An Efficient Deep Reinforcement Learning Model for Urban Traffic Control

Yilun Lin, Xingyuan Dai, Li Li, and Fei-Yue Wang

***Abstract*—Urban Traffic Control (UTC) plays an essential role in Intelligent Transportation System (ITS) but remains difficult. Since model-based UTC methods may not accurately describe the complex nature of traffic dynamics in all situations, model-free data-driven UTC methods, especially reinforcement learning (RL) based UTC methods, received increasing interests in the last decade. However, existing DL approaches did not propose an efficient algorithm to solve the complicated multiple intersections control problems whose state-action spaces are vast. To solve this problem, we propose a Deep Reinforcement Learning (DRL) algorithm that combines several tricks to master an appropriate control strategy within an acceptable time. This new algorithm relaxes the fixed traffic demand pattern assumption and reduces human invention in parameter tuning. Simulation experiments have shown that our method outperforms traditional rule-based approaches and has the potential to handle more complex traffic problems in the real world.**

***Index Terms*—Urban traffic control, Traffic signal timing, Deep reinforcement learning**

I. Introduction

URBAN Traffic Control (UTC) systems aim to better schedule vehicles' movements, exploit the capacity of existing road networks and mitigate traffic congestion in urban areas without significant cost. However, it remains challenging to design an appropriate UTC system, since it is hard to accurately describe the complex nature of urban traffic networks to find proper signal timing plans.

Early UTC systems were mainly built on some simplified traffic flow models and under the assumption of relatively fixed traffic demand patterns within in a short period [1]. However, the success of such approaches relies on the tedious adjustment of experienced transportation engineers. Moreover, the correlations between intersections often vary noticeably from time to time and thus make the pre-defined signal timing plan not optimal.

To solve this problem, model-free data-driven UTC methods, especially reinforcement learning (RL) based UTC methods, received increasing interests in the last decade, along with the fast development of artificial intelligence theory and intelligent control techniques. Instead of optimizing signal timing plan according to simplified traffic flow models, these approaches aim to self-learn the optimal timing policy by analyzing thousands of samples between the change of traffic states and control actions. The invention of human experts in parameter tuning could be replaced by online learning, too.

Among various model-free data-driven UTC methods, Reinforcement Learning (RL) based traffic control receives increasing attention [2], since RL has been successfully used in many applications other than traffic control. In general, Reinforcement Learning allows the system to learn how to choose its behaviors based on feedback from the environment. For traffic control problems, RL based approaches usually take the traffic flow states around the intersections as the observable states, the change of signal timing plans as actions, and the traffic control performance as feedback. After transformation, the traffic control problem will be treated as a standard RL problem and solved by using some standard RL algorithms.

Initial RL based approaches considered the signal timing for isolated intersections [3]. Most of them consist of a classical algorithm like Q-Learning [4] and SARSA [5] to control the timing of a single intersection [6]–[10]. Conventional RL based approaches used tables to record and describe the relationship between the states and actions. As a result, it is difficult to use them for UTC problem with multiple intersections, since the dimension of state-action spaces is too vast to learn.

One solution to this problem is to apply divide-and-conquer policy: divide the studied road network region into small grids containing a few intersections and then solve the traffic control problem for each grid respectively, in the lower-level. In the upper-level, each grid is treated as an agent and is allowed to cooperate to seek a globally optimal solution [11]–[17]. However, multi-agent approaches ease the difficulties of optimization while introducing other problems. For example, it is hard to obtain a real global optimal global control, since each agent usually can only receive limited information [18].

Another solution is to directly attack UTC problems with multiple intersections by using some advanced algorithms to overcome the curse of dimension. For example, Deep Learning (DL) [19], as one of the most recent and successful breakthroughs in AI research, has been introduced and combined with RL methods. The benefit of DL lies in its capability to quickly learn and capture the relationship between the states and actions by using a data structure (deep neural networks) that is more efficient than tables. The integration of DL and

This work was supported in part by National Natural Science Foundation of China (Grant No. 61533019, 71232006), the Beijing Municipal Science and Technology Commission Program under Grant D171100000317002, the Beijing Municipal Commission of Transport Program under Grant ZC179074Z. (Corresponding author is *Li Li*)

Y. Lin, X. Dai and F.-Y. Wang are with the State Key Laboratory for Management and Control of Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100080, China(Email: {linyilun2014, daixingyuan2015, feiyue.wang}@ia.ac.cn) .

Y. Lin, X. Dai are also with University of Chinese Academy of Sciences, Beijing 100049, China.

L. Li is with Department of Automation, TNList, Tsinghua University, Beijing 100084, China (Tel: +86(10)62782071, Email: li-li@tsinghua.edu.cn).

All authors are also affiliated with Qingdao Academy of Intelligent Industries, Qingdao, Shandong, 266109, China.

RL, widely known as Deep Reinforcement Learning (DRL), has already shown its potential by successfully solving video games [20], 3D locomotion [21], Go game [22] and many other problems.

One of the earliest attempts to solve traffic control problem via DRL methods proposed by Li et al. [23] used the Deep Q-Network [20] to control a single intersection. In the follows, researchers had extended such method by applying it to different scenarios, such as traffic light coordination [24]. Such methods have also been improved by proposing new traffic state encoding methods [25], or using different models such as Deep Deterministic Policy Gradient [26].

However, existing DRL based UTC models do not always work well in scenarios with multiple intersections because of the following shortcomings. First, some deep neural networks (e.g., the Deep Q-Network applied in [23]) used for model the relationship between the states and the actions do not fit for large-scale UTC problems that contain multiple intersections. Second, some reward functions recommended for RL do not appropriately characterize the desired state of traffic systems when the correlations between intersections become highly interlaced. Third, some algorithms designed for the training of DRL based UTC models cannot keep a proper balance between solution space exploration and optimal solution seeking. These algorithms are too slow to reach a satisfactory solution for large-scale UTC problems.

To solve these problems, we propose an efficient DRL model dedicated to large-scale UTC problems. First, it uses Residual Networks (ResNet) [27] as the deep neural network model to learn the relationship between the states and the actions. Second, we test different reward functions and design a hybrid reward, in which the throughput of the traffic network, along with the balance of queueing length around intersections is chosen as the performance indexes. Third, it applies a new policy update algorithm, called clipped Proximal Policy Optimization (PPO) algorithm. Moreover, we allow this new model to work with the relaxed traffic demand pattern assumption and the human invention in parameter tuning is significantly reduced.

Tests show that this new model could be optimized within an acceptable time for a traffic grid. Compared with previous DRL models which take thousands of episodes to converge, our method takes only less than 50 episodes to converge for a more complex environment. The entire training stage took only several hours on a workstation with two GPUs. Simulation results show that this deep learning powered UTCS can increase the average capacity of traffic system by 10.91% while reducing the average waiting time by 15.57% compared with the fixed-time controller.

Fig. 1 shows the techniques that we used to handle the interlaced difficulties. To better explain our findings, we organize the paper in the following way. First, we will briefly introduce the background of reinforcement learning in *Section II* for further discussion. Then, we will present how to consider a UTC problem from the viewpoint of DRL in *Section III*. Simulation results will be demonstrated in *Section IV*. Finally, we conclude our contributions and discuss some future applications in *Section V*.

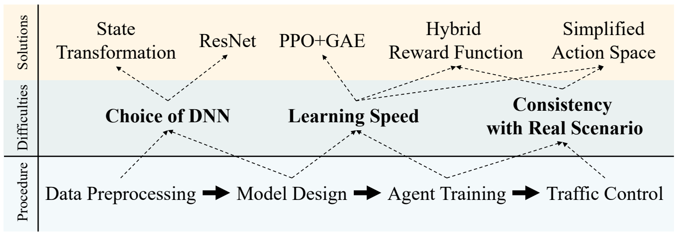


Fig. 1. The major problems addressed in this paper and our contributions.

II. Reinforcement Learning Background

To better present our findings, it is necessary to briefly review the basic idea of reinforcement learning in this section and list the terms/symbols that will be used in the follows; see Table I.

TABLE I
Summary of Notations

Symbol	Meaning
s_t	State of the environment at time step t
a_t	Action taken by the agent at time step t
r_t	Immediate return given by the environment for a_t
R_t	The overall return given by the environment at time step t
π	The policy
$V(s_t)$	The value of s_t, which is the overall return on an infinite time horizon since time step t
$Q(s_t, a_t)$	The Q-value of s_t by taking action a_t
A_t	The abbreviations of the advantage $A(s_t, a_t)$
θ	The parameters of policy/critic model

In RL problems, we assume that an agent interacts with an environment $\mathcal{E}$ over a number of discrete time steps to maximize the reward [28]. An RL problem is often represented by a quintuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_a(s, s'), \mathcal{R}_a(s, s'), \gamma \rangle$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of possible actions. $\mathcal{P}_a(s, s')$ is the probability that action a will lead to state s' from state s in time step t, and $\mathcal{R}_a(s, s')$ is the corresponding expected immediate reward. $\gamma \in [0, 1]$ is the discount factor, which represents the difference in importance between future rewards and present ones.

Our goal is to choose a policy function π that will maximize some cumulative function of the random rewards, typically the expected discounted sum over a potentially infinite horizon from each state s_t:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{1}$$

The policy function π is usually defined as a mapping from the state s_t to the action a_t. Because we usually do not know the state transition probability function $\mathcal{P}_a(s, s')$ in advance, we learn $\mathcal{P}_a(s, s')$ and meanwhile seek the optimal policy by trial-and-error search. At each time step t, the agent receives a state s_t , and selects an action a_t according to its policy π. In return, the agent receives the next state s_{t+1} and receives a reward signal r_t. The process continues until the agent reaches a terminal state after which the process restarts.

To find the desired policy function recursively, we introduce the action value function and the value function. The action value function $Q^{\pi}(s,a)=\mathbb{E}[R_t|s_t=s,a]$ is the expected return for selecting action a following policy π in state s. The optimal action value function $Q^{*}(s,a)=\max_{\pi} Q^{\pi}(s,a)$ gives the maximum action value for state s and action a achievable by any policy.

Similarly, the value of state s under policy π is defined as $V^{\pi}(s)=\mathbb{E}[R_t|s_t=s]$, which is the expected return for following policy π from state s. The optimal value function $V^{*}(s)=\max_{\pi} V^{\pi}(s)$ gives the maximum value for state s achievable by any policy.

In this paper, we consider the neural network-based value function and the associated policy-based methods [29]. Here, we parameterize the policy as $\pi(a|s;\theta)$ and update the parameters θ to maximize the cumulative return. By performing the approximate gradient ascent on $\mathbb{E}[R_t]$, the parameterized policy $\pi(a|s;\theta)$ tends to choose the action a that maximizes future return from state s. One of the earliest algorithms for such methods, called REINFORCE algorithm [30] updates the policy parameters θ in the direction $\nabla_{\theta}\log\pi(a_t|s_t;\theta)R_t$, which is an unbiased estimate of $\nabla_{\theta}\mathbb{E}[R_t]$.

However, the REINFORCE family of algorithms are still time-consuming when the state-action space is large to explore and learn. Most recent works use a variant of this approach called Advantage Actor-Critic (A2C) architecture [28], [31]. In this architecture, we do not strictly follow the direction indicated by the gradient ascent of $\mathbb{E}[R_t]$. Instead, we consider the policy gradient scaled by *advantage* $A_t(a_t,s_t)$ instead of cumulative return R_t.

The advantage $A_t(a_t,s_t)$ is calculated using the return subtracting a learned *baseline* function $b_t(s_t)$. $b_t(s_t)$ can be interpreted as excessive profit gained by taking action a_t in state s_t. In such setting, the policy π is viewed as the actor and the baseline b_t is viewed as the critic. The resulting gradient is estimated as

$$\nabla\theta=\hat{\mathbb{E}}\left[\nabla_{\theta}\log\pi_{\theta}(a_t|s_t)A_t\right] \tag{2}$$

where the expectation $\hat{\mathbb{E}}[\dots]$ indicates the empirical average over a batch of samples.

In this paper, we follow the above idea but use several improved algorithms which yield significantly faster training times and higher data efficiency in many applications. We will present the details of these algorithms later in the *Section III-E*.

III. RL-Based Urban Traffic Control System

In this section, we explain how to build an urban traffic control system using the reinforcement learning method. Instead of designing an RL model that can be used in a specific situation but cannot be generalized well, we aim to provide an architecture that can handle most cases with little adjustment.

A. State Space

In this paper, the data obtained from different sensors are formatted into a 2-D $H\times W$ tensor. More precisely, we format the data collected at the time step t into a triple $\langle C,H,W\rangle$, where C is the number of channels, H is the height of input tensor, and W the width of input tensor.

For example, let us consider the road network illustrated in the bottom of Fig. 2 in the rest of this paper. In this 3×3 grid with 9 intersections, each intersection has 4 arms whose length is 500 meters. Eight sensors are placed on each traffic light to monitor the halting vehicle number and the mean speed. Each sensor is capable of monitoring 150 meters length at most.

Since two types of information: the halting vehicle number and the mean speed are collected in each intersection, the sub-state can then be formatted into a $2\times4\times4$ tensor as shown in the upper left of Fig. 2. The blank cell indicates zero-padding operations. Therefore, the complete state s_t agent received is in a shape of $\langle 2,12,12\rangle$.

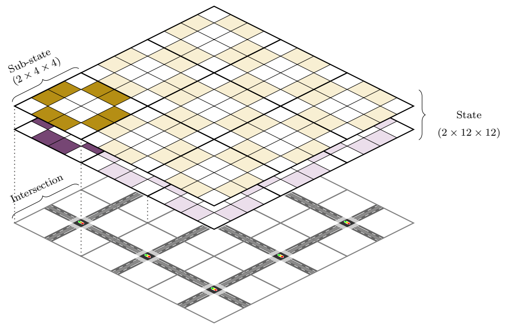


Fig. 2. The traffic grid and corresponding formatted tensor.

B. Action Space

The setting of action space is critical to the successful applications of the RL model. Most previous works use a discrete action space, in which the agent chooses a phase from all possible phases to execute in every time step t. In this paper, we use a similar but simplified action space.

For each intersection, we predefine the possible phases and the order; see Fig. 3 for a demonstration. Specially, in this paper, we assume that a yellow light phase lasting for 3 seconds will be first applied if the traffic light switch from green to red.

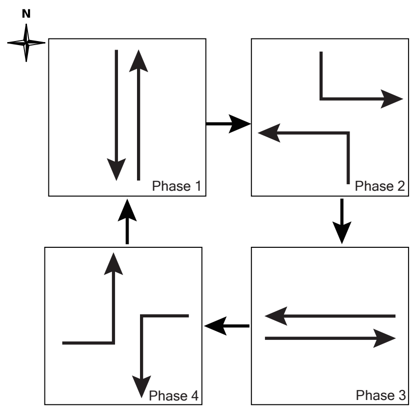


Fig. 3. 4 phases of traffic light, **Straight(NS,SN)**, **TurnLeft(NE,SW)**, **Straight(WE,EW)**, **TurnLeft(WN,ES)** in each intersection. Turn right is always allowed and not shown in this figure.

For every second, the agent can choose either maintain current phase or switch to next one once the minimal phase duration time (5 seconds in this paper) has passed. Since we are using a centralized way to build the controller, the output of actor is in a shape of $\langle N_{\text{TLS}}, 2\rangle$, where 2 indicates the number of the discrete probabilities of choices: either maintaining or switching current phase for each intersection. Here, $N_{\text{TLS}} = 9$ is the number of traffic lights.

C. Reward Function

Unlike the score in many game scenarios, there is no concise yet perfect indicator of the traffic control performance. Generally, it is essential to make the reward reflect the nature of the optimal policy. Meanwhile, it is also vital to avoid the sparse or fluctuate reward signal that is unexpected in a smooth and acceptable learning process.

Various performance indices (e.g., the change in the number of queued vehicles, the change in cumulative vehicle delay, the change in vehicle throughput or the imbalance between different arms of each intersection) had been used to evaluate the traffic system during the last two decades.

In this paper, we divide the reward signal into two parts.

One part is called as the global reward that can lead the agent to learn optimal strategy to maximize the capacity of the whole road network [32]. More precisely, we choose the net outflow of the road network as the global part of the reward. The net outflow is calculated by subtracting the income volume $\|\text{Veh}_t^{(\text{in})}\|$ from the outcome volume $\|\text{Veh}_t^{(\text{out})}\|$ within the selected area at each time step t:

$$r_t^{\text{Global}} = \|\text{Veh}_t^{(\text{out})}\| - \|\text{Veh}_t^{(\text{in})}\| \tag{3}$$

Noted that our experiments are conducted in simulated environments, in which vehicles may teleport (be removed from the network immediately) due to congestions or collisions, we count the outcome volume $\|\text{Veh}_t^{(\text{out})}\|$ without the teleporting vehicles in the experiments.

The other part is called as the local reward that urges the agent to learn to balance the traffic situation for each intersection. Though the local reward does not relate with the capacity of the road network directly, it has been proved to be useful for improving the performance of the controller in many works [33], [34].

The local part of the reward signal also helps to stabilize the agent behavior. Therefore, we choose the opposite of absolute imbalance of each intersection as some previous work did [18], [35], [36]. It is defined as the absolute negative difference between queue length in north-south/south-north direction and those in east-west/west-east direction, i.e.

$$r_t^{\text{TLS}_i} = -\left|\max q_t^{\text{WE}} - \max q_t^{\text{NS}}\right| \tag{4}$$

For each intersection TLS_i, q_t^{WE} is the number of halting vehicle in lanes from west to east or vice versa. Similarly, q_t^{NS} is that from north to south or vice versa.

The complete hybrid reward function can then be formed by summing up the global and local parts.

$$r_t = \beta r_t^{\text{Global}} + (1-\beta)\frac{1}{N_{\text{TLS}}}\sum_{i}^{N_{\text{TLS}}} r_t^{\text{TLS}_i} \tag{5}$$

✓

where β will be gradually increased from 0 to 1 during the learning process. In other words, we let the agent focus on the local tasks first, then use the learned representation to optimize the global behavior.

D. The Deep Neural Network

In reinforcement learning, we model the agent as an Advantage Actor-Critic (A2C) model. The actor refers to a parameterized policy that defines how actions are selected, and the critic is a method that evaluates each action the agent took. In the context of DRL, both actor and critic are implemented by a deep neural network. The structure of this neural network is demonstrated in Fig.4.

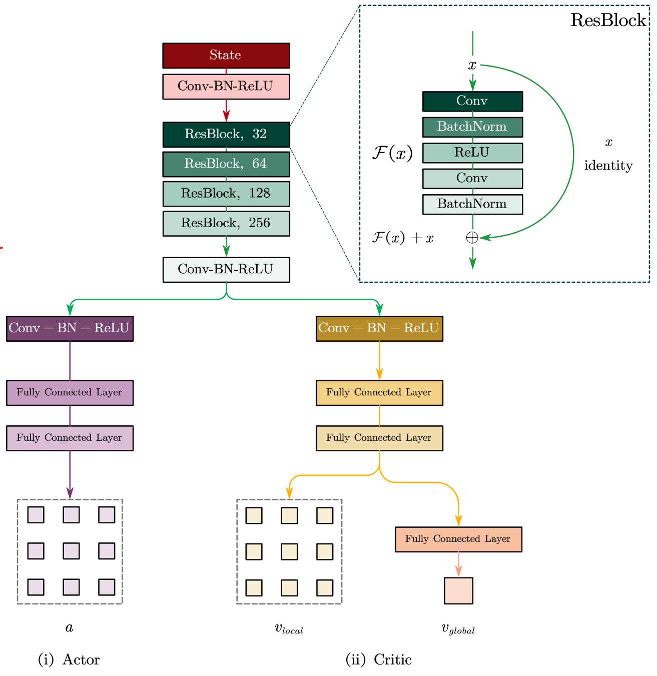


Fig. 4. Schematic of the neural network used in this paper.

The input of the neural network is the state of the system. The input will be fed into shared layers because it is believed that using shared layers for both actor and critic can bring both knowledge transferring and computational efficiency benefit [22]. In this paper, we use 4 stacked Residual Blocks [27] as the shared layers, the output channel of each block is 32, 64, 128, 256 respectively.

Since we use the A2C model here, we set two separate parts to follow the shared layers. On the left bottom of the figure is the *actor*, which has 2 fully-connected layers. It outputs a $\langle N_{\text{TLS}}, 2\rangle$ tensor through a Softmax function, corresponding to the probability to maintain or switch for N_{TLS} intersections.

On the right bottom of the figure is the *critic*. It contains two fully-connected layers as the actor, but with two separate parts of outputs. There are N_{TLS} linear scalar outputs in the left side as the *local critic*, which indicates the local value in each intersection. In the right side, there are two fully-connected layers with one linear scalar output, which is the *global critic* representing the global value.

The outputs of the network are three tensors. Since there are 9 intersections in the simulation environment, the first output is the policy $\pi(s_t)$ in the shape of $\langle 9,2\rangle$. The second output is the local critic $v_{\text{local}}(s_t)$ in the shape of $\langle 9,1\rangle$. The last output is the global critic $v_{\text{global}}(s_t)$, which is a scalar.

E. Learning Algorithm

In general, the parameters of the actor are updated with respect to the critic's evaluation, and the parameters of the critic are updated with respect to the distance between the evaluation and the real return. ~~The standard workflow~~ using modern deep learning library is to define two objective functions respectively, L^{PG} and L^{VF} first, then optimize the parameters of networks with respect to them iteratively. The vanilla actor-critic model is hard to train and requires hyper-parameters tuning carefully, due to the data correlation brought by policy-based methods, high sample complexity for critic model optimization, and the inefficient policy update algorithms. To address these problems, we adopt three recently proposed methods to accelerate the learning speed of the controller.

First, we adopt a parallel reinforcement learning paradigm by synchronously training agents on multiple instances of the environment, and update the network averaging over all the actors. Under such a paradigm, the agents will be experiencing a variety of different states and likely to be exploring different parts of the environment at any given time step. Moreover, we can encourage each actor-learner to use different exploration policies to maximize this diversity. Since the overall changes being made to the parameters by multiple actor-learners applying online updates in parallel are likely to be less correlated in time than a single agent applying online updates, this parallelism can accelerate the exploring speed and decorrelate the data into a stationary process [37].

In our experiments, each of N (parallel) actors collects T time steps of data in each iteration. Then we construct the objective function on these NT time steps of data and optimize it with Adam [38] algorithm for K epochs.

Second, we use an exponentially-weighted estimator of the advantage function, called General Advantage Estimation (GAE) [39], to further accelerate the learning process. As discussed in *Section II*, using advantage function can lower variance while estimating the overall sum of return. However, such an approach typically requires a large number of samples to learn the advantage function. GAE is a recently proposed trick to deal with this problem. It is closely analogous to the TD(λ) algorithm [28]. Compared with vanilla advantage estimation algorithm, which will only bootstrap from the (learned) value function for one step (analogous to TD(0)), GAE can bootstrap for several steps. By increasing the coefficient λ, such method lower the bias of estimation at the cost of increased variance, and therefore can accelerate the learning speed if λ is correctly selected.

Let us define

$$\boxed{\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)} \tag{6}$$

Since δ_t^V is actually an unbiased approximation of advantage at time step t, we can therefore consider a series of k-step estimate $\hat{A}_t^{(k)}$

✓

$$\begin{cases} \hat{A}_t^{(1)} = \delta_t^V \\ \hat{A}_t^{(2)} = \delta_t^V + \gamma\delta_{t+1}^V \\ \quad\cdots \\ \hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l \delta_t^V \end{cases} \tag{7}$$

A truncated version of generalized advantage estimator $\hat{A}_t^{GAE(\gamma,\lambda)}$ can then be defined as the exponentially-weighted average of these k-step estimators:

$$\begin{aligned} \hat{A}_t^{GAE(\gamma,\lambda)} &:= (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + \ldots\right) \\ &= (1-\lambda)\left(\delta_t^V + \lambda\left(\delta_t^V + \gamma\delta_{t+1}^V\right) + \ldots\right) \\ &= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \end{aligned} \tag{8}$$

We use a simplified notation $\hat{A}_t$ to represent $\hat{A}_t^{GAE(\gamma,\lambda)}$ in following paper.

Third, we adopt a new policy update algorithm, called clipped Proximal Policy Optimization (PPO) algorithm [29]. This algorithm seeks to guarantee a monotonic improvement of stochastic policy by introducing a probability ratios $\mathbf{r}_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$, where θ_{old} are the parameters of actor model before current update.

Instead of using advantage directly in policy gradient as mentioned in Eq. (2), such algorithm uses a truncated advantage clip $(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$. This clip term removes the incentive for moving r_t outside of the interval $[1-\epsilon, 1+\epsilon]$, where ϵ is a hyperparameter that changes during the training process. Such a setting will ignore the change of probability ratio when it would make the objective improve, and only include the change when it makes the objective worse. Then we can construct a surrogate objective function L^{PG} whose gradient is the policy gradient estimator.

$$L^{\text{PG}}(\theta) = \hat{\mathbb{E}}\left[r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right] \tag{9}$$

On the other hand, the critic model needs to be trained as well before it can evaluate the value function precisely. The traditional method is to define an objective function L_t^{VF}, then optimize the model by the backpropagation algorithm. Following previous works [29], [37], we define the loss function $L_t^{\text{VF}}(\theta)$ as a squared-error loss between value function and the accumulative return $(V_\theta(s_t) - R_t)^2$, where R_t is calculated according to Eq. (1) and (5). For the reason of computational stability, all rewards are normalized into 1 overall running simulations during the training process.

We modify L^{VF} into a similar form as L^{PG} since we use shared layers for both actor and critic. The modified objective function is average over the unclipped squared-error loss and the clip loss.

$$\begin{aligned} L^{\text{VF}}(\theta) =& (V_{\theta_{\text{old}}}(s_t) + \text{clip}\left[V_\theta(s_t) - V_{\theta_{\text{old}}}(s_t), 1-\epsilon, 1+\epsilon\right] \\ & - R_t)^2 \end{aligned} \tag{10}$$

The objective can further be augmented by adding an entropy bonus to ensure sufficient exploration, as suggested in past works [30], [37]. The following objective function, which will be **maximized** at each iteration, can then be obtained by combining all these terms:

$$L_t(\theta) = \hat{\mathbb{E}}\left[L_t^{\mathrm{PG}}(\theta) - c_1 L_t^{\mathrm{VF}}(\theta) + c_2 S[\pi_\theta](s_t)\right] \tag{11}$$

Here, c_1, c_2 are coefficients of critic loss and entropy bonus, and S denotes the entropy bonus.

Hyperparameters used for following experiments are listed in Table II, where α is linearly annealed from 1 to 0 during the learning process to decay the learning speed.

TABLE II
HYPERPARAMETERS

Hyperparameter	**Value**
Horizon (T)	64
Learning rate (Adam)	$1.0 \times 10^{-4} \times \alpha$
Num. episodes	50
Num. epochs	3
Minibatch size	64×16
Discount (γ)	0.99
GAE parameter (λ)	0.95
Number of actors	16
Clipping parameter ϵ	$0.1 \times \alpha$
L^{VF} coeff. c_1 in Eq. (11)	1.0
Entropy coeff. c_2 in Eq. (11)	0.01

IV. Simulation Results

To validate the effectiveness of the proposed DRL model, we carry out a number of simulation tests. All experiments were conducted using the traffic micro-simulator SUMO v0.32.0 and its Python API [40].

A. Traffic Demand Settings for Simulation Tests

For each instance of simulation, the initial state is a traffic network without any vehicles, then vehicles with a random destination and a corresponding route will be inserted randomly into the network. Each simulation will last for 1 hour (3600 seconds).

Since we seek to propose a method that can be generalized for any situations, the traffic demand is generated randomly via a Binomial distribution $B(b, \frac{1}{np})$ to mimic general cases, where b is the maximum number of simultaneous arrivals and $\frac{1}{p}$ is the expected arrivals in a second. In the training phase, b and p are sampled uniformly from $[10, 60]$ and $[0.1, 2]$, which means the traffic production is around 1800 to 36000 veh/h.

To introduce reasonable randomness, we divide an hour in simulation into 4 periods. For every 15 minutes, the routings of vehicles will be alerted. We use two normal distributions to characterize the routings of vehicles. One distribution controls the probabilities that via which edge a vehicle enters the network and the other controls via which edge a vehicle leaves. Such settings can provide directional routes which are often seen in the real traffic scenarios. It is illustrated by an example in Fig. 5.

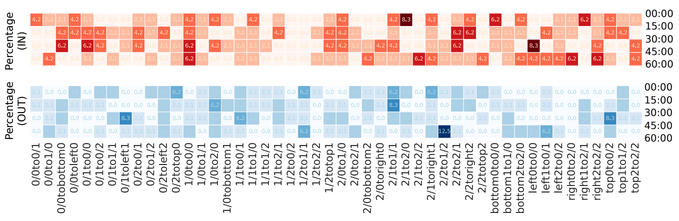


Fig. 5. An example of randomly generated vehicle routes. The x-axis is the index of each edge, and the y-axis is the periods. The number on each cell indicates the percentage of vehicles entering/leaving the network through the specific edge in that period. For example, in the last period $(45:00-60:00)$, there is about 25% of total incoming vehicles entering the traffic network from edges **1/0to1/1, 2/1to2/2, right0to2/0, right2to2/2**, and 12.5% of total incoming vehicles set the edge **2/2to1/2** as their destinations. See [41] for details of implementation.

B. Performance Comparison

We compare our method with fixed-time and vehicle-actuated controllers. In these conventional controllers, the offset of each phase is optimized using Webster Formulation according to the generated trips. The duration of a phase range from 5 to 45 seconds for the vehicle-actuated controller.

The performance is evaluated under three criteria. The first criterion is the number of arrival vehicles, which indicates that for the given period, how many vehicles have arrived at their destination through the controlled area:

$$Arr = \sum_{t=0}^{T} \|\mathrm{Veh}_{\mathrm{out}}\| \tag{12}$$

The second criterion is the average waiting time, indicating the time each vehicle has spent in halting speed in average:

$$\overline{T}_{wait} = \hat{\mathbb{E}}\left[\sum_{i=0}^{N} T_{wait}^{(\mathrm{veh}_i)}\right] \tag{13}$$

The third criterion is the time loss, which is the gap between the ideal time and actual time it spends to arrive at its destination:

$$\overline{T}_{loss} = \hat{\mathbb{E}}\left[\sum_{i=0}^{N}\left(T_{real}^{(\mathrm{veh}_i)} - T_{ideal}^{(\mathrm{veh}_i)}\right)\right] \tag{14}$$

For the criterion $\overline{T}_{wait}$ and $\overline{T}_{loss}$, we only consider vehicles that have arrived at its destination.

C. Training Speed

In this paper, the agent is built with PyTorch [42] and communicates with simulation environment via the Traci library [43]. Both simulations and deep learning process are run on a workstation with Intel Core i7-6700K CPU, 32GB RAM and 2 Nvidia GeForce Titan X GPUs.

As mentioned in Table II, the agent is trained for 50 episodes, and each episode has 3600 simulation steps. In each episode, the DRL model will be updated for 3 epochs at every 128 simulation steps. An epoch is a single pass through the entire training set, followed by testing of the verification set. That means the whole training process contains 180000 forward passes (50×3600, for traffic lights control) and 4219

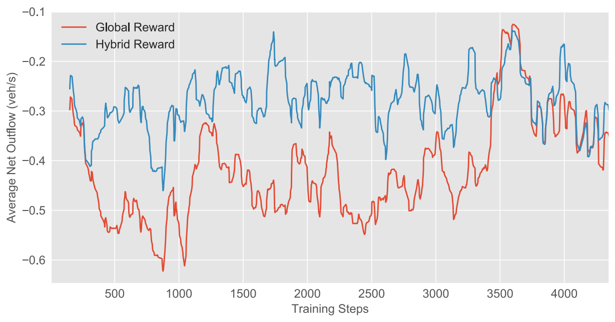


Fig. 6. Average net outflow during training process.

backward passes ($50 \times 3600/128 \times 3$, for neural network update). The total process lasts about 7 hours 30 minutes on our workstation.

As illustrated in Fig. 6, we test two models with the same structure, except one updated by only global reward (the net outflow), while another by the hybrid reward that includes both global reward and local reward (the opposite of absolute imbalance of each intersection). We can see that the one using hybrid reward achieves significantly better performance than the other one within a few episodes.

D. Performance Comparison

We compare different controllers on 360 different traffic demand settings. There are 6 different kinds of traffic demands ranging from 1800 to 36000 veh/h and 6 different randomness, i.e., $b = 10, 20, 30, 40, 50, 60$. Such setting forms 36 combinations with different traffic demand and randomness. For each combination, there are 10 simulations generated.

Fig. 7 shows the average performance of different controllers under the given traffic demands. As the experiments have shown, DRL based method is advanced than fixed-time and vehicle-actuated controllers in unsaturated and saturated cases, but its performance is getting close to the fixed-time controller once the traffic system becomes over-saturated. Among all situations, the average throughput of traffic system increases by 25.19% and 37.81% at maximum compared with fixed-time and vehicle-actuated controllers, while the average waiting time reduces by 18.68% and 28.54% at the same time. More detail results can be found in Appendix A.

To better understand the experiment results, we draw the Macroscopic Fundamental Diagrams (MFD) [44]–[46] for three typical traffic demand settings when different controllers are applied. Fig. 8 have shown that, for all these traffic conditions, the traffic accumulation (the number of vehicles in the traffic network) is the lowest under the control of DRL strategies, so is its increasing rate.

Such phenomenon prove our DRL controller could better evacuate the vehicles-in-net than traditional controllers. Unlike the vehicle-actuated controller which performs well only in the unsaturated situations, the DRL controller outperforms the fixed-time controller in all situations. It is believed that such phenomenon happened because the vehicle-actuated controller can only be implemented to an isolated intersection [47]. Due to such short-sightseeing, vehicle-actuated controller leads the traffic system to a local optimum. In contrast, our DRL controller considers the global state to make better decisions and thus achieves better performance.

V. Conclusions

In this paper, we propose an efficient DRL based approach for UTC. The simulation experiments have shown that our method performs better than tradition UTC approaches and can handle more complex environments while using fewer computing resources.

It should be pointed out that there are several things to be fathomed for this new DRL model. For example, how to transform the state into a proper format for the more general, unstructured traffic network might be one of the most urgent problems needed to be discussed. In addition, whether we should apply some other neural networks for better performance needs to be answered. We hope that this paper can provide a good start point for the following studies and expect to obtain new achievements in the near future.

References

[1] P. B. Hunt, D. I. Robertson, R. D. Bretherton, and R. I. Winton, “SCOOT-a traffic responsive method of coordinating signals,” Tech. Rep., 1981.

[2] B. Abdulhai and L. Kattan, “Reinforcement learning: Introduction to theory and potential for transport applications,” *Canadian Journal of Civil Engineering*, vol. 30, no. 6, pp. 981–991, 2003.

[3] S. El-Tantawy, B. Abdulhai, and H. Abdelgawad, “Design of reinforcement learning parameters for seamless application of adaptive traffic signal control,” *Journal of Intelligent Transportation Systems*, vol. 18, no. 3, pp. 227–245, 2014.

[4] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[5] G. A. Rummery and M. Niranjan, “On-Line Q-Learning Using Connectionist Systems,” Tech. Rep., 1994.

[6] B. Abdulhai, R. Pringle, and G. J. Karakoulas, “Reinforcement learning for true adaptive traffic signal control,” *Journal of Transportation Engineering*, vol. 129, no. 3, pp. 278–285, 2003.

[7] T. L. Thorpe, “Vehicle Traffic Light Control Using SARSA,” Online]. Available: citeseer.ist.psu.edu/thorpe97vehicle.html, Tech. Rep., 1997.

[8] S. Richter, D. Aberdeen, and J. Yu, “Natural actor-critic for road traffic optimisation,” in *Advances in Neural Information Processing Systems*, 2007, pp. 1169–1176.

[9] L. Shoufeng, L. Ximin, and D. Shiqiang, “Q-Learning for adaptive traffic signal control based on delay minimization strategy,” in *Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference On*. IEEE, 2008, pp. 687–691.

[10] A. ad Salkham, R. Cunningham, A. Garg, and V. Cahill, “A collaborative reinforcement learning approach to urban traffic control optimization,” in *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology-Volume 02*. IEEE Computer Society, 2008, pp. 560–566.

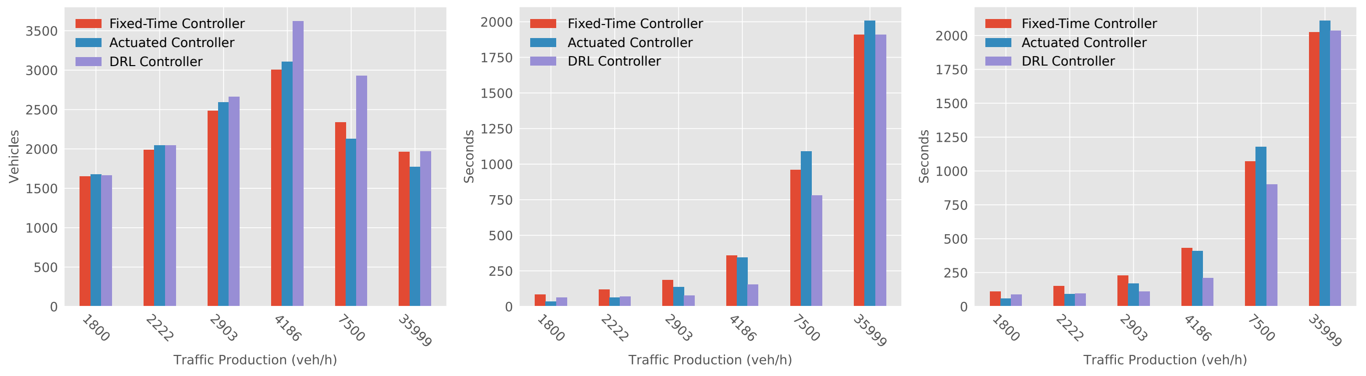
[11] M. A. Wiering, “Multi-agent reinforcement learning for traffic light control,” in *Machine Learning: Proceedings of the Seventeenth International Conference (ICML’2000)*, 2000, pp. 1151–1158.

[12] E. Camponogara and W. Kraus, “Distributed learning agents in urban traffic control,” in *Portuguese Conference on Artificial Intelligence*. Springer, 2003, pp. 324–335.

[13] D. de Oliveira, A. L. Bazzan, B. C. da Silva, E. W. Basso, L. Nunes, R. Rossetti, E. de Oliveira, R. da Silva, and L. Lamb, “Reinforcement Learning based Control of Traffic Lights in Non-stationary Environments: A Case Study in a Microscopic Simulator.” in *EUMAS*, 2006.

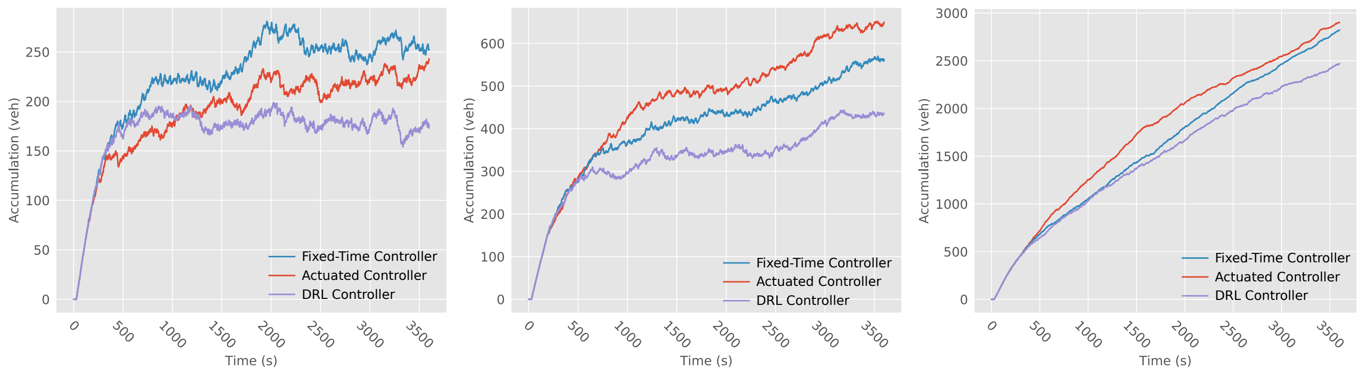
[14] Bo Chen and H. H. Cheng, “A Review of the Applications of Agent Technology in Traffic and Transportation Systems,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 2, pp. 485–497, Jun. 2010. [Online]. Available: http://ieeexplore.ieee.org/document/5462881/

[15] P. G. Balaji, X. German, and D. Srinivasan, “Urban traffic signal control using reinforcement learning agents,” *IET Intelligent Transport Systems*, vol. 4, no. 3, pp. 177–188, 2010.



(a) Arrived vehicles (b) Waiting time (c) Time losses

Fig. 7. Average performance of different controllers.



(a) Unsaturated situation(2400 veh/h) (b) Saturated situation(3600 veh/h) (c) Over-saturated situation(7200 veh/h)

Fig. 8. Macroscopic fundamental diagrams for different controllers. The x-axis is the simulation steps, and the y-axis indicates the number of vehicles in the traffic grid.

[16] I. Arel, C. Liu, T. Urbanik, and A. G. Kohls, "Reinforcement learning-based multi-agent system for network traffic signal control," *IET Intelligent Transport Systems*, vol. 4, no. 2, pp. 128–135, 2010.

[17] F. Caselli, A. Bonfietti, and M. Milano, "Swarm-Based Controller for Traffic Lights Management," in *AI*IA 2015 Advances in Artificial Intelligence*, M. Gavanelli, E. Lamma, and F. Riguzzi, Eds. Cham: Springer International Publishing, 2015, vol. 9336, pp. 17–30. [Online]. Available: http://link.springer.com/10.1007/978-3-319-24309-2_2

[18] L. Li and D. Wen, "Parallel Systems for Traffic Control: A Rethinking," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1179–1182, Apr. 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7328734/

[19] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. [Online]. Available: http://www.nature.com/nature/journal/v521/n7553/abs/nature14539.html

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and others, "Human-Level Control through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[21] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. Riedmiller, and D. Silver, "Emergence of Locomotion Behaviours in Rich Environments," *arXiv:1707.02286 [cs]*, Jul. 2017. [Online]. Available: http://arxiv.org/abs/1707.02286

[22] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017. [Online]. Available: https://www.nature.com/nature/journal/v550/n7676/full/nature24270.html

[23] L. Li, Y. Lv, and F. Y. Wang, "Traffic signal timing via deep reinforcement learning," *IEEE/CAA Journal of Automatica Sinica*, vol. 3, no. 3, pp. 247–254, Jul. 2016.

[24] E. Van der Pol and F. A. Oliehoek, "Coordinated deep reinforcement learners for traffic light control," in *In Proceedings of NIPS*, vol. 16, 2016.

[25] W. Genders and S. Razavi, "Using a Deep Reinforcement Learning Agent for Traffic Signal Control," *arXiv:1611.01142 [cs]*, Nov. 2016. [Online]. Available: http://arxiv.org/abs/1611.01142

[26] N. Casas, "Deep Reinforcement Learning for Urban Traffic Light Control," 2017.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[28] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press Cambridge, 1998, vol. 1, no. 1. [Online]. Available: http://www.cell.com/trends/cognitive-sciences/pdf/S1364-6613(99)01331-5.pdf

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv:1707.06347 [cs]*, Jul. 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[30] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[31] T. Degris, P. M. Pilarski, and R. S. Sutton, "Model-free reinforcement learning with continuous action in practice," in *American Control Conference (ACC), 2012*. IEEE, 2012, pp. 2177–2182.

[32] S. Lin, Q.-J. Kong, and Q. Huang, "A Simulation Analysis on the Existence of Network Traffic Flow Equilibria," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 4, pp. 1706–1713, Aug. 2014. [Online]. Available: http://ieeexplore.ieee.org/document/6744590/

[33] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, vol. 99, 1999, pp. 278–287.

[34] H. Van Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes, and J. Tsang, "Hybrid reward architecture for reinforcement learning," in

Advances in Neural Information Processing Systems, 2017, pp. 5392–5402.

[35] W.-H. Lin, H. K. Lo, and L. Xiao, “A Quasi-Dynamic Robust Control Scheme for Signalized Intersections,” *Journal of Intelligent Transportation Systems*, vol. 15, no. 4, pp. 223–233, Oct. 2011. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/15472450.2011.620490

[36] Y. Tong, L. Zhao, L. Li, and Y. Zhang, “Stochastic programming model for oversaturated intersection signal timing,” *Transportation Research Part C: Emerging Technologies*, vol. 58, pp. 474–486, Sep. 2015. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0968090X15000273

[37] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” *arXiv preprint arXiv:1602.01783*, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[38] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv:1412.6980 [cs]*, Dec. 2014. [Online]. Available: https://arxiv.org/abs/1412.6980

[39] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, “High-Dimensional Continuous Control Using Generalized Advantage Estimation,” Jun. 2015. [Online]. Available: http://sci-hub.cc/http://arxiv.org/abs/1506.02438

[40] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, “Recent development and applications of SUMO-Simulation of Urban MObility,” *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, 2012.

[41] “Tools/Trip - Customized Weights - SUMO User Documentation.” [Online]. Available: http://sumo.dlr.de/wiki/Tools/Trip#Customized_Weights

[42] A. Paszke, S. Gross, S. Chintala, and G. Chanan, “PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration,” 2017. [Online]. Available: http://pytorch.org/

[43] A. Wegener, M. Piórkowski, M. Raya, H. Hellbrück, S. Fischer, and J.-P. Hubaux, “TraCI: An interface for coupling road traffic and network simulators,” in *Proceedings of the 11th Communications and Networking Simulation Symposium*. ACM, 2008, pp. 155–163. [Online]. Available: http://dl.acm.org/citation.cfm?id=1400740

[44] N. Geroliminis, “Increasing mobility in cities by controlling overcrowding,” PhD Thesis, UC Berkeley: Institute of Transportation Studies (UCB), 2007.

[45] C. F. Daganzo and N. Geroliminis, “An analytical approximation for the macroscopic fundamental diagram of urban traffic,” *Transportation Research Part B: Methodological*, vol. 42, no. 9, pp. 771–781, 2008.

[46] N. Geroliminis and C. F. Daganzo, “Existence of urban-scale macroscopic fundamental diagrams: Some experimental findings,” *Transportation Research Part B: Methodological*, vol. 42, no. 9, pp. 759–770, 2008.

[47] Peng Jing, Hao Huang, and Long Chen, “An Adaptive Traffic Signal Control in a Connected Vehicle Environment: A Systematic Review,” *Information*, vol. 8, no. 3, p. 101, Aug. 2017. [Online]. Available: http://www.mdpi.com/2078-2489/8/3/101

Appendix A
Average performance in Different Situations

TABLE III
Arrived vehicles for different traffic production

Traffic Demand	Fixed-Time Controller	Actuated Controller	RL Controller
1800	1648.65	**1677.93**	1661.65
2222	1988.47	2042.02	**2045.85**
2903	2482.17	2589.30	**2664.23**
4186	3004.28	3107.12	**3623.52**
7500	2339.50	2125.35	**2928.90**
36000	1964.62	1771.48	**1968.42**
Average	2237.95	2218.87	**2482.09**

TABLE IV
Waiting time for different traffic production

Traffic Demand	Fixed-Time Controller	Actuated Controller	RL Controller
1800	83.86	**35.22**	63.62
2222	118.97	**63.71**	67.54
2903	187.12	135.40	**77.76**
4186	357.60	343.59	**155.24**
7500	957.42	1089.52	**778.53**
36000	1909.33	2006.02	**1908.98**
Average	602.38	612.24	**508.61**

TABLE V
Time loss for different traffic production

Traffic Demand	Fixed-Time Controller	Actuated Controller	RL Controller
1800	110.54	**58.40**	88.96
2222	150.62	**89.98**	94.37
2903	229.47	170.17	**108.04**
4186	432.99	408.04	**208.11**
7500	1071.85	1178.03	**901.13**
36000	**2024.27**	2107.58	2033.29
Average	669.96	668.70	**572.32**

TABLE VI
Arrived vehicles for different randomness

Randomness b	Fixed-Time Controller	Actuated Controller	RL Controller
10	2284.63	2247.12	**2535.43**
20	2237.17	2212.35	**2477.45**
30	2206.18	2197.73	**2468.25**
40	2234.57	2226.12	**2460.35**
50	2258.28	2237.42	**2499.20**
60	2206.85	2192.47	**2451.88**
Average	2237.95	2218.87	**2482.09**

TABLE VII
WAITING TIME FOR DIFFERENT RANDOMNESS

Randomness b	Fixed-Time Controller	Actuated Controller	RL Controller
10	590.86	608.23	**499.11**
20	606.57	621.20	**515.40**
30	612.76	618.29	**512.77**
40	610.04	617.10	**519.57**
50	589.51	602.18	**498.75**
60	604.54	606.45	**506.07**
Average	602.38	612.24	**508.61**

TABLE VIII
TIME LOSS FOR DIFFERENT RANDOMNESS

Randomness b	Fixed-Time Controller	Actuated Controller	RL Controller
10	659.34	665.86	**564.34**
20	674.15	676.97	**577.78**
30	681.18	675.09	**576.14**
40	674.72	672.30	**580.53**
50	657.54	659.99	**565.13**
60	672.80	661.99	**569.98**
Average	669.96	668.70	**572.32**