Sultan Kübra Can

A Master's Thesis

AGU 2022

TRAFFIC LIGHT MANAGEMENT USING REINFORCEMENT LEARNING METHODS

A THESIS
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF ABDULLAH GUL UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Sultan Kübra Can
July 2022

TRAFFIC LIGHT MANAGEMENT USING REINFORCEMENT LEARNING METHODS

A THESIS
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE OF
ABDULLAH GUL UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Sultan Kübra Can
July 2022

SCIENTIFIC ETHICS COMPLIANCE

I hereby declare that all information in this document has been obtained in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

Name-Surname: Sultan Kübra Can
Signature :

REGULATORY COMPLIANCE

M.Sc. thesis titled Traffic Light Management Using Reinforcement Learning Methods has been prepared in accordance with the Thesis Writing Guidelines of the Abdullah Gül University, Graduate School of Engineering & Science.

Prepared By	Co-Advisor	Advisor
Sultan Kübra Can	Asst. Prof. Dr. Mustafa Coşkun	Prof. Dr. Vehbi Çağrı Güngör

Head of the Electrical and Computer Engineering Program

Assoc. Prof. Dr. Kutay İçöz

ACCEPTANCE AND APPROVAL

M.Sc. thesis titled Traffic Light Management Using Reinforcement Learning Methods and prepared by Sultan Kübra Can has been accepted by the jury in the Electrical and Computer Engineering Graduate Program at Abdullah Gül University, Graduate School of Engineering & Science.

06/06/2022

(Thesis Defense Exam Date)

JURY:

Advisor : Prof. Dr. Vehbi Çağrı Güngör

Member : Dr. Burcu Güngör

Member : Dr. Fehim Köylü

APPROVAL:

The acceptance of this M.Sc. thesis has been approved by the decision of the Abdullah Gül University, Graduate School of Engineering & Science, Executive Board dated ….. /….. / ……….. and numbered …………..……. .

……….. /………... / ………..

(Date)

Graduate School Dean
Prof. Dr. İrfan ALAN

ABSTRACT

TRAFFIC LIGHT MANAGEMENT USING REINFORCEMENT LEARNING METHODS

Sultan Kübra Can
M.Sc. in Electrical and Computer Engineering
Advisor: Prof. Dr. Vehbi Çağrı Güngör
Co-advisor: Dr. Mustafa Coşkun

July 2022

Traffic lights have been around since 19th century, and aims to ease the chaos happening in intersections. It's recorded that, people spend hours in traffic leading degradations in human health and environment. Even though its main purpose is to reduce traffic congestion and decrease the number of accidents, most of the approaches cannot adapt very well to fast changing dynamics and growing demands of the intersections with modern world developments. Fixed-time approaches use predefined settings, and to maximize its success time slots are identified. Although there are successful attempts, they don't answer today's demands of traffic. To overcome this problem, adaptive controllers are developed, and detectors and sensors are added to systems to enable adoption and dynamism. Recently, reinforcement learning has shown its capability to learn the dynamics of complex environments such as urban traffic. Although it was studied in single junction systems, one of the problems was the lack of consistency with how the real world system works. Most of the systems assume the environment is fully observable or actions would be freely executed using simulators. This study aims to merge usefulness of reinforcement learning methods with real world constraints. The experiments conducted have shown that, with queue data obtained from sensors located at the beginning and at the end of the roads and limited action spaces it works very well and A2C is able to learn the dynamics of the environment while converging and stabilizes itself in a respectively short duration.

Keywords: Deep Reinforcement Learning, Urban Traffic Control

ÖZET

PEKİŞTİRMELİ ÖĞRENME YÖNTEMİ TABANLI TRAFİK IŞIK YÖNETİM SİSTEMLERİ

Sultan Kübra Can
Elektrik ve Bilgisayar Mühendisliği Anabilim Dalı Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Vehbi Çağrı Güngör
Eş-danışman: Dr. Mustafa Coşkun

Temmuz 2022

Trafik ışıkları, 19. yüzyıldan bu yana aktif olarak kavşaklardaki karmaşıklığı ve düzensizliği azaltmak amacı ile faaliyet gösteriyorlar. Kaynaklara göre, insanlar trafikte saatler geçiriyor, ki bu da hem insan sağlığı hem de çevre bakımından bozulmalara sebep oluyor. Trafik ışıklarının görevi trafik sıkışıklığını ve kaza sayısını azaltmak olsa da, şu an çalışan çoğu sistem modern zamanın gelişmeleri ile artan isteklere ve hızlı değişen kavşak dinamiklerine uyum sağlayamıyor. Bunlardan biri olan sabit zamanlı sistemler, önceden tanımlanmış ayarları kullanıyorlar ve performansını daha da artırmak için zaman dilimleri tanımlanıyor. Başarılı girişimler ve düzeltmeler görülse de, bugünün ihtiyaçlarına cevap veremiyorlar. Daha sonra, sistemlere sensörler ve detektörler eklenerek daha akıllı, dinamik ve adaptif sistemler geliştirildi. Son çalışmalar ise, pekiştirmeli öğrenmenin ve özellikle pekiştirmeli derin öğrenmenin kavşaklar gibi karmaşık ortamların dinamiklerini öğrenebildiğini gösterdi. Tekli kavşaklarda buna yönelik çalışmalar olmasına rağmen, gerçek dünya ile tam olarak tutarlı olmadığı, simülatörler vasıtasıyla tüm ortamın görünür ve karar verilen aksiyonların sınırsız olabileceğinin varsayıldığı fark edildi. Bu çalışma, pekiştirmeli öğrenme yöntemlerinin başarısı ve sağladığı fayda ile gerçek dünyanın sınırlarını birleştirmeyi hedeflemektedir. Bu çalışmada yapılmış olan deneyler gösteriyor ki, her bir yolun başına ve sonuna yerleştirilmiş olan sensörler vasıtası ile elde edilen kuyruk değerleri ve kısıtlı aksiyonlar kullanılarak geliştirilen pekiştirmeli öğrenme yöntemleri iyi bir performans sergiliyor ve özellikle A2C yöntemi çevrenin dinamiklerini öğrenerek nispeten kısa sürede yakınsıyor ve stabil hale geliyor.

Anahtar kelimeler: Pekiştirmeli Öğrenme, Şehiriçi Trafik Yönetimi

Acknowledgements

First and foremost, all praises and thanks to Allah for His blessings.
I would like to express my sincere gratitude to my primary supervisor, Prof. Dr. Vehbi Çağrı Güngör, who guided and supported me throughout my time as a bachelor's and a master's student. I would also like to thank my academic co-advisor Dr. Mustafa Coşkun for giving me opportunity to do research in this field and providing guidance throughout. The completion of this study could not have been possible without the expertise of them. I am extremely grateful to my beloved parents and family for their love, prayers and encouragements. I am very much thankful to my husband for his continuing support.
I also want to thank Adam Thahir, a fellow master's student and colleague that I've worked closely for providing me support and in-depth guidance and my friends Hatice Kübra Öcal and Habibe Sultan Özhan.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
SuMO	Simulation of Urban MObility
TraCI	Traffic Control Interface
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
A2C	Advantage Actor-Critic
TRPO	Trust Region Policy Optimization
PPO	Proximal Policy Optimization
SMC	Secure Multiparty Computation

To my beloved parents,
Ali Kılıç and Şadiye Kılıç

Chapter 1

Introduction

Traffic lights have been used in many cities since 19th century with gas-lit traffic lights in London, which was soon followed by a single electric light in Ohio in 1914 and a network of traffic lights with a manual switch in Utah in 1917 [1]. The purpose of traffic lights is to control traffic to prevent traffic jams and congestion in the intersections and to decrease the number of accidents caused by uncontrolled intersections.

Traffic congestion and jams are practical problems in today's world causing delays, increased costs because of fuel and increased amount of harmful gas emission. As stated in Texas A&M's 2021 Urban Mobility Report, the amount of fuel consumption caused by traffic congestion is about 1.7 billion gallons with an annual delay of 4.3 billion hours which led to a cost of 100 billion dollars [2].

In a study by Kahneman [31] shows that commuting in traffic is one of the least enjoyable activities we do. As stated in [2], an average American spends 42 hours annually in the traffic and [32] states that traffic is the main concern of the people, exceeding personal safety or finances. Aside from the delay and cost it causes, traffic congestion leads to health problems, especially respiratory problems [33] related to long-term exposure to harmful gas emissions.

As the development of cities continue, we see more vehicles on the road. In a report from Turkish Statistical Institute, it's stated that the number of vehicles registered in March 2022 increased by 47.2% compared with the previous month [34]. Therefore, the traffic problem seems to remain and would be more problematic if not solved. With the development of modern cities, the traffic lights are automated and being optimized to increase the vehicle flow of the intersection using variety of methods such as fixed-time approaches, adaptive approaches or more intelligent deep learning approaches as a solution.

1.1 Offline Approaches

Offline or fixed-time traffic light control methods use predefined traffic light settings (the sequence of green, yellow and red phase durations) that are deduced and optimized based on offline historical traffic data analyzation. To maximize its success and optimize the predefined timings, time slots has to be identified. The actuated methods contain a set of defined rules where any of them could be triggered by violations or extreme situations to adapt the traffic environment more. However, even though these kind of methods work well in constant environments, they cannot adapt to the dynamism of the traffic environment and demands.

1.2 Adaptive Controllers

Adaptive controllers have been utilizing the use of sensors and data collection to adapt the changes in the traffic. An example to these methods is Sydney Coordinated Adaptive Traffic System (SCATS) **[3]** where real time data is used to control the traffic lights. However, it is deemed to be extremely costly and requires dramatic amount of modifications to existing road systems. Another notable early traffic system to be used in UK is Split Cycle and Offset Optimization Technique (SCOOT) **[4]** where traffic lights are controlled only by small adjustments to avoid dramatic changes and focus more on long-term flow of the intersection using real time data by detectors. One of the more recent approaches is Intelligent Traffic Light Controlling algorithm (ITLC) **[5]**. It uses vehicular ad-hoc networks (VANETs) **[6]** as the backbone, which requires all the vehicles in network to be equipped with a form of GPS to identify its properties, and aims to decrease the vehicle wait times.

1.3 Reinforcement Learning Based Approaches

Machine learning, especially reinforcement learning, methods are used to make the agents or algorithm learn what the dynamics are and act based on those insights. As an example to reinforcement learning, Wiering has proposed different variations of reinforcement learning methods to be applied to traffic environment and created the Green Light District (GLD) simulator with the purpose of demonstration **[7]** which was used in other works **[8]** for further development. Recently, reinforcement learning gained much attention as Deep

Q-Networks (DQN) has shown robust performance on Atari games **[9]**. One of the similar approaches called Deep Deterministic Policy Gradient (DDPG) **[10]** has shown success on large continuous action and state spaces.

The methods to be applied at real world traffic environments have to have a robust performance under a wide variety of conditions and has to react fast when a dramatic change happens. As deep reinforcement learning has shown its success, in our approach, we are applying several on-policy reinforcement learning algorithms with real world constraints to achieve robust performance with an ability to adapt safely to real world intersections as the main contribution of this work.

Chapter 2

Reinforcement Learning

Reinforcement learning is a subfield of machine learning that learns the solution by trial and error [11]. The main elements in these methods are the *agent* and the *environment*. We define the environment as the world that the agent sees and interacts with. The *agent*, is not given what *actions* ($a \in \mathcal{A}$) to take under specific situations, but rather, it must discover the best action to take by interacting with the given partially observable environment *states* ($s \in \mathcal{S}$) and apply the action to the environment. By defining a *reward* ($r \in \mathcal{R}$) function as a signal from the environment, the actions of the agent are evaluated that will make the agent to take better actions by learning the optimum *policy* ($\pi(a|s)$) to maximize its cumulative reward, or *return*. Agent learns which actions and states produce higher rewards through *exploration* and performs actions accordingly through *exploitation.*

Each state that the agent observes is associated with a value ($V(s)$), or state-action pair, calculated by a value function to predict the expected rewards under the taken policy, which is used to evaluate how good a given state is. Here, the main goal of the reinforcement learning is to learn policy and value function.

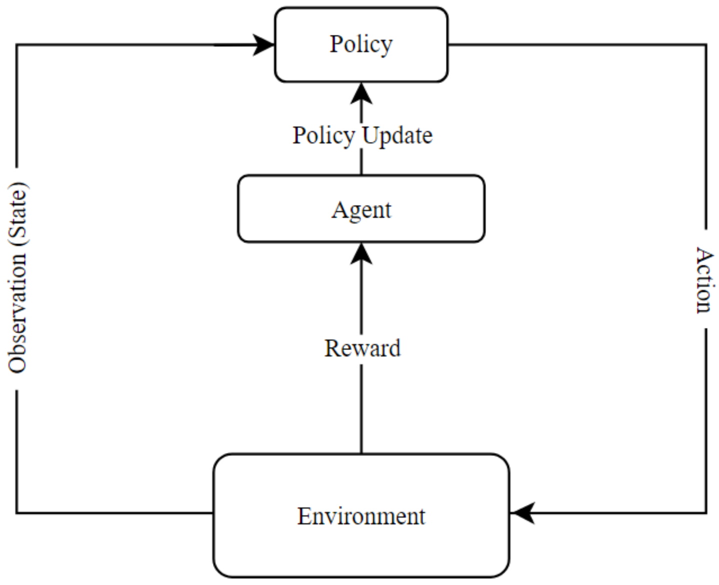


Figure 1. Reinforcement Learning Diagram

The interaction between the environment and the agent is a sequence of states (observations from the environment), actions and rewards in time ($t = 1,2, \dots, T$). During this process of learning, the agent accumulates knowledge of environment, learns an optimal policy by taking the best actions until the process is terminated. The sequence of states and actions is described by a trajectory (τ) and could be represented as below:

$$\tau = (s_0, a_0, s_1, a_1, \dots)$$

where the very first state s_0 is randomly sampled and state transitions at $t + 1$ depend on only the most recent action a_t. Actions come from the agent according to its determined policy.

Reward function is a critical element for an agent to learn the environment, it depends on the current state, the latest action taken, and the next state obtained by executing the latest action:

$$r_t = R(s_t, a_t, s_{t+1})$$

Even though there are variety of approaches to solve RL problems, *policy optimization* methods are used in this work due to proven effectiveness of policy optimization methods on continuous control problems. These methods require actions to be drawn from a probability distribution that is generated by a policy $a_t \sim \pi_\theta(a_t|s_t)$ where θ represents the parameters. They optimize parameters θ either by gradient ascent on the objective given in (1), like A2C, or maximizing local approximations of the objective, like how PPO behaves. This optimization is nearly always performed on-policy, meaning that each update uses data collected while executing the current version of the policy.

Policy gradient methods are a subclass of policy optimization and they aim to learn the policy using a parameterized function respect to θ, $\pi_\theta(a|s)$. The optimum value of θ could be found by gradient ascent to produce the highest expected reward by maximizing the score function given in (1) where $d^\pi(s)$ is state distribution, $Q^\pi(s,a)$ is state-action value and $\pi_\theta(a|s)Q^\pi(s,a)$ is action distribution.

$$J(\theta) = \sum_{s\in\mathcal{S}} d^\pi(s)V^\pi(s) = \sum_{s\in\mathcal{S}} d^{\pi\theta}(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s)Q^{\pi\theta}(s,a) \tag{2.1}$$

2.1 Advantage Actor-Critic (A2C)

There are two main components in policy gradient based methods: policy model and value function. In vanilla policy gradient methods, the value function is not optimized although it would be very useful as the value function could assist the policy update. However, Actor-Critic methods takes optimization of value function into consideration and consists of two models:

- **Critic** model updates and optimizes value function parameters $\boldsymbol{w}$.
- **Actor** model is used to update the policy parameters $\boldsymbol{\theta}$, using critic model evaluations.

A2C [12] is a policy gradient algorithm designed to be used specifically on parallel training. Multiple critic models learn the value function by multiple actor models as they're being trained in parallel and both global parameters $\boldsymbol{\theta}$ and $\boldsymbol{w}$ are updated asynchronously using local gradients of the parameters.

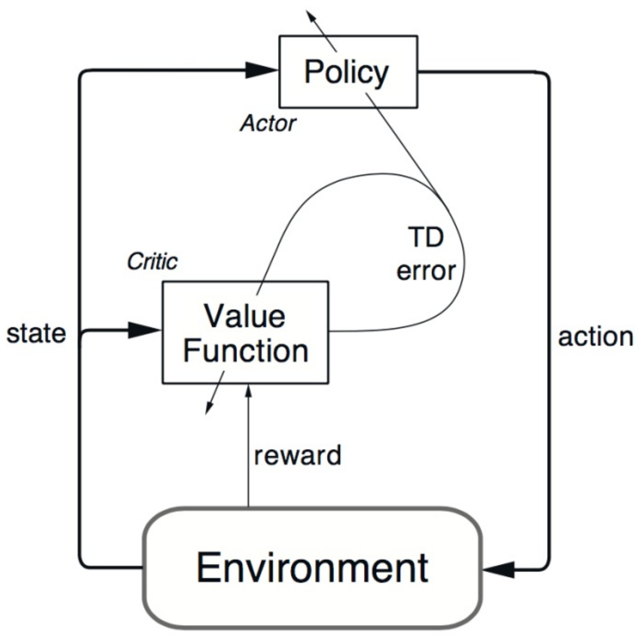


Figure 2. Actor Critic Model Representation

2.2 Trust Region Policy Optimization (TRPO)

To increase training stability, changing policy too much by parameter updates at one step should be avoided. TRPO [13] takes this idea into account by enforcing a constraint called KL divergence (trust region constraint) to limit policy updates at each iteration. It still aims to maximize the objective function while labeling behavior policy as $\pi_{\theta_{old}}(a \mid s)$ (2.2), however, by applying trust region constraint, it enforces the distance between new and old policies to be small enough, within a parameter δ and the improvement would be monotonic:

$$J(\theta) = \mathbb{E}_{s\sim p^{\pi_{\theta_{old}}}, a\sim\pi_{\theta_{old}}}\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{old}}(a \mid s)}\hat{A}_{\theta_{old}}(s,a)\right)$$

$$\mathbb{E}_{s\sim p^{\pi_{\theta_{old}}}}[D_{KL}(\pi_{\theta_{old}}(.\mid s) \mid\mid \pi_\theta(.\mid s))] \leq \delta \tag{2.2}$$

2.3 Proximal Policy Optimization (PPO)

PPO adopts the same idea of TRPO by implementing a similar constraint. PPO [14] simplifies the idea by using a clipped surrogate objective by forcing ratio between new and old policies denoted as $r(\theta)$, to stay within a relatively small interval at $[1-\epsilon, 1+$

$\epsilon]$ given ϵ is a hyperparameter. PPO has been tested and it produced good results with much greater simplicity.

$$r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} \tag{2.3}$$

Additionally, to encourage enough exploration, the objective function is augmented with an error term on the value function $(c_1(V_\theta(s) - V_{\text{target}})^2)$ and an entropy term $(c_2H(s, \pi_\theta(.)))$ where objective function becomes:

$$J^{\text{CLIP'}}(\theta) = \mathbb{E}[J^{\text{CLIP}}(\theta) - c_1(V_\theta(s) - V_{\text{target}})^2 + c_2H(s, \pi_\theta(.))] \tag{2.4}$$

where c_1 and c_2 are hyperparameter constraints and $H(s, \pi_\theta(.))$ is entropy function.

Chapter 3

Methodology

The methodology we have followed could be categorized into five categories:

1. Traffic Simulation
2. State Space Representation
3. Action Space Definition
4. Reward Function Design
5. Vehicle Generation Method

3.1 Traffic Simulation

We use simulator packages to create the network, simulate and interact with the environment to able to produce valuable results. Traffic simulations in microscale are designed and executed using a microscopic traffic simulation package, SUMO (Simulation of Urban MObility) [15], which allows each vehicle to be designed and tracked individually through the network. In our network, each road is represented with *edges* and the *lanes* in each road are displayed individually. Each *phase* is connected to lanes, and *connections* represents the ways that vehicles can pass. *Routes* are the defined to depict which routes a vehicle can take according to the lane change algorithm. The connections and phase configurations are shown in section 4.1.

At intersections, phases are switched on and off, where switching on means turning all the lights included in related phase to green and switching off means turning all the lights to first yellow and then red. Each phase has a green time assigned to it and after green times are executed for all the phases, it’s called a cycle [16]. The codes for the behavior of intersection and phases can be found at appendix section. Intersection and phase representation is shown in figure 3 and total cycle time is calculated as below where tp is the phase duration and tt is transition time:

$$t_{cycle} = tp_i + tt_i + tp_{i+1} + tt_{i+1} + \cdots$$

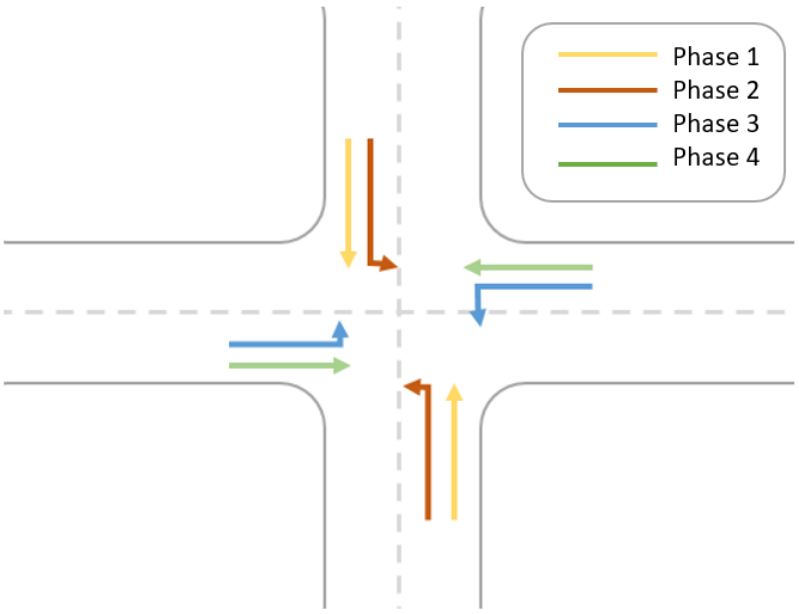


Figure 3. Intersection and phase representation

OpenAI Gym [17] is a toolkit to enable development of reinforcement learning algorithms. The traffic environment is designed and implemented using Gym toolkit, which has defined properties and functions to modify according to each specification. The step function is called each step of the simulation with the selected action, and is used to return four values:

- **Observation** is the representation of the environment state.
- **Reward** is the amount of reward given by the previous action of the agent.
- **Done** is a true/false value indicating whether or not the episode ended. If the episode ended or quitted, the environment is reset by reset function.
- **Info** is a dictionary to be used for debugging.

The agent then chooses an action based on the observation and reward returned by step function as defined in chapter 2.

3.2 State Representation

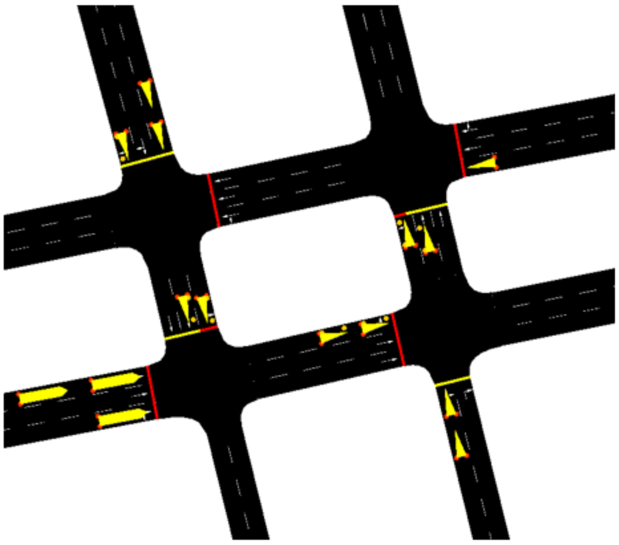
As we are using a traffic simulator for evaluation purposes, the traffic states at each step in the environments are fully observable to us. However, in order to adapt our system to the real world, how we obtain the state must be available in a typical real world traffic setup. We have seen that, the most noticeable datasets are obtained by traffic detectors [18] and one of the most common detectors to obtain real time information of the traffic are induction loops that are placed under the pavement [19]. Therefore, we assume that there are loop sensors at the beginning and the end of each incoming lane and constrain the state space with the vehicle count and queue length inside the lane.

There are two principles adapted to choose a state:

a) the agent has the information it needs to make a good decision and

b) there is no unneeded information as unneeded information can lead to extra training time and slower learning by increasing the computation cost due to larger state spaces.

Therefore, the current state of the simulation environment is represented as a nxm matrix where n is the edge count of the specific intersection and m is the maximum lane count, similar to the concept used in [20]. While counting the vehicles, the whole capacity of lanes is taken into account and the vehicles waiting inside the intersections are ignored.



$$\begin{bmatrix} 1 & 0 & 2 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \\ 0 & 2 & 0 \end{bmatrix}$$

Figure 4. State Space Representation of an Intersection

3.3 Action Space Definition

Some approaches use single real value and use that value to set the duration of the next phase [21] and some of the other approaches use action to decide which phase to make a transition into [22]. However, these kinds of approaches could lead to chaotic situations as the agent doesn't fully utilize the intersection dynamics as each phase should be set depending on all the phases in the intersection and doesn't take cycle demands into account. In order to avoid this, we set the phase (green time) durations off all phases at the beginning of each cycle. This way, if a network has k phases, the action vector would have k components where each element of the action vector being a positive integer number for each phase duration.

The transition (yellow) time T_t of a traffic light is the time where the traffic lights turn to yellow when going from green to red or red to green to enable vehicles to slow down before a red light. As SUMO implements a car-following and mostly collision-free model, it's difficult for the agent to optimize transition times. Therefore, transition times are not learnt and set with a default value of ten seconds.

The agents in reinforcement learning typically use either a discrete (or multi discrete) or continuous (or Box) action space [11]. Using continuous values, actions are expressed as real values in the given range, but discrete action space allows the agent to take a distinct action to perform from a finite action set [23]. Using discrete action set allows one to simplify the model both conceptually and computationally as the action is limited to a given number instead of being infinite like continuous action space utilizes. Therefore, we have defined a discrete action space as given:

$$A_d = \{a_1, a_2, \dots, a_k\}$$

where a is the phase duration for each phase in the range of $[0, a_{max} - a_{min}]$, a_{min} is the minimum phase time and a_{max} is the maximum phase time. a_{min} is set with a default value of ten seconds. Default values are chosen according to the typical transition time and typical minimum phase time executed in the city of Kayseri. This definition allows us to utilize real world problems better as in the real world, as we have green time limits that each phase can execute.

3.4 Reward Function Design

To define the solution in the best way, the traffic has to be analyzed in a good way. To achieve this, two measurements are analyzed in this approach:

- Queue detection: Queue is defined as “A line of vehicles, bicycles, or persons waiting to be served by the system in which the flow rate from the front of the queue determines the average speed within the queue.” [24] To reduce and eliminate queueing, no lane should exceed a number of vehicles on it calculated based on the following formula:

$$q\_max_i = \begin{cases} lane\_capacity_i * 0.6, & lane\_length_i > 100 \\ lane\_capacity_c * 0.8, & lane\_length_i \leq 100 \end{cases} \quad (3.1)$$

  If q_i exceeds q_max_i, then the episode is ended with a punishment by a value R_q where q_i is the current queue length at lane i.

- Congestion measurement: Traffic congestion is defined as a condition that occurs when the use of the intersection increases and causes slower speeds for vehicles and longer trip times which leads to queues eventually [25]. Congestion is detected by first checking if there is enough queue in each lane which is denoted as $lane\_base_i$ and whether or not the vehicles are passing through the intersection easily by checking if there are less vehicles passed through the intersection in the cycle than the average count of vehicles passed at previous k cycles with the following formula:

$$lane\_base_i = lane\_capacity_i * 0.3 \quad (3.2)$$

$$passed_{current} = total_{current} - total_{previous} \quad (3.3)$$

$$congestion = \begin{cases} true, & q_i > lane\_base_i \\ & and\ passed_{current} < passed_{avg} \\ false, & otherwise \end{cases} \quad (3.4)$$

  If a congestion is detected, then the episode is ended with a punishment by a value R_c.

3.5 Vehicle Generation Method

Interaction with the simulation is enabled by TraCI (Traffic Control Interface) [26]. TraCI gives access to the traffic simulation and allows to retrieve and use values of the network and all the vehicles in the network. The vehicle generation process takes place at the beginning of the simulation. Episodes are allowed to run until they finish regardless of punishment or until 10,000 time steps have passed. After each episode is ended, the present values are checked to determine whether or not a level, a vehicle trip setting generated, is solved. After a level is solved, new vehicles are generated, meaning a new level is created. Otherwise, the episode starts with the current level again. The simulation process is shown in figure 5.

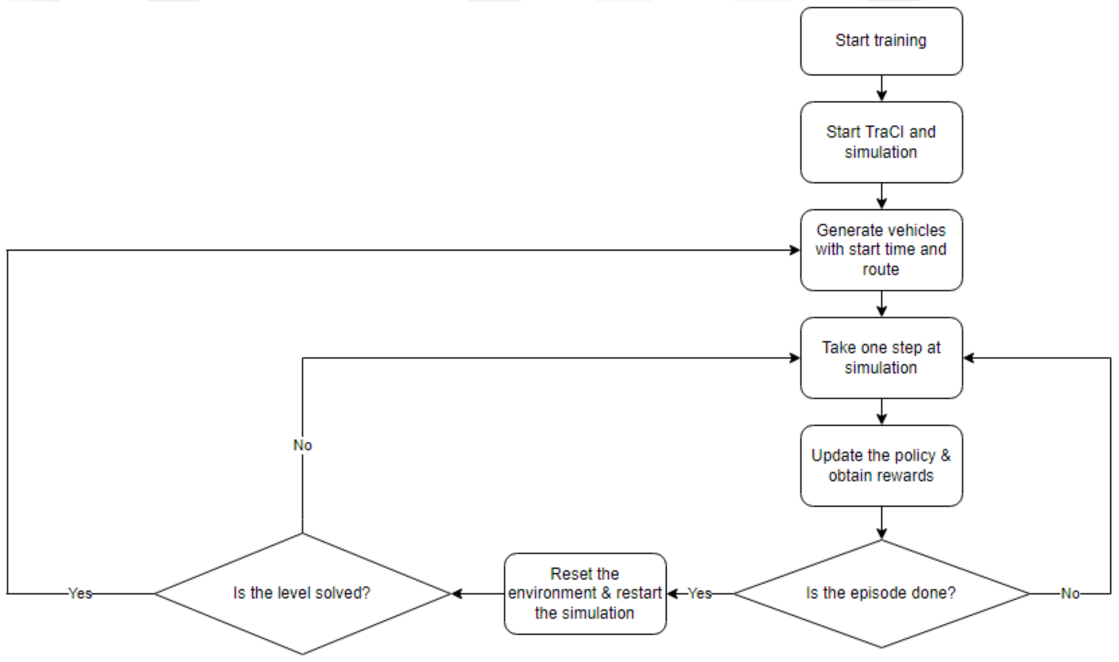

Start training
Start TraCI and simulation
Generate vehicles with start time and route
Take one step at simulation
Update the policy & obtain rewards
Is the episode done?
No
Yes
Reset the environment & restart the simulation
Is the level solved?
No
Yes

Figure 5. The simulation process

Vehicles are generated randomly according to the real life trends at chosen intersections and each configuration is called a **level**. The intersection flow information are obtained using monitoring techniques and then analyzed. Then each vehicle is generated with a departure time, a departure position and a route using that flow information enhanced by randomness.

Figures 5 and 6 shows the daily vehicle trends at Intersection A and Intersection B. Each line in the plots shows the load of each edge throughout the time from morning to evening. We can deduce that, the load of edge A in Intersection A increases over time while the loads in the other edges remain moderately same. On the other hand, two edges in Intersection B show a notable increase while Edge G and A shows a moderate increase.

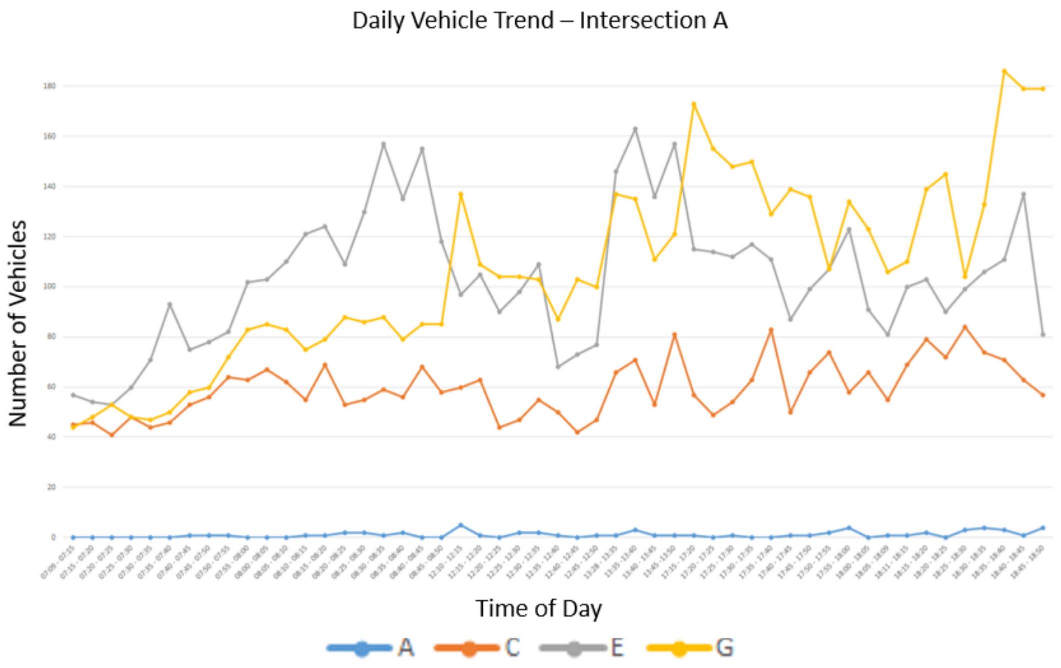


Figure 6. Daily Vehicle Trend at Intersection A

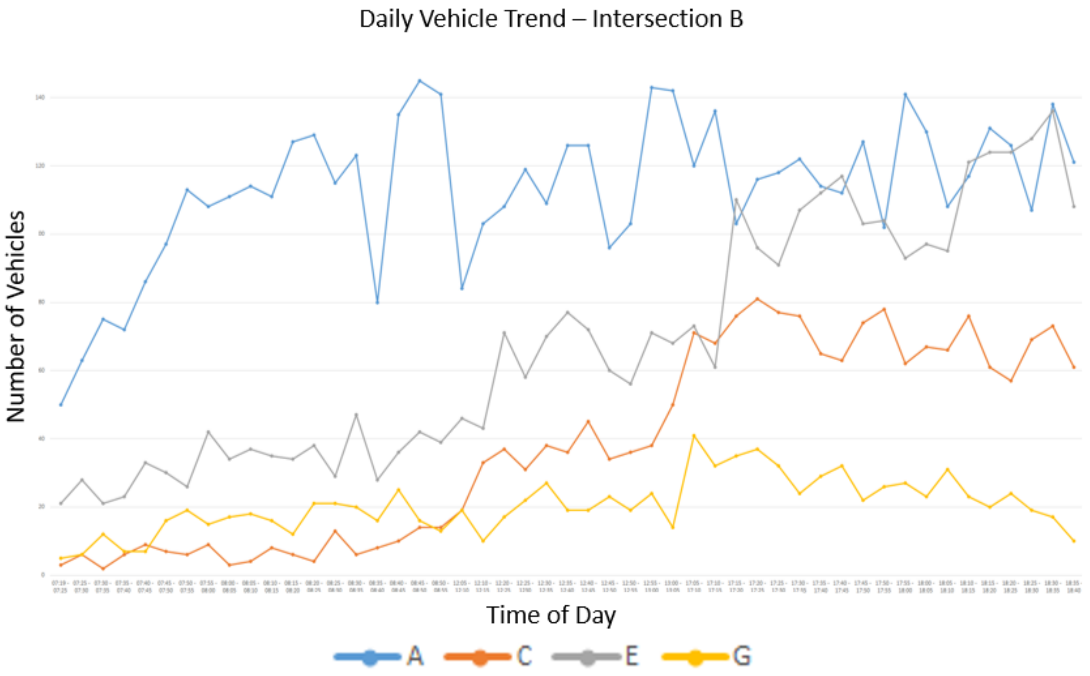


Figure 7. Daily Vehicle Trend at Intersection A

Chapter 4

Results

4.1 Experiment Design

In order to evaluate how well the algorithms work, we have used two different intersections with similar features of road and phase number selected in the city of Kayseri. The real images of the intersections are obtained from Yandex Maps [27], and the network is drawn using NetEdit [28], which is a tool provided by SUMO package that is used to create and modify traffic networks. All the phases in the intersections are obtained by monitoring the intersections and the connections are drawn to indicate which paths a vehicle can use and are shown as blue and brown lines in figures 8 and 10. The real length of the roads are obtained using OpenStreetMap [29] API.

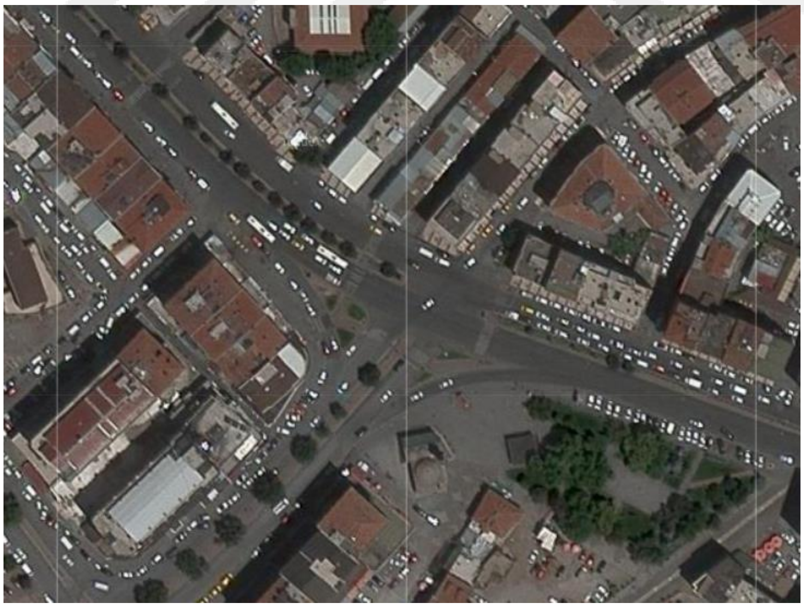


Figure 8. Real World View of Intersection A

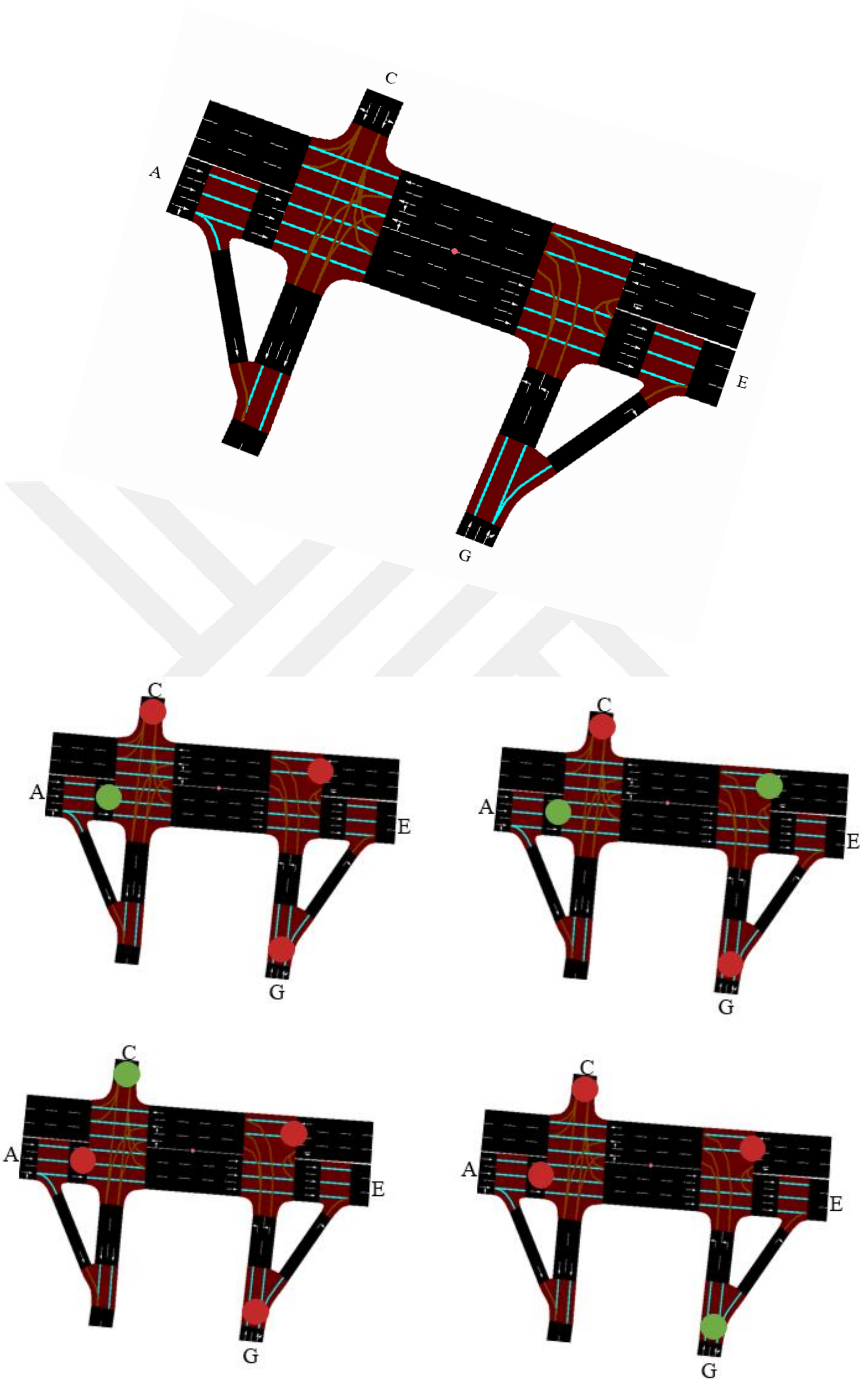


Figure 9. Network Drawing and Phase Configurations of Intersection A

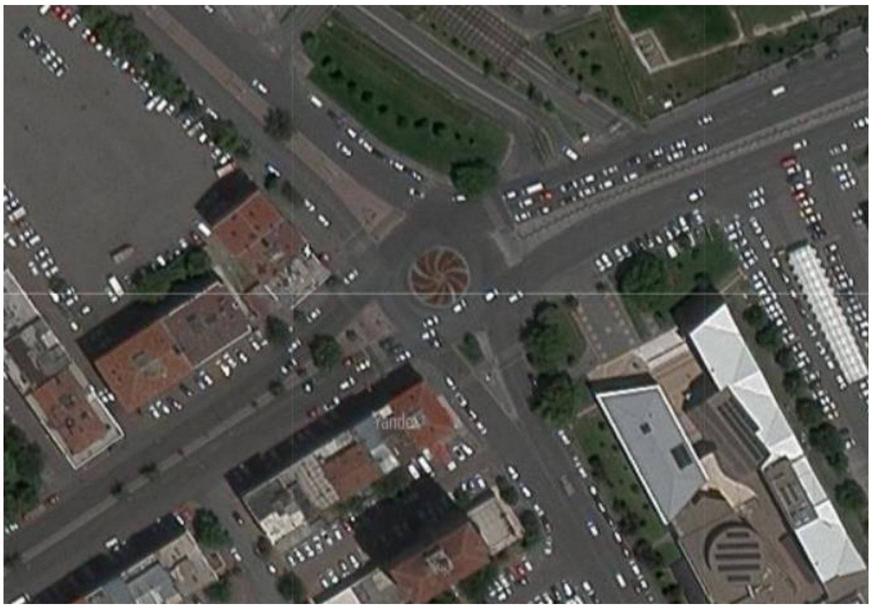


Figure 10. Real World View of Intersection B

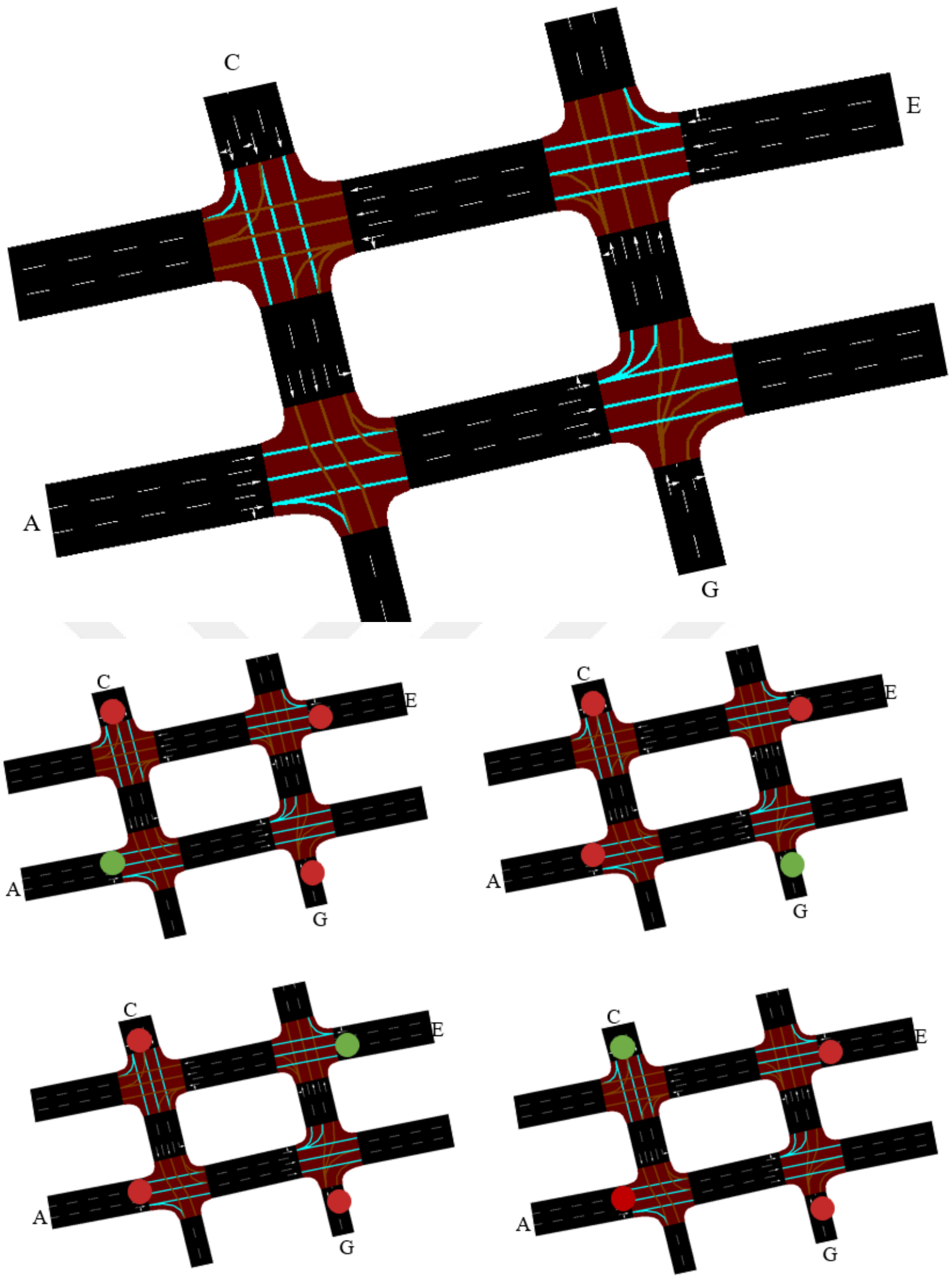


Figure 11. Network Drawing and Phase Configurations of Intersection B

The algorithms explained in chapter 2 are implemented using Stable Baselines [30] from OpenAI to both junctions and adaptive parameter noise is added for further exploration. As we are using parameter noise, we used MLP for both the policy and the value function. The SuMO simulation is stochastic by its nature as a microscopic traffic simulator. Therefore, the results obtained at each level depend on the random seed set. In order to avoid algorithm overfitting to a single level, we randomize the seed and change the network at each level as number of roads, maximum number of lanes and phases are same for both networks.

4.2 Results

To be able to evaluate the performances of the algorithms, we compare the total reward gained and discounted rewards at one episode although our main reference measure will be the total reward. Each simulation is run for 200000 time steps due to computation costs. Runtime of the algorithms are compared in figure 12, and even though there is no dramatic difference, PPO is the most performant algorithm while TRPO falls behind by ~10.5 hours each run.

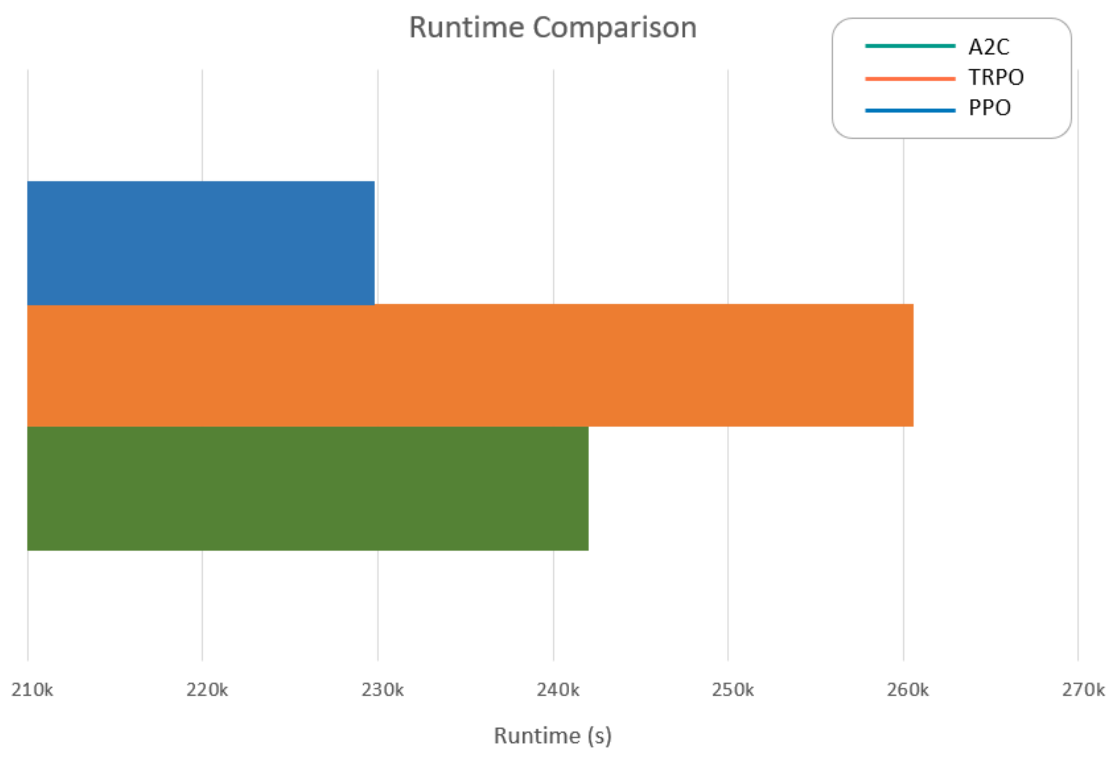


Figure 12. Total reward with smoothing gained by step for each algorithm

In figure 13, we can find the performance comparison of the algorithms by comparing the total reward gained by episode. Although TRPO and PPO reaches a similar level, A2C outperforms them in a notably short period. A2C was able to eliminate queueing and congestion successfully. Additionally, it can be said that A2C is much stable by figure 13 where reward plots are shown without smoothing. We can see that there are steep decreases in the plots given in figure 14, which means the algorithm is punished for either queueing or creating a traffic congestion. And even though TRPO couldn't converge within the specified time range, it shows a promising trend in reward accumulation.

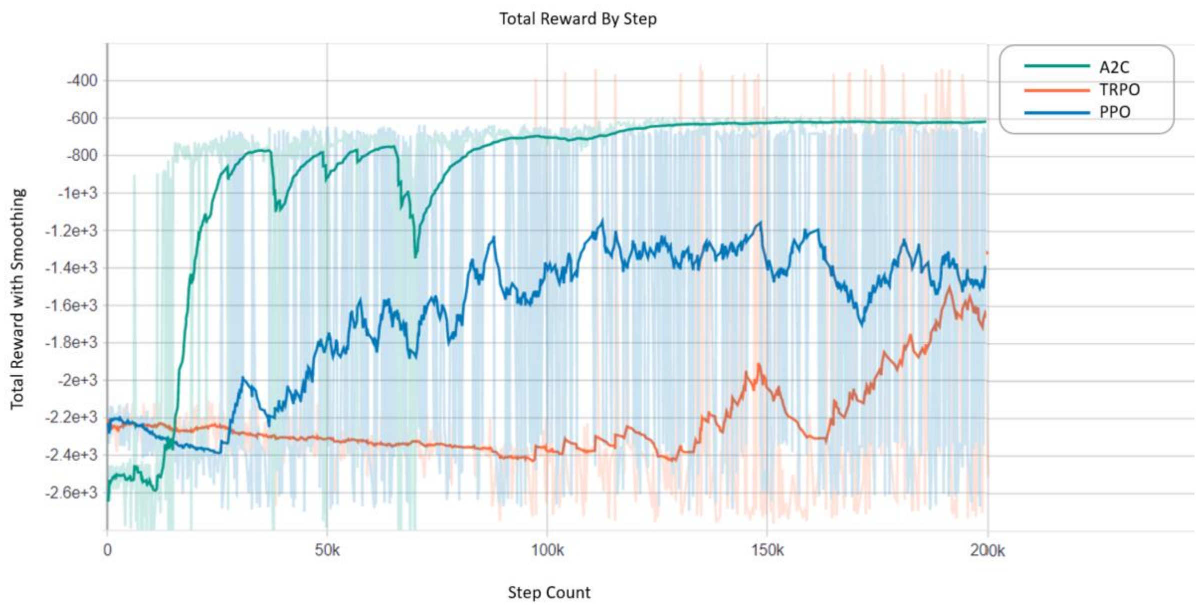


Figure 13. Total reward with smoothing gained by step for each algorithm

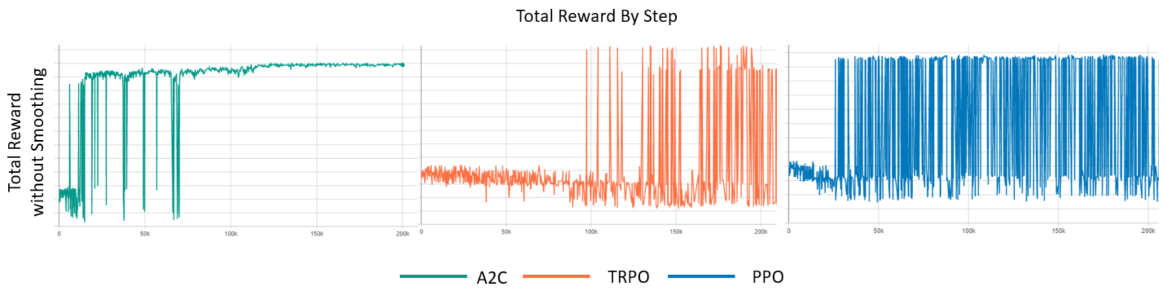


Figure 14. Total reward without smoothing

As the purpose of this thesis is to decrease the queue length inside intersections and decrease the waiting times, we are monitoring the queue lengths during training of the agents. We can see how queue length decreases over time before it stabilizes using A2C algorithm in figure 15. By figure 15, we deduce that the optimized queue length is highly dependent on the q_max_i value mentioned in chapter 3.4. Therefore, we could say that the q_max_i value itself needs to be optimized and R_c value should depend on the q_i instead of being a default, fixed value.

As TRPO and PPO algorithms were not able to converge and stabilize, the results are not presented.

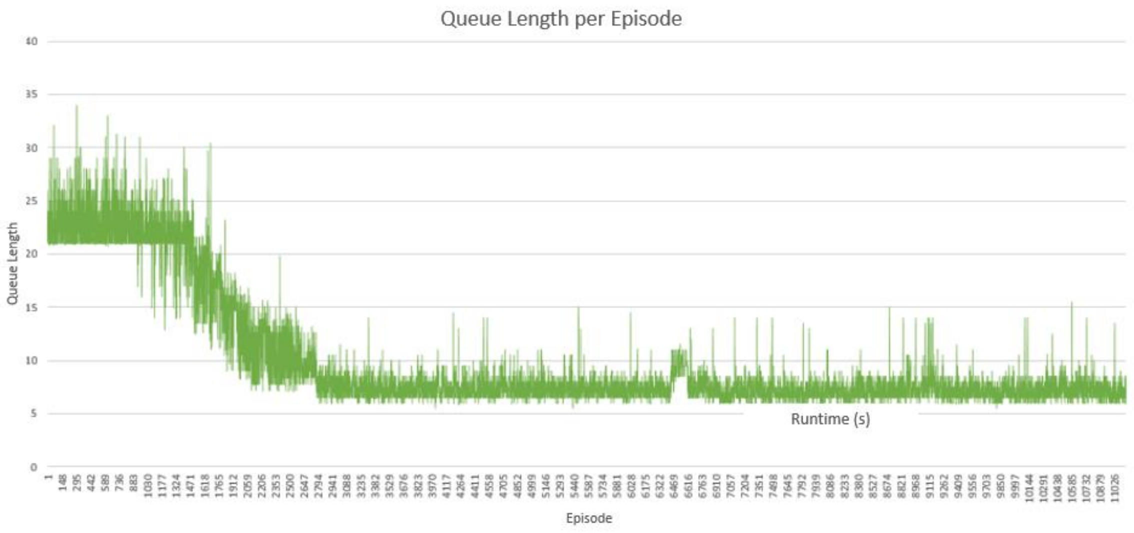


Figure 15. Queue Length per Episode

In figure 16, discounted reward plot can be found. We can see that while discounted reward for TRPO gradually increases, PPO shows an increase until halfway and then converges with no further increase. Even though A2C performed best in terms of total reward, we see a decrease in discounted reward.

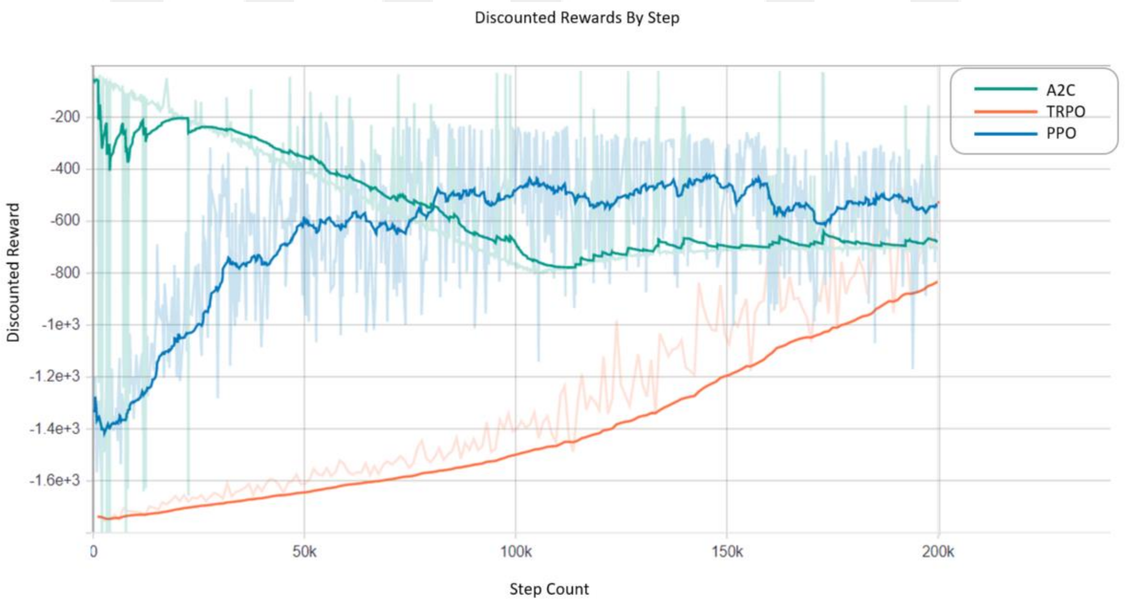


Figure 16. Calculated discounted reward by step for each algorithm

Chapter 5

Conclusions and Future Prospects

5.1 Conclusions

We studied the effectiveness of reinforcement learning to real world problems, specifically traffic environments and have successfully applied three of the on-policy reinforcement learning algorithms to the defined networks. We have obtained good results for both networks using A2C, meaning that reinforcement learning is able to converge very well while staying stable under changing dynamics. Even though the other methods couldn’t show a notable improvement, we can see that reward accumulation by TRPO shows a promising trend, which means there could be a room for improvement with hyper parameter tuning and longer training periods.

5.2 Societal Impact and Contribution to Global Sustainability

As described in chapter 1, traffic congestion has been a problem that affects society and environment from multiple ways. Besides causing environmental damage, people suffer from health problems, especially respiratory issues, fuel costs and loss of time. The approach this thesis provides aims to address traffic issues by increasing the flow of the vehicles by optimizing the phase and cycle times inside intersections and thus decreasing the time vehicles wait and queue lengths at the intersections. This thesis provides a way to achieve this purpose by observing the traffic patterns and training a reinforcement learning agent with randomly generated samples with real world constraints.

Additionally, this thesis requires only the loop sensors at the entry and exit points of the roads in the intersections to obtain queue length. Therefore, there is no need to make

dramatic changes to the roads which leads to both monetary and energy costs as deployment of loop sensors are relatively low-cost and does require minor adjustments.
As described in chapter 4, we have seen improvements and valuable results from our experiments. Our model is able to converge and stabilize in a relatively short time and we can see the queue lengths are decreasing and thus contributes to economic growth and sustainable cities and communities achieved by optimally working intersections.

5.3 Future Prospects

The proposed environment design and chosen algorithms work well in the provided environments and provided vehicle generation trends. However, using this approach in real time requires generalization of trends and networks, moreover synchronization between intersections. To achieve this, further studies on curriculum learning [35], [36] and multi-agent [37], [20] methods is needed.

BIBLIOGRAPHY

[1] R. Ross, "Invention of the Traffic Light," Live Science, 2016. [Online]. Available: https://www.livescience.com/57231-who-invented-the-traffic-light.html.

[2] T. Lomax, D. Schrank and B. Eisele, "2021 Urban Mobility Report," 2021. [Online]. Available: https://mobility.tamu.edu.

[3] New South Wales Government, "SCATS: Sydney Coordinated Adaptive Traffic System," 2011. [Online]. Available: https://www.qtcts.com.au/media/512152-RTA532_SCATS_A4_Product_Brochure_07.pdf..

[4] R. D. Bretherton, "Scoot Urban Traffic Control System—Philosophy and Evaluation," *IFAC Proceedings Volumes,* vol. 2, no. 23, pp. 237-239, 1990.

[5] M. B. Younes and A. Boukerche, "An Intelligent Traffic Light scheduling algorithm," in *39th Annual IEEE Conference on Local Computer Networks Workshops,* 2014.

[6] V. Hamakumar ve H. Nazini, «Optimized traffic signal control system at traffic intersections using VANET,» %1 içinde *Chennai Fourth International Conference on Sustainable Energy and Intelligent*, 2013.

[7] M. Wiering, J. Vreeken, J. v. Veenen and A. Koopman, "Simulation and Optimization of Traffic in a City," in *Intelligent Vehicles Symposium, 2004 IEEE,* 2014.

[8] L. Prashanth and S. Bhatnagar, "Reinforcement Learning With Function Approximation for Traffic Signal Control," in *IEEE Transactions on Intelligent Transportation Systems*, 2011.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra ve M. Riedmiller, «Playing Atari with Deep Reinforcement Learning,» *CoRR,* 2013.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, "Continuous control with deep reinforcement learning," in *ICLR* , 2016.

[11] R. S. Sutton and A. G. Barto, "1.1 Reinforcement Learning," in *Reinforcement Learning: An Introduction*, The MIT Press, 2017.

[12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *Proceedings of Machine Learning Research,* no. 48, pp. 1928-1937, 2016.

[13] J. Schulman, S. Levine, P. Abbeel, M. Jordan and P. Moritz , "Trust Region Policy Optimization," *Proceedings of Machine Learning Research,* no. 37, pp. 1889-1897, 2015.

[14] J. Schulman, F. Wolski, P. Dhariwal and A. Radford, "Proximal Policy Optimization Algorithms," *OpenAI Researches,* 2017.

[15] Eclipse Foundation, "Simulation of Urban MObility," Eclipse Foundation, [Online]. Available: https://www.eclipse.org/sumo/.

[16] F. H. A. (FHWA), "Traffic Signal Design," in *Signal Timing Manual - Second Edition*, Federal Highway Administration (FHWA), 2008.

[17] OpenAI, "Gym," OpenAI, [Online]. Available: https://gym.openai.com/.

[18] Papers With Code, "PeMSD7 - Traffic Dataset," Papers With Code, 2012. [Online]. Available: https://paperswithcode.com/dataset/pemsd7.

[19] A. Spears, "Sensors At Traffic Lights," Eltec Corp., 2019. [Online]. Available: https://elteccorp.com/news/other/are-there-sensors-at-traffic-lights/.

[20] J. A. Calvo and I. Dusparic, "Heterogeneous Multi-Agent Deep Reinforcement Learning for Traffic Lights Control," in *AICS 2018*, 2018.

[21] H. Joo and Y. Lim, "Traffic Signal Time Optimization Based on Deep Q-Network," *Applied Sciences*, 2021.

[22] M. Coşkun, A. Baggag and S. Chawla, "Deep Reinforcement Learning for Traffic Light Optimization," in *IEEE International Conference on Data Mining Workshops (ICDMW)*, 2018.

[23] W. Masson, P. Ranchod and G. Konidaris, "Reinforcement Learning with Parameterized Actions," in *AAAI Conference on Artificial Intelligence*, 2016.

[24] Federal Highway Administration (FHWA), "Definition, Interpretation, and Calculation of Traffic Analysis Tools Measures of Effectiveness," in *Traffic Analysis Tools*, U.S. Department of Transportation, 2021.

[25] Ministry of Transport, New Zealand, "The Congestion Question, Could road pricing improve Auckland's traffic?," Auckland Council, 2019.

[26] German Aerospace Center (DLR), "TraCI," German Aerospace Center (DLR), [Online]. Available: https://sumo.dlr.de/docs/TraCI.html.

[27] Yandex, "Yandex Maps," Yandex, [Online]. Available: https://yandex.com.tr/harita.

[28] German Aerospace Center, "NetEdit," German Aerospace Center, [Online]. Available: https://sumo.dlr.de/docs/Netedit/index.html.

[29] «OpenStreetMap,» [Çevrimiçi]. Available: https://www.openstreetmap.org/.

[30] OpenAI, "Stable Baselines," OpenAI, 2018. [Online]. Available: https://stable-baselines.readthedocs.io/en/master/.

[31] D. Kahneman, A. B. Krueger ve D. A. Schkade, «A Survey Method for Characterizing Daily Life Experience: The Day Reconstruction Method,» *Science*, cilt 306, no. 5702, pp. 1776-1780, 2004.

[32] N. Lelyveld and S. Grad, "Traffic still tops crime, economy as top L.A. concern, poll finds," Los Angeles Times, 2015. [Online]. Available: https://www.latimes.com/local/lanow/la-me-ln-traffic-still-tops-crime-economy-as-top-l-a-concern-poll-finds-20151007-story.html.

[33] J. J. Kim, S. Smorodinsky, M. Lipsett, B. C. Singer, A. T. Hodgson ve B. Ostro, «Traffic-related Air Pollution near Busy Roads The East Bay Children's Respiratory Health Study,» *American Journal of Respiratory and Critical Care Medicine*, cilt 5, p. 170, 2004.

[34] Tutkish Statistical Institute, TUIK, "Road Motor Vehicles, March 2022," Tutkish Statistical Institute, TUIK, March 2022. [Online]. Available: https://data.tuik.gov.tr/Bulten/Index?p=Road-Motor-Vehicles-March-2022-45706.

[35] L. Weng, "Curriculum for Reinforcement Learning," 2020. [Online]. Available: https://lilianweng.github.io/posts/2020-01-29-curriculum-rl/.

[36] OpenAI, "Quantifying Generalization in Reinforcement Learning," OpenAI, 2018. [Online]. Available: https://openai.com/blog/quantifying-generalization-in-reinforcement-learning/.

[37] N. Casas, "Deep Deterministic Policy Gradient for Urban Traffic Light Control," 2017.

APPENDIX

Appendix A: Intersection and Phase Behaviors

```
class Phase:
    def __init__(self, traci, intersection_id, intersection_config, phase_id,
phase_lanes):
        self.traci = traci
        self.intersection_id = intersection_id
        self.intersection_config = intersection_config
        self.phase_id = phase_id
        self.phase_lanes = phase_lanes

    def set_tl_green(self):
        cnf_len = int(len(self.intersection_config) / 2)
        for i in range (cnf_len):
            tl_id = self.intersection_id + '_' + self.intersection_config[i *
2]

            tl_state_id = self.intersection_config[i * 2 + 1] - 1
            self.traci.trafficlight.setPhase(tl_id, tl_state_id * 3)

    def set_tl_yellow(self):
        cnf_len = int(len(self.intersection_config) / 2)
        for i in range (cnf_len):
            tl_id = self.intersection_id + '_' + self.intersection_config[i *
2]

            tl_state_id = self.intersection_config[i * 2 + 1] - 1
            self.traci.trafficlight.setPhase(tl_id, tl_state_id * 3 + 1)

    def set_tl_red(self):
        cnf_len = int(len(self.intersection_config) / 2)
        for i in range (cnf_len):
            tl_id = self.intersection_id + '_' + self.intersection_config[i *
2]

            tl_state_id = self.intersection_config[i * 2 + 1] - 1
            self.traci.trafficlight.setPhase(tl_id, tl_state_id * 3 + 2)

    def get_phase_occupancy(self):
        occupancy = 0
        lane_occupancies = []
        for lane in self.phase_lanes:
            lane_occupancy = len(self.traci.lane.getLastStepVehicleIDs(lane))
            occupancy += lane_occupancy
            lane_occupancies.append(lane_occupancy)

        return occupancy, lane_occupancies


class Intersection:
    def __init__(self, name, id, traci, phases, phase_times, cycle_time):
        self.name = name
        self.id = id
        self.yellow_time = 1
        self.red_time = 1

        self.cycle_time = cycle_time
        self.current_cycle = 1

        self.traci = traci

        self.phases = self.init_phases(phases)
        self.phase_len = len(phases)
        self.order = [i for i in range ( self.phase_len ) ]
        self.current_order = 0

        self.phase_time = phase_times
        self.phase_time_cumulative = [ sum( self.phase_time[:i + 1] ) for i
in range( self.phase_len ) ]

        self.current_time = 0
        self.current_yellow_time = 0
        self.current_red_time = 0
        self.current_green_time = 0

        self.current_phase = 0
        self.current_phase_type = 0 #0 for green, 1 for yellow, 2 for red

        self.occupancy = None
        self.prev_occupancy = None

    def init_phases(self, phases):
        phases_arr = []
        for i, phase in enumerate(phases):
            phases_arr.append(Phase(self.traci, self.name, phase['config'],
i, phase['lanes']))

        return phases_arr

    def reset(self):
        self.current_cycle = 1
        self.current_time = 0
        self.current_yellow_time = 0
        self.current_red_time = 0
        self.current_green_time = 0
        self.current_phase = 0
        self.current_phase_type = 0 #0 for green, 1 for yellow, 2 for red
        self.current_order = 0


    def set_phase_time(self, new_phase_time):
        # increase all the phase duration by minimum phase duration
        pt = [npt + 10 for npt in new_phase_time]
        self.phase_time = pt

    def take_step(self):
        self.current_time += 1

        if (self.current_phase_type == 1): # if yellow time is over go on
with red
            if (self.current_yellow_time == self.yellow_time):
                self.current_phase_type = 2
                self.current_yellow_time = 0
                self.current_red_time = 1

                self.phases[self.current_phase].set_tl_red()

            else :
                self.current_yellow_time += 1

        elif (self.current_phase_type == 2): # if red is over go on with
green
            if (self.current_red_time == self.red_time):
                self.current_phase_type = 0
                self.current_order = (self.current_order + 1) %
(self.phase_len)
                self.current_phase = self.order[self.current_order]

                if (self.current_phase == 0):
                    self.current_cycle += 1

                self.current_red_time = 0
                self.current_green_time = 1

                self.phases[self.current_phase].set_tl_green()

            else :
                self.current_red_time += 1

        elif (self.current_phase_type == 0): # if green time is over go on
with yellow
            if (self.current_green_time ==
self.phase_time[self.current_phase]):
                self.current_phase_type = 1
                self.current_green_time = 0
                self.current_yellow_time = 1

                self.phases[self.current_phase].set_tl_yellow()

            else :
                self.current_green_time += 1

    def get_occupancies(self):
        self.prev_occupancy = self.occupancy
        occupancies = [0 for i in range (len(self.phases))]
        lane_occupancies = [ [] for i in range (len(self.phases))]

        for i in range (len(self.phases)):
            occupancies[i], lane_occupancies[i] =
self.phases[i].get_phase_occupancy()

        self.occupancy = occupancies

        return occupancies, lane_occupancies
```

Appendix B: Interaction With the Model and Gym Environment

```
obs = environment.reset()
while True:
    action, _states = model.predict(observation)
    observation, rewards, is_done, info = environment.step(action)
    if (is_done):
        break
    environment.render()
```

CURRICULUM VITAE

EXPERIENCE

2017	Summer Intern, Kayseri Ulaşım A.Ş., Kayseri, TURKEY
2018	Summer Intern, Boğaziçi University, Istanbul, TURKEY
2018 – 2022	Computer Engineer, Kayseri Ulaşım A.Ş., Kayseri, TURKEY
2022 – Present	Computer Engineer, Apptec360, Basel, SWITZERLAND (Remote)

EDUCATION

2014 – 2019	B.Sc., Computer Engineering, Abdullah Gul University, Kayseri, TURKEY
2019 – Present	M.Sc., Electrical and Computer Engineering, Abdullah Gul University, Kayseri, TURKEY