Knowledge-Based Systems 241 (2022) 108304

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Distributed agent-based deep reinforcement learning for large scale traffic signal control

Qiang Wu [a,b], Jianqing Wu [c], Jun Shen [d,*], Bo Du [e], Akbar Telikani [d], Mahdi Fahmideh [f], Chao Liang [g]

[a] Institute of Fundamental and Frontier Sciences, University of Electronic Science and Technology of China, Chendu, China
[b] Yangtze Delta Region Institute (Huzhou), University of Electronic Science and Technology of China, Huzhou, China
[c] School of Information Engineering, Jiangxi University of Science and Technology, Ganzhou, China
[d] School of Computing and Information Technology, University of Wollongong, Wollongong, Australia
[e] SMART Infrastructure Facility, University of Wollongong, Wollongong, Australia
[f] School of Business at University of Southern Queensland, Queensland, Australia
[g] School of Economics and Management, Southwest Jiaotong University, Chengdu, China

ARTICLE INFO

Article history:
Received 3 November 2021
Received in revised form 11 January 2022
Accepted 24 January 2022
Available online 3 February 2022

Keywords:
Traffic signal control
Reinforcement learning
Nash Equilibrium
Nash-A3C
Distributed computing architecture

ABSTRACT

Traffic signal control (TSC) is an established yet challenging engineering solution that alleviates traffic congestion by coordinating vehicles' movements at road intersections. Theoretically, reinforcement learning (RL) is a promising method for adaptive TSC in complex urban traffic networks. However, current TSC systems still rely heavily on simplified rule-based methods in practice. In this paper, we propose: (1) two game theory-aided RL algorithms leveraging Nash Equilibrium and RL, namely Nash Advantage Actor–Critic (Nash-A2C) and Nash Asynchronous Advantage Actor–Critic (Nash-A3C); (2) a distributed computing Internet of Things (IoT) architecture for traffic simulation, which is more suitable for distributed TSC methods like the Nash-A3C deployment in its fog layer. We apply both methods in our computing architecture and obtain better performance than benchmark TSC methods by 22.1% and 9.7% reduction of congestion time and network delay, respectively.

1. Introduction

For many decades, urban traffic congestion has become a worldwide problem with adverse influence on economy and environment. For example, within just one single year, traffic jams made the economic loss up to 121 million US dollars and produced 25,396 tons of carbon dioxide in the United States [1]. The energy consumption caused by traffic congestion and the resulting greenhouse gases such as carbon dioxide will increase the greenhouse effect. To reduce excessive pollution emissions caused by traffic congestion for improving climate through technological innovation.

Improving traffic signal control (TSC) is a promising and cost-efficient way to relieve traffic congestion and maximize traffic flow. Traditional adaptive TSC models design hand-crafted rules by analyzing real traffic data to replace older naive methods which replied on fixed time controllers. However, they are not sufficiently dynamic in adapting to the current traffic status and solely use historical data for time adjustment [2].

Reinforcement learning (RL) is a new research direction in the sense dynamically adjusting traffic lights, normally based on real-time traffic flow. RL has the characteristics of being "data-driven, self-learning, and no model", which is a promising method for complex urban traffic networks and environments. However, the number of states increases in some very complex urban traffic scenarios, leading to the exponential complexity if applying the traditional RL because much longer time is needed for exploring all state–action pairs [3,4]. Moreover, more storage resources are required to store transiently learned Q-values [5]. Deep RL (DRL) is more suitable than traditional RL for the development of TSC systems, which have been significantly boosted by leveraging other deep learning methods. In DRL process, the agent gets rewards and tries to maximize the expected rewards via deep neural networks (DNNs) [6] in the interaction process between traffic signals and traffic conditions.

Consider a standard four-approach intersection where each approach has three "left-turn, through and right-turn" traffic directions. According to the traffic rules, it will be eight different traffic movements (phases). With more complex scenarios, learning the optimal TSC policy becomes much more difficult for an RL

* Corresponding author.
E-mail addresses: qiang.wu@uestc.edu.cn (Q. Wu), jianqing.wu@jxust.edu.cn (J. Wu), jshen@uow.edu.au (J. Shen), bdu@uow.edu.au (B. Du), at952@uowmail.edu.au (A. Telikani), mahdi.fahmideh@usq.edu.au (M. Fahmideh), liangchao@swjtu.edu.cn (C. Liang).

https://doi.org/10.1016/j.knosys.2022.108304
0950-7051/

agent. Assume the vehicle capacity of a lane is n, the exploration space will increase to $N \times 8 \times n^8$ considering N intersections.

Internet of Things (IoT) now is used to facilitate traffic management in many smart cities. It allows different components in the whole transportation systems, such as vehicles, lights, and cameras, to connect and transfer information between themselves. Regarding TSC, traffic lights are connected to each other to optimize signal phase at intersections. Integration of RL into IoT can facilitate decision making in TSC and reduce the latency. Prior studies [7–9] suggest that IoT platforms relying on computing technologies cloud, fog, and edge can empower TSC performance for improved data processing, analysis, and RL training.

However, often the goal for alleviating traffic congestion nowadays is not achieved by optimizing and controlling the traffic signal of just one intersection independently. In contrary, it requires multiple intersections to collaborate together, extending from local optimization to global optimization, and finding multiple intersections' global optimization control towards an optimal or sub-optimal solution for a whole transport network oriented TSC. Multi-agent reinforcement learning (MARL) has been promising to handle TSC at a large scale. Existing MARL methods have tried to achieve global traffic signal optimization control by distributing the RL algorithm to local agents [10], which have obtained some good results. However, several shortcomings remain unsolved:

(1) Current TSC systems still rely heavily on simplified rule-based methods. While very few adaptive TSC methods based on RL are really deployed in actual use.
(2) How to reduce the problem space and explore different scenarios more efficiently for the RL algorithm?
(3) The existing traffic simulation environment [11] cannot simulate scenarios close to the real urban traffic environment, hence the lab-simulated distributed algorithms are actually not readily for deployment.

Further, the video data collected by local devices is relatively large and often results in significant network latency [12]. Meanwhile, existing traffic simulation software rarely considers network latency and distributed algorithms deployment. In order to tackle the above challenges, this paper makes the following contributions:

(1) We proposed distributed agent-based algorithms leveraging Nash Equilibrium and RL, namely Nash Advantage Actor–Critic (Nash-A2C) and Nash Asynchronous Advantage Actor–Critic (Nash-A3C), where such a combination was not reported in literature so far;
(2) We built a distributed computing Internet-of-Things (IoT) architecture for traffic simulation, which is more suitable for distributed TSC methods like the Nash-A3C deployment. In this framework, data collected in the edge layer using IoT devices are analyzed by distributed DRL on the worker fog nodes. The master fog node aggregates the results from worker nodes and transfers to the cloud platform to make a global decision and broadcasts the actions to the traffic lights.
(3) Experimental results demonstrated that our Nash-A3C has consistent and outstanding performance on reducing average waiting time at the global system level. This demonstrates the two-folds superiority of our methods as mentioned above.

The remainder of this paper is organized as follows: Section 2 introduces the related works. Section 3 describes Nash Equilibrium in game theory and the definition of the TSC problem. Section 4 details the Nash-A3C algorithm and proposes a distributed computing architecture using fog nodes. Section 5 reports the experimental results. Section 6 summarizes the paper.

2. Related works

The urban TSC problem has been expanded from the optimization of one intersection to multiple intersections which is usually viewed as a typical multi-agent system. The traffic signal controller at each intersection is regarded as an agent. Each agent controls the traffic signal and needs online cooperation. The overall aim of the multi-agent system is to reduce traffic jam. Deployment of edge computing and IoT devices facilitates traffic control. For example, the TSC algorithm of an agent at an intersection needs to be deployed to the edge device in a "lightweight" mode, considering the limitations of the capabilities of network transmission bandwidth and underlying cloud computing. This section investigates literature review on both RL- and IoT-aided TSC.

2.1. Reinforcement learning-aided traffic signal control

TSC is regarded as an "agent" with decision-making ability at intersections. By observing the real-time traffic flow, the current traffic status S_t and the reward R_t are obtained. According to the current status, the agent selects and executes the corresponding actions (change or keep lights). Then, the agent observes the effect of the action on the intersection traffic to obtain the new traffic state S_{t+1} and the new reward R_{t+1}. The agent evaluates the action, and optimizes strategies until converging to an optimal "state and action".

Using RL in TSC, the intermediate actions and rewards can be learnt regarding long-term traffic conditions in real time (e.g., the waiting time and the number of vehicles in a queue). RL models include three types of representations: (1) state refers to the decision making factors, such as the speed, the queue length, and the position of vehicles at a lane. (2) Actions can be, for instance, traffic phase splits and traffic phases. (3) Examples of reward indicating the appropriateness of state–action pairs are travel delay and travel time of vehicles [5]. This section classifies RL-based TSC methods into two categories: single-agent RL and multi-agent RL (Table 1). The second column of the table shows that the TSC model is either single-agent or multi-agent. The third column mentions the RL algorithm used for modeling. The fourth, fifth, and sixth columns demonstrate states, actions, and rewards defined in RL model, respectively. From the table, most works focused on queue length that indicates the number of vehicles in the traffic lane green phase and traffic phase are two widely-used actions, meaning the time for green lines is adjusted and the directions that vehicles are allowed to pass the intersection at the same time, respectively. Waiting time and traffic delay are two widely-defined rewards in RL-enables TSC frameworks. Traffic delay means the additional time a vehicle spend in traffic congestion, while, waiting time is the time a vehicle waits in the queue at an intersection. Fig. 1 depicts the differences between two paradigms. As, in TSC, drivers' behavior influences the state transition between states, RL-based TSC are often considered as model-free methods.

2.1.1. Single-agent reinforcement learning

One of the first contributions of RL to TSC by [13] is based on Q-learning. In [17], discrete traffic state encoding is used and a CNN model trained using Q-learning with experience replay. Wan and Hwang [19] developed a value-based DRL model which was modified from the Deep Q-network(DQN). Bellman's equation is used to express optimal action-value function.

In many practical decision making problems, the traffic signal controller is constructed by DNNs for TSC. Li et al. [18] used stacked auto-encoders to learn the Q-function of RL for TSC. In this way, the inputs of agents can be compressed and stored in an efficient manner. In other work [3], convolutional neural network (CNN) is employed. Wei et al. [2] incorporated both reward and policy interpretation in designing DQN-embedded TSC system.

Table 1
Typical works on RL-based TSC methods.

Ref./Year	Scale	Model	State	Action	Reward
[13]/2003	Single	Q-learning	Queue length	Binary phase	Fixed penalty
[14]/2010	Multiple	Q-learning	Arriving vehicles Queue length Traffic delay	Green phases	Traffic delay
[15]/2014	Multiple	Q-learning	Arriving vehicles Queue length Cumulative delay	Green phases	Traffic delay
[16]/2014	Multiple	Q-learning	Queue length	Traffic phases	Stage costs
[17]/2016	Single	Q-learning	Queue length	Signal phase	Vehicle delay
[18]/2016	Single	DQN	Queue length Waiting time Average speed Phase duration	Right ways	Traffic delay Queue length
[19]/2018	Single	DQN	Turn conditions	Signal phase	System delay
[3]/2018	Single	DQN	Vehicles matrix	Phase duration	Waiting time
[2]/2018	Single	DQN	Queue length Waiting time Vehicle number	Traffic phases	Queue length Waiting time Vehicle number
[20]/2019	Multiple	A2C-RNN	Cumulative delay	Traffic phases	Waiting time
[5]/2020	Multiple	DQN	Vehicle speed	Traffic phases	Travel delay
This paper	Multiple	NashA2C NashA3C	Vehicle queue	Traffic phases	Waiting time

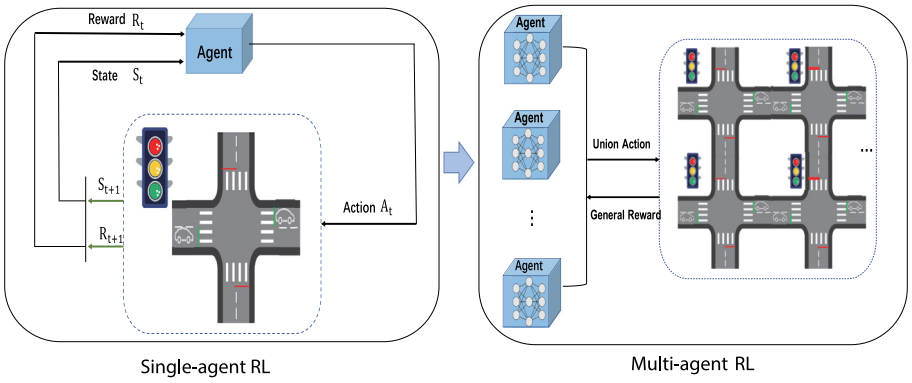

Fig. 1. An illustration of RL methods in TSC.

2.1.2. Multi-agent reinforcement learning

As agents may apply actions to the environment for whole rewards, we define a multi-agent reinforcement learning (MARL) environment as a tuple $[(X_1, A_1), (X_2, A_2) \ldots, (X_m, A_m)]$ where X_m is any given agent and A_m is any given action. The new state of the environment is the result of a set of joined actions defined by $A_1, A_2, \ldots, A_m$. In other words, the complexity of MARL scenarios increases with the number of agents in the environment. The urban TSC can be viewed as a typical multi-agent system. Deep MARL is mainly concerned with the cooperative and coordinated control actions of multiple states of intersections, which extend the DRL algorithm to multi-agents. In MARL, each agent could estimate the action probability model of other agents without real-time computing, but it was still difficult to update the estimation model in a dynamic environment. MARL can be applied in TSC in three ways: (1) traffic networks are partitioned into different regions and a RL agent is considered to control each region; (2) transfer learning strategy is allowed; and (3) each intersection is controlled by one RL agent.

Prabuchandran et al. [16] considered each traffic signal as an independent agent and round-robin strategy was used for deciding about the signal duration of each agent's phases through multi-agent Q-learning. Q-factors of each agent are updated based on the cost feedback signal received from its neighboring agents. Chu et al. [20] formulated Independent Q-learning (IQL) on A2C under limited communication. Local observations are first included in the state. Then, each local agent learns its own policy independently and enables modeling of other agents. The global Q-function is distributed over the local agents. Rasheed et al. [5] investigated multi-agent DQN in an urban area in Malaysia. El-Tantawy and Abdulhai [14] proposed a three-states' definition using Q-learning based on the number of arriving vehicles to the green direction, the number of queued vehicles in the red direction, and the queue length. An extension of this work with different on-policy and off-policy RL algorithms was presented in [15]. Immediate delay, cumulative delay, queue length and the number of stops were four reward functions.

2.2. Traffic simulation environment

Urban traffic simulation systems can usually be expressed as inclusive of most traffic facilities such as roads, vehicles, and traffic lights (as illustrated in Fig. 2), which can be categorized into traditional system and open-source system, such as SUMO [21], CityFLow [11].

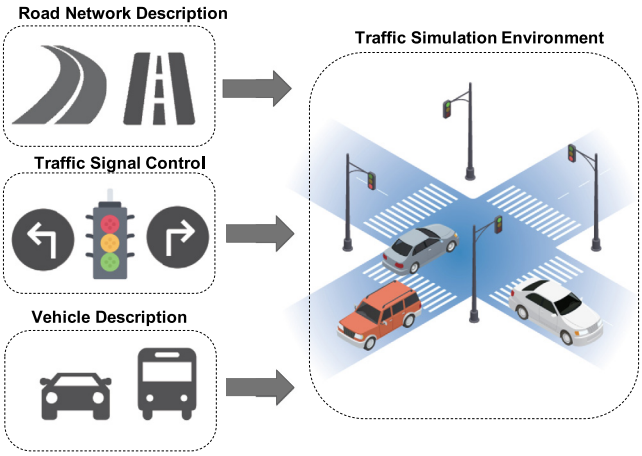

Fig. 2. An illustration of an urban traffic simulation system.

Table 2
Typical works on IoT-aided TSC methods.

Ref./Year	Edge platform	Fog platform	Cloud platform
[23]/2018	✗	✗	√
[24]/2018	✗	√	✗
[25]/2019	✗	√	✗
[1]/2019	√	√	√
[26]/2019	√	✗	✗
[7]/2020	√	✗	✗
[9]/2021	✗	✗	√
[27]/2021	✗	√	✗
[8]/2021	✗	√	✗
This paper	√	√	√

2.3. IoT computing-aided traffic signal control

In the IoT paradigm [22], a large number of objects are connected to the network by using telecommunication technologies. Cloud computing, fog computing, and edge computing are key parts of an IoT infrastructure. Table 2 present works on IoT computing for TSC.

2.3.1. Cloud computing

Cloud computing is a paradigm facilitating ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources with minimum service provider's interaction. Giant companies, such as Amazon and Microsoft, have become major cloud computing providers. A cooperative multi-agent DRL mechanism based on edge computing architecture was developed in [9]. The global actor–critic learning tasks are decomposed local actor–critic sub-problems for each intersection. The input of the network is image frames from traffic states. Parallel training tasks are first deployed on the cloud nodes and trained multi-agent models are used in the cloud nodes for real-time decision making. A distributed genetic algorithm-based TSC based on cloud computing was proposed in [23]. Traffic flow data is processed using computing nodes on the cloud platform. Besides, a migration strategy was designed in which some worse chromosomes at a node is replaced with better ones at other node based on a probability.

2.3.2. Fog computing

Distant cloud computing centers may cause significant travel delay and network congestion. In addition, edge computing does not have sufficient computing resources on edge devices. To address these challenges, fog computing has been introduced. It is not necessary to send everything to the cloud, and fog nodes are capable to analysis data and make decisions.

FogFly [24] distributes fog nodes at TSC units and regions to gather and analyze traffic data from vehicles. Worker nodes embedded at each intersection collect data from vehicles using short-range connections (e.g., WiFi and RFID) and send data to a master node for statistical analysis of traffic conditions. In [25], genetic algorithm was employed within a fog computing platform to optimize the phase timing of an intersection. In this architecture, fog nodes are responsible for optimizing the timing task is a real-time manner and then, regional optimized timing is transferred to a centralized cloud. A fog computing-based TSC using fuzzy inference engine was proposed by Gamel et al. [27]. This method aimed to reduce the average waiting time of the vehicles at the intersections. Vehicle to Infrastructure (V2I) protocol was applied for direct interaction between vehicles and the infrastructure (e.g., GPS sensors and traffic light signals). Fuzzy inference engine calculates the waiting time for each traffic lane based on the number, locations, and sizes of waiting vehicles.

Wu et al. [1] employed RL in the fog environment to decrease queue length and stopping times. This framework consists of three components of connected vehicles, fog computing, and cloud computing. The data (vehicle speed and position) collected by vehicles in each intersection is sent to the fog nodes to be analyzed by RL. The cloud component receives intersections' conditions from fog nodes to control traffic lights between intersections. A multi-agent distributed TSC based on the fog computing and Q-learning-based RL was developed in [8]. Also, puzzle difficulty of the Hash collision was considered in the fog computing platform to deal with security problems.

2.3.3. Edge computing

With the development of cloud computing, edge computing emerges as a novel form of computing technology, which shifts from centralized to more decentralized form by bringing computation and data nodes together. Compared with conventional cloud computing, it provides shorter response time and better reliability. To save bandwidth and reduce latency, data is processed at the edge. Thus, mobile devices of certain users can complete partial workload at the edge of the network. Similarly, in intelligent transportation systems, edge devices can be deployed on roadsides and vehicles for better communications and control between connected moving objects. Wu et al. [7] deployed MARL in edge computing architecture. In this framework, agents share the learned strategies with others to obtain a global optimization for TSC. Zhou et al. [26] employed edge computing nodes to collect traffic data. RL algorithm is applied at each edge node to tune timing of TSC at each intersection.

2.4. Game theory for TSC

Game theory initially focused on winning and losing problems in human games such as chess, bridge, and gambling based on predicting behavior of individuals in the game. Game theory refers to the use of each other's strategies to change their confrontation strategies in a fair game to achieve the goal of winning. Equilibrium existence theorem and Nash equilibrium are widely used in many disciplines.

Villalobos et al. [28] model signalized intersections as finite controlled Markov chain to optimize the congestion into an intersection, which is seen as noncooperative game where each player try to minimize its queue. Khanjary et al. [29] perform game theory to reach the Nash Equilibrium point without motivation to change any policy for TSC, improving the traffic significantly. Elhenawy et al. [30] develop chicken-game model that assumes vehicles can communicate with a central agent at the intersection

to provide their instantaneous speeds and locations. The algorithm assumes that vehicles obey the Nash equilibrium solution of the game. Bui et al. [31] propose two game models for TSC optimization using algorithmic game theory. TSC agents are able to make real-time decisions based on the density of vehicles in two phases of North–South and East–West.

3. Preliminaries

3.1. Nash Equilibrium in game theory

In game theory, Nash Equilibrium refers to a non-cooperative game with two or more participants, assuming that each participant knows the strategy of the other participants. There is no conceptual solution on when participants can change their strategy to benefit themselves. Any static game has at least one Nash Equilibrium whereas many MARL methods try to converge to a Nash Equilibrium.

In short, the action a ($a \in A$) of each agent is under the policy of π^{*a}, and the policy of other agents is π^{*-a}. The objective is to get the most significant reward R.

$$\forall a : R^a\left(\pi^{*a}, \pi^{*-a}\right) \geq R^a\left(\pi^a, \pi^{*-a}\right), \forall \pi^a \tag{1}$$

Let us first define a tuple of stochastic games: $G = \langle X, U, R, A, N, \gamma \rangle$ (a participator in the game corresponding to an agent in RL), where X is the environment state space; U is the joint action from agents. For example, $Agent^i$ in state $x \in X$; A is the action space of one agent $a \in A$; N is the number of agents; R is the target reward function. Joint action is $u^a \in U$; reward function is $r_i(s, u, a)$. We use $u_{i,t} \in U$ to represent the action of $Agent^i$ at time t, and $u_{-i,t}$ to represent the action of $Agent^{-i}$ (all agents expect $Agent^i$). We assume that the game process is a Markov decision process (MDP). The reward function at time t is $r_i(x_t, u_{i,t}, u_{-i,t})$. $Agent^i$ selects an action policy based on deterministic Markov $\pi_i(x)(x \in X)$. The goal of $Agent^i$ is to choose a policy π and to maximize the objective function R_i.

3.2. Problem definition

In the multi-agent TSC problem, we define: each agent who controls the change (duration) of the traffic signal is $Agent^i(i \in N)$; π_i is all acceptable traffic light duration control policies of $Agent^i$; R_i is the reward of the degree of congestion at the intersection of $Agent^i$ and other agent ($Agent^{-i}$) in the environment, which can be calculated according to the specific indicators of vehicle queue length.

$$\mathcal{R}_i\left(x; \pi_i, \pi_{-i}\right) = \mathbb{E}\left[\sum_{t=0}^{-\infty} -\gamma_i^t r_i\left(x_t, \pi_{i,t}, \pi_{-i,t}\right)\right] \tag{2}$$

In the above formula, π_i^* is the policy of $Agent^i$ under other agents $Agent^{-i}$ action selection policies π_{-i}^*, $\gamma \in [0, 1]$. The result of the objective function is based on the policy selection of each agent. Each agent intelligently controls its own action policy, but needs to choose its own actions responding to the behavior of other agents. $Agent^i$ seeks to optimize the behavior of the objective function when the behavior of other agents is stable. Finally, all agent's policies reach the Nash equilibrium state.

Based on the Nash Equilibrium in game theory and deep MARL, we need to find a set of parameters θ of DNN, so that the estimated Q function Q^θ satisfies the Nash–Bellman formula as much as possible [32].

4. Methodology

In this section, we propose Nash Equilibrium based actor–critic algorithms, named *Nash Advantage Actor–Critic (Nash-A2C)* and *Nash Asynchronous Advantage Actor–Critic (Nash-A3C)* inspired by the DRL model of Actor–Critic [33] algorithms and the equilibrium optimizer [34]. Moreover, we develop a distributed computing IoT architecture and optimize the traffic simulation environment in this section inspired by the service-oriented architecture [35] and RL framework [36].

4.1. MARL algorithms

4.1.1. Actor–critic

Algorithms of DRL are divided into two types: value-based and policy-based. The reward function of the policy-based algorithms is not stable in estimating the actual interaction with the environment.

The actor–critic model combines the advantages of policy gradient and Q-learning, using Q-learning to estimate the reward function instead of estimating it in multiple interactions with the environment, as seen in below formula. Hence, the Actor–Critic can make the reward process more stable.

$$E\left[G_t^n\right] = Q^{\pi_\theta}\left(s_t^n, a_t^n\right) \tag{3}$$

A baseline is often added to Q value to make the feedback positive or negative. The baseline here usually uses the state value function as the advantage function, so that the value function based on advantage becomes: $r_t^n + V^\pi(s_{t+1})^n - V^\pi(s_{t1}^n)$. Thus, the function based on the policy gradient is:

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(r_t^n + V^\pi\left(s_{t+1}^n\right) - V^\pi\left(s_t^n\right)\right)\nabla \log p_\theta\left(a_t^n | s_t^n\right) \tag{4}$$

The policy network $\pi(\theta)$ consists of the parameters of the actor, the value function network $V(\theta)$ includes the parameters of the critic. Actor and critic are estimated in each interaction with the environment to make the reward estimation process more stable. This is the basic theoretical explanation of the advantage Actor–Critic (A2C) algorithm.

Asynchronous advantage actor–critic (A3C) [37] is based on A2C, and employing the concept of parallel universe, which simulates different environments, performs A2C with different environments and then centralizes parameters to a cloud center for updating parameters. In this sense, it is an algorithm structure that inherently adapts to cloud computing.

4.1.2. Nash-A2C

Computation of the advantage function in DRL might have positive or negative reward results, making the training process more efficient. Inspired by A2C algorithm, we also retain the advantage function in the design of MARL algorithm.

We define the Q function of $Agent^i$ as $\left(\hat{Q}_i^\theta(x; \boldsymbol{u})\right)_{i \in N}$; the estimate value function of $Agent^i$ is $\left(\hat{V}_i^\theta(x; \boldsymbol{u})\right)_{i \in N}$, where $\boldsymbol{u}$ is the union action; and $\boldsymbol{x}$ is the union state.

$$\hat{\boldsymbol{A}}^\theta(x; \boldsymbol{u}) = \hat{Q}^\theta(x; \boldsymbol{u}) - \hat{\boldsymbol{V}}^\theta(x; \boldsymbol{u}) \tag{5}$$

We apply the actor–critic model as backbone for our algorithm. As discussed above, the actor is a policy function and the critic is a value function. We divide the parameter set θ into the value function parameter set and the policy function parameter set $\theta = (\theta_V, \theta_A)$, where θ_V represents the model of the value function $\hat{V}^{\theta_V}$; θ_A represents the parameters of the agent (participant) action selection policy $\hat{\pi}^{\theta_A}$. The goal of the algorithm is to minimize the loss of the sampled samples and the Nash–Bellman equation $\mathcal{L}_m(\theta)$:

$$\frac{1}{M}\sum_{m=1}^{M}\left\|\hat{\boldsymbol{V}}^\theta(x_m) + \hat{\boldsymbol{A}}^\theta(x_m; \boldsymbol{u}_m) - \boldsymbol{r}(x_m; \boldsymbol{u}_m) - \gamma_i \hat{\boldsymbol{V}}^\theta\left(x_m'\right)\right\|^2 \tag{6}$$

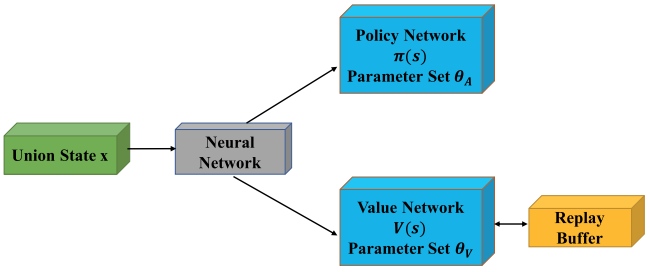

Fig. 3. The structure of Nash-A2C.

To simplify the above expression, we define y_m:

$$\left\| \hat{\boldsymbol{V}}^{\theta_V}(x_m) + \hat{\boldsymbol{A}}^{\theta_A}(x_m; \boldsymbol{u}_m) - \boldsymbol{r}(x_m; \boldsymbol{u}_m) - \gamma_i \hat{\boldsymbol{V}}^{\theta_V}\left(x'_m\right) \right\|^2 \tag{7}$$

In short, $y_m = \hat{\mathcal{L}}(y, \theta_V, \theta_A)$.

Inspired by the deep Q-networks (DQN [38]) algorithm, our method also introduces a memory buffer (replay buffer) to store triples $y_t = (x_{t-1}, u, x_t)$, which represents the previous state of the environment x_{t-1}; the union operation performed u; the next state of the environment x_t; and the reward y_t that passes through this state (We use vehicles queue as state of traffic environment and the average waiting time as the reward.). We can randomly sample a piece of memory information from the replay buffer and use stochastic gradient descent (SGD) to update the parameters. The algorithm also uses ϵ -greedy exploration to optimize the action strategy. The overall algorithm structure is shown in Fig. 3.

Further, the Nash-A2C algorithm process is detailed in Algorithm 1:

Algorithm 1: Nash-A2C algorithm.

Init Episode $B > 0, b = 1$; Minibatch Size $\hat{M} > 0$, Game Step N
Init Replay Buffer $\mathcal{D}$, θ_V and θ_A
repeat
  Reset, go to the x_0
  repeat
    Select $u \leftarrow \pi^{\theta_A}(x)$ or select u randomly (e.g.,ϵ-greedy)
    Observe $y_t = (x_{t-1}, u, x_t)$
    Store y_t to Replay Buffer $\mathcal{D}$
    Sampling from Replay Buffer:$\mathcal{Y} = \{y_i\}_{i=1}^{\hat{M}}$
    Optimize $\frac{1}{M+1}\sum_{y\in\mathcal{Y}\cup\{y_t\}} \hat{\mathcal{L}}(y, \theta_V, \theta_A)$ fix θ_A update θ_V
    Optimize $\frac{1}{M+1}\sum_{y\in\mathcal{Y}\cup\{y_t\}} \hat{\mathcal{L}}(y, \theta_V, \theta_A)$ fix θ_V update θ_A
  until $t > N$
until $b > B$
return θ_V and θ_A

4.1.3. Nash-A3C

The stability of the Nash-A2C algorithm during training is not very satisfactory because of the randomness of union action selection. Inspired by the asynchronous superior actor–critic algorithm(A3C) [37], we introduce the concept of *parallel space–time* to allow the Nash-A2C algorithm to be executed simultaneously in multiple environments. Multiple explorations in multiple environments aim to explore more policies and accelerate the speed of learning to have a stable learning process.

The Nash-A3C algorithm initially executes multiple instances in parallel on multiple environment copies, allowing different agents recording a history of interaction with the environment. As Fig. 4 shows, our algorithm sets a global network including actor network parameters θ_A and critic network parameters θ_V. The branch actor network and critic network in each environment will copy the parameters of the global network and introduce a replay buffer to store the triples $y_t^i = (x_{t-1}^i, u^i, x_t^i)$, where x_{t-1}^i is the operation performed in the union action u^i, with the next state of the environment as x_t^i, and the reward y_t.

We can randomly sample a piece of memory information from the replay buffer and use SGD to update the parameters in a similar way as Nash-A2C. The algorithm also uses ϵ -greedy exploration to optimize the action policy. The result of interaction between the environment and the multi-agent is returned to the global network. In this way, we can search policy space and reach the Nash equilibrium point in an efficient way.

The Nash-A3C algorithm process is detailed in Algorithm 2.

Algorithm 2: Nash-A3C.

Initialize the global parameters set θ_V and θ_A, global counter T
Initialize the branch parameters $\theta_V'^1$ and $\theta_A'^1$ … $\theta_V'^m$ and $\theta_A'^m$
Initialize t=1
repeat
  Reset policy gradient and value gradient $d\theta_V$=0, $d\theta_A$=0
  Sent global parameters to local $\theta'_V = \theta_V$, $\theta'_A = \theta_A$
  $t_{start} = t$
  repeat
    Select union action $u \leftarrow \pi^{\theta'_A}(x)$ or randomly select u (e.g.,ϵ-greedy)
    Observe State–Action–State $y'_t = (x'_{t-1}, u', x'_t)$
    Store to the Replay Buffer $\mathcal{D}$
    Sampling from Replay Buffer:$\mathcal{Y} = \{y'_i\}_{i=1}^{\hat{M}}$
    Optimize $\frac{1}{M+1}\sum_{y\in\mathcal{Y}\cup\{y_t\}} \hat{\mathcal{L}}(y, \theta'_V, \theta'_A)$ fix θ'_A update θ'_V
    Optimize $\frac{1}{M+1}\sum_{y\in\mathcal{Y}\cup\{y_t\}} \hat{\mathcal{L}}(y, \theta'_V, \theta'_A)$ fix θ'_V update θ'_A
    $t \leftarrow t+1$;$T \leftarrow T+1$;
  until To terminal x_t or $t - t_{start} == t_{max}$
  repeat
    i ← t
    Get reward r_t and new state x_{t+1}
    Accumulated gradient $d\theta_A$:
    $d\theta \leftarrow d\theta + \nabla_{\theta_A} \log\pi(a_i|x_i;\theta_A)(R - V(x_i;\theta_V))$
    Accumulated gradient$d\theta_V$:
    $d\theta_V \leftarrow d\theta_v + \partial(R - V(x_i;\theta_V))^2/\partial\theta_V$
    $i \leftarrow t$
  until $t > t_{start}$
  Update θ_V and θ_A according to $d\theta_V$ and $d\theta_A$
until $T > T_{max}$

4.2. A distributed computing IoT architecture for traffic simulation

In order to deploy distributed TSC algorithms like Nash-A3C in a real environment, we should consider overall architecture and network delay in the traffic simulation environment. To solve this issue, we introduce a distributed computing IoT architecture based on CityFLow [11], including one cloud computing layer, one fog computing layer (several fog computing nodes) and also one edge computing layer, which can be used for real-world deployment (as shown in Fig. 5).

(1) Cloud Layer: cloud computing layer gathers all critical information from multi-intersections in the city, which, in return, helps fog computing nodes to make more efficient decisions and provide more computing resources.
(2) Fog Layer: this Layer is the crucial part of the framework, which aims at producing intelligent signals to control traffic lights from fog nodes (FN). Most of TSC methods can be deployed on FN.

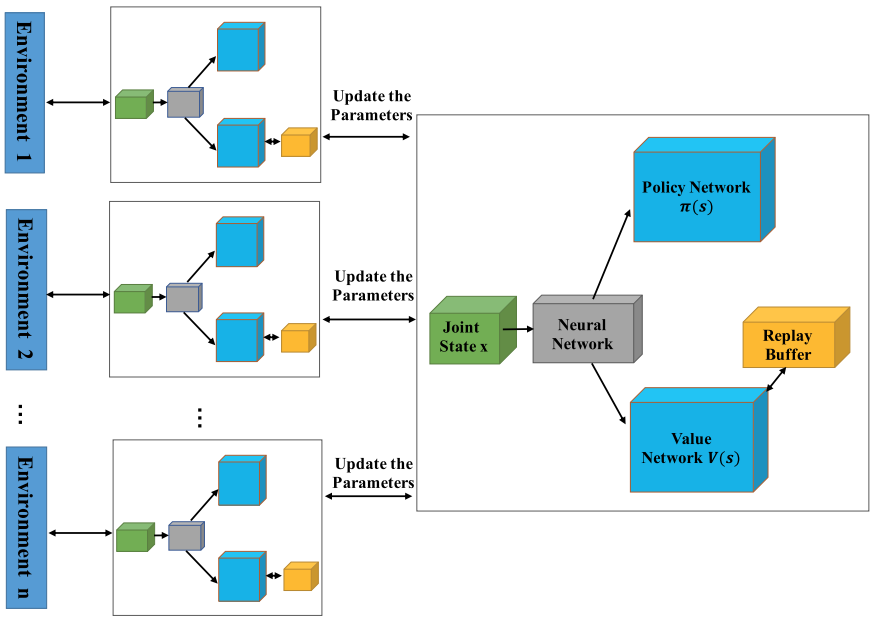

Fig. 4. The structure of Nash-A3C.

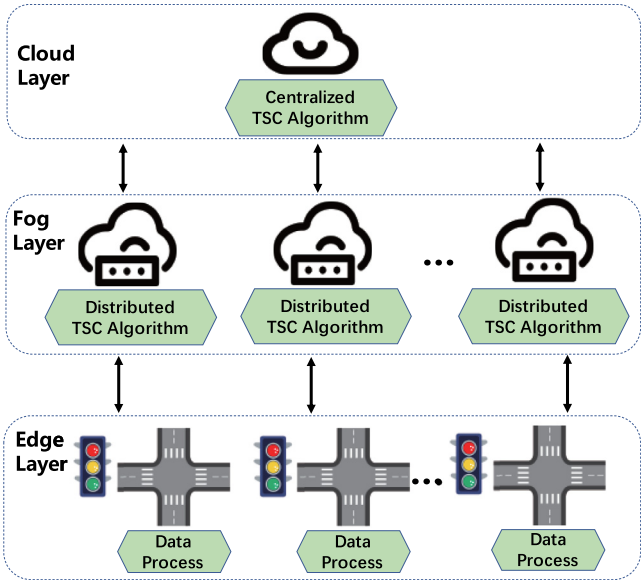

Fig. 5. The distributed computing architecture in traffic simulation environment.

(3) Edge Layer: We deploy edge computing equipment close to every traffic light. An edge computing device needs to have three functions: (1) it should detect vehicles' information (location, direction, velocity) from the surveillance video within its intersection in real-time and record the vehicle; (2) it should send its state (vehicles' information) to the FN nearby and receive the control command from the FN; (3) it should be able to control the traffic signal from FN.

4.3. Discussion

As shown in Table 3, assume the scenario of two cars coming along different roads towards an intersection. Each one now has two choices: 'Go' (continue driving) or 'Stop' (hit the brakes). We call the cars Player1 and Player2.

For each of the four tuples, the first number corresponds to the payoff to Player1 and the second one to Player2. If both players 'Go,' it will cause an accident. Thus, the high negative payoffs for both. If one Goes and the other Stops, the stopping one gets a small negative payoff -1, while the one who continues driving gets the best possible payoff 4. The worst case would be if both Stop, causing negative payoffs for both -3.

Table 3
Corresponding pay-off matrix for player1 and player2. (Player1 corresponds to the rows, and Player2 corresponds to the columns.)

Player1	Player2	
	Go	Stop
Go	$-10,-10$	$4,-1$
Stop	$-1, 4$	$-3,-3$

Table 4
'Best responses' on the payoff matrix.

Player1	Player2	
	Go	Stop
Go	$-10,-10$	$4^*, -1^*$
Stop	$-1^*, 4^*$	$-3,-3$

Consider Player1: if Player2 decides to Go, Player1's better payoff comes from doing Stop ($-1 > -10$). On the other hand, if Player2 decides to Stop, Player1 should Go ($4 > -3$). We also get these 'best responses' on the payoff matrix (as Table 4 shown).

So we have two different Nash equilibrium, including Player1 prefers (Go, Stop) and Player2 prefers (Stop, Go). But we must realize that each player will choose one of these two scenarios.

Based on the time instant when the two cars arrive at the intersection, the traffic lights will tell one player to stop while the other to Go—thus forcing the whole 'Game' into one of its Nash equilibrium. Obviously (like all Game theory simulations), this is a simplified version of what happens. Still, you get the basic idea: traffic lights push the game towards one of its Nash equilibrium. However, we will optimize the traffic lights action for a better Nash equilibrium for all intersections.

5. Experiment and results

In this section, we first introduce the implemented urban traffic simulator as discussed earlier. Then, we demonstrate the

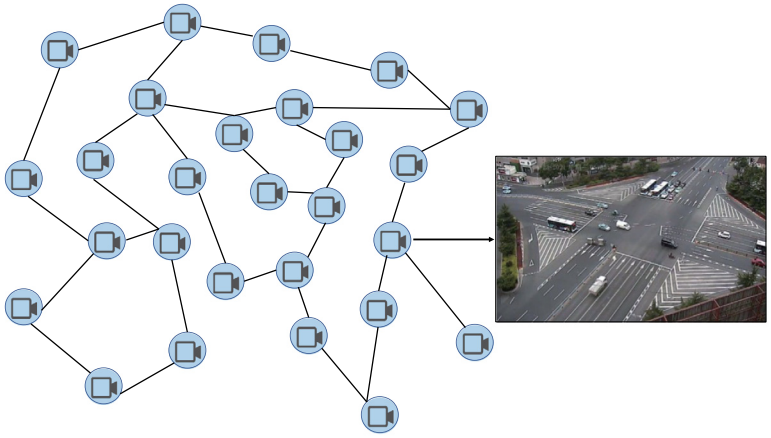

Fig. 6. The experiments simulator of urban traffic. Each node can also be regarded as the edge computing unit.

application of the Nash-A2C, Nash-A3C, and other methods on the simulator and collected data for comparing the performance of all models. Furthermore, we deploy all algorithms on our distributed computing architecture that is simulated at every intersection.

5.1. Experiment settings

5.1.1. Distributed environment settings

We use different processes on one server to simulate different intersections based on CityFlow. We start 27 edges and fog processes on the server and a cloud process. The 27 edges processes start first to save computing resources and stop after the data resources are preprocessed. We deploy NashA2C on the cloud process when implementing NashA2C and deploy NashA3C on every fog process and the cloud process when applying NashA3C.

5.1.2. Simulation environment settings

We build the simulation environment for experiments as the optimization methods of traffic simulation with surveillance video data and intersections topology of one city in China. Our urban multi-intersection simulation platform, as shown in Fig. 6, is based on real traffic flow data collected at 27 intersections connecting 45 roads. The initial traffic flow at each intersection is configured according to traffic flow dataset. Traffic signals are allocated to each intersection and are controlled through the simulator algorithm interface. Every 15 min, the traffic flow of each traffic intersection is updated through the traffic flow prediction algorithm.

We build two simulation environments as follows:

(1) The first environment, which does not consider the date or results delay from the edge computing node to the cloud computing center, but only collects the operating results of different algorithm models.
(2) The second environment considers the network transmission latency in industrial-scale deployments. We set the network transport latency from the intersection to the computing center to 6 s and the network transport latency for edge computing devices to 1 s.

5.1.3. Neural network hyper-parameters settings

The neural networks of Nash-A2C and Nash-A3C algorithms have the following hyperparameters settings: the policy network $\pi(x)$ and the value network $V(x)$ are set with four hidden layers, with 20, 60, 60 and 20 nodes, respectively. The layers are connected via ReLU activation; the main neural network has three hidden layers, each containing 20 nodes. ReLU is activated to connect the layers. The size of the initial replay buffer is 6000; the learning rate is set to 0.01.

Table 5
The average experimental results of 27 intersections.

Model	Average velocity (m/s)	Average waiting time (s)
Fixed-time	10.15	166.71
Q-learning	18.44	135.64
DQN	20.11	112.24
Nash-Q	24.12	90.75
Nash-A2C	**29.71**	**70.14**
Nash-A3C	**33.81**	**61.21**

5.2. Experiment process

The experimental process is divided into two parts. In the first part, the network delay is not considered. In the second part, the network delay is considered for the corresponding comparison.

5.2.1. Regardless of network delay

At first, we input the real traffic flow data in the simulation environment and make the algorithm interface available for the traffic light signal control algorithms without considering network delay. We run classical fixed-time, Q-learning [2], Deep Q-network (DQN) [38], Nash-Q [32], Nash-A2C, Nash-A3C algorithms separately in the simulation system.

The results of the test processes are shown in Tables 5 and 6.

As shown in Fig. 7, we can observe that, regarding the average velocity of vehicles over the 27 intersections, our methods performed faster in the same simulator for at least 23.3%; for the average waiting time of vehicles over the 27 intersections, our methods obtained a less waiting time at least 22.1%, and Nash-A3C performed best. As shown in Table 5, for the results of every intersection, our methods did not sacrifice the waiting time at intersections to achieve overall performance. Furthermore, the performance of every intersection was optimized at different levels.

5.2.2. Consideration of network delay

Similarly we run fixed-time, Q-learning [2], DQN [38], Nash-Q-learning (Nash-Q) [32], Nash-A2C, Nash-A3C algorithms in our simulation system. Nash-A2C and Nash-A3C are separately deployed on edge computing devices while other methods only have one computing center, as to take network delay into consideration for industrial deployment in real applications. We collected the cumulative waiting time of all vehicles in one episode.

The simulator feedback of each traffic intersection was recorded every minute. Similarly, the experiment uses 3000 training episodes, and the execution time for each training episode is 30 min.

Table 6
Experiment results of average waiting time at each intersection. (The id is the number of each intersection.)

ID	Fixed-time	Q-learning	DQN	Nash-Q	Nash-A2C	Nash-A3C
1	175.27	141.96	116.47	93.00	**72.48**	**66.09**
2	188.35	152.68	125.12	98.54	**78.62**	**73.90**
3	155.28	125.58	103.25	84.54	**63.11**	**54.15**
4	197.72	160.36	131.32	102.50	**83.01**	**79.49**
5	155.23	125.54	103.22	84.52	**63.08**	**54.12**
6	168.97	136.80	112.30	90.33	**69.53**	**62.33**
7	157.68	127.55	104.84	85.55	**64.23**	**55.58**
8	161.32	130.53	107.24	87.09	**65.94**	**57.76**
9	185.23	150.13	123.06	97.22	**77.16**	**64.76**
10	176.47	142.95	117.26	93.51	**73.05**	**66.80**
11	161.21	130.44	107.17	87.05	**65.89**	**57.69**
12	125.96	101.55	83.86	72.12	**49.35**	**36.65**
13	131.62	106.19	87.60	74.52	**52.01**	**40.03**
14	169.15	136.95	112.42	90.41	**69.61**	**62.43**
15	175.52	160.84	131.70	102.75	**101.03**	**79.84**
16	132.47	106.89	88.16	74.88	**52.41**	**40.53**
17	164.12	132.83	109.10	69.67	**67.25**	**59.43**
18	166.87	135.08	110.92	89.44	**68.54**	**61.07**
19	150.39	121.57	100.02	82.47	**60.81**	**51.23**
20	177.77	144.01	118.12	94.06	**73.66**	**67.58**
21	153.63	124.23	102.16	83.84	**62.33**	**53.17**
22	167.54	135.63	111.36	89.73	**68.86**	**61.47**
23	177.62	143.89	141.59	140.09	**90.30**	**67.49**
24	186.84	151.45	124.12	97.90	**77.91**	**72.99**
25	175.36	142.04	116.53	93.04	**72.53**	**66.14**
26	175.23	141.93	116.44	92.98	**72.46**	**66.06**
27	188.35	152.68	125.12	98.54	**78.62**	**73.90**

Table 7
The average cumulative waiting time and delay time of six methods in one episode.

Model	The average cumulative time (s)	The average delay time (s)	The average delay rate
Fixed-time	31173.9	0	**0**%
Q-learning	23116.5	4831.3	20.9%
DQN	21052.7	4526.2	21.5%
Nash-Q	20981.3	4678.8	22.3%
Nash-A2C	**19870.7**	**1828.1**	**9.2%**
Nash-A3C	**18990.8**	**1842.1**	**9.7%**

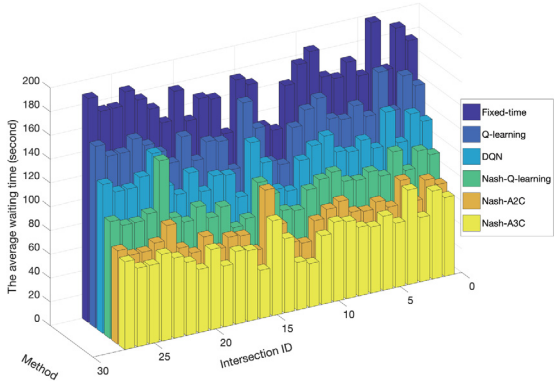

Fig. 7. Experiment results of average waiting time at each intersection.

After the training process, we test the algorithms in 1000 episodes. The result of the test process is shown in Table 7.

Regarding the average value of the cumulative waiting time of all vehicles in one episode, our methods outperform others.

Although the average delay time of the fixed-time method is 0, the delay rate of Nash-A2C is the lowest.

In summary, it can be observed that Nash-A3C shows consistent and outstanding performance by reducing average waiting time at a global system level rather than local optimum at some intersections, although Nash-A2C shows the best performance when the network delay is considered.

6. Conclusion

In this paper, we proposed two novel MARL algorithms, Nash-A2C and Nash-A3C, which intuitively introduced the Nash equilibrium theory into the actor–critic architecture of deep reinforcement learning. Additionally, we built the distributed IoT computing architecture of urban traffic, which was much more suitable for the distributed methods like Nash-A3C for TSC. We applied these methods to the distributed computing architecture with promising numerical results. The results of the experiments showed that our methods outperformed other models by 22.1% and 9.7% reduction of congestion time and network delay, respectively.

In future research, we will continue to study a lightweight MARL model. It is easier for industrial deployment without a performance drop.

CRediT authorship contribution statement

Qiang Wu: Conceptualization, Methodology. **Jianqing Wu:** Data curation, Writing. **Jun Shen:** Original draft preparation, Supervision. **Bo Du:** Writing – reviewing. **Akbar Telikani:** Software, Validation. **Mahdi Fahmideh:** Reviewing and editing. **Chao Liang:** Software, Writing – editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant Nos. 61673150, 11622538). Our work acknowledges the Science Strength Promotion Programme of UESTC, Chengdu, China.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.knosys.2022.108304.

References

[1] Q. Wu, J. Shen, B. Yong, J. Wu, F. Li, J. Wang, Q. Zhou, Smart fog based workflow for traffic control networks, Future Gener. Comput. Syst. 97 (2019) 825–835.
[2] H. Wei, G. Zheng, H. Yao, Z. Li, Intellilight: A reinforcement learning approach for intelligent traffic light control, in: Proceedings of the 24th ACM SIGKDD International Conference On Knowledge Discovery & Data Mining, 2018, pp. 2496–2505.
[3] X. Liang, X. Du, G. Wang, Z. Han, Deep reinforcement learning for traffic light control in vehicular networks, 2018, arXiv preprint arXiv:1803.11115.
[4] H. Xu, C. Zhang, J. Wang, D. Ouyang, Y. Zheng, J. Shao, Exploring parameter space with structured noise for meta-reinforcement learning, in: Proceedings of the 29th International Joint Conference on Artificial Intelligence, IJCAI, vol. 190, 2020, pp. 3153–3159.
[5] F. Rasheed, K.-L.A. Yau, Y.-C. Low, Deep reinforcement learning for traffic signal control under disturbances: A case study on sunway city, Malaysia, Future Gener. Comput. Syst. 109 (2020) 431–445.
[6] A. Zhu, F. Chen, H. Xu, D. Ouyang, J. Shao, Empowering the diversity and individuality of option: Residual soft option critic framework, IEEE Trans. Neural Netw. Learn. Syst. (TNNLS) (2021).
[7] Q. Wu, J. Wu, J. Shen, B. Yong, Q. Zhou, An edge based multi-agent auto communication method for traffic light control, Sensors 20 (15) (2020) 4291.
[8] H. Qin, H. Zhang, Intelligent traffic light under fog computing platform in data control of real-time traffic flow, J. Supercomput. 77 (5) (2021) 4461–4483.
[9] Y. Zhang, Y. Zhou, H. Lu, H. Fujita, Cooperative multi-agent actor–critic control of traffic network flow based on edge computing, Future Gener. Comput. Syst. 123 (2021) 128–141.
[10] A. Singh, T. Jain, S. Sukhbaatar, Learning when to communicate at scale in multiagent cooperative and competitive tasks, 2018, arXiv preprint arXiv:1812.09755.
[11] H. Zhang, Y. Ding, W. Zhang, CityFlow: A multi-agent reinforcement learning environment for large scale city traffic scenario, in: The Web Conference 2019 - Proceedings of the World Wide Web Conference (WWW), 2019, pp. 3620–3624.
[12] D. Ouyang, J. Shao, Y. Zhang, Y. Yang, H.T. Shen, Video-based Person re-identification via self-paced learning and deep reinforcement learning framework, in: Proceedings of the 26th ACM International Conference on Multimedia, ACM MM, 2018, pp. 1562–1570.
[13] B. Abdulhai, R. Pringle, G.J. Karakoulas, Reinforcement learning for true adaptive traffic signal control, J. Transp. Eng. 129 (3) (2003) 278–285.
[14] S. El-Tantawy, B. Abdulhai, An agent-based learning towards decentralized and coordinated traffic signal control, in: 13th International IEEE Conference on Intelligent Transportation Systems, IEEE, 2010, pp. 665–670.
[15] S. El-Tantawy, B. Abdulhai, H. Abdelgawad, Design of reinforcement learning parameters for seamless application of adaptive traffic signal control, J. Intell. Transp. Syst. 18 (3) (2014) 227–245.
[16] K. Prabuchandran, H.K. AN, S. Bhatnagar, Multi-agent reinforcement learning for traffic signal control, in: 17th International IEEE Conference on Intelligent Transportation Systems (ITSC), IEEE, 2014, pp. 2529–2534.
[17] W. Genders, S. Razavi, Using a deep reinforcement learning agent for traffic signal control, 2016, arXiv preprint arXiv:1611.01142.
[18] L.Y. Li Li, W. Feiyue, Traffic signal timing via deep reinforcement learning, IEEE/CAA J. Automat. Sin. 3 (3) (2016) 247–254.
[19] C.-H. Wan, M.-C. Hwang, Value-based deep reinforcement learning for adaptive isolated intersection signal control, IET Intell. Transp. Syst. 12 (9) (2018) 1005–1010.
[20] T. Chu, J. Wang, L. Codecà, Z. Li, Multi-agent deep reinforcement learning for large-scale traffic signal control, IEEE Trans. Intell. Transp. Syst. 21 (3) (2019) 1086–1095.
[21] D. Krajzewicz, Traffic simulation with sumo– simulation of urban mobility, in: International Series in Operations Research & Management Science, vol. 145, 2015.
[22] S.-V. Oprea, A. Bâra, Edge and fog computing using IoT for direct load optimization and control with flexibility services for citizen energy communities, Knowl.-Based Syst. 228 (2021) 107293.
[23] Y. Zhang, Y. Zhou, Distributed coordination control of traffic network flow using adaptive genetic algorithm based on cloud computing, J. Netw. Comput. Appl. 119 (2018) 110–120.
[24] Q.T. Minh, C.M. Tran, T.A. Le, B.T. Nguyen, T.M. Tran, R.K. Balan, Fogfly: A traffic light optimization solution based on fog computing, in: Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers, 2018, pp. 1130–1139.
[25] C. Tang, S. Xia, C. Zhu, X. Wei, Phase timing optimization for smart traffic control based on fog computing, IEEE Access 7 (2019) 84217–84228.
[26] P. Zhou, T. Braud, A. Alhilal, P. Hui, J. Kangasharju, Erl: Edge based reinforcement learning for optimized urban traffic light control, in: 2019 IEEE International Conference On Pervasive Computing And Communications Workshops, PerCom Workshops, IEEE, 2019, pp. 849–854.
[27] S.A. Gamel, A.I. Saleh, H.A. Ali, A fog-based traffic light management strategy (TLMS) based on fuzzy inference engine, Neural Comput. Appl. (2021) 1–19.
[28] I. Alvarez, A. Poznyak, A. Malo, Urban traffic control problem a game theory approach, in: 2008 47th IEEE Conference on Decision and Control, 2008, pp. 2168–2172, http://dx.doi.org/10.1109/CDC.2008.4739461.
[29] M. Khanjary, Using game theory to optimize traffic light of an intersection, in: CINTI 2013 - 14th IEEE International Symposium on Computational Intelligence and Informatics, Proceedings, 2013, pp. 249–253.
[30] M. Elhenawy, A.A. Elbery, A.A. Hassan, H.A. Rakha, An intersection game-theory-based traffic control algorithm in a connected vehicle environment, in: 2015 IEEE 18th International Conference on Intelligent Transportation Systems, 2015.
[31] K.-H.N. Bui, J.E. Jung, D. Camacho, Game theoretic approach on real-time decision making for IoT-based traffic light control, Concurr. Comput.: Pract. Exp. 29 (11) (2017) e4077.
[32] J. Hu, M.P. Wellman, Nash Q-learning for general-sum stochastic games, J. Mach. Learn. Res. 4 (Nov) (2003) 1039–1069.
[33] I. Grondman, L. Busoniu, G.A. Lopes, R. Babuska, A survey of actor-critic reinforcement learning: Standard and natural policy gradients, IEEE Trans. Syst. Man Cybern. C 42 (6) (2012) 1291–1307.
[34] A. Faramarzi, M. Heidarinejad, B. Stephens, S. Mirjalili, Equilibrium optimizer: A novel optimization algorithm, Knowl.-Based Syst. 191 (2020) 105190.
[35] F.-F. Chua, C.-S. Lee, Collaborative learning using service-oriented architecture: A framework design, Knowl.-Based Syst. 22 (2009) 271–274, http://dx.doi.org/10.1016/j.knosys.2009.01.003.
[36] H. Tang, Y. Wang, X. Liu, X. Feng, Reinforcement learning approach for optimal control of multiple electric locomotives in a heavy-haul freight train: A double-switch-Q-network architecture, Knowl.-Based Syst. 190 (2020) 105173.
[37] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International Conference On Machine Learning, PMLR, 2016, pp. 1928–1937.
[38] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, 2013, arXiv preprint arXiv:1312.5602.