Multi Agent Deep Recurrent Q-Learning for Different Traffic Demands

Azlaan Mustafa Samad

Multi Agent Deep Recurrent Q Learning for Different Traffic Demands

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 30, 2020 at 09:30 AM.

Student number: 4962702
Thesis committee:
Dr. ir. F. Oliehoek, TU Delft, Daily Supervisor
A. Czechowski TU Delft, Interactive Intelligence Group Member
Prof. dr. ir. C. Vuik, TU Delft, Chair Professor
Prof. dr. ir. A. W. Heemink, TU Delft, Full Professor

Contents

Abstract

In today's scenario due to rapid urbanisation there has been a shift of population from rural to urban areas especially in developing countries in search of better opportunities. This has lead to unprecedented growth of cities leading to various urbanisation problems. One of the main problems that comes across in urban areas is the increased traffic congestion. This has led to pollution and health issues among people. With the current advancement in *Artificial Intelligence*, especially in the field of *Deep Neural Networks* various attempts have been made to apply it in the field of Traffic Light Control. This thesis is an attempt to take forward the problem of solving traffic congestion thereby reducing the total travel time. One of the contributions of this thesis is to study the performance of *Deep Recurrent Q-network* models in different traffic demands or congestion scenarios. Another contribution of this thesis is to apply different coordination algorithms along with *Transfer Learning* in *Multi-Agent Systems* or multiple traffic intersections and study their behaviour. Lastly, the performance of these algorithms are also studied when the number of intersections increase.

Acknowledgement

This thesis has been possible due to the continuous support of my supervisor Dr. Frans A. Oliehoek who kept me motivated to achieve better results and implement novel techniques. I would also like to thank my other supervisor Prof. Kees Vuik for accepting a Computer Science graduation project despite the course I am pursuing is Applied Mathematics. He also helped me throughout with his valuable suggestions for the thesis and the time spent in Delft. It would be unfair to not mention A. Czechowski's contribution. He gave me very valuable and indeed helpful suggestions with programming and debugging. He has been a constant support and a motive force. I would also like to thank Prof. Reinhard Nabben from TU Berlin during my first year of Master's course at TU Berlin.

Lastly, I would also like to thank my family and friends for the love and support throughout my life.

Abbreviations and Symbols

Abbreviations

RL:	Reinforcement Learning
MDP:	Markov Decision Process
MAS:	Multi Agent System
TP:	Transfer Planning
DRQN:	Deep Recurrent Q-Network
DQN:	Deep 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
MSE:	Mean Squared Error
TLC:	Traffic Light Control
i.i.d:	Independent and Identically Distributed
ER:	Experience Replay
VE:	Variable Elimination
CG:	Coordination Graph
BC:	Brute Coordination
MP:	Max-Plus
IC:	Individual Coordination

Symbols

[!h] $\mathcal{S}$:	State space
$\mathcal{A}$:	Action space
G_t:	Expected cumulative reward
R_t:	Reward at time step t
$v_\pi(s)$:	Value function
$q_\pi(s,a)$:	Q-function or Q-value 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
$\vec{a^*}$ or **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
n:	Number of intersections
p:	Number of Edges in a Coordination Graph
d:	Average number of neighbours per agent
q:	Probability of a car entering the simulation environmnet at each time step
Q_{ij}:	Factored or local Q-function for agent i and j
a_i:	Action of agent i
m:	Total number of iteration for MaxPlus Algorithm
t:	Time step

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 issues such as delays, accidents, noise, air pollution and monetary losses. Based on a 2012 study by Washington University, it was concluded that long commutes eat up exercise time leading to various health issues like obesity problems, lower fitness levels, higher blood pressure—all strong predictors of heart disease and diabetes[4]. Today's traffic light systems leads to inefficient traffic flow and therefore longer travel times.

In some recent research work [31], Reinforcement Learning(RL) has been used to reduce the total travel time thereby reducing delays. This is executed by treating the traffic light control problem as a sequential decision making problem. In a sequential decision making problem, an agent which is the traffic light intersection, first observes the traffic environment. The environment conveys the agent relevant information about the traffic. Then the agent based on its observation takes an action, that is it changes the traffic signals thereby receiving rewards. In RL, the agent(or the traffic light intersection) learns from trial and error by interacting with the environment. The goal of the agent is to maximise the reward(or reduce the average travel time). 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 or simply said it contains the information about the traffic *environment*, the *actions* are the different traffic lights configurations(red, green or yellow) which the agent can choose from, while the *rewards* are formulated by taking into account different factors like waiting time, delays etc. The reward function specific to the traffic control problem are formulated to penalise the agent when there is an increase in the travel time. As mentioned earlier that the agent learns from trial and error, this means that the agent explores by taking different actions and eventually learns which actions are better suited for a given environment.

This thesis is a continuation of previous work [31] [28] [26] [27] wherein Deep Q-Learning is used to predict Q-values which chooses an action in order to maximise the total cumulative reward of the state. The Q-value tells the agent which action to perform so that it can gain more rewards in the long term. 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 intersections are more than one. In multiple intersection case, each intersection needs to coordinate with its neighbours since a poor performance of one intersection can lead to traffic congestion in other lanes. Moreover instead of training the agent for different number of intersections in a bigger problem, a variant of *Transfer Learning* known as *Transfer Planning* [19] can be used. This method uses the idea of breaking down multiple intersection problem into smaller source problems(or problems containing fewer traffic intersections). These source problems can be trained and then used in a bigger problem while they coordinate with each other to reduce total travel time for the entire system. In order to define relationships or dependencies between these sub-problem, Coordination Graphs [6] can be used. Thus the problem reduces to decomposition of global coordination problem into local coordination problem to find a global optimal joint action. In order to coordinate between the intersections, various coordination algorithms are available

in the literature. In this thesis the algorithms used for coordination are Brute Coordination, Max-Plus and Individual Coordination. It is interesting to study the performance of these algorithms in multiple intersection case. While Brute Coordination algorithm scales exponentially with increase in the number of coordinating agents, Max-Plus is considered a good approximate algorithm for coordination since it involves communicating with its neighbouring agents by exchanging *messages*. These algorithms can be compared with Individual Coordination which involves absolutely no communication.

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

1.0.1. Research Questions And Contributions

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

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

- *How does the TLC agent perform for different traffic demands?*
  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 conclusions can be drawn on how to choose low, medium or high traffic congestion.

- *How does the different coordination algorithms perform computationally in case of Traffic Light-Control problem?*
  This can be studied by measuring the time complexity and the actual runtime for the different algorithms as the number of intersections increase. There can be several factors influencing the performance as the number of agents are scaled up.

1.0.2. Outline

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

2

Background

In this section, Reinforcement learning is introduced which is used to train the traffic light agent. When we think of learning, we think of interaction. In order to learn something, we always rely on feedback. Our learning process is dependent on what feedback we receive during the learning process. Suppose, when an infant is learning simple things about it surrounding, it interacts with the surrounding and perceives the feedback from the surrounding using its sensory organ and then this feedback helps him or her about the consequences. These experiences gained during the learning process can be useful for lifetime. Therefore, learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence [23]. Then in order to mathematically formalise RL, Markov Decision Process(MDP) is dicussed. In a MDP, we see how an agent interacts with the environment, and receives rewards which leads to the ultimate goal of the task. An agent is simply the infant, and the environment is the surrounding of the infant. So when an infant touches something hot, it perceives displeasure(reward) in the form of burn though its sensory organs and immediately takes his or hand off, which is the action taken by the agent. This helps the infant to know that this action was bad and it should not take this action when it encounters the same situation again. This is how an infant learns. In order to mathematically form this *goodness* of a particular state, four subparts of RL is discussed: *reward function, policy, value function and state-value function* However a MDP is an idealised framework of RL, therefore we need real life framework for the learning process. Thus we introduce RL algorithms in order to generalise real life scenarios in a better way. We discuss two types of RL methods namely, *model-free* and *model-based*. In a model-based method an agent is able to predict the outcome of its action. While in a model-free method, the agent estimates value function and then takes its action accordingly. Then we build on the idea of model-free RL methods using neural networks, which is a way to estimate these value function approximately. This forms the main idea of implementation used in this thesis.

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. An agent is the one that learns and takes decision or action. In simple words, it is a mapping from state to actions. In order to take action, an agent observes the environment and makes its decision based on the visibility of environment. The agent 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. When an agent takes a action based on the current state, it lands in a different state and receives a reward. Thus, the next state is influenced by its current action and therefore subsequent rewards. The agent is not told which action to take instead it must learn which action yields most reward. In order to increase its reward, an agent must take an action that has in the past been fruitful or has secured rewards. But in order to find these actions, it must try a lot of different actions. This leads to a dilemma of exploration and exploitation, an agent must trade-off between exploration and exploitation.

Next we discuss the mathematical framework to describe a Reinforcement Learning.

2.0.2. Markov Decision Process

Markov Decision process (MDP) is the idealised mathematical framework of sequential decision making used in RL. As shown in the Fig. 2.1, an agent observes the *environment*, environment is best described as the surrounding an agent is in, excluding itself. In this process, the agent observes the environment at each time step $t = 0, 1, 2, 3\ldots$ in the form of state s_t and 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}.

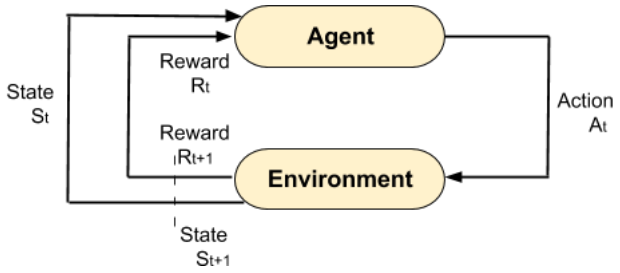

Figure 2.1: The agent–environment interaction in a Markov Decision Process.

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 maximise the total reward it receives over time, this means maximising not just the immediate reward but the cumulative reward in the long run. A reward signal is a function which helps an agent achieve its ultimate goal. It should be framed in such a way that by maximising cumulative reward, it is able to achieve the final task. 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 R_i is the reward at time step i and 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:

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

Now another important part of RL is discussed which is known as *policy*. A *policy* is a mapping from state to probabilities on which action to select. 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.

A *Value function* is an estimate of how good it is for an agent to be in a particular state, which is defined in terms of 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 the *Action-Value function* or popularly known as the Q-function. It is similar to value function except that it takes into account an action as well. An 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 $V_\pi(s)$, since the number of times the state is encountered is infinite. If the average is kept for each action taken when in a particular state, then this will converge to the action-value function $q_\pi(s,a)$.

The Bellman Equation 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 $A \in \mathcal{A}(s)$.

It is 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:

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

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.

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

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

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(the property of probability *p* for each possible value of s_t and r_t depends only on the immediately preceding state and action s_{t-1} and a_{t-1}) and sufficient computational power. Therefore, in many cases the transition probability is unknown. Under such a scenario, the agent uses RL algorithms. The agent learns a mapping from state to actions from interacting with the environment and receiving feedback. Thus we can define two types of Reinforcement Learning:

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

These two methods are different but the core problem is similar, which is the estimation of value functions. In a Model-based method, an agent can predict how the environment responds to its action. In other words, it is able to predict the next state and reward based on its current state and reward. While model-free relies on learning from experience. Having established the fundamentals of RL through MDP, we will look at different ways to estimate these value functions through model-free methods.

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. However, in many practical cases, the state space is enormous. And for such enormous state space, we need large look-up tables which leads to memory issues as well as computational time increases tremendously. Therefore we need a way to overcome this memory and computational time issue. In the following subsection, how to overcome these issues with approximation are discussed.

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):

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

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. Machine Learning algorithms assumes data samples used for training to be uncorrelated as well as independently and identically distributed(i.i.d). However, due to function approximation of the q values, these assumptions are violated. The reasons for violations are discussed in the subsequent sections.

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, whilst further reducing the parameters required to set up the model [20]. 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.

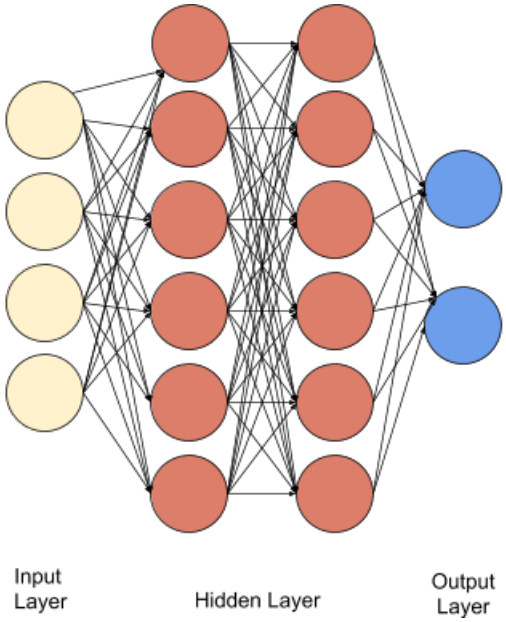

Figure 2.2: Artificial Neural Network [32]

- 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 [1]. 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 [25]. 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. Issues with Deep Q Learning

In this section various problems related to DQN are discussed and related solutions available in the literature are explained.

2.1.1. High Correlation

As mentioned earlier, the data samples used to train the DQN are highly correlated in nature. This is due to the fact that the next state-action pair (s_{t+1}, a_{t+1}) is highly dependent on the current state-action pair (s_t, a_t). Thus the transition probability is highly dependent on the current state-action (s, a) pair.

2.1.2. Data Distribution

As the DQN is trained, the Q-values are *globally* iteratively updated based on the data samples stored in the replay memory M. This update of Q-values leads to a non-stationary data distribution because the data sample at a particular time step can follow a very different distribution when compared to a data sample from a later stage of training. Thus, the samples tend to be heavily correlated as well as dependent in nature with non-identical distribution.

2.1.3. Moving Q-Target

As seen earlier, a DQN is updated using the MSE. Thus when we update the weight based on MSE, the weights of both the current Q-value as well as the Q-target is updated. This thus changes the target Q-value causing the issue of moving Q-targets. Thus as we move closer to the target Q-values, we also change the target values, thereby we keep chasing the target without getting closer to it. This leads to either oscillation or divergence thereby destabilising the system.

2.2. Overcoming Issues

In order to overcome the aforementioned issues with DQN, the following methods are suggested which are widely used.

2.2.1. Experience Replay

In order to overcome the issue of highly correlated and non-i.i.d data samples, a widely used approach is Experience Replay(ER) [15][17]. In this approach, a large database(replay memory M) is created in order to store experiences before the starting of the training procedure. These stored data samples are later uniformly sampled from the replay memory as mini-batches in order to train the network. This gets rid of the issue of non-stationary data distribution as well as the correlation. The experiences stored are in the form of $< s_t, a_t, r_t, s_{t+1} >$ tuples.

2.2.2. Freezing Target Network

As mentioned earlier, when we *globally* update the weights of the Q-network, the target Q-value changes too. A way to overcome this problem of moving target would be to freeze the Q-target values. This is implemented by creating two Q-networks, one primary and one target network. The target Q-network is used to estimate the target Q-values and kept frozen for a certain number of iteration, this period is known as *freeze interval* [17]. After the *freeze interval*, the parameters of the updated primary network is copied to the target network. This alleviates the problem of moving targets.

3

Traffic Light System

3.1. Single-Agent System

3.1.1. 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 [13] 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. 3.1.

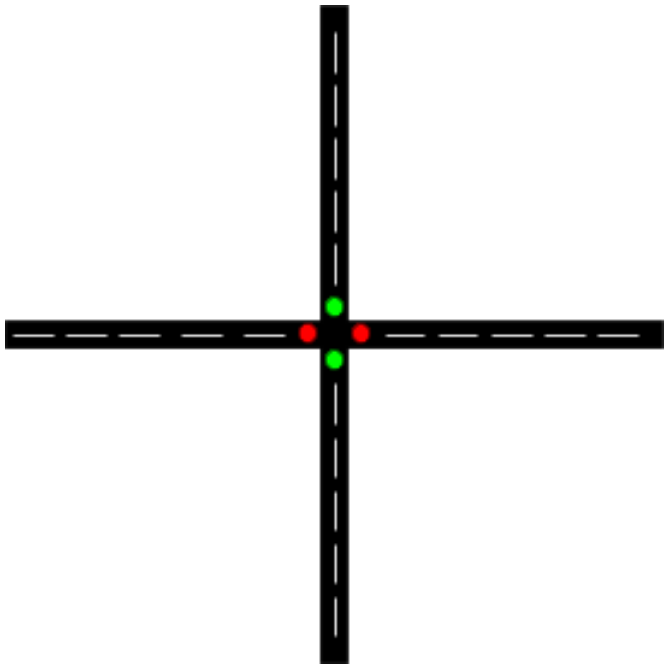

Figure 3.1: A Single Traffic Light Intersection.

3.1.2. 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 works [26][27][28][13][14][21]. The idea proposed is to divide the intersection into a network of grids such that each element of the grid can contain a maximum of one vehicle. The smallest elements of the grid are small-sized square shaped. Depending on the presence of a 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. 3.2 and 3.3. In addition to representation of the cars in the matrix, traffic light is also an important information for the agent [26] [28]. 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. These values are again chosen from previous works [26] [28]. Similar to the position of vehicles in the binary matrix represented by 1, the traffic light values are represented by their particular values at their specific positions in the binary state matrix(corresponding to their actual position in the traffic environment).

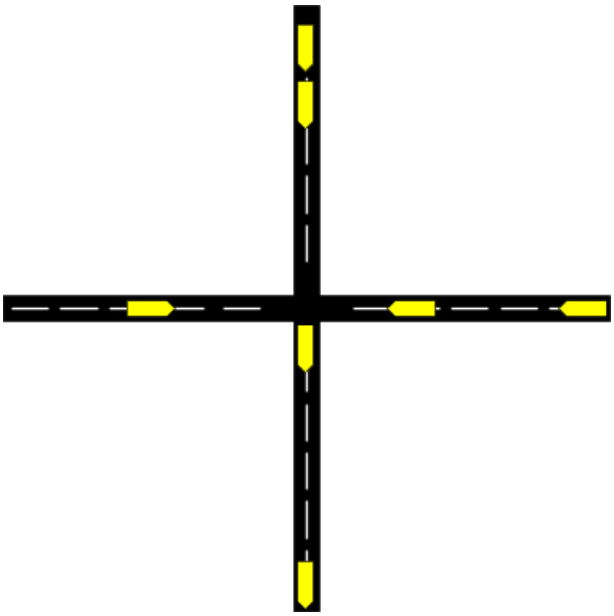

Figure 3.2: Representation of Traffic Environment.

Figure 3.3: Binary Matrix representation of corresponding Traffic Environment.

3.1.3. 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.

$$Normalised\ Delay = 1 - \frac{Vehicle\ Speed}{Maximum\ Allowed\ Speed} \tag{3.1}$$

The reward function used throughout this thesis is based on previous work [28] represented as:

$$Reward = -0.5 \sum_{i=1}^{N} d_i - 0.5 \sum_{i=1}^{N} w_i \tag{3.2}$$

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

3.1.4. Actions

The actions in case of a 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 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 [28], the possible actions made available to the agent are GrGr and rGrG as shown in the Fig. 3.4 and 3.5.

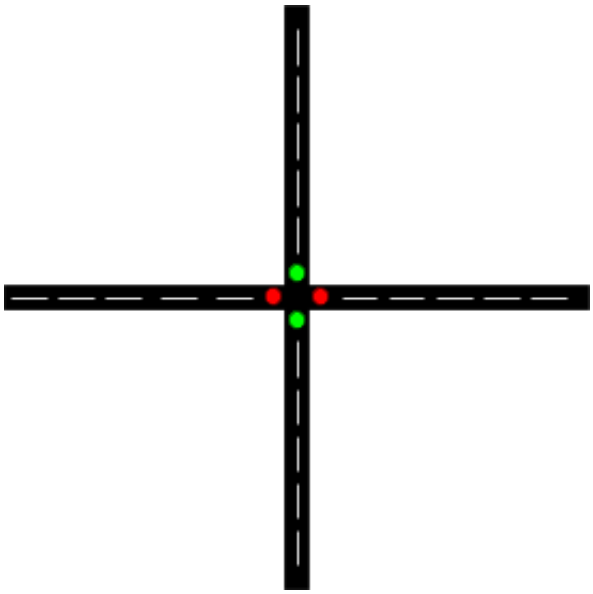

Figure 3.4: Representation of Action GrGr.

3.1.5. Yellow Light

The 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 [28], it is assigned to be 4 seconds. This provides enough time for the traffic to come to a halt safely between switching of actions.

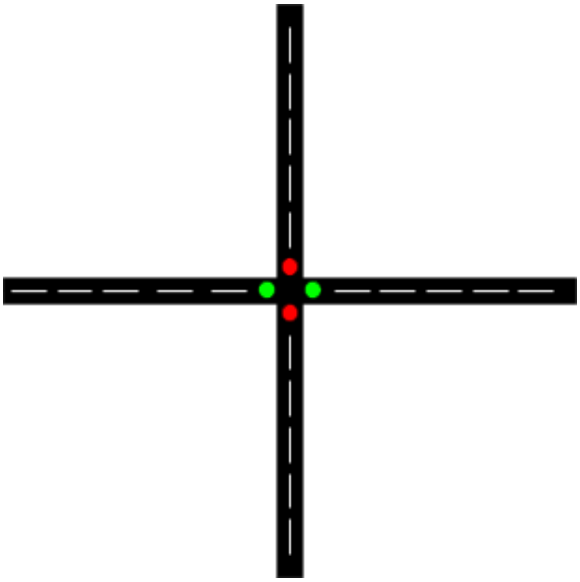

Figure 3.5: Representation of Action rGrG.

3.2. 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 3.6 & Fig. 3.7.

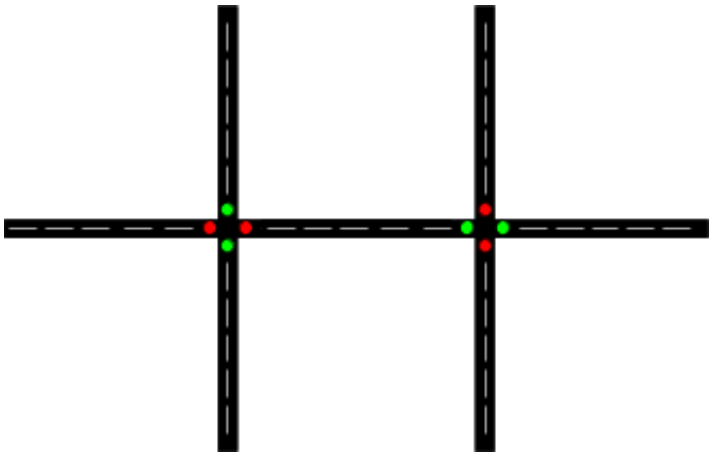

Figure 3.6: Two Agent Traffic Light Scenario.

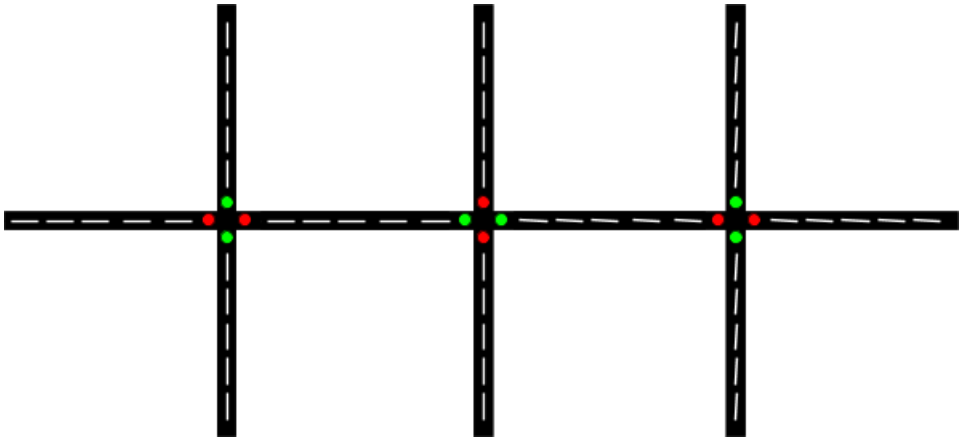

Figure 3.7: Three Agent Traffic Light Scenario.

3.3. 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:

$$\vec{a^*} = (a_1, a_2, ..., a_N) \tag{3.3}$$

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

Another way to approach MAS is to factorise the global Q-function into local Q-functions and then update during training is performed either in coordination with other agents or individually as discussed in subsequent sections. This part of the chapter discusses different ways in which multiple agents can be trained together. Since the training process involves minimising the cost function and updating the actual values, various ways are available in the literature to perform these updates in a multi agent learning process.

3.3.1. 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:

$$Q(\mathbf{s}, \mathbf{a}) = \sum_{i=1}^{n} Q_i(\mathbf{s}, a_i) \tag{3.4}$$

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

$$Q_i(\mathbf{s}, a_i) := Q_i(\mathbf{s}, a_i) + \alpha[R_i(s, a) + \gamma \max_{a_i'} Q_i(\mathbf{s}', a_i') - Q_i(\mathbf{s}, a_i))] \tag{3.5}$$

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.

3.3.2. Distributed Q-Learning

In this approach [22], 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.

$$Q_i(\mathbf{s}_i, a_i) := (1-\alpha)Q_i(\mathbf{s}_i, a_i) + \alpha[R_i(\mathbf{s}, \mathbf{a}) + \gamma \sum_{j \in \{i \cup \Gamma(i)\}} f(i,j) \max_{a_j'} Q_j(\mathbf{s}', a_j')] \tag{3.6}$$

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 [2].

In this chapter we discussed single and multiple traffic intersections, the state, reward function and different actions available to the agent. We also discussed the training process involved in multi agent cases. However, as the number of agents increase new models need to be trained every time. This is highly unfeasible. Therefore in order to circumvent this issue, we introduce different coordination algorithms which is combined with transfer learning approach in a multi agent case.

4

Multi Agent Coordination

In this chapter different ways in which individual agents in a MAS can coordinate are discussed. In multi agent coordination, individual agents contribute to the joint optimal action by selecting their individual action based on their individual payoff function. In our case, the individual payoff function is the individual Q-value function contributing to the global payoff function which is the linear combination of individual payoff function. Here, the three different ways to coordinate in MAS are also discussed. The first one being Individual Coordination which does not guarantee optimal action selection since it has very limited observability of its neighbouring agents. Then we discuss the Variable Elimination(VE) [7] algorithm, this method always leads to optimal joint action, however scales exponentially as the number of agents increases. Finally we discuss,the Maxplus algorithm [10] which is an approximation algorithm for VE, analogous to belief propagation in Bayesian networks [10]. This is an anytime algorithm which can be stopped anytime to give approximate results and thus it is very much applicable in real life scenario.

As stated above, each agent selects its action individually and the joint action is the contribution of each individual action. This joint action may or may not be optimal in nature, since the action of one agent influences other agents. In many real life scenarios, the influence of one agent is however localised. This means that the action of one agent depends on the action of a subset of agents in a MAS. The agents which are located close to each other affect each other more compared to an agent located spatially far from these agents. Therefore in order to define dependencies between subset of agents close to each other, Coordination Graphs(CG) [7][6] are used. Using CGs, we can decompose a global coordination problem into smaller sub-problems.

The problem with Multi agent learning was discussed in the previous chapter. This part of the thesis introduces a variant of Transfer learning, namely Transfer Planning, which circumvents the problem of training the network for multiple agents. It uses the idea of breaking down a multi agent system into smaller sub-problems, then training smaller sub-problems and transferring it to bigger systems. It saves a lot of computational time and expensive calculations.

4.1. Coordination Graph

As discussed in the introduction to this chapter, each agent selects its individual action based on its local Q-function. This local(or individual) Q-function(or payoff function) can be used to compute the global payoff function. A formal representation of the coordination problem is as follows: Each agent selects its action a_i thus, forming a joint action $a^* = (a_1, a_2, ..., a_n)$ for a MAS composed of *n* agents. This gives rise to a global payoff function $R(\mathbf{a})$ for the entire MAS. This global payoff function should not be confused with reward function introduced in chapter 1. The coordination problem is to find a joint optimal action a^* that maximises $R(\mathbf{a})$, thus $\mathbf{a}^* = \operatorname{argmax}_a R(\mathbf{a})$ [7].

A Coordination Graph [6][4] is a graphical representation of the decomposition of the global payoff function($R(\mathbf{a})$) into a set of smaller local factors(f_{ij}), where each factor is a function involving less number of variables, depending on the subset of the agents that the factor represent.

A coordination graph is represented as $CG = (V, E)$, where V are vertices and E are edges as shown in Fig. 4.1. The vertices(or circles) represent the agents and the edges which represent the relation or

dependencies between the agents. The local payoff function is represented by f_{ij}, where $(i,j) \in E$ for agents a_i and a_j are connected via edges.

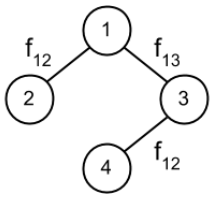

Figure 4.1: Coordination Graph.

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

$$R(\mathbf{a}) = \sum_{(i,j)\in E} f_{ij}(a_i, a_j) \tag{4.1}$$

The joint optimal action is the one that maximises the global payoff function $R(\mathbf{a})$ [6]:

$$\mathbf{a}^* = \text{argmax}_a R(\mathbf{a}) \tag{4.2}$$

Next, we discuss the VE algorithm which makes use of the CG in order to find the joint optimal action.

4.2. Variable Elimination

Variable Elimination(VE) [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 the joint action space, for large scale agents, the problem scales exponentially. Since it is not an anytime algorithm [12](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 for the coordination graph in the Fig. 4.1.

The equation below represents the global payoff function as a sum of local payoffs.

$$R(\mathbf{a}) = f_{12}(a_1, a_2) + f_{13}(a_1, a_3) + f_{34}(a_3, a_4) \tag{4.3}$$

The order of elimination of agents followed is 1,2,3,4. At first we eliminate agent 1. Since agent 1 does not influence local payoff $f_{3,4}$, we apply maximisation step for the other two local payoffs.

$$\max_{\mathbf{a}} R(\mathbf{a}) = \max_{a_2,a_3,a_4} \{f_{34}(a_3, a_4) + \max_{a_1}[f_{12}(a_1, a_2) + f_{13}(a_1, a_3)]\} \tag{4.4}$$

This gives a conditional payoff function function $\phi_{23}(a_3) = max_{a_1}[f_{12}(a_1, a_2) + f_{13}(a_1, a_3)]$ which depends on the action of agent 2 and 3. Since we have eliminated agent 1, the equation 5.3 reduces to:

$$\max_{\mathbf{a}} R(\mathbf{a}) = \max_{a_3,a_4}[f_{34}(a_3, a_4) + \phi_{23}(a_2, a_3)] \tag{4.5}$$

The next agent in the elimination step is agent 2, this gives us equation 4.6 using $\phi_3(a_3) = max_{a_2}\phi_{23}(a_3)$.

$$\max_{\mathbf{a}} R(\mathbf{a}) = \max_{a_3,a_4}[f_{34}(a_3, a_4) + \phi_3(a_3)] \tag{4.6}$$

Finally, we eliminate agent 3, using $\phi_4(a_4) = \max_{a_3}[f_{34}(a_3, a_4) + \phi_3(a_3)]$. Lastly, we select the best action for agent 4 that maximises equation 5.7.

$$\max_{\mathbf{a}} R(\mathbf{a}) = \max_{a_4} \phi_4(a_4) \tag{4.7}$$

VE does not depend on the elimination order and always produces joint optimal action. 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 [29]. VE does guarantee convergence, however it is highly restricted due

to scalability issues. It is computationally expensive as well as time constrained. Hence in order to use it in the real life situations, where we often need to make decisions in a fixed amount of time, this method might not be feasible. Therefore an approximate coordination technique that gives an acceptable solution to the TLC problem is needed. One of these method is discussed in the subsequent section 4.4.

4.3. Brute Coordination

The Brute Coordination(BC) algorithm is a naive approach to coordination in multi agent systems. In this approach, the global Q-function is calculated for all possible combinations of actions for the given number of agents. The global Q-function is the linear combination of local Q-function as given by the eqn. 4.3. The action combination that leads to maximisation of the global Q value is chosen as the optimal joint action to be implemented in the environment. The Algorithm 1 is the implementation of BC. The Q_f is the Q-function for the agents connected by edges in CG which in the algorithm is represented as factored Q-function. The term $a_f^{'}$ represents the action combination of the agents connected with edges in CG, this action is the part of the joint action for all the agents.

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_f(a'_f)
    sum_q_value.append(q_value)
a* = argmax_a' sum_q_value
return a*
```

As is the case with VE, BC also does not scale effectively in terms of computation time when the number of agents increase. This is so because as the number of agents increase, the joint action space increases exponentially. Therefore the algorithm takes more time to try all the combination of joint action in order to find the optimal joint action that maximises the global Q-function.

4.4. Max-Plus Algorithm

The Max-Plus(MP) [10] is an inference algorithm based on message passing. It is used to find the maximum a posteriori (MAP) state in graphical models. It uses messages which are passed over the edges between the connected agents. These messages are essentially the conditional payoff function similar to the one in VE. Messages containing information over locally optimal actions are exchanged between agents iteratively to find the optimal joint action. It is based on a message passing parameter as shown:

$$\mu_{ij}(a_j) = \max_{a_i} \left[f_{ij}(s, a_i, a_j) + \sum_{k \in ne(i) \backslash j} \mu_{ki}(i) + c_{ij} \right] \tag{4.8}$$

The message from the agent i to the agent j is as shown above, where $ne(i)\backslash j$ is the set of $i's$ neighbours, excluding j. This message represents the approximated payoff that i can achieve for a given action of j. It is computed by maximising over the actions of agent i, the sum of the payoff function (f_{ij}) and all the incoming messages to agent i from all its neighbours other than j.

In case of cyclic graphs, the MP algorithm does not guarantee convergence. However they have been applied in TLC problem before [26]. Some issues need to be taken care of in order to apply MP in cyclic graphs. For instance, in a cyclic graph, an outgoing message from an agent i becomes a part of its incoming message. Since the messages are summed up as shown in equation 5.8, this leads to continuous increase in message values. Therefore we add a normalisation constant at every outgoing message to tackle this issue. This normalisation constant is c_{ij} as shown below:

$$c_{ij} = -\frac{1}{|A_j|} \sum_{a_j \in A_j} \mu_{ij}(a_j) \tag{4.9}$$

Still it might be possible that messages do not converge to a fixed point. Therefore, the algorithm needs to receive a deadline signal to report the best optimal joint action found so far. This is useful in situations where there is fixed amount of time to compute action because of the nature of the problem as is the case in the TLC problem.

At the end of exchanging messages for a particular agent i, the best individual action is computed from $a_i' = \textbf{argmax}_{a_i} g_i(a_i)$, where $g_i(a_i)$ is the sum of all messages received by the agent i from all its neighbours. As explained earlier, MP can be used as an anytime extension, that is the algorithm can be used to report the optimal joint action found so far as soon as it receives a deadline signal. In order to implement this, when all agents have exchanged messages, the global payoff is computed based on the local action of the agent. This global payoff function and its corresponding joint action is updated only when it is improved upon during the next iteration steps. As soon as the deadline signal is received, the best optimal joint action so far is reported. The Algorithm 2 represents the pseudocode Max-Plus implementation.

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 $g_i(a_i) = \sum_{j \in \Gamma(i)} \mu_{ji}(a_i)$ 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 $\mathbf{a}^*$

4.5. Individual Coordination

This is a coordination method which involves absolutely no exchange of messages between the agents. Each agent chooses its individual action based on its payoff function, that is the action which maximises its individual payoff function. Thus the joint action is the combination of individually selected actions of each agent. This approach is fast compared to BC or MP, however it does not guarantee convergence.

Algorithm 3: Individual Coordination.

initialise sum_q_value = list[ ]
for *all agent* a_i **do**
  a_i^* = $\mathbf{argmax}_{\mathbf{a_i}'}\ Q_i$
$\mathbf{a}^*$ = $(\mathbf{a_i^*})$
return $\mathbf{a}^*$

4.6. Transfer Planning

In section 3.3, we discussed how we can train MAS by dividing the global Q-function into individual Q-functions and updating them using different techniques. However, this leads to an issue of training every new MAS we come across, that is if the number of agents in a MAS changes we need to train the model again. This becomes cost ineffective and time and computationally expensive. Therefore, an approach to MAS could be to use pre-trained smaller systems and use them in a bigger MAS, along with coordination algorithms to coordinate between these smaller systems in a large MAS. This will reduce the need and effort to rebuild and train models for every new system we come across, thereby reducing expensive calculations, energy wastage and saving time.

Based on the previous work [26][27][19] [18], Transfer Planning(TP) is a good approach for solving Multi-agent systems. In Transfer Planning [19], Q-function is learnt for a subproblem of a larger multi-agent problem. TP steps can be briefly explained as follows:

- Identification of the source task or a number of source tasks which can be transferred: A MAS can have a configuration which requires one or more than one source problem. This depends on the differences between the source tasks in which the MAS can be divided.
- Training the model on the source task/tasks: In order to export the source problem in a MAS, we first need to train them on the source task.
- Effectively transferring the learned source tasks to a bigger problem(target task): This is closely related to the first step. If the source tasks are not identified effectively or when identified but not used properly in MAS, the results can be compromising.

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 a Q-function for each separate subproblem.

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.

4.7. TP with Coordination Algorithm in Traffic Light Problem

In this section, we discuss how can one define a source problem and use them in a MAS with traffic control problem perspective.

In a TLC problem with multiple traffic intersections, we first identify the different sources. For a TLC problem with four intersections, we identity two different source problems used in this thesis. The first one is as shown in the Fig. 4.2. This source problem consists of two agents. Here we have a source problem indicated by the dotted lines. However, the configuration of the two source problem is different, one being horizontal(red) and the other being vertical(blue). Instead of defining these as two different source problems, we train our source problem for either one of the configurations and apply a transformation to use it for the vertical case.

The second source problem that we define for four intersections is as shown in the Fig. 4.3. This source problem consists of a single agent only.

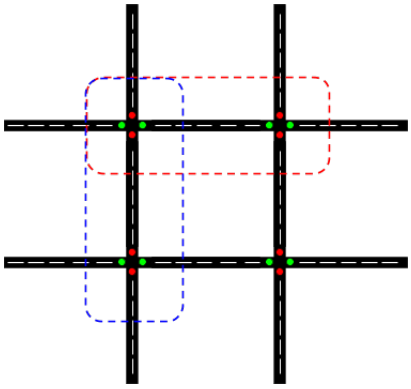

Figure 4.2: Two agent source problem.

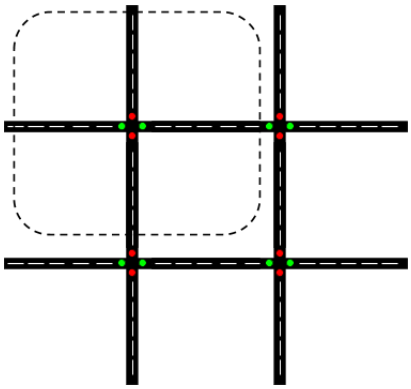

Figure 4.3: Single source problem.

After identification of the source problem, we train our models for these source problems. For a single agent source problem we train the model for a single intersection in order to learn Q-values for its individual action. Whereas, for a two agent source problem, we train the model to learn Q-values for joint action of the two agents together.

The next and the final step is to effectively transfer the learned source tasks to a bigger problem. For the four intersection TLC problem, we load or transfer the source problem for each configuration similar to the source problem. In a two agent source case, the source problems are imported(transferred) for every pair of two intersections as shown in Fig. 4.4. The black and dark blue sources represent the horizontal whereas yellow and cyan-blue source represent the vertical source problem transferred to the four intersection.

In a single agent source case, the source problem is transferred for every traffic intersection as shown in Fig. 4.5.

This completes the transferring of source problems into a bigger MAS. The next step is to coordinate between these source problems using coordination algorithm explained earlier in order to find an optimal joint action.

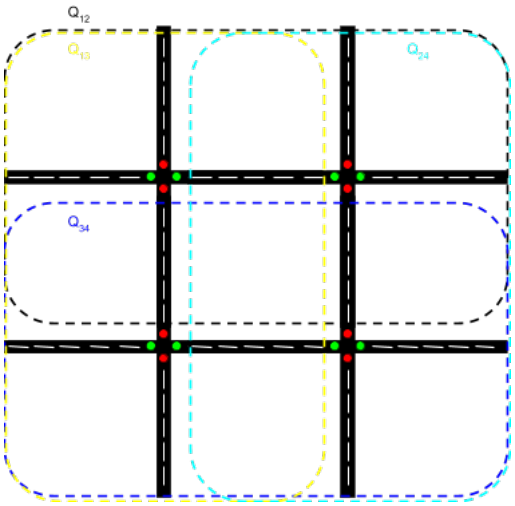

Figure 4.4: Transferred Two source problem in MAS.

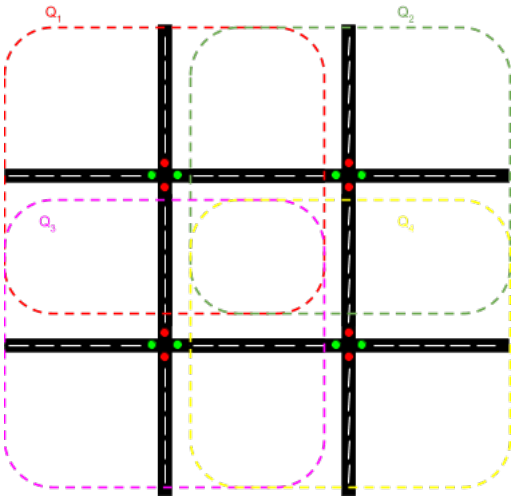

Figure 4.5: Transferred Single source problem in MAS.

5

Experimental Setup

Traffic flow problems can be visualised as a Single or Multi intersection 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) [11] 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.

5.0.1. Network Architecture

The network architecture followed is based on previous work in traffic light problem using DRQN approach [28] [8]. The network architecture represented in Fig. 5.1 is the skeleton of the neural network. The input matrix to the network architecture is the state matrix represented in Fig. 3.3. The height and width of the state matrix represent the rows and columns. Since the training is done in mini-batches of samples, the input matrix to the architecture comprises of one mini-batch of state matrix(that is a fixed number of state-matrix). The subsequent layers are convolutional layers each followed by a ReLU which is then followed by a linear layer. This layer is then followed by LSTM layers which extracts historical information from the sequence of state matrices. Finally the network terminates to approximate Q-values for each of the state matrix in the mini-batch. The width of the Q-value layer is the same as the number of action of the agent, each representing a Q-value for a particular action. Knowing the Q-values for each action for a given state, an action can be selected accordingly.

5.0.2. 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 5.1. Algorithm 4 describes the DRQN approach along with Experience Replay(ER) used to train the agent.

5.0.3. Evaluation

After every 10,000 steps, each model is evaluated by performing a purely greedy policy. Evaluation is done by running 8 SUMO simulations and the results are plotted in terms of the average reward and the average travel time along with shaded region showing two standard deviations. The evaluation method used is depicted by Algorithm 5.

5.0.4. SUMO(Simulation of Urban Mobility)

In order to conduct TLC experiments, SUMO is used. SUMO [11] is free, open source software that allows for a realistic simulation of different traffic networks. It comes with a 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

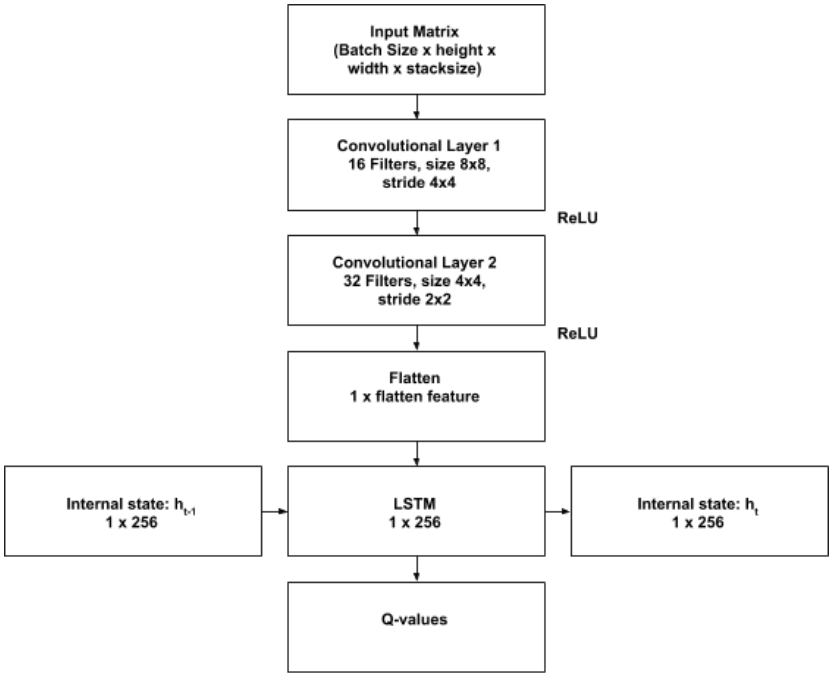

Figure 5.1: Network Architecture.

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 5.1: Parameters used for Training.

like vehicles, pedestrians, public transport as well as cyclists. SUMO is implemented in C++ and only uses portable libraries. The input binary state matrix is obtained based on this environment, and the actions are communicated to SUMO software to implement it in the environment. At every time step of the simulation, we can obtain the total waiting time and the total delay from SUMO, thus building our reward function. However, in order to obtain the total average travel time, we need to wait till the end of the simulation.

5.0.5. Demand Data generation

In SUMO [11] 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 6, the probability q (car probability) is the probability of a car entering the environment at each time step. Therefore, for a given value of q and and total simulation time N, the total number of vehicles entering the simulation environment is $q * N$. This probability q of a vehicle entering the simulation environment at a given time step is used to define the congestion scenario in Section 5.0.6.

Algorithm 4: 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

Algorithm 5: Evaluation Algorithm

Initialise Environment, state s_0
∀ factor $f \in CG$ initialise saved DRQN model Q_f
Define i = total number of simulation per model
while *i* < *total number of simulation* **do**
  for *factor* $f \in CG$ **do**
    get local or factored q-value Q_f
  end
  compute joint action a^* using the choice of algorithm
  for *factor* $f \in CG$ **do**
    take action a_f^*
  end
  Obtain reward r, next state s'
end

5.0.6. 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 [24]. Up to a certain level of vehicles, the traffic flow is smooth without any delays, however with an increase in the number of vehicles above a certain threshold sets in 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, increased travel time or queuing. In order to define traffic congestion for our problem we use one of the definitions available in the literature [3] where, congestion is calculated in terms of increase in travel

Algorithm 6: 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
```

time due to the introduction of additional vehicles in the traffic scenario.

In Fig. 5.2, two plots are shown, one representing the average travel time $t = f(q)$ as a function of different car probability(q)(volume of traffic). This q is same as the car probability defined in the Demand Data generation Algorithm 6. Increasing q means increasing the probability of a car entering the SUMO environment every time step and thus the total number of cars for a given N (total simulation time).

The average travel time is calculated at the end of every simulation. It is defined as the total sum of the time needed by vehicles for route completion divided by the total number of vehicles that entered the SUMO environment during a simulation.

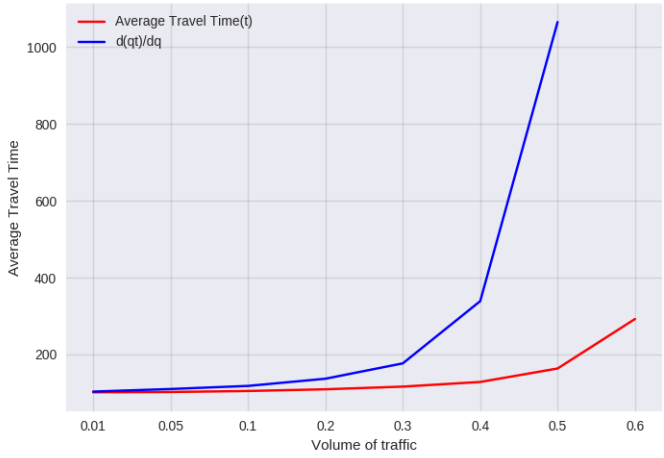

Figure 5.2: Representation of Concept of Traffic Congestion .

Car probability	Congestion
0.05	Low
0.2	Medium
0.4	High

Table 5.2: Different Traffic Congestion or Demand scenarios.

In order to calculate the average travel time for different traffic volumes q, eight simulations were carried out and average travel time(in red) was computed and plotted. The second plot(in blue) is the average travel time due to increase in the traffic volume $\frac{\partial(qt)}{\partial q} = t + qf'(q)$. The term $\frac{\partial(qt)}{\partial q}$ is the change in travel time due to change in the volume of traffic or the car probability q. Since $t = f(q)$ is

a function of q, therefore $\frac{\partial t}{\partial q} = f'(q)$. The difference between the two graphs, indicate the increase in the average 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 an 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 5.2 lists the three different congestion configurations used throughout the thesis. Thus the total number of vehicles entering the simulation environment for the low, medium and high congestion case are 180, 720 and 1440 respectively. Thus there is a four fold increase in number of vehicles from low to medium and two fold increase from medium to high congestion case.

6

Single Agent Experiment

In this section, the experimental results for single agent comprising of one traffic intersection for different congestion is presented. Single intersection is as shown in the Fig. 6.1. The results are expressed in terms of the average reward and the average travel time.

6.0.1. Experimental Setup

The agent is trained for 1,000,000 million time steps and evaluated every 10,000 step by performing 8 simulations. The plots are the average reward and the average travel time of 8 simulations for every evaluated model. The shaded region is the region of two standard deviations.

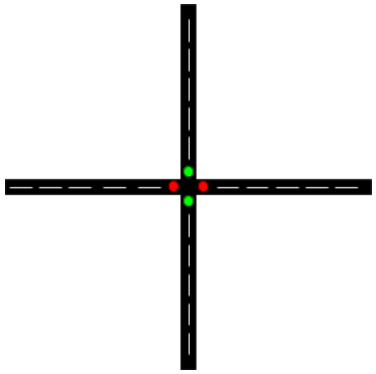

Figure 6.1: Single Agent Traffic Intersection.

6.0.2. Results and Discussion

The model is trained for different traffic congestion as mentioned before. The graphs depicts the fact that for the low congestion problem, the agent quickly learns a good policy and maintains it throughout. As we upscale to the medium traffic demand, the agent learns good policy and maintains it with a slight dip in reward towards the end. The agent still remains stable throughout for medium congestion. However, the stability decreases significantly in case of the high traffic demand. In the high congestion scenario, the agent learns good policy initially and then forgets. It recovers soon and then forgets again. Towards the end of the graph, the agent recovers and maintains it. However, there is an increase in instability when compared to medium and low congestion cases.

The aim of the TLC problem is to reduce the average travel time. The average travel time shown in the graph is the ratio of total sum of time needed by vehicles to accomplish the route by the total number of vehicles. It is not a one to one mapping of the reward function. However, the reward function used is a good estimate of the average travel time as shown in the Fig. 6.2 & 6.3. This is the reason that even though the rewards are same for the medium and high congestion case at the beginning of the training, the average travel time is significantly different for them. From the average travel time plot,

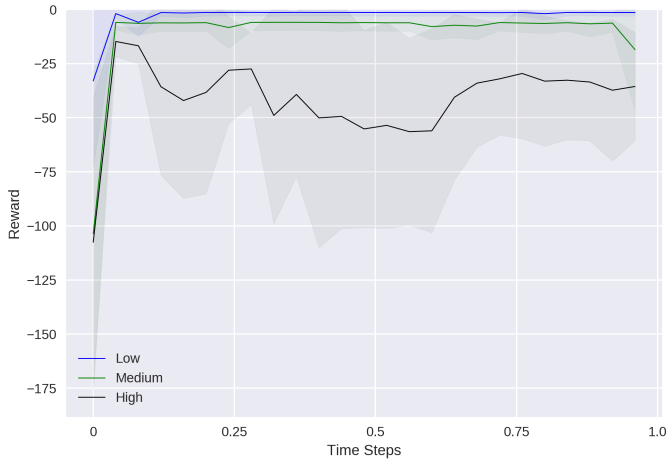

Figure 6.2: Single Agent reward for different Traffic Congestion.

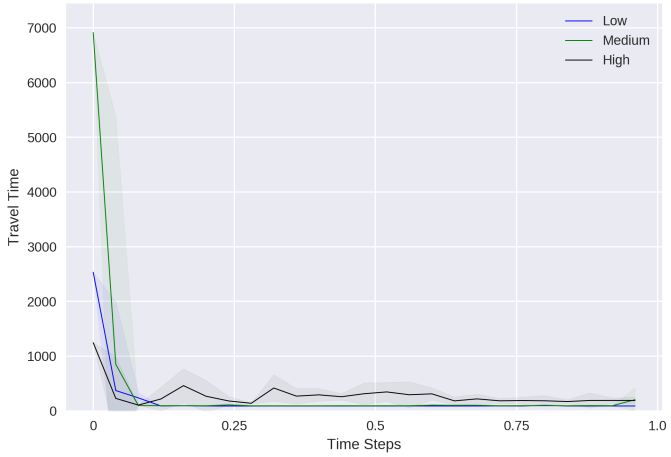

Figure 6.3: Single Agent Average Travel Time for different Traffic Congestion.

it is clearly depicted that for the low and medium case congestion, it becomes constant after the initial training and remains so throughout. However for the high congestion case, there is little fluctuation in the graph initially, but then it becomes constant towards the end. However, it takes more time for the simulation to end in high congestion case compared to the low and medium case since the number of vehicles are more for the latter.

For the low and medium congestion case, the average travel time seem to coincide with each other, whereas for the high congestion is remains above through the graph. This is so because even though there is four fold increase in the number of vehicles from low to medium congestion, the increase in travel time due to addition of extra vehicles in the lane is less when compared to high congestion case, where there is two fold increase in number of vehicles from medium to high congestion. This is depicted in the Fig. 5.2 where the divergence between the graph increases from low to medium and becomes even larger when reaching the high congestion.

7

Multi Agent Experiments

In this section, experimental results for Multi agent cases comprising of more than one traffic intersection for different traffic demands are presented. The performance of different algorithms when accompanied with TP for four and higher number of intersections are compared. The results are expressed in terms of average reward and the average travel time similar to single intersection case.

7.1. Two Intersections

In this part of the multi agent experiment, a scenario with two agents or two traffic intersections are considered. The action space of the agent is bigger when compared to the single agent case, since the agent learns Q-values for the joint action of the two intersections together. In the single agent case, the actions available to the agent were either 0 or 1. In this case the joint action space is $A = \{(0,0),(0,1),(1,0),(1,1)\}$, thus when an agent chooses an action it controls both the intersections together. In other words both the intersections are controlled in a centralised manner.

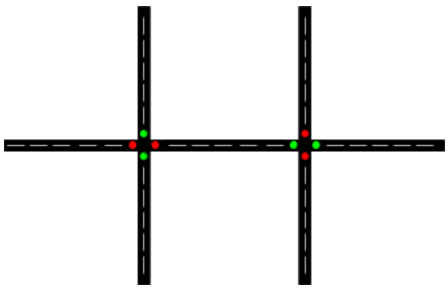

Figure 7.1: Two Agent Traffic Intersection.

7.1.1. Experimental Setup

For all multiple intersection scenarios, the agent is trained for 1,000,000 million time steps and evaluated every 10,000 step by performing 8 simulations. The plots are the average reward and the average travel time of 8 simulations for every evaluated model. The shaded region is the region of two standard deviations. For maxplus implementation, the following Table 7.1 displays the parameters used:

Number of iterations	30
Type of variant used	Centralised with random agent schedule
Noise	Gaussian random noise
Normalisation	Used

Table 7.1: Parameters for Max-Plus Algorithm.

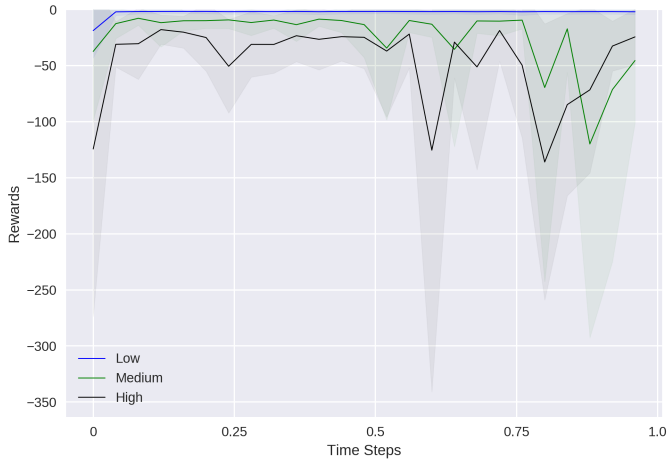

Figure 7.2: Two Agent Average Reward for different Traffic Congestion.

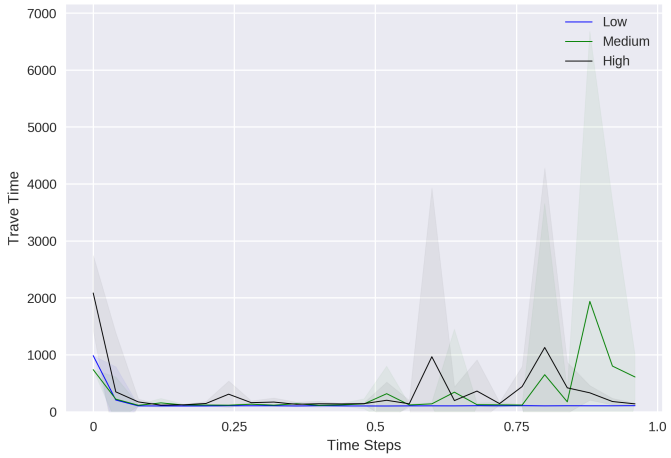

Figure 7.3: Two Agent Average Travel Time for different Traffic Congestion.

7.1.2. Results and Discussion

In case of the two agent scenario, we observe an increase in instability for the medium and high congestion case. The fluctuations are higher towards the end of the graph. For the low congestion case, the agent learns optimal policy and maintains it throughout. The increase in instability can be attributed to multiple reasons. Firstly since the matrix size is kept constant for the single and two agent case, the resolution of the state matrix is decreased. In two agent case, the same state matrix has to account for a bigger simulation environment. Thus there is compression of the environment thereby reducing the resolution of the state matrix. Secondly there is overlapping of binary values of vehicles in each cell due to compression. That is since the resolution is low, each matrix element is no longer a representation of a single vehicle. Instead, a cell can now have more than one vehicle in it. This leads to the violation of our earlier consideration that the state matrix are binary in nature. Therefore, in multi agent cases keeping the state matrix constant and upscaling leads to increase in values in each cell. Thus a single cell in a state matrix can now have values of two or three or even more depending on the number of cars close to each other in the environment. This leads to a complex learning problem for the agent. The agent now has to learn not just the binary values but also learn the different values in each cell.

During the first half of the Fig 7.2, the agent learns optimal policy and maintains it. But during the second half of the graph, it forgets this optimal policy, recovers and then forgets again, and this cycle

continues. The fall in rewards towards the end are even higher than before training. One possible explanation to this could be that when the agent comes across a very different traffic scenario and it forgets previously learnt policies in trying to learn the new scenario.

7.2. Four Intersections

In the four intersection case as shown in the Fig. 7.4, for every pair of agents connected with edges in CG representation(or every traffic intersection connected via a lane), we transfer the pre-trained two agent from the previous two intersection case. For the vertical pair of agents, we transform the pre-trained horizontal intersection by taking transpose of the state matrix. Then we coordinate between each intersection(or vertex in CG) in order to achieve our goal of minimum travel time. For the Individual Coordination algorithm, we transfer pre-trained single agent case for every intersection and use the coordination Algorithm 2.

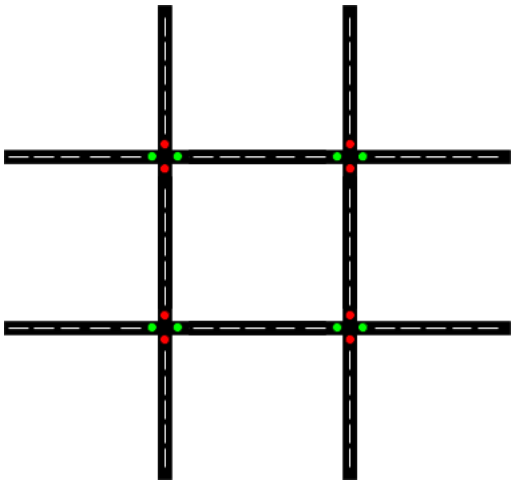

Figure 7.4: Four Agent Traffic Intersection.

7.2.1. Results and Discussion

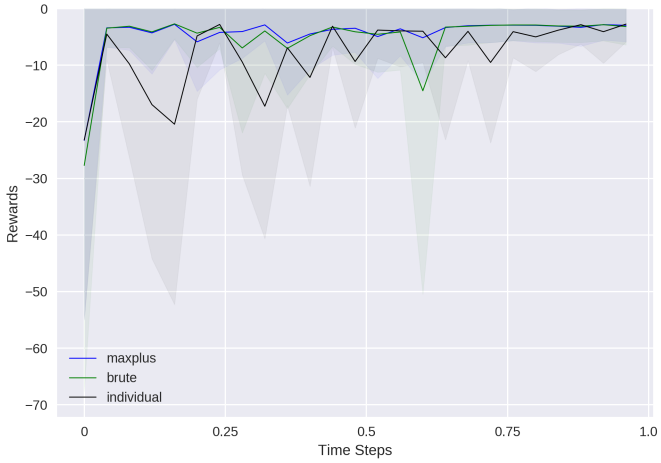

Figure 7.5: Four Agent Average Reward for low Traffic Congestion.

In case of low congestion, coordination using MP and Brute Coordination(BC) gives a stable reward function as compared to the Individual Coordination(IC). The fluctuations in the case of Individual Coordination are comparitively high throughout the training period. This is so because there is abso-

lutely no communication or partial observability between the agents. MP and BC algorithms perform similar to each other and their graphs almost coincide throughout the plot. The messages in case of MP converge and the average difference between the last two messages are between the order of 10^{-5} to 10^{-8}. When comparing the low traffic demand between two and four intersections, none of the algorithms perform as good as the agent in two intersections case. The pattern of the plots for two and four intersections are also very different. While there are fluctuations present in the four intersections, in the two agent case, the agent quickly learns good policies and maintains it throughout. However, towards the end of the plots, the four intersections performance improves and becomes close to the performance curve in the two intersections case. The performance of IC is particularly bad, as is visible from the shaded region for IC. The fluctuations are particularly high during the initial part while towards the end it improves. While for BC and MP, the fluctuations are not as high.

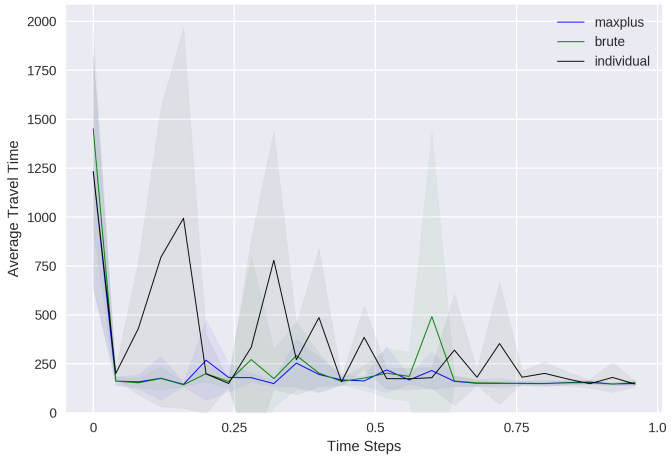

Figure 7.6: Four Agent Average Travel Time for low Traffic Congestion.

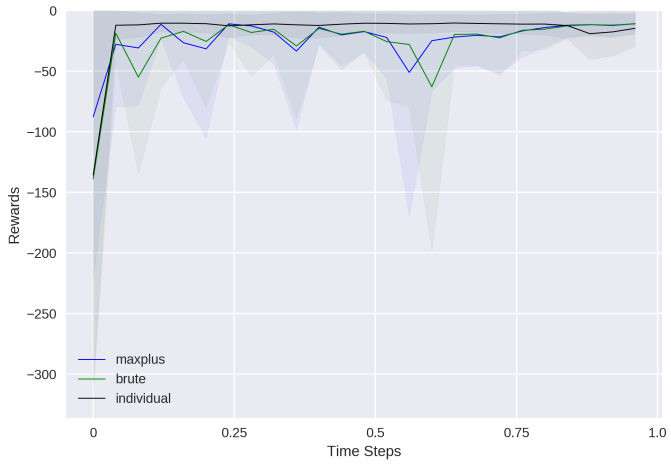

Figure 7.7: Four Agent Average Reward for medium Traffic Congestion.

The reward plot for the medium congestion shows that the IC surprisingly performs very well when compared to the other algorithms. In fact IC performance for four intersection is even better than two intersections case. This is a rarity since each agent's action selection that maximises its Q-function does not guarantee selection of optimal joint action. When comparing the medium traffic demand between four and two intersection, a peculiar behaviour is observed. It is observed that fluctuations for

two and four intersections do not coincide. That is in the two intersections case, the fluctuations are high towards the end, while it is fairly stable for the four intersections. In fact, towards the end for the four intersections case, all algorithms perform very well.

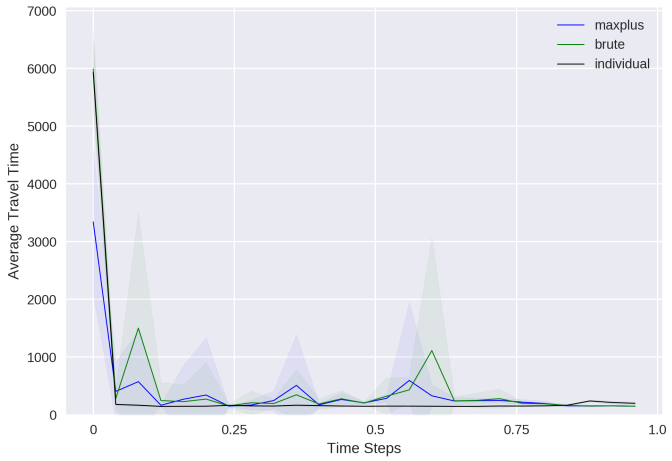

Figure 7.8: Four Agent Average Travel Time for medium Traffic Congestion.

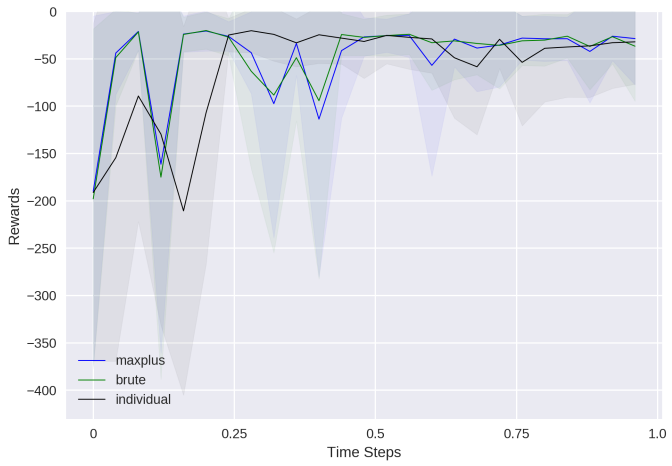

Figure 7.9: Four Agent Average reward for high Traffic Congestion.

In the high congestion case, it is observed that the rewards are highly fluctuating initially but eventually it becomes stable towards the end of the plot for all algorithms. Therefore, it can be inferred that for a given traffic demand, at the end of training period the performance is improved for all algorithms for four intersections.

However, it is also observed that as the congestion increases there is also an increase in fluctuations for all the algorithms. When the high traffic demand plots are compared for the two and four intersections, behaviour similar to medium traffic demand is observed. That is during the initial part of the plot, when the two intersections case is fairly stable, high amount of fluctuations are observed for four intersections. And towards the end when two intersections plot fluctuates, the four intersections plot becomes stable.

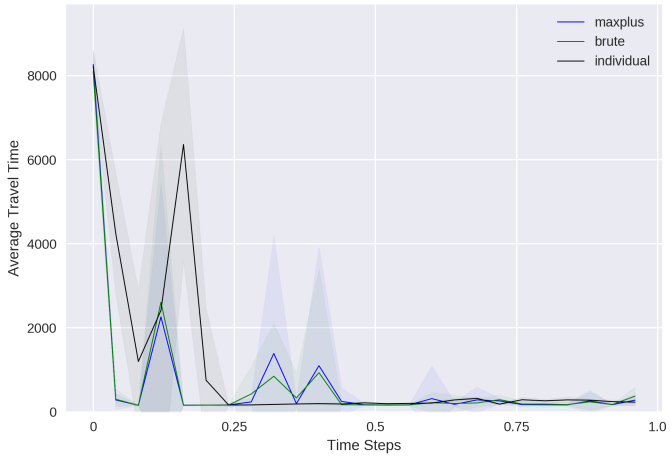

Figure 7.10: Four Agent Average Travel Time for high Traffic Congestion.

7.3. Six Intersections

In this part of the chapter, six intersections scenario is evaluated as shown in the Fig. 7.11. The Transfer Planning approach is similar to the four intersections.

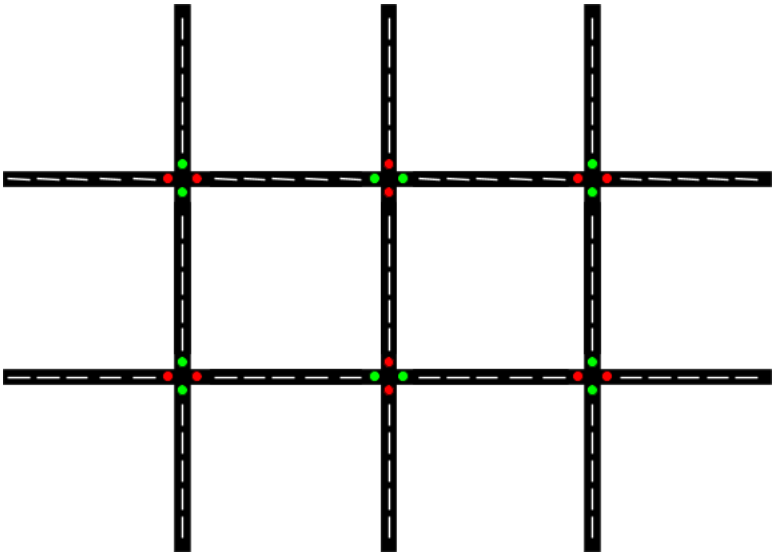

Figure 7.11: Six Agent Traffic Intersection.

7.3.1. Results and Discussion

The six intersections scenario with the low traffic demand, experiences an increase in instability when compared to the four and the two intersections scenarios for BC and MP. The pattern followed when comparing six intersections to two intersections is similar to the one followed between four and two intersections. That is during the initial part of the plot the graphs are highly fluctuating in nature. However towards the end, the plots become fairly stable with the decrease in fluctuations. In case of the IC for low traffic demand, the fluctuations are high during the initial period but eventually the graph becomes stable. For BC and MP, in the mid region of the plot, the performance is worse than IC.

When the congestion is increased from low to medium, there is an increase in average reward and the average travel time, which is expected since the number of cars entering the simulation environment is increased by four times. Unlike the case of medium traffic demand in the four intersections, where

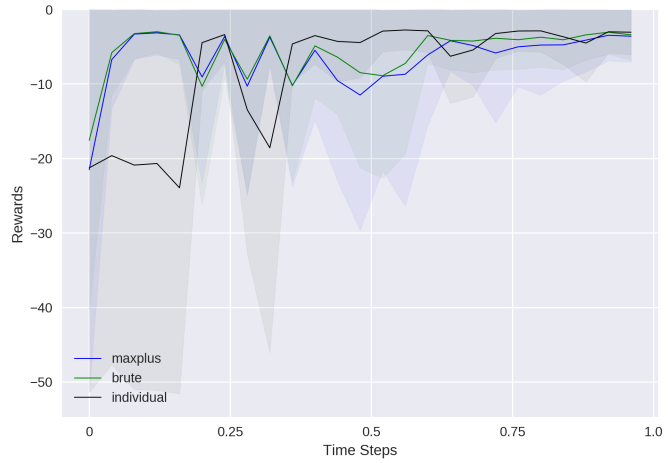

Figure 7.12: Six Agent Average Reward for low Traffic Congestion.

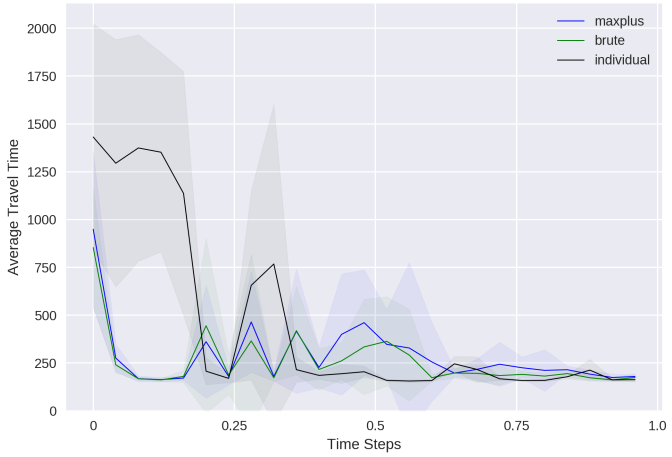

Figure 7.13: Six Agent Average Travel Time for low Traffic Congestion.

the fluctuations eventually become less, the fluctuations here are comparatively higher towards the end. Thus, it does not seem to follow the same pattern. From the Fig 7.14 & 7.15, it is depicted that even towards the end of the plots, the rewards and the average travel time are high and fluctuating. Even though the values of average travel time and the average reward seem to improve at the very end, it cannot be said that on being evaluated further it will remain the same. This is so because even during the initial part of the plot, the values become less, but then they start to fluctuate and increase. Thus, it is difficult to draw inference from the medium traffic demand plots of the six intersections.

Increasing the traffic demand from medium to high, increases the fluctuations in the plots for both the average travel time and the average rewards. The Fig. 7.16 & 7.17 are relatable to its medium counterparts. They behave in the same manner that is they have high fluctuations throughout the graphs. However, in the high traffic demand it is visible that at the very end that the fluctuations increase and the average travel time plots for different algorithms are not low. Towards the end the MP algorithms experiences a sudden dip in rewards, similar to the initial part of the curve. Though MP follows BC plot throughout, it diverges at the very end.

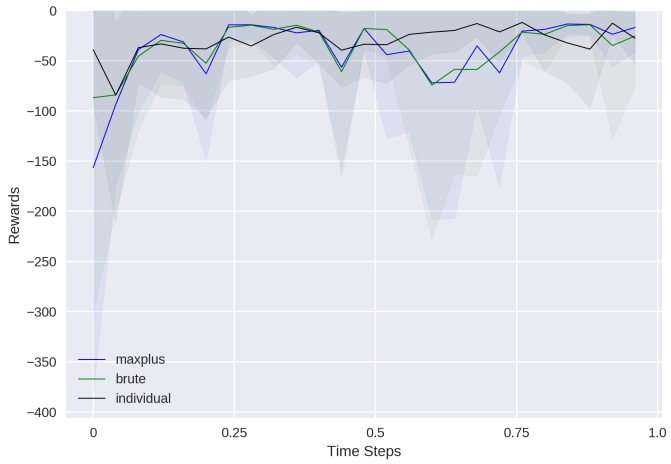

Figure 7.14: Six Agent reward for medium Traffic Congestion.

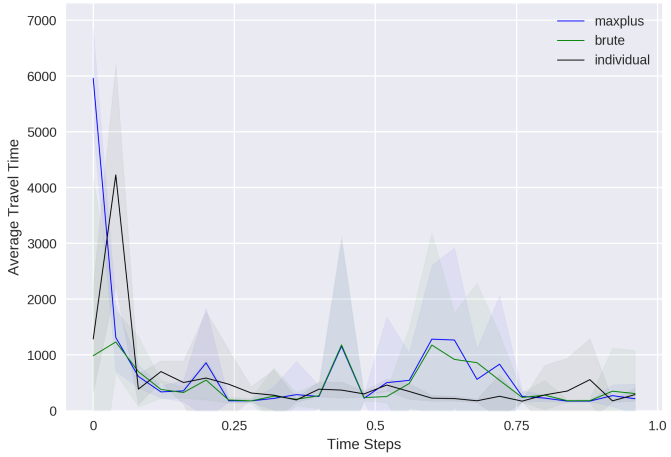

Figure 7.15: Six Agent Average Travel Time for medium Traffic Congestion.

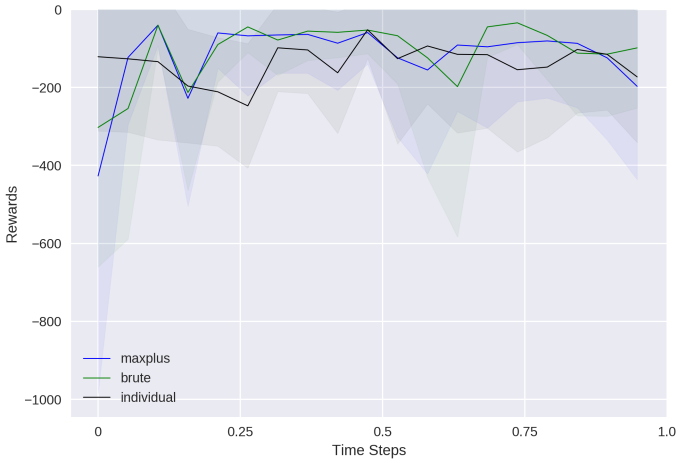

Figure 7.16: Six Agent reward for High Traffic Congestion.

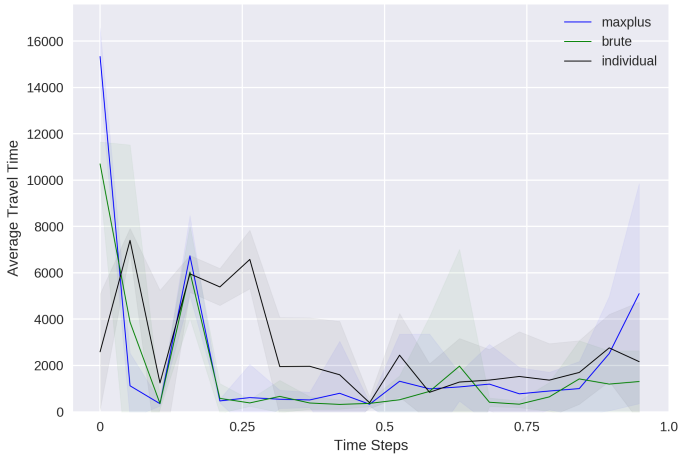

Figure 7.17: Six Agent Average Travel Time for High Traffic Congestion.

7.4. Eight Intersections

The traffic light scenario for the eight intersections are as shown in the Fig. 7.18.

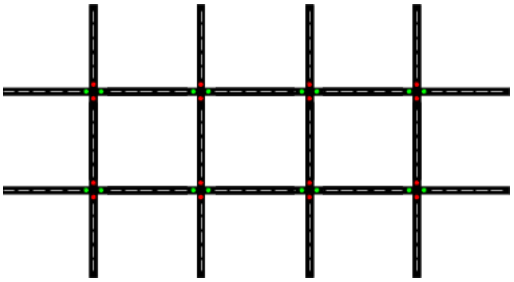

Figure 7.18: Eight Agent Traffic Intersection.

7.5. Results and Discussion

The performance of the eight intersections agent in the low congestion case, when compared to four and six intersections are similar. That is in the beginning of the graphs, there are high fluctuations and towards the end it becomes less. The average travel time and the average reward plots also exhibit improve in the performance towards the end. The pattern followed by the eight intersections plot with respect to two intersections is similar to the one exhibited by four and six intersections. In case of IC, the performance initially is highly fluctuating however as the training progresses, it outperforms both the other algorithms.

With the increase in the traffic demand from low to medium, BC and MP initially performs poorly, however they recover and towards the end of the plot a low average travel time is achieved. The fluctuations are however higher for all algorithms compared to its four and two intersections counterparts. IC performs really well initially but towards the very end there is a sudden increase in the average travel time and the dip in the average reward.

The increase in the traffic demand from medium to high causes the BC and MP plots to be fluctuating in the beginning. However there is some recovery during the second half of the graph. But towards the very end there is an increase in the fluctuations. IC plots also follow the pattern similar to BC and MP.

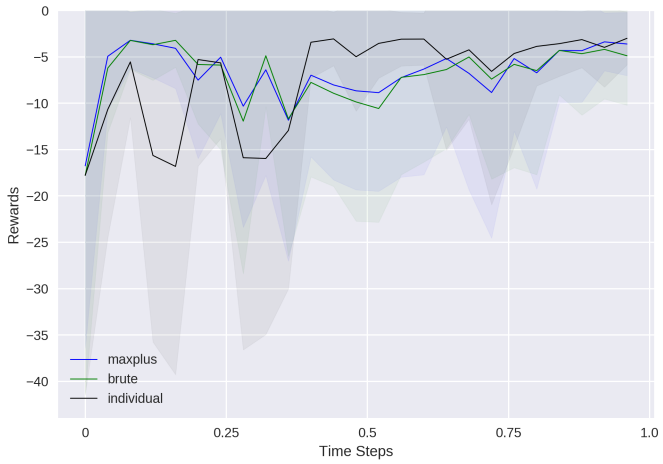

Figure 7.19: Eight Agent Average Reward for low Traffic Congestion.

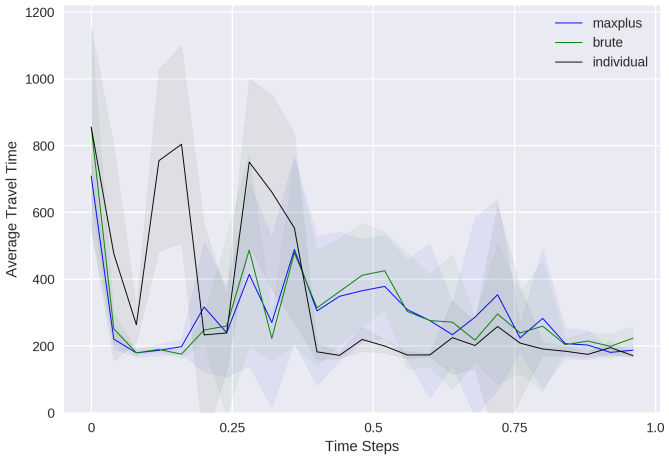

Figure 7.20: Eight Agent Average Travel Time for low Traffic Congestion.

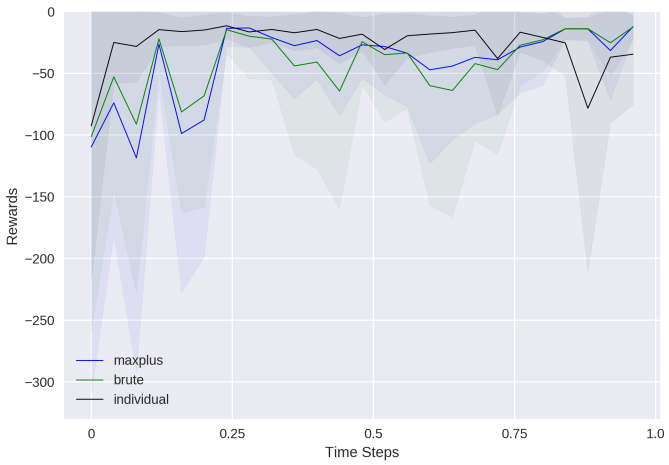

Figure 7.21: Eight Agent Average Reward for medium Traffic Congestion.

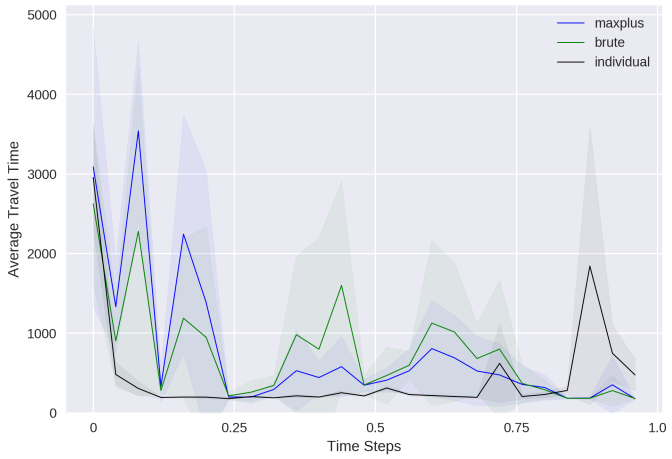

Figure 7.22: Eight Agent Average Travel Time for medium Traffic Congestion.

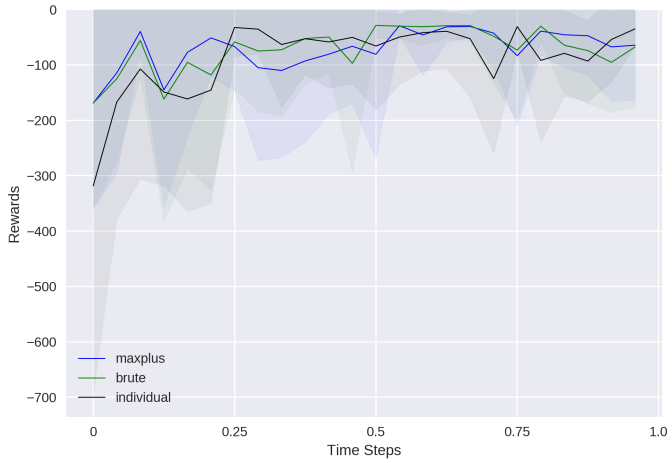

Figure 7.23: Eight Agent Average Reward for highTraffic Congestion.

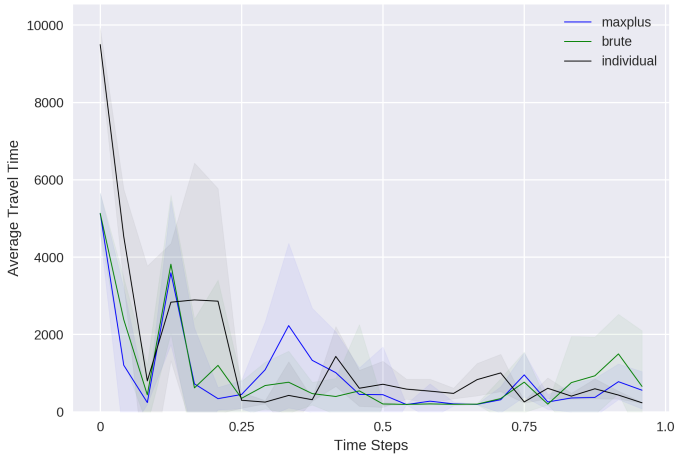

Figure 7.24: Eight Agent Average Travel Time for high Traffic Congestion.

7.5.1. Coordination Algorithm Comparison

It is interesting to study the behaviour of different algorithms for the different number of intersections with different traffic demands. A good way to visualise this is by using a Heatmap. A Heatmap is a way to depict sequential or divergence relationship between multiple variables in terms of colour. The heatmaps are plotted for the different number of intersections on x-axis and different traffic demands on y-axis. The values inside each cell represents the average travel time over all the evaluated models. In the TLC problem, ideally with the increase in the number of traffic intersections or the traffic demand, the average travel time should go up. This is so because with the increase in the number of agents, each agent is prone to less observability (only in the case of IC, since there is no communication between agents) of the entire traffic environment since the number of its indirect agents (agents which are not directly connected) increase. In case of multiple intersections a wrong action selection by an intersection can have a multiplying effect on the entire traffic environment. Thus higher number of intersections are more prone to higher travel times. And also with the increase in traffic demand, we see an increase in average travel time as was shown for different intersections.

In case of Individual coordination, the heatmap justifies our argument that individual coordination does not guarantee optimal joint action selection. Going vertically down (or with increasing congestion) for the four intersections case, first the average travel time decreases for medium traffic demand and then it increases for high demand. This was also depicted in the Fig 7.7 where IC performed better than other algorithms. For the six and eight intersections scenario, an expected trend (increase in average travel time when going vertically down) is shown by IC. However, going horizontally (along the fixed traffic demand), a pattern is observed. That is first there is an increase in average travel time from four to six intersections and then there is a decrease moving further from six to eight intersections. The Table 7.2, contains the standard deviations for the IC heatmap.

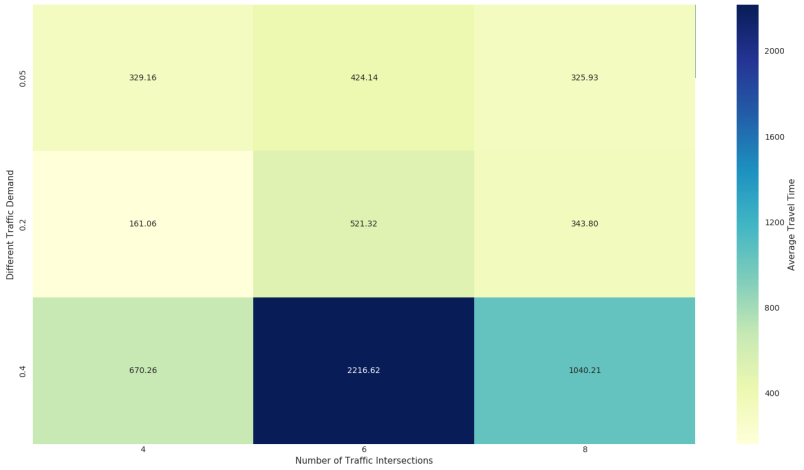

Figure 7.25: Heat Map for Individual Coordination for different traffic intersections with different demands.

Traffic Demand	Four intersections	Six intxersections	Eight interseections
0.05	287.17	454.72	230.4
0.2	22.91	848.9	395.1
0.4	146.02	2158.03	1253.8

Table 7.2: Standard deviation for Individual Coordination Heatmap.

The Fig. 7.26 shows the heatmap for BC along with its standard deviations in Table 7.3. Going vertically down, the average travel time increases for a given intersection, and so does the standard deviations. Going horizontally along the increasing number of intersections, a pattern similar to IC is observed only for the high traffic demand case. While for low and medium demands, there is an

increase in the average travel time.

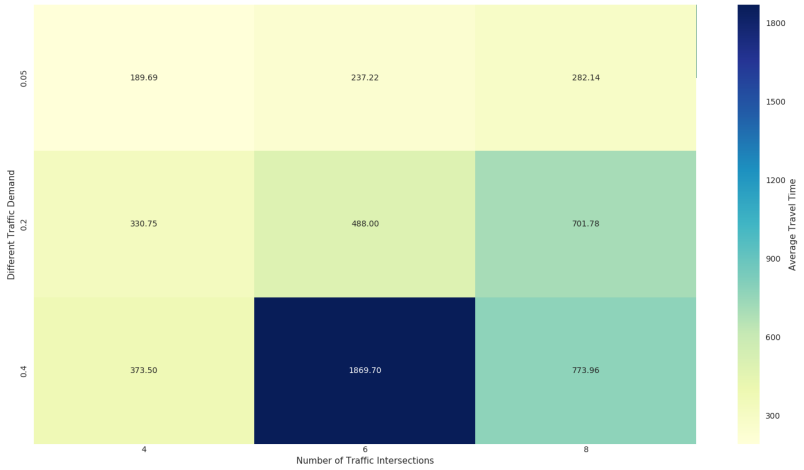

Figure 7.26: Heat Map for Brute Coordination for different traffic intersections with different demands.

Traffic Demand	Four intersections	Six intxersections	Eight interseections
0.05	133.4	118	115.1
0.2	429.4	503	636.82
0.4	697.26	2293.73	949

Table 7.3: Standard deviation for Brute Coordination Heatmap.

The heatmap for Max-Plus algorithm is shown in the Fig 7.27. The vertically downward trend is as expected. Moving horizontally along the low traffic demand sees an increase in the average travel time as the number of intersections increase. This pattern is also followed in the medium traffic demand. Whereas for the high traffic demand, pattern followed is similar to BC and IC. That is with the increase in the number of intersections from four to six, there is an increase in the average travel time. But going from six to eight intersections, there is a fall in the average travel time. Since the Max-Plus is an approximate algorithm for BC, they follow the same pattern. However this peculiar pattern is only exhibited by the six intersections case. This can also mean that with the increase in the number of intersections, the TP approach with coordination algorithms may or may not perform well.

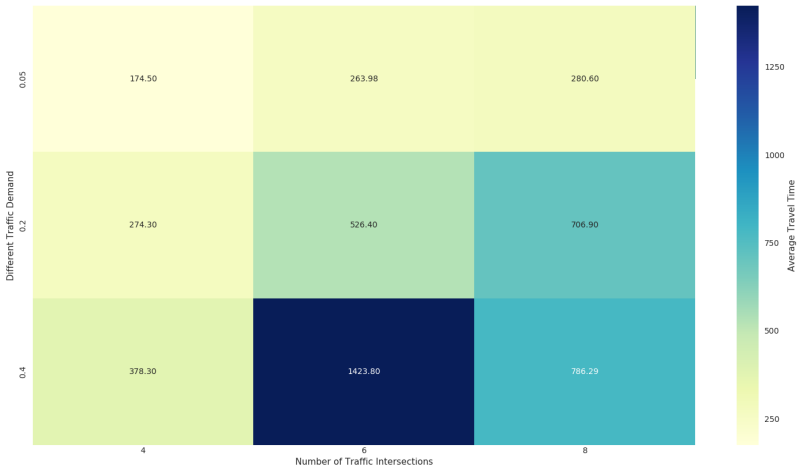

Figure 7.27: Heat Map for Max-Plus for different traffic intersections with different demands.

Traffic Demand	Four intersections	Six intxersections	Eight interseections
0.05	46.6	128.7	114.9
0.2	255.1	528.4	898.61
0.4	697.47	1837.2	908.1

Table 7.4: Standard deviation for Max-Plus Heatmap.

Thus the assumption that higher number of intersections will increase the average travel time is falsified. When the heatmaps as well as the plots for BC and MP are considered, it is observed that both the plots follow each other closely. Since MaxPlus is an approximate method for coordination, which circumvents the task of listing all possible joint action as in case of BC, it should not perform better than BC. Since both the algorithms are being evaluated for the same evaluation seed for route generation, MP should perform at best as BC and not better. This suggests a bug in the code for MP. In order to investigate this, both BC and MP were subjected to same set of Q-values, and the action and the global payoff computed by both were reported. For all such experiments the global payoff for both the MP and BC coincided. And the actions reported by them were same. To be more sure the same experiment was performed for different number of agents, and similar behaviour was observed. Thus it is difficult to draw any conclusion, however it maybe due to the internal working of SUMO.

8

Algorithm Performance Analysis

The Max-Plus algorithm in one iteration, for each agent i computes μ_{ij} and sends it to all its neighbours $j \in \Gamma(i)$. This process continues until all messages are converged, or a 'deadline' signal is received and the current joint action is reported. In case of anytime extension, the current computed joint action is used after every iteration to find the global Q-value. If this global Q-value is found to improve then the best action is updated as the current computed joint action. Brute coordination on the other hand lists all the possible joint actions and computes the corresponding global Q-function. The joint action that maximises the global payoff is reported as the optimal joint action. In case of Individual Coordination, individual actions(a_i) which maximise local payoff function (Q_i) is selected for each agent, then the joint action is reported which is the combination of these actions.

The MP algorithm implementation has complexity $O(mnd^3)$ where m is the number of iterations, n is the number of agents and d is the average number of neighbours per agent. While for BC, the complexity is $O(2^n p)$, where p is the total number of edges in the CG or the total number of lanes connected with traffic intersections at either end. For Individual Coordination, this is $O(n)$.

The following are the attributes of the traffic light intersection used to arrive at the complexity notations:

- The total number of actions per agent is equal to 2, that is *rGrG* or *GrGr*.
- The local Q-function(for BC and MP) Q_{ij} is the joint Q-function comprising of only two agents i and j.

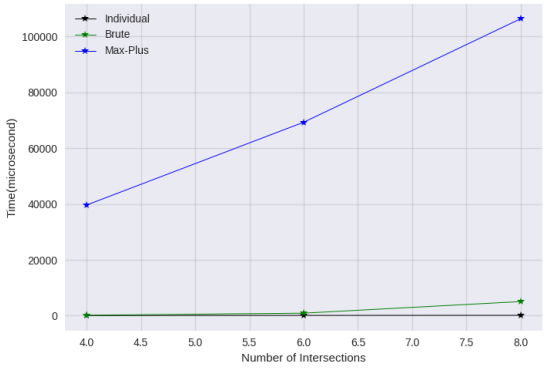

Figure 8.1: Timing results for different Algorithm.

In the implementation of MP algorithm, the total number of iterations performed were 30. After 30 iteration, the best joint action found so far was reported. The Fig. 8.1 shows the timing result(on linear scale) for different algorithms as the number of intersections increase. From the plot, it is clear

that BC outperforms Max-Plus algorithm in terms of computation time. There is a negligible increase in computation time between BC and IC as the number of intersections increase from four to six. Whereas there is a slight increase from six to eight intersections. However MP consumes large amount of time for all the intersections. While IC is the fastest it fails to produce good results compared to the others. The difference between IC and BC will become more prominent as the number of intersections are increased.

Number of intersections(n)	Average number of neighbours per agen(d)	Total possible joint action
4	2	2^4
6	2.33	2^6
8	2.5	2^8

Table 8.1: Attributes of Different intersections.

For the linear arrangement of traffic intersections, in which there is a single chain of traffic intersections connected one after another, the average number of neighbours per agent can be calculated using the formula $\frac{(n-2)*2+2}{n}$ and this changes the Big-O complexity for MP to $O(mn)$. Whereas for the distribution of traffic intersections used in this thesis, the average number of neighbours per agent can be calculated using the formula $\frac{(n-4)*3+8}{n}$. This also changes the Big-O complexity for MP to $O(mn)$. For the arrangement of traffic intersections used in this thesis p varies according to the table 8.2.

Number of intersections(n)	Number of Edges(p)
4	4
6	7
8	10
10	13
12	16
14	19
16	22

Table 8.2: The number of edges(p) with respect to number of traffic intersections(n).

From Fig. 8.2 to 8.5, the complexity graphs are plotted on log scale for different number of intersections(with similar arrangement to the one used in this thesis) with different number of iterations for Max-Plus algorithm.

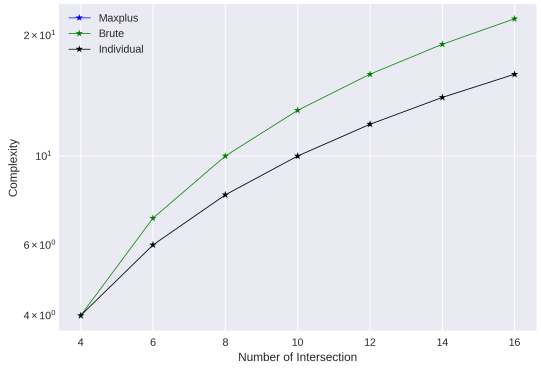

Figure 8.2: Complexity graph for different Algorithms with 1 iteration for Max-Plus.

It can be observed that for one iteration MP plot coincides exactly with IC. In this case both algorithms are linear in nature with complexity $O(n)$. In case of four intersections($p = 4$), all the algorithms are coincident. But with the increase in the number of intersections the plots begin to diverge.

In case of two iterations for MP, a significant change can be observed for all intersections between

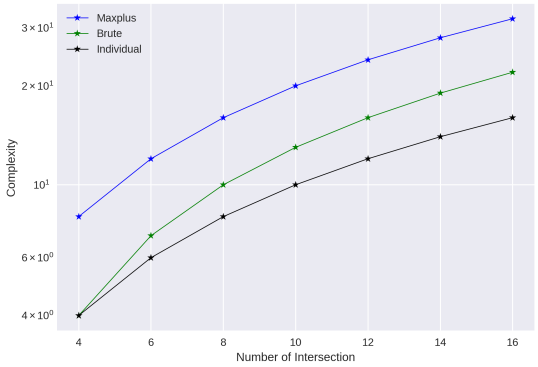

Figure 8.3: Complexity graph for different Algorithms with 2 iteration for Max-Plus.

MP and other algorithms. From Fig 8.4 and Fig. 8.5 it is observed that the difference becomes even larger as the number of iterations increases.

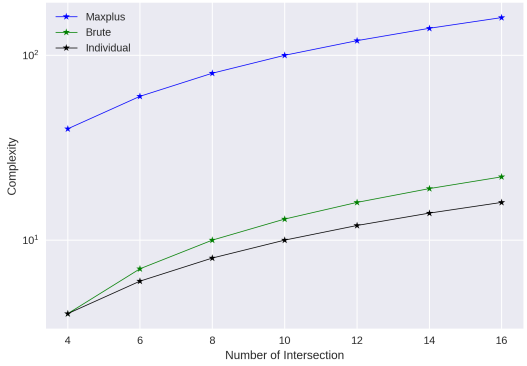

Figure 8.4: Complexity graph for different Algorithms with 10 iteration for Max-Plus.

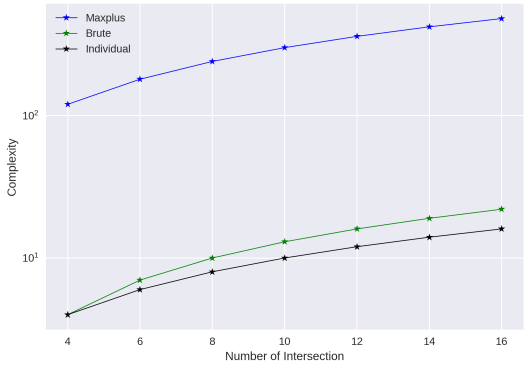

Figure 8.5: Complexity graph for different Algorithms with 30 iteration for Max-Plus.

9 Discussion

The Performance of DRQN seems promising for multiple intersections with different traffic demands coordinating using different algorithms. However there are still some drawbacks which needs to be improved upon. Here the results are discussed and possible outcomes are enlisted.

First the performance of multi agent case with different traffic demand is discussed. In case of IC, the performance is unpredictable in nature. For the four intersections scenario it performs comparable to MP and BC towards the end of the plots for all traffic demands. However it is fluctuating in the beginning, but towards the end the performance improves considerably well. This is also true for the six intersections scenario with low demand. However for the low and medium demands, the plots do not show any considerable dip or improvement throughout despite an initial improvement. For the eight intersections with low demand, there are fluctuations in the initial part of the plot, however during the second half there is a considerable improvement in its performance. In the case of the medium demand, there is an improvement in performance which is maintained throughout. However towards the very end, there is an increase in the fluctuations. Finally for the high demand case, there are high fluctuations in the beginning, and then the agent learns a good policy. And then forgets it again towards the end.

The performance of the BC and MP are mostly similar despite the nature of traffic intersections being cyclic. In case of the four intersections with low demand there is an initial improvement in the performance and this is maintained throughout the training period. However in the case of medium and high demands, there are initial fluctuations present(higher for the former) but towards the end there is a recovery and the performance is improved. Similar behaviour is observed for the six intersections with low demand. Whereas for the medium and high demands, the fluctuations are present for most part of the graphs. In case of the eight intersections, the low demand plots are similar to the four and six intersections scenarios. The plots for the medium demand case are similar to the low demand case. With the increase in congestion to high demand, there is an improvement in performance with respect to the initial part of the plot. However, this improvement is accompanied by fluctuations throughout. The forgetfulness of the agent can be due to the fact that when it encounters a very different scenario which it has never experienced before. In trying to learn the new experience, older learnt policies are forgotten. This phenomenon is known as catastrophic forgetting [16].

Thus for all the intersections with low demand, the plots for BC and MP improves as the training continues. And the increase in the traffic demand and the number of intersections causes an increase in the fluctuations. A possible reason can be the decrease in the resolution of state matrix with the increase in the number of intersections.

It is also seen that the behaviour of multi-agent intersections can be very different from their source problem. Thus the combination of Transfer Planning with coordination algorithms can perform very different than their source problem.

Lastly, the performance analysis of algorithms were investigated. It showed that the BC, MP and IC exhibit the Big-O complexity of $O(mnd^3)$, $O(2^n p)$ and $O(n)$ respectively. The performances are dependent on the factors like the number of iterations performed, the number of traffic lanes with intersections at either end (or edges in CG) and the average number of neighbours per agent. For maximum of one iteration, the complexity plots for MP and IC are coincident, whereas it is higher for BC. And when the

number of iterations are increased, the MP time performance deteriorates. However, the performance of MP can be studied under lower number of iterations for different traffic scenarios. However, the actions may or may not be optimal.

10 Conclusion

The Traffic Light Control problem was studied for different traffic demands for multiple intersections using different coordination algorithms.

- *How does the Transfer Planning approach combined with the Max-Plus or the Brute Coordination algorithm perform when compared to the Individual Coordination?*
  The Transfer Planning approach significantly saves computational cost and time. It is fruitful when combined with an optimal coordination algorithm. However Individual Coordination lacks observability of the global environment. It acts independently and there is no sort of communication between the direct or indirect neighbours. For most cases towards the end of the training period its performance improves. However its performance is still unpredictable. The performance of Brute Coordination and Max-Plus are similar to each other and they also possess a consistent behaviour unlike IC. The Max-Plus algorithm has better observability than IC due to message passing mechanism. While the BC selects the joint action corresponding to the maximum global payoff.

- *How does the TLC agent perform for different traffic demands?*
  For a given number of intersections, the increase in the traffic demand causes an introduction of fluctuations. There is an increase in the average travel time and the dip in rewards as the traffic demand increases. The increase in fluctuations when the demand increases varies for the different- number of intersections.

- *How does the different coordination algorithms perform computationally in case of Traffic Light Control problem?*
  The Individual Coordination performance is exceptionally fast, followed by Brute Coordination and then Max-Plus. Despite being fast, IC has unpredictable behaviour and sometimes it can perform better than any other algorithms. While in other cases its performance is compromised. The Max-Plus algorithm performance is dependent on the total number of iterations performed. However for low iterations, the MP can outperform BC, however the resulting actions may or may not be optimal.

10.1. Future Work

The area of Deep Reinforcement Learning is an upcoming field and a lot more exploration still needs to be done. With the recent application in the field of TLC, there is a lot of motivation to further explore this field. The thesis opens up a few more ideas that still needs to be investigated. These ideas are suggested below.

Higher training time: In this thesis, some results show that towards the end the performance of the agent is improving. Therefore, it is worth trying to train the model for higher time steps and observe their performance.

Increase in the Resolution of State Matrix: It has been observed that keeping the size of state matrix constant while upscaling the number of traffic intersections, relevant information is compressed. While in previous work [26], increasing the resolution has lead to better performance. It would be interesting to see the implication of increased matrix size for different demand scenarios.

Distral(Distill & transfer learning): In this approach a joint objective function is used to optimise the model. This method aims to find a distilled policy π_0 by regularising the task specific policies π_i. Therefore an agent can be trained to learn policy π_0 (which can be followed for all congestion scenarios) with regularisation of π_i(demand specific policy).

Multi Task learning: For every congestion scenario, the agent had to be trained. This leads to an increase in computational costs and well as time. It would be very efficient to train a single model that learns to perform in different congestion cases.

Curriculum Learning: This is an approach which can be applied in case of training the agent for the high congestion case. It was observed from the single agent high congestion plot that the learning is fairly unstable. This method can circumvent this problem by training the agent for smaller congestion scenarios first and then gradually increasing the congestion level.

Coordination Algorithms with better convergence properties: There exists algorithms that have better convergence properties for cyclic graphs. A tree based reparameterization [30] method that combines the exact solutions of different cycle-free subgraphs.

Double DQN: As it was observed from the results that with the increase in demand, the fluctuations increase. This may be caused by overestimation of Q-values. Therefore using two separate networks for action evaluation and action selection can be helpful in reducing fluctuations. This method has shown to improve performance.

Bibliography

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

[2] 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.

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

[4] Jeancarlo Josué Argüello Calvo and Ivana Dusparic. Heterogeneous multi-agent deep reinforcement learning for traffic lights control. In *The 26th Irish Conference on Artificial Intelligence and Cognitive Science*, pages 1–12, 2018.

[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] 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.

[12] 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.

[13] 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.

[14] 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.

[15] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[16] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

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

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

[19] 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.

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

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

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

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

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

[25] 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.

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

[27] 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.

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

[29] 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.

[30] Martin Wainwright, Tommi Jaakkola, and Alan Willsky. Tree consistency and bounds on the performance of the max-product algorithm and its generalizations. *Statistics and computing*, 14(2): 143–166, 2004.

[31] 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.

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