RealLight: DRL based Intersection Control in Developing Countries without Traffic Simulators

Sachin Kumar Chauhan, Rijurekha Sen
Department of Computer Science
IIT Delhi
`{csz188012, riju}@cse.iitd.ac.in`

Abstract

Effective traffic intersection control is a crucial problem for urban sustainability. State of the art research seeking Artificial Intelligence (AI), for example Deep Reinforcement Learning (DRL) based traffic control are using traffic simulators, ignoring the shortcomings of traffic simulators used to train the DRL control algorithms. These simulators are limited in capturing fine nuances in traffic flow changes, which can make the trained models unrealistic. This is especially true in developing countries, where traffic flow is non-laned and chaotic, and extremely hard to simulate based on standard microscopic-model based traffic simulation rules. In the given paper, we seek to do away with traffic simulators, and try to train DRL systems with 40 hours of real traffic data deploying cameras at a New Delhi busy traffic intersection, making intelligent traffic intersection control more realistic for developing countries, and hence has been termed as RealLight.

1 Introduction and Background

To learn a good policy, a Reinforcement Learning (RL) agent interacts with the environment and tries to understand the effect of its actions. A suitable target environment is necessary to allow RL learn the desired functionality. A basic RL based traffic control system is depicted in Fig. 1. For real problems, ideally RL should learn the policies interacting with real environments. For traffic control, learning a policy via RL needs exploration which is not allowed by urban authorities on real roads. So, we need alternatives to the real environment. For RL training in lab machines, the behaviour of environment is usually simulated by software modules.

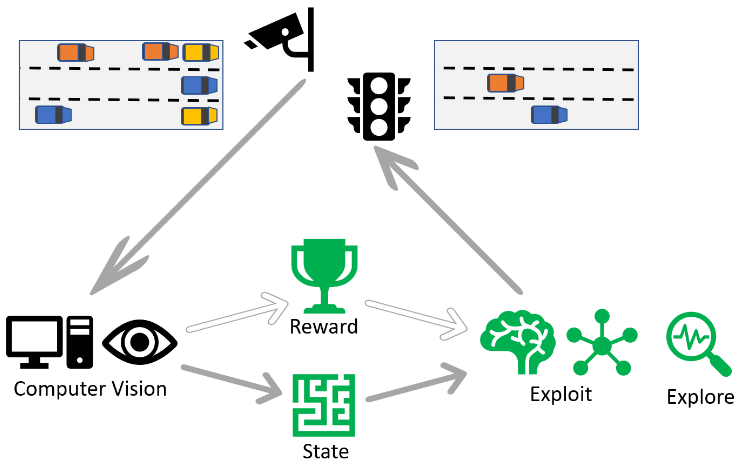
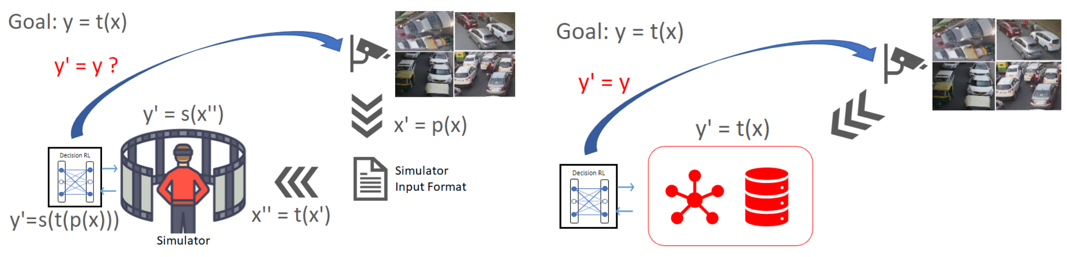


Figure 1: An RL controlled Traffic System

At any instant, the observation we receive from simulators like Zhang et al. (2019); Krajzewicz et al. (2002) is processed to get state and reward. Recent researches, like Wei et al. (2018, 2019); Chauhan et al. (2020), utilize the simulator to show convergence for configured traffic data. Good simulators also provide the observations in terms of vehicles and their placements at each simulation step, which can be processed to extract metrics related to vehicle count and travel times, in addition to immediate rewards and next state for the RL agent training.

2 Issues with Traffic Simulators

Simulator usually contains the physics of traffic movement written as pieces of code and utilizes many hyper-parameters. To give traffic information, we need to adhere to the input requirements of the simulator, providing when+where+which traffic is entering the composed road network. The internal state of the traffic in the virtual road network is selected by the simulator based on the complex configurations. This conversion, from the presence of traffic on the road to the entry of vehicles at the start of road, triggers transformation loss. Thus, as depicted in Figure 2a, the RL agent would learn a combination of these transformations and the peculiarities of simulator along-with traffic behavior.



(a) RL sees spurious transformations with simulators (b) With real data, RL learns the actual behaviour

Figure 2: RL behaviour in simulated vs real scenarios

The traffic on real roads has highly complex structure. Figure 4 shows the simple environment supported in a state-of-the-art traffic simulator, vs. a real traffic intersection. We deployed cameras at a busy 3-approach intersection in a developing country, as shown in Fig. 3, to gather and analyze traffic information.

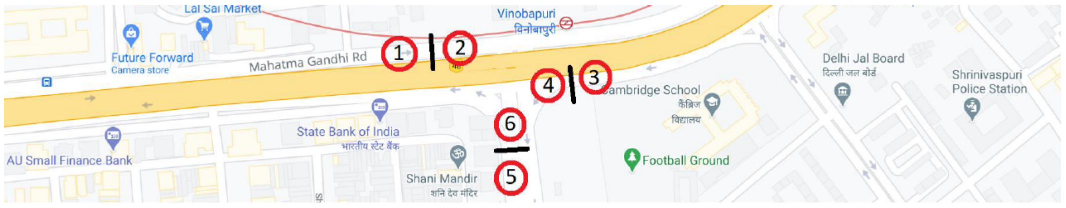
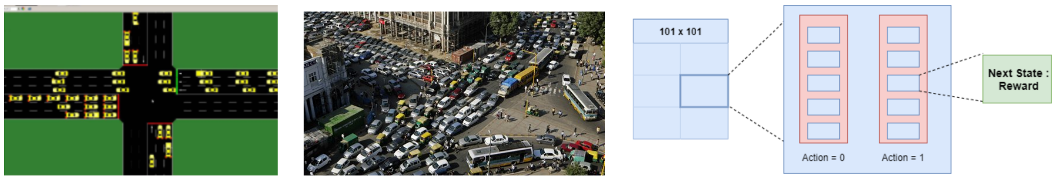


Figure 3: Location of the intersection and installed cameras in New Delhi. Approach 1 has cameras 1 and 2, approach 2 has cameras 3 and 4, approach 3 has cameras 5 and 6.

It is not easy to design experiments to get internal states equivalent to that on the real roads, despite giving simulator-input processed from the same real data and even after tuning the hyper parameters. Even if the parameters are tuned to generate a particular scenario from the road, it's not easy to update the configurations to adapt to the changes in real time traffic. A slight increase in the vehicle density (count per unit time) would saturate the simulated intersection, and a slight decrease would cause a free flow. So, effective use of simulator is highly unlikely to generate the time varying behaviour of the heterogeneous and unruly traffic from the developing countries.



(a) Simulated traffic (b) Real traffic (c) Real Data Structure

Figure 4: Real vs Simulated Environments and Our Data Structure

2.1 Issues with Demonstrations

Some researchers have also shown the feasibility to learn the control policies from demonstrations. An actor-critic method was shown to converge to demonstrations (Xiong et al. (2019)) captured from an SOTL (Cools et al. (2006)) algorithm. For real problems, the gathered demonstrations do not necessarily contain the expert decisions. It is a raw data over the various situations in a given duration, and the onus to learn the right policy lies with the RL. As the raw data can be huge, it usually takes longer training time for RL to converge. Also, as the data is unfiltered and expected to contain outliers (naturally occurring or coming from data capturing inefficiencies), the same would impact the learning of the agent. REALLIGHT seeks to utilize the benefits of both the environment alternatives (simulator and demonstrations).

3 Our approach: REALDS

In this paper, we present a Transition Matrix type Data-structure (named REALDS) which keeps a summary of the traffic behaviour. REALDS, alongwith an accompanying wrapper module, will provide same facility as a regular simulator. The benefit of REALDS would come from the hidden patterns in the data mapped directly from the real scenarios, and its ability to store the frequently occurring situations (states) over an extended period of time. Figure 2b shows the overall working of REALDS, which avoids the multiple spurious transformations in Figure 2a.

3.1 Features of REALDS

REALDS can summarize the states, actions, rewards, next-state, along with other metadata. It can support multiple instances of reward and next-state for the same state action pair and can even be scaled to hold multiple type of rewards. It can provide other information useful for specific experiments and generating metrics. It can be generated utilizing the demonstrations gathered from either the simulations in virtual environments, or the camera deployments done on real roads, though we restrict to the latter here. The design of REALDS is shown in Figure 4c, and we will describe its various components next.

[1] RL influenced State and Reward Design: The purpose of REALDS is to facilitate an effective RL convergence for the real data in offline settings. So, its a good choice to process the real traffic information in the same manner as eventually desired by the RL agent. In a recent research work Chauhan et al. (2020), it has been shown that Queue Density based *State* and Stop density based *Reward* can give an effective convergence to the RL training. Let $green$ denote the queue density on the road approach with green signal and red denote total queue density on all other road approaches with red signals. An effective state is presented as $< green, red >$. We use this Queue Density based 2-dimensional state for our analysis.

[2] Getting Action for Signal Change: For convergence of RL training, we would need to find the effect of various actions taken by the agent. For our settings, the action set would be *Keep* the same phase or *Switch* to the next phase in cycle. So, the knowledge of traffic signal change 0 or 1 would be a necessity as it is the only input of choice from the agent to the environment (REALDS). This can be easily computed by knowing the current phase information at all time instants. There are three ways to get the phase information:

- As we work with real scenarios, the best way is to get the phase policy from the traffic controller. But it is not always feasible as the controller might be a third party solution and the business needs will refrain them from collaborating.
- We can process the camera frame to extract current phase from traffic light color. This would require some extra computation per frame.
- We can process the captured density data to get a measure of signal change. This can be done offline but might give an approximation of the actual setting.

With the information from our deployment partner that the signal policy is constant throughout the day and across different days, we deployed in-situ processing to extract signal information and analyzed the data to get the cycle and phase lengths. Using the phase information, we could find the timestamps of the day where an action of signal change happened. Also, the 2-dimensional state is formed by taking (a) the queue density of the single green approach (b) averaging the queue densities for the approaches with red signal.

[3] Quantization of States: Quantization is performed to discretize the values of Queue Density from various approaches. This is necessary to integrate the continuous values to a linked data structure. The 2 dimensional-state representation and a quantization of lvl levels would give $(lvl + 1)^2$ possible state space. In Figure 4c, the 101x101 structure holds the state space for a quantization of 100 levels. The high quantization tracks changes in density due to smaller vehicles like motorcycles. For only big vehicles, lesser quantization is sufficient.

[4] Formulation of SARS tuples For the consideration of a final Reward, we sum the Stop Densities of all approaches. We use this as a continuous value without discretization. Along-with the quantized states and Action values, we form the standard SARS tuple for Q-Learning (RL) as <*State, Action, Reward, NextState*>. NextState is the state of the next time instant in consideration.

[5] MisraGries or CounterBased Summary: In the raw data processed for multiple days, some states <*green, red*> will appear multiple times and maybe for both the action choices (0 and 1). We can group all their tuples. For effective storage and utilization, we can find the most occurring NextStates for each action choice for each State. The Reward for the same <State,NextState> pairs are averaged to form a single tuple per pair, separately for the Actions. We can use Misra and Gries (1982) algorithm or count based processing to find the heavy hitters among the many samples. Misra-Gries is a streaming algorithm to find most occurring samples in an online sequence. In the offline setting, both Misra-Gries and Count based filtering can extract heavy hitters.

So, the REALDS incorporates adjacency list data-structures per Action per State. We keep the most frequent NextStates for each given State, along with corresponding average Reward. For a limit of 10 most occurring NextStates, REALDS would require upto 2 *(Actions)* x 2 *(NextState + Reward)* x 10 *(NextState)* = 40 entities per State. It would require 40 x 101^2 odd memory in total. Figure 4c shows the most frequent NextState per Action for a State. The REALDS can be extended with other information like the frequency of NextStates.

Extracting most frequent NextStates and discarding the outliers (rarely occurring NextState) come up as an underlying design choice for REALDS. The discarded outliers might contain the special scenarios which needs special attention from the RL perspective. In the presented work, we hold this insight, and focus to present REALDS (with most frequent NextStates) as an effective alternative to virtual simulators and demonstrations. We try to show that REALDS can provide a mechanism for getting a general RL Policy. We will analyze the discarded outliers for their peculiar properties (if any) as a part of our future-work, and will see if their inclusion can improve the existing policy or pave a way for a secondary policy for special circumstances.

4 RL Experiments with REALDS

To analyze the suitability of REALDS in training RL based intersection control agents, we perform some experiments. For training, we utilize a Deep Neural Network with properties as described in Table 1. The 2-dimensional <*Green, Red*> State is passed to the 2-layered 10x10 DNN to output *Keep* or *Change* Action. For exploration, we select a random Action with 0.8 to 0.2 probability and 0.95 decay factor. Based on the Action value, we select any of - (a) the only (b) a random (c) a nearest - NextState from REALDS, with the corresponding *Stop Density* Reward. The Reward, the lower the better, is applied with a weight of -0.25 as a feedback to the DNN.

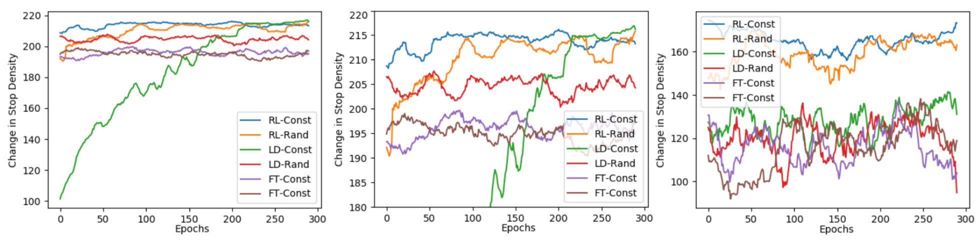
4.1 Baselines

To compare the performance of REALDS, we opted two baselines -
The signal holds to the current phase till a fixed period to time, and then switches to the next phase in cyclic order. This kind of pre-configured time policy is used at most of the intersections in the world, and is a suitable baseline for most research experiments.

(b) Learning from Demonstrations (LD): The real traffic data being processed gives us a sequence of demonstrations as a side product. We utilize the learning the same, to effectively show the stability of REALDS over the same ground data. Whereas REALDS contains the data in a structured way and allows RL to control the actions, with *Learning from Demonstrations*, RL cannot suggest actions during training and blindly accepts the actions as coming from the raw data.

State	Reward	RewardWt	Input	Output	Param	Loss Fn	Optimizer
Queue Density	Stop Density	-0.25	2	2	162	MSE	RMSprop

Table 1: Fully Connected DNN properties



(a) Full 40days data for train+test (b) 34 days train and 6 days test

Figure 5: RL convergence for different train-test splits

4.2 Variants

As REALDS contains traffic information from a wide duration, the initial state per epoch might impact the performance. So, we experiment all the methods over two scenarios -
(a) Const: Start the epoch from a fixed initial State. We selected a mid point <50,50> for our experiments.
(b) Rand: Start the epoch from a random initial State. For trainable methods, the initial State selected for testing in an epoch will be independent of the selection in training part.

For *Fixed Time*, there is no training involved, and we simple test as per *Const* or *Rand* variant. For *Learning from Demonstrations*, using the same initial State will have no significance in training part. So, we always train from random offsets, and test as per *Const* or *Rand* variant. Also, due to the randomness involved in selecting the NextState for REALDS, we might get a different trail in different epochs despite starting from the same State and following the same Action policy. Hence, (a) FixedTime for *Const* will not be a straight line, and (b) unlike *Learning from Demonstrations*, REALDS will start from the same initial State for *Const* experiments, allowing more liberty to the baseline.

To allow a suitable comparison among all the selected methods, we test them over the same REALDS implementation. The *Fixed Time Control* can be applied to any mechanism supporting awaiting a action input. The Learning from Demonstrations need a suitable platform to test the learnt policy, and REALDS is the suitable alternative for current comparison. With an Action received from any of the methods, REALDS shifts to an appropriate NextState. The process continues for an epoch of 3600 seconds, with metrics being accumulated for reporting.

4.3 Metrics and Results

REALDS does not have vehicle count information due to being populated from the Queue Density of the heterogeneous vehicles. Hence, metrics related to the exact count of cleared vehicles cannot be calculated. We plot the performance over *Change in Stop Density*, which is the accumulation of all decrease in Stop Density over an epoch. We attribute the increase in Stop Density as an effect of incoming traffic and ignore that in the metric accumulation. A high value of given metric signifies the ability of the method to select series of actions to clear the standing traffic.

As depicted in Table 6b, we perform two types of experiments with REALDS - (a) with full 40 days data for both the training and testing, and (b) with split data, 34 days data from 3 months for training and 6 days from a month for testing. The corresponding plots are shown in Figure 5(a) and 5(b) respectively. For baseline, we take a Fixed Time (FT) policy with 60 seconds time per approach/phase. Learning from Demonstrations, our second baseline, is denoted by LD. REALLIGHT is denoted by RL. The two variants are denoted by Const and Rand. The vertical-axis present the metrics as *Change in Stop Density* and horizontal-axis denote the 3600 seconds long consecutive epochs.

As seen from the graphs, REALLIGHT converges faster and to a higher value, compared to both baselines. The middle plot zooms the y-axis, to compare better with demonstration based baselines (LD). With train-test split data in Figure 5, where RL is trained with Sep-Nov data and tested on Dec data, the benefits of REALLIGHT becomes clearer. FT does not fit into the traffic pattern in Dec, while LD cannot learn as well as REALLIGHT, from too many traffic states. With 40 days of data spanning 4 months, these results show great promise in gradually moving RL training from simulations to the real intersection settings.

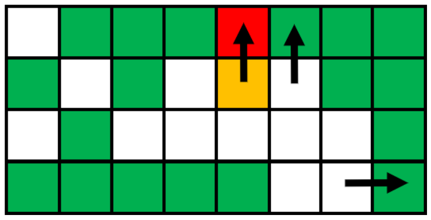
5 Conclusion and Future-work

In the presented research work, we analyzed the benefits REALDS, to be utilized in place of simulator for RL based traffic experiments, which improves on effective learning, computation required and training speed. We will work further to effectively evaluate the REALDS with other methods.

Appendix

A Handling of Unobserved States

For a *State-Action* never observed in real scenarios, the REALDS will contain zero entries, which would halt the usual flow of simulation. To overcome this, RealSim would contain mechanisms to let the simulation continue with the best approximation possible. So, for any State with no information in the REALDS, we can switch to the nearest populated state.



(a) Jumping to the nearest state

	Month	Days	Split	Full
1	Sep	14	Train	Train + Test
2	Oct	10		
3	Nov	10		
4	Dec	6	Test	

(b) Real data gathered

Figure 6: Nearest state and Real data gathered

Figure 6a illustrates this by showing an empty Orange State be replaced with an adjacent and populated Red State. Similarly, the other 2 arrows show the same scenario. For a State with multiple candidates, we can select one based on a policy or randomly. To see the effectiveness of this approach of handling unobserved states, we gathered 40 days of traffic video data spanning over 4 months, as shown in Table 6b, with the help of our industry partner. We extract the data for a duration of 7AM-5PM local time each day to avoid discrepancy in the density calculation computer vision algorithms during the periods of low natural light. We populate the REALDS with this data.

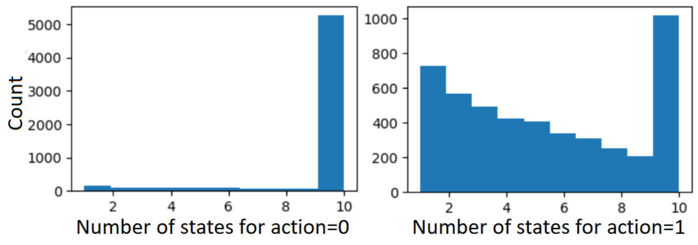


Figure 7: REALDS statistics for 40 days data

The Count of States (vertical axis) for the number of NextStates (horizontal axis) for both Actions is shown in Figure 7. We can observe that NextStates for Action=1 (4737 states) are more sparse than Action=0 (6135 states). This is due to the Action=1 occurring fewer times, during the time of signal switch. Still we have good balance of NextState alternatives for both actions, and will use the same for RL training in the next section.

B REALDS as an RL training environment

For RL training, a wrapper module, with the knowledge of current *State*, will access the REALDS for an *Action* selected by the RL agent. From the multiple choices of NextState per State-Action, the module selects one entry as the response, containing average Reward and NextState (along with other metadata if present). The selection policy can be random, round robin, probabilistic etc as configured. RL agent will use the Reward for the experience and NextState for moving on. The RL agent and REALDS module interaction is similar to a usual Agent Environment interaction. The REALDS and the wrapper module exploiting it as described above, will jointly form an environment equivalent module (named **RealSim**), facilitating simulation based on real traffic data. So, RealSim will behave same as a virtual simulator environment, with an option to load different data configurations (REALDS) from different real demonstrations.

C Maintaining and using REALDS

As REALDS contains summary of the real situations for a specified duration, there might be separate data-structures for weekdays, weekends, rainy-weather, peak-hour etc, which maybe of different duration (continuous or periodic). A suitable RL agent can be trained on one or more REALDS summaries at a backend server. A new intersection will be initialized with suitable RL-Agents trained using REALDS from same or other intersection's prior data. Control agent will transit to an appropriate RL-Agent configuration as per the control specifications from traffic management authority.

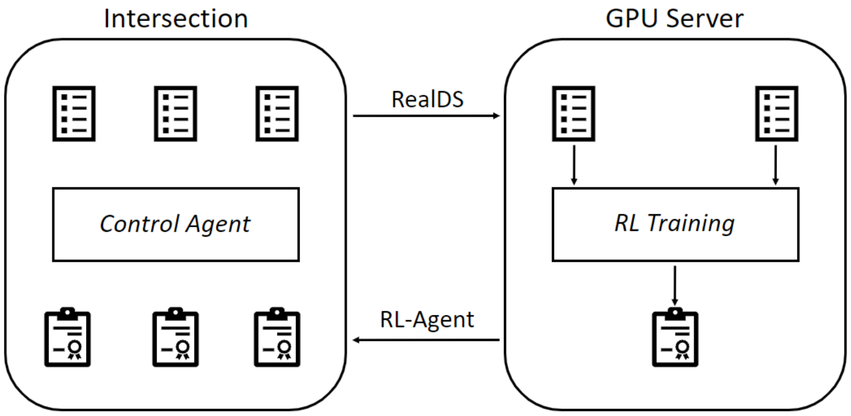


Figure 8: Maintaining REALDS and RL-Agents

Only the RL inference (and not the training) will happen on the edge-device deployed at the intersection. For the purpose of preparing new REALDS, Misra and Gries (1982) can be utilized to store the summaries in a space and computational efficient manner *in situ* at the intersection. Hence, all the heavy processing of RL training on the prepared REALDS can be done on (single) high capacity backend server for various intersections. Using REALDS will reduce the computational cost of training as well (as compared to real-demonstrations), since the data is presented in a compact form. The new RL-Agents can replace or enhance the existing configurations of intersections after human approval. An image depiction is shown in Figure 8.

Acknowledgments and Disclosure of Funding

We would like to thank Aabmatica Technologies Pvt Ltd (New Delhi) for their support in camera deployment and the real data collection.

References

Sachin Chauhan, Kashish Bansal, and Rijurekha Sen. 2020. EcoLight: Intersection Control in Developing Regions Under Extreme Budget and Network Constraints. *Advances in Neural Information Processing Systems 33 pre-proceedings (NeurIPS 2020).*

Seung-Bae Cools, Carlos Gershenson, and Bart D'Hooghe. 2006. Self-Organizing Traffic Lights: A Realistic Simulation. *Advances in Applied Self-Organizing Systems* (10 2006). `https://doi.org/10.1007/978-1-84628-982-8_3`

Daniel Krajzewicz, Georg Hertkorn, Christian Feld, and Peter Wagner. 2002. SUMO (Simulation of Urban MObility); An open-source traffic simulation. *4th Middle East Symposium on Simulation and Modelling (MESM2002)*, 183–187.

Jayadev Misra and David Gries. 1982. Finding repeated elements. *Science of computer programming* 2, 2 (1982), 143–152.

Hua Wei, Chacha Chen, Guanjie Zheng, Kan Wu, Vikash Gayah, Kai Xu, and Zhenhui Li. 2019. PressLight: Learning Max Pressure Control to Coordinate Traffic Signals in Arterial Network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Anchorage, AK, USA) *(KDD '19)*. 1290–1298.

Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. 2018. IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining* (London, United Kingdom) *(KDD '18)*. Association for Computing Machinery, New York, NY, USA, 2496–2505. `https://doi.org/10.1145/3219819.3220096`

Yuanhao Xiong, Guanjie Zheng, Kai Xu, and Zhenhui Li. 2019. Learning Traffic Signal Control from Demonstrations. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (Beijing, China) *(CIKM '19)*. Association for Computing Machinery, New York, NY, USA, 2289–2292. `https://doi.org/10.1145/3357384.3358079`

Huichu Zhang, Zhenhui Li, Siyuan Feng, Chang Liu, Yaoyao Ding, Yichen Zhu, Zihan Zhou, Weinan Zhang, Yong Yu, and Haiming Jin. 2019. CityFlow: A Multi-Agent Reinforcement Learning Environment for Large Scale City Traffic Scenario. *The World Wide Web Conference on - WWW '19* (2019). `https://doi.org/10.1145/3308558.3314139`