Transportation Research Part C 125 (2021) 103059

Contents lists available at ScienceDirect

Transportation Research Part C

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

Network-wide traffic signal control optimization using a multi-agent deep reinforcement learning

Zhenning Li [a], Hao Yu [b,*], Guohui Zhang [c], Shangjia Dong [d], Cheng-Zhong Xu [a]

[a] State Key Laboratory of Internet of Things for Smart City, University of Macau, Macao
[b] School of Transportation, Southeast University, China
[c] Department of Civil and Environmental Engineering, University of Hawaii at Manoa, United States of America
[d] Department of Civil and Environmental Engineering, University of Delaware, United States of America

ARTICLE INFO

Keywords:
Multi-agent reinforcement learning
Knowledge sharing
Adaptive traffic signal control
Deep learning
Transportation network

ABSTRACT

Inefficient traffic control may cause numerous problems such as traffic congestion and energy waste. This paper proposes a novel multi-agent reinforcement learning method, named KS-DDPG (**K**nowledge **S**haring **D**eep **D**eterministic **P**olicy **G**radient) to achieve optimal control by enhancing the cooperation between traffic signals. By introducing the knowledge-sharing enabled communication protocol, each agent can access to the collective representation of the traffic environment collected by all agents. The proposed method is evaluated through two experiments respectively using synthetic and real-world datasets. The comparison with state-of-the-art reinforcement learning-based and conventional transportation methods demonstrate the proposed KS-DDPG has significant efficiency in controlling large-scale transportation networks and coping with fluctuations in traffic flow. In addition, the introduced communication mechanism has also been proven to speed up the convergence of the model without significantly increasing the computational burden.

1. Introduction

Traffic signal control is an efficient method of protecting traffic participants at intersections where multiple streams of traffic interact. Because of the capability of responding to fluctuating traffic demands, the adaptive traffic signal control (ATSC) system has been broadly implemented, and has attracted considerable interest in the research community (van de Weg et al., 2018). By receiving and processing data from strategically placed sensors, the ATSC system works based on real-time traffic dynamics about travel demand, traffic conditions and performance measurements of the control logic, such as traffic volume, queue length, vehicle speed, and travel time (Wang et al., 2019). Compared to the fixed-time control, ATSC has been proved to improve the quality of service that travelers experience on roadways, especially in busy urban areas, while reducing travel time by more than 10% on average (USDOT, 2017). However, a malfunctioning ATSC system may bring about more serious congestion and even traffic accidents (Li et al., 2019).

Numerous interdisciplinary approaches have been applied to improve the efficiency of ATSC in the past two decades, including but not limited to fuzzy logic (Cheng et al., 2016), case-based reasoning (Elkosantini et al., 2011), artificial neural networks (Ghanim and Abu-Lebdeh, 2015), genetic algorithm (Odeh et al., 2015), and immune network algorithm (Darmoul et al., 2017). Among these approaches, reinforcement learning (RL), which takes sequential actions rooted in Markov Decision Process (MDP) with a rewarding or penalizing criterion, is a promising approach to optimize the ATSC system. It is a type of goal-oriented algorithm

* Corresponding author.
E-mail address: seudarwin@gmail.com (H. Yu).

https://doi.org/10.1016/j.trc.2021.103059
Received 18 September 2020; Received in revised form 22 January 2021; Accepted 20 February 2021
Available online 4 March 2021

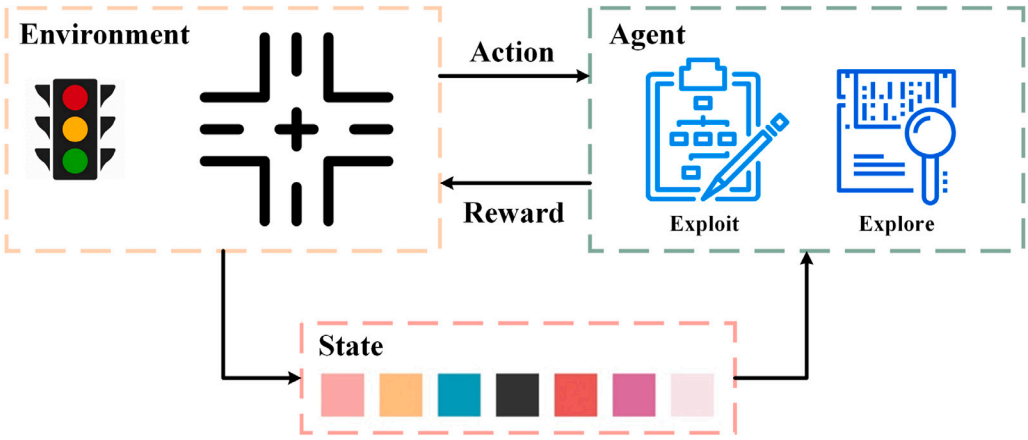


Fig. 1. Reinforcement Learning Framework for Traffic Light Control.

that learns how to achieve complex objectives in specific dimensions by learning from the experience. Different from conventional model-driven approaches that only work well under certain assumptions (Hegyi et al., 2005; Liu et al., 2009) (e.g., deterministic arrival rate, unlimited road capacity), RL is able to learn how to achieve complex objectives in specific dimensions by learning from the experience of interacting with the environment. Therefore, it has the potential to solving the ATSC optimization problem without strong assumptions, and to learn good strategies directly from trials and errors (Wei et al., 2019).

Let us take one of the most common settings for ATSC optimization using a single-agent RL framework in previous studies as an illustration (Prashanth and Bhatnagar, 2010). As shown in Fig. 1, the control unit in an isolated intersection is treated as an *agent*, and interacts with the traffic *environment* in a closed-loop MDP. The control policy is obtained by mapping from the traffic *states* (e.g., waiting time, queue length, total delay, etc.) to the corresponding optimal control *actions* (e.g., phase shift, cycle length change, green time extension, etc.). The agent iteratively receives a feedback *reward* for actions taken and adjusts the policy until it converges to the optimal control results. During the decision process, the policy that the agent takes combines both *exploitation* of already learned policies and *exploration* of new policies that never met before. Studies using similar RL frameworks to manipulate ATSC are not rare in the last two decades and have provided beneficial references for research (Abdulhai et al., 2003; Arel et al., 2010; Genders and Razavi, 2016; Mousavi et al., 2017; Zhang et al., 2020). For instance, a single-agent model-free Q-learning algorithm was developed for optimizing signal timing in a single intersection (Abdulhai et al., 2003). In this study, the authors used queue length as the state representation and accumulative delay between two action cycles as the reward. The comparison between the algorithm and the fixed-time control strategy under different traffic flows demonstrated the advantages of the RL-based method in dealing with fluctuating traffic flow demand. Single-agent SARSA-based algorithms using similar state–action–reward settings were also widely implemented in previous studies to solve the ATSC optimization problem of isolated intersections (Thorpe and Anderson, 1996; Wen et al., 2007; El-Tantawy et al., 2014).

However, many additional challenges arise when extending such a single-agent RL to a multi-agent system such as ATSC in a transportation network with multiple **interacting** intersections. First, in the commonly used centralized learning scheme, different agents are represented by a single agent and collaboratively executed towards a joint policy. The foremost drawback of this structure is that as the number of agents increases, the input size increases linearly, while the output joint policy space grows exponentially, making it difficult to tackle the network-wide ATSC optimization problem, especially in a large network. For instance, in the work of Prashanth and Bhatnagar (2010), they trained one centralized agent to determine the joint actions for all intersections in the network. However, this approach did not perform as well as expected due to the dimension curse of action space. In addition, in this kind of centralized optimization framework, collecting and processing information must be repeated to some extent, therefore it may face stability issues during deployment. Further complications of the *exploration–exploitation dilemma* also arise due to the presence of multiple intersections. The exploration–exploitation trade-off requires online RL algorithms that balance between the exploitation of the agent's current knowledge with exploratory information gathering actions taken to improve that knowledge (Buşoniu et al., 2010). Agents have to not only explore information about the environment, but also the information from other control units in order to adapt to their behaviors. However, too much exploration can undermine the stability of other agents and affect the efficiency of the algorithm, making it more difficult to explore the agent's learning tasks. While in a decentralized structure, since multiple control units need to interact with each other during the optimization, therefore, the property in an MDP that the reward distribution and dynamics need to be stationary is violated as the reward received by an agent also depends on other agents' actions (Shou et al., 2020). This issue, known as the *moving-target* problem, i.e., the best policy changes as the other agents' policies change, eliminates convergence guarantees and introduces extra learning instabilities. It becomes more troublesome, as our environment is characterized by a partially observable feature, that is, the agent cannot fully access the entire state space, and coordination is essential (Zhang et al., 2019). In an ATSC system, the need for coordination comes from the fact that the environmental impact of any agent's actions also depends on actions taken by other agents. Therefore, each agent's action must be consistent with others in order to achieve their expected results, e.g., minimizing network-wide delay, maximizing total throughput, etc.

In order to mitigate these issues, many previous studies have adopted various independently modeling RL frameworks, in which they treated each intersection as one agent and trained them separately (Abdoos et al., 2011; Aslani et al., 2017; Xiong et al., 2019). For instance, Mannion et al. (2016) developed three RL-based ATSC optimization algorithms and compared the control performance of them with a fixed-time control via experiments designed in a simple 3×3 grid network with uniform traffic flow. In their settings, each RL agent is responsible for controlling the light sequences of a single junction and can only observe the traffic state on the lanes directly connecting with the intersection. It should be noted that many early attempts like this work did not fully take advantage of communication protocols between different agents, and they assumed the agent can only adapt to its private observations accordingly without considering any other parts of the entire environment (Khamis and Gomaa, 2012). In some simple scenarios (such as small synthetic network and arterial network), this processing method has been proven to have acceptable performance (Brys et al., 2014; Prashanth and Bhatnagar, 2011). However, when the network becomes more complex, the non-stationary problem can no longer be neglected. Previous studies have found that if there is no communication or coordination mechanism among agents, the learning process usually lasts significantly longer than expected, or even is unable to converge to stationary policies (Nowé et al., 2012). The underlying reason for this situation is that the agent is unable to timely respond and adapt to the environment change before the environment has already changed again.

A recently proposed algorithm, named Multi-Agent Deep Deterministic Policy Gradient (MADDPG) (Lowe et al., 2017), is a trending state-of-the-art *multi-agent RL* paradigm and could partially address the above-mentioned problems encountered by single-agent RLs. This algorithm is enlightened of the actor–critic algorithm (Konda and Tsitsiklis, 2000) in which each agent has an actor to select actions and a critic to evaluate them. MADDPG utilizes a *centralized learning and decentralized execution* paradigm where a group of agents can be trained simultaneously by applying a centralized method. In addition, this approach can address the emergence of environment non-stationary and has been shown to perform well in a variety of mixed competitive and cooperative environments (Wang et al., 2020; Qie et al., 2019). However, MADDPG is still not the ideal panacea for our problem. First, as the studied network-wide ATSC optimization is a partially observable and sequential multi-agent decision-making problem, each agent is unable to observe the underlying Markov state. Taking into account the fact that traffic traverses the network from one intersection to another, cooperation among intersections is essential to help the agent have a better understanding of the entire environment. Based on the engineering experience, it is quite common that a well-functioning phase setting at this intersection without considering the state of the entire traffic environment may lead to significant congestion at other intersections after a period of time. In addition, if each agent only takes its optimal actions based on its own observation, in order to adapt to fluctuating traffic flows, the agent may need to constantly shift its phase setting, which may, unfortunately, result in the control system being less robust and effective. Numerous studies have also proven that considering cooperation during ATSC optimization could reduce phase adjustment times and achieve better control performance (e.g., larger average speed, less delay, etc.) (Wei et al., 2019; Arel et al., 2010; Chu et al., 2019). Since intersections are able to interact, developing an efficient communication protocol to allow agents to share their observations with others may be a promising way to achieve cooperation and collaboration among traffic signals. However, under the MADDPG framework, agents can only share each other's actions and observations during training through the critics, and there are no means to develop an explicit form of communication through their experiences.

In order to address all the aforementioned issues, based on the MADDPG framework, we further propose a novitiate multi-agent RL algorithm allowing agents to efficiently communicate, named KS-DDPG (**K**nowledge **S**haring-**D**eep **D**eterministic **P**olicy **G**radient). In our setting, communication among agents is realized via a carefully-designed *knowledge sharing mechanism*. The knowledge that represents the collective understanding of the environment by all agents is stored in the shared *knowledge container*. It is a highly compact vector with a pre-defined capacity and is constantly updated condition of the local observation history of each agent during the modeling process through non-linear gating-based operations. With the knowledge sharing mechanism, each agent could capture relevant aspects of the environment, and interpret and reconstruct the shared knowledge in its own way. The agent makes its decision based on not only its private observation but also its unique understanding of the shared knowledge obtained via the communication protocol. As far as we are aware, this is the first paper using such techniques. The remainder of this paper is organized as follows. Section 2 reviews related work on MDP and reinforcement learning. Section 3 presents the formalization of the problem setup, the details of the proposed method, and the description of the learning process. In Section 4, the proposed method is evaluated by a grid simulated network and a real-world traffic network in Montgomery County, Maryland, respectively. Section 5 summarizes this paper and discusses future research directions.

2. Related work

In this section, we will introduce concepts and terminology that are related to our work, including MDP, classical reinforcement learning algorithms, and the MADDPG algorithm, respectively.

2.1. Multi-agent extension of Markov decision process

The network-wide ATSC optimization problem can be described as a Partially Observable Markov Decision Process (POMDP) with N **interacting** agents. This formulation assumes a set of state, $\mathcal{S}$, containing all the states characterizing the environment; a set of actions $\mathcal{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_N\}$ where each $\mathcal{A}_i$ is a set of possible actions for the i^{th} agent; and a set of observations $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_N\}$, one for each agent in the environment. Each $\mathbf{o}_i$ in $\mathcal{O}_i$ indicates a partial characterization of the current state and is private for the agent. State transitions are controlled by the current state and one action from each agent: $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_N \mapsto \mathcal{S}$, $i = 1, \ldots, N$. Each agent also has an associated reward function based on the joint actions, $r_i : \mathcal{S} \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_N \mapsto \mathbb{R}$, $i = 1, \ldots, N$. The joint

policy is given by $\pi := (\pi_{\theta_1}, \ldots, \pi_{\theta_N}) \subset \Pi$. Because the rewards of the agents depend on the joint policy, the return for each agent is denoted as follows:

$$R_i^{\pi}(S) = E(\sum_{k=0}^{\infty} \gamma^k r_{i,k+1} | s_0 \in S, \pi) \tag{1}$$

where γ is discount factor of future rewards, s_0 is the initial state.

2.2. Q-learning algorithms

Q-learning is a classical RL method that fits the Q-function with a parametric model Q_θ. The action-value function in Q-learning for policy $\boldsymbol{\pi}$ is given as $Q^{\boldsymbol{\pi}}(s,a) = \mathbb{E}[R|s^t = s,\ a^t = a]$. This Q function can be recursively rewritten as $Q^{\boldsymbol{\pi}}(s,a) = \mathbb{E}_{s'}\left[r(s,a) + \gamma \mathbb{E}_{a' \sim \boldsymbol{\pi}}\left[Q^{\boldsymbol{\pi}}\left(s', a'\right)\right]\right]$. On the other hand, deep Q-networks (DQN) learns the action-value function Q^* corresponding to the optimal policy by minimizing the loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'}\left[\left(Q^*(s,a|\theta) - y\right)^2\right], \text{where } y = r + \gamma \max_{a'} \overline{Q}^*\left(s', a'\right) \tag{2}$$

where $\overline{Q}$ is a target Q function whose parameters are intermittently updated with the latest θ, which helps to stabilize learning.

2.3. Policy gradient algorithms

The major idea of policy gradient (PG) algorithms is to directly adjust the parameters θ of the policy in order to maximize the objective $J(\theta) = \mathbb{E}_{s\sim p^{\boldsymbol{\pi}},\ a\sim \boldsymbol{\pi}_\theta}[R]$ by taking steps in the direction of $\nabla_\theta J(\theta)$. Using the Q function defined previously, the gradient of the policy can be written as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim p^{\boldsymbol{\pi}},\ a\sim \boldsymbol{\pi}_\theta}[\nabla_\theta \log \boldsymbol{\pi}_\theta(a|s)\, Q^{\boldsymbol{\pi}}(s,a)] \tag{3}$$

where $p^{\boldsymbol{\pi}}$ is the state distribution. There are several methods to estimate $Q^{\boldsymbol{\pi}}$, resulting in different practical algorithms, for instance, REINFORCE (Duan et al., 2016), actor–critic (Konda and Tsitsiklis, 2000), etc. Specifically, the actor–critic algorithm uses an approximation, $Q_\varpi(s,a)$, to estimate the true action-value function $Q^{\boldsymbol{\pi}}(s,a)$.

A recent extension of the policy gradient framework using deterministic policies $\mu_\theta : S \mapsto \mathcal{A}$, named deterministic policy gradient (DPG), combined the benefits of DQN and actor–critic algorithms (Silver et al., 2014). In particular, if the action space $\mathcal{A}$ and the policy $\boldsymbol{\mu}$ are continuous (then the Q-function is presumed to be differentiable with respect to the action argument), the gradient of the objective $J(\theta) = \mathbb{E}_{s\sim p^{\boldsymbol{\mu}}}[R(s,a)]$ can be rewritten as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim D}\left[\nabla_\theta \boldsymbol{\mu}_\theta(a|s)\, \nabla_a Q^{\boldsymbol{\mu}}(s,a)\ \ |_{a=\mu_\theta(s)}\right] \tag{4}$$

One of the major advantages of DPGs is that although stochastic policy gradients are integrated into both state and action spaces, DPGs are only integrated over the state space, requiring fewer samples in problems with large action spaces. Previous studies have shown that DPG greatly improves the stochastic policy gradient equivalence in high-dimensional continuous control problems (Silver et al., 2014).

2.4. Multi-agent deep deterministic policy gradient algorithm

In the article of Lowe et al. (2017), the authors proposed a multi-agent extension of the actor–critic policy gradient methods where the critic is augmented with extra information about the policies of other agents. More specifically, in the game with N agents, let $\boldsymbol{\mu}_i$ be the set of continuous policies with respect to parameters $\boldsymbol{\theta}_i$. Then, the gradient of the expected return for agent i is equal to

$$\nabla_{\theta_i} J(\mu_i) = \mathbb{E}_{x,a\sim D}\left[\nabla_{\theta i} \boldsymbol{\mu}_i\left(a_i|o_i\right) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}\left(x, a_1, \ldots,\ a_N\right) |_{a_i=\boldsymbol{\mu}_i(o_i)}\right] \tag{5}$$

where $Q_i^{\boldsymbol{\mu}}\left(x, a_1, \ldots,\ a_N\right)$ is the *centralized action-value function* that takes as input of all agents and state information x, and outputs the Q-value for agent i. $\mathcal{D}$ is the experience reply buffer and contains the element of $\left(x,\ x',\ a_1, \ldots,\ a_N, r_1, \ldots,\ r_N\right)$, which keeps a memory of previous action–reward pairs and train the network with samples. Furthermore, $Q_i^{\boldsymbol{\mu}}$ is updated as:

$$\mathcal{L}\left(\theta_i\right) = \mathbb{E}_{\mathbf{x},a,r,\mathbf{x}'\sim D}\left[\left(Q_i^{\boldsymbol{\mu}}\left(x, a_1, \ldots,\ a_N\right) - y\right)^2\right] \tag{6}$$

where y is the expected return computed by target network, and is equals to

$$y = r_i + \gamma Q_i^{\boldsymbol{\mu}'}\left(\mathbf{x}', a_1', \ldots,\ a_N'\right) |_{a_j'=\boldsymbol{\mu}_j'(o_j)} \tag{7}$$

where $\boldsymbol{\mu}' = \left\{\boldsymbol{\mu}_{\theta_1'},\ \ldots, \boldsymbol{\mu}_{\theta_N'}\right\}$ is the set of *target policies* with delayed parameters θ_i'.

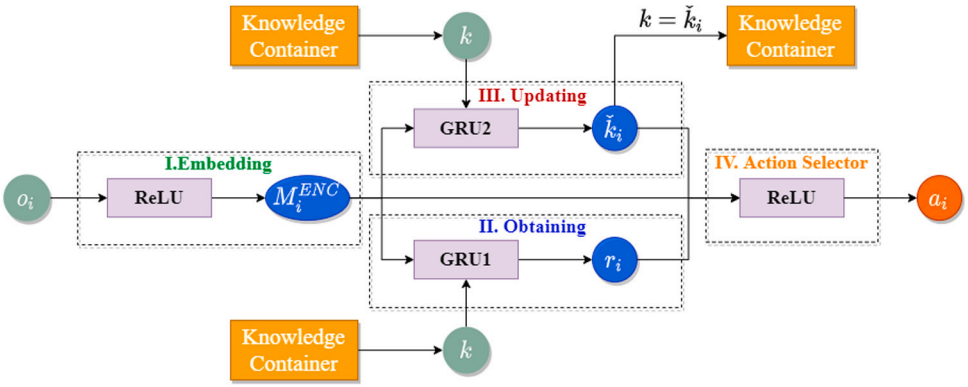


Fig. 2. Flowchart of knowledge-sharing.

3. Methodology

3.1. Problem statement

We consider a road network with $N\,(N > 1)$ signalized intersections, and adopt a multi-agent extension of partially observable Markov decision process characterized by the components $\langle S, \mathcal{O}, \mathcal{A}, \mathcal{P}, r, \pi, \gamma \rangle$ as mentioned in Section 2.1. More specifically, we assume that each intersection is controlled with a control unit that is able to adapt its phase setting based on the traffic patterns or human inputs. Traffic sensors are implemented at each lane of each intersection and are able to automatically collect real-time traffic data and send them to the nearest control unit. A novel knowledge-sharing mechanism enabled reinforcement learning approach is proposed to solve the network-wide ATSC optimization problem. Through the proposed knowledge-sharing framework, each agent is able to explicitly capture relevant traffic state of the system observed by other agents (i.e., control units), and therefore could simultaneously learn from both its private observations and the collectively learned representation of the system that accumulates through experiences coming from others to optimize its policy. The agent can also interpret the shared knowledge in its own unique way as needed to optimize its policy. In addition, this knowledge-sharing mechanism is designed to be used in both training and execution.

3.2. Knowledge sharing architecture

As aforementioned, a knowledge-sharing mechanism is proposed as the communication protocol for the agent to share and utilize its understanding of its and others' observations. A centralized *knowledge container* C is designed to store the *collective knowledge* $\mathbf{k} \in \mathbb{R}^{1\times K}$ that progressively collected by all agents as they interact. $\mathbf{k}$ is a highly compact vector and represents the collective understanding of the environment by all agents. Each agent has its own way to interpret it and rebuild it. When making a decision, the agents need to taking both its private observation but also its own comprehension of the collective knowledge $\mathbf{k}$ into account via the help of the communication protocol. As presented in Fig. 2, each agent needs to first execute an *Observation Embedding* operation to encode its private observation into a latent space. Then, prior to taking an action, the agent has to execute an *Knowledge Obtaining* operation in order to access the knowledge container, and initially retrieve and interpret the knowledge stored in it. After obtaining the knowledge, the agent will perform a *Knowledge Updating* operation, and update the current knowledge content based on its own observations. During training, these operations are learned without imposing any prior constraints on the nature of the collective knowledge. Whilst in the execution process, the agents use the communication protocol that they have learned to obtain and update the knowledge over the whole episode. In order to build a trainable *end-to-end* model, we use deep neural networks as function approximators for policies, and use learnable gated functions to facilitate each agent's interactions with the knowledge. The details of these operations are introduced and discussed below. In order to make the description more concise, time-related subscripts in the formulas will be omitted once there is no ambiguity.

I. Observation Embedding. At time t, the up-to-date n-dimensional private observations received by agent i, i.e., the traffic volume on each entrance lane and the current phase at the ith intersection, $\mathbf{o}_i$, are mapped on to an embedding latent space $\mathbf{M}_i^{ENC} \in \mathbb{R}^{1\times m}$, through a multi-layer perceptron:

$$\mathbf{M}_i^{ENC} = \text{Embed}(\mathbf{o}_i) = \text{ReLU}(\mathbf{o}_i \mathbf{W}_{i,oM} + \mathbf{b}_{i,M}) \tag{8}$$

where $\mathbf{W}_{i,oM} \in \mathbb{R}^{n\times m}$ and $\mathbf{b}_{i,M} \in \mathbb{R}^{1\times m}$ are the weight matrix and the bias vector to learn, and ReLU represents the ReLU activation function. The second element of the subscript of $\mathbf{W}_{i,oM}$ indicates that the vector is related to the observation $\mathbf{o}$ and the embedding result $\mathbf{M}^{ENC}$. Similarly, the M in the subscript of $\mathbf{b}_{i,M}$ implies this vector is related to $\mathbf{M}^{ENC}$. The following formulas will also use this notation method. The embedding $\mathbf{M}_i^{ENC}$ represents latent state of the current traffic condition at the ith intersection, and plays a fundamental role in selecting new actions and in the following knowledge obtaining and updating phases.

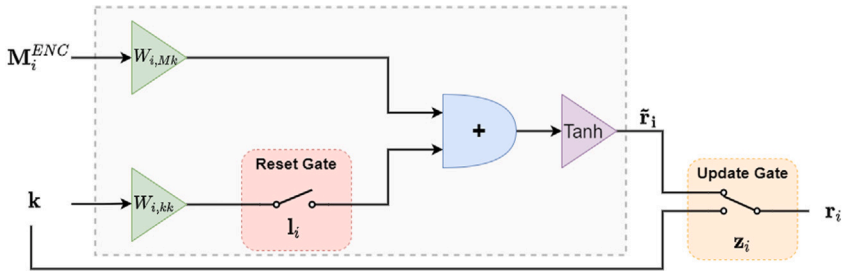


Fig. 3. A Brief Summary of Knowledge Obtaining Operation.

II. Knowledge Obtaining. This obtaining operation is performed after the encoding operation and allows the agent to acquire and comprehend the collective knowledge $\mathbf{k}$ previously captured by other agents. By interpreting this knowledge content, the agent has access to what others have learned. The intuition behind this setting is that for the intersections in the network, some of them may have strong correlations, i.e., some traffic patterns always travel through them in a sequence; while others have limited interactions since only a few travelers are likely to choose the routes that consist of them conjointly. Therefore, taking observations obtained by other agents into account may have significant favorable impacts on the agent's control performance when making decisions. Instead of gathering all historical observations of others, we use an *update gate and reset gate* mechanism inspired by Gated Recurrent Unit (GRU) (Chung et al., 2014) to let the agent itself decides what data in the collective knowledge is kept or threw away. Therefore, compared to fully communicating, this operation could remarkably enhance the execution efficiency.

Let the vector $\mathbf{r}_i \in \mathbb{R}^{1\times K}$ denote the knowledge content that the agent i decides to obtain at time t, then the update gate vector $\mathbf{z}_i \in \mathbb{R}^{1\times K}$ that helps the model determine how much of the past knowledge (from previous time steps) needs to be passed along to the future can be written as

$$\mathbf{z}_i = \sigma_g(\mathbf{M}_i^{ENC}\mathbf{W}_{i,Mz} + \mathbf{k}\mathbf{W}_{i,kz} + \mathbf{b}_{i,z}) \tag{9}$$

where $\mathbf{k}$ is the knowledge content already stored in the container, $\mathbf{W}_{i,Mz} \in \mathbb{R}^{m\times K}$ and $\mathbf{W}_{i,kz} \in \mathbb{R}^{K\times K}$ are the weights to be learned, $\mathbf{b}_{i,z} \in \mathbb{R}^{1\times K}$ is the bias term, and σ_g is the sigmoid activation function to squash the result between 0 and 1.

Essentially, the reset gate vector $\mathbf{l}_i \in \mathbb{R}^{1\times K}$ is used from the model to decide how much past knowledge to forget. To calculate it, we use a similar formula like that in Eq. (9) :

$$\mathbf{l}_i = \sigma_g(\mathbf{M}_i^{ENC}\mathbf{W}_{i,Ml} + \mathbf{k}\mathbf{W}_{i,kl} + \mathbf{b}_{i,l}) \tag{10}$$

where $\mathbf{W}_{i,Ml} \in \mathbb{R}^{m\times K}$ and $\mathbf{W}_{i,kl} \in \mathbb{R}^{K\times K}$ are also learnable weight matrices, and $\mathbf{b}_{i,l} \in \mathbb{R}^{1\times K}$ is the bias term. Assuming $\tilde{\mathbf{r}}_i \in \mathbb{R}^{1\times K}$ is the candidate knowledge obtaining vector, and its calculation method is as follows:

$$\tilde{\mathbf{r}}_i = \tanh(\mathbf{M}_i^{ENC}\mathbf{W}_{i,Mk} + \mathbf{l}_i \odot (\mathbf{k}\mathbf{W}_{i,kk}) + \mathbf{b}_{i,k}) \tag{11}$$

where $\odot$ is the Hadamard (element-wise) product operator between two vectors, tanh is the activation function and is applied to every element of its input. Here we use a nonlinearity in the form of tanh to ensure that the values in $\tilde{\mathbf{r}}_i$ remain in the interval $(-1, 1)$. The result is a candidate since we still need to consider the action of the update gate. Therefore, at the last step, we incorporate the effect of the update gate $\mathbf{z}_i$. This determines the extent to which the final knowledge content ($\mathbf{r}_i$) obtained by agent i at time t is just that already exists in the container($\mathbf{k}$) and by how much the new candidate knowledge ($\tilde{\mathbf{r}}_i$) is used. The update gate $\mathbf{z}_i$ can be used for this purpose, simply by taking elementwise convex combinations between $\tilde{\mathbf{r}}_i$ and $\mathbf{z}_i$, that is,

$$\mathbf{r}_i = \mathbf{z}_i \odot \mathbf{k} + (1 - \mathbf{z}_i) \odot \tilde{\mathbf{r}}_i \tag{12}$$

The knowledge obtaining operation can be further summarized as Fig. 3. We use a single pole single throw switch to represent the effect of the reset gate, and a single pole double throw switch to demonstrate the effect of the update gate. When $\mathbf{l}_i$ equals to 0, indicating there are no impacts of previous knowledge on the current knowledge content; otherwise, the impacts exist. It should also be noteworthy that when $\mathbf{z}_i$ equals to 0 and $\mathbf{l}_i$ equals to 1, this operation degenerates into a standard RNN. In summary, the reset and update gates help the agent capture short-term and long-term dependencies in sequences respectively. All the weights and bias vectors in this operation are learnable and agent-specific, indicating that each agent is able to interpret the knowledge in its own unique way.

III. Knowledge Updating. In the updating phase, we also use a GRU based operation to let the agent itself decides what in its observations should be updated and what to discarded. Assuming the agent first generates a candidate knowledge updating content, $\hat{\mathbf{k}}_i \in \mathbb{R}^{1\times K}$, which depends on its own observations and the stored knowledge through a non-linear mapping, that is,

$$\hat{\mathbf{k}}_i = \tanh(\mathbf{M}_i^{ENC}\mathbf{W}_{i,M\hat{k}} + \mathbf{p}_i \odot (\mathbf{k}\mathbf{W}_{i,k\hat{k}}) + \mathbf{b}_{i,\hat{k}}) \tag{13}$$

where $\mathbf{W}_{i,M\hat{k}} \in \mathbb{R}^{m\times K}$ and $\mathbf{W}_{i,k\hat{k}} \in \mathbb{R}^{K\times K}$ are weights to learn, $\mathbf{b}_{i,\hat{k}} \in \mathbb{R}^{1\times K}$ is the bias vector, and $\mathbf{p}_i \in \mathbb{R}^{1\times K}$ is the reset gate which is similar to Eq. (10), and can be further written as

$$\mathbf{p}_i = \sigma_g(\mathbf{M}_i^{ENC}\mathbf{W}_{i,Mp} + \mathbf{k}\mathbf{W}_{i,kp} + \mathbf{b}_{i,p}) \tag{14}$$

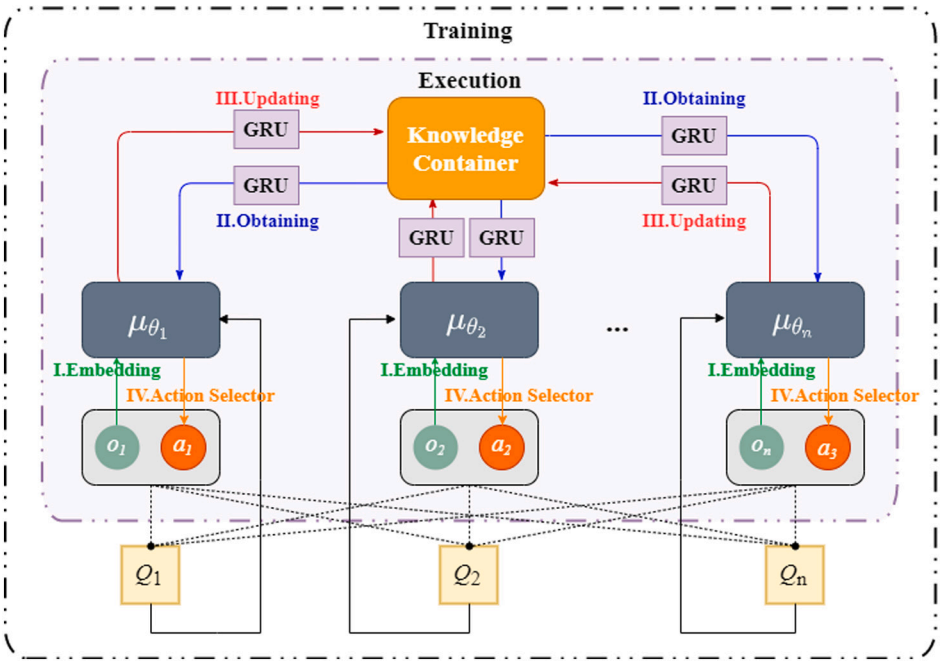


Fig. 4. Framework of KS-DDPG algorithm.

where $\mathbf{W}_{i,Mp} \in \mathbb{R}^{m\times K}$ and $\mathbf{W}_{i,kp} \in \mathbb{R}^{K\times K}$ are learnable weight matrices, and $\mathbf{b}_{i,\hat{k}} \in \mathbb{R}^{1\times K}$ is the bias vector.

Following Eq. (9), we can have the update gate $\mathbf{q}_i \in \mathbb{R}^{1\times K}$ as

$$\mathbf{q}_i = \sigma_g(\mathbf{M}_i^{ENC}\mathbf{W}_{i,Mq} + \mathbf{k}\mathbf{W}_{i,kq} + \mathbf{b}_{i,q}) \tag{15}$$

where $\mathbf{W}_{i,Mq} \in \mathbb{R}^{m\times K}$, $\mathbf{W}_{i,kq} \in \mathbb{R}^{K\times K}$ and $\mathbf{b}_{i,q} \in \mathbb{R}^{1\times K}$ are learnable weight metrics and bias vector, respectively.

Agent i then finally generates an updated knowledge $\check{\mathbf{k}}_i$ as a weighted combination based on its current observations and previous knowledge $\mathbf{k}$, as follows:

$$\check{\mathbf{k}}_i = \mathbf{q}_i \odot \mathbf{k} + (1 - \mathbf{q}_i) \odot \hat{\mathbf{k}}_i \tag{16}$$

The updated knowledge $\check{\mathbf{k}}_i$ is stored in the container C as $\mathbf{k}$ and made accessible to all the agents. At each time step, agents sequentially obtain and update the content of knowledge using above procedures.

IV. Action Selector. Once the agent completing both obtaining and updating operations, it can develop its action, a_i which depends on its current encoding of its observations $\mathbf{M}_i^{ENC}$, its own interpretation of the current knowledge content $\mathbf{r}_i$ and its updated version $\check{\mathbf{k}}_i$, that is

$$a_i = \phi_{\theta_i^a}^{ACT}(\mathbf{M}_i^{ENC}, \mathbf{r}_i, \check{\mathbf{k}}_i) \tag{17}$$

where $\phi_{\theta_i^a}^{ACT}$ is a neural network parameterized by θ_i^a. Therefore, the resulting policy function can be further written as a composition of functions:

$$\boldsymbol{\mu}_{\theta_i}(\mathbf{o}_i, \mathbf{k}) = \varphi_{\theta_i^a}^{ACT}\left(\mathbf{M}_i^{ENC}(\mathbf{o}_i), \mathbf{r}_i(\mathbf{o}_i, \mathbf{k}), \check{\mathbf{k}}_i(\mathbf{o}_i, \mathbf{k})\right) \tag{18}$$

in which θ_i contains all the relevant parameters from the encoding, obtaining, and updating phases.

3.3. Learning algorithm

All agent-specific policy parameters, i.e., θ_i, are learned end-to-end. Inspired by the MADDPG algorithm introduced in Section 2.4, an actor–critic model enables knowledge sharing within a centralized learning and decentralized execution framework is adopted. The framework of the proposed KS-DDPG algorithm is presented in Fig. 4. In actor–critic models, the actor is used to select actions, while the critic is used to evaluate the actor's actions and provide feedback. Since the input of the policy of our model (as described in Eq. (18)) is $(\mathbf{o}_i, \mathbf{k})$, we therefore use Eq. (5) to rewrite the gradient of the expected return for agent i as

$$\nabla_{\theta_i} J(\boldsymbol{\mu}_{\theta_i}) = \mathbb{E}_{\mathbf{x},a,\mathbf{k}\sim\mathcal{D}}\left[\nabla_{\theta_i}\boldsymbol{\mu}_{\theta_i}(\mathbf{o}_i, \mathbf{k})\nabla_{a_i} Q^{\boldsymbol{\mu}_{\theta_i}}(\mathbf{x}, a_1, \ldots, a_N)|_{a_i=\boldsymbol{\mu}_{\theta_i}(\mathbf{o}_i,\mathbf{k})}\right] \tag{19}$$

where D is the replay buffer storing past experiences. It contains transitions in the form of $(\mathbf{x}, \mathbf{x}', a_1, \ldots, a_N, \mathbf{k}, r_1, \ldots, r_N)$. $Q^{\mu_{\theta_i}}$ is updated according to Eqs. (6) and (7). The detailed pseudo-code of the learning and execution procedure is presented in Algorithm 1.

Algorithm 1 KS-DDPG Algorithm for N Interacting Agents

Initialize actors $(\boldsymbol{\mu}_{\theta_1}, \ldots, \boldsymbol{\mu}_{\theta_N})$ and critics networks $(Q_{\theta_1}, \ldots, Q_{\theta_N})$
Initialize actor target networks $(\boldsymbol{\mu}'_{\theta_1}, \ldots, \boldsymbol{\mu}'_{\theta_N})$ and critic target networks $(Q'_{\theta_1}, \ldots, Q'_{\theta_N})$
Initialize replay buffer D
for episode=1 to E **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Initialize knowledge container C, and receive initial state $\mathbf{x}$
  for $t = 1$ to **max episode length do**
    for agent $i = 1$ to N **do**
      Receive observation $\mathbf{o}_i$ and the collective knowledge $\mathbf{k}$
      Set $\mathbf{k}_i = \mathbf{k}$
      Generate knowledge encoding vector $\mathbf{M}_i^{ENC}$ using Eq. (8)
      Generate knowledge obtaining vector $\mathbf{r}_i$ using Eq. (12)
      Generate knowledge updating vector $\check{\mathbf{k}}_i$ using Eq. (16)
      Store the new knowledge in the container $\mathbf{k} \leftarrow \check{\mathbf{k}}_i$
      Generate new time dependent noise instance $\mathcal{N}_t$
      Select action a_i using Eq. (17)
    end for
    Set $\mathbf{x} = (\mathbf{o}_1, \ldots, \mathbf{o}_N)$ and $\boldsymbol{\Phi} = (\mathbf{k}_1, \ldots, \mathbf{k}_N)$
    Execute actions $\mathbf{a} = (a_1, \ldots, a_N)$, observe rewards r and next state $\mathbf{x}'$
    Store $(\mathbf{x}, \mathbf{x}', \mathbf{a}, \boldsymbol{\Phi}, r)$ in replay buffer D
  end for
  for agent $i = 1$ to N **do**
    Sample a random minibatch Θ of S samples $(\mathbf{x}, \mathbf{x}', \mathbf{a}, \boldsymbol{\Phi}, r)$ from D
    Set $y = r_i + \gamma Q^{\boldsymbol{\mu}'_{\theta_i}}(\mathbf{x}', a'_1, \ldots, a'_N)|_{a'_j = \boldsymbol{\mu}'_{\theta_j}(\mathbf{o}_j, \mathbf{k}_j)}$
    Update critic by minimizing the loss:

$$\mathcal{L}(\theta_i) = \frac{1}{S} \sum_{(\mathbf{x}, \mathbf{x}, \mathbf{a}, \boldsymbol{\Phi}, r) \in \Theta} \left(y - Q^{\boldsymbol{\mu}_{\theta_i}}(\mathbf{x}, a_1, \ldots, a_N) \right)^2$$

    Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} \boldsymbol{J} \approx \frac{1}{S} \sum_{(\mathbf{x}, \mathbf{x}', \mathbf{a}, \boldsymbol{\Phi}, r) \in \Theta} \left(\nabla_{\theta_i} \boldsymbol{\mu}_{\theta_i}(\mathbf{o}_i, \mathbf{k}) \nabla_{a_i} Q_i^{\boldsymbol{\mu}_{\theta_i}}(\mathbf{x}, a_1, \ldots, a_i, \ldots, a_N)|_{a_i = \boldsymbol{\mu}_{\theta_i}(\mathbf{o}_i, \mathbf{k}_i)} \right)$$

  end for
  Update target network parameters for each agent i:

$$\theta'_i = \tau \theta_i + (1 - \tau) \theta'_i$$

end for

As illustrated in Algorithm 1, only the learned actors are used to make decisions and select actions during execution. Each agent takes its action successively. The agent receives its private observations, obtains the knowledge stores in C, generates a new version of the knowledge, stores it back into C and selects its action a_i using $\boldsymbol{\mu}_{\theta_i}$. The policy of the next agent is then driven by the update C. An experience replay buffer contains the tuples $(\mathbf{x}, \mathbf{x}', \mathbf{a}, \boldsymbol{\Phi}, r)$ is adopted to record experiences of all agents. By minimizing the loss function $\mathcal{L}(\theta_i)$, the model attempts to improve the estimate of the critic network which is used to improve the policy itself through update with the sampled policy gradient $\nabla_{\theta_i} \boldsymbol{J}$.

3.4. Agent design

State Space and Observation Space. We assume that at time t agent i can only observe part of the system state $\mathbf{s}_i \in S$ as its observation $\mathbf{o}_i \in \mathcal{O}$. Although sensing additional combinations of data from upstream and downstream may have more advantageous impacts on the control performance, the dramatically increased state space can readily result in an intractability model. In order to limit the state space, two effortlessly collected elements, including the current phase and traffic volume in each entrance lane, were selected as key indicators to describe the state of the agent. The volume of a lane is defined as the number of vehicles on the lane, which equals the sum of queuing vehicles and moving vehicles on the entrance lane.

Set of Actions. Different action schemes have significant influences on traffic signal control strategies. In current ATSC systems, such as SCOOT and SCATS, there are several different control strategies, for instance, phase shift, phase length change, phase itself,

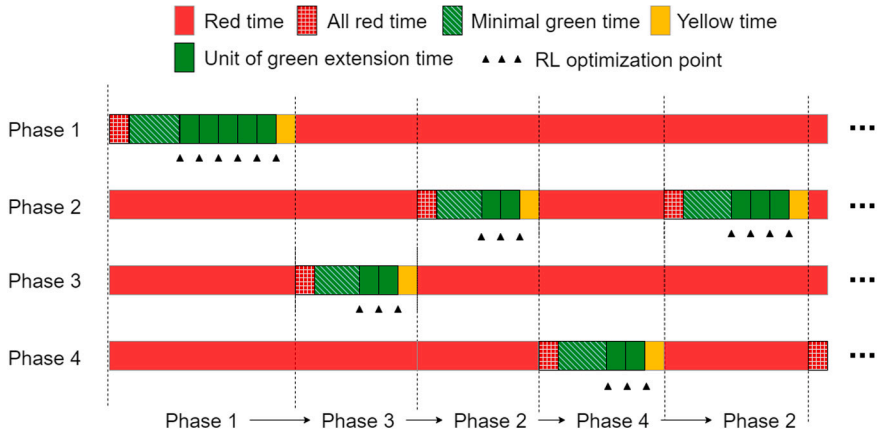


Fig. 5. Diagram of Candidate Actions in an Acyclic ATSC.

etc., or a combination of some of the above. Undoubtedly, using a flexible candidate action set could considerably increase the action space as well as the computational burden. In contrast, fixing the phase structure and phase sequence during the optimization process may bring forth lags in timing plan updating, making it unable to respond to the traffic flow fluctuation. Therefore, this kind of control method is a responsive ATSC rather than a real-time ATSC.

In order to seek a balance between the benefits of flexible action setting and its impacts on the ease of problem handling, it is assumed that the cycle lengths are not fixed in this case, but can only be adjusted between the maximum and minimum limits to ensure practicality. As shown in Fig. 5, the agent accounts for a variable phasing sequence in an acyclic timing scheme with an unfixed cycle length and an unsettled phasing sequence. More specifically, the control action is either to extend the current phase (i.e., choose the number of units of green extension) or to switch to any other phase, and consequently, some unnecessary phases may be skipped according to the fluctuations in traffic. In addition, each chosen phase is assumed to have a length limit with predetermined minimal green time, yellow time, and red time as the settings in engineering practice. It can be considered that it is desirable to limit the variability of consecutive cycles and ensure traffic safety, although some degradation of performance is likely.

Reward. Due to the complex spatial–temporal structure of traffic data, the reward should be spatially decomposable and can be easily measured after an action. Therefore, the reward of an agent is defined as the reduction of the average delay of vehicles at all entrance lanes associated with the control units, i.e., the difference between the average delays of two successive decision points. A positive value of the reward indicates that the delay is reduced by this value after executing the selected action; otherwise, this action results in an increase in the average delay.

4. Experiments

4.1. General setups

The proposed KS-DDPG algorithm is evaluated in the Simulation of Urban MObility (SUMO) platform. SUMO is an open-source traffic simulation package designed by the Institute of Transportation Systems at the German Aerospace Center to handle large road networks and has already been widely used in transportation-related research (Krajzewicz et al., 2012). SUMO was chosen because of its scalability as it includes an API called TraCI (Traffic Control Interface). TraCI allows users to query and modify the simulation status and extend the existing SUMO functionality. Finally, the OpenAI Gym which is a framework designed for the development and evaluation of reinforcement learning algorithms runs compatible with SUMO to set up a simulation environment (Brockman et al., 2016).

The numerical experiments are carried out through two different networks: (1) *Grid*: a Manhattan-style virtual 10 × 10 grid network with the same signalized intersection structures; and (2) *MoCo*: a real-world network extracted from downtown Montgomery County, Maryland with heterogeneous intersections of both signalized and non-signalized ones. The detailed structures of the two networks are illustrated in Fig. 6. In addition, Table 1 presents the detailed information of lanes and phase combinations at each signalized intersection in both experiments.

In both experiments, all vehicles are set to be homogeneous. The car-following behavior is modeled with the Intelligent Driver Model (Treiber et al., 2000), and the lane-changing behavior is modeled with the SUMO built-in dynamics models (Erdmann, 2015). The maximum speed of all vehicles is 50 mph, and the maximal acceleration and deceleration rates are 3.3 feet/s^2 and 15 feet/s^2, respectively. In the Grid experiment, traffic flows are generated according to Poisson distribution from the intermediate nodes at the edge of each direction and merge into the four corner nodes. The volume of each traffic flow is constant and equal to 750 pcu/h. The OD matrices are first randomly generated, and then stored and used without any other further changes in the subsequent experiment. In the MoCo experiment, in order to test the robustness and effectiveness of the proposed KS-DDPG algorithm, the traffic demand

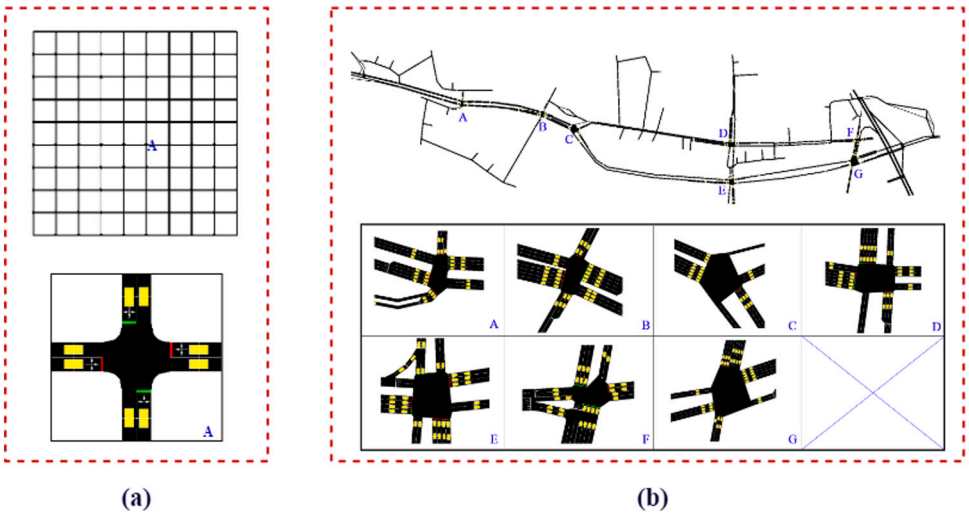


Fig. 6. Network Structures in Numerical Experiments (a)*Grid* (b) *MoCo*.

Table 1
Information regarding signals under the control.

Experiment	Intersection ID	Lanes of approaches	Legal movement of phases	Aberrations
Grid	All	NB:1*LTR	NL+NT	Approaches:
		SB:1*LTR	SL+ST	NB: Northbound
		WB:1*LTR	WL+WT	SB: Southbound
		EB:1*LTR	EL+ET	WB: Westbound
MoCo	A	NB:1*L+1*TR	NL+NT	EB: Eastbound
		SB:1*L+1*TR	SL+ST	
		WB:1*L+2*T+1*TR	WL+WT	
		EB:1*L+2*T+1*TR	EL+ET	
	B	NB:1*LT+1*R	NL+NT	Movement of a Lane:
		SB:1*LT+1*R	SL+ST	T: Through
		WB:1*LT+3*T+1*R	WL+WT	L: Left turning
		EB:1*L+3*T+1*R	EL+ET	R: Right turning
	C	NB:1*T	NT	
		EL+ETWB:3*L+2*T	WL+WT	
		EB:2*T+1*R	ET	Legal Movement of a Phase:
	D	NB:1*L+2*T+2*R	NL+NT	First digit: Approach
		SB:1*L+1*T+1*TR	SL+ST	Second digit:
		WB:2*L+2*T+1*R	WL+WT	Turning Movement
		EB:1*L+1*T+1*TR	EL+ET	
	E	NB:1*L+1*TR	NL+NT	
		SB:1*L+2*T+1*TR	SL+ST	
		WB:1*L+2*T+1*R	WL+WT	
		EB:1*LT+1*T+1*TR	EL+ET	
	F	NB:1*L+1*T+1*TR	NL+NT	
		SB:2*L+1*T+1*TR	SL+ST	
		WB:1*LT	WL+WT	
		EB:1*L+1*TR	EL+ET	
	G	NB:2*L+1*TR	NL+NT	
		SB:1*LT+1*TR	SL+ST	
		WB:1*L+1*T+1*TR	WL+WT	
		EB:1*L+2*T+2*R	EL+ET	

data consisting of both congestion generation and recovery processes are used as the input to simulate the recurrent congestion situation. The volume increased from almost 332 pcu/h/ln to 694 pcu/h/ln and then gradually recovered to 358 pcu/h/ln at the end of simulation time. The daily Origin–Destination (OD) matrices are extracted from the Montgomery-National Capital Park and Planning Commission, and the traffic volumes are obtained from the Maryland Department of Transportation and utilized to estimate the OD matrix from the raw daily OD matrices. For the unsignalized intersections, the default stop-sign control logic in SUMO is utilized. In addition, other basic settings of all signalized intersections in both experiments are listed in Table 2. The KS-DDPG algorithm is trained in episodes, and different random seeds are used for generating different training and evaluation episode, but the same seed is shared for the same episode. The KS-DDPG consists of two networks, i.e., the actor network and the critic network. The actor network receives the state and the obtained knowledge and outputs the actions. A one-layer network with 2^9 units is used

Table 2
Basic settings of signalized intersections.

Terms	Through	Left (if applicable)
Unit of green extension (s)	2	2
Minimum green (s)	15	5
Maximum green (s)	60	25
Yellow (s)	3	
Red clearance (s)	3	
Cycle length (s)	50 to 120	

for observation embedding, and a one-layer network with 2^8 units is used for actor selector. We used networks with three hidden layers (2^{10}, 2^9 and 2^8 units) for critics, and networks with two hidden layers (2^9 and 2^8 units) for the actors. The Adam optimizer with a learning rate of 0.001 for critic and 0.0001 for policies are selected. The reward discount factor is set to be 0.95, the size of replay buffer is 10^6, and the batch size is 2^{10}. We update network parameters after every 100 samples added to the replay buffer using soft updates with $\tau = 0.01$.

4.2. Benchmarks

The performance of the proposed algorithm is compared with the fixed-time traffic control, a state-of-the-art network-wide signal control method named MaxPressure (Varaiya, 2013), and three well-trained benchmark RL algorithms in a simulation replication with the same random state, including DQN introduced in Section 2.2, DDPG introduced in Section 2.3, and MADDPG introduced in Section 2.4, respectively. Webster's method is used to determine the fixed-time control plan for each intersection (Webster, 1958). For the MaxPressure approach, we follow its original setting, which is to greedily choose the phase that maximizes the pressure (i.e., a metric related to the upstream and downstream queue lengths) (Varaiya, 2013). In both DQN and DDPG, the agents are trained independently. In other words, each intersection is controlled by an agent. The agent does not share observations and parameters with others, but independently updates their own networks. For the MADDPG algorithm, all settings except for knowledge sharing related parameters are the same as KS-DDPG. In addition, different hyperparameters for benchmarks are tested several times to ensure adequate performance, and fine-tuned values are injected into these models. Finally, all algorithms are trained with 1,200 episodes given episode horizon $T = 720$ steps. All the computations are performed in an NVIDIA DGX-2 system.

4.3. Experiment results and discussions

4.3.1. Convergence comparison

The average reward per agent per training episode of each algorithm which quantifies how well the task has been solved is calculated as

$$\overline{R} = \frac{1}{T}\sum_{t=0}^{T-1}\left(\frac{1}{N}\sum_{\forall i \in N} r_{i,t}\right) \tag{20}$$

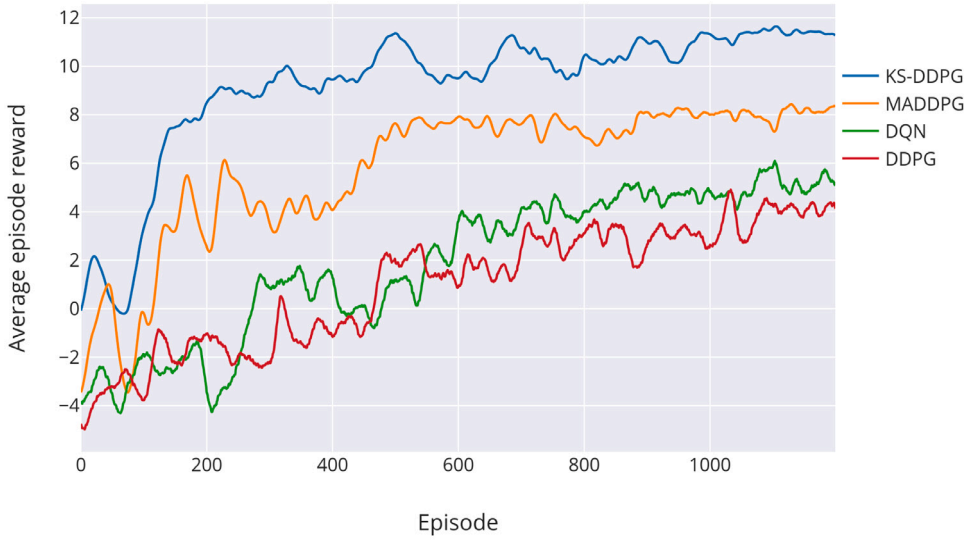
The convergence comparison results calculated by Eq. (20) between KS-DDPG and other RL-based baselines for both experiments are demonstrated in Fig. 7. In general, as RLs learn from accumulated experience and eventually achieves a local optimum, the training curve will increase and then converge. It should be noted that the presented curves are the best ones selected from all tests, and they are smoothed with a moving average of 5 points.

From Fig. 7(a), we can find that for the large-scale Grid experiment, the two independently trained RLs, i.e., DQN and DDPG, fail to learn very stable policies in given 1,200 episodes. Although the curves have shown some promising trends, given the great computational costs of running the full set of simulations, it was not affordable to let it run indefinitely, therefore, we only presented the convergence results within 1,200 episodes. In contrast, the two multi-agent algorithms, MADDPG and KS-DDPG, both successfully converge as their rewards fluctuate in very small ranges at the end of the training. Moreover, compared to MADDPG, KS-DDPG presents a more superior learning curve.

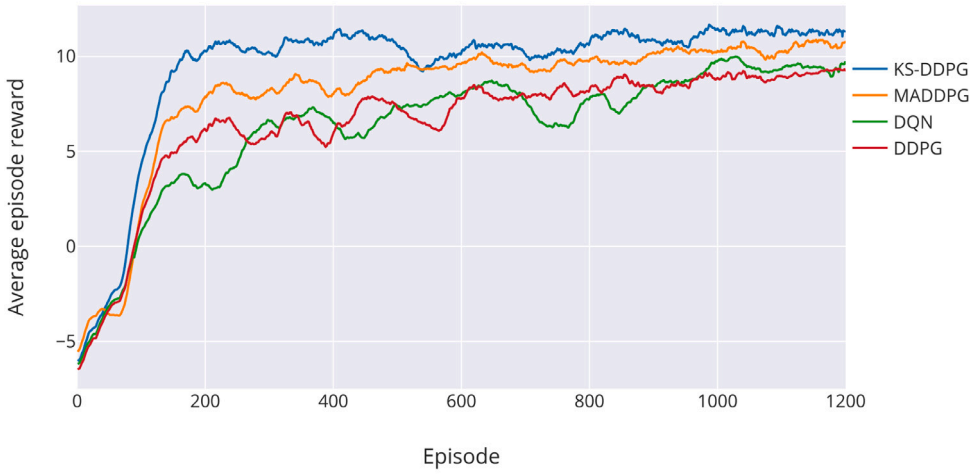
Nevertheless, for the small-scale MoCo experimental results presented in Fig. 7(b), the difference between learning curves of KS-DDPG and MADDPG is not as significant as in the Grid experiment. It may indicate that the policy inferring mechanism of MADDPG could yield acceptable performance when the number of agents is not too large. In addition, although not very stable, DQN and DDPG are also able to obtain positive rewards, implying that they have learned favorable policies to mitigate delays during the training process.

As the MADDPG algorithm is what our method builds on, the comparison between KS-DDPG and MADDPG (in which agents cannot communicate with others) has great potential to better demonstrate the improvements brought by the proposed commutation protocol. Based on several benchmark empirical performance metrics, including *jumpstart performance*, *asymptotic performance*, and *time to threshold* [1] Taylor and Stone (2009), we can discover that KS-DDPG performs betters than MADDPG in both experiments

[1] *Jumpstart performance*: The initial performance after the first episode; *Asymptotic performance*: The final learned performance after total episodes; *Time to threshold*: The learning time needed to achieve a prespecified performance level.



(a) Grid Experiment



(b) MoCo Experiment

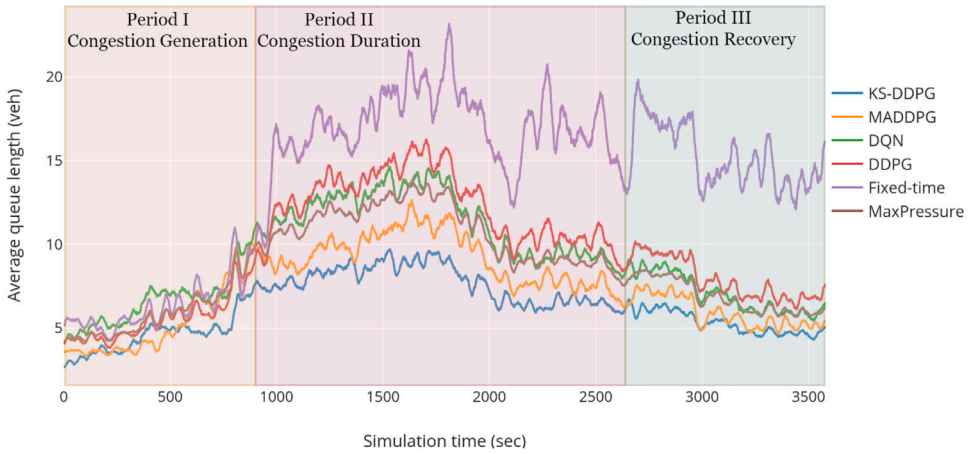
Fig. 7. Training Curves for KS-DDPG and Other RL Baselines.

by a huge amount. These results indicate that the knowledge-sharing mechanism does not retard model convergence, but on the contrary speeds up the search for optimal policy. Besides, these results also imply that this explicit communication technique could help MARL learn better policies more effortlessly in complex environments.

4.3.2. Control performance comparison

The performance comparison between KS-DDPG and other baselines with reference to the average queue length of the entire network at each simulation step in the Grid experiment is presented in Fig. 8(a). It is noteworthy that the obvious variability of these curves is caused by phase switching. It can be observed that all the RL-based methods have smaller average queue lengths than the fixed-time control, and KS-DDPG outperforms others as expected. In addition, both DQN and DDPG have very similar performance with the state-of-the-art model-driven traffic control optimization approach, i.e., MaxPressure. These results demonstrate that all RL-based control methods, especially KS-DDPG, have certain improvements in reducing intersection congestion compared with traditional methods. Table 3 further demonstrates the comprehensive comparison results these control methods. These metrics in Table 3 are firstly spatially aggregated at each time over evaluation episodes, then the temporally averaged results are calculated and presented. As the results show, KS-DDPG achieves significant performance improvements over all the baselines. Compared with MaxPressure, on average, KS-DDPG reduces about 28.9% queue length, 35.1% intersection delay, and 21.0% number of stops, and increases about 18.3% vehicle speed. KS-DDPG also has about at least 10% improvement on each measurement than the MADDPG control method.

(a) Grid Experiment



(b) MoCo Experiment

Fig. 8. Average Queue Length for KS-DDPG and Other Baselines. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 3
Control performance comparison in grid experiment.

Metrics	KS-DDPG	MADDPG	DQN	DDPG	Fixed-time	MaxPressure
Avg. Reward	9.20	5.84	1.88	1.05	NA	NA
Avg. Queue Length [veh]	5.45	6.03	7.22	7.67	9.82	7.66
Avg. Intersection Delay [s]	30.74	35.73	45.71	47.10	62.21	47.35
Avg. Vehicle Speed [feet/s]	22.34	20.59	18.25	19.03	13.77	18.88
Avg. Number of Stops [/s]	6.28	6.73	7.99	8.02	11.35	7.90

Similarly, the evaluation results on queue lengths of the MoCo experiment are presented in Fig. 8(b). Besides, the detailed comparisons on control metrics of these methods are listed in Table 4. Different from that in the Grid experiment, based on the V/C ratio, we roughly divide the whole simulation process into three periods, including congestion generation, congestion duration, and congestion recovery. It can be discovered that when the traffic volume is not large (V/C <0.5), all the methods have acceptable control performance. When the volume increases sharply in Period II, the average queue lengths of different methods also grow significantly, and pretty noticeable differences of them could be perceived in the figure. During this period, the maximum average queue length of the fixed time control has reached above 25 veh/ln, indicating the network is highly congested. In contrast, the KS-DDPG has a more favorable control outcome, as its maximum average queue length is less than 10 veh/ln. The control performance of other RLs are all better than the fixed time but worse than KS-DDPG in Period II, and MADDPG performs the best among these three RLs. Besides, MaxPressure performs better than DQN and DDPG, but the difference is not very significant. In Period III, all RLs and MaxPressure demonstrate a significant trend to alleviate congestion. However, for fixed-time, this trend is not apparent. As

Table 4
Control performance comparison in MoCo experiment.

Metrics	KS-DDPG	MADDPG	DQN	DDPG	Fixed-time	MaxPressure
Avg. Reward	9.47	8.17	6.31	6.73	NA	NA
Avg. Queue Length in Period I [veh]	4.57	5.26	6.60	5.54	6.35	5.93
Avg. Queue Length in Period II [veh]	6.80	8.71	9.65	9.01	12.62	8.88
Avg. Queue Length in Period III [veh]	5.47	6.28	7.26	8.20	13.55	7.13
Avg. Queue Length [veh]	6.27	7.34	9.03	9.44	13.85	8.53
Avg. Intersection Delay [s]	41.75	50.33	62.72	65.14	93.27	62.04
Avg. Vehicle Speed [feet/s]	20.45	18.02	16.20	16.16	13.55	17.23
Avg. Number of Stops [/s]	11.76	13.82	15.47	16.06	24.35	16.22

Table 5
Training time for different models.

Experiment	Avg. training time for one episode (minutes)			
	KS-DDPG	MADDPG	DQN	DDPG
Grid	36.37	30.86	125.28	111.11
MoCo	5.22	5.13	8.13	7.85

shown in the green area of Fig. 8(b), KS-DDPG can promptly alleviate congestion as expected, while the average queue lengths of other RLs rebounded to some extent with the fluctuation of traffic. Interestingly, as shown in Table 4, the performance gaps between KS-DDPG and other baselines become significantly larger than those in the Grid experiment. For instance, in the Grid experiment, the number of stops for MADDPG, DQN, DDPG, fixed-time, and MaxPressure are 1.07, 1.27, 1.28, 1.81, and 1.26 times than that for KS-DDPG, respectively. However, these numbers in the MoCo experiment became 1.17, 1.32, 1.37, 2.01, and 1.38, respectively. Similar findings can also be observed in the comparison of these methods in other metrics. All in all, the above results prove that KS-DDPG has a sound control outcome, no matter during the periods of congestion generation, duration or recovery.

In conclusion, compared with conventional transportation and RL-based baselines, KS-DDPG achieves consistent performance improvements across different road networks and traffic conditions. The advantage becomes even larger as the assessed traffic scenario changes from synthetic regular network to real-world irregular and dynamic environment. We believe these favorable results are attributed to the explicit communication among agents enabled by the knowledge-sharing mechanism. More specifically, since fixed-time and MaxPressure controls are not able to learn from the feedback of the environment, it is not surprising that they can barely achieve satisfactory results. DQN and DDPG also show limited capabilities in alleviating congestion, especially in dynamic traffic situations, mainly because they independently optimize the strategy of a single intersection. Furthermore, our approach also has obtained at least 10% control improvements over the state-of-the-art multi-agent RL, i.e., MADDPG. Since there are no communication channels in MADDPG, agents are unable to share their understanding of the environment during modeling. Instead, this algorithm utilizes a probabilistic network and maximizes the log probability of outputting another agent's observed actions to infer other agents' policies. The growing performance gaps between KS-DDPG and MADDPG from the Grid experiment to the MoCo experiment also demonstrate that direct cooperation through knowledge sharing is better than implicit cooperation through inference of other policies.

In addition, we also compare the training time of KS-DDPG and other RL-based baselines, and the results are presented in Table 5. In order to keep fair comparisons, all the methods are individually evaluated on the computing platform. It can be discovered that the training time for the proposed KS-DDPG is comparable to that of MADDPG in both experiments, indicating the extra computational burden brought by the communication channel is acceptable.

5. Conclusions

In this work, we have introduced KS-DDPG, a novel multi-agent reinforcement learning framework, to optimize network-wide ATSC. A knowledge-sharing mechanism is utilized as the intra-agent communication protocol to improve coordination skills among agents. The knowledge stored in the knowledge container is a representation of the collective learning of the traffic environment collected by all agents, and is used to better form individual policies for each agent. The proposed algorithm is compared with both state-of-the-art conventional transportation and RL-based methods by careful evaluation in synthetic and real-world scenarios. Results show that KS-DDPG achieves superior performance relative to all the baselines with acceptable computational costs. Compared with MADDPG, the significantly more stable training curves and favorable traffic control results strongly prove the importance of introducing explicit communication channels to achieve agent cooperation.

One of the main limitations of the algorithm is that all agents need to perform communication during the entire modeling process, resulting in limited overall communication efficiency. One possible solution is to add a "master agent" to determine the appropriate time for each agent to access the knowledge container. Although this measure adds an extra layer of complexity to the model, it may reduce the number of knowledge container accesses required in a large-scale system and increase overall scalability. Besides, in future work, we will consider using heterogeneous vehicles to build a more realistic traffic flow. We will also try to optimize traffic control and route guidance coordinately under the mixed flow consisting of both connected and conventional vehicles.

CRediT authorship contribution statement

Zhenning Li: Conceptualization, Methodology, Investigation, Writing - original draft. **Hao Yu:** Conceptualization, Methodology, Investigation, Writing - review & editing. **Guohui Zhang:** Methodology, Writing - review & editing. **Shangjia Dong:** Conceptualization, Methodology, Supervision. **Cheng-Zhong Xu:** Methodology, Validation, Writing - review & editing.

Acknowledgments

This paper is supported by National Key Research and Development Program of China (No. 2019YFB2102100), the Science and Technology Development Fund of Macau SAR (File no. 0015/2019/AKP), Guangdong-Hong Kong-Macao Joint Laboratory of Human-Machine Intelligence-Synergy Systems (No. 2019B121205007), and the National Natural Science Foundation of China (No. 61803083).

References

Abdoos, M., Mozayani, N., Bazzan, A.L., 2011. Traffic Light Control in Non-Stationary Environments Based on Multi Agent Q-Learning. ITSC, pp. 1580–1585.
Abdulhai, B., Pringle, R., Karakoulas, G.J., 2003. Reinforcement learning for true adaptive traffic signal control. J. Transp. Eng. 129 (3), 278–285.
Arel, I., Liu, C., Urbanik, T., Kohls, A.G., 2010. Reinforcement learning-based multi-agent system for network traffic signal control. IET Intell. Transp. Syst. 4 (2), 128–135.
Aslani, M., Mesgari, M.S., Wiering, M., 2017. Adaptive traffic signal control with actor-critic methods in a real-world traffic network with different traffic disruption events. Transp. Res. C 85, 732–752.
Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W., 2016. Openai gym. arXiv preprint arXiv:1606.01540.
Brys, T., Pham, T.T., Taylor, M.E., 2014. Distributed learning and multi-objectivity in traffic light control. Connect. Sci. 26 (1), 65–83.
Buşoniu, L., Babuška, R., De Schutter, B., 2010. Multi-agent reinforcement learning: An overview. In: Innovations in Multi-Agent Systems and Applications-1. Springer, Berlin, Heidelberg, pp. 183–221.
Cheng, J., Wu, W., Cao, J., Li, K., 2016. Fuzzy group-based intersection control via vehicular networks for smart transportations. IEEE Trans. Ind. Inf. 13 (2), 751–758.
Chu, T., Wang, J., Codecà, L., Li, Z., 2019. Multi-agent deep reinforcement learning for large-scale traffic signal control. IEEE Trans. Intell. Transp. Syst. 21 (3), 1086–1095.
Chung, J., Gulcehre, C., Cho, K., Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.
Darmoul, S., Elkosantini, S., Louati, A., Said, L.B., 2017. Multi-agent immune networks to control interrupted flow at signalized intersections. Transp. Res. C 82, 290–313.
Duan, Y., Chen, X., Houthooft, R., Schulman, J., Abbeel, P., 2016. Benchmarking deep reinforcement learning for continuous control. In: International Conference on Machine Learning, 1329–1338, June.
El-Tantawy, S., Abdulhai, B., Abdelgawad, H., 2014. Design of reinforcement learning parameters for seamless application of adaptive traffic signal control. J. Intell. Transp. Syst. 18 (3), 227–245.
Elkosantini, S., Mnif, S., Chabchoub, H., 2011. 2011 IEEE Symposium on Computational Intelligence in Vehicles and Transportation Systems (CIVTS) Proceedings. A system for the traffic control in signposted junctions 52–58.
Erdmann, J., 2015. SUMO's lane-changing model. In: Modeling Mobility with Open Data. Springer, Cham, pp. 105–123.
Genders, W., Razavi, S., 2016. Using a deep reinforcement learning agent for traffic signal control. arXiv preprint arXiv:1611.01142.
Ghanim, M.S., Abu-Lebdeh, G., 2015. Real-time dynamic transit signal priority optimization for coordinated traffic networks using genetic algorithms and artificial neural networks. J. Intell. Transp. Syst. 19 (4), 327–338.
Hegyi, A., De Schutter, B., Hellendoorn, H., 2005. Model predictive control for optimal coordination of ramp metering and variable speed limits. Transp. Res. C 13 (3), 185–209.
Khamis, M.A., Gomaa, W., 2012. Enhanced multiagent multi-objective reinforcement learning for urban traffic light control. In: 2012 11th International Conference on Machine Learning and Applications, Vol. 1. IEEE, pp. 586–591.
Konda, V.R., Tsitsiklis, J.N., 2000. Actor-critic algorithms. In: Advances in Neural Information Processing Systems. pp. 1008–1014.
Krajzewicz, D., Erdmann, J., Behrisch, M., Bieker, L., 2012. Recent development and applications of SUMO-simulation of urban mobility. International Journal on Advances in Systems and Measurements 5 (3&4).
Li, Z., Wu, Q., Yu, H., Chen, C., Zhang, G., Tian, Z.Z., Prevedouros, P.D., 2019. Temporal-spatial dimension extension-based intersection control formulation for connected and autonomous vehicle systems. Transp. Res. C 104, 234–248.
Liu, H.X., Wu, X., Ma, W., Hu, H., 2009. Real-time queue length estimation for congested signalized intersections. Transp. Res. C 17 (4), 412–427.
Lowe, R., Wu, Y.I., Tamar, A., Harb, J., Abbeel, O.P., Mordatch, I., 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In: Advances in Neural Information Processing Systems. pp. 6379–6390.
Mannion, P., Duggan, J., Howley, E., 2016. An experimental review of reinforcement learning algorithms for adaptive traffic signal control. In: Autonomic Road Transport Support Systems. Birkhäuser, Cham, pp. 47–66.
Mousavi, S.S., Schukat, M., Howley, E., 2017. Traffic light control using deep policy-gradient and value-function-based reinforcement learning. IET Intell. Transp. Syst. 11 (7), 417–423.
Nowé, A., Vrancx, P., De Hauwere, Y.M., 2012. Game theory and multi-agent reinforcement learning. In: Reinforcement Learning. Springer, Berlin, Heidelberg, pp. 441–470.
Odeh, S.M., Mora, A.M., Moreno, M.N., Merelo, J.J., 2015. A hybrid fuzzy genetic algorithm for an adaptive traffic signal system. Adv. Fuzzy Syst.
Prashanth, L.A., Bhatnagar, S., 2010. Reinforcement learning with function approximation for traffic signal control. IEEE Trans. Intell. Transp. Syst. 12 (2), 412–421.
Prashanth, L.A., Bhatnagar, S., 2011. Reinforcement learning with average cost for adaptive control of traffic lights at intersections. In: 2011 14th International IEEE Conference on Intelligent Transportation Systems. ITSC, IEEE, pp. 1640–1645.
Qie, H., Shi, D., Shen, T., Xu, X., Li, Y., Wang, L., 2019. Joint optimization of multi-UAV target assignment and path planning based on multi-agent reinforcement learning. IEEE Access 7, 146264-146272.
Shou, Z., Di, X., Ye, J., Zhu, H., Zhang, H., Hampshire, R., 2020. Optimal passenger-seeking policies on E-hailing platforms using Markov decision process and imitation learning. Transp. Res. C 111, 91–113.
Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M., 2014. Deterministic policy gradient algorithms. June.
Taylor, M.E., Stone, P., 2009. Transfer learning for reinforcement learning domains: A survey. J. Mach. Learn. Res. 10 (7).
Thorpe, T.L., Anderson, C.W., 1996. Tra c Light Control using Sarsa with Three State Representations. Technical Report, Citeseer.

Treiber, M., Hennecke, A., Helbing, D., 2000. Congested traffic states in empirical observations and microscopic simulations. Phys. Rev. E 62 (2), 1805.
USDOT, 2017. Adaptive Signal Control Technology. United States Department of Transportation Center for Accelerating Innovation.
van de Weg, G.S., Vu, H.L., Hegyi, A., Hoogendoorn, S.P., 2018. A hierarchical control framework for coordination of intersection signal timings in all traffic regimes. IEEE Trans. Intell. Transp. Syst. 20 (5), 1815–1827.
Varaiya, P., 2013. Max pressure control of a network of signalized intersections. Transp. Res. C 36, 177–195.
Wang, R.E., Everett, M., How, J.P., 2020. R-maddpg for partially observable environments and limited communication. arXiv preprint arXiv:2002.06684.
Wang, S., Huang, W., Lo, H.K., 2019. Traffic parameters estimation for signalized intersections based on combined shockwave analysis and Bayesian network. Transp. Res. C 104, 22–37.
Webster, F.V., 1958. Traffic Signal Settings, No. 39.
Wei, H., Xu, N., Zhang, H., Zheng, G., Zang, X., Chen, C., Li, Z., 2019. Colight: Learning network-level cooperation for traffic signal control. In: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, pp. 1913–1922, November.
Wen, K., Qu, S., Zhang, Y., 2007. A stochastic adaptive control model for isolated intersections. In: 2007 IEEE International Conference on Robotics and Biomimetics. ROBIO, IEEE, pp. 2256–2260.
Xiong, Y., Zheng, G., Xu, K., Li, Z., 2019. Learning traffic signal control from demonstrations. In: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, pp. 2289–2292, November.
Zhang, R., Ishikawa, A., Wang, W., Striner, B., Tonguz, O.K., 2020. Using reinforcement learning with partial vehicle detection for intelligent traffic signal control. IEEE Trans. Intell. Transp. Syst..
Zhang, R., Leteurtre, R., Striner, B., Alanazi, A., Alghafis, A., Tonguz, O.K., 2019. Partially detected intelligent traffic signal control: Environmental adaptation. In: 2019 18th IEEE International Conference on Machine Learning and Applications. ICMLA, IEEE, pp. 1956–1960.