Reinforcement learning methods for traffic light control: a state of the art review and TLCOPT - an open source tool for easier experimentation

Keywords: reinforcement learning, agent, traffic light control, optimization, tools, execution management.

Abstract: Traffic light control optimization is nowadays an important part of a smart city with the advancement of sensors, IoT and edge computing capabilities. The traffic control optimization method targeted by our work follows a general trend in the community: dynamically switching traffic light phases depending on the current traffic state. Reinforcement learning was lately adopted in the literature as it has been shown to outperform previous methods. In our literature review, we found a gap in the tools that connect the research area of reinforcement learning with traffic control and simulation environments. Our primary goal in this work is to bridge this technical gap and facilitate the development of both independently. The secondary goal is to provide an introduction and overview of the state-of-the-art in reinforcement methods for the traffic signal control optimization problem. We also evaluate different algorithms for training policies to observe their performing particularities from both performance of results and efficiency.

1 INTRODUCTION

A *smart city* is defined by the European Commission as a place where networks and public services are efficiently harnessed through communication and information technologies for the benefit of its citizens and businesses (Commission, 2021). Its aim is to create a region with sustainable economic growth and to achieve high quality in the fields of economy, mobility, environment, people, law and administration. The topic our group deals with is the optimization of the traffic flow within a city through an efficient management of the available traffic lights. This type of optimization is usually done by dynamically calibrating the timing of traffic signal phases at connected intersections. As the literature review shows, the latest trend in solving this problem is through reinforcement learning (RL) methods. The main reasons for using RL methods are as follows: (a) The optimization problem can be first simulated and optimized in a simulated environment until an optimal strategy for timing the traffic light phases is found. (b) The current state of the art in using RL methods outperforms previous work (baseline methods), as shown in Section 2. (c) The solution chosen for intelligent traffic light control must be dynamic, making decisions over time that depend on deep insight into the overall state of the city or region being optimized.

Existing work in this area attempts to solve the optimization problem at two different levels:

1. **Microscopic** - Considering the optimization of individual vehicles (e.g., how long does a vehicle wait at a traffic light), especially when optimizing a single intersection or a series of connected intersections.
2. **Macroscopic** - divides the physical space of the city into streets, lanes, areas and aggregates the metrics and data on large areas.

In general, microscopic-level optimizations are more valuable and detailed, but at the cost of being harder to train and converge than macroscopic.

The purpose of the work presented in this paper is threefold:

1. To provide a framework that allows separation between the methods used for optimizations and the knowledge and datasets of transportation administration. In this way, RL researchers can reuse their expertise more easily and with less technical effort to perform optimizations without knowing deep technical details about traffic management in general. On the other hand, users who are familiar with traffic management concepts can immediately reuse existing open source RL libraries out of the box with minimal knowledge in this particular area.
2. An overview of the state of the art in reinforcement learning methods, tools, and datasets available for experimentation. This section serves as an introduction to the field for both research sides mentioned above.
3. Comparison of state-of-the-art algorithms on a real-world dataset using the RESCO benchmark (Ault and Sharon, 2021).

The implementation is open source and available at https://github.com/unibuc-cs/TrafficFlowOptimization. At the level of the framework itself, we identified the following innovations in our work:

- First open source tool at the microscopic level that can leverage various modern single and multi-agent algorithms from popular open source RL libraries such as TFAgents (Guadarrama et al., 2018), OpenAI Baselines (Dhariwal et al., 2017), PFRL (Fujita et al., 2021), or RLlib (Liang et al., 2018).
- A comprehensive list of output options with relevant default metrics for real-world scenarios. The user can easily add new metrics around our default metrics.
- Code-level separation between methods used for optimization and traffic management concepts and their technical depth. See section 3 for more details.
- Plotting functions built on top of tensorboard, specifically adapted for the metrics of the traffic light control problem.
- A method to plot locations on a map and simulate traffic under various conditions, even in the absence of real-world data.
- Distributed training with RLlib using strategies to connect to the SUMO simulator (Krajzewicz et al., 2012). Multiple workers and/or environments per workers variants are supported.

The paper is structured as follows. Section 2 contains a detailed overview of RL methods used for traffic light control optimization and datasets or benchmarks available for experimentation. Section 3 contains an architectural sketch of our framework implementation and its features. Evaluation is performed in Section 4. Finally, conclusions and our future work plan are presented in Section 5.

2 RELATED WORK AND INTRODUCTION TO THE FIELD

The workflow used in the literature generally consists of capturing portions of a real city with one or more intersections connected, transferring the environment and the captured real traffic data into a simulation environment, and then training algorithms to find an optimal strategy for calibrating the traffic light timing. The sub-methods used range from classical Q-learning methods to policy gradients (PG) with actor-critic to natural gradients (NG). Real-world traffic data is collected using sensors, video cameras, or statistical data on the population, its traffic patterns, lane density over time, etc. A summary of these methods can be found in (SUMBA, 2020).

If traffic data is not available or not sufficient for a city, there are some methods in the literature that can be used to simulate and optimize strategies based on realistic data. The *Four-Step* model described in (Ortúzar and Willumsen, 2011) and (Button and Hensher, 2007) is one of the most common methods to replicate real traffic flow along routes in simulated environments. The method divides the area into traffic analysis zones and then models the traffic flow considering the possible activities in the area (e.g., schools, universities, business and commercial places, etc.). Although it is used to simulate scenarios at the macroscopic level, it can also be adapted at the microscopic level by first generating probability distributions, from which the behavior is then retrieved for each individual vehicle. The *Agent-Based Demand* (SUMBA, 2020), (Innes and Kouhy, 2011), traffic generation is more suited to the microscopic level by generating activities and trajectories for each vehicle agent.

We divide this section into two parts. In the first part, we present some previous work and attempts to use open-source or commercial libraries that can be used for experiments and how they compare to our work. The second part is devoted to the state of the art of algorithms used for traffic light control optimization, focusing on reinforced learning methods, which seems to be the latest trend in solving the targeted optimization problem through our work.

2.1 Environment Simulation Tools

The base layer of any reinforcement learning algorithm requires an environment in which it can operate. In our case study, a traffic simulator is needed that replicates real traffic data or samples from an approximately real distribution. After a review of open source tools, we concluded that SUMO (Krajzewicz et al., 2012) is the most widely used and maintained tool. Users can take advantage of its ability to replay data, generate random traffic data, and it has an API to interact with from different environments, as we needed in our implementation. It also supports direct import of real parts of urban infrastructure via the OSM standard (OSRM, 2020) and ASAM (for Standardization of Automation and Systems, 2021) (note: the difference between OSM and ASAM is that the roads in ASAM also have a 3D topology, which allows for more realistic experiments when cities do not

necessarily have apartment ground). It also supports integration with CARLA (Dosovitskiy et al., 2017) for 3D simulation and visualization. An alternative to SUMO was MatSim (Horni A. and K.W, 2016). However, at the moment it seems to be less maintained and provide less support than our first choice.

2.2 Datasets

As for datasets, we considered datasets that can either be used directly, i.e., provide data that can be used directly at the microscopic level, or that can be adapted to our goal, i.e., contain data sources that can generate probabilities that can be sampled by the simulator at simulation time. Caltrans (of Transportation, 2021) collects data from street-level sensors and induction loops. It is used by the state of California to monitor traffic. The dataset, which is publicly available online, provides researchers with raw data collected from real traffic to experiment with. Induction loop data is counted and traffic density data is stored on local regional servers every 30 seconds. The dataset is created using the aggregated data from these local servers and is constantly updated. The data can be imported into SUMO simulator by connecting the locations of data sensors on longitude and latitude with the urban infrastructure obtained from OSM. A tool in our framework enables this connection in a simple way for experiments. Although originally intended for the macroscopic level, we found that the data can be used to create distribution probabilities, which can then be retrieved at the individual vehicle level to simulate a realistic traffic scenario.

Datasets from (Horni A. and K.W, 2016) can be imported into SUMO to replicate real-world traffic distribution as well.

Bolt (BOLT, 2020) provides real traffic data directly at the micro and macro levels. At the micro level, the data comes from each driver in a time series format where each entry includes a timestamp (recorded at regular intervals of 2 seconds on average), location (latitude and longitude), vehicle orientation, and acceleration. At the macro level, it provides information about the average speed on roads and lanes reported at different times, linking the collected data to real events and specific times of the year, month and day. Information is also provided on intersections and the distribution of routing in different directions.

An open-source tool for benchmarking algorithms and evaluating them against a real-world dataset collected in Cologne (Germany) is described in RESCO (Ault and Sharon, 2021). We use their dataset in our evaluation. Although this work includes a set of known algorithms and compares them to using the dataset based on a custom implementation via the PRFL library, the purpose of the work is to provide a benchmark and a complex dataset rather than a toolkit for experimentation with different methods and algorithms. Our paper attempts to bridge this gap and provides source code and tools to separate the technical aspects of controlling the simulation and datasets from the reinforcement learning methods, state and reward matching, algorithms for training, and managing their execution.

2.3 Reinforcement learning methods for traffic light control

The baseline controllers (following (Ault and Sharon, 2021)) used in general for evaluating against the RL methods are the following:

1. Fixed-time control: where each phase of a traffic signal at an intersection is activated for a fixed duration and follows a fixed cycle.
2. Max-pressure control: where the phase combination with the maximal joint pressure is activated. (Chen et al., 2020).
3. Greedy-control: where the phase combination with the maximal joint queue length and the number of approaching vehicle count is activated first (Chu et al., 2019)].

An important motivation for our work is that these baseline methods are outperformed by most recent work that uses RL as a method for finding dynamic policies that optimize the process of switching traffic light phases. In this section, we review the state of the art of methods that use reinforcement learning for the traffic signal control optimization problem targeted by our work, and present how the different methods can be connected from theory to simulators and real traffic scenarios. Due to space limitations, we first refer readers new to the field of reinforcement learning to (S. Sutton and G. Barto, 2018).

One of the first seminal works on optimizing traffic with RL is (Wiering et al., 2004). The authors model a set of interconnected intersections using a graph G with two different types of nodes: (a) entry nodes - where vehicles enter according to certain distributions, (b) nodes for each intersection equipped with traffic lights. The created simulation environment considers as state per vehicle $S = (P, D)$, where P is the position of the vehicle, while D is the destination point. For each individual, the shortest path (with a 10% error) is considered. The reward for each individual agent v at time t, $R(v,t)$, is -1 if the car has to wait at a traffic light, or 0 if it can proceed without

waiting. The total reward at time t is then the sum of the rewards, $\mathcal{R}(t) = \sum_{v \in AllVehicles(t)} R(v,t)$. In terms of an RL problem, the optimization problem means finding the optimal traffic light switching times such that we minimize the waiting times of vehicles in G over a given period of time, i.e., obtain a policy that yields better rewards over time. The state of the RL problem definition is the state of all traffic lights and the number of vehicles waiting at each traffic light. The actions in the environment represent the switching of the traffic lights between their different states, i.e., two states - green and red. A tabular Q-learning is used to evaluate the value of state and action transitions: $Q(s,a) = \sum s' R(s') + \gamma V(s')$, where $V(s')$ is the known average value for state s'. These values are determined and maintained during the episodic simulation in the environment. The final result is a trained strategy that performs actions according to the different states.

Although the ideas in (Walraven et al., 2016) were originally applied to the problem of relieving traffic congestion on highways, they can also be applied to optimizing traffic flow in cities, especially at the entrances and exits of bridges or at the intersection of major arterials with minor arterials. The work is based on the idea of limiting the speed on certain sections to avoid congestion. It divides a highway into N sections and considers the average speeds on each section, v_i and the density, k_i on that section. The main evaluation metric is the total travel time of vehicles. Since the equations in the paper are specifically designed for highways, we will only list the main ideas and explain how we can adapt them for the optimization case in a city. The authors describe the problem as a Markov decision process and then use Q-learning to optimize the mentioned metric. In this system, the state at time t is mainly represented by $s_t = \{(v_i, k_i\})$, where i iterates on the space of all sections of the road. The considered reward is 0 if the minimum speed on all sections is higher than a defined threshold u, negative otherwise, depending on the waiting time of cars on all sections between two consecutive time points t and $t+1$, and a parameter c:

$$r_t = \begin{cases} 0 & \text{if} \min\{v_i((t+1)c) \\ & \mid i = 1, \ldots, N > u\}, \\ -h(tc,(t+1)c) & \text{otherwise} \end{cases}$$

, where

$h(b,e) = T \sum_p \left(\sum_{i=1}^{N} [M_i L_i k_i(p)] + \sum_{i=0}^{N} w_i(p) \right)$, and M_i, L_i, w_i, p represents the number of sections, lanes, segments, respectively. The number of vehicles queued at each entry/exit of a road p. The actions are represented by the speed limit intervals applied on the sections to get the best rewards. The system is then trained to estimate $Q(condition, action)$ from real data and find a greedy strategy that dynamically sets the speed limits. Even if the speed limit condition is practically impossible to achieve, it can be computed the other way around in the case of a city: How could we set the traffic lights to achieve this speed limit if we assume that cars are traveling at the speed limit.

The work in (Lin et al., 2018) notes that previous work is based only on local optimizations at intersections, and proposes an extension of the methods to manage multiple intersections simultaneously. While the proposed model is a toy in terms of experiments since 9 intersections are considered in a grid, the work is very valuable in laying the foundations for the use of Deep Reinforcement Learning in traffic signal control optimization. The state space is modified so that each intersection has sensors that collect 2 types of information: the number of waiting vehicles and the average speed of vehicles per direction. Since there are at most 4 by 4 choices within an intersection, a state representing each intersection is a tuple of the form (2,4,4). The action space consists of 4 phases representing the possible states of the traffic light at an intersection (Fig. 1). The action of the traffic light agent is then one that either moves the phase from the current one to the next. Formally, the action is a tuple of the form $(N_{TLS}, 2)$, where the first axis has dimension N_{TLS}, i.e., the number of traffic lights in the region under consideration, and the second axis has dimension 2, since it represents the probability of either staying in the current state or moving to the next one. Each phase is assumed to last at least 5 seconds, and the transition between phases (yellow color state) takes a fixed time of 3 seconds. We use these timings in our work and consider them as default parameters unless adjusted by the users. Following previous experiments, the authors propose two types of rewards:

1. Local reward: measures the difference between the maximum number of vehicles waiting at the intersection from the NS (north-south) and WE (west-east) directions at each traffic light i, Eq. 1.

$$r_t^{\text{TLS}_i} = -\left|\max q_t^{\text{WE}} - \max q_t^{\text{NS}}\right| \qquad (1)$$

2. Global reward: it measures the difference between the number of vehicles that passed the traffic light and the number of vehicles that were stopped at all traffic lights at time t, Eq. 1.

$$r_t^{\text{Global}} = \left\| \text{Veh}_t^{(\text{out})} \right\| - \left\| \text{Veh}_t^{(\text{in})} \right\| \qquad (2)$$

Your final reward is calculated as in eq. 3 combined, where β assumes a value of 0 at the beginning of the training and increases to 1 with the success of the training episode. Intuitively, local optimization on each intersection is tried more at the beginning, then

we move on to optimizing the global state and coordinating between different intersections.

$$r_t = \beta r_t^{\text{Global}} + (1-\beta)\frac{1}{N_{\text{TLS}}}\sum_{i}^{N_{\text{TLS}}} r_t^{\text{TLS}\,i} \qquad (3)$$

The actor-critic method is used, and PPO (Schulman et al., 2017) is chosen for training stability. A novelty of the work is the use of deep meshes to map the state of 9 intersections in the city. Instead of considering numerical values as in previous works, the authors divide the region into a grid of size *WxHxC*. Convolution, ResNet and fully connected layers are used to map the state for the Actor and the Critic. A similar solution is presented in (Casas, 2017), but the authors use DDPG (Lillicrap et al., 2019) instead of PPO. However, we have not yet seen a solid comparison between the two methods.

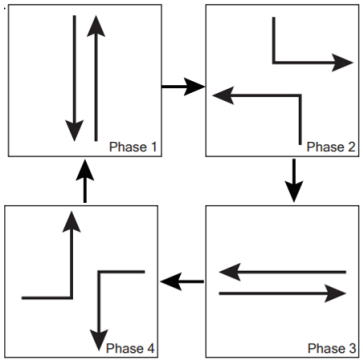

Figure 1: The four phases of a traffic light agent considered in (Lin et al., 2018). The agent's actions can either maintain the phase or cycle from the current phase to the next

.

Another novel approach to represent the state space for a deep reinforcement learning solution is the work presented in (Genders and Razavi, 2016). First, they partition the traffic lanes into segments of maximum capacity for the discretization of physical space, similar to the idea in (Walraven et al., 2016). Formally, the state space becomes: $S \in (\mathbb{B} \times \mathbb{R})^{\frac{l}{c} \times n} \times P$, where: l is the length of a lane, c is the length of a cell into which the cells have spatially partitioned the space under consideration, n is the size of the physical space (a grid); $\mathbb{B}$ is a boolean value indicating whether or not there is a car at that location, R is the average speed of the cars in that cell, and P is the phase under consideration at that time. If we consider A as the set of actions in the system (similar to Fig. 1), then $Pis \in \mathbb{B}^{|A|}$. For training their methods, the authors use A3C (Mnih et al., 2016) for efficient parallelization of the experiments. This is something that we also consider in our work and tools. We strongly believe that high level parallelization of experiments is very important to get better policies in less time when computational resources are available.

In (Wu et al., 2020), the authors make the transition to a multi-agent reinforcement learning system where each traffic light intersection is a single agent. They modify a version of the DDPG algorithm where the strategies of the agents are synchronized between each traffic light agent. The paper thus proposes a transition from multi-objective learning (where the reward was a single vector simulating multiple agents) to a true multi-agent system. It is also mentioned that the use of LSTM layers for both the Actor and the Critic stabilizes the results, since the agents generally have only partial access to observations in the environment.

The Deep Q-learning implementation is used in many other works, but recently the implementation in (Ault et al., 2020) has gained a lead of up to 19.4% over the other solutions using the same learning method. The novelty of their work is the representation of the convolutional Q-network, which uses features such as queue length, number of approaching vehicles, total speed of approaching vehicles, and total waiting time. These are then aggregated into convolutional layers over the lanes that form the same approaching road. The reward formula used is (minus) the total waiting time at the traffic light.

In (Chen et al., 2020), the authors introduce the concept of pressure to coordinate multiple intersections. Pressure is defined as the difference between the length of queues in the arriving lanes of an intersection and the length of queues in the receiving lane of a downstream intersection. They use the FRAP (Zheng et al., 2019) model and pressure as both a state and a reward for a DQN agent distributed across all intersections.

A novel topology and architecture is FMA2C (Chu et al., 2019), which enables cooperation between signal control agents (one per intersection), called workers, in a hierarchical topology. Neighboring workers are coordinated by a local discounted neighbor reward, discounted neighbor states, and joint actions. Workers are the leaf of a hierarchical network in which the leading agents are trained to optimize traffic flow in their assigned region. The higher-level goals of their leading agent are adopted by the workers coordinated below them.

In the same domain, it is also worth noting how Bolt (BOLT, 2020) uses a Markov model to estimate the time it takes to get from an origin to a destination. This uses the collected data (as explained in the previous section) and the standard routing engine OSRM (OSRM, 2020) to find the shortest path between two

points on the globe.

3 Framework description

The methods behind our framework are based on Deep Reinforcement Learning. The end user can extend it through functional hooks (documentation is available on the repository address) from different points to achieve different: (a) optimization goals as rewards, (b) observations - what information is available from the city/traffic system and how it can be mapped in a numerical format, (c) the algorithms chosen for training the traffic light agents, (d) the platform solution for parallelizing and distributing the work when a wide range of computational resources is available.

3.1 Single and Multi Agent RL Support

Our framework supports both single and multi-agent modes. In single agent mode, a single RL agent is created to control all traffic lights in a given area. In multi-agent mode, there is a single agent per traffic light that acts cooperatively as a zero-sum game as much as possible. Figure 2 shows the software stack used within our framework for experiments between single and multi-agent modes. In single agent mode, the framework can leverage a number of open source libraries to demonstrate its flexibility, such as TFAgents (Guadarrama et al., 2018), PFRL (Fujita et al., 2021), OpenAI Baselines (Dhariwal et al., 2017). To provide a similar abstraction for both modes, we use the PeetingZoo library implementation (Terry et al., 2020a). This allows the framework to provide common OpenAI Gym interfaces (Brockman et al., 2016) for both modes, as required by the RL research community. In the case of multi-agent RL mode, the machine learning algorithms also need preprocessing operations such as normalizing the length of actions and observations, since the number of lanes may be different for each traffic light agent (intersection). For these operations, we use the Supersuit (Terry et al., 2020b)package, which builds wrappers on top of existing code to perform the required preprocessing operations without changing the user's source code. We felt this was the best option available, as it allowed researchers to bring in their existing RL source code more quickly for experimentation. At the execution management level, we leverage the RLlib library (Liang et al., 2018), which contains a set of implemented multi-agent algorithms out of the box and a transparent parallelization execution paradigm, to operate on high-performance computing infrastructures.

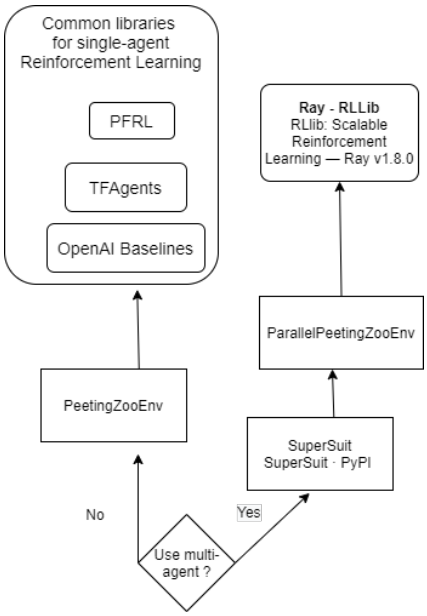

Figure 2: The software stack used in our framework to support single and multi-agent reinforcement learning.

3.2 Environments Abstraction

The full structure of a shared environment using the OpenAI Gym (Brockman et al., 2016) interface is shown in Figure 3. Our environment class is wrapped around two decorators from the PeetingZoo library, which in turn uses some of our preprocessing hooks created with the SuperSuit tool: PadActions and PadObservation - to populate the observation and action spaces so that all agents have the same dimensionality, regardless of the number of lanes each traffic light agent has, OrderEnforcer - to add an order to the execution steps between synchronizing agents (note that we support both asynchronous and synchronous decision making in multi-agent mode; both variants can be addressed in practice, as smart cities can coordinate traffic lights to enforce an order for switching traffic lights), OutOfBoundary - mechanism for checking harmlessness to ensure that data is read/written within correct bounds. The SumoEnvironmentPZ class serves as an adapter between the base environment AECEnv, implemented in the PeetingZoo library based on the Agent Environment Cycle Game (Terry et al., 2021) idea, and the SumoEnvironment class. The difference between PeetingZooEnv and ParallelPeetingZooEnv is that the former assumes that each agent knows what decision the other agents have made at each step, and therefore acts in a fixed order. The second uses the concept of Partially Observable Stochastic Games (POSGs) (Tomásek et al., 2021), where the agent cannot see what other agents have done in the current step, i.e.

they act in parallel.

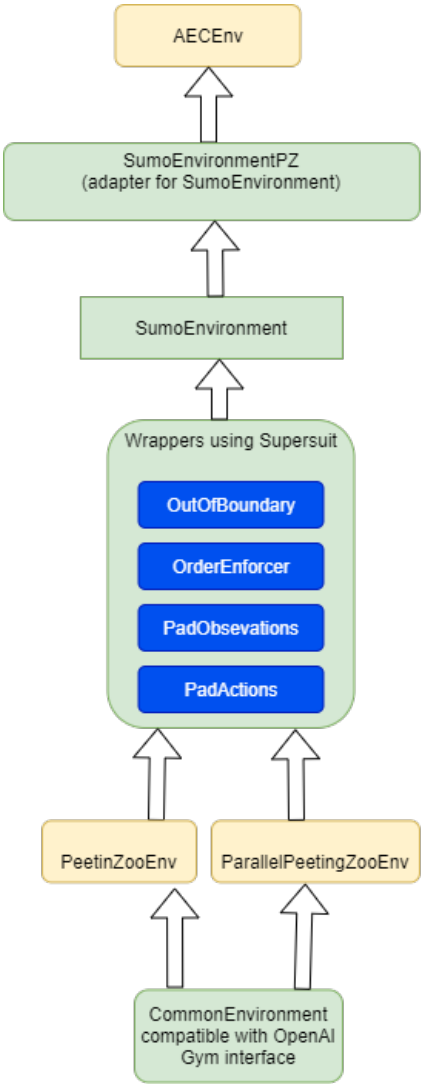

Figure 3: The layered architecture of our OpenAI Gym compatible interface (Brockman et al., 2016) for environments abstracting both single and multi-agent execution modes. The green colored boxes are implemented in our framework, the yellow ones are from the PeetingZoo library (Terry et al., 2020a), while the blue color represents reusable components from Supersuit (Terry et al., 2020b)

Our core implementation of the environment class resides in the SumoEnvironment class, which communicates with SUMO (Krajzewicz et al., 2012) to manage the simulation. The full picture of this component is shown in Figure 4. First, a scenario is procedurally loaded into the episode within the SUMO simulator. The scenario contains a traffic infrastructure layout, which can be a real or a toy example for experimentation, and the simulation data for the trajectories of each vehicle, which comes either from recorded real data sets or from fitted probability distributions from which samples are drawn. The second step is to use the API provided by SUMO to retrieve the information about the traffic light phases, the current fixed times and the lane topology at the intersection. For each intersection, the SumoEnvironment instance creates a TrafficLight object instance that contains this information and manages it over time. There are three important operations that can be overridden by the user for their own experiments. We provide state-of-the-art out-of-the-box implementations for each:

1. compute_observation: this function retrieves stored information about the state of each lane at the intersection, current phase, timing information, etc. from SUMO environment. The user can override this to create convolution spaces, for example. We also implement some of the latest proven standard implementations as examples.
2. compute_rewards: it returns the reward between different time steps of an episode. The example rewards implemented by default are based on stored data for features such as: average wait time at each traffic light and lane, pressure, total number of vehicles waiting in each lane, and density. The user can calculate other rewards based on these characteristics or expand the possible characteristics according to the documentation.
3. take_action: In our default implementation, this follows the state of the art - either no action is taken or it moves to the next phase defined in the simulation data for the particular controlled intersection. A possible example of a custom experiment could be to override the default behavior to switch to different phases as needed, not necessarily in a consecutive cycle as the default.

4 Evaluation

We evaluated our implementation with the RESCO (Ault and Sharon, 2021) benchmark and show the results in a specific case involving a scenario with 8 intersections in the Cologne (Germany) area (Figure 5) with real data on road routing, arrival and destination times of cars. As in the original benchmark, we consider the recorded data in the time interval between 25200 and 28800 seconds. The platform used for the training was a cluster of 8 GPUs, Nvidia A100.

Our goal in the evaluation (given also the limited space) was to compare how different RL methods perform from different angles. We chose one of our standard reward, observation, and action spaces

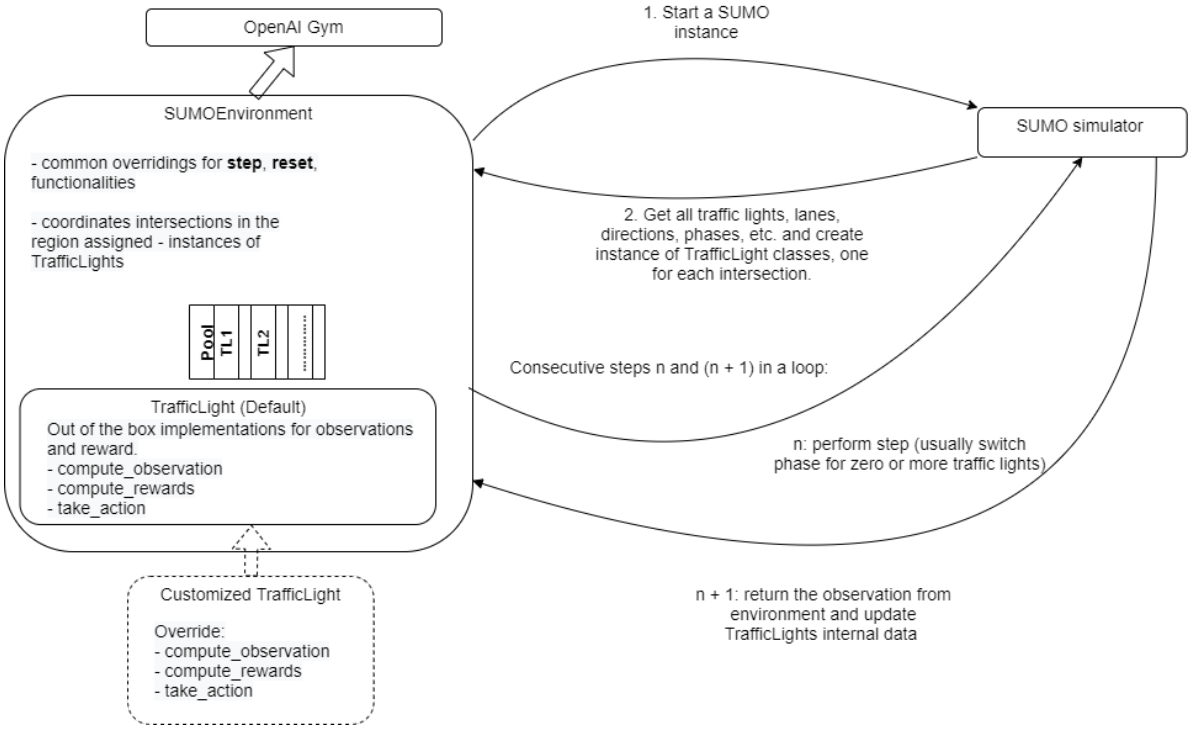

Figure 4: The interaction flow between an environment class, the SumoEnvironment class, and the simulator SUMO (Krajzewicz et al., 2012). The class instantiates a TrafficLight object for each intersection in the scenario loaded in SUMO

.

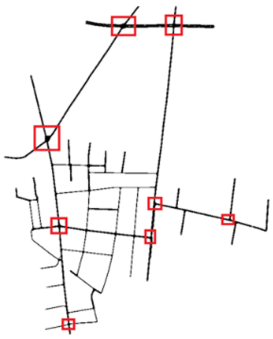

Figure 5: The region in Cologne where the data used in the analysis was collected. The red boxes show the locations of the intersections. Each is assigned a traffic light agent

by following the literature and observing the similarities between the state of the art (Lin et al., 2018). The performance of the algorithms could change significantly if these standard operations are overridden by the user in various ways. The framework provided has a custom plotting mechanism on Tensorboard that is specific to the metrics for the traffic light control optimization problem. According to the documentation provided in the repository, the set of metrics captured can be easily extended by the end user.

The algorithms selected for comparison were chosen from the best performing algorithms in different classes: DQN (Wang et al., 2016) (version with duel networks, target networks and prioritized experience rendering), PPO (Schulman et al., 2017), A3C (Mnih et al., 2016) and SAC (**?**). We divide the evaluation options into three categories:

- Best runs for each algorithm: the graphs in Figures 6, 7, 8, 9 below show the best runs for each algorithm evaluated and the following metrics collected: The average and the maximum waiting time of vehicles at a traffic light, the average and the maximum number of vehicles per lane in the evaluated scenario.
- Training progress over time: the graphs in Figures 10, 11, 12, 13 show how the same set of metrics as in the first point above evolve over time (realtive to the episode index).
- Training efficiency: how quickly each algorithm manages to train a given number of episodes (Figure 14). This can be valuable for large scenarios (in terms of the number of intersections to optimize).

As a conclusion from the comparison of algorithms, we can say that the best performance in terms of collected relevant metrics are (Mnih et al., 2016) and (Haarnoja et al., 2018). When evaluating the

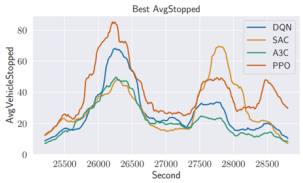

Figure 6: Best run comparison for the average number of vehicles stopped at a traffic light

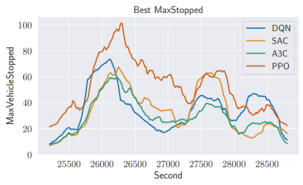

Figure 7: Best run comparison for the maximum number of vehicles stopped at a traffic light

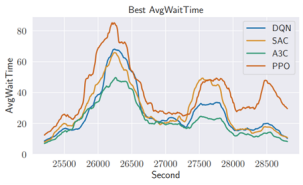

Figure 8: Best run comparison for the average waiting time of vehicles at a traffic light

.

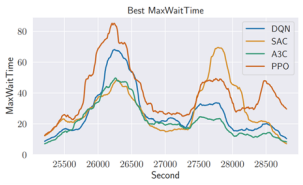

Figure 9: Best run comparison for the maximum waiting time of vehicles at a traffic light

.

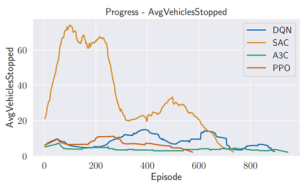

Figure 10: Comparison of progress for the average number of vehicles stopped at a traffic light.

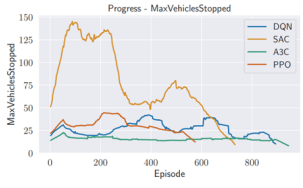

Figure 11: Progress comparison for the maximum number of vehicles stopped at a traffic light.

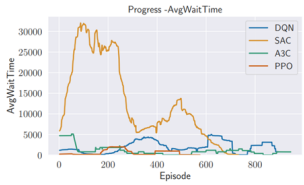

Figure 12: Progress comparison for the average waiting time of vehicles at a traffic light.

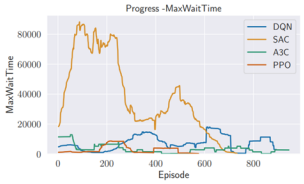

Figure 13: Progress comparison for the maximum waiting time of vehicles at a traffic light.

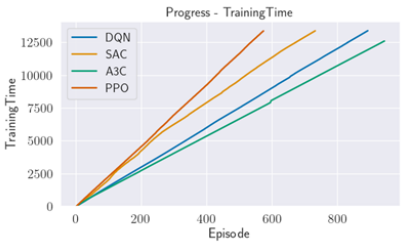

Figure 14: Compare training time speed to get to different episode counts.

most efficient method in terms of training time, PPO (Schulman et al., 2017) proved to be the optimal solution. This was to be expected considering the theoretical trade-offs between the different classes of algorithms in terms of episode sample complexity and training stability. Future work may draw further conclusions between these classes in the presence of very large scenarios with a significant number of intersections that are simultaneously optimized cooperatively.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we first presented the state of the art in traffic light control optimization using reinforcement learning methods and dynamic switching of traffic light phases. We then justified the importance of developing a set of tools to separate reinforcement learning (RL) methods from the technical aspects of applying the techniques to the targeted optimization problem. We hope that our work will help the community to perform more experiments in a simpler way than before. In the paper, we have compared a number of well-known algorithms in the field of RL. Our plan is

to conduct more experiments and adopt various modern methods from the computer vision field for state mapping. We also want to provide more out of the boxes implementations for various rewards, observations mapping and metrics that prove to outperform the state of the art results. Graph neural networks (GNN) (Deepmind and Google, 2020) that could map the environment states for the RL agents is another topic that interests our group too.

REFERENCES

Ault, J., Hanna, J. P., and Sharon, G. (2020). Learning an interpretable traffic signal control policy.

Ault, J. and Sharon, G. (2021). Reinforcement learning benchmarks for traffic signal control. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

BOLT (2020). Simulating cities for a better ride-hailing experience at bolt.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Button, K. and Hensher, D. (2007). *Handbook of Transport Modelling*, volume 1. Emerald Publishing.

Casas, N. (2017). Deep reinforcement learning for urban traffic light control. *Master Thesis in Advanced Artificial Intelligence, Department of Artificial Intelligence Universidad Nacional de Educación a Distancia*.

Chen, C., Wei, H., Xu, N., Zheng, G., Yang, M., Xiong, Y., Xu, K., and Li, Z. (2020). Toward a thousand lights: Decentralized deep reinforcement learning for large-scale traffic signal control. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3414–3421.

Chu, T., Wang, J., Codecà, L., and Li, Z. (2019). Multi-agent deep reinforcement learning for large-scale traffic signal control.

Commission, E. (2021). Smart cities.

Deepmind and Google (2020). Traffic prediction with advanced graph neural networks.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. (2017). Openai baselines. https://github.com/openai/baselines.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). Carla: An open urban driving simulator.

for Standardization of Automation, A. and Systems, M. (2021). Asam opendrive.

Fujita, Y., Nagarajan, P., Kataoka, T., and Ishikawa, T. (2021). Chainerrl: A deep reinforcement learning library. *Journal of Machine Learning Research*, 22(77):1–14.

Genders, W. and Razavi, S. (2016). Using a deep reinforcement learning agent for traffic signal control.

Guadarrama, S., Korattikara, A., Ramirez, O., Castro, P., Holly, E., Fishman, S., Wang, K., Gonina, E., Wu, N., Kokiopoulou, E., Sbaiz, L., Smith, J., Bartók, G., Berent, J., Harris, C., Vanhoucke, V., and Brevdo, E. (2018). TF-Agents: A library for reinforcement learning in tensorflow. https://github.com/tensorflow/agents. [Online; accessed 25-June-2019].

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

Horni A., Nagel, K. and K.W, A. (2016). *The Multi-Agent Transport Simulation*, volume 1. Ubiquity Press.

Innes, J. and Kouhy, R. (2011). *The Activity-Based Approach*, pages 243–274. Palgrave Macmillan UK, London.

Krajzewicz, D., Erdmann, J., Behrisch, M., and Bieker-Walz, L. (2012). Recent development and applications of sumo - simulation of urban mobility. *International Journal On Advances in Systems and Measurements*, 3 and 4.

Liang, E., Liaw, R., Moritz, P., Nishihara, R., Fox, R., Goldberg, K., Gonzalez, J. E., Jordan, M. I., and Stoica, I. (2018). Rllib: Abstractions for distributed reinforcement learning.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2019). Continuous control with deep reinforcement learning.

Lin, Y., Dai, X., Li, L., and Wang, F.-Y. (2018). An efficient deep reinforcement learning model for urban traffic control.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning.

of Transportation, C. D. (2021). Caltrans performance measurement system.

Ortúzar, J. d. D. and Willumsen, L. (2011). *Modelling Transport, Fourth Edition*.

OSRM (2020). Routing machine project osrm.

S. Sutton, R. and G. Barto, A. (2018). Reinforcement learning: An introduction second edition.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

SUMBA, P. (2020). Guidance for transport modelling and data collection.

Terry, J. K., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sulivan, R., Santos, L., Perez, R., Horsch, C., Dieffendahl, C., Williams, N. L., Lokesh, Y., Sullivan, R., and Ravi, P. (2020a). Pettingzoo: Gym for multi-agent reinforcement learning. *arXiv preprint arXiv:2009.14471*.

Terry, J. K., Black, B., and Hari, A. (2020b). Supersuit: Simple microwrappers for reinforcement learning environments. *arXiv preprint arXiv:2008.08932*.

Terry, J. K., Grammel, N., Black, B., Hari, A., Horsch, C., and Santos, L. (2021). Agent environment cycle games.

Tomásek, P., Horák, K., Aradhye, A., Bosanský, B., and Chatterjee, K. (2021). Solving partially observable stochastic shortest-path games. In Zhou, Z., editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 4182–4189. ijcai.org.

Walraven, E., Spaan, M. T., and Bakker, B. (2016). Traffic flow optimization: A reinforcement learning approach. *Engineering Applications of Artificial Intelligence*, 52:203–212.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. (2016). Dueling network architectures for deep reinforcement learning.

Wiering, M., Vreeken, J., Veenen, J., and Koopman, A. (2004). Simulation and optimization of traffic in a city. pages 453 – 458.

Wu, T., Zhou, P., Liu, K., Yuan, Y., Wang, X., Huang, H., and Wu, D. O. (2020). Multi-agent deep reinforcement learning for urban traffic light control in vehicular networks. *IEEE Transactions on Vehicular Technology*, 69(8):8243–8256.

Zheng, G., Xiong, Y., Zang, X., Feng, J., Wei, H., Zhang, H., Li, Y., Xu, K., and Li, Z. (2019). Learning phase competition for traffic signal control.