The Twenty-Sixth International Symposium on Artificial Life and Robotics 2021 (AROB 26th 2021),
The Sixth International Symposium on BioComplexity 2021 (ISBC 6th 2021),
ONLINE, January 21-23, 2021

Generation of Traffic Flows in Multi-Agent Traffic Simulation with Agent Behaviour Model based on Deep Reinforcement Learning

Junji Zhong[1,2,*], Hiromitsu Hattori[1]

[1]Ritsumeikan University, Japan
[2]Dalian University of Technology, China

[*]is0541hx@ed.ritsumei.ac.jp

Abstract: In multi-agent based traffic simulation, agents are always supposed to move following existing instructions, and mechanically and unnaturally imitate human behaviours. The human drivers perform acceleration or deceleration irregularly all the time, which seems unnecessary in some conditions. For letting agents in traffic simulation behave more like humans and recognize other agents' behaviours in complex conditions, we propose a unified mechanism for agents learn to decide various accelerations by using deep reinforcement learning based on a combination of regenerated visual images revealing some notable features, and numerical vectors containing some important data such as instantaneous speed. By handling batches of sequential data, agents are enabled to recognize surrounding agents' behaviour and decide their own acceleration. In addition, we can generate a traffic flow behaving diversely to simulate the real traffic flow by using an architecture of fully decentralized training and fully centralized execution without violating Markov assumptions.

Keywords: multi-agent system, microscopic traffic simulation, deep reinforcement learning, physical acceleration model, agent behaviour simulation

1 INTRODUCTION

Traffic simulation provides perspective and approximate data about traffic flow in a certain region by constructing imitated models. Multi-agent simulation has been a promising approach to conduct traffic simulation based on a microscopic manner[1, 2]. Although multi-agent-based traffic simulation has been widely studied for over decades, it sometimes does not work very well or performs unnaturally since behavior models assigned to agents tend to be simple or abstracted ones based on theories and knowledge from existing literature. For instance, in some cases, agents to imitate vehicles are constructed as entities moving under predefined specific rules to reproduce stereotyped driving behaviors. The accumulation of such driving behaviors sometimes results in unnatural traffic flows and would provide false perspectives to investigate real-life traffic. The agent should be constructed to enable flexible driving behaviors according to the surrounding situations.

when an agent is driving on the road in ordinary traffic conditions, it will encounter other agents which can appear from anywhere, and the situations can become extremely intractable. Apparently, the human drivers control cars to drive at various speeds depending on complex conditions. And the value of speeds should to be continuous. It is obviously impossible for designers to complete every specific rule in every possible situation to control the vehicles continuously.

In this paper, we make use of a Deep Reinforcement Learning (DRL) algorithm, Actor Critic [3], and an architecture of fully decentralized training and fully centralized execution to enable agents recognize other agents' behaviours and decide their own speeds.

DRL algorithms are used in several large scale control fields like recommender systems [4]. Traffic simulation are supposed to be handle as multi-agent system. The architectures of multi-agent reinforcement learning (MARL) have already been applied for researches [5]: (1) Fully decentralized. (2) Fully centralized. (3) Centralized training with decentralized execution. (4) Model of other agents. However, these architectures have some features which are not suitable to large scale simulation like sharing large data among agents. Furthermore, Single reinforcement algorithm like Deep Q Learning (DQL) [6] applied to multi-agent system suffers from the problem violating Markov assumptions required for convergence [7] and hence we use an architecture of fully decentralized training and fully centralized execution which are possible to apply for large scale simulation by the approach called repeated and partial training.

Human drivers decide their accelerations by observing the surrounding cars and the relative speed. We extract the agents to which the learning agent should pay attention, and then regenerate images with the learning agent at the centre to be the visual inputs. Here are also numerical vectors to be numerical inputs containing some important data such as instantaneous speed. The approach called repeated and partial training in simulation allocate models to agents and can generate diverse traffic flow in the frame of fully decentralized

©ISAROB

training and fully centralized execution. This model will not violate Markov assumptions and provides reliable traffic flow in which agents behave variously and more naturally.

2 RELATED WORKS

In traffic simulation, previous researches use DRL models to simulate specific conditions, such as prompting the agents to navigate through an intersection. In Volodymyr Mnih's work [6], they use DQL for agents to decide the time to start and whether accelerate or decelerate. But DQL can only make discrete choices instead of outputting some continuous values like acceleration. In Giulio Bacchiani's work [8], they use the Actor Critic algorithm and give the same images about a certain scenario like roundabout for within 6 agents to decide acceleration while avoiding collision. It makes sense that they use a sequence of visual and numerical inputs which will be easier for agents to recognise other agents' acceleration. But in larger simulation, there will be more agents and situations will become extremely complex because the other agents can unpredictably appear from anywhere.

The relationship among agents in multi-agent system is considered to be cooperative, competitive or mixed cooperative competitive and so on. The single agent RL algorithm, like DQL or policy gradient (PG) [10] Algorithms, can't converge since the environment becomes non-stationary and all agents' policies are changing as we discuss in Section 3.3. In Ryan Lowe's work [7], they use the architecture of centralized training with decentralized execution and the algorithm called multi-agent deep deterministic policy gradient (MADDPG). The MADDPG makes agents to learn to cooperate and compete by inputting the actions of all agents a_n and the set of all observations into an original Deep Deterministic Policy Gradient (DDPG) network [11]. Although it doesn't need to share the information during executing and works well in their experiments including no more than five agents, it is impossible to share huge amount of information especially the set of all observations when too many agents in one training. For avoid sharing information, the architecture model of other agents, like in Natasha Jaques's work [12], agents use probability model to sample other agents' actions and choose actions which infer the causality of counterfactual actions. It works without being trained in centralization and sharing huge data. Especially, they use LSTM [13], a kind of recurrent neural network, to handle sequential data in a different way with this paper. There is a restriction that agents affect others irreversibly, thus can avoid circular dependency. But simulation is synchronous rather than asynchronous.

In simulation,it sometimes doesn't need to concern about the relationship among the agents if all the agents act within a certain range which can guarantee the efficiency of simulation and the environment can become relatively stable. On the other hand, it is necessary for agents to recognize other agents' behaviours. Since simulation executes not in real time, we can decide all agents' actions at every cycle centrally. In addition, extracting surrounding agents as features appropriately can let agents learn to recognise others efficiently. And finally, the agents can drive safe and efficiently in diverse conditions. In this paper, through taking advantage of these features of simulation, we use the architecture of fully decentralized training and fully centralized execution, and use an approach called repeated and partial training to generate diverse traffic flow.

3 BACKGROUND

3.1 Reinforcement Learning

In this paper, we make use of partially observable Markov games based on Markov decision processes (MDPs) [9], in which the agents are supposed to learn patterns only from their own partial observations. There will be N agents in an environment, and at time t the agents will receive their own observation $o_t^1, o_t^2 ... o_t^n$. After take actions $a_t^1, a_t^2 ... a_t^n$ on observations and their policies $\pi_a^1, \pi_a^2 ... \pi_a^n$, the agents will get their time t's reward $r_t^1, r_t^2 ... r_t^n$. For agent i, the MDPs can be described as:

$$o_t^i \rightarrow a_t^i | \pi(a_t^i | o_t^i, \pi_a^1, \pi_a^2 ... \pi_a^n) \rightarrow o_{t+1}^i | P(o_{t+1}^i | o_t^i, a_t^i) \quad (1)$$

3.2 Actor Critic

Agents are supposed to learn to maximize their reward $r_t^1, r_t^2 ... r_t^n$ in reinforcement learning. Here we use actor critic algorithm for agents learning.

For agent $1, 2...n$ in time t, the set of actions $a_t^1 | \pi(a_t^1 | o_t^1), a_t^2 | \pi(a_t^2 | o_t^2) ... a_t^n \pi(a_t^n | o_t^n)$ given by the actor $\pi_a^1, \pi_a^2 ... \pi_a^n$ will be criticised by the critic on the critics' parameters $\pi_c^1, \pi_c^2 ... \pi_c^n$. The γ is a discount factor of total rewards R of a whole trajectory τ from time t:

$$R = \sum_{T=t}^{\tau} \gamma^{'-t} r_t^i \quad (2)$$

For agent i to maximize R, the R's gradient of the π_a^i policy $\nabla_{\pi_a^i} J$ and the critics' parameters π_c^i's loss L_{Critic} need to be minimized according to these equations:

$$advantage = r_t^i + \gamma V^{\pi_c^n}(o_{t+1}^i) - V^{\pi_c^i}(o_t^i) \quad (3)$$

$$\nabla_{\pi_a^i} J = -\frac{1}{N} \sum_t [advantage \nabla_{\pi_a^i} log P(a_t^i)|_{o_t^i}] \quad (4)$$

$$\mathscr{L}_{Critic} = \frac{1}{N} \sum_t [advantage]^2 \quad (5)$$

The actor critic algorithm uses deep learning approach to approximate the values such as the value of state $V^{\pi_c^i}(o_t^i)$ and minimize the loss and gradient. Therefore, there will be an

action network and a critic network for every learning agent during the training time.

3.3 Markov Assumption

Single reinforcement algorithm like DQL or PG applied to multi-agent system will suffer the problem violating Markov assumptions required for convergence. Here is an example using the PG for MARL: for agent i, the objective function based on the expectation of sate value function to get optimal policy π_a^i[14]:

$$\arg\max_{\pi_a^i} J^i(\pi_a^1, \pi_a^2 ... \pi_a^n) = \mathbb{E}[V^{\pi_c^i}(o_t^n; \pi_a^1, \pi_a^2 ... \pi_a^n)] \quad (6)$$

The value function depends on observation and all agents' policies, even if agent i gets its optimal policy π_a^{i*}, since the other agents are updating their policies $\pi_a^1, ...\pi_a^{j\neq i} ...\pi_a^n$ independently, the π_a^{i*} will no longer be the optimal policy during the training time. As a result, all the agents update policies independently, the model may never converge. In this paper, we use the approach, repeated and partial training to tackle the problem as explained in Section 4.4.

3.4 Physical Model

The simulation time is discrete, we set two kinds of speed in order to describe the continuous movement of agents: the instantaneous speed v_{It}^i. which is continuous and used by the agent to decide acceleration and get reward, while the average speed v_{At}^i is used for simulating and detecting the collision. The physical quantities are computed in accordance with the laws of physics:

$$v_{It}^i = v_{It-1}^i + a_t^i t \quad (7)$$

$$v_{At}^i = (v_{It}^i t + 0.5 a_t^i t^2)/T \quad (8)$$

The time unit T in simulation is set to be $T = 1s$ while the executing time t is not always be $1s$ since the minimum of speed should be $0m/s$ and agent cannot reverse.

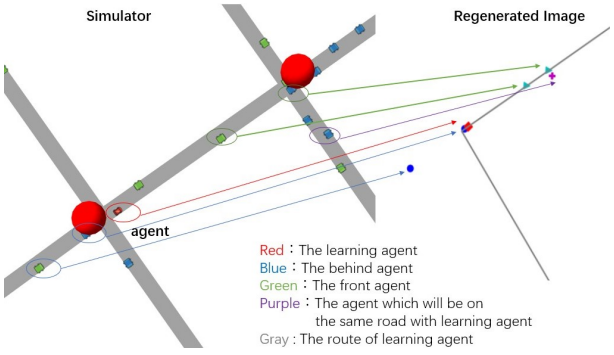


Fig. 1. The process of regenerating image

4 METHODS

4.1 Visual and Numerical Input

The agents' partial observations $o_t^1, o_t^2, ...o_t^n$ consist of visual and numerical inputs which are considered to be mixed inputs in DRL. The human drivers decide their accelerations by observing the surrounding cars and analyzing the relative speed. While for agents, it is too complex to recognize other agents' accelerations when there are so many agents in a simulation. Thus, for complexity reduction, we extract the agents which the learning agent should pay attention to, and then regenerate images with the learning agent at the center. In our instance, for the learning agent to recognize, these agents will be extracted:

1) Front agents on the same road with the learning agent
2) Behind agents on the same road with the learning agent
3) Agents will be on the same road with the learning agent

We use different color to label them and regenerate a 500*500 pixel picture for every observation o_t^n with appropriate proportion. An instance of this process is shown in Fig.1..

In addition, we set numerical vectors containing important data from three aspects about instantaneous speed v_{It}^i, target speed v_{Target} and the distance between with the front agent.

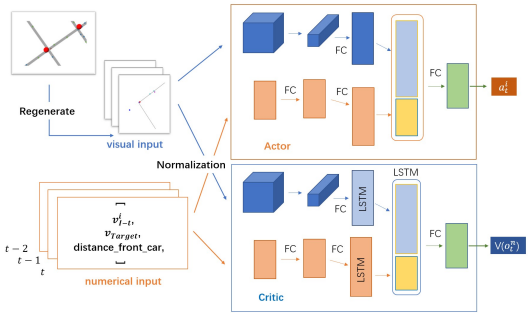


Fig. 2. The architecture of network

4.2 Use LSTM to Tackle the Sequential Data

It's notable that the data given to agents as observations are changing continuously instead of randomly distributed. The instantaneous speed v_{It}^i is affected by actions a_{t-1}^i while the target speed v_{Target} is a constant value. As well as the images which show sequence of agents' movements and indicate accelerations. In this proposal, we use LSTM to tackle batches of sequential data by using time $t-3 \sim t$ the three groups data once. The batch of images and vectors enable agents to figure out the surrounding agents' accelerations and decide their own accelerations more efficiently.

4.3 Network Architecture

The batch of images and vectors will be processed respectively and then combined together while the three layers of LSTM will be allocated correspondingly. For pushing agents do exploration, the output of actor network will obey Gaussian distribution. The mean and variance of Gaussian distribution are the values which the agents are also supposed to learn. The architecture of the learning agent in the training time is represented in Fig.2..

4.4 Dynamic Reward Shaping

In training time, the dynamic reward shaping conducts agents to learn to reach the target speed and slow down to avoid collision. The dynamic rewards will be given specifically for possible conditions, and the restrictions can be relatively flexible to suit the extremely complex situations. Additionally, terminal rewards for collisions but no penalty of jam for simplifying the inputs and allowing agents not think overwhelmingly.

$$r_t^i = r_{terminal}^i + r_{speed}^i \tag{9}$$

$$r_{terminal}^i = -0.1, \quad whencollision \tag{10}$$

The acceleration of agents is constricted to within $a_t^i \in [-8m/s^2 \sim 6m/s^2]$, so it costs time to achieve the target speed v_{Target} when agents start or decelerate if there are agents in front. We set a $4s$ buffer time called T_{buffer} that occurs when agents start or the distance between with front agents below safe distance. The buffer time T_{buffer} decays in normal time. So the speed reward r_{speed}^i will be given as:

$$r_{speed}^i \begin{cases} v_{It}^i < v_{Target} \begin{cases} T_{buffer} : 0.001 \times v_{It}^i / v_{Target} \\ 0.001 - 0.002(v_{Target}^i - v_{It}^i)/v_{Target}) \end{cases} \\ v_{It}^i \geq v_{Target} : (0.001 - 0.004(v_{It}^i - v_{Target})/v_{Target}) \end{cases} \tag{11}$$

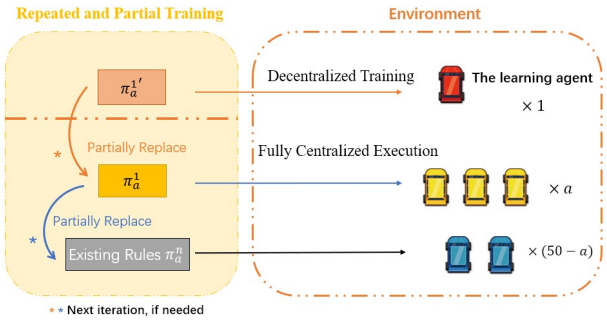


Fig. 3. The architecture of fully decentralized training and fully centralized execution

4.5 Repeated and Partial Training

As the discussion in 3.3, the model may not converge when all agents are updating policies independently and thus the environment becomes unstable. The approach we use called repeated and partial training, which means holding only one agent to sample data in environment while other agents don't update policy until the only one learning agent gets its optimal policy. Then the optimal policy will be allocated to all of or a part of all agents and get into the next training iterations. So the objective function changes along the training step as:

$$\begin{aligned} &\arg\max_{\pi_a^i} J^i(\pi_a^1, \pi_a^1 ... \pi_a^i ... \pi_a^n, \pi_a^n) \\ &= \mathbb{E}[V^{\pi_c^i}(o_t^n; \pi_a^1, \pi_a^1 ... \pi_a^i ... \pi_a^n, \pi_a^n)] \end{aligned} \tag{12}$$

$$\begin{aligned} &\arg\max_{\pi_a^{i'}} J^i(\pi_a^{1'}, \pi_a^{1'} ... \pi_a^{i'} ... \pi_a^{n'}, \pi_a^{n'}) \\ &= \mathbb{E}[V^{\pi_c^{i'}}(o_t^n; \pi_a^{1'}, \pi_a^{1'} ... \pi_a^{i'} ... \pi_a^{n'}, \pi_a^{n'})] \end{aligned} \tag{13}$$

$$\vdots$$

$$\begin{aligned} &\arg\max_{\pi_a^{i(N)}} J^i(\pi_a^{1(N)}, \pi_a^{1(N)} ... \pi_a^{i(N)} ... \pi_a^{n(N)}, \pi_a^{n(N)}) \\ &= \mathbb{E}[V^{\pi_c^{i(N)}}(o_t^n; \pi_a^{1(N)}, \pi_a^{1(N)} ... \pi_a^{i(N)} ... \pi_a^{n(N)}, \pi_a^{n(N)})] \end{aligned} \tag{14}$$

At all time, only the learning agent i's policy $\pi_a^i, \pi_a^{i'} ... \pi_a^{i(N)}$ will be updated and then allocated to other agents, so it's possible for agents which act in the similar way to get their relatively optimal policy if designers think the result is adaptable in simulation or other fields.

In our paper, this approach can be fitted into the frame of fully decentralized training and fully centralized execution and represented in the Fig.3.. Different agents will execute based on different policies centrally in simulation, but there is only keeping one learning agent to sample data at all time. Detailed description of instance is offered in Section 5.3.

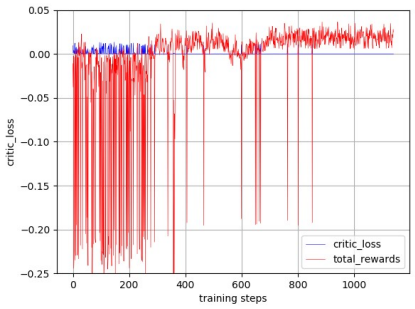


Fig. 4. The learning curve of initial training

5 EXPERIMENT

5.1 Settings

In the experiment, we adapt GAMA [15] as simulator to build environment for simulation. The agents are assigned to move from a random starting point to another random goals, and they have to face countless situations during this process.They are expected to reach the goal and achieve the target speed without colliding with other agents.

The minimum distance is set to be 0 so agents can move sticking others. All the agents' target speed is $v_{Target} = 16m/s$, and the acceleration of the learning agents is constricted within $a_t^i \in [-8m/s^2, 6m/s^2]$. There are 8 nodes as start point and 12 nodes as goal point for agents, and totally about 50 agents in the environment making the environment seems busy and giving enough opportunities to encounter others. Furthermore, agents can't achieve the target speed v_{Target}, so we can observe their behaviours in low speeds.

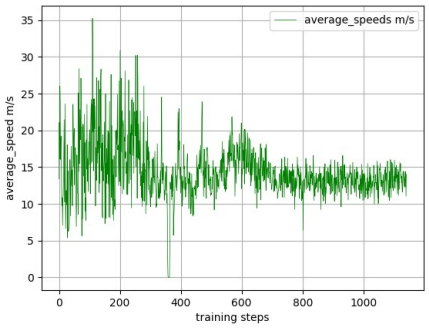


Fig. 5. The change of average speed in initial training

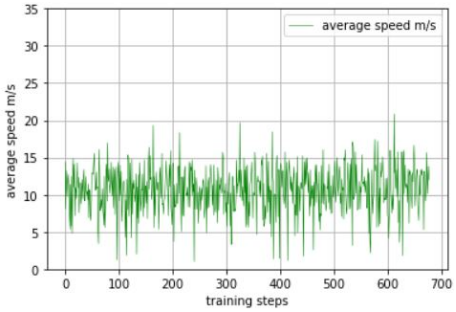


Fig. 6. The average speed of the learning agent in subsequent training

5.2 Initial Training

At first, we set only one learning agent to sample data and update its policy π_a^1, while other 50 agents move based on existing rules π_a^n as the objective function for the learning agent:

$$\arg\max_{\pi_a^1} J^i(\pi_a^1, \overbrace{\pi_a^n \ldots \pi_a^n}^{50}) = \mathbb{E}[V^{\pi_c^1}(o_t^n; \pi_a^1, \overbrace{\pi_a^n \ldots \pi_a^n}^{50})] \quad (15)$$

The Fig.4. shows the learning curve of learning agent that the agent can get its optimal policy within 1000 training steps in which the rate of collision can be controlled under 1%.

We can see the agent can learn to control its acceleration which leads the instantaneous speed v_{It}^i to get closer to target speed v_{Target} from the Fig.5. which reveals the change of learning agent's average instantaneous speed. Here the average instantaneous speed is above $13m/s$.

5.3 Subsequent Training

The frame of fully decentralized training and fully centralized execution is used in the same way as the initial training. We set only one learning agent to sample data and update its policy $\pi_a^{1'}$. The previously trained weight π_a^1 is allocated to 30 agents and they will execute centrally while another 20 agents still move based on π_a^n as:

$$\arg\max_{\pi_a^{1'}} J^i(\pi_a^{1'}, \overbrace{\pi_a^1 \ldots \pi_a^1}^{30}, \overbrace{\pi_a^n \ldots \pi_a^n}^{20}) \quad (16)$$
$$= \mathbb{E}[V^{\pi_c^{1'}}(o_t^n; \pi_a^{1'}, \overbrace{\pi_a^1 \ldots \pi_a^1}^{30} \ldots \overbrace{\pi_a^n \ldots \pi_a^n}^{20})]$$

The Fig.3. illustrates the architecture of fully decentralized training and fully centralized execution. In this time, the learning agent is allowed to update the π_a^1 to $\pi_a^{1'}$ and set a small learning rate about 1×10^{-6}. The result comes out to a broader speed range of the learning agent showed in Fig.6..

In this experiment, the average speed is lower than the one in initial training. It makes sense since the agents move based on policy π_a^1 and $\pi_a^{1'}$ can keep a distance from others and make the road more crowded and complex. Agents can't achieve the target speed v_{Target} , so we can observe their behaviours in low speeds. There are three patterns of agents: in Fig.7. the agents moving based on existing rules π_a^n (blue and green) stick with others; the agents moving based on policy π_a^1 and $\pi_a^{1'}$ can keep a distance from others (yellow and red).

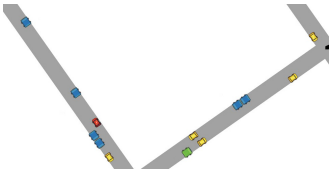
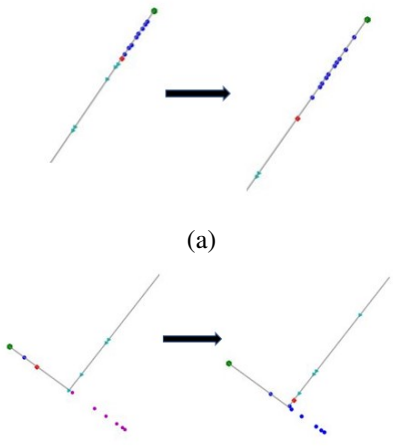


Fig. 7. The behaviours of agent in subsequent training

Furthermore, we found the learning agent move based on policy $\pi_a^{1'}$ can learn to move safe in some difficult conditions. The Fig.7. reveals observations of learning agent (red) which suggests it can wait and crawl in a queue and also can catch a chance to cut in line. Obviously, there are also other agents move based on policy π_a^1 in the queues.



(a)

(b)

Fig. 8. The learning agent is crawling in a queue (a) and cutting in line (b)

6 CONCLUSION

We generate traffic flow with agents which can recognize other agents' behaviour and decide own accelerations by using an architecture of fully decentralized training and fully centralized execution to avoid violating Markov assumptions. Agents' observations are composed of regenerated visual images and numerical vectors which enable the agents move safe in some difficult situations like crawling in a queue or cutting in line. By using the approach called repeated and partial training, we generate a very diverse traffic flow containing agents move based on policy or existing rules. As a result, we found that after training, agents move based on policy can keep a distance from front agents while agents move based on existing rules move sticking with others when roads are crowded.

Future works can be directed toward the more difficult situations like overtaking or sophisticated intersections. In addition, we are investigating parallel computing for fast simulation applied trained model.

7 ACKNOWLEDGMENTS

This research was partially supported by a Grant-in-Aid for Challenging Research (Exploratory) (19K21572, 2019-2021) from Japan Society for the Promotion of Science (JSPS).

REFERENCES

[1] Hattori H, Nakajima Y and Yamane S (2011), Massive Multiagent-Based Urban Traffic Simulation with Fine-Grained Behavior Models. Journal of Advanced Computational Intelligence and Intelligent Informatics, vol. 15, no. 2, pp. 233–239.

[2] M.Schaefer, J.Vokřínek, D.Pinotti, and F.Tango (2016), Multi-agent traffic simulation for development and validation of autonomic car-to-car systems, in Autonomic Road Transport Support Systems, pp. 165–180,Springer.

[3] V.Mnih, A.P.Badia, M.Mirza, A.Graves, T.Lillicrap, T.Harley, D.Silver, and K.Kavukcuoglu(2016), Asynchronous methods for deep reinforcement learning, in International conference on machine learning, pp. 1928–1937.

[4] X.Chen, S.Li, H.Li, S.Jiang, Y.Qi, and L.Song (2019), Generative adversarial user model for reinforcement learning based recommendation system," in International Conference on Machine Learning, pp. 1052–1061, PMLR.

[5] J.Foerster,I.A.Assael,N.DeFreitas,andS.Whiteson(2016), Learning to communicate with deep multi-agent reinforcement learning, in Advances in neural information processing systems, pp. 2137–2145.

[6] V.Mnih, K.Kavukcuoglu, D.Silver, A.A.Rusu, J.Veness, M.G.Bellemare, A. Graves, M.Riedmiller, A.K.Fidjeland, G.Ostrovski, et al.(2015), Human-level control through deep reinforcement learning, nature, vol. 518, no. 7540, pp. 529–533.

[7] R.Lowe, Y.I.Wu, A.Tamar, J.Harb, O.P.Abbeel, and I.Mordatch (2017), Multi-agent actor-critic for mixed cooperative-competitive environments, in Advances in neural information processing systems, pp. 6379–6390.

[8] G. Bacchiani, D. Molinari, and M. Patander (2019), Microscopic traffic simulation by cooperative multi-agent deep reinforcement learning, in Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, pp.1547–1555.

[9] M.L.Littman (1994), Markov games as a framework for multi-agent reinforcement learning, in Machine learning proceedings 1994, pp. 157–163, Elsevier.

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

[11] T.P.Lillicrap, J.J. Hunt, A.Pritzel, N.Heess, T.Erez, Y.Tassa, D.Silver,and D.Wierstra (2015), Continuous control with deep reinforcement learning, arXiv preprint arXiv:1509.02971.

[12] N.Jaques, A.Lazaridou, E.Hughes, C.Gulcehre, P.Ortega, D.Strouse,J.Z.Leibo, and N.DeFreitas (2019), Social influence as intrinsic motivation for multi-agent deep reinforcement learning," in International Conference on Machine Learning, pp.3040–3049, PMLR.

[13] S.Hochreiter and J.Schmidhuber (1997), Long short-term memory, Neural computation, vol. 9, no. 8, pp. 1735– 1780.

[14] M.Tan (1993), "Multi-agent reinforcement learning: Independent vs. cooperative agents," in Proceedings of the tenth international conference on machine learning, pp. 330– 337.

[15] P.Taillandier (2014), "Traffic simulation with the gama platform," in Eighth International Workshop on Agents in Traffic and Transportation, pp. 8–p.