as a whole.

2.3.2. Distributed Q-Learning

In this approach [20], each agent's local Q-function is defined based on its individual action and state $Q_i(\mathbf{s}_i, a_i)$. Each agents coordinates with only a subset of agents. Each local Q-function is updated based on its neighbour j using a weight function$f(i,j)$ which determines how much the neighbour j contributes to the Q-function of agent i.

$$Q_i(\mathbf{s}_i, a_i) := (1-\alpha)Q_i(\mathbf{s}_i, a_i) + \alpha[R_i(\mathbf{s}, \mathbf{a}) + \gamma \sum_{j \in \{i \cup \Gamma(i)\}} f(i,j)\max_{a_j'} Q_j(\mathbf{s}', a_j')] \tag{2.26}$$

However, this approach leads to non-stationarity since both the agent and its neighbour are learning at the same time. A change is policy of the neighbour will lead to change in policy of the agent, thus leading to a condition of moving targets [3].

2.4. Coordination Algorithm

In this part of the chapter, various coordination algorithms are discussed which are used to factor MAS into smaller local sub-problems which coordinate with each other in order to find a joint optimal action[11]. As discussed earlier, one way to approach MAS is through a centralised approach, where a joint action is selected out of all possible joint actions, which maximises the global payoff. However, being aware of the challenges to this, it is not considered a feasible approach. Therefore in order to circumvent the underlying inherent challenges, the sections below suggest different popular approaches to handle MAS.

2.4.1. Coordination Graph

Coordination graph [6] is a graphical representation of the decomposition of the global payoff function into a set of smaller local factors, where each factor is a function involving less number of variables, depending on the subset of the agents that factor represent.

A coordination graph is represented as $CG = (V, E)$, where V are vertices and E are edges as shown in the Figure 2.10. The vertices represent the agents and the edges represent the relation or dependencies between the agents connected via the edge.

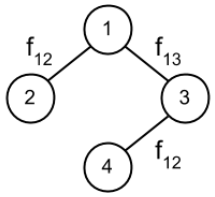

Figure 2.10: Coordination Graph.

The global payoff function is given by the equation below, where $i \in V$ and $(i,j) \in E$:

$$R(\mathbf{a}) = \sum_{(i,j)\in E} f_{ij}(a_i, a_j) \tag{2.27}$$

The joint optimal action is the one that maximises the global payoff function $R(\mathbf{a})$:

$$\mathbf{a}^* = \text{argmax}_a R(\mathbf{a}) \tag{2.28}$$

2.4.2. Variable Elimination

Variable Elimination [7] is an algorithm used to solve the coordination graphs by eliminating agents one at a time and maximising over that agent, that is finding the best action of the eliminated agent for each action of the non-eliminated agents. Even though it is much faster than maximising over joint action space and is an exact inference algorithm, but for large scale agents the problem scales exponentially. Since it is not a anytime algorithm [13](an algorithm which can be stopped anytime during running and get approximate results), it cannot be run for fewer iterations to get an approximate answer. Variable elimination is illustrated below for the coordination graph in the Fig. 2.10.

$$R(\mathbf{a}) = f_{12}(a_1, a_2) + f_{13}(a_1, a_3) + f_{34}(a_3, a_4) \tag{2.29}$$

$$\max_{\mathbf{a}} R(\mathbf{a}) = \max_{a_2,a_3,a_4} \{f_{34}(a_3, a_4) + \max_{a_1}[f_{12}(a_1, a_2) + f_{13}(a_1, a_3)]\} \tag{2.30}$$

$$\max_{\mathbf{a}} R(\mathbf{a}) = \max_{a_3,a_4}[f_{34}(a_3, a_4) + \phi_3(a_3))] \tag{2.31}$$

$$\max_{\mathbf{a}} R(\mathbf{a}) = \max_{a_4} \phi_4(a_4) \tag{2.32}$$

However, this is not always appropriate for real-time MAS where decision making must be done under time constraints.In these cases, an anytime algorithm that improves the quality of the solution over time would be more appropriate [26].

2.4.3. Brute Coordination

Brute Coordination is a naive approach to coordination in multi agent systems. In this approach, Q value is calculated for all possible combination of actions for the given number of agents. The action combination that leads to maximisation of the total combined Q value is chosen as the action to be implemented in the environment. This is explained clearly in the Algorithm 1 below:

Algorithm 1: Brute Force Coordination.

```
initialise sum_q_value = list[ ]
for all possible joint action combination a' do
    q_value = 0
    for all factored Q-function from 1 to n do
        q_value += Q_i(a'_i)
    sum_q_value.append(q_value)
a* = argmax_a' sum_q_value
return a*
```

2.4.4. Max-Plus Algorithm

Max-Plus [10] is an inference algorithm based on message passing, used to find the maximum a posteriori (MAP) state in graphical models. It converges to an exact solution for acyclic graphs but cannot guarantee convergence for cyclic graphs. It is based on a message passing parameter as shown:

$$\mu_{ij}(a_j) = \max_{a_i} \left[f_{ij}(s, a_i, a_j) + \sum_{k \in ne(i) \setminus j} \mu_{ki}(i) \right] \tag{2.33}$$

Message containing information over locally optimal actions are sent between agents to iteratively find the optimal joint action. Thus, a message from i to j is as shown above, where $ne(i)\backslash j$ is the set of $i's$ neighbours, excluding j. In short, i sends a message to j that consists of a maximization over i and $j's$ factor and the messages i has received from its neighbours that are not j. By iteratively sending messages, max-plus converges to the maximal joint action in acyclical graphs. Moreover, the algorithm has an anytime solution, meaning that it can be run using fewer iterations, and still get an approximate solution.

2.4.5. Transfer Planning

Based on the previous work by Oliehoek et al.[23][24][17] [16], transfer planning can be a good approach for solving Multi-agent systems. In Transfer Planning [17], Q-function is learnt for a subproblem of a larger multi-agent problem. Training on the source problem results in an approximation of its Q-function. Provided that the source problem and other subproblems are similar, we can then re-use the source problem's Q-function for each subproblem in the larger multi-agent problem, rather than training

Algorithm 2: Pseudo-code of the centralised max-plus algorithm for $G = (V, E)$.

initialise $\mu_{ij}(a_j) = \mu_{ji}(a_i) = 0$ for $(i,j) \in E, a_i \in A_i, a_j \in A_j$
initialise $g_i(a_i) = 0$ for $i \in V, a_i \in A_i$, and $m = -\infty$
while *fixed point = false and deadline to send action has not yet arrived* **do**
  // run one iteration,
  fixed point = true
  for *every agent* i **do**
    for *all neighbours* $j = \Gamma(i)$ **do**
      compute $\mu_{ij}(a_j) = \max_{a_i} \left[f_{ij}(a_i, a_j) + \sum_{k \in \Gamma(i) \setminus j} \mu_{ki}(i) \right] + c_{ij}$
      send message $\mu_{ij}(a_j)$ to agent j
      if $\mu_{ij}(a_j)$ *differs from previous message by a small threshold* **then**
        fixed point = false
      end
    end
    compute $g_i(a_i) = f_i(a_i) + \sum_{j \in \Gamma(i)} \mu_{ji}(a_i)$ and
    $a_i' = \textbf{argmax}_{a_i} g_i(a_i)$
  end
  $\mathbf{a}' = (a_i')$
  if *use anytime extension* **then**
    if $R(\mathbf{a}') > m$ **then**
      $\mathbf{a}^* = \mathbf{a}'$ and $m = R(\mathbf{a}')$
    end
  else
    $\mathbf{a}^* = \mathbf{a}'$
  end
end
return $\mathbf{a}^*$

a Q-function for each separate subproblem. in the environment introduced by multiple agents learning and acting simultaneously.

This transfer planning approach circumvents two problems present in multi-agent reinforcement learning. The first is the non stationarity in the environment introduced by multiple agents learning and acting simultaneously. By training on a source problem. the environment dynamics do not change during learning. The second is the cost of training many agents simultaneously. Because the source problems are independent, they can be solved independently (e.g. sequentially). Moreover, exploiting symmetries of our source further reduces the computational cost.

Transfer Planning can be formalised in the following way:

- $\sum$ the set of source problems $\sigma \in \sum$;
- $\mathcal{D}^\sigma = \{1^\sigma, ..., n^\sigma\}$, the agent set for source problem σ;
- E maps each Q-value component e to a source problem $E(e) = \sigma \in \sum$;
- $\mathrm{A}^\sigma : \mathbb{A}(\mathrm{e}) \to \mathcal{D}^{\mathbf{E(e)}}$ maps agent indices $i \in \mathbb{A}(\mathrm{e})$ in the target task to indices $A^e(i) = j^\sigma \in \mathcal{D}^\sigma$ in the source task σ for that particular component $\sigma = E(e)$.

3

Experimental Setup

Traffic flow problem can be visualised as a Single or Multi Agent problem, which can be trained in order to optimise the traffic flow. In terms of traffic light control, the agent can be modelled as the traffic signal itself, which takes certain actions like changing the traffic light and receives rewards based on these actions and tries to maximise the cumulative long term reward effectively leading to smooth flow without any delays. The simulation of the traffic flow can be performed in one of the simulation software known as SUMO(Simulation of Urban Mobility) [12] discussed in the subsequent section. This section also covers the neural network architecture used, training and evaluation procedure as well as the design and modelling of traffic simulation and congestion.

3.0.1. Network Architecture

The network architecture followed is based on previous work in traffic light problem using DRQN approach [25] [8].

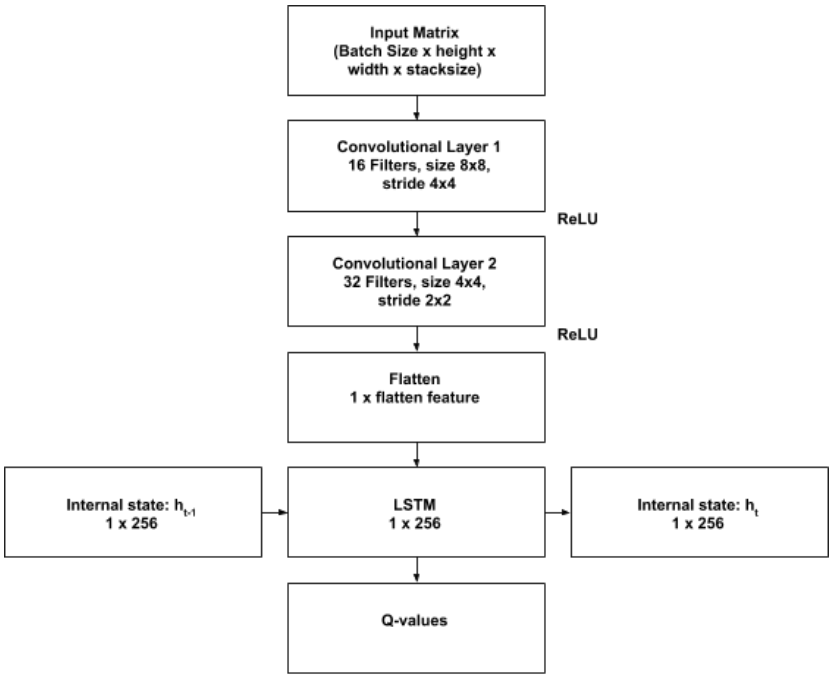

Figure 3.1: Network Architecture.

3.0.2. Training

The agents are trained for 1,000,000 time steps using the network described above. During the filling of replay memory, the exploration rate is set to $\epsilon = 1$, and once the memory is filled it is reduced to as mentioned in the Table 3.1. Algorithm 3 describes the DRQN approach along with Experience Replay(ER) used to train the agent.

Matrix Size	84 x 84
Matrix State Type	Binary Position Matrix + Light Configuration
Discount factor(γ)	0.99
Learning Rate(α)	0.00025
Exploration Rate(ϵ)	0.1
Freeze Interval	30,000
Optimiser	Adam
Batch Size	32
History Size(τ)	10
Replay Memory Size	30,000
Stack Size	1

Table 3.1: Parameters used for Training.

Algorithm 3: Deep Recurrent Q-Learning Algorithm with Experience Replay

Initialise Primary Q-Network Q_{θ^0} and Target Network Q_{θ^-} ;
Initialise the Replay Memory M = [ ];
Initialise Environment, state s_0, Action space $U(A)$;
$i \leftarrow 0$
while $i < M$ **do**
  Randomly select an action a_i from the action space $a_i \sim U(A)$.
  Obtain reward r_i, next state s_i'
  Add $(< s_i, a_i, r_i, s_i' >)$ to the memory M
end
while i < maximum training time **do**
  With probability ϵ select random action a_t;
  Otherwise a_t = argmax$'_a$Q$_{\theta^0}(s_t, h_{t-1}, a_t)$;
  Obtain reward r_i, next state s_i';
  Add $(< s_i, a_i, r_i, s_i' >)$ to the memory M
  if i % Freeze interval $== 0$ **then**
    $Q_{\theta^0} \leftarrow Q_{\theta^-}$
  end
  $B = (s_j, a_j, r_j, s_j')...(s_{j+\tau}, a_{j+\tau}, r_{j+\tau}, s_{j+\tau}')_{j=1}^{\text{batch size}} \subseteq M$
  for each sequence$(s_j, a_j, r_j, s_j')...(s_{j+\tau}, a_{j+\tau}, r_{j+\tau}, s_{j+\tau}') \in B$ **do**
    $h_{j-1} \leftarrow 0$;
    for $k = j$ *to* $k = j + \tau$ **do**
      update the hidden state $h_k = Q_{\theta^0}(s_k, h_{k-1})$
    end
    $y_j = r_{j+\tau} + \gamma Q_{\theta^-}(s_{j+\tau}'$, argmax$_{a'}Q_{\theta^0}(s_{j+\tau}', h_{j+\tau}), h_{j+\tau})$;
    $\Omega_j = (y_{j+\tau} - Q_{\theta^0}(s_{j+\tau}, a_{j+\tau}, h_{j+\tau-1}))^2$
  end
end

3.0.3. Evaluation

After every 10,000 steps, each model is evaluated by performing purely greedy policy. Evaluation is done by running 8 SUMO simulations and the results are plotted in terms of reward per time step and average travel time along with standard error. Finally the best performing model is selected based on the minimum average travel time for 8 simulations of all the evaluated models and it is then used to

evaluate scaled up traffic scenarios using Transfer Planning approach along with different Coordination Algorithm.

3.0.4. SUMO(Simulation of Urban Mobility)

SUMO [12] is a free, open source software that allows for a realistic simulation of different traffic networks. It comes with plethora of tools which can be used for visualisation, emission calculations, network importing and finding the route. It allows to import maps from OpenStreetMap, VISUM, VISSIM etc. It can be used to model multimodal traffic like vehicles, pedestrians, public transport as well as cyclists. SUMO is implemented in C++ and only uses portable libraries.

3.0.5. Demand Data generation

In SUMO [12] the demand data refers to when and where vehicles are generated in the traffic environment and which route they follow during the simulation. Each vehicle generated in the environment is assigned a route randomly from source to destination. In the Demand Data algorithm below, the probability p (car probability) is the probability of a car entering the environment at each time step.

Algorithm 4: Demand Data generation

```
Define different possible Routes for the car.
Initialise Route probability p
Initialise Route List = [ ]
for t= 0 to N do
    for Routes in Possible Route do
        sample ρ ~ U(0, 1)
    end
    if p > ρ then
        Route List.append(Routes) at time t.
    end
end
```

3.0.6. Traffic Congestion

The term Traffic congestion is defined as a condition which leads to longer travel time, increase in vehicle queuing or delays. One of the more formal definition is that it is the situation where the introduction of an additional vehicle into a traffic flow increases the journey times of the others [21]. Up to a certain level of vehicles, the traffic flow is smooth without any delays, however with increase in the number of vehicles above a threshold leads to congestion. These vehicles not only increase their own travel time but also cause delay for other vehicles.

There is no standard metric to define congestion, however most measures include increased delays, travel time or queuing. In order to define traffic congestion for our problem we use one of the definitions available in the literature [4]. In the book [4], congestion is calculated in terms of increase in travel time to introduction of additional vehicles in the traffic scenario.

In the Figure 3.2, two plots are shown, one representing the average travel time$t = f(q)$ as a function of different car probability(q)(volume of traffic). In order to calculate average travel time for different traffic volume, 8 simulations were carried out and average of travel times was computed and plotted. The second plot is the average travel time due to increase in the traffic volume $\frac{\partial(qt)}{\partial q} = t + qf'(q)$. The difference between the two graphs, indicate the increase in the travel time due to introduction of additional vehicles. Up to a certain point both the graphs coincide indicating the fact that, till this point of coincidence the additional vehicles introduced do not cause increase in travel time. However, after the point of coincidence, the two graphs begin to diverge, indicating that the additional vehicles are increasing the overall travel time. As we go along the x-axis this difference becomes larger and larger, which means the average traffic speed increases and the travel time decreases leading to higher congestion. Based on the above inference, table 3.2 lists the three different congestion configurations used throughout the thesis.

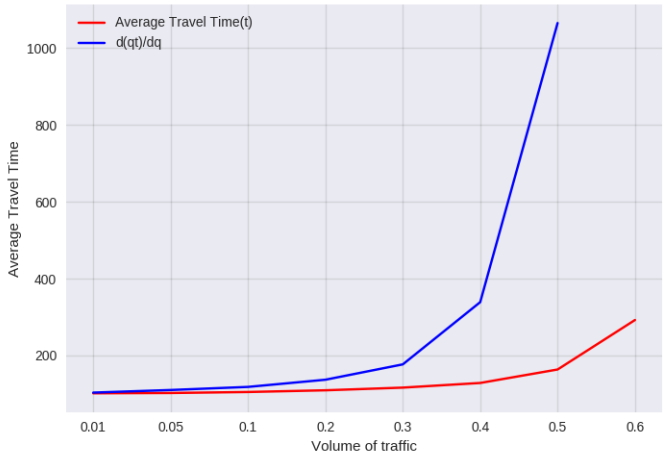

Figure 3.2: Representation of Concept of Traffic Congestion .

Car probability	Congestion
0.05	Low
0.2	Medium
0.4	High

Table 3.2: Different Traffic Congestion scenarios.

Bibliography

[1] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. Sumo–simulation of urban mobility: an overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.

[2] Christopher M Bishop. *Pattern Recognition and Machine Learning, 2006*. Springer, 2006.

[3] Lucian Bu, Robert Babu, Bart De Schutter, et al. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[4] Alberto Bull. *Traffic Congestion: The Problem and how to Deal with it*. Number 87. United Nations Publications, 2003.

[5] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998(746-752):2, 1998.

[6] Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored mdps. In *Advances in neural information processing systems*, pages 1523–1530, 2002.

[7] Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *ICML*, volume 2, pages 227–234. Citeseer, 2002.

[8] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*, 2015.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Jelle R Kok and Nikos Vlassis. Using the max-plus algorithm for multi-agent decision making in coordination graphs. In *Robot Soccer World Cup*, pages 1–12. Springer, 2005.

[11] Jelle R Kok and Nikos Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7(Sep):1789–1828, 2006.

[12] Daniel Krajzewicz, Georg Hertkorn, Christian Feld, and Peter Wagner. Sumo (simulation of urban mobility); an open-source traffic simulation. pages 183–187, 01 2002. ISBN 90-77039-09-0.

[13] Lior Kuyer, Shimon Whiteson, Bram Bakker, and Nikos Vlassis. Multiagent reinforcement learning for urban traffic control using coordination graphs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 656–671. Springer, 2008.

[14] Li Li, Yisheng Lv, and Fei-Yue Wang. Traffic signal timing via deep reinforcement learning. *IEEE/CAA Journal of Automatica Sinica*, 3(3):247–254, 2016.

[15] Xiaoyuan Liang, Xunsheng Du, Guiling Wang, and Zhu Han. Deep reinforcement learning for traffic light control in vehicular networks. *arXiv preprint arXiv:1803.11115*, 2018.

[16] Frans Oliehoek. *Value-based planning for teams of agents in stochastic partially observable environments*. Amsterdam University Press, 2010.

[17] Frans A Oliehoek, Shimon Whiteson, and Matthijs TJ Spaan. Approximate solutions for factored dec-pomdps with many agents. In *Proceedings of the 2013 International Conference on Autonomous agents and Multi-Agent Systems*, pages 563–570. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[18] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[19] Tobias Rijken. *DeepLight: Deep reinforcement learning for signalised traffic control*. PhD thesis, Master's Thesis. University College London, 2015.

[20] Jeff Schneider, Weng-Keen Wong, Andrew Moore, and Martin Riedmiller. Distributed value functions. In *ICML*, pages 371–378, 1999.

[21] Ian Thomson and Alberto Bull. *La congestión del tránsito urbano: causas y consecuencias económicas y sociales*. CEPAL, 2001.

[22] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[23] Elise van der Pol. Deep reinforcement learning for coordination in traffic light control. 2016.

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

[25] Jaimy van Dijk. Recurrent neural networks for reinforcement learning: an investigation of relevant design choices. 2017.

[26] Nikos Vlassis, Reinoud Elhorst, and Jelle R Kok. Anytime algorithms for multiagent decision making using coordination graphs. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*, volume 1, pages 953–957. IEEE, 2004.

[27] MA Wiering. Multi-agent reinforcement learning for traffic light control. In *Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000)*, pages 1151–1158, 2000.

[28] Wikipedia.org. Ann. `https://commons.wikimedia.org/wiki/File:Colored_neural_network.svg`.