TRAFFIC CONTROL OPTIMIZATION USING FRAMEWORK SUMO-ATCLIB

Anonymous authors
Paper under double-blind review

ABSTRACT

This article proposes the *sumo-atclib* framework, which provides a convenient uniform interface for testing adaptive control algorithms with different limitations, for example, restrictions on phase duration, phase sequences, restrictions on the minimum time between control actions, which uses the open source microscopic transport modeling environment SUMO. The framework shares the functionality of controllers (class *TrafficController*) and a monitoring and detection system (class *StateObserver*), which repeats the architecture of real traffic light objects and adaptive control systems and simplifies the testing of new algorithms, since combinations of different controllers and vehicle detection systems can be freely varied.

At the same time, the algorithms themselves use the same interface and are abstracted from the specific parameters of the detectors, network topologies, that is, it is assumed that this solution will allow the transport engineer to test ready-made algorithms for a new scenario, without the need to adapt them to new conditions, which speeds up the development process of the control system, and reduces design overhead. At the moment, the package contains examples of *MaxPressure* algorithms and the Q-learning reinforcement learning method, the database of examples is also being updated. The framework also includes a set of SUMO scripts for testing algorithms, which includes both synthetic maps and well-verified SUMO scripts such as Cologne and Ingolstadt.

1 INTRODUCTION

Despite the large number of results in the field of adaptive traffic management, this task is still relevant, which is confirmed by the regular appearance of new research. New approaches based on the latest results in the field of Reinforcement Learning (hereinafter referred to as RL) and artificial neural networks are also actively developing. However, there is some discrepancy between the theoretical and practical results. This is due to the fact that, as a rule, algorithms are initially based on model data, which in a real situation is difficult to obtain, especially when controlling in real time, as well as the control method itself, for example, choosing the next phase in real time is a rather difficult method to implement in practice, due to hardware limitations(controllers, communication with a traffic light object) or existing local regulations.

For example, in the absolute majority of the studied works (11) (table 1), the proposed algorithms control traffic light objects by selecting the next phase, that is, every few seconds the algorithm decides to leave the current phase or select any of the list of phases. However, in practice, such management is not always technically available. More often, the most available option is to update the phase durations once in a longer period of time, for example, 15 minutes.

Such a shift in focus in the adaptive control algorithms being developed has also led to a shift in the developed tools. Thus, in the main open benchmarks and ready-made frameworks, only one type of traffic light object management is available. All this complicates the direct comparison of algorithms, since they could use different control methods in different studies. It is also difficult to simulate the operation of the selected algorithm for a specific intersection, since most likely you will have to implement it yourself under existing conditions. This paper proposes a unified framework for the development and testing of adaptive traffic management algorithms, which provides a larger set of available management methods, easily adapts a once-written algorithm to a specific road net-

Table 1: Adaptive traffic control articles analysis.

Control action	articles number	percentage, %
time distribution between phases	5	12
switch/continue phase	8	19
choose next phase	26	62
define phase duration	3	7

work. The framework is written in python and uses the SUMO(3) package as a transport modeling environment.

2 EXISTING FRAMEWORKS

Existing frameworks for the development of adaptive control algorithms use ready-made open packages for modeling traffic flows, among which SUMO and CityFlow (12) can be distinguished. SUMO - Simulation of Urban MObility, is an open source traffic simulation package. It is the most widespread, has a graphical interface, wide functionality (up to the simulation of pedestrians, public transport, the choice of algorithms for following the leader, etc.), is regularly updated. Despite the fact that it is written in C++, it has connectors for various languages that allow you to control the simulation in real time. CityFlow was originally built as an environment for building RL adaptive control algorithms. It was stated that this package has higher performance compared to SUMO in highly loaded scenarios (12), which was rather due to the TraCI(10) connector to SUMO, since the article (5) shows that the simulation time of the same scenarios is comparable. Also, this package has not been updated since November 2021. At the moment, 2 sumo frameworks can be distinguished for the development and debugging of algorithms-rl and LibSignal.

The sumo-rl(4) framework uses SUMO as an environment for transport modeling. This framework also contains a set of RESCO (1) scripts, there is an example of an implemented agent. However, it only implements control by selecting the next phase. Also, the space of possible actions in it is quite poor, it consists only of the possibility of choosing the next phase, but it lacks information about adjacent intersections and pressure, in terms of the *MaxPressure* algorithm, is considered only the general one at the intersection, which makes it impossible to directly apply this algorithm without modifying the framework.

The LibSignal(5) framework can also use SUMO as a modeling module, but at the same time it is possible to use CityFlow. However, this framework also does not have the ability to simulate a situation with scheduled management. However, it is more flexible and already offers more possibilities for customizing the state space, but this requires a fairly deep dive into the framework code.

3 PROPOSED FRAMEWORK

In the proposed framework, it was also decided to use SUMO as an environment for modeling traffic flow. This is the most popular package to date, regularly updated, and has open source code. The framework also includes a set of scenarios for testing algorithms - RESCO. The block diagram of the proposed library is shown in the figure 1

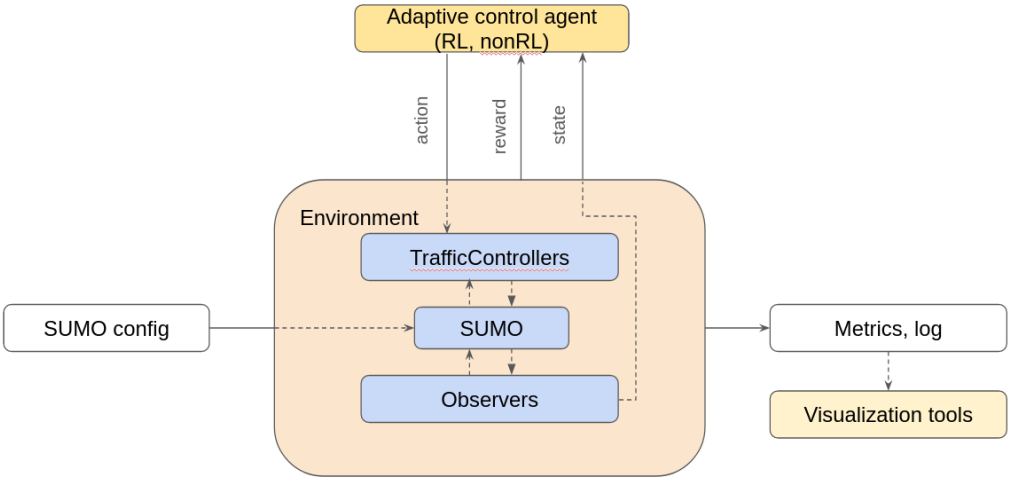

Figure 1: The block diagram of the framework.

The environment consists of a SUMO simulation, intersection controllers (*TrafficControllers* in the diagram) and surveillance systems (*StateObservers*). Controllers control traffic lights and are implemented in two classes - *TrafficLightsScheduleController* and *TrafficRealTimeController*, the first allows you to simulate scheduled control, makes it possible to set acceptable phases durations for a certain period of time, does not imply real-time control, while the second class of controllers, on the contrary, simulates real-time control, and requests with some frequency (for example, 5 seconds) an action Depending on the initialization, it allows you to control in two modes: prolongation of the current phase/activation of the next one (cyclic mode) or selection of the next phase from all possible ones (acyclic mode). Thus, 3 control methods are available - according to a schedule, switching to the next phase, choosing the next phase, and in one simulation, different controllers can be placed at different intersections, which allows you to simulate the existing hardware stack of a specific road network for testing adaptive control algorithms. Another feature of the proposed framework is the unification of lanes (lane in SUMO) into ”roads”, class *Road*. This is due to the fact that in SUMO, and in other environments too, the road from one intersection to another may consist of several sections (edges on the graph), that is, there is one path, but it consists of several sequentially connected edges, this may be, for example, due to constrictions/extensions, thus, if you automatically build a graph for a new scenario, it may turn out that the algorithm controls lanes of several tens of meters at the entrance and exit, and all traffic lights are not connected to each other, although in fact they are adjacent. In order to deal with such a problem, the *Road* class was introduced, which combines the lanes that make up one branch (one can see the difference on the figure 2).

The *Observer* class simulates the operation of a surveillance system on a road network, allowing, for example, to set the surveillance area. In this framework, it is possible to test not only algorithms based on reinforcement learning, but also classical model-based ones, such as *MaxPressure* (8; 9) or evolutionary algorithms in the case of scheduled management.

4 Experiments

This section demonstrates the capabilities of the framework. To build metrics, the methods *MaxPressure*, *MaxPressure* - in cycle mode and *Qlearning* - an approach based on reinforcement learning methods were used, and metrics for a manually selected schedule were also obtained. The methods were tested on a SUMO model of two connected intersections in Moscow, the map is shown in the figure 3.

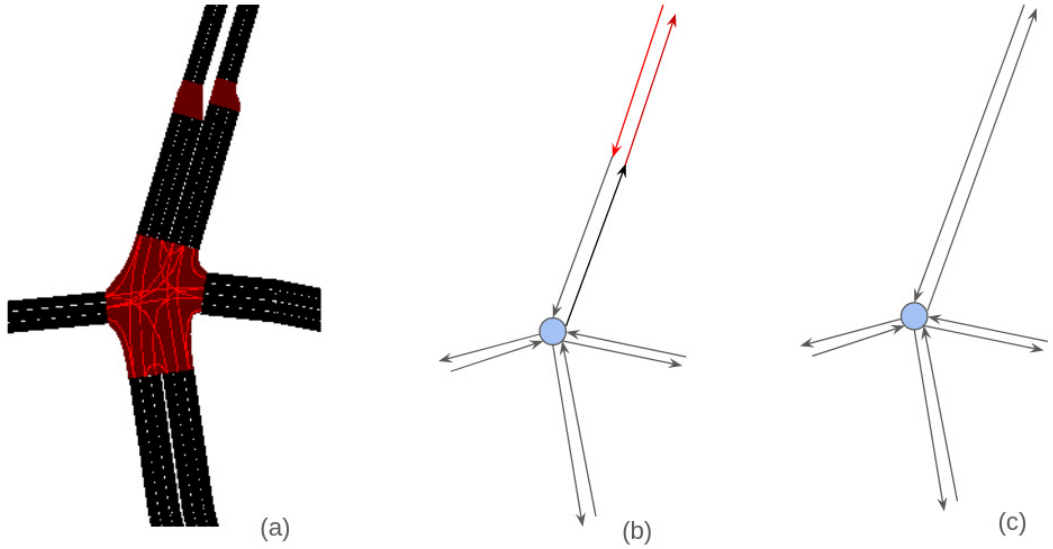

Figure 2: Merging lanes into one road (*Road* class).

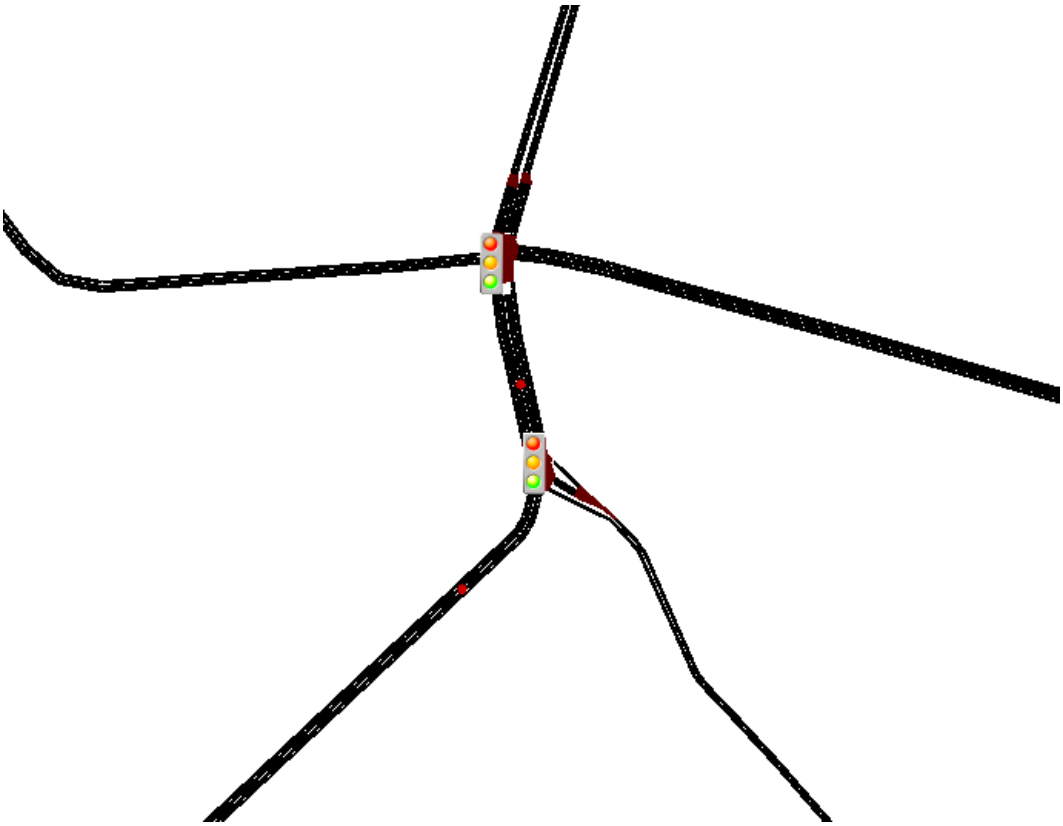

Figure 3: On the map, the northern intersection: the intersection of Atlasov-Moskvitina-Chumakov, and the southern one: the intersection of Atlasov-Valuevskoe-Khabarova

4.1 ALGORITHMS DESCRIPTION

Max-pressure: The algorithm proposed in (9) minimizes the "pressure" at the intersection. The pressure of the phase, informally speaking, is defined as the difference between the number of vehicles in line on the incoming roads of the intersection, which the phase allows to travel, and the number of cars queuing at adjacent intersections of outgoing roads. This decentralized algorithm controls only one traffic light object using information from adjacent intersections, however, it is claimed that under certain conditions it guarantees the stabilization of queues at all intersections of the network.

More formally, the *weight* of the direction is first considered according to the following rule:

$$w(l,m) = q(l,m) - \sum_{n \in Out_m} p(m,n) * q(m,n)$$

where $q(l,m)$ represents the length of the queue from edge l to edge m, $p(m,n)$ the proportion of vehicles heading from m to n. *Pressure* then counts as:

$$\gamma_\phi = \sum_{(l,m) \in \phi} w(l,m)$$

Where ϕ is a set of intersection directions allowed for the phase, in the form of input-output pairs (l,m).

The standard implementation of the algorithm proposed in (9). A variant with real-time control, where the phase sequence may not be observed. The algorithm is described below.

Algorithm 1: MaxPressure

Data: Minimal phases duration t_{min}
Time from the beginning current phase of i-th junction t_i
$\phi_i, \phi_{i,0}$ current and initial phases of the i-th junction
foreach *i junction* **do**
    $\phi_i \leftarrow \phi_{i,0}$;
    $t_i \leftarrow 0$;
end
foreach *timestep* **do**
    foreach *i-th junction* **do**
        if $t_i \geq t_{min}$ **then**
            Evaluate pressure for the i-th junction for each phase $\gamma_{i,\phi}$;
            if $\arg\max_\phi \gamma_{i,\phi} \neq \phi_i$ **then**
                Change current phase to $\arg\max_\phi \gamma_{i,\phi}$;
                $t_i \leftarrow 0$;
            else
                Do not change current phase;
            end
        end
    end
end

The figure 4 shows a listing of a program implementing this algorithm using the interface of the proposed framework.

The described algorithm assumes real-time control, however, this is not always applicable in practice, so its modification is often used, which does not change the order of phases in the cycle, but only distributes the duration of each phase in the cycle. Unlike the original algorithm, here the duration of the cycle phases is simply set in proportion to the corresponding pressures. It is also not difficult to implement this option in the proposed framework, and the figure 5 shows a listing of this algorithm within the framework.

```
env = SumoEnvironment(args)
initial_state, _, done, _ = env.reset()
actions = {}
while not done["__all__"]:
    obs, _, done, i = env.step(actions)
    actions = {}
    for ts in obs.keys():
        pressures = obs[ts]["pressure"]
        actions[ts] = np.argmax(pressures)
env.close()
```

Figure 4: Acyclic *MaxPressure* - free choice of phases, the implementation in the proposed interface.

```
env = SumoEnvironment(args)
initial_state, _, done, _ = env.reset()
actions = {}
while not done["__all__"]:
    obs, _, done, i = env.step(actions)
    actions = {}
    for ts in obs.keys():
        pressures = obs[ts]["pressure"]
        actions[ts] = env.tls_controllers[ts]["available_time"] *
        pressures / sum(pressures)
env.close()
```

Figure 5: Cyclic *MaxPressure* - fixed order of phases, control of time distribution between phases, the implementation in the proposed interface.

Unlike classical algorithms, Reinforcement Learning algorithms have an agent who learns in the process of interacting with the environment, figuratively speaking, gaining experience and drawing conclusions. In order to demonstrate the capabilities of the framework in supporting RL methods, an example of agent training using the Q-learning (7) method was also implemented. The pseudocode of the algorithm is presented in the algorithm 2.

Algorithm 2: Q-learning.

Data: α - smoothing parameter
ϵ - exploration parameter
$Q(s, a)$ - Q-function values table
$s \in S$ - state vector (queues, pressures, ...)
$a \in A$ - action
s_0 - initial state
foreach *step* k **do**
  a_k with probability ϵ is chosen uniformly from A, and with probability $1 - \epsilon$ is defined as $a_k \leftarrow \arg\max_a Q(s_k, a)$
  r_k, s_{k+1} - environment output after step a_k
  Update Q: $Q(s_k, a_k) \leftarrow Q(s_k, a_k) + \alpha(r_k + \gamma \max_a Q(s_{k+1}, a) - Q(a_k, s_k))$
end

This is an example of the off-policy method, since an agent, generally speaking, can learn from examples not only of its interaction with the environment, but also from samples of other agents (policies). That is, you can use the replay buffer (experience replay) and generally speaking, separately collect examples and update the Q-table.

4.2 RESULTS

The simulation result is shown in figures 6 and 7, where *expert schedule* - the phase durations are expertly selected and fixed throughout the episode; *MaxPressure (cyclic)* - algorithm *MaxPressure*, where the duration of the phases is set once before the cycle, the sequence of phases remains unchanged; *MaxPressure (acyclic)* - algorithm *MaxPressure*, where a new phase is selected every 5 seconds, or the current one remains, the sequence of phases changes (this one performs better than cyclic one, similar to (6)); *Q-agent (cyclic)* - implementation of the Q-learning approach, the sequence of phases does not change, the algorithm decides every 5 seconds to switch to the next phase or not; *Q-agent (free choice of phases)* - implementation of the Q-learning approach, in which the algorithm selects the next phase every 5 seconds, the sequence of phases changes. The experiment code can be found in the project repository: `https://github.com/zhelyazik/sumo-atclib/tree/main/experiments` (2).

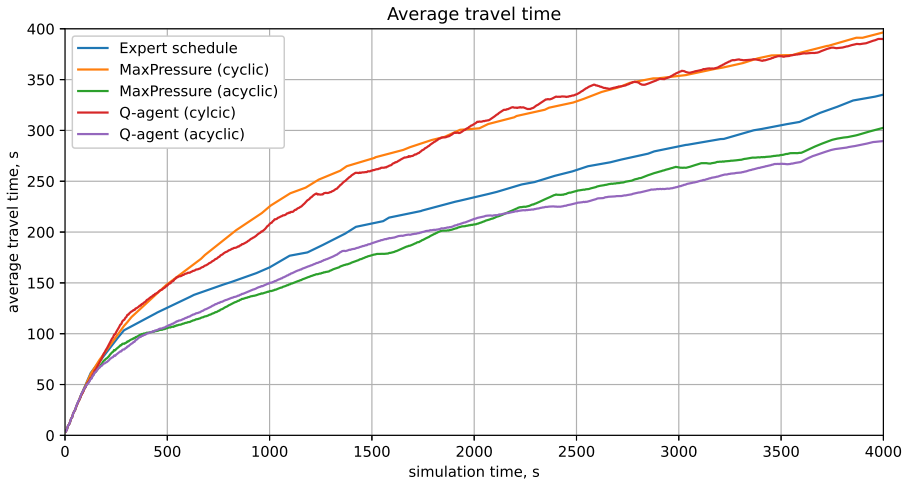

Figure 6: The graph of the average trip time per car, depending on the step of the simulation, the calculation uses jointly the cars that are on the way and arrived at the destination.

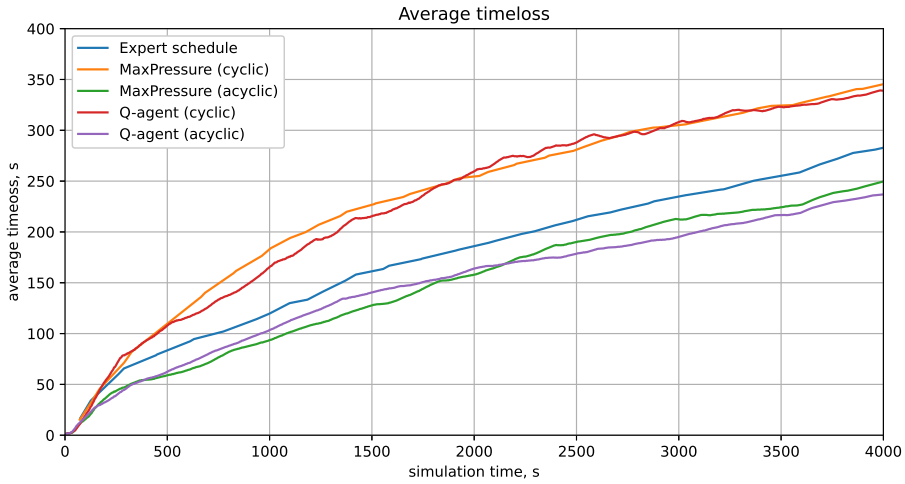

Figure 7: The graph of the average delay in a trip per car, depending on the step of the simulation, the calculation uses jointly the cars that are on the way and arrived at the destination. The delay time of the vehicle is calculated as the total time of all stops.

5 Conclusions

The purpose of this work is to develop a tool that will significantly accelerate the development and research of adaptive traffic management algorithms. Initially, a ready-made framework (sumo-rl) was taken, which as a result of edits was significantly redesigned and its functionality expanded. The framework is available on GitHub at the link `https://github.com/zhelyazik/sumo-atclib` (2).

This tool can be useful for engineers and researchers in the field of traffic management to model and test adaptive control algorithms being developed.

However, there are several limitations in the current version. For example, the reward function, which is used to train RL agents, is still implemented inside the framework and there is no way to connect a custom version. The situation is similar with the *StateObserver* class - it has been fixed so far, and there is no possibility of additional configuration, however, developers can change the framework themselves if necessary. Currently, controllers are implemented that allow you to control traffic light objects in the mode of selecting the next phase (any) in real time, switching to the next phase in the cycle in order, setting the time distribution by phases in the cycle (schedule).

Despite the functionality already implemented, further development is necessary in order to make the framework more flexible and convenient for engineers. The primary task is to fill the framework, in addition to the script base, with a set of ready-made implementations of adaptive control algorithms.

References

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

[2] Kazorin V.: SUMO-ATCLIB. `https://github.com/zhelyazik/sumo-atclib`

[3] Lopez, Pablo Alvarez and Behrisch, Michael and Bieker-Walz, Laura and Erdmann, Jakob and Flötteröd, Yun-Pang and Hilbrich, Robert and Lücken, Leonhard and Rummel, Johannes and Wagner, Peter and Wiessner, Evamarie: Microscopic traffic simulation using sumo. 2018 21st international conference on intelligent transportation systems (ITSC), pp. 2575-2582. IEEE (2018).

[4] Lucas N. Alegre: SUMO-RL. https://github.com/LucasAlegre/sumo-rl.

[5] Mei, Hao and Lei, Xiaoliang and Da, Longchao and Shi, Bin and Wei, Hua: Libsignal: an open library for traffic signal control. Machine Learning, pp. 1?37 (2023).

[6] Sun X., Yin Y.: A simulation study on max pressure control of signalized intersections. Transportation research record, vol. 2672(18), pp. 117?127 (2018).

[7] Sutton R. S., Barto A. G.: Reinforcement learning: An introduction. MIT press (2018).

[8] Varaiya P.: A universal feedback control policy for arbitrary networks of signalized intersections. Published online (2009). http://paleale.eecs.berkeley.edu/varaiya/papers\textbackslashps.dir/090801-Intersectionsv5.pdf

[9] Varaiya P.: Max pressure control of a network of signalized intersections. Transportation Research Part C: Emerging Technologies, vol. 36, pp. 177?195 (2013).

[10] Wegener, Axel and Piórkowski, Michał and Raya, Maxim and Hellbrück, Horst and Fischer, Stefan and Hubaux, Jean-Pierre: TraCI: an interface for coupling road traffic and network simulators. Proceedings of the 11th communications and networking simulation symposium., pp. 155?163 (2008).

[11] Wei, Hua and Zheng, Guanjie and Gayah, Vikash and Li, Zhenhui: A survey on traffic signal control methods. arXiv preprint arXiv:1904.08117 (2019).

[12] Zhang, Huichu and Feng, Siyuan and Liu, Chang and Ding, Yaoyao and Zhu, Yichen and Zhou, Zihan and Zhang, Weinan and Yu, Yong and Jin, Haiming and Li, Zhenhui: Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario. The world wide web conference, pp. 3620?3624 (2019).