Exploring the Effects of Enhanced Environmental Data for Real-World Traffic Signal Control Using Reinforcement Learning

Submitted April 2025, in partial fulfilment of

the conditions for the award of the degree **Computer Science.**

20474806

School of Computer Science

University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature _____FM_____

Date _17__/___04__/_2025__

I hereby declare that I have all necessary rights and consents to publicly distribute this dissertation via the University of Nottingham's e-dissertation archive.

ABSTRACT

This paper covers the development of a Reinforcement Learning based agent and a Linear Optimisation model that are both capable of routing cars, pedestrians and emergency vehicles across an intersection in a safe and efficient manner.

We propose two agents capable of handling the Traffic Signal Control problem: CIPHER (Control of Intersections for Pedestrians, Highway vehicles and Emergency Responders) and CIPHER+, which incorporates future vehicle arrival data into the state representation. To assess the impact of this more thorough state space, we compare CIPHER and CIPHER+ against six baseline methods from a range of traffic signal control paradigms (static, adaptive and RL-based) using real-world traffic data. Additionally, a linear optimisation model is described; this model was used for both guiding the agent's design and evaluating its effectiveness.

The results conclude that CIPHER outperforms CIPHER+ across all simulated traffic densities, and that CIPHER is capable of handling cars, pedestrians and emergency vehicles in a safe manner. However, the results also highlight a trade-off when comparing CIPHER against algorithms focused solely on cars and optimising their travel times. These findings highlight the potential for more RL-based work in a real-world deployment situation, as well as the usefulness of linear optimisation and classic adaptive TSC fundamentals in RL agent design.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

INTRODUCTION & MOTIVATION

Modern road traffic systems face a series of diverse challenges: rush hour, emergency vehicle prioritization and imperfect driver behaviour are just some of the driving forces behind why the problem of near-optimal traffic signal control is a challenging one to solve. Original fixed-time control algorithms fail to cope with the ever-changing demands of modern road networks. This results in increased journey times, increased vehicle emissions and reduced road user safety. A well-adapted traffic signal control algorithm should consider all road users, with a focus that extends beyond maximizing junction throughput to also prioritizing driver safety, emergency vehicle response times, and extending fairness to pedestrians and cyclists, in the same manner it does for cars. The economic cost of the problem alone makes it an enticing one to solve. One study [18] estimates that traffic congestion costs Americans an average of 97 hours a year. This figure extends up to 164 hours in population-dense cities such as Boston, with an annual cost of $2291 per driver. Meanwhile, in Europe, urban traffic congestion is estimated to cost almost 1% of the EUs annual GDP [6], equating to EUR270 billion. Another study [3] found a direct correlation between traffic congestion and CO_2 emissions, showcasing the need for proper traffic management in our increasingly carbon conscious world. Recent advancements in machine learning techniques, namely reinforcement learning (RL), have opened the door to Computer Science and Optimisation researchers in the context of the TSC problem. RL-based systems can be trained to effectively handle and process a wide variety of environmental data to optimise road traffic signals based on a set of varied success criteria, in real time. Whilst numerous studies have shown the effectiveness of RL in the context of the Traffic Signal Control problem, a systematic literature review demonstrates two critical gaps in the problem that I will attempt to address:

- There still exists a large gap between real-world and simulation, most pre-existing literature ignores the presence of certain road actors such as emergency vehicles and pedestrians. My research will consider fairness and response times to all road users, not just cars.
- Settling a debate from conflicting literature over the effect of an increased number of states and providing enhanced environment data to my model.

Chapter 2

PROBLEM BACKGROUND

This section introduces the fundamentals of the traffic signal control problem, as well as the technical preliminaries of reinforcement learning and linear optimization that are required to understand this paper.

2.1 Traffic Signal Control

The traffic signal control (TSC) problem involves optimizing the sequencing and duration of traffic lights to improve efficiency while maintaining safety for road users. Road users include, but are not limited to: cars, pedestrians, emergency vehicles, trams and public transport. The traffic signal control problem is often applied to a variety of road traffic networks (RTNs), some may focus exclusively on one intersection, whereas others will cover whole vast road networks. Traffic lights govern intersections by regulating the flow of traffic through a series of traffic phases. A traffic phase is a predefined combination of signals where specific lanes are granted a green light, whilst others will remain red. Certain constraints will apply to each phase:

- Two conflicting phases cannot be assigned to green at the same time, as this would cause crashes.
- A green phase must be preceded and followed by a mandatory yellow phase of a nominated duration, to ensure driver safety.

Objectives of traffic signal control will vary but are often focused on minimising overall driving waiting time or maximizing junction throughput, all whilst ensuring emergency vehicles are given priority over other road users, and no user is waiting for an unfair duration.

2.2 Reinforcement Learning

Reinforcement learning (RL) is a subset of artificial intelligence that enables a model to learn through trial and error, rather than utilizing a specific training dataset. RL leverages the Markov decision process (MDP). MDP is a framework for decision making in an environment with uncertain outcomes. Formulating a problem using the MDP involves defining five main components:

- Agent – the learner or decision maker.

- Reward – A numerical variable returned to the agent that indicates how favourable a particular state is to be in.
- Actions – the set of actions the agent can undertake; this action is executed on the current environment to produce a new state.
- State – represents the current configuration of the environment in which the agent is deployed.
- Policy - The set of learned actions the agent will take when in a particular state.

RL agents select the most appropriate action based on the long-term reward they can expect to see from it; this value is called a Q-value and there is one assigned to every state-action pair. The Q-values are stored in a Q-table and learned through a process called Q-learning. Q-values are updated using the Bellman equation:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$

Where:

- $Q(s, a)$ is the expected future reward for taking action a in state s.
- r is the immediate reward received after taking action a in state s.
- γ is the discount factor – a hyperparameter which determines the importance of future rewards ($0 \leq \gamma \leq 1$).
- s' is the next state after taking action a in state s.
- a' is a possible action in the next state s'.
- $\max_{a'} Q(s', a')$ is the maximum Q-value of the next state s' over all possible actions a'.

Deep Reinforcement Learning

Classical reinforcement learning will struggle to scale for large problems due to the curse of dimensionality – as the state / action space grows so will the respective Q-table. This large Q-table soon becomes computationally infeasible to query and update. This issue is particularly common in environments with continuous state spaces or a diverse set of potential actions. To address this limitation, deep reinforcement learning (DRL) replaces the Q-table with a neural network that is used to approximate Q-values and derive an effective policy. The Bellman equation is still used, serving as a comparison value to back-propagate weights and biases in the network. The model

receives state information through the input nodes, and outputs Q-values for each possible action, the action with the largest Q-value is enacted upon the environment. Pseudocode for the DRL training process is defined below:

```
while Training:
    # E is a hyper-parameter used to balance exploration and
    # exploitation
    if random() > E:
        # Forward pass through model to generate Q-value for each
        # action, pick max Q-value
        pick_best_action()
    else:
        pick_random_action()

    next_state, reward = apply_action(action)
    store_transition(state, action, next_state, reward)
    state = next_state

    # Pick random subset of transitions
    batch = sample_transitions()

    # Compute Q-values for every transition, returns Q value of
    # best action for each state
    predicted_q_values = model.predict(batch.state)

    # Work out actual Q-value obtained from applying transition
    actual_q_values = bellman_equation(batch.reward)

    loss = huber_loss(predicted_q_values, actual_q_values)

    # Update model weights
    backpropogate_model(loss)
```

2.3 Linear Optimisation

Linear and Discrete Optimisation (LO) is a form of mathematical optimization in which problem variables can be either continuous or discrete. It is not discussed here for brevity; an interested reader can find more information from [2].

Chapter 3

RELATED WORK

This literature aims to provide a comprehensive account and comparison of the various approaches to formulating and solving the TSC problem. TSC methods can be categorized into one of three main types:

- Fixed Time Systems: These operate based on a cyclical structure with a fixed time for each traffic phase.
- Actuated / Dynamic Systems: These follow rule-based conditions to dictate signal activation, such as extending green phases for a specific road during rush-hour.
- Adaptive Systems: These will dynamically adjust the phase length and cycle pattern based on real-time traffic data, to optimize flow in response to current traffic conditions. Notable adaptive algorithms include Websters [34], MaxPressure [32] and SOTL [14].

More recent research has focused on leveraging computational methods in the problem context to develop adaptive systems by incorporating heuristics, meta-heuristics, artificial intelligence and mathematical optimization. This literature review will focus on the latter two.

3.1 Reinforcement Learning Literature Review

Frameworks Used

Reinforcement learning was first applied to the problem through Q-learning by Kakazu [20], with the addition of genetic algorithms to efficiently search and update hyperparameters across training. Despite being a valid first step in applying RL to the problem, the paper rapidly encounters the curse of dimensionality due to the wide and complex state space of the problem. The model is only able to operate within a small RTN. One paper [27] addressed this with function approximation techniques employed to reduce state-space and negate the issues encountered by Kakazu. More details on this are discussed later on in Section 3.1.

Modern research has focused on leveraging neural networks through DRL, to cope with the high-dimensional state space. A large variety of studies [17][30][10][40][29] have demonstrated the superior results that can be achieved through DRL models. The absence of a large Q-table that was noted in other studies makes the system more precise and scalable.

Simulator Choices

Urban simulators are an effective way to model vehicle flow and capture key complexities such as driver behaviour and vehicle acceleration. Using a sophisticated simulator will ensure more accurate adaptation to the model's real-world deployment environment.

The simulation tools available fall into one of three categories:

- Microscopic simulators consider individual driver behaviour and their interactions with other road users.
- Macroscopic simulators abstract individual details and focus on vehicular flow, without modelling individual vehicle dynamics.
- Mesoscopic simulators are a hybrid of both approaches and will model overall vehicle flow with some consideration to individual behaviour.

With the exception of [20] and [30], who developed their own macroscopic environment, the other papers considered here used third-party, microscopic simulators. The most popular simulator noted was SUMO [23] used by 3 papers [17][10][29], other simulation environments used were VISSIM [11] and GLD [39]. Another literature review [28] collated 57 papers and found that 33% of papers studied used VISSIM whilst 25% of papers used SUMO, this highlights the two software's dominant superiority across the field of urban simulation.

One notable shortcoming across the literature is the lack of simulation parameters disclosed, meaning it is hard to replicate and verify results. Factors such as: max car speed, acceleration, road length, vehicle length and driver imperfection rating are often omitted, despite the sensitivity of final results regarding these parameters.

Actors Involved

Simulation tools allow for the inclusion of various road actors. SUMO allows for the inclusion of cars, cyclists, pedestrians, emergency vehicles (EMVs) and public transport. Despite this, most papers studied focus exclusively on cars. This shortcoming limits the real-world validity of many studies and is a major barrier to deploying systems into the real-world.

One exception to this statement is EMVLight [29]. The paper proposes a framework for coupling efficient EMV routing with a RL-based TSC agent. The model aims to reduce travel times for both civilian vehicles and EMVs. Similarly, IVPL [40] extends the simulation environment to involve pedestrians. The work is based off a pre-existing model [9] but extended to penalize pedestrian

delays. Real-world validity of this work is extended by simulating jaywalking and studying the effects this realistic factor has on model performance.

Despite the efforts of these two studies, to the authors knowledge, there are no available studies that focus on developing a RL agent that can effectively handle all actors it would encounter in a real-world deployment. Further investigation into this area would not only reduce overall congestion but also potentially result in optimized emergency response times as well as increased walkability and pedestrian safety.

Intersections Modelled

The TSC problem extends to a wide variety of road configurations but is most commonly studied in the context of one or more networked four-way intersection. This could be attributed to the complexity that makes solving such a formulation highly desirable: incoming vehicles from all four directions, competing for control of one singular intersection.

Whilst some studies [17][10][40] focus on creating an agent that can effectively control just one of these intersections, others [20][27][30][29] proved the scalability of their models across multiple intersections networked together. Such a setup allows for experimentation on the results of increased environment context for the models. Figure 3.1 showcases results from Tham [30] who networked two RL-based agents together to allow for sharing of information. This model (Case III) was compared to two solitary models (Case II) and a fixed time model (Case I). As shown by the graph, allowing the networked agents to communicate environmental data between one another resulted in significantly reduced average delay within the simulation.

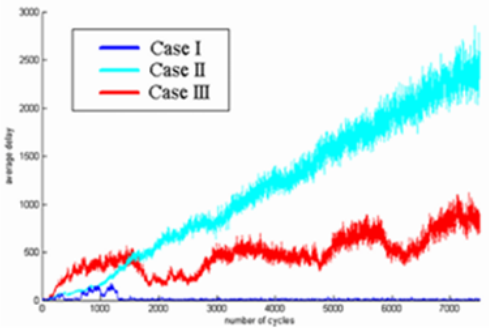

Figure 3.1: Comparison of average delay for different TSC models, adapted from (Tham2007).

State Representation

As outlined in Section 2.2, solving problems such as TSC requires them to be formulated as an MDP. Given the infinite, continuous state space, efficient representation of the current environment state will enable a model to effectively converge on an optimal policy. Common state factors observed across the literature are:

- Current phase – a vector representation of lights that are currently green.

- Vehicles in lane – the number of vehicles currently on a lane.
- Queue lengths – the number of vehicles queueing on a lane, a vehicle can be defined as queueing if its current speed is 0 m/s (stationary).

As noted earlier, Q-learning based algorithms are hard to apply to large problems with complex state spaces, due to the curse of dimensionality. However, [27] applied function approximation techniques to categorize the continuous state space within discrete categories. Reducing the size of the state/action pair space from 10^{101} to just 200. Although this enabled the agent to handle larger intersections initially, this proved to be unscalable due to the state space still growing exponentially.

Networked, multi-intersection agents naturally extend their state space with information from other intersections. One paper's [30] state information would include the action and Q-value obtained from any downstream agents, so that upstream agents could anticipate incoming traffic. State space size also grows with actors present in the simulation, as there is more information to represent. IVPL [40] was focused on the inclusion of pedestrians, and state information would include pedestrian numbers on each crossing. Meanwhile, another paper [29] included comprehensive information on emergency vehicles: the distance of the EMV to the intersection, the ETA of the EMV at the intersection, and the EMVs proposed next step.

By extending state representation to allow for multi-agent co-ordination and varied road users, research is closing the gap between real-world and simulation. However, the computational resources required to develop a comprehensive policy grows exponentially with the state space, and this remains an open challenge in DRL-based TSC.

Agent Actions

All RL-based algorithms studied have their action focused on picking the next, most-optimal traffic phase. Careful design is needed to design an action space large enough that it can comprehensively respond to all possible states, whilst remaining small enough that training the model is computationally feasible. The majority of papers noted in this study will choose phases in an acyclic manner. One paper [38] instead opted for a cyclical structure and a binary action space $\{0, 1\}$, where 0 opts to stay in the current phase and 1 transitions to the next. Whilst this policy may be more efficient to train, it gives a much more limited flexibility in comparison to some of the other studies.

Some papers [29][10] opt to provide the model with a pre-configured set of phases, where the action space size reflects the different phases available to the model, which will output a score for each, indicating the suitability.

Other papers [30][17] have taken a more flexible approach, allowing the model to select both the next active phase, and its duration, Tham chose a continuous action space $(0 \dots n)$, where n is the number of phases available to the model, which will output a (bounded) value for each, indicating the green time it should be allocated. The author notes it is important to consider the negative effects of such a large action space, the model failed to scale past even a simple 2-light intersection.

[40] uses a hybrid rule-based/RL approach to iteratively clear pedestrians and allocate a constant amount of green time to each car lane. If there are still vehicles remaining in the simulation the RL-agent will allocate additional green time to each lane according to the policy, if there are no vehicles present the cycle repeats. Whilst this approach can ensure fairness across both groups, it does not allow for pedestrian lights and vehicle lights to be green at the same time, even if the two routes do not co-inflict.

Reward Function Choices

Effective reward functions allow for dictation of the model's priorities and can differ from problem to problem. Some reward functions in the literature focus on minimizing cumulative user delay, this correlates well with the real-world objective of minimizing user travel times. Furthermore, this value can also be exponentially weighted to penalize longer wait times. However, it also requires constant vehicle tracking, something much easier done in a simulation environment than real world.

Another common function is to maximize junction throughput, this reward style will encourage smooth traffic flow but may result in unfairness to certain road users, as there is no incentive for the model to regulate wait times.

Training and Testing Datasets

Due to there being no known optimal solution for the TSC problem, evaluation is usually done through comparison of different models. A significant challenge regarding this is the low availability of open source, consistent datasets, meaning most studies use synthetic data for the training and evaluation. Arrivals are often simulated with a Poisson or Gaussian distribution to mimic arrival patterns.

Certain papers [40][17] have used real-world datasets to evaluate their models. However, it is observed in an alternative literature review [28] that 42.2% of studies would use generated data for evaluation. This once again, raises questions about the real-world validity of the models.

From the literature review, it is clear that the lack of real-world data is a serious barrier to standardization and real-world deployment of models. The lack of datasets could be attributed to privacy concerns, or practical difficulties in collecting such a comprehensive dataset. Nevertheless, work focused on capturing and processing detailed traffic data would be a valuable contribution to

the field.

Results

From the literature, it is appropriate to say that RL-based TSC algorithms are capable of outperforming non-RL adaptive ones and fixed time algorithms in a variety of metrics including junction throughput, vehicle delay and overall travel time. Studies that included pedestrians [40] and EMVs [29] were also able to extend fairness to these actors.

Interestingly, there are conflicting results across the literature about including information on the wider environment for the model. [30] observed significantly improved results from networking two intersections together and [29] successfully routed EMVs by predicting their arrival times. In contrast, other papers [5] [42] were able to outperform various benchmark models through minimalist state design, leading to reduced model training times and costs.

Summary of related studies

Study	RL Paradigm	State							Action		Reward			Simulation Environment Used
		Current Phase	Vehicles per Lane	Queue Lengths	Arrival Time	Vehicle Positions	Vehicle Speeds	Upcoming Phase	Cyclic	Acyclic	Minimizing Delay	Minimizing Waiting Times	Maximizing Throughput	
Kakazu et al.	Stochastic Learning Automata	✓		✓						✓			✓	None
Tham et al.	Q-Learning			✓	✓					✓		✓		SMLP
Ducrocq et al.	DQL	✓				✓	✓			✓	✓			SUMO
Gao et al.	DQL	✓				✓	✓			✓		✓		SUMO
Genders et al.	DQL	✓	✓	✓		✓				✓	✓			SUMO
Oliehoek et al.	DQL	✓				✓				✓	✓	✓		SUMO
Wei et al.	DQL	✓	✓	✓				✓	✓					Real-world Data
Yazdani et al.	DQL					✓	✓			✓	✓			VISSIM
Bouktif et al. (a)	DQL	✓								✓	✓			SUMO
Zheng et al.	DQL	✓	✓							✓	✓			SUMO

Table 3.1: A comparison table summarising the studies mentioned within this literature review and some additional ones not discussed for brevity.

Studies cited in order of table appearance: [20][30][10][12][13][26][37][40][5][42]

Conclusion

The literature shows extreme potential for the application of RL paradigms in the context of TSC and DRL, specifically as a promising approach for handling such a complex problem. However, it also highlights the significant gap between real-world and simulation. The majority of papers cited only consider cars, meaning it is impossible to evaluate how the models will perform when deployed and encounter EMVs or pedestrians. Although certain papers have set out to include one or the other, to the author's knowledge, there is no model capable of effectively handling all 3.

Another consistent theme across the review was the impact that the curse of dimensionality has on model design choices, and how a concise state representation will result in shorter training times without necessarily a performance loss. There is literature that suggests the opposite is true in

some cases, Tham [30] and Su [29] both saw very positive results from giving context from a wider environment to the model.

Furthermore, there is a lack of standardization across the field, models are often designed and trained across varying simulation environments and road networks. Detailed simulation parameters are often not included, making it hard to reproduce results.

From this review, it is clear that future work should be focused on:

- Producing real-world datasets for standardized evaluation across models.
- Developing systems capable of optimizing vehicle, pedestrian and EMV travel times.
- Conducting research into the effects of supplying increased environmental data to the model.

3.2 Linear Optimization Literature Review

Recent work has also focused on using LO methods to solve the problem. These models leverage mathematical programming techniques to determine (near) optimal traffic settings. A systematic review of the remaining literature highlights two flaws also commonly seen within RL-based studies: a lack of real-world validity and a high computational cost associated with running these models.

Despite these flaws, LO-based methods are able to provide a near globally optimal solution, meaning they can provide a strong baseline for comparison against RL-based approaches and serve as a highly valuable area of future research.

Introduction

Out of the four papers [33][21][4][16] noted in this study, all formulate the key decision variable as a binary array $\mathrm{x}[S][T] \in \{0, 1\}$, where:

- S represents a specific traffic signal present in the network.
- T represents a discrete time step.
- $\mathrm{x}[S][T]$ indicates whether signal S is green at time T.

This binary formulation offers a flexible and effective representation of the problem as a Mixed Integer Linear Programme (MILP). However, the large number of binary variables does not optimize well, particularly when using the branch-and-bound method used by LO solvers today.

Intersections Modelled

Unlike the microscopic approach noted amongst the RL-based studies, all LO models took a macroscopic approach to modelling the intersections. This can be attributed to the challenges in representing microscopic factors, such as individual driver behaviour, mathematically; as well as the inability to integrate simulators like SUMO into LO models.

Similar to the RL-based papers studied, the literature in this field covers both singular and networked intersections. However, all intersections modelled represented a simplistic layout, with one lane per incoming direction. This reduces the problem size and simplifies constraints in that only one light can be active at a time. There remains an important direction for future research into models of more complex intersections, which would theoretically allow for multiple lights to be safely green at the same time.

Furthermore, the vast majority of LO studies focus exclusively on vehicle traffic, mirroring a similar limitation seen in RL studies. Amongst the four papers reviewed, three exclusively considered cars, while only one paper [16] had the additional actor of light-rail systems. This study was able to successfully model constraints surrounding this and optimize the phases in regard to both traffic flow and rail systems simultaneously. Despite this, the bias towards cars in the literature is evident. The lack of extraneous road actors and the omission of microscopic factors make the results from these models mere approximations, rather than fully developed solutions that would be expected to work in a real-world environment.

Queue Transmission Models

A fundamental aspect of modelling the TSC is the equation used to represent vehicle flow across a road network and the build-up of traffic queues. This area of the problem will be subsequently referred to as the queue evolution strategy. One of the first such strategies was initially proposed by Miller [24]. This is called the Queue Transmission Model (QTM). Miller states that the queue for one approach at a single intersection, at a given time step t, can be modelled as:

$$q(t+1) = q(t) + q_{\text{in}}(t) - q_{\text{out}}(t)$$

where:

- $q(t)$ is the queue length at time t.
- $q_{\text{in}}(t)$ is the number of vehicles arriving at time t.
- $q_{\text{out}}(t)$ is the number of vehicles departing at time t.

Across the modern literature, three primary approaches were observed that aim to utilize and build upon Miller's work:

- **CTM (Cell Transmission Model)** Initially proposed by Daganzo [7] and building directly upon the work of a QTM proposed by Miller. This was incorporated into models by [21] and [4]. Daganzo broke each road segment into discretized distances of a fixed length. A section is denoted by its origin (o) and destination (d) as (od). The continuous variable v_{od} denotes the number of vehicles on the section (od) at time t. Therefore, the queue update equation can be modelled as:

  $$v_{od}(m+1) = v_{od}(m) - p_{od}^{out}(m) + p_{od}^{in}(m),$$

  where p_{od}^{out} and p_{od}^{in} represent the outflow and inflow traffic of a section, respectively.

  Whilst the extension of the CTM upon the QTM provides a finer representation of vehicle positions within the network and partially addresses the lack of microscopic representation noted previously, it also increases the problem space by representing each road as a series of segments rather than as a single large cell. That said, employing a CTM is essential for modelling multiple networked intersections, as demonstrated by the two previously mentioned papers.

- **Extended CTM** Building upon the work of Daganzo, Guilliard [16] correctly stated that a CTM is only advantageous (and necessary) over a QTM when a roadway does not diverge or merge into another. Guilliard then employed a hybrid approach for the complex road networks modelled, only using a CTM when strictly necessary. Further optimizations employed by Guilliard are discussed in the optimisations section of this review. Due to this extended CTM, and other optimisations, Guilliard was able to scale the model to effectively handle a 3×3 grid of intersections —more than any other paper considered in this study.

- **Kinetic Wave Model** Proposed by [8] and utilized by [33], this model states that traffic flow can be characterized by the following parameters: - u_r (forward wave speed) - w_r (backward wave speed) - $q_r^{\max}$ (saturation flow rate) - κ_r^j (jam density)

  Unlike the CTM, this model captures the stochasticity of traffic flow, a valuable microscopic factor that most models do not incorporate. However, implementing this equation within the model also results in more binary representations than the previous two queue evolution strategies.

Objective Functions

A variety of objective functions were noted across the literature, perhaps the most sophisticated being presented by [16] which leveraged the extended CTM to calculate exact travel times for vehicles in the simulation. Other objective functions would focus on minimizing total traffic in the network at any given time [21] or similarly, maximizing junction throughput [4].

Constraints

Multiple similarities were also noted across the literature in terms of the problem constraints noted. Multiple papers [4] [16] [33] imposed constraints on both the minimum and maximum green time. The respective authors cited various reasons such as mimicking pedestrian clearance, safety, and realism. Furthermore, all papers modelled cars travelling through the network by using a constant saturation rate parameter. Formulated as: $d_{cell,time} \leq \lambda$. Where λ is the maximum number of vehicles that can travel across a cell per time step and $d_{cell,time}$ represents the number of vehicles departing from a specific $cell$ at a given $time$. One notable shortcoming of all papers cited above is the lack of explicit yellow signals within the problem. Yellow signals will cause lower throughput than a green signal due to the necessary acceleration/deceleration of cars from the preceding or succeeding red phase. This omission could be attributed to the fact that the inclusion of yellow signals would introduce another set of binary decision variables. Instead, researchers modelled this lost start-up time in various ways: [4] introduced a constraint to reduce the previously mentioned saturation rate parameter at the initiation of a green signal, to account for the lost start-up time. Meanwhile, [16][33] both introduced a penalty for frequent phase switching in the objective function. Whilst being a valid attempt at modelling the impact of yellow signals, neither approach proposed here captures this constraint directly.

Optimizations

A consistent theme throughout the literature is the high number of binary variables involved in representing the problem, despite optimization attempts such as the extended CTM. Other successful optimization attempts include splitting time into discrete intervals. This was an optimization proposed by [33], where instead of a decision variable being represented as an array of $time \times lanes$ size, it is represented by an array of size $\left(\frac{\text{time}}{\delta}\right) \times lanes$, where δ is the discretized time step value.

Although this is an efficient reduction of problem size, it is important to note the loss of fine-grained control with such an approach. This is due to all arrival and departure information from time steps n to $n + \delta$ being collated within one array cell.

Alongside the extended QTM model, [16] laid out further optimizations by using a rolling horizon to consider just a small problem window at a time. This drastically reduces the problem space. Furthermore, when running the model on networked intersections, the model was run on subsets

of the network in parallel. Although successful in reducing the computational cost of solving the issue, it was acknowledged by the researcher that this approach would inevitably produce only a locally optimal solution to the problem.

Model Evaluation

Across the literature, there are notable flaws in the data used to evaluate model results. No papers studied made use of real-world data and instead opted to model traffic arrivals assuming either a linear [16][4][21] or Poisson [33] distribution. These distribution methods do not effectively capture the stochastic and variable demand that modern RTNs encounter. Future studies should look to include real-world data when evaluating their models. All studies reported levels of success in developing a model that can find (near) optimal solutions to the TSC problem; however, all studies also cited the computational times as being the key barrier to deploying these solutions to the real world. One interesting set of results comes from [33], who experimented with the level of control the model had over signal green times. Three different scenarios were tested: model has full control over a phase's green time, green time is constrained to an exact integer value, green time is calculated using Webster's formula [34]. It was recorded that giving the model full control over green time gave the most optimal solution.

Conclusion

The literature review has shown that studies often take a similar approach to problem representation. Common themes include modelling the environment macroscopically and utilizing binary decision variables to represent signal status. Whilst having been proved through successful results as a correct representation of the problem, the large problem space and large number of binary variables lead to a high computational cost being associated with the problem. Although certain optimization attempts have been made and were successful in reducing the computational complexity, reductions were not enough to allow for the modelling of additional microscopic constraints. The omission of modelling microscopic factors and the lack of real-world data used in previous studies means that models are not an accurate reflection of the real-world problem seen today. Future work should focus on modelling the problem as one with fewer binary decision variables, consider the inclusion of more realistic constraints where possible, and introduce real-world datasets for model evaluation.

Chapter 4

METHODOLOGY

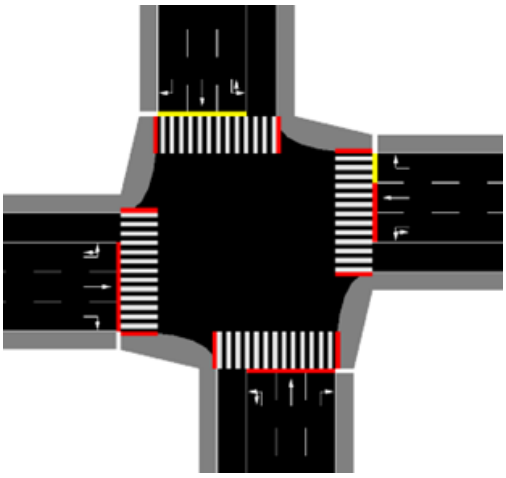

Figure 4.1: A 4-way intersection with 3 incoming lanes and 1 outgoing lane per direction.

The TSC problem has been applied to a variety of different road networks and environments. This paper focuses on the most common one: a classic 4-way intersection with 3 incoming lanes and one outgoing lane in each way (shown above). Across each incoming way is also a pedestrian crossing. With this configuration, there are 16 traffic lights that must be configured as one of three potential colours to form the phases.

4.1 Agent Design

State

The state representation of the agent combines both continuous and discrete factors to effectively represent the wide range of road actors present in a simulation at time t. The state of an agent at a given time t can be defined as:

- **Queue lengths** – The number of cars currently waiting in queue j at time t. A car is only defined as 'waiting' if its current speed is zero.
- **Phase status** – A binary array representing whether a light is green or not at time t. Since the state is captured before a phase transition occurs, information on yellow phases is not relevant here.

- **EMV distance** – The distance of the nearest Emergency Vehicle (EMV) approaching from each direction, or 0 if no EMV is present.
- **Pedestrian flag** – A binary value indicating whether the number of pedestrians at a crossing exceeds zero.
- **Enhanced environmental data** – To satisfy one of the experimental goals of this paper, investigations were carried out upon the effect of providing the model with the expected number of vehicles arriving per lane before the next action decision. Results of this experiment are discussed in Section 6.2.

  A lookahead value of 30 time steps was chosen for this parameter. This was intentionally equivalent to the amount of time steps between model inference times, as information past this point would be covered by remaining state variables.

As highlighted by [41], data normalization helps enhance numerical stability of neural network-based models. Therefore, all continuous state information was normalized using L1 array normalization (Equation 4.1) to normalize values between a range of 0–1.

$$x_{\text{norm}}[i] = \frac{x[i]}{\sum_j x[j]} \tag{4.1}$$

This is with the exception of EMV distances, which were normalized to be between 0 and 1 according to a known upper and lower bound (0 and lane length).

Actions

The action space for the model consists of 36 different traffic light phases. Phases are not constrained to a cyclical structure, but each phase is forced to be active for 15 simulation time steps.

As observed by [44], providing models with a large action space will be more expensive in both time and computational resources when training the model to converge on an optimal policy. For the environment defined above, there are 16 traffic lights, each with 3 possible states (red, yellow, green). This results in 16^3 (4096) possible traffic light configurations. However, the majority of these are either unsafe or inefficient. The following assumptions and constraints have been applied to reduce the action space down to 36:

- **Mandatory Yellow Phase:** Every green phase of a traffic light must be preceded and succeeded by a yellow phase in the interests of driver safety. This is handled externally by the environment; the model does not need to predict yellow values. This reduces the action space size to 256 (16^2).

- **Conflict-Free Routes:** Routes that conflict with one another (e.g., vehicle paths from two different routes that cross) should not be active simultaneously, as this would result in crashes or gridlock. A brute-force approach was used to calculate and remove 45 phases, leaving an action space of size 211.
- **Maximizing Active Routes:** It can be assumed that the most efficient implementation will have the maximum number of routes (4) active at the same time. This assumption further narrows down the action space to just 36 potential phases the model can select.

Reward

As concluded by [22], waiting-time-based reward functions result in increased mean-vehicle speed as well as a reduction in CO_2 emissions. Waiting-time centred reward functions were also noted to outperform vehicle-speed based rewards and emission-based reward functions across a span of metrics. Thus, the reward for the model is defined as follows:

$$R = \min \left(w_{\text{cars}} + w_{\text{emvs}} + w_{\text{peds}} + b_{\text{collisions}} \right)$$

Where:

- $w_{\text{cars}} = \textit{wait-time}\,(C_{\text{start}} \cap C_{\text{end}})$, the total waiting time of cars present in the simulation at the start or end of the phase.
- $w_{\text{emvs}} = \textit{wait-time}\,(E_{\text{start}} \cap C_{\text{end}})$, the total waiting time of emergency vehicles present at the start and end of the phase.
- $w_{\text{peds}} = \textit{wait-time}\,(P_{\text{start}} \cap C_{\text{end}})$, the total waiting time of pedestrians present at the start and end of the phase.
- $b_{\text{collisions}}$, a penalty applied if any collisions occur during the phase.
- $C_{\text{start}}, C_{\text{end}}$, the sets of cars present in the simulation at the start and end of the phase, respectively.
- $E_{\text{start}}, E_{\text{end}}$, the sets of emergency vehicles present at the start and end of the phase, respectively.
- $P_{\text{start}}, P_{\text{end}}$, the set of pedestrians present at the start and end of the phase, respectively.
- $\textit{wait-time}(\cdot)$, a function that computes the total waiting time for the given set.

The reward function is defined so as not to penalize the model for actors that entered the simulation during the execution of the traffic phase (i.e. actors the model was previously unaware of). It gives the model an incentive to minimize pedestrian and vehicle wait times. In addition to this, a collision penalty is applied to reward safe routing of emergency vehicles. Some pre-existing work has implemented a squared reward function, to penalize longer waiting times more; however, it was observed that this would cause numerical instability in the CIPHER model and was therefore not included.

4.2 Linear & Discrete Optimization

Model 1

The literature review concluded that a valuable contribution to the field would be to formulate the TSC problem as one with fewer binary variables, as this would allow for the modelling of more precise and realistic constraints. We attempt to define a Linear Optimization model with the following configuration:

Constants

- T: Total simulation time
- E: Number of lanes
- K: Maximum number of on/off periods per light
- $A_{e,t}$: Number of vehicles arriving to lane e at time t
- γ: The saturation rate, e.g. the fraction of a vehicle that can leave per time step when a light is green
- Γ: The saturation rate per time step for when a light is yellow.
- $C_{e,e'}$: Conflict matrix where $C_{e,e'} = 1$ means signals e and e' conflict

Decision Variables

An initial attempt was made to propose a new model that utilized two continuous decision variables to indicate whether a signal was active or not.

- $x_{on,e,k} \geq 0$: Start time of green phase for lane e at activation k
- $x_{off,e,k} \geq 0$: End time of green phase for lane e at activation k

Additional helper variables for tracking queue length and departures were also used:

- $q_{e,t} \geq 0$: Queue length at lane e at time t
- $d_{e,t} \geq 0$: Number of vehicles departing from lane e at time t

Constraints

An additional constraint not seen in the pre-existing literature was also needed, to ensure a light cannot reactivate until after it has been switched off.

$$x_{off,e,k} \leq x_{on,e,k}, \quad \forall e \in \{1,\ldots,E\}, \forall k \in \{1,\ldots,K\} \tag{4.2}$$

Also, conflicting signals cannot be active at the same time:

$$x_{on,e,k} \geq x_{off,e',m} \quad \text{or} \quad x_{on,e',m} \geq x_{off,e,k}, \quad \forall e, e' \in \{1,\ldots,E\}, \forall k, m \in \{1,\ldots,K\} \text{ where } C_{e,e'} = 1 \tag{4.3}$$

Queue Evolution

Using enhancements proposed by [16], it can be concluded that for our network, Daganzo's [7] CTM will suffice, as there are no diverging roadways in our network. Therefore, our queue evolution strategy is as follows:

$$q_{e,t+1} = q_{e,t} + A_{e,t} - d_{e,t}, \quad \forall e \in \{1,\ldots,E\}, \forall t \in \{1,\ldots,T-1\} \tag{4.4}$$

Departure Constraints

Departures from a lane cannot exceed queue length:

$$d_{e,t} \leq q_{e,t}, \quad \forall a \in \{1,\ldots,E\}, \forall t \in \{1,\ldots,T\} \tag{4.5}$$

However, issues were encountered with the model when trying to calculate $d_{e,t}$:

$$d_{e,t} \leq \gamma \cdot \sum_{k=1}^{K} (t \geq x_{on,e,k} \wedge t < x_{off,e,k}), \quad \forall e \in \{1,\ldots,E\}, \forall t \in \{1,\ldots,T\} \tag{4.6}$$

With linear optimization techniques, it is not possible to implement comparisons between continuous decision variables. This means it cannot be known when a light is active. Therefore, it was concluded that it was not possible to implement the TSC in a continuous manner and this approach was unfortunately abandoned.

Model 2

Following the failed experimentation of the previous model, a new formulation was created with the aim of applying certain noted optimizations to allow for modelling of a light being yellow, whilst still representing the decision variables as binary. We reuse the constants defined earlier, with the exception of K and the introduction of two binary decision variables:

- $x_{E,T} \in 0, 1$: Whether signal E is green at time T.
- $y_{E,T} \in 0, 1$: Whether signal E is yellow at time T.

Constraints (4.5), (4.4), as well as all helper variables were reused.

Constraints

Constraint (4.3) was rewritten to accommodate the new formulation:

$$\forall e_1, e_2 \in \text{Lanes}, \forall t \in \text{Time}, \quad C[e_1][e_2] = 1 \Rightarrow x[e_1][t] + y[e_1][t] + x[e_2][t] + y[e_2][t] \leq 1 \tag{4.7}$$

Alongside a constraint to enforce the yellow phase both before and after a phase transition:

$$\forall e \in \text{Lanes}, \forall t \in \{1, \ldots, T-1\}, \quad y_{e,t} \geq x_{e,t-1} - x_{e,t}, \quad y_{e,t} \geq x_{e,t+1} - x_{e,t} \tag{4.8}$$

Finally, the problematic constraint (4.6) was rewritten with the new binary variables:

$$d[e][t] \leq \Gamma x[e][t] + \Upsilon y[e][t] \tag{4.9}$$

Objective Function

An objective function focused on minimising cumulative queue length was leveraged for this model:

$$\min \sum_{t \in \mathrm{T}} \sum_{e \in \mathrm{E}} q[e][t] \tag{4.10}$$

Optimizations

Owing to the problem now having twice the number of binary decision variables, optimizations needed to be introduced. Alongside the simplified CTM being employed, it was decided that time steps would be discretized to intervals of 5, as initially proposed by [33]. For an instance of the problem with $E = 16$ lanes and $T = 1000$, this reduces the number of binary decision variables from 16,000 to 3,200.

Model 3

The literature review also suggested improvements in the types of actors represented in the model. Therefore, this formulation was extended to include the same actors as the reinforcement learning model described earlier: pedestrians and EMVs.

This required extending Model 2 with the following constants and decision variables:

- P: Number of Pedestrian Crossings
- $ped\_q_{p,t} \geq 0$: Number of pedestrians at crossing p at time t
- $ped\_d_{p,t} \geq 0$: Number of pedestrians departing from crossing p at time t
- $emv\_q_{e,t} \geq 0$: Number of emergency vehicles in queue e at time t
- $emv\_d_{e,t} \geq 0$: Number of emergency vehicles departing from lane e at time t

Constraints

The following constraint was defined to mimic EMV priority:

$$\forall e \in \mathrm{E}, \forall t \in \mathrm{T}, \quad d[e][t] \leq \mathrm{emv\_q}[e][t] \tag{4.11}$$

And to mimic pedestrian flow:

$$\forall e \in \mathrm{P}, \forall t \in \mathrm{T}, \quad d[e][t] \leq x[e][t] \cdot \upsilon \tag{4.12}$$

Where υ is a constant $\geq$ the maximum expected pedestrian queue size.

Objective Function

Finally, the objective function was updated to reflect this:

$$\min \sum_{t \in \mathrm{T}} \sum_{e \in \mathrm{E}} (q[e][t] + \mathrm{emv\_q}[e][t]) + \sum_{t \in \mathrm{T}} \sum_{p \in \mathrm{P}} P[p][t] \tag{4.13}$$

Various iterations of this model were used to feed into training decisions for the RL-based model described previously, as well as for evaluating the effectiveness of the final model.

It is important to note that this model only gives an approximate representation and does not model all micro/macroscopic factors seen in SUMO. Factors such as driver imperfection behaviour, lane-changing and travel time from a car's entry point to the end of a queue are all not represented here, therefore it is to be expected that the result of the LO model is actually slightly lower than the true optimal value.

Chapter 5

IMPLEMENTATION

5.1 RL Agent Implementation

As concluded by [1], SUMO is an excellent choice of simulator for designing TSC algorithms. Due to its open-source flexibility, GUI and strong integration capabilities, it was decided that SUMO would be used to model the environment within which the agent operates. The lack of extensive real-world data sources meant that data augmentation strategies had to be used for training the agent. Vehicles were generated according to a Poisson distribution (Equation 5.1) with a mean inter-arrival rate of 2 simulation time steps.

$$P(k) = \frac{\lambda^k e^{-\lambda}}{k!}, \tag{5.1}$$

Where k is the number of vehicles expected to arrive within λ time.

It was noted across the literature that there was a lack of simulation parameters provided to the reader which limits reproducability. To alleviate this, all relevant simulation parameters are included below in Table 5.1.

Variable Name	**Description**	**Value**
`GREEN_TIME`	How long a traffic light stays green (s) for	15
`YELLOW_TIME`	How long a traffic light stays yellow for (s)	5
`NUM_VEHICLES`	The number of vehicles that arrive during the simulation	400
`ARRIVAL_RATE`	The mean inter-arrival time between two vehicles in the simulation	2
Vehicle Acceleration	The acceleration ability of vehicles [m/s^2]	0.8
Vehicle Deceleration	The deceleration ability of vehicles [m/s^2]	4.5
Emergency Vehicle Speed Factor	The percentage by which emergency vehicles can exceed the road's speed limit	50%
Vehicle Sigma	The driver imperfection score	0.5
Vehicle Length	Vehicle length [m]	5
Vehicle Max Speed	The vehicle's maximum velocity [m/s]	70
Lane Speed	The maximum speed allowed on this lane [m/s]	11.11
Node Offset	The distance of each edge's start points from the central junction	500

Table 5.1: Simulation parameters

All other values not stated can be assumed to be the SUMO defaults and can be found in the SUMO documentation [23].

Traffic Signal Control Intricacies

As described in Section 4.1, numerous steps were taken to provide the model with an action space small enough to allow for rapid convergence on a policy, whilst the mandatory 36 other yellow phases would be enforced by the environment wrapper itself. However, often two successive phases will have green lights in common, switching these lights to yellow and back to green again would unnecessarily slow down cars and reduce junction throughput. A specific light control module was developed to negate this; a system was built off base-3 arithmetic to efficiently represent over 443 unique phases and allow the environment wrapper to select the most efficient one according to the agent's action, whilst maintaining the RL agent's small action space.

Code

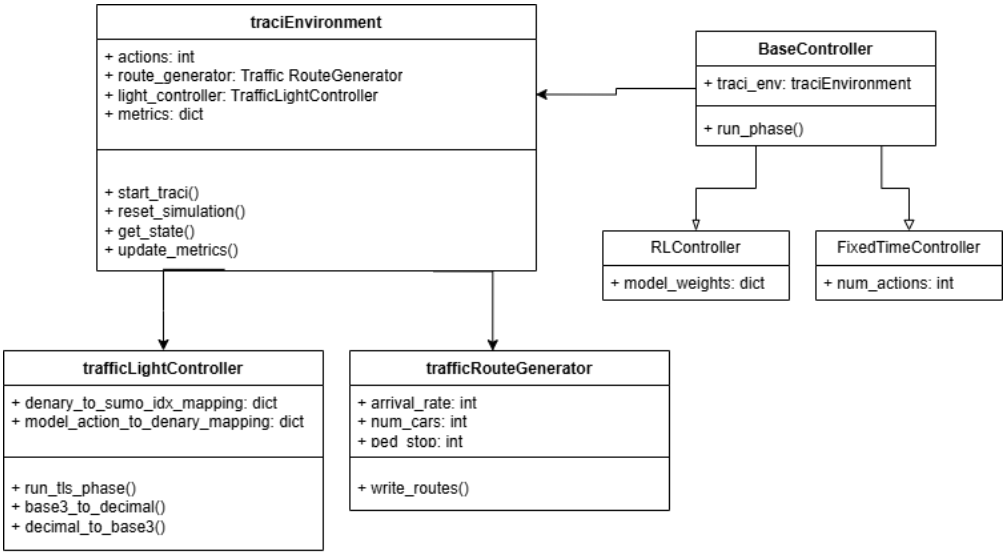

Figure 5.1: A UML diagram showing the structure of classes within the project.

Communication through SUMO was originally handled via the Traci API, a web-socket based library provided by the SUMO developers for retrieving and updating various simulation parameters. However, due to the high communication overhead caused by the socket-based protocol used, the project was migrated to LibSumo (a more efficient C++ API). This change significantly reduced training times for the model. The core implementation was written in Python, leveraging LibSumo to interface with the simulation environment. The programme consists of the following core classes:

- Environment Wrapper: Provides a standardized interface for the RL agent to interact with SUMO.

- Simulation Runner: Loads a specific traffic signal control (TSC) policy and steps through the simulation, with an optional GUI for real-time visualization.
- Traffic Signal Controller : An implementation of the previously described traffic light control system.
- Traffic Route Generator : Automatically generates and writes the data needed for SUMO to generate cars according to a Poisson distribution.
- Controller Modules: Two controllers were implemented for debugging purposes and experimentation:
  - Fixed-Time Controller: Operates on a predefined cyclical schedule.
  - RL-Based Controller: Selects actions based on a trained reinforcement learning model.

The framework for my DRL model was taken from the PyTorch Docs [25] and adapted to function in my chosen environment. Instances of the model were originally trained locally, but soon computational demand grew and training was offloaded to the university's GPU cluster for more efficient training. The model comprises of 3 hidden layers, each with 128 neurons per layer and a ReLu activation function. The agent was trained on a different set of randomly generated arrival times each iteration, to avoid being over-fit to one specific arrival pattern. It is important to note that mean-inter-arrival times between cars were kept consistent across the entire training process to ensure numerical stability of the reward function.

Reinforcement Learning Technical Challenges Encountered

The main technical challenge encountered over the course of development was the training times for each instance of the model. As state space grows the model has more weights to tune with each iteration, and back propagating this correctly is costly. Furthermore, due to the complex environment within which the agent is operating, convergence on an optimal policy would take time. Training one instance of the model can take up to 24 hours in some cases.

This effect was negated by optimizing the code where possible. As noted earlier, upgrading to the LibSumo API reduces the communication overhead in interacting with the simulator. Other optimizations were also implemented, such as limiting batch size initially during training and only scaling this number up when an acceptable model configuration was discovered. These changes helped reduce training times, but model training times were still a big technical blocker throughout the course of the project. Another technical challenge was the frequent experimentation needed between minor different versions of the model. Successive model versions would often only have

small tweaks of independent variables such as the state information, reward function, or hyper-parameters. However, changing variables in isolation was a necessity due to the highly sensitive nature of the environment in which the agent is deployed. To minimize the time spent comparing two models, a comprehensive metric logging system was created. The system automatically collects and graphs metrics surrounding cars, EMVs, Pedestrians and the environment as a whole. This allowed for easy comparison of results visually, in order to decide which is the better model.

5.2 Linear & Discrete Optimization Implementation

Language Choices

The model itself was implemented using OPL, a modelling language for combinatorial optimisation, and CPlex [19] as the solver. The model was programmed according to the specification outlined in Section 4.2 with one minor modification: the pedestrian and car departure arrays were aggregated together for simplicity, as the majority of constraints shared are the same. A similar approach was taken with the pedestrian and car arrival vectors, to reduce redundancy without altering model functionality.

Rolling Horizon

As initially proposed by Guilliard [16], a rolling horizon approach was utilised to drastically reduce processing times. A planning-horizon of 100 time-steps was used, with a control horizon of 50 time-steps. The rolling horizon itself was originally attempted within CPlex itself, but later undertaken in Python due to its simplicity. Across the literature, it is common to use libraries such as Pulp to allow for direct modelling of Linear Optimization problems within Python. However, this was decided against due to the slow execution times of Python in comparison to the C++ based OPL. Instead, Python was used to iteratively execute the OPL model via the command line and feed queue lengths at the end of one iteration into the start of the next.

One key limitation of this approach is that this will result in only a locally optimal solution as each iteration only optimizes over a subset of the entire problem space. However, this trade-off allows the model to process significantly longer time periods, thus allowing for modelling of more intricate simulations.

Chapter 6

RESULTS

This section of the paper details the training process for the models and how results from the Linear Optimization model were used to feed into this. It then details a systematic comparison of travel times obtained from CIPHER and 6 other TSC models, before finally giving a final evaluation of CIPHER and its capabilities when handling cars, EMVs and pedestrians.

6.1 Linear & Discrete Optimization Model Input

When training the models, a hybrid approach was taken to avoid expensive tuning of the green time hyper-parameter, results from the linear optimization model were analysed to influence this variable. Green time dictates the fixed amount of time steps that a light is green for. Whilst a shorter green time will give the model finer control over the simulation and its decisions, it will also result in more inference calls on the model. This in turn increases training times. Therefore, careful balancing of this parameter is essential.

The linear optimization model described in Section 4.2 was run using a synthetic arrivals dataset and the results analysed. Vehicles were generated according to a Poisson distribution, with a mean inter arrival rate of 0.5 units. Vehicles had an equal chance of spawning across all lanes. Meanwhile, pedestrians were changed according to a linear distribution with an equal chance of spawning on all crossings.

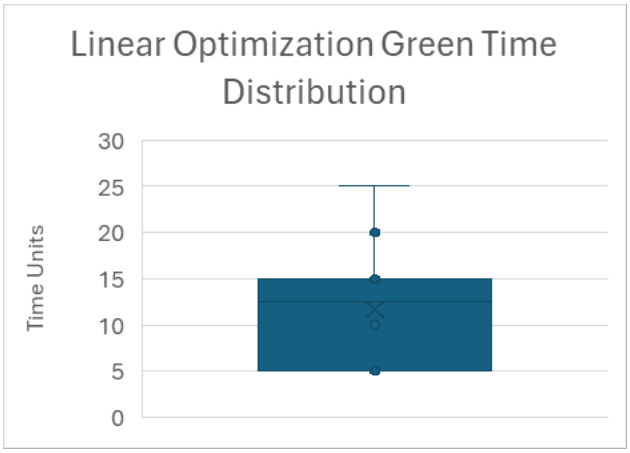

Figure 6.1: A box plot showing the distribution of green time selected by the LDO model.

Figure 6.1 shows the distribution of green time as chosen by the linear optimization model, from this we can conclude that the most optimal green time duration lies in-between 10-15 time units. Therefore, a green time of 15 time units was selected to balance model control and computational efficiency. This value was hardcoded into the final CIPHER model. Leveraging the optimal nature of the LO model negated the need for costly hyper-parameter tuning within the RL agent itself.

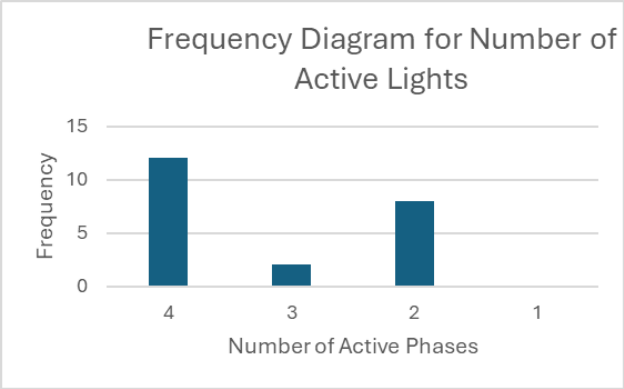

Figure 6.2: A bar chart showing the frequency of total phases active at a given time, as chosen by the LDO model.

Another notable set of results from this running of the LO model is shown in Figure 6.2. Post-processing of the linear optimization data was conducted to analyse the number of lights active at each time step. The graph shows the frequency that each number of lights was active for. This chart supports the assumption made in Section 4.1, that most of the optimal phase configurations will have 4 lights active at a time.

6.2 Comparison Against Other Models

As noted in the literature, there are no pre-existing models which account for emergency vehicles and pedestrians in the road network, meaning direct comparison between CIPHER and similar models was not possible. However, a simplified version of CIPHER was taken for comparison. This simplified model only considered cars in its state representation. This could at least serve as a basic measure of the effectiveness of the model's design. To assess the effectiveness of CIPHER and CIPHER+, 6 TSC models were chosen for comparison, covering a range of traditional, adaptive and RL-based techniques:

- Fixed – The static fixed time algorithm used in the majority of road networks today.
- Adaptive – Adaptive, mathematical-based models. Namely: Websters, SOTL and MaxPressure.
- Adaptive RL – Two RL-based algorithms were also included in the comparison. Both algorithms used a deep reinforcement learning framework.
  - DDPG is a RL based algorithm that allows for selection of both the next phase, and its cycle length.

- DQN works similarly to CIPHER, simply selecting the next best phase and keeping it active for a hardcoded cycle length.

Both RL agents' training data was generated using a non-homogeneous Poisson distribution. Meanwhile the state representation consists of lane density and the last green phase, the reward functions are both focused on minimizing total vehicle delay. Further implementation details of all models mentioned and the source code can be found at [13].

All models were evaluated using a real-world dataset, obtained from a traffic intersection in Hangzhou China [36][35][43]. The original dataset comprises of vehicle arrivals and routes over a one-hour period. Arrival times from this dataset were then linearly scaled down to half an hour and 15 minutes respectively, this allowed for testing of models across 3 different traffic demands: low, medium and high. All models were evaluated based on vehicle travel times.

Low Density Comparison

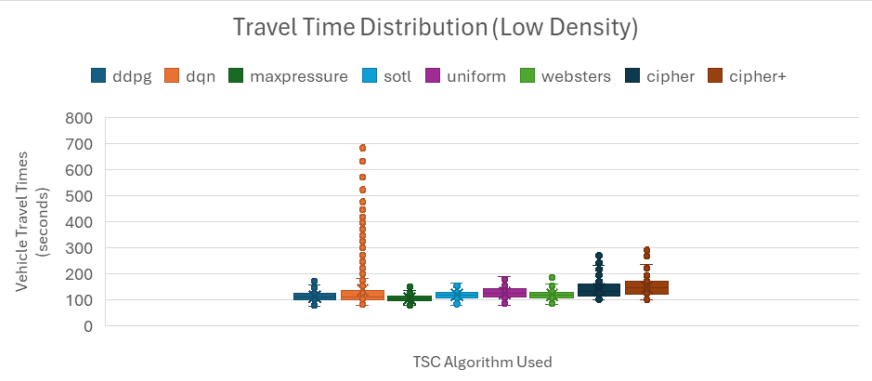

Figure 6.3: A boxplot showing the distribution of travel times received from various TSC algorithms when run on a simulation with a mean inter-arrival rate of 1.2.

Figure 6.3 presents the distribution of travel times from each model in a low-density (1.2 second mean inter-arrival rate) environment. CIPHER and CIPHER+ exhibit the longest mean travel times among the eight models, indicating a sub-optimal performance. However, both models also result in fewer extreme outliers in comparison to DQN, suggesting that whilst they may increase average travel times, they also result in fairer wait times than some RL-based methods.

Another relevant observation is the narrower inter-quartile range of CIPHER compared to CIPHER+, implying that the inclusion of future arrival data does not enhance the model's ability to optimize traffic flow.

Medium Density Comparison

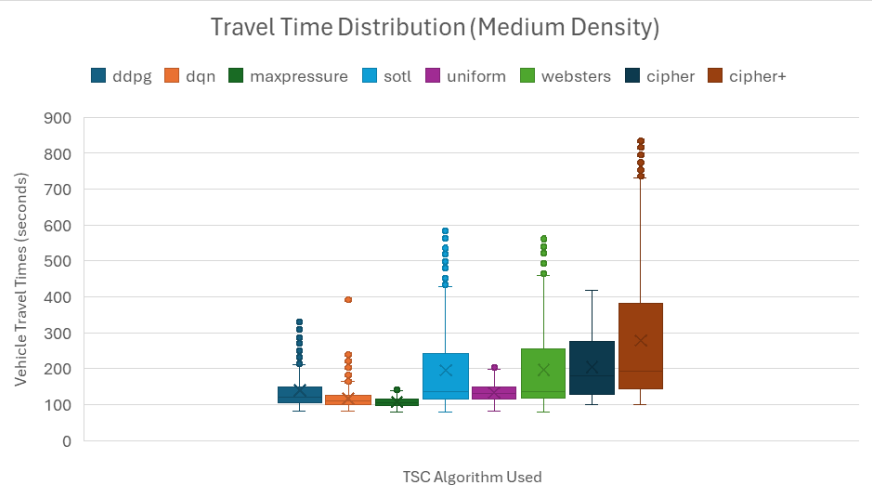

Figure 6.4: A boxplot showing the distribution of travel times received from various TSC algorithms when run on a simulation with a mean inter-arrival rate of 0.6.

Figure 6.4 showcases the performance of algorithms during a medium density simulation. From this we can see DDPG, DQN and CIPHER all achieving superior results at higher congestion levels, in comparison to Websters and SOTL. Meanwhile, MaxPressure still serves as the most optimal control method.

High Density Comparison

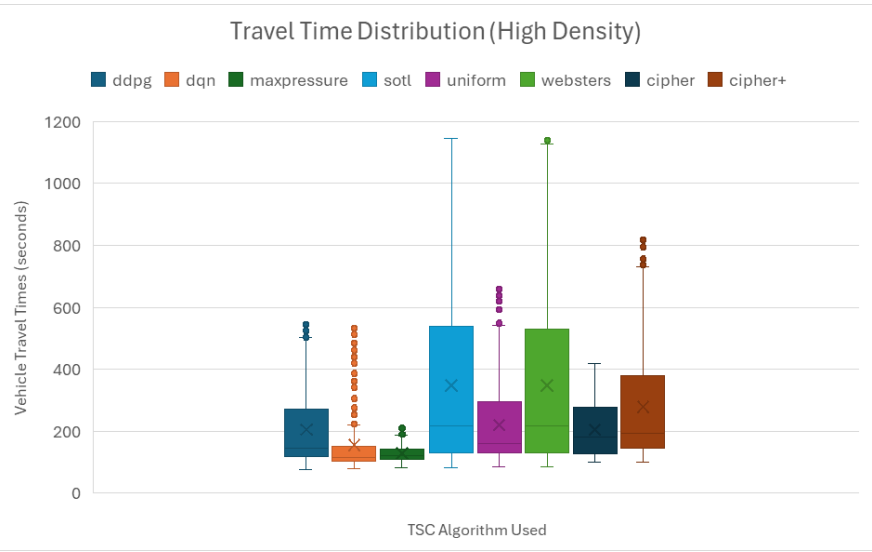

Figure 6.5: A boxplot showing the distribution of travel times received from various TSC algorithms when run on a simulation with a mean inter-arrival rate of 0.3.

Figure 6.5 showcases the performance of algorithms during a high density simulation. The results show that most RL-based methods (DDPG, DQN and CIPHER) are able to outperform Websters and SOTL at higher congestion levels. Interestingly this does not hold true for CIPHER+, which exhibits significantly worse performance than CIPHER, reinforcing the hypothesis that future arrival data is not relevant state information.

Comparison of Carbon Emissions

As one final metric of model effectiveness, the environmental impact of each algorithm was assessed by calculating the estimated CO_2 emissions. Under the assumption that an idling vehicle consumes 0.5 gallons of fuel per hour [15], and each gallon produces 8887 grams of CO_2 [31], the total emissions per scenario were computed and shown in Table 6.1. In this table, the emissions from CIPHER serve as a baseline for comparison.

Algorithm Used	Low	Medium	High	Total	Change (%)
DDPG	135.602	169.929	248.1427	553.67	-17%
DQN	166.584	141.463	187.887	495.93	-26%
MaxPressure	127.262	129.958	155.2297	412.45	-38%
SOTL	143.342	236.417	390.6487	770.41	16%
Uniform	151.781	161.169	267.0762	580.03	-13%
Websters	143.848	239.476	389.5698	772.89	16%
CIPHER	169.99	248.51	248.5099	667.01	0%
CIPHER+	179.143	338.254	338.2538	855.65	28%

Table 6.1: Performance comparison of traffic signal control algorithms under different traffic densities.

Discussion of Comparative Results

Naturally, the results from Table 6.1 correlate with the box plots observed previously. We can see CIPHER resulting in fewer emissions than Websters and SOTL, whilst being outperformed by MaxPressure, Uniform time control, DDPG and DQN. Cipher+ was the worst performing algorithm in comparison, resulting in 28% more emissions. Combined with the overwhelming data from the boxplots, we can conclude that CIPHER outperforms CIPHER+ across all 3 testing instances, showcasing that future arrival data is not an effective state representation factor for RL based TSC algorithms.

CIPHER's inferiority to the two other RL algorithms studied could be attributed to various factors. e.g., the more sophisticated state representation DDPG and DQN had, which used lane density as opposed to queue length. This metric gives a more general representation of how traffic is moving across the entire lane, as opposed to just capturing vehicles that are queueing. Alternatively, the

improved results could be attributed to the models' strategy used for generating training data: a Poisson distribution shifted according to a sine wave, used to model fluctuating demand across running of the simulation.

One thing evident across all 3 charts is the superiority of the adaptive MaxPressure algorithm in comparison to all other algorithms shown here. Furthermore, it would be trivial to extend the MaxPressure implementation to account for pedestrians. However, unlike CIPHER, it is still unable to dynamically respond to EMVs and adapt in a way that minimizes their travel times over others.

6.3 Final Model Evaluation

The final CIPHER model was retrained, with the inclusion of state and reward factors that represent pedestrians and emergency vehicles. Similarly to the vehicle-only models, this version was trained on new instances of arrival data with every iteration, to improve model generalisation. Pedestrians arrived at a linear rate of 1 pedestrian every 12 seconds whilst vehicles arrived according to a Poisson distribution with a mean inter-arrival rate of 2 seconds. EMVs had a 2% chance of spawning.

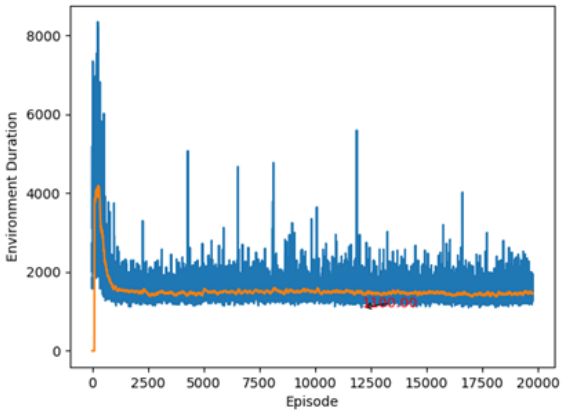

Figure 6.6: A line graph showing the number of simulation time-steps it took each iteration of CIPHER to process 400 incoming vehicles over a set time-period.

The model was trained for 20000 iterations, in an attempt to find a more optimal policy. However, as shown by Figure 6.6 training results plateaued after approximately 1000 iterations. Training time for this model took 18 hours, 25 minutes and used 2.66GB of RAM, the graph indicates that such drastic training times were not necessary, and the model developed an optimal policy long before this.

The model was evaluated using a synthetic dataset of the same data generation strategy as its training data. Naturally this draws questions about overfitting, but due to the lack of real-world data involving all 3 road actors present, this was a necessity. Waiting times for each actor present in

the simulation were collected and analysed, Figure 6.7 shows these results. It is evident CIPHER correctly prioritizes EMVs first, wait times for these vehicles range between 0 and 33 seconds. CIPHER is also able to provide pedestrians with very low wait times. The high difference between wait times for cars and pedestrians does raise concerns about the fairness of the system, this is most likely attributed to the fact that pedestrian crossings are not subject to the same bottleneck as car lanes: in that only one car can leave a lane at a time, but many pedestrians can cross together.

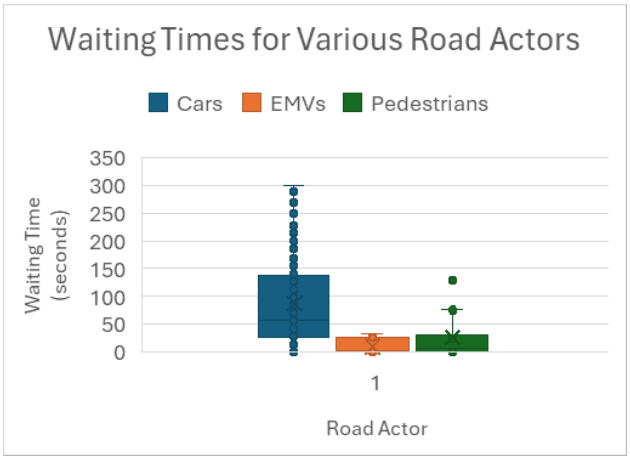

Figure 6.7: A boxplot showing the distribution of travel times observed across all 3 road actors present in CIPHER evaluation.

To evaluate the final model's optimality, synthetic datasets were generated with varying mean inter-arrival times (0.3, 0.6 and 1.2s) between cars, to simulate varying traffic densities. Due to the limited availability of metrics from the LO model, both models were evaluated on the number of seconds it took to process 400 vehicles. Table 6.2 shows this data.

Density	**RL Model**	**LO Model**	**% Diff**
Low	1375	1155	19%
Medium	1550	1180	31%
High	1250	1200	4%

Table 6.2: Comparison of RL and LO models across different traffic densities.

Reiterating from Section 4.2 it is important to note that the results from the Linear Optimization model will only be of a local optimum, due to the rolling horizon approach undertaken. Furthermore, the lack of microscopic constraints modelled means that this is not a truly equal comparison to CIPHER. Nevertheless, we can see the LO model performed fairly consistently across all scenarios, CIPHER shows some fluctuations between the 3 scenarios, indicating more training would be

required for the lower and medium densities. On the high density a time difference of just 4% was noted between the two models, showcasing the models potential in high congestion scenarios.

From this data, and the box plot shown previously, we can conclude CIPHER is able to effectively manage and direct traffic flow across an intersection, especially in high density situations. It can also be concluded that CIPHER is able to give attention and fairness to pedestrians, EMVs and regular cars simultaneously.

6.4 Conclusion

From the results presented in this paper, it can be concluded that future arrival data is not a useful vector to include in an RL agents state space. The worse results suggest this data simply serves as increased model noise and that CIPHER outperforms CIPHER+. Furthermore, it is evident that whilst RL agents can out-perform some adaptive algorithms, MaxPressure is still superior when using vehicle travel times as a metric. Notably, an adaptive algorithm's shortcomings lie in the fact it cannot dynamically respond to EMVs to give priority. This paper shows it is possible to design RL agents capable of handling such events, whilst acknowledging the compromise in-terms of overall performance.

Additionally, we show that LO models are capable of handling the same circumstances, and with the implementation of certain optimizations such as batching, a simplified CTM and discretising time steps, it is possible to explicitly model yellow transitions. This in turn results in more realistic results. Despite these optimisations, the large number of binary variables mean that high inference times remain a challenge, making real-time deployment unrealistic.

The charts also show the Uniform (Fixed Time) controller often being on par with or outperforming the more advanced adaptive methods. It is important to note that distribution of cars in the real-world dataset was moderately symmetrical, and the Fixed time algorithm would not be expected to perform so well when arrivals aren't evenly distributed across all lanes.

6.5 Future Directions

This paper has set the foundations for several potential future directions. The results show the potential of RL agents against the limitations of the classical adaptive algorithms, whilst acknowledging that they currently fall short. Incorporation of traditional adaptive elements into an RL agent, such as a reward function based on minimizing pressure, could produce interesting results. The paper has also proposed a first of its kind system that is able to handle junctions holistically, considering all road actors present without explicitly favouring a subset. Further work that aims to build on and improve the foundations of CIPHER would help improve the models use cases.

Alternatively, one crucial topic not discussed in this paper is deployment of CIPHER to a real world

environment, future work could focus on integrating the model with vision models to calculate state data, as opposed to exploiting the perfect knowledge and easy access to data CIPHER has when integrating with SUMO.

The scarcity of real-world training data for RL agents remains a challenge. CIPHER was mostly outperformed by the other RL algorithms studied; this could potentially be due to the more sophisticated approach taken in generating vehicle arrival data. Future work should systematically evaluate various traffic distribution models to determine their effect on RL training and how it impacts the model's overall generalization capability.

Finally, the development of a more comprehensive linear optimization model will help guide and evaluate future RL solutions to the problem. Whilst our LO model provided valuable insights, reducing the reliance on binary variables even further is critical to achieving scalability. Future work could explore other mathematical optimization methods such as dynamic programming, or a heuristic approach.

Chapter 7

SUMMARY & REFLECTIONS

7.1 Project Management

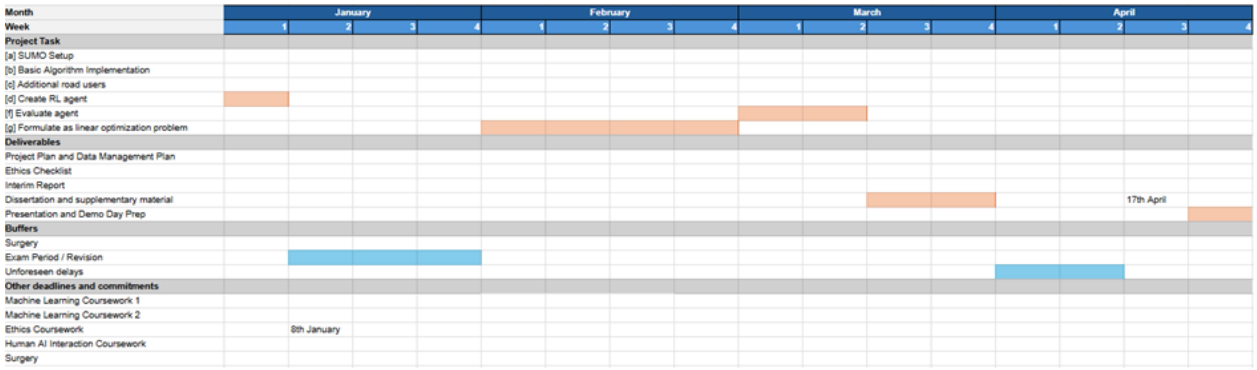

Figure 7.1: A Gantt chart created in November, showing expected timescales for various tasks in the project.

Overall, I am very happy with how the project was managed. Figure 7.1 shows a portion of the Gantt chart that was initially proposed as part of my interim report. The chart shows I expected to finish all project work and have my final report written up by the end of March. This estimation was correct, and I was able to finish all work in the allocated timeframe, allowing me to use my two-week buffer to iterate on my report and make small final improvements.

Although I finished on time overall, I did need to adjust my timelines on individual tasks to meet this deadline. I was expected to be finished with the RL agent in early January. Unfortunately, due to the unexpectedly high model training times this did not happen. I found myself making small tweaks and retraining the model until the middle of February. This meant I was often working on other tasks in parallel, particularly the Linear Optimization model. I would alter the model, retrain, and work on my Linear Optimization model whilst waiting for the model to finish training.

Due to me only being able to run one job on the GPU cluster at a time, I needed to minimize my downtime to ensure I constantly had a job running: this was done by queueing successive alterations of my main script to allow a new version of the model to begin training as soon as the previous one had ended.

7.2 Contributions & Reflections

Perhaps one of the more novel ideas presented in this paper was the idea of using the near-optimal solutions obtained from a Linear Optimization model to feed into the RL agents design, enabling me to save time and GPU cluster resources. Analysis of the results obtained from this model can be used to both influence design (e.g. configuring an optimal green time constraint) and validate my design decisions (forcing the model to always have 4 phases active).

In light of this, if I were to repeat this project I would start by designing the Linear Optimization model, to allow it to influence my design decisions from the very start. For example, I would like to experiment with various objective functions to help influence my agents reward scheme; however, due to me creating the LO model in the last 2 months of the project, this was not possible as I had run out of time.

Additionally, I would aim to keep model training times low by employing optimisation techniques early on. I should have started optimising the training loop as soon as job times started to increase, instead I waited until it was infeasible to continue with the current training times. This cost me valuable time and is the main reason that completing the design of my RL agent overran.

7.3 Legal, Social, Professional & Ethical Issues

From a social context, reduced traffic will result in reduced emissions on our environment [3] because of reduced idle-times and less stop / start driving patterns. The environmental impacts of CIPHER were considered and calculated in Section 6.2. Although CIPHER was outperformed by 4/7 of the algorithms it was compared against, a secondary environmental impact should also be considered: traffic control systems such as CIPHER, that are fair to pedestrians, will result in greater walkability within cities and may encourage more people to walk as opposed to drive to places. This will then also have a direct effect on reducing queue lengths.

From a social perspective it is important to consider the consequences of poor TSC. Improper signalling will result in crashes / emergency vehicle manoeuvring, putting road users at risk. To address this, CIPHER is hardcoded to have a mandatory yellow phase before and after a light turns green, rather than leaving that decision to the model.

Finally, legal liability must be considered. It is important to establish who would be at fault if CIPHER was deployed in a real-world environment and a crash would occur due to improper signalling. Furthermore, for CIPHER to be deployed in a real-world environment it would require systems to collect data on current traffic levels in each lane. If the system were to be deployed, proper GDPR handling would need to be implemented to ensure that the protected data is handled and processed lawfully.

BIBLIOGRAPHY

[1] Eman A. Algherbal and Nedal T. Ratrout. "A Comparative Analysis of Currently Used Microscopic, Macroscopic, and Mesoscopic Traffic Simulation Software". In: *Transportation Research Procedia* 84 (2025), pp. 495–503.

[2] David Applegate et al. *A Practical Guide to Discrete Optimization*. 2014.

[3] Shashank Bharadwaj et al. *Impact of congestion on greenhouse gas emissions for road transport in Mumbai metropolitan region*. 2017.

[4] S. M. A. Bin Al Islam and Ali Hajbabaie. "Distributed coordinated signal timing optimization in connected transportation networks". In: *Transportation Research Part C: Emerging Technologies* 80 (2017), pp. 272–285.

[5] Salah Bouktif et al. "Deep reinforcement learning for traffic signal control with consistent state and reward design approach". In: *Knowledge-Based Systems* 267 (2023), p. 110440.

[6] Panayotis Christidis and Juan Nicolás Ibanez Rivas. *Measuring Road Congestion*. 2012.

[7] Carlos Daganzo. "The cell transmission model, part ii: Network Traffic". In: *Transportation Research Part B: Methodological* 29 (1995), pp. 79–93.

[8] Carlos F. Daganzo. "A variational formulation of kinematic waves: basic theory and complex boundary conditions". In: *Transportation Research Part B* 39.2 (2005), pp. 187–196.

[9] A. G. Dobinson and W. Kwok. "The Sydney coordinated adaptive traffic (SCAT) system philosophy and benefits". In: *IEEE Transactions on Vehicular Technology* 29 (1980), pp. 130–137.

[10] Romain Ducrocq and Nadir Farhi. *Deep Reinforcement Q-Learning for Intelligent Traffic Signal Control with Partial Detection*. 2021. URL: https://arxiv.org/abs/2109.14337.

[11] Martin Fellendorf and Peter Vortisch. *Microscopic traffic flow simulator VISSIM*. Fundamentals of Traffic Simulation, 2011.

[12] Juntao Gao et al. *Adaptive Traffic Signal Control: Deep Reinforcement Learning Algorithm with Experience Replay and Target Network*. 2017.

[13] Wade Genders and Saiedeh Razavi. *An Open-Source Framework for Adaptive Traffic Signal Control*. 2019. URL: https://arxiv.org/abs/1909.00395.

[14] Carlos Gershenson. *Self-Organizing Traffic Lights*. arXiv. 2004.

[15] F. Gromovic. *How much gas does idling use?* https://rerev.com/articles/how-much-gas-does-idling-use/. Accessed: 2025-03-16. 2022.

[16] I. Guilliard. *Mixed Integer Linear Programming for Traffic Signal Control*. 2020.

[17] Z. Huang. "Reinforcement learning based adaptive control method for traffic lights in intelligent transportation". In: *Alexandria Engineering Journal* 106 (2024), pp. 381–391.

[18] Inrix. *INRIX: Congestion Costs Each American 97 hours, $1,348 A Year*. 2018.

[19] International Business Machines Corporation. *CPlex User's Manual for CPlex*. 157. 2009.

[20] S. Kakazu et al. "Genetic reinforcement learning for cooperative traffic signal control". In: *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*. Vol. 1. 10. 1994, pp. 223–228.

[21] M.A.S. Kamal et al. "Traffic Signal Control in an MPC Framework Using Mixed Integer Programming". In: *IFAC Proceedings Volumes* 46.21 (2013), pp. 645–650.

[22] A. Liu et al. "Impact of Reward Function Selection on DQN-Based Traffic Signal Control". In: *IEEE International Conference on Green Energy and Smart Systems*. Vol. 10. 2024, pp. 1–6.

[23] Pablo A. Lopez. "Microscopic Traffic Simulation using SUMO". In: *International Conference on Intelligent Transportation Systems*. 2018, pp. 2572–2582.

[24] A. J. Miller. "Settings for Fixed-Cycle Traffic Signals". In: *Journal of the Operational Research Society* 14 (1963), pp. 373–386.

[25] Adam Paszke and PyTorch. *Reinforcement Q Learning*. `https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html`. Accessed: 2025-10-20. 2024.

[26] Elise van der Pol and Frans A. Oliehoek. *fransoliehoek*. `https://www.fransoliehoek.net/docs/VanDerPol16LICMAS.pdf`. Accessed: 2024-11-26.

[27] L. A. Prashanth and Shalabh Bhatnagar. "Reinforcement Learning with Function Approximation for Traffic Signal Control". In: *IEEE Transactions on Intelligent Transportation Systems* 12.2 (2011), pp. 412–421.

[28] S.S.S.M Qadri, M.A. Gökçe, and E Öner. "State-of-art review of traffic signal control methods: challenges and opportunities". In: *European Transport Research Review* 12 (2020).

[29] Haoran Su et al. "EMVLight: a Multi-agent Reinforcement Learning Framework for an Emergency Vehicle Decentralized Routing and Traffic Signal Control System". In: *Transportation Research Part C* 146 (2022).

[30] S. Tham and Chee-Khian Tham. "SensorGrid for Real-Time Traffic Management". In: *International Conference on Intelligent Sensors, Sensor Networks and Information*. Vol. 3. 2007, pp. 443–448.

[31] United States Environmental Protection Agency. *Greenhouse Gas Emissions from a Typical Passenger Vehicle*. `https://www.epa.gov/greenvehicles/greenhouse-gas-emissions-typical-passenger-vehicle`. Accessed: 2025-03-16. 2024.

[32] Pravin Varaiya. "Max pressure control of a network of signalized intersections". In: *Transportation Research Part C: Emerging Technologies* 36 (2013), pp. 177–195.

[33] Kentaro Wada et al. "An optimization modeling of coordinated traffic signal control based on the variational theory and its stochastic extension". In: *Transportation Research Procedia* 23 (2017), pp. 624–644.

[34] Webster. *Traffic Signal Settings*. 1958.

[35] Hua Wei et al. "A Survey on Traffic Signal Control Methods". In: *arXiv preprint arXiv:1904.08117* (2019).

[36] Hua Wei et al. "Colight: Learning network-level cooperation for traffic signal control". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2019, pp. 1913–1922.

[37] Hua Wei et al. "IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control". In: vol. 24. July 2018, pp. 2496–2505.

[38] Hua Wei et al. "PressLight: Learning Max Pressure Control to Coordinate Traffic Signals in Arterial Network". In: July 2019, pp. 1290–1298.

[39] M. Wiering et al. "Simulation and optimization of traffic in a city". In: *IEEE Intelligent Vehicles Symposium, 2004*. 2004, pp. 453–458.

[40] Mobin Yazdani et al. "Intelligent vehicle pedestrian light (IVPL): A deep reinforcement learning approach for traffic signal control". In: *Transportation Research Part C: Emerging Technologies* 149 (2023), p. 103991.

[41] Jiahui Yu and K. Sohn. *Normalization effects on deep neural networks*. arXiv. 2022.

[42] Guanjie Zheng et al. *Diagnosing Reinforcement Learning for Traffic Signal Control*. 2019.

[43] Guanjie Zheng et al. "Learning phase competition for traffic signal control". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2019, pp. 1963–1972.

[44] Jie Zhu, Wu Fengge, and Zhao Junsuo. *An Overview of the Action Space for Deep Reinforcement Learning*. 2022.