University of The West of England

School of Computing and Creative Technologies - CATE

CATE Internship: Multi Agent Reinforcement Learning for Traffic Light Control and Optimisation

Joe Lyall
Artificial Intelligence MSc

Supervisor

Marco Perez Hernandez

July 11, 2024

1 Abstract

This paper proposes Multi Agent Reinforcement Learning as a solution for the problem of Traffic Light Control and Optimisation. Traffic Lights are still seen to cause unnecessary build-ups of traffic in day-to-day journeys. If traffic lights in a given local network could learn from their common daily traffic flows and cooperate with others nearby then they can synchronise their light signals to allow for a smoother journey for everyone involved at all times. The problem was modelled using a custom environment using the help of graph theory and Python. It can help model current issues with Traffic Light Control, such as Observability and scale. The environment has multiple algorithms attempt to optimise it, MADDPG and PPO. Both are tested on the environment, with differing observations and numbers of agents. MADDPG performs better than PPO, reaching convergence on most tests.

2 Introduction

Traffic light Control is a well researched problem, with many differing approaches to the Optimisation of it. Despite this, still large queues often form at traffic lights unnecessarily, wasting precious time and increasing the chance of accidents on our roads. Optimising traffic light control could reduce future congestion and lower the chance of accidents making journeys smoother and keeping traffic flow steady.

In research, Multi Agent Reinforcement Learning has been considered as a possible solution, but it has not been backed by enough evidence to be taken seriously by industry. Not enough research has looked into the possible effect of comparing traffic lights having full or partial observability of the rest of their network. For example, it may be that in a large network of traffic lights, only the nearby lights and their traffic flow may affect others' decisions. A change in observability may affect the performance of the traffic light in optimising its traffic flow.

Another challenge which must be tackled is scale. Optimising Traffic signals becomes exponentially a lot more difficult to control with an increasing number of traffic lights in the network. However, in big cities across the world, these large networks are prominent and everyones time would not be wasted stuck in a city traffic jam. This is not an issue that can be ignored for its difficulty and must be tackled head on.

This paper will present Multi Agent Reinforcement Learning as a solution to Traffic Light Control and Optimisation, detailing the chosen methodology, before going on to evaluate how MARL is implemented, and testing some algorithms on differing observabilities over a range of tests.

3 Literature Review

[11] proposes an open source implementation of using reinforcement learning for traffic light control. It wraps the SUMO traffic control environment to test on it and tests a range of MARL algorithms on the environment. The tests were average and maximum waiting times of vehicles at a traffic light, average and max number of vehicles per lane, and training efficiency. PPO performed best in every test, proving why it is regarded as the state of the art in the field and showing that Multi Agent Reinforcement Learning can be used on traffic light control.

Another environment which models city traffic in a large scale is called CityFlow. It introduces a new environment for multi-agent Reinforcement Learning and is based off of traffic simulation benchmark SUMO. It uses pybind11 to link C++ code to a pythonic interface. It uses linked lists for representing the amount of traffic in each lane and a similar concept is used in this report to model traffic flow. They divide intersections up into segments and the crosspoints of the intersections are responsible for identifying any cars within them and then deciding which vehicle should go in its desired direction. Vehicles must follow two rules: fully stop at red signal, and yellow if possible. They develop a priority order, so straight moving vehicles get priority over vehicles that want to turn across them. These features, among others, help CityFlow [10] successfully model large scale traffic flow.

[7] tests Multi Agent Reinforcement Learning for traffic signal control using the VISSIM traffic simulator developed by PTV-tag. They tested multiple different algorithms on a 9-junction and a 20-junction road network, with the best performer being a version of Q-learning called Q-UCB, which computes the action to be selected based on the upper confidence bound index. The index is constructed as a sum of two terms.

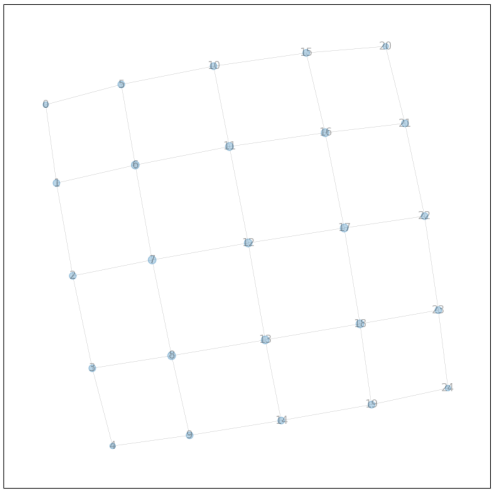

Figure 1: Example shape of a graph that is analysed

The first term corresponds to the learnt Q-function for the state-action pairs. The second term is inversely proportional to the number of times an action has been chosen in a given state.

[5] tests multiple algorithms against a new algorithm that they propose, called co-DQL. This algorithm introduces the idea of mean-field learning. This takes into account that, as the agent numbers increase, it is not feasible for a Q learning algorithm to calculate the join action function for each agent, as it normally does. The core idea is that the interactions within the population of agents are approximated by those between an agent and the average of its neighbouring agents. [1] uses a similar algorithm for traffic signal control.

[2] test a Policy iteration algorithm using the simulation environment CORSIM, developed by the Federal Highway Administration. It was found that the policy iteration algorithm performed better than its counterpart, a fixed-time control algorithm. These tests were run on a 9 light grid network.

4 Methodology

4.1 Problem Specification

The problem that this paper attempts to tackle is a Traffic Light Control Optimisation problem. The problem is that traffic lights often can be out of sync with those nearby, causing unnecessary build-up of traffic. If traffic lights in a given local network could learn from their common daily traffic flows and cooperate with others nearby then they can synchronise their light signals to allow for a smoother journey for everyone involved at all times.

4.2 Traffic Light Environment

The environment that has been built to model this problem draws from graph theory in order to accurately model a traffic light network. In this case, one agent is one traffic light, which is one node on the graph. Each node is joined by an edge, which represents a section of road in between 2 traffic lights. The graph that is used can have differing topologies. When the number of agents is a square number, the graph takes the shape of a lattice, or grid. This has been chosen as the only topology which is tested in this paper, because it can best represent an intersection, which is the shape of traffic light network which causes the most traffic related issues. An example of this is seen in figure 1.

Within this environment, the possible states are a vector with 3 possible values, representing the 3 different light states of the traffic light. The length of the vector for the state space can change depending on the amount of observability each agent has. With full observability, each agent can see all nodes, and with partial observability, each agent can only know about its immediate neighbours. The only possible actions are stop and go. This is because these are the only actions a vehicle will take when encountering a traffic light.

Each edge connecting 2 nodes on the graph has a traffic flow value assigned to it in this environment. The reward function rewards a pair of agents, or nodes, when the road or edge connecting them both has a traffic flow of less than 3.

$$X < 3 = Y + 1 \tag{1}$$

This equation shows the reward function. X represents the traffic flow value assigned to an edge and y represents the corresponding reward for the two agents, or nodes, on each end of the edge. This reward function should help encourage the traffic light network to work together with its neighbouring agents, changing their light states according to one another in order to keep their traffic flows down.

This structure for modelling the environment, with a state to observe, an action, and a reward, follows the logic of a Markov Decision Process(MDP). An MDP is a set of probabilities predicting the likelihood of every single possible outcome for each agents possible action. Each agent is attempting to maximise its expected sum of discounted rewards. It is not always true to assume that the agent can see all of the state, or environment, at all times. Therefore, an adjusted approach is known as a Partially Observable Markov Decision Process(POMDP). Players, or agents, obtain only a partial observation of the state, defined by an observation function. [4] This is the benchmark framework for MARL algorithms.

4.3 MARL as a solution

A Reinforcement Learning(RL) algorithm is a well researched solution to an MDP or POMDP. Algorithms interact with and optimise these environments, using Markov Decision Processes. Multiple agents all acting in parallel, on the same environment, is known as a Markov Game, or Stochastic Game. Multi Agent Reinforcement Learning (MARL)algorithms are designed for solving these Stochastic Games. [6]

MARL algorithms can be value-based or policy-gradient algorithms. Value based methods have been the standard approach to Reinforcement Learning problems from the start. This involves each agent individually optimising the state-action value function, which is known as the Q-function. The policy for each agent is therefore defined by the action with the largest Q value. This has some limitations, however: The policy is more deterministic; the optimal policy is often stochastic. Also, very small changes can cause an action to be selected, which leads to less convergence towards the optimal policy.

Policy-gradient algorithms use an independent function approximator to create a stochastic policy using its own parameters. This allows for the policy to be represented by a neural network, instead of possibly being solved by one. A typical method would have the representation of the state of each agent as input, policy parameters as weights and action probabilities as output. The policy parameters are updated proportionally to the gradient of policy performance. These algorithms can also be known as actor-critic algorithms. The actors, or agents, are usually decentralized and so act separately, but all usually work under one centralized reward function, or critic. This centralization of the reward function has become widely used in RL, even in value-based methods, because it reduces computational time. Value-based approaches may be easier to implement, but policy-gradient algorithms are found to perform at a much higher standard when tested in a range of differing environments and so have become the benchmark for RL. [8] Two of the many available Policy Gradient algorithms are used for testing in this paper, PPO and MADDPG.

Proximal Policy Optimization Algorithms (PPO) improves on other leading contenders in the field, such as Deep Q Learning and other policy gradient methods, by using clipped probability ratios which form a lower bound of the performance of the policy. PPO also has built-in policy and value optimisation, in which data is sampled from the policy and then several epochs of optimization are performed on that data, and the cycle repeats. [9] PPO performs significantly better than most available MARL algorithms on continuous control tasks, such as the movement of pistons.

MADDPG is an extension of DDPG, which is very effective for single agent problems, designed by the OpenAI team. [12] It improves on other policy gradient methods' performances in simple multi-agent settings. They extend actor-critic policy gradient methods by augmenting the critic, or reward function, with extra information about the policies of other agents. They operate under simple constraints to make

their algorithm general purpose: Learned policies can only use their own observation at test time, and there is no particular structure for communication between agents, so there is no differentiable communication channel. They use centralized training, with decentralized execution, which is a very popular paradigm in RL.

5 Evaluation

5.1 Environment Implementation

The environment structure is implemented using Pettingzoos API. Pettingzoo is a library based on OpenAI's Gym library for RL, but designed for multiple agents. They have two different frameworks for environments. The base environment API follows a Markov Decision Process, and they have a Parallel environment API which follows a Markov Game, with every agent acting at the same time. This allows anyone to create their own MARL environment, using the Pettingzoo codebase. [3]

In order to model the state and action spaces, Pettingzoo utilizes the Gym API's spaces and provides lots of helpful wrappers, to wrap the environment with which help with environment error handling and making sure that all actions are in bounds. The action space is modelled very simply, with a Discrete space of size 2. This represents two values, 0 and 1. They also provide Multi-Discrete spaces, which is utilized for the state space. This is a discrete space of size 3, which means 0, 1 and 2 represent the 3 available traffic light states, but it can become a vector of those discrete values of a chosen length, where all the values in one vector will be between 0 and 2. The length variable in a Multi-Discrete space allows the observability to be adjusted in the environment, with the help of networkx.

To be able to utilize graph theory effectively in Python, Networkx is used, which is a popular graph plotting and analysis library. Networkx has inbuilt functions to help find the nearest neighbours of a certain node, or traffic light. The amount of neighbouring nodes can be tweaked using a depth value which is set for 1 for all experiments.

Traffic light states can only update in the linear way that an actual traffic light will. So when the action is stop, the light can only go from green towards red, and when the action is go, the light goes from red towards green. Traffic light states then affect the change of traffic flows.

Traffic flows, representing traffic between two agents, has 2 'traffic' added to it if both agents, or nodes, have a red light. The flow number has 2 taken away from it if both lights are green. Traffic flows are limited to between 0 and 20, just to stop them getting too unrealistic as there is a possibility of them increasing a lot, especially when there are many agents in the network.

To visualise the outputs that this environment uses, the environment has two render modes. One simply prints a progress report to the terminal, and another outputs an image of the graph which is used to model the traffic network within the environment.

5.2 Algorithm Implementation

PPO operates using a neural network acting as its agent. In order for the observations to be in a form that is recognisable, batchify functions convert observations and returns to torch arrays. When the environment is only using partial observations, vectors can all be different lengths, so the batchify functions add padding to the vectors so they are the same length when input into the neural network. It also has a coefficient for entropy, value loss, and clip fractions. The entropy coefficient denotes how much exploration or exploitation, a common MARL dilemma, each agent is using within its behaviour. This is set relatively low, as the number of actions in the environment is low and so there isnt much to 'explore'. PPO also has coefficients for speed and magnitude of policy updates. These are both set low, so that it doesnt counteract the algorithm's need to exploit the current action.

MADDPG also uses neural networks for the agent. This paper implements MADDPG through a reinforcement learning framework called AgileRL. [13] AgileRL instantiates multiple MADDPG agents and trains them all at once in a Genetic algorithm style, with tournament selection deciding the best agent within the population. The best performing agent in training is saved to evaluate against the environment. This is evaluated using training fitness scores. To keep a level of fairness when testing against PPO, the population size used is only 2.

Table 1: Table of experiment settings

Number of agents	Number of episodes	Observability
4	500	Full
9	1000	Partial
16	3000	
25	5000	
49	10000	
100		

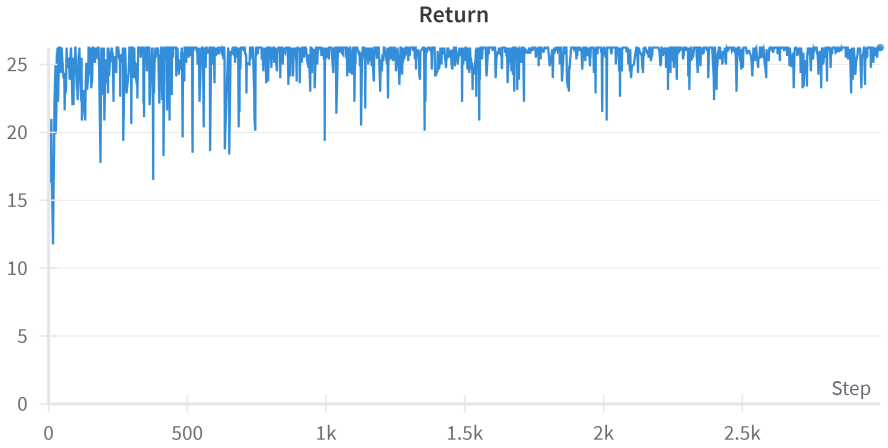

Figure 2: PPO, 3000 episodes, 16 agent network, full observation

5.3 Experiments

Table 1 above shows all the possible variations of experiments that were performed on both the MADDPG and PPO algorithm acting on the custom built traffic light control environment.

6 Results and Discussion

Figures 2, 3, 4 and 5 show full and partial observability tests for both algorithms on 16 agents.

The x axis values differ because testing did not carry on if an algorithm converged, and MADDPG converged after 1000 episodes on full observability for 16 agents, and 500 episodes on partial observability. The y axis values differ because PPO is calculating mean episodic reward, whereas MADDPG is calculating fitness, which is just the reward value.

Results were obtained by configuring a Weights and Biases(WandB) account, and using their API to log values such as returns and value loss of the chosen algorithms. WandB then produce helpful graphs that show the training of the algorithm, which were used in this report to compare algorithm performance. Both PPO and MADDPG's performance was monitored in this way, with agents having both full and partial observability in tests. The algorithms were being judged on convergence during training, which shows that the agents have optimised their approach to the given problem.

From the results obtained, it was found that PPO struggled to converge on all tests performed whereas MADDPG converged successfully by 3000 episodes at latest, on all numbers of agents and all observabilities. The only exception to this was 100 agents, which was not fully completed for MADDPG due to computational resource limits. PPO performed the best with regards to time, being a lot faster in training than MADDPG. Observability changes did not have much of an effect on convergence. The algorithms either converged at

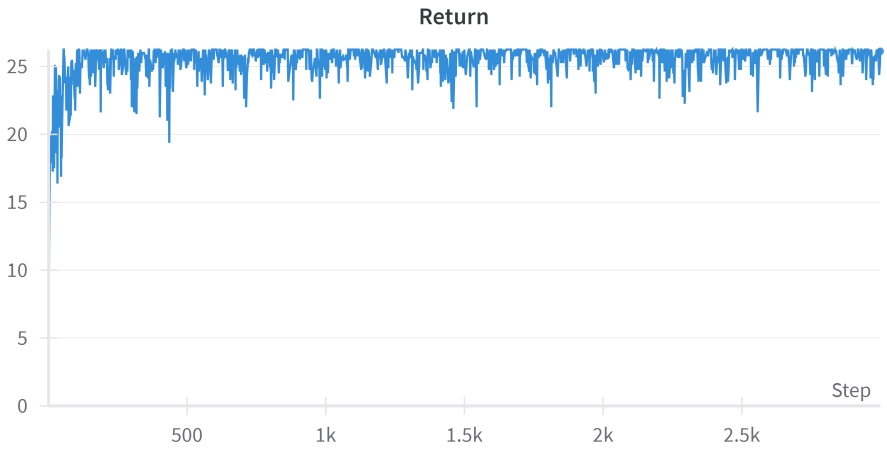

Figure 3: PPO, 3000 episodes, 16 agent network, partial observation

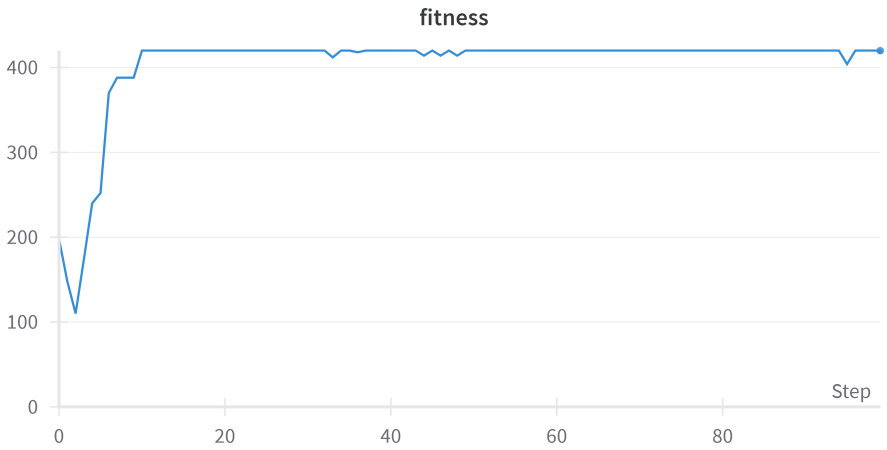

Figure 4: MADDPG, 1000 episodes16 agent network, full observation

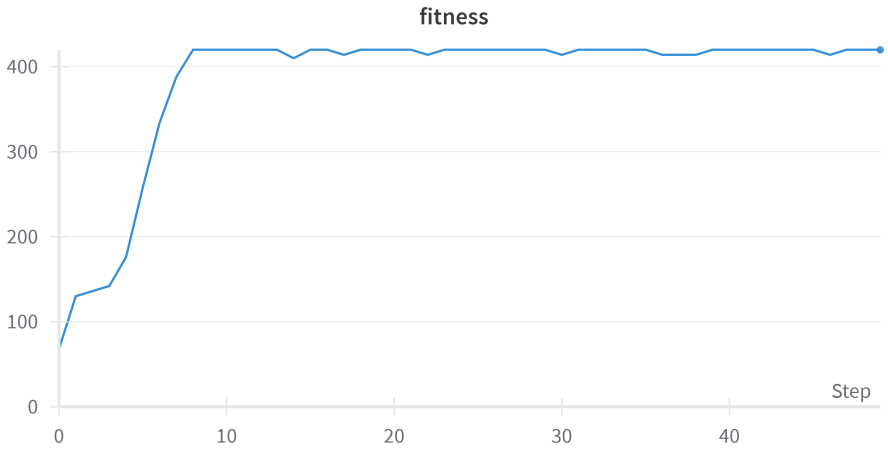

Figure 5: MADDPG, 500 episodes, 16 agent network, partial observation

similar times or did not at all. However, it did make training faster due to having to process less variables, especially at larger agent numbers. It also caused the plots to have less variance in their lowest and highest values, for the same reason.

This result may have been for a number of possible reasons. It could have been because MADDPG is designed for agents to behave in mixed competitive and co-operative environments, which actually represents the traffic light control problem better, instead of PPO which is designed better for full co-operation. The very small number of possible actions may have caused the algorithm to 'explore' more than it needed to, meaning changes in return values, which ends in lack of convergence.

Another possible factor to explain the test results is that the MADDPG algorithm instantiates multiple agents per episode and hence runs similarly to a genetic algorithm. The best performing agent has their fitness value recorded. This was kept as low as possible to make it as fair a test as possible, but may have still affected the results in comparison to PPO. All hyper-parameters were kept the same across both algorithms to ensure as fair a test as possible.

The reward function was also changed to a three tiered reward to test if that would have any effect on convergence:

$$20 > x > 10 = Y + 1 \tag{2}$$

$$10 > x > 3 = Y + 3 \tag{3}$$

$$X < 3 = Y + 5 \tag{4}$$

Due to time constraints, this new reward function was only tested on 16 agents. This agent number was randomly chosen from the available tests. This reward function had no real effect on results. MADDPG converged after 3000 episodes, as it did on the previous reward function, and PPO still struggled to converge, as seen in figures 6, 7, 8 and 9

7 Conclusion

In conclusion, Multi Agent Reinforcement Learning is a good solution for Traffic Light Control, because it enables problems such as observability and scale to be modelled with ease using Python Libraries Networkx and Pettingzoo. Algorithms such as PPO and MADDPG prove effective on a varying number of agents, with MADDPG performing better than PPO when tested over a varying number of agents on a grid shape graph topology. MADDPG achieved convergence in a relatively low number of episodes.

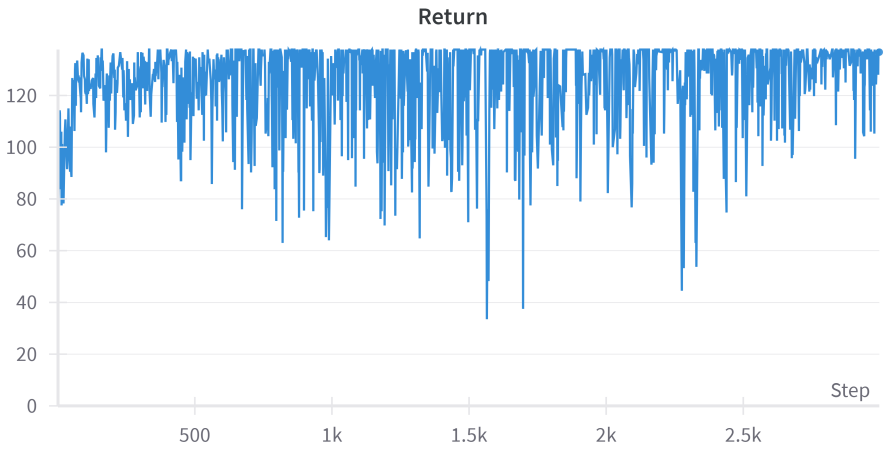

Figure 6: PPO, 16 agents, 3000 episodes, full observation

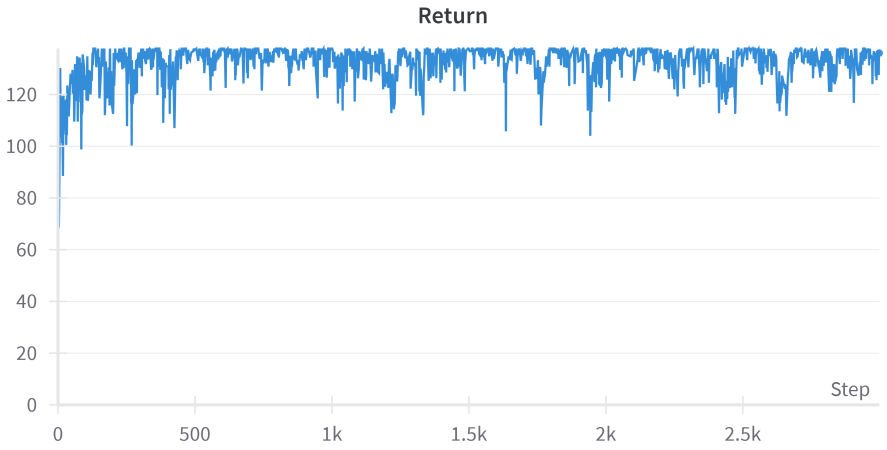

Figure 7: PPO, 3000 episodes, 16 agent network, partial observation

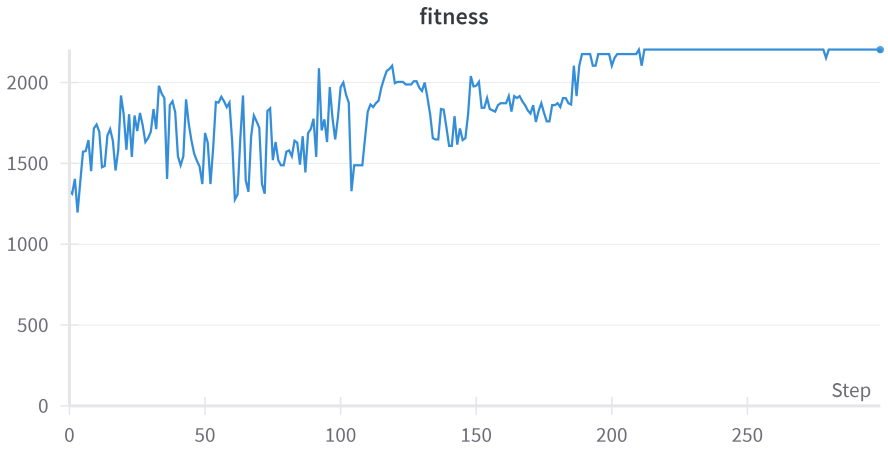

Figure 8: MADDPG, 3000 episodes, 16 agent network, full observation

Figure 9: MADDPG, 3000 episodes, 16 agent network, partial observation

8 Reflection

This study has taught many valuable skills that a developer might need, such as working in containerized environments with Docker, and working with external linux servers through SSH tunnels. Developing with Intelligent Agents and Environments and getting them to interact is a very powerful method and will be used in many ways in future, hopefully especially in Traffic Light Control, as has been proven to be possible and to have a positive impact in this paper.

The study has its limits and could be improved or extended in the following ways:

This study does not look into optimising very large numbers of agents with MARL algorithms. This could be a way of extending this study, but that brings its own issues. This is because the more agents there are, the more difficult it is to reach a good result, especially when modelling cooperation tasks. This is largely because an increase in the number of agents increases the size of the joint action space of all agents exponentially. [1]

Other topologies such as linear graph, cyclic graph or complete graph would also be implemented and tested if testing time was not limited. This would help model and test the coordination of all kinds of traffic light networks that can be found around the world.

MARL can solve two kinds of task, cooperating and competing. This study only looks into full cooperation of the traffic lights as this is deemed as the best way to obtain the best outcomes. However, the two categories of MARL tasks can also be combined into a mixed motive setting. Mixed motive has individual rewards for agents, but cooperative and competitive motivations coexist. Competitive motivations are modelled using a zero-sum game, which would mean introducing logic for agents to lose rewards as others gain into the reward function. [4]

This is a good extension to this study because It could be argued that traffic flow could be best represented as a mixed motive MARL problem. In reality, a person in traffic has joint motivations of not crashing but also individual motivations such as time pressure or differing directions that crossover at a junction.

9 Acknowledgements

This study was funded by UWE College of Arts, Technology and Environment under the Public/Community and Knowledge Exchange Scheme.

References

[1] Shou, Z., Chen, X., Fu, Y. and Di, X., 2022. Multi-agent reinforcement learning for Markov routing games: A new modeling paradigm for dynamic traffic assignment. Transportation Research Part C: Emerging Technologies, 137, p.103560. [Accessed 5/7/24]

[2] Xu, Y., Xi, Y., Li, D. and Zhou, Z., 2016. Traffic signal control based on Markov decision process. IFAC-PapersOnLine, 49(3), pp.67-72. [Accessed 5/7/24]

[3] Terry, J.K., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sullivan, R., Santos, L., Perez, R., Horsch, C., Dieffendahl, C. and Williams, N.L., PettingZoo: A Standard API for Multi-Agent Reinforcement Learning. Advances in Neural Information Processing Systems, 18, p.15. [Accessed 5/7/24]

[4] Du, Y., Leibo, J.Z., Islam, U., Willis, R. and Sunehag, P., 2023. A review of cooperation in multi-agent learning. arXiv preprint arXiv:2312.05162. [Accessed 5/7/24]

[5] Wang, X., Ke, L., Qiao, Z. and Chai, X., 2020. Large-scale traffic signal control using a novel multiagent reinforcement learning. IEEE transactions on cybernetics, 51(1), pp.174-187. [Accessed 5/7/24]

[6] Littman, M.L., 1994. Markov games as a framework for multi-agent reinforcement learning. In Machine learning proceedings 1994 (pp. 157-163). Morgan Kaufmann. [Accessed 5/7/24]

[7] Prabuchandran, K.J., AN, H.K. and Bhatnagar, S., 2014, October. Multi-agent reinforcement learning for traffic signal control. In 17th International IEEE Conference on Intelligent Transportation Systems (ITSC) (pp. 2529-2534). IEEE. [Accessed 5/7/24]

[8] Sutton, R.S., McAllester, D., Singh, S. and Mansour, Y., 1999. Policy gradient methods for reinforcement learning with function approximation. Advances in neural information processing systems, 12. [Accessed 5/7/24]

[9] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347. [Accessed 5/7/24]

[10] Zhang, H., Feng, S., Liu, C., Ding, Y., Zhu, Y., Zhou, Z., Zhang, W., Yu, Y., Jin, H. and Li, Z., 2019, May. Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario. In The world wide web conference (pp. 3620-3624).

[11] Paduraru, C., Paduraru, M. and Stefanescu, A., 2022. Traffic Light Control using Reinforcement Learning: A Survey and an Open Source Implementation. In VEHITS (pp. 69-79). [Accessed 5/7/24]

[12] Lowe, R., Wu, Y.I., Tamar, A., Harb, J., Pieter Abbeel, O. and Mordatch, I., 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. Advances in neural information processing systems, 30. [Accessed 5/7/24]

[13] Ustaran-Anderegg, N., Pratt, M. (2024) [computer program] Available at: (https://github.com/AgileRL/AgileRL/tree/main) [Accessed 5/7/24]