Phase 4 - Literature Survey and Results

Jason Weeks
Department of Computer Science
Mississippi State University
Starkville, Mississippi, USA
jcw1044@msstate.edu

Andrew McBride
Department of Computer Science
Mississippi State University
Starkville, Mississippi, USA
ahm228@msstate.edu

Andrei Roskelley Garcia
Department of Computer Science
Mississippi State University
Starkville, Mississippi, USA
ar2888@msstate.edu

Jacquies Turner
Department of Computer Science
Mississippi State University
Starkville, Mississippi, USA
jt2485@msstate.edu

I. Introduction

This paper will present the relevant literature in the process of traffic signal control using machine learning techniques, along with the timeline of research developments over time.

II. Taxonomy

Research on traffic signal methods falls into three categories. These include single-agent learning models, multi-agent reinforcement learning models (MARL), and large-scale multi-agent learning models, with the large-scale MARL actively having new deep learning models applied in recent research.

A. Single-agent Learning Models

Before 2010, traffic prediction models were designed using single-agent reinforcement learning models. These models were used to predict the movement of traffic using a single system, meaning a single neural network predicted how traffic lights should function. While innovative, they are generally not scalable for multiple intersections and can more effectively be used to predict how traffic signals function at a single intersection. These models were rarely used in practice and traffic signals used fixed intervals (threshold signaling) [1]. As a result, we will not focus too much on these for our traffic problem.

B. Multi-agent Learning Models

From 2010-2018, research began to focus on multi-agent learning reinforcement (MARL) models, a more practical approach to traffic signal control. These use several systems, or agents, to predict traffic flow for traffic signal control more accurately, but tend to be limited to knowledge of their own environment and not nearby intersections [2].

During this research period, the majority of models used a variation of Q-learning. While not perfect, there were indications that there were improvements in terms of moving traffic over traditional timer methods. [2] Eventually, a distributed MARL signal was designed such that each agent may learn the environment independently and communicate the signals it received to nearby agents to improve performance. This further came with the benefit of allowing one agent to fail without taking other agents down, increasing reliability and scalability [5]. This innovation led to major improvements that have influenced the modern design of reinforcement learning techniques for traffic signal control.

C. Large Scale Multi-Agent Learning Models

Since 2019, research indicates that MARL agents have been improved for scalability, performance, and real-world applications and become open-source. The field has seen enormous growth from the initial small-scale solutions, and large-scale implementations for real-world cities continue to become more practical and powerful.

Early breakthroughs in large-scale MARL focused on a more difficult problem: urban traffic management. This initially came with a new open-sourced model, CityFlow, a multi-agent reinforcement model which could handle large road networks with thousands of intersections in a simulated environment [7]. These large-scale models have shown significant improvements in performance while working at a larger scale, making them more adaptable to real situations.

In recent years, strong reinforcement learning models, such as long short-term memory (LSTM) and spatiotemporal graph convolutional neural networks (STGCN) have been applied to research, and external factors like pedestrian movements have begun being addressed. These advancements show more promise in scalability and performance but are still in progress. There are still opportunities to test different models which may perform better in synchronization with other agents. While accounting for pedestrians marks a major step in optimizing the traffic signal problem, it opens more questions, such as how we can account for emergency vehicles, namely ambulances or police cars, or collisions that disturb the natural flow of traffic. Despite these questions, these models are very powerful in comparison to previous threshold traffic signal management systems.

D. Taxonomy Figure

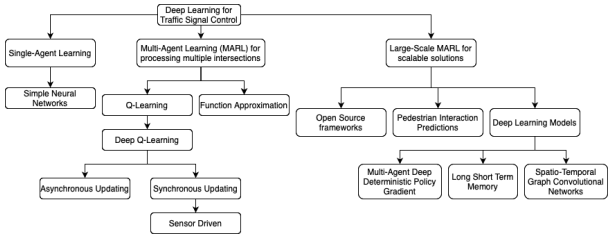

Fig. 1. Research Taxonomy for Traffic Signal Control

III. Chronological Overview

Based on the major research contributions, we have concluded that the papers fall into several categories. These categories tend to reflect the time period of the research contribution, with older papers using less powerful Q-learning simulations, and more recent papers using more powerful neural networks that account for more factors. Figure 2 highlights a chronological overview of the categorization of papers over time.

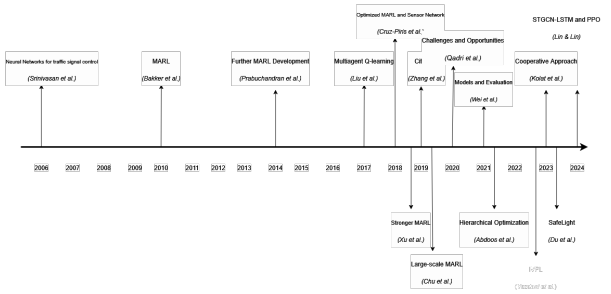

Fig. 2. Research Timeline for Traffic Signal Control

IV. Research Gaps

One current research gap is the types of models used. Early MARL models used various Q-learning techniques, with more recent studies using more advanced deep learning techniques, such as LSTM, but transformer-based models and RNN models seem to have been less tested. There are opportunities to test and benchmark different complex deep learning frameworks against each other.

Another research gap regards real-world application. Several large cities have adapted deep learning techniques for traffic signal processing, though a large majority of cities do not yet have these technologies implemented. Therefore, they could see a change in performance depending on the size of the city and their road systems.

A third research gap appears to be the influence of various real-world distractions that can occur, ranging from road accidents, pedestrians, and emergency vehicles. Although Du et al. have begun to take into account pedestrians [13], there are still several factors that remain unaccounted for in the traffic signal prediction process, which can back up traffic worse than threshold options.

V. Simulation Environment and Data Preparation

To effectively evaluate MARL models for traffic signal control, we will employ CityFlow, a multi-agent traffic simulation engine. CityFlow enables efficient traffic modeling by allowing pre-designed scenarios to be simulated instead of relying on real-time data. This approach provides greater flexibility in the definition and testing of traffic environments, making it ideal for research applications. CityFlow is particularly suited for large-scale urban traffic modeling. By simulating traffic flow in a controlled environment, we can test different reinforcement learning models without disrupting real-world traffic. The engine allows for scalable simulations with thousands of intersections, making it a powerful tool for evaluating various traffic control algorithms. For this study, CityFlow will be used to model the traffic network of Starkville, Mississippi, a city that currently employs traditional timer-based traffic light systems. This provides a relevant testing ground for machine learning-based traffic optimization, as improvements in traffic flow could have tangible real-world benefits.

The dataset used for traffic simulation consists of two primary components: roadnet.json and flow.json. The roadnet.json file defines the city's infrastructure, including the road layout, intersections, lane directions, traffic signals, and turning rules. This data is extracted from OpenStreetMaps (OSM) for Starkville and will be converted into a SUMO network before being integrated into CityFlow. Meanwhile, the flow.json file captures the dynamic movement of vehicles within the network, specifying entry and exit points, common routes, and characteristics such as vehicle priority and speed. Since real-world traffic flow data is not readily available, SUMO tools will be used to generate realistic traffic patterns, ensuring an accurate simulation environment to test reinforcement learning-based traffic control strategies.

Starkville presents an ideal case study due to its unique combination of urban and suburban traffic patterns. Unlike many cities that have adopted sensor-based or pressure plate-based signaling systems, Starkville still relies on traditional timer-based traffic signals, making it a prime candidate for testing reinforcement learning-based optimizations.

VI. Implementation

Our implementation will involve using CityFlow traffic simulation to estimate how traffic flows in Starkville and customize the minutia to our liking much more efficiently than a tool like SUMO. We can collect data from the environment such as congestion levels and states of other environments, along with a set of possible actions to implement a deep Q-learning model, which will reward or punish our model based on actions taken.

Because deep learning is computationally challenging, we will use a joint model. With this model, every action taken will result in a punishment or reward score for the model as a whole. This will allow us to train only one neural network at a time and comes with the added benefit of rewarding actions based on how they affect the state of other intersections. A model like this has historically proven to be more efficient

than existing threshold signaling. Our ultimate goal is to prove that this model can provide better traffic signal decisions than existing threshold systems such as those used in Starkville.

VII. Importance and Challenges

Because traffic light timing plays a crucial role in creating frustrated drivers and accidents, it is critical to find the most effective strategy to create smooth traffic flow through traffic signal control. As a result, our plan for the next two months is to implement a MARL model to predict traffic signals for the city of Starkville Mississippi, a city that uses threshold signaling for various times of the day to move traffic. It could be trained to predict various scenarios, including high-intensity traffic after an athletic event or lighter traffic during school breaks, with the ultimate goal of outperforming existing methodologies for traffic signal processing during all times of the day.

However, there are a variety of challenges with implementation. One challenge is the difficulty in finding optimal Q values as there is no ground truth to compare it to, the agent has to guess and learn from the experience and rewards it receives. This method is also very computationally challenging, as the agent has to constantly update the Q-value, especially with very large and complex models. There is also difficulty in balancing exploration vs exploitation, if the agent explores too much, then the Q-value may not converge to its optimal value, and if it exploits what it knows too much then the value may get stuck at a suboptimal level. Solving this would require understanding and the use of certain exploration strategies. There are also problems that could arise with the environment. The environment can slowly change over time, so our Q-value may become irrelevant, so the agent would have to relearn its environment. There are also many unexpected situations that may arise that may require the model to quickly learn and adapt to the unknown situation, or try to reinforce the model to account for these situations.

VIII. Development plan

Implementing a traffic simulation for Starkville, MS using CityFlow and Q-Learning over a two-month period, we will begin with the research and environment setup in the first two weeks. This includes installing and configuring CityFlow, generating traffic flows using built-in tools, and setting up a simulation model based on the Starkville road network and intersections. If real-world traffic data is available, we will incorporate it to enhance accuracy; otherwise, default CityFlow scenarios will be used. During weeks three to six, we will develop and train the Q-Learning algorithm by optimizing hyperparameters such as learning rate and exploration rate and running simulations to evaluate its performance. The effectiveness of Q-Learning-controlled signals will be compared against traditional fixed-time signals using metrics like vehicle waiting times and overall traffic flow improvements. In the last two weeks, we will focus on optimizing and validating the model, testing it under different traffic conditions specific to Starkville, Ms. The model will be deployed in a full-scale SUMO simulation, validated against available traffic data, and adjusted accordingly. The project will conclude with documentation of the implementation details and findings, while also identifying opportunities for future improvements, such as deep reinforcement learning for more advanced control. Ultimately, this project aims to develop an intelligent traffic light control system that reduces congestion and improves traffic efficiency in Starkville, Mississippi.

IX. Implemented Application

A. Simulation Setup

After starting implementation, we decided that CityFlow was not the appropriate tool for this problem, despite our research. While a powerful tool, it struggled to handle the data format provided by OSM data, resulting in slow performance and broken intersections. We then decided moving to the SUMO simulator was a more worthwhile decision, as its efficiency has improved over time and documentation is more appropriate for our specific research area.

Once SUMO was installed, we downloaded public data for the city of Starkville Mississippi's road structure using the tool OpenStreetMap and used SUMO's netedit to process the data into a usable format. Some changes were made, such as fixing traffic signal placement using junctions and fixing non-signal-based intersection interactions.

Our ultimate goal with this project is to test how neural network-influenced traffic signals direct the flow of traffic, so we needed to set up various simulation environments to simulate realistic traffic scenarios within Starkville. To do this, we used SUMO's built-in flow generator, with various parameters to result in light, medium, and intense traffic scenarios. This would allow us to test how each method of traffic signal control reacts to various events such as morning and afternoon rush hours, post-game traffic, or light summer traffic.

Using the configuration file, we can use the SUMO GUI, which comes included with the SUMO installation package. This tool allows us to run a visualization of the simulation with regards to the currently implemented method of traffic signal control, and make observations from the point of view of drivers without quantitative analysis, making observations about our model to make improvements. a

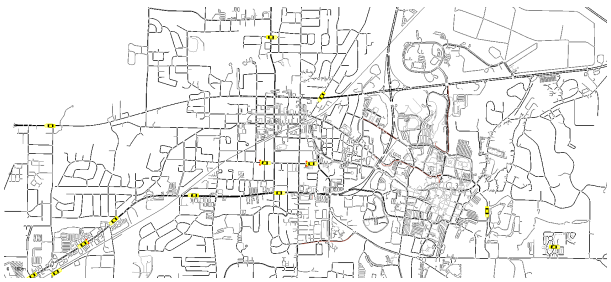

Fig. 3. Example of SUMO running in SUMO-GUI

B. Benchmarking

In order to properly evaluate the performance of threshold and neural network models, a method of evaluation was needed beyond qualitative methods, such as observation or review. As a result, we implemented various benchmarks to quantitatively decide the quality of a method's approach.

Our first method of benchmarking involves calculating the average waiting time of a vehicle. This is the most likely representation of how frustrated one might feel. A lower waiting time is therefore a great benchmark, as it shows that traffic is moving and drivers are less likely to get frustrated and develop road rage or get into an accident.

The next is the average trip time. Because we will benchmark similar environments, we can get the average trip times of all combined vehicles in a simulation, and use it as a quantitative performance metric. Faster trips indicate a successful implementation of traffic signal control, as people can get to their end destination faster. It also serves as a benchmark for other metrics, likely indicating shorter queue lengths, fewer stops, and less time waiting at stops on average.

Our final benchmark involves calculating queue lengths, or the average length of cars in line at an intersection. We can calculate both the max and average, to ensure the model is acting appropriately and not building super long lines to get the average down. A long line indicates that the model has a lot of buildup, and might fill later intersections with traffic, resulting in frustrated drivers and numerous stops if not carefully considered; therefore, it is critical that we minimize both the average queue length and the max queue length with our neural network.

Because there are several benchmarks, we will consider our model successful if any evaluation metric improves without a meaningful reduction of another metric. For example, if the average trip time decreases, but the other metrics remain very similar to their threshold results, we will consider the model a success.

C. Neural Network Implementation

Our decentralized multi-agent reinforcement learning (MARL) architecture is built around three key components: the Q-network itself, a Deep Q-Network(DQN)-based agent wrapper (complete with replay buffer and learning logic), and a SUMO–TraCI environment interface that supplies state and reward information every control interval. Traci allows us to easily modify our simulation and get state information from within the current timestep, while PyTorch allows us to efficiently build and implement our neural networks. This implementation lacks a central coordinator. Each agent acts on its own in coordination with other agents. The agents were designed to represent an intersection with a traffic signal controller and could accept the state of the lanes leading to the intersection, the state of the signals, and the state of vehicles at each intersection such as stopped time. We define the neural network in models.py as a simple multilayer perceptron. Given an input dimension equal to the total number of lanes feeding into a signalized intersection and an output dimension equal to the number of possible signal phases, we stack two hidden layers of configurable width (48 units each in our experiments) with ReLU activations and terminate with a linear layer producing a Q-value for each phase.

Concretely, the helper function

```
build_mlp(input_dim, output_dim,
hidden_layers):
    layers   = []
    prev     = input_dim
    for h in hidden_layers:
        layers.append(nn.Linear(prev, h))
        layers.append(nn.ReLU())
        prev = h
    layers.append(nn.Linear(prev,
    output_dim))
    return nn.Sequential(*layers)
```

constructs the sequential network, allowing us to easily swap hidden layer sizes or depths via configuration.

Each intersection is managed by an instance of `DQNAgent` (in policies.py), which maintains both an *online* network and a *target* network. At initialization, we load the target's weights from the online network and set up an Adam optimizer (learning rate 0.001), a discount factor $\gamma = 0.99$, and an ε-greedy exploration schedule that decays ε from 1.0 to 0.05 by a factor of 0.995 per learning step. Experiences are stored in a fixed-capacity deque buffer (size 5,000), and we sample mini-batches of 32 transitions once the buffer is sufficiently large. In each learn() call we compute the predicted Q-values for state–action pairs, form TD-targets via

$$Q_{\text{target}} \;=\; r \;+\; \gamma\,(1-d)\,\max_{a'} Q_{\text{target}}(s', a')$$

and minimize the mean-squared error between predictions and targets. After each gradient update, we decay ε, and every 100 updates we synchronize the target network with the online network. To scale this to multiple intersections, we wrap several `DQNAgent` objects in `MultiDQNAgent`, forwarding action selection, memory storage, and learning calls independently to each agent.

On the environment side (multi_env.py, subclassing BaseEnv), we employ the Python library TraCI (traffic control interface) to simulate traffic in Starkville, MS. Our initial plan was to predict the optimal states of every intersection with a traffic light in Starkville, MS, however, upon implementation, we realized our available hardware and the time frame would not allow us to do this. Instead, we opted to only use what we decided were points of interest, or intersections where traffic is particularly heavy and prone to congestion. Most of these points were decided by considering the importance of the road, the typical time spent at intersections, and personal experience. This method ensures that we don't expend computational resources at intersections that either see little traffic or do fine managing traffic on their own. Doing so significantly decreases the computational requirements to run the model, and allows us to focus only on critical points. At startup we invoke traci.start with a step length of 0.1s and a control interval of

5s, disabling verbose logs to save overhead. We automatically collect, for each traffic-light signal (TLS), the set of incoming lanes by inspecting traci.trafficlight.getControlledLinks. For every control step, get_state() returns a list of queue lengths, i.e. the number of halting vehicles per lane, by calling traci.lane.getLastStepHaltingNumber on each lane. We then compute a scalar reward for each TLS as the negative sum of its queue lengths, which incentivizes the agents to minimize congestion. When step(actions) is called, we apply each agent's chosen phase via traci.trafficlight.setPhase, advance the simulation by the control interval, and then return the new state, reward list, and a done flag (triggered when no vehicles remain).

The overall training loop lives in main.py, which begins by parsing our configuration file. This configuration specifies 200 training episodes, an agent name "multi_dqn," and all key hyperparameters (learning rate, γ, ε schedule, hidden-layer sizes, buffer and batch sizes, and target-update frequency). For each episode, we reset the environment, then for up to 2,400 steps (equivalent to 0.75 hours of simulated time) we query the multi-agent for actions, step the environment, store transitions, and trigger a learning update immediately. Episode returns (the sum of all TLS rewards) are logged each episode, and upon completion of training, we save each agent's weights to disk. A similar loop runs in evaluation mode (with greedy action selection and no learning) to report average performance over ten episodes.

D. Cross Validation

One potential concern over the implementation of our model is not the model itself, but the trustworthiness of the results. As a result, the design of the model took careful consideration into preparing valid results through cross validation and critical analysis. Various situations were considered for training, while only a single traffic scenario was used for testing and benhcmarking.

The first way we validated our results is by creating new scenarios. Each Episode of the model sees a completely new traffic scenario within Starkville, MS, granted with similar size due to computational limitations of large scale traffic. We then printed the reward, the punishment part of the Q-score each episode the ensure progress and exploration, and the results reveal relative consistency with a strong balance of exploration and exploitation. Even with new situations, the score grew and grew over time (please note that the reward is negative). The model was trained for 150 episodes in this scenario, to reach convergance while exploring as much as possible.

During our benchmarking phase we generated one testing scenario. This scenario uses a moderate-light traffic, based on Starkville, Mississippi's population and is used as a baseline to ensure that the improved results are not a result of a more or less congested traffic pattern. When we run the benchmarks we carefully ensured that this was performed to make sure results are trustworthy, and the the model did not opimize only one single situation it saw. When doing this, the results were overwhelmingly positive, and ensures that MARL based Deep Q-Learning is a valid way to optimize traffic signal controllers, as seen in section "Results over baseline."

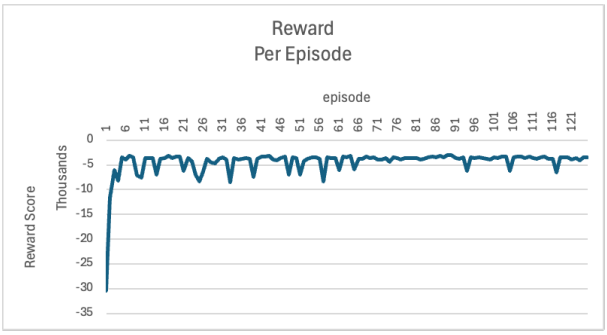

Fig. 4. Reward over 150 Episodes

X. Results over baseline

In order to evaluate our model we did two tests, a qualitative analysis, which involves viewing the simulation to compare the original simulation to the baseline, and a quantitative analysis where benchmarks were run to evaluate the model with numbers representing the time taken driving and length of queues. Both methods are critical to the effective analysis of the model. While training involves generating various traffic scenarios, or events where cars start at one point and end at another point, our testing involves one scenario, which uses similar traffic patterns seen in training. Using a single traffic scenario allows us to make observations of threshold and neural network-based signals. This ensures that while testing, each vehicle takes the same route so we can observe how our traffic signal management system performs relative to other methods.

A. Qualitative Analysis

When observing the baseline threshold-based traffic signals, it did an acceptable job managing light-medium traffic (2000 vehicles per hour), but saw relative redundancy with lights changing while no cars were in the opposite direction, causing congestion on major roads. It felt familiar, with minor traffic on less congested roads, such as backroads connected to highways and highways without traffic stops, but was certainly not as effective as possible.

The MARL Deep Q-learning model saw notable improvements in terms of responsiveness. When there were no signs of incoming traffic or lines in stopped lanes, it skipped the redundancy of changing the state of traffic lights resulting in a generally smoother flow of traffic. When it did change, it seemed intelligent, considering how long cars had been waiting along with how many cars were waiting. While our model did not allow collisions to be considered, the more continuous flow of traffic saw fewer emergency stops, and generally safer roads. Analysis also revealed that vehicles would often arrive at their destination sooner than their threshold counterparts, with less time spent on the road generally resulting in safer roads with less congestion.

There were some limitations of MARL Deep Q-learning despite its generally better analysis. Oftentimes, a light would not change for long periods of time to direct more intense traffic at the expense of shorter lines. This would generally result in shorter average drive lengths, with one drive taking exceptionally long. This was relatively controlled, however, with no cars getting completely stuck, and eventually being directed even if it took a minute. It was also important to choose the right intersections as points of interest, as intersections with little to no congestion would flip their signals the moment a car came to an intersection. It was generally better for areas with extremely light traffic to stick with threshold signals. For points like these, a potentially better option is sensor-based traffic signal controllers, which use more concrete rules to change traffic but make observations similar to our learning model to inform these decisions.

B. Quantitative Analysis

While simple observations provide great insight into the performance of our model, numbers to support our observations are potentially more important. As mentioned earlier, we developed several benchmarks based on the results of a simulation. Each benchmark provides a real insight that indicates the performance of a traffic signal control method without the need to make observations.

Testing the average trip time was one of the most important benchmarks implemented, since longer trips result in less cars on the road and a shorter time spent on the road, the odds of a traffic incident decreases dramatically. When using traditional threshold signals the average trip time took 695.34 seconds, or 11 minutes and 34 seconds. When using neural network based traffic signals a notable decrease in average trip time was observed, decreasing to 587.61 seconds, or 9 minutes and 47 seconds. With a nearly 108 second decrease in average trip time, we saw a 15% reduction in average trip time.

We also analyzed time spent waiting for both benchmarks, to see where the decrease in average trip time came from. In theory, this result would come from a decreased wait time, but some of it could come from the decreased traffic and increased speeds associated with better traffic signal management. With threshold signals, the average time spent waiting was 298.4 seconds, but with neural networks this decreased to 195.76 seconds. This resulted in a decrease of 102 seconds, confirming that the decreased wait time significantly contributed to the lower average trip time.

Another important metric to decrease it the average length of stopped cars at any point in time, known as the average queue length. Because our wait time decreased, we can expect a notable decrease in average queue length, as less cars need to stop and wait. This is confirmed, as we achieve an average queue length of 230 vehicles with threshold neural networks, and a queue length of 55 vehicles when using neural networks. This is an astronomical decrease, with an average of 175 less vehicles being stopped at any point in time. This is a nearly 76% decrease in average queue length, indicating a significant improvement in performance over baselines.

Finally, the maximum length of cars was determined in order to ensure that traffic did not ever back up very bad. Because the average queue length decreased greatly, it is safe to assume that the maximum queue length decreased significantly, which is confirmed by our results. When using traditional threshold signals the maximum queue length reached 1373 vehicles, and the maximum queue length reached 316 vehicles. This indicates that the peak congestion was significantly improved through our model over baselines, preventing large backups which can be frustrating and potentially dangerous.

Each of the benchmarks and their comparison to the baseline can be seen below.

Metric	**Without Q-Learning**	**Deep Q-Learning**	**Change**
Trip Time	695.34 seconds	587.61 seconds	↓107.33
Waiting Time	298.40 seconds	195.76 seconds	↓102.64
Avg. Queue	230 vehicles	54.98 vehicles	↓175.02
Max Queue	1373 vehicles	316 vehicles	↓1057

TABLE I
PERFORMANCE COMPARISON BETWEEN WITHOUT Q-LEARNING AND DEEP Q-LEARNING.

C. Analysis of Overall Results

Both qualitative and quantitative analysis reveal that deep Q learning is effective at modeling the traffic signal controllers within Starkville Mississippi, and for both traffic speeds and safety the Mississippi Department of Transport (MDOT) should consider updating the traffic signal control methods for points where traffic gets particularly congested.

With the average time spent driving decreasing drivers are generally safer, and large sections of traffic are less likely to make sudden stops. The shorter queue lengths result in less frustrated drivers, and quicker driving times, which also indicates a smoother flow of traffic and safer roads. This did come at the cost of computational power, resulting in a slower model that took time to train. It does perform faster than real time, but given the wrong situation could slow down the model where it could not make real time predictions; however, because the results were overall positive, there likely exists a cheaper and more efficient way to direct traffic.

While deep Q-Learning with MARL objectively performed well, computation times were relatively slow and implementation would prove very costly. Because of this, the MDOT should also investigate sensor based traffic signal controllers. These signals are cheaper to install, but consider timers and congestion levels with more strictly defined rules. Implementing these would avoid the high cost of training, implementation, and upkeep. Because using state data increased the performance with neural networks, its likely that a simple understanding of the state of the road could help direct traffic without implementing an expensive and unpredictable model. While still having some cost, it would decrease time spent driving and keep the roads of Mississippi safer.

XI. The Codebase and Demoing

A. Installation

In order to download the code one would need to go to github, and view chandlerweeks/TrafficSignalManagement. The link to the repository is: https://github.com/ChandlerWeeks/TrafficSignalManagement. Once here, fork the repository and make a local clone on your device. If a release is available at the time of reading the paper, that link will allow direct access of the codebase.

The model was not designed within a virtual environment, due to conflicts between operating systems of installations, so one will need to install the dependencies for the code beforehand, most noteably traci, likely via the pip install traci through a terminal.

In order to run the code, SUMO will also need to be installed, following SUMO's tutorial for installation for the necessary operating system. Once installed, the path variables will need to be set for sumo and sumo-gui in order for the code to run.

B. Live Demo

Once every prerequisite has been setup, to test installation and configuration, one should open SUMO-gui and open the file in data/simulation.sumocfg. If the file runs and traffic is observed, then the testing traffic flow scenario has been loaded and the code is ready to run.

In order to run training, navigate to the TrafficSignalManagement file in a terminal instance and run the provided command. `python traffic_signal_nn/main.py --config traffic_signal_nn/config/config_dqn_city.ini train`. Common errors include not having SUMO variables added to the path, and not having pip modules installed. Each episode will run, taking about a minute to simulate an hour of traffic. The reward will be printed to the console to let the user observe the loss, to see when the model explores and when it converges. Parameters can be modified for training within traffic_signal_nn/config/config_dqn_city.ini, including the neural network parameters and the selected roads. The code can take hours at a time to run, and a demo can be provided if necessary, via youtube.

In order to benchmark the baseline for SUMO, the scripts benchmark.py and get_queue_length.py were developed. benchmark simply gets the results of the runtime through the file data/output/tripinfo.xml. It contains information about the trip, such as average trip time, average waiting time, and average delay, but does not contain information on average queue length. Before running benchmarks, it is important to run the get queue length script, which obtains the average queue lengths seperate from what SUMO provides. One must run the benchmark scenario provided through simulation.sumocfg within SUMO-gui to get the correct output. In order to run the benchmarks, simply run `python benchmark.py` after running the simulation, along with `python .\get_queue_length.py --config .\data\simulation.sumocfg --output .\data\output\queue_length.csv` to determine the queue lengths.

In order to run the benchmarks with the model, one must run the trained model first to get the trip information. This can be ran through this command `python -m traffic_signal_nn.eval.run_model --config traffic_signal_nn/config/config_dqn_city.ini`, which runs the trained model on the same situation as the previous benchmark, for consistency as mentioned in cross-validation. Just like without the neural network, the queue length is not obtained for the benchmarks, so one will need to run `python .\traffic_signal_nn\eval\get_model_queue_length.py --config .\traffic_signal_nn\config\config_dqn_city --output .\data\output\queue_length.csv` in order the get the updated queue lengths. Once all of this has been done, one can simply run `python benchmark.py` to get the benchmark results for the trained model.

Setup of the project can be rather time consuming, as mentioned, a video can provided to the reviewers running the script completely setup if necessary.

XII. Acknowledgment of LLM Use

Large language models were not used in the writing or analytical process of writing this paper. However, it was used exclusively to summarize papers during the initial research process, just to ensure that the paper was suitable for our research needs as an aid to abstracts. Not every paper analyzed was used.

Large language models were used in assistance of coding, particularly for debugging and assistance with small code snippets, but never used to generate large parts of a funciton, or codebits. The code is developed by human, but AI assisted with some lines.

Work Distribution

The majority of research findings were made by Jason Weeks and Andrew McBride with help from Andrei by branching out from the original CityFlow paper [7] and finding predecessors and related articles in the field.

Based on the research findings, Jason Weeks, Jacquies Turner, and Andrew McBride contributed to the research taxonomy and timelines. These were made by narrowing down the most important research articles in our original findings.

The final distribution of the work is as follows.

- 25% - Jason Weeks
- 25% - Andrew McBride
- 25% - Jacquies Turner
- 25% - Andrei Roskelley Garcia

Reference Acknowledgment

References [1] through [15] highlight the most significant contributions to the study of traffic signal control using machine learning techniques. They highlight how the research has

progressed over time and should help guide the development of a model for traffic prediction.

The other sources provide great insight but might have had a smaller research impact. Despite this, they are still high-quality papers and can be used to help solve our problem. These are still critical to understanding the problem at hand and have a meaningful impact on the development of our traffic prediction model.

REFERENCES

[1] Bakker, B., Whiteson, S., Kester, L., & Groen, F. C. (2010). Traffic light control by multiagent reinforcement learning systems (pp. 475-510). Springer Berlin Heidelberg.

[2] Prabuchandran, K. J., AN, H. K., & 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.

[3] Srinivasan, D., Choy, M. C., & Cheu, R. L. (2006). Neural networks for real-time traffic signal control. IEEE Transactions on intelligent transportation systems, 7(3), 261-272.

[4] Xu, M., An, K., Vu, L. H., Ye, Z., Feng, J., & Chen, E. (2019). Optimizing multi-agent based urban traffic signal control system. Journal of Intelligent Transportation Systems, 23(4), 357-369.

[5] Liu, Y., Liu, L., & Chen, W. P. (2017, October). Intelligent traffic light control using distributed multi-agent Q learning. In 2017 IEEE 20th international conference on intelligent transportation systems (ITSC) (pp. 1-8). IEEE.

[6] Cruz-Piris, L., Rivera, D., Fernandez, S., & Marsa-Maestre, I. (2018). Optimized sensor network and multi-agent decision support for smart traffic light management. Sensors, 18(2), 435.

[7] Zhang, H., Feng, S., Liu, C., Ding, Y., Zhu, Y., Zhou, Z., ... & 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).

[8] Chu, T., Wang, J., Codecà, L., & Li, Z. (2019). Multi-agent deep reinforcement learning for large-scale traffic signal control. IEEE transactions on intelligent transportation systems, 21(3), 1086-1095.

[9] Qadri, S. S. S. M., Gökçe, M. A., & Öner, E. (2020). State-of-art review of traffic signal control methods: challenges and opportunities. European transport research review, 12, 1-23.

[10] Wei, H., Zheng, G., Gayah, V., & Li, Z. (2021). Recent advances in reinforcement learning for traffic signal control: A survey of models and evaluation. ACM SIGKDD Explorations Newsletter, 22(2), 12-18.

[11] Abdoos, M., & Bazzan, A. L. (2021). Hierarchical traffic signal optimization using reinforcement learning and traffic prediction with long-short term memory. Expert systems with applications, 171, 114580.

[12] Kolat, M., Kővári, B., Bécsi, T., & Aradi, S. (2023). Multi-agent reinforcement learning for traffic signal control: A cooperative approach. Sustainability, 15(4), 3479.

[13] Du, W., Ye, J., Gu, J., Li, J., Wei, H., & Wang, G. (2023, June). Safelight: A reinforcement learning method toward collision-free traffic signal control. In Proceedings of the AAAI conference on artificial intelligence (Vol. 37, No. 12, pp. 14801-14810).

[14] Yazdani, M., Sarvi, M., Bagloee, S. A., Nassir, N., Price, J., & Parineh, H. (2023). Intelligent vehicle pedestrian light (IVPL): A deep reinforcement learning approach for traffic signal control. Transportation research part C: emerging technologies, 149, 103991.

[15] Lin, T., & Lin, R. (2024). Smart City Traffic Flow and Signal Optimization Using STGCN-LSTM and PPO Algorithms. IEEE Access.

[16] Haydari, A., & Yılmaz, Y. (2020). Deep reinforcement learning for intelligent transportation systems: A survey. IEEE Transactions on Intelligent Transportation Systems, 23(1), 11-32.

[17] Wei, H., Xu, N., Zhang, H., Zheng, G., Zang, X., Chen, C., ... & Li, Z. (2019, November). Colight: Learning network-level cooperation for traffic signal control. In Proceedings of the 28th ACM international conference on information and knowledge management (pp. 1913-1922).

[18] Chen, C., Wei, H., Xu, N., Zheng, G., Yang, M., Xiong, Y., ... & Li, Z. (2020, April). Toward a thousand lights: Decentralized deep reinforcement learning for large-scale traffic signal control. In Proceedings of the AAAI conference on artificial intelligence (Vol. 34, No. 04, pp. 3414-3421).

[19] Zheng, G., Xiong, Y., Zang, X., Feng, J., Wei, H., Zhang, H., ... & Li, Z. (2019, November). Learning phase competition for traffic signal control. In Proceedings of the 28th ACM international conference on information and knowledge management (pp. 1963-1972).

[20] Wei, H., Zheng, G., Gayah, V., & Li, Z. (2019). A survey on traffic signal control methods. arXiv preprint arXiv:1904.08117.

[21] Wei, H., Zheng, G., Yao, H., & Li, Z. (2018, July). Intellilight: A reinforcement learning approach for intelligent traffic light control. In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 2496-2505).

[22] Yang, L., Luo, P., Change Loy, C., & Tang, X. (2015). A large-scale car dataset for fine-grained categorization and verification. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3973-3981).

[23] Milan, A. (2016). MOT16: A benchmark for multi-object tracking. arXiv preprint arXiv:1603.00831.

[24] Mao, F., Li, Z., & Li, L. (2022). A comparison of deep reinforcement learning models for isolated traffic signal control. IEEE Intelligent Transportation Systems Magazine, 15(1), 160-180.

[25] Noaeen, M., Naik, A., Goodman, L., Crebo, J., Abrar, T., Abad, Z. S. H., ... & Far, B. (2022). Reinforcement learning in urban network traffic signal control: A systematic literature review. Expert Systems with Applications, 199, 116830.

[26] Claus, C., & Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. AAAI/IAAI, 1998(746-752), 2.

[27] Arel, I., Liu, C., Urbanik, T., & Kohls, A. G. (2010). Reinforcement learning-based multi-agent system for network traffic signal control. IET Intelligent Transport Systems, 4(2), 128-135.

[28] Jácome, L., Benavides, L., Jara, D., Riofrio, G., Alvarado, F., & Pesantez, M. (2018, October). A survey on intelligent traffic lights. In 2018 IEEE International Conference on Automation/XXIII Congress of the Chilean Association of Automatic Control (ICA-ACCA) (pp. 1-6). IEEE.

[29] Mei, H., Lei, X., Da, L., Shi, B., & Wei, H. (2024). Libsignal: an open library for traffic signal control. Machine Learning, 113(8), 5235-5271.

[30] Eom, M., & Kim, B. I. (2020). The traffic signal control problem for intersections: a review. European transport research review, 12, 1-20.

[31] El-Tantawy, S., & Abdulhai, B. (2010, September). An agent-based learning towards decentralized and coordinated traffic signal control. In 13th International IEEE conference on intelligent transportation systems (pp. 665-670). IEEE.

[32] Tang, Z., Naphade, M., Liu, M. Y., Yang, X., Birchfield, S., Wang, S., ... & Hwang, J. N. (2019). Cityflow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 8797-8806).

[33] Cools, S. B., Gershenson, C., & D'Hooghe, B. (2013). Self-organizing traffic lights: A realistic simulation. Advances in applied self-organizing systems, 45-55.

[34] Bouktif, S., Cheniki, A., Ouni, A., & El-Sayed, H. (2023). Deep reinforcement learning for traffic signal control with consistent state and reward design approach. Knowledge-Based Systems, 267, 110440.

[35] Ducrocq, R., & Farhi, N. (2023). Deep reinforcement Q-learning for intelligent traffic signal control with partial detection. International journal of intelligent transportation systems research, 21(1), 192-206.

[36] Wang, X., Ke, L., Qiao, Z., & Chai, X. (2020). Large-scale traffic signal control using a novel multiagent reinforcement learning. IEEE transactions on cybernetics, 51(1), 174-187.

[37] Wei, H., Chen, C., Zheng, G., Wu, K., Gayah, V., Xu, K., & Li, Z. (2019, July). Presslight: Learning max pressure control to coordinate traffic signals in arterial network. In Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 1290-1298).

[38] Zang, X., Yao, H., Zheng, G., Xu, N., Xu, K., & Li, Z. (2020, April). Metalight: Value-based meta-reinforcement learning for traffic signal control. In Proceedings of the AAAI conference on artificial intelligence (Vol. 34, No. 01, pp. 1153-1160).

[39] Zhang, H., Liu, C., Zhang, W., Zheng, G., & Yu, Y. (2020, October). Generalight: Improving environment generalization of traffic signal control via meta reinforcement learning. In Proceedings of the 29th ACM international conference on information & knowledge management (pp. 1783-1792).

[40] Navarro-Espinoza, A., López-Bonilla, O. R., García-Guerrero, E. E., Tlelo-Cuautle, E., López-Mancilla, D., Hernández-Mejía, C., & Inzunza-González, E. (2022). Traffic flow prediction for smart traffic lights using machine learning algorithms. Technologies, 10(1), 5.

[41] Natafgi, M. B., Osman, M., Haidar, A. S., & Hamandi, L. (2018, November). Smart traffic light system using machine learning. In 2018 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET) (pp. 1-6). IEEE.

[42] Behrendt, K., Novak, L., & Botros, R. (2017, May). A deep learning approach to traffic lights: Detection, tracking, and classification. In 2017 IEEE International Conference on Robotics and Automation (ICRA) (pp. 1370-1377). IEEE.

[43] Ottom, M. A., & Al-Omari, A. (2023). An adaptive traffic lights system using machine learning. Int. Arab J. Inf. Technol., 20(3), 407-418.

[44] Kumar, N., Mittal, S., Garg, V., & Kumar, N. (2021). Deep reinforcement learning-based traffic light scheduling framework for SDN-enabled smart transportation system. IEEE Transactions on Intelligent Transportation Systems, 23(3), 2411-2421.

[45] Sagiroglu, S., Yavanoglu, U., & Guven, E. N. (2007, December). Web based machine learning for language identification and translation. In Sixth International Conference on Machine Learning and Applications (ICMLA 2007) (pp. 280-285). IEEE.

[46] Li, Y., Zhang, Y., Li, X., & Sun, C. (2024). Regional multi-agent cooperative reinforcement learning for city-level traffic grid signal control. IEEE/CAA Journal of Automatica Sinica, 11(9), 1987-1998.

[47] Peng, X., Gao, H., Han, G., Wang, H., & Zhang, M. (2023). Joint Optimization of Traffic Signal Control and Vehicle Routing in Signalized Road Networks using Multi-Agent Deep Reinforcement Learning. arXiv preprint arXiv:2310.10856.

[48] Zhang, Y., Yu, Z., Zhang, J., Wang, L., Luan, T. H., Guo, B., & Yuen, C. (2023). Learning Decentralized Traffic Signal Controllers with Multi-Agent Graph Reinforcement Learning. IEEE Transactions on Mobile Computing.

[49] Wang, K., Shen, Z., Lei, Z., & Zhang, T. (2024). Towards Multi-agent Reinforcement Learning based Traffic Signal Control through Spatio-temporal Hypergraphs. arXiv preprint arXiv:2404.11014.

[50] Chen, F., Chen, Z., Biswas, S., Lei, S., Ramakrishnan, N., & Lu, C. T. (2020, November). Graph convolutional networks with kalman filtering for traffic prediction. In Proceedings of the 28th international conference on advances in geographic information systems (pp. 135-138).

[51] Li, Z., Xu, C., & Zhang, G. (2021). A deep reinforcement learning approach for traffic signal control optimization. arXiv preprint arXiv:2107.06115.

[52] Zhang, Y., Zheng, G., Liu, Z., Li, Q., & Zeng, H. (2024). MARLens: understanding multi-agent reinforcement learning for traffic signal control via visual analytics. IEEE transactions on visualization and computer graphics.