Expert Systems With Applications 176 (2021) 114896

Contents lists available at ScienceDirect

Expert Systems With Applications

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

A deep reinforcement learning-based method applied for solving multi-agent defense and attack problems

Liwei Huang [a], Mingsheng Fu [a,b,*], Hong Qu [a], Siying Wang [a], Shangqian Hu [a]

[a] School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China
[b] School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore

ARTICLE INFO

Keywords:
Multi-agent cooperation
Defense and attack
Deep reinforcement learning
Multi-agent reinforcement learning

ABSTRACT

Learning to cooperate among agents has always been an important research topic in artificial intelligence. Multi-agent defense and attack, one of the important issues in multi-agent cooperation, requires multiple agents in the environment to learn effective strategies to achieve their goals. Deep reinforcement learning (DRL) algorithms have natural advantages dealing with continuous control problems especially under situations with dynamic interactions, and have provided new solutions for those long-studied multi-agent cooperation problems. In this paper, we start from deep deterministic policy gradient (DDPG) algorithm and then introduce multi-agent DDPG (MADDPG) to solve the multi-agent defense and attack problem under different situations. We reconstruct the considered environment, redefine the continuous state space, continuous action space, reward functions accordingly, and then apply deep reinforcement learning algorithms to obtain effective decision strategies. Several experiments considering different confrontation scenarios are conducted to validate the feasibility and effectiveness of the DRL-based methods. Experimental results show that through learning the agents can make better decisions, and learning with MADDPG achieves superior performance than learning with other DRL-based models, which also explains the importance and necessity of mastering other agents' information.

1. Introduction

Multi-agent system can be defined as a group of autonomous, interacting entities sharing a common environment, which they perceive with sensors and upon which they act with actuators (Bu et al., 2008). Multi-agent cooperative control provides a promising paradigm for studying how agents can learn coordinated behaviors in multi-agent systems (Yu et al., 2015), and it is finding increasing applications in a wide range of real-world domains, such as robotics (Kim and Chung, 2006), distributed control (Goldman and Zilberstein, 2004), and resource management (Galstyan et al., 2005).

Multi-agent control has been an active research area for the past few years, and there has been numerous works that study multiple agents control problem. An efficient algorithm based on pseudo spectral (PS) and level set (LS) was presented in Robinson et al. (2018) to generate optimal trajectories for multiple vehicles in complex non-convex maze-like environments. Kantaros and Zavlanos (2016) proposed a distributed, hybrid control scheme which resulted in servicing a collection of tasks in complex environments by the available robots in a network. A novel k-degree smoothing method was put forward to smooth the initial trajectories obtained by improved ant colony optimization (IACO) of multiple unmanned aerial vehicles, and the cooperation scheme was also induced by k-degree smoothing (Huang et al., 2016). A co-evolutionary improved genetic algorithm (CIGA) was presented in Qu et al. (2013) for global path planning of multiple robots, which employed a co-evolution mechanism together with an improved genetic algorithm. A self organizing map-based approach which integrated the advantages and characteristics of biological neural systems was presented in Yi et al. (2017), and the approach was capable of dynamically planning the paths of a swarm robots in 3-D environments under uncertain situations. The authors proposed a novel approach to determine the optimal trajectory for multi-robots in a clutter environment using hybridization of improved particle swarm optimization (IPSO) with differentially perturbed velocity algorithm in Das et al. (2016). All those above-mentioned articles are conducted under various conditions, and when facing new environments, the environments must be re-modeled, and the proposed algorithms must be re-applied to obtain the final results, which results in the sensibility and inadaptability to new

* Corresponding author at: School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China.
E-mail addresses: liweihuang@uestc.edu.cn (L. Huang), fms@uestc.edu.cn (M. Fu), hongqu@uestc.edu.cn (H. Qu), wangsiying527@126.com (S. Wang), hushangq@163.com (S. Hu).

https://doi.org/10.1016/j.eswa.2021.114896
Received 28 June 2020; Received in revised form 23 October 2020; Accepted 7 March 2021
Available online 17 March 2021

environments.

Planning has long been studied in artificial intelligence, but research in reinforcement learning has brought a number of powerful innovation (Littman, 2015). Reinforcement learning (RL) (Sutton and Barto, 2018) has been widely applied to multi-agent systems. As a simple and classical reinforcement learning method, a Q-learning based approach was proposed to deal with traffic lights control problem which was modeled as a multi-agent system (Abdoos et al., 2011). In Park et al. (2001), the authors adopted modular Q-learning architecture which included learning modules and a mediator module to assign a proper action to an agent in the multi-agent system under robot soccer situation. Benefiting from the improvement of high performance computational hardware, deep neural networks-based reinforcement learning, namely deep reinforcement learning (DRL) shows great potential for solving complex multi-agent cooperative control problems. On the one hand, the natural way for finding policies for a multi-agent system is to learn decentralized value functions or policies directly and individually. Tan proposed to train independent state-action value functions for each agent through traditional Q-learning (Tan, 1993), and Tampuu et al. further extended this to deep neural networks by using DQN (Tampuu et al., 2017). However, learning independently affects the learning processes of other agents, and leads to instability of the environment. On the other hand, centralized learning by taking other agents' information into consideration can avoid the effects of the non-stationary factors. Chen et al. presented a multi-agent collision avoidance algorithm based on deep reinforcement learning which included a value network to encode the joint configuration with the neighbors (Chen et al., 2017). Moreover, with the help of actor critic (AC) architecture (Mnih et al., 2016), some works have presented the hybrid value-based and policy-based learning methods, as well as the methods with centralized training and decentralized execution. For the AC-based deep reinforcement learning, Lillicrap proposed the deep deterministic policy gradient (DDPG) algorithm (Lillicrap et al., 2015) to deal with the continuous control problem, as continuous control for multi-agents is very important and practical. The authors also put forward an adaption of actor-critic method (Lowe et al., 2017) which considered action policies of other agents and was able to successfully learn policies that required complex multi-agent coordination. Since then, a number of works based on RL or DRL have been proposed to deal with various multi-agent cooperation scenarios (Pendharkar and Cusatis, 2018; Foerster et al., 2018; Vinyals et al., 2019; Jaderberg et al., 2019; Li et al., 2019; Silva et al., 2019).

The RoboFlag competition and its modifications, as a classical multi-agent cooperative control issue, have been well studied in D'Andrea and Murray (2003), Earl and D'Andrea (2002a), Earl and D'Andrea (2002b), Earl and D'Andrea (2007). However, these methods are based on linear programming (LP), and can not effectively adapt to new environments. DRL-based approaches have natural advantages in solving continuous control problems, as their learning processes are the processes of continuous interactions with environments. In this paper, we propose to apply the DRL-based multi-agent cooperative control methods to deal with a simplified version of RoboFlag competition, which we call defense and attack in the rest of this paper. The contributions of this paper mainly include: (1) we reconstruct the multi-agent defense and attack environment on basis of the open-source Multi-Agent Particle Environment[1]; (2) we introduce deep reinforcement learning algorithms to address the considered problem, and redefine the state space, the action space and reward functions accordingly; (3) several different DRL-based learning models are employed to train both the attack agents and defense agents; (4) several experiments and comparisons are conducted to verify the effectiveness of the applied DRL models.

The rest of this paper is organized as follows. Section 2 is devoted to the problem statement and environment modeling. The new multi-agent cooperative planning based on DRL algorithms for defense and attack is given in Section 3. Several simulations and comparisons are conducted in Section 4. Conclusions and future work are drawn in the last section.

2. Problem formulation and environment modeling

In this section, we firstly introduce the multi-agent defense and attack problem, and then model the environment through Markov Decision Processes (MDP) to further prepare for the training of deep reinforcement learning algorithms.

2.1. Multi-agent defense and attack

In this paper, we consider a simplified version of the RoboFlag competition problem (D'Andrea and Murray, 2003), which we call multi-agent defense and attack. In the considered problem, there are two teams of robots, which are the defenders and attackers, and a circular region called defense zone in the playing field, as shown in Fig. 1.

Two sub-problems of the defense and attack issue are considered in this paper, which are described as follows.

(1) Multi-agent defense and attack with rule-based attackers.

In this sub-problem, the attackers perform their actions according to the given rules, and the defenders take actions based on the deep reinforcement learning model, namely ruled attackers versus intelligent defenders. Note that the rule-based defenders versus intelligent attackers situation also belongs to this sub-problem, since effective defense rules are much harder to design compared with the attack rules, we don't consider this situation in this paper.

(2) Multi-agent defense and attack with DRL-based attackers.

In this sub-problem, both attackers and defenders take actions through the deep reinforcement learning models, namely intelligent attackers versus intelligent defenders. Under this situation, the attackers cooperate to achieve their goal, so do the defenders.

2.2. Environment modeling

In multi-agent defense and attack problem, there are two kinds of agents in the considered problem with agent type $k \in \{ATK, DFS\}$, where *ATK* represents the attack agents, and *DFS* stands for the defense agents.

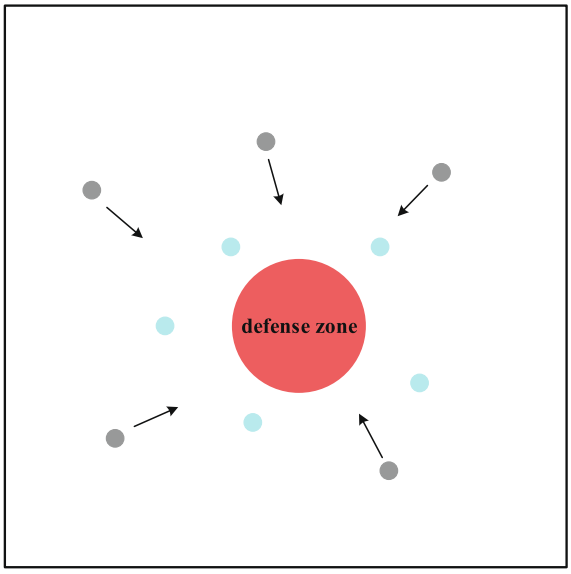

Fig. 1. The defense and attack playing field: the red circular region is the defense zone, the blue circles are the defenders, and the black circles are the attackers.

[1] The environments are publicly available: https://github.com/openai/multiagent-particle-envs.

All the intelligent agents learn their policies through interacting with the environment. We model the problem as Markov Decision Processes (MDP), and our goal is to maximize the accumulated discounted rewards of all agents. We adopt independent quintuple $(S_i^k, A_i^k, P_i^k, R_i^k, \gamma)$ to represent each agent i with type k, and the elements are explained respectively as follows.

(1)S_i^k: The state space, which is an infinite set and consists of all states of the agent.
(2)A_i^k: The action space, which contains all actions of the agent. It is also an infinite set.
(3)P_i^k: The state transition function $S_i^k \times A_i^k \times \widetilde{S_i^k} \to [0,1]$, where $P_i^k(s_i^k, a_i^k, \widetilde{s_i^k})$ is the transition probability from state s_i^k to state $\widetilde{s_i^k}$ by taking action a_i^k. For deterministic transition, $P_i^k(s_i^k, a_i^k, \widetilde{s_i^k}) \in \{0,1\}, \forall(s_i^k, a_i^k, \widetilde{s_i^k})$.
(4)R_i^k: The reward function, which indicates the immediate reward paid to agent with action a_i^k under state s_i^k.
(5)γ: The discount factor $0 \leqslant \gamma < 1$, which indicates the relative importance of future reward and current reward.

2.2.1. The state space

In the multi-agent defense and attack environment, the attack agents and defense agents have the same state representations. Let the set of all attack agents be AT, and the number of attack agents is $|AT| = N$. Similarly, let the set of all defense agents be DF, and the number of defense agents is $|DF| = M$.

In general, the features in the defense agents and attack agents are similar. For a given agent, attacker or defender, three kinds of information are involved in the state vector, which are (1) the information about the given agent itself, (2) the relative locations and velocities of the agents with the same type, and (3) the relative locations and velocities of the agents with the opponent type, respectively.

Specially, we denote $s_{at,i,t}$ as the state vector of attacker at_i at time instant t, which is defined as follows,

$$s_{at,i,t} = \left[self_{at,i,t}, ats_{at,i,t}, dfs_{at,i,t}\right], \tag{1}$$

where $self_{at,i,t}$ represents the features of at_i itself, $ats_{at,i,t}$ stands for the features of other attackers, $dfs_{at,i,t}$ stands for the features of defenders, and $s_{at,i,t}$ is the connection of these three feature vectors.

Similarly, the state vector $s_{df,j,t}$ for defender df_j at time instant t is defined as

$$s_{df,j,t} = \left[self_{df,j,t}, ats_{df,j,t}, dfs_{df,j,t}\right], \tag{2}$$

where $self_{df,j,t}$ represents the features of df_j itself, $ats_{df,j,t}$ stands for the features of attackers, $dfs_{df,j,t}$ stands for the features of other defenders, and $s_{df,j,t}$ is the connection vector of the three feature vectors.

The specific features of the components in (1) $self_{at,i,t}$, $ats_{at,i,t}$, and $dfs_{at,i,t}$ can be viewed in A.1. Likewise, the specific features of the components in (2) $self_{df,j,t}$, $ats_{df,j,t}$, and $dfs_{df,j,t}$ can be viewed in A.2.

2.2.2. The action space

In this paper, we consider continuous action space for the agents, and the action is expressed as a two-dimensional velocity vector $[vx, vy]$, which makes the attack and defense agents can move with a variable speed and in arbitrary direction.

In order to effectively simulate the motion behavior of actual agents, the agent model in this paper outputs a two-dimensional acceleration vector $[\widehat{ax}, \widehat{ay}]$ of the agent instead of the direct velocity vector. The corresponding velocity can be obtained through the following velocity formula.

$$\begin{cases} vx_{k,i,t} = vx_{k,i,t-1} + \widehat{ax}_{k,i,t} \times \Delta t, \\ vy_{k,i,t} = vy_{k,i,t-1} + \widehat{ay}_{k,i,t} \times \Delta t, \end{cases} \tag{3}$$

where $\widehat{ax}_{k,i,t}$ and $\widehat{ay}_{k,i,t}$ indicate the acceleration of the i-th agent with type k in x axis and y axis, respectively, and Δt represents the time interval.

Although $\widehat{ax}_{k,i,t}$ and $\widehat{ay}_{k,i,t}$ can be obtained directly through the network model, the direct values of $\widehat{ax}_{k,i,t}$ and $\widehat{ay}_{k,i,t}$ has problems in stability because of the overfitting problems Lowe et al. (2017). Therefore, to solve the problem of directly generating continuous actions, we adopt the strategy integration method to obtain $\widehat{ax}_{k,i,t}$ and $\widehat{ay}_{k,i,t}$, and the specific process is as follows.

Firstly we define a set $\widehat{A}$ with five basic policy actions, shown as

$$\widehat{A}_{k,i,t} = \left(\widehat{a}_{left,k,i,t}, \widehat{a}_{right,k,i,t}, \widehat{a}_{up,k,i,t}, \widehat{a}_{dwon,k,i,t}, \widehat{a}_{stop,k,i,t}\right), \tag{4}$$

where $\widehat{a}_{left,k,i,t} \in [0,1]$ is the acceleration of the left direction along x axis, $\widehat{a}_{right,k,i,t} \in [0,1]$ is the acceleration of the right direction along x axis, $\widehat{a}_{up,k,i,t} \in [0,1]$ is the acceleration of the up direction along y axis, $\widehat{a}_{down,k,i,t} \in [0,1]$ is the acceleration of the down direction along y axis, and $\widehat{a}_{stop,k,i,t} \in [0,1]$ is the acceleration to maintain still. To constrain the action values of basic strategy in $\widehat{A}_{k,i,t}$, we have $\widehat{a}_{left,k,i,t} + \widehat{a}_{right,k,i,t} + \widehat{a}_{up,k,i,t} + \widehat{a}_{down,k,i,t} + \widehat{a}_{stop,k,i,t} = 1$. Thus, $\widehat{ax}_{k,i,t}$ and $\widehat{ay}_{k,i,t}$ can be calculated by

$$\begin{cases} \widehat{ax}_{k,i,t} = \left(-\widehat{a}_{legt,k,i,t} + \widehat{a}_{right,k,i,t}\right) \times \alpha, \\ \widehat{ay}_{k,i,t} = \left(-\widehat{a}_{down,k,i,t} + \widehat{a}_{up,k,i,t}\right) \times \alpha. \end{cases} \tag{5}$$

where α is the sensitivity coefficient to constrain the range of acceleration. Therefore, the actual action output of the agent network is transformed into producing five values of the basic policy actions in set $\widehat{A}$.

2.2.3. Reward function

Reward function indicates the performance of the performed action under a specific situation. In this subsection, we will redefine the reward functions for both attack agents and defense agents to make the employed DRL algorithms more advantageous in solving the considered problem. To avoid the problem of sparse reward, we utilize the distance based reward functions.

For the attack agents, their goal is to reach the defense zone. Obviously, the closer the attack agents are to the defense zone, the greater the threat they have. Therefore, the reward of the attack agents should be inversely proportional to the distances between themselves and the defense zone. Then, the reward function of attack agent i at time instant t is defined as follows,

$$r_{at,i,t} = -\sqrt{x_{at,i}^2 + y_{at,i}^2}. \tag{6}$$

For the defense agents, their rewards are based on the distances also. Besides, the reward functions of defense agents need to take account of the attackers, and the reward function of defense agent j at time instant t is defined as

$$r_{df,j,t} = \lambda \widehat{r}_{df,j,t} + (1-\lambda)\overline{r}_{df,j,t}, \tag{7}$$

where there are two opposite terms considering the relationship between the attacker and the defense zone $\widehat{r}_{df,j,t}$, and the relationship between the attacker and the defender $\overline{r}_{df,j,t}$. Their definitions are shown in Eqs. (8) and (9), respectively.

$$\widehat{r}_{df,j,t} = \sum_{i=0}^{N} \sqrt{x_{at,i}^2 + y_{at,i}^2}. \tag{8}$$

In (8), we consider the distances between all attackers and the defense zone. If the attacker is far away from the defense zone, the defender will get a bigger reward. Note that if attacker at_i is captured by the defender, it will no longer be able to move forward to the defense zone, namely the $vx_{at,i}$ and $vy_{at,i}$ of at_i will be 0 after being captured. However, we still consider the position where it finally left off in (8) to avoid the phenomenon that the value of $\hat{r}_{df,j,t}$ becomes smaller when the attacker at_i is successfully defended.

$$\overline{r}_{df,j,t} = -\sqrt{\left(x_{at,l} - x_{df,j}\right)^2 + \left(y_{at,l} - y_{df,j}\right)^2}, \tag{9}$$

where l can be obtained through the following equation,

$$l = argmin_{i=0,N}\sqrt{\left(x_{at,i} - x_{df,j}\right)^2 + \left(y_{at,i} - y_{df,j}\right)^2}. \tag{10}$$

In (9), we consider the distance between the defender and the nearest attacker. Obviously, the smaller the distance between them, the better reward the defender should get. Therefore, $\overline{r}_{df,j,t}$ is inversely proportional to the distance between the defender and its nearest attacker. Here, for defender df_j, only the nearest attacker is considered to guide df_j to focus on a specific attacker, which can avoid df_j paying attention to multiple attackers at the same time, and can avoid df_j constantly swinging among different attackers.

3. Agent model and learning processes

In multi-agent defense and attack environment, multiple different agents can be trained through different models. In this paper, we classify our problem into two categories, one is single agent model and learning process, the other is multi-agent model and learning process. In single agent model, each agent only considers the information of itself, while in multi-agent model, it is necessary to consider the information of itself as well as other agents.

3.1. Single agent model and learning processes

We adopt traditional deep reinforcement learning algorithm to deal with the single agent model. In the considered problem, the action space should be continuous, thus, we apply deep deterministic policy gradient (DDPG) algorithm (Lillicrap et al., 2015) to train the single agent. Since there are similarities between the attack agents and defense agents in states and actions, the only difference is the reward function definition. Therefore, we take the attack agents as examples to further illustrate the applied model and learning algorithm. For the defense agents, we only need to replace the reward function.

3.1.1. The network structure of DDPG

DDPG is an Actor Critic (AC)-based DRL model. Compared with value-based DRL methods, such as DQN, the generation of action strategy and the evaluation of action are completed through two separate components in AC-based model. Whereas in DQN, the action selection strategy is determined by the predicted actions with the highest Q-value each time. Therefore, in AC architecture, there are two deep neural networks, one is responsible for generating strategy (policy network), and the other is for evaluating the strategy (Q network).

Thereby, for any attack agent at_i, there is a corresponding policy network $P_{at,i}$, and a corresponding Q network $Q_{at,i}$. The definitions of $P_{at,i}$ and $Q_{at,i}$ are displayed in (11) and (12), respectively.

$$\widehat{a}_{at,i,t} = P_{at,i}\left(s_{at,i,t}\middle|\theta_{at,i}\right), \tag{11}$$

where policy network $P_{at,i}$ generates action $\widehat{a}_{at,i,t}$ according to the state $s_{at,i,t}$. $\theta_{at,i}$ is the parameter that the policy network needs to learn. Note that $\widehat{a}_{at,i,t}$ is a continuous action, and we need to convert it into the actual action through strategy integration instead of sampling the actions.

$$q_{s_{at,i,t},a_{at,i,t}} = Q_{at,i}\left(s_{at,i,t}, a_{at,i,t}\middle|\widehat{\theta}_{at,i}\right), \tag{12}$$

where Q network generates the evaluation value $q_{s_{at,i,t},a_{at,i,t}}$ for state $s_{at,i,t}$ and action $\widehat{a}_{at,i,t}$ obtained by policy network $P_{at,i}$. $\widehat{\theta}_{at,i}$ is the parameter that the Q network needs to learn. Note that in DDPG, the polices and evaluations of different agents are independent, which means the policy and evaluation of each agent are determined only by its own state and action.

In this paper, the policy network and Q network are constructed by multi-layer perception (MLP), and the architectures are shown in Fig. 2. The policy network has one input layer, two hidden layers and one output layer. Both the input layer and output layer are fully connected (FC) networks, which is defined in (13). In each hidden layer, there are 64 neurons with ReLu as the activation function, and ReLu is defined in (14). The output layer has 5 neurons which represents the five basic actions, and we adopt the Softmax activation function to assure the sum of the actions' coefficients is 1. The definition of Softmax is displayed in (15).

$$y = \sigma(Wx + b), \tag{13}$$

where x represents the input of the network, y stands for the output of the network, W is the parameter matrix of the FC layers to be learned, b is the bias vector to be learned, and σ is the activation function.

$$ReLu\left(x\right) = \begin{cases} x, & if \quad x > 0, \\ 0, & if \quad x \le 0. \end{cases} \tag{14}$$

$$softmax(X)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}, \tag{15}$$

where $softmax(X)_j$ represents the j-th output of all K outputs.

The architecture of Q network is similar to policy network, but the input and output are slightly different. In Q network, the input is the connection vector of state $s_{at,i,t}$ and action $\widehat{a}_{at,i,t}$ generated by policy network. We do not use neurons to represent the output (None) to ensure that the range of evaluation value is in real number field.

3.1.2. Learning processes of DDPG

Since DDPG has two separate networks, we will illustrate the learning processes of the two networks correspondingly. The learning process of Q network is based on Bellman function, which describes the

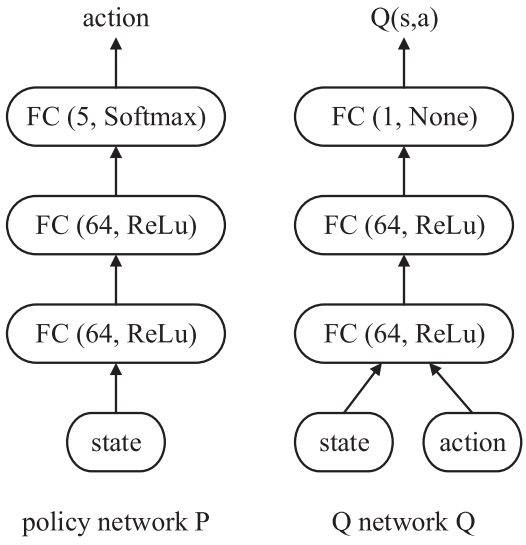

Fig. 2. The architecture of policy network P and Q network Q.

relationship between the evaluation value of current state-action pair and the evaluation value of the next state-action pair. The Q value can be formally defined as follows.

$$Q\left(s,a\right) = \underset{s^{'}\sim P}{E}\left[r\left(s,a\right)+\gamma \underset{a^{'}}{max}Q\left(s^{'},a^{'}\right)\right]. \tag{16}$$

The goal of Q network is to minimize the TD error, the smaller the TD error, the more accurate the evaluation of the estimated state-action pair. Through (16), we can get the TD error, which is

$$L\left(\theta\right) = \underset{(s,a,r,s^{'})}{E}\left[\left(Q(s,a|\theta)-\left(r+Q(s^{'},a^{'}|\theta^{'})\right)\right)^{2}\right]. \tag{17}$$

Therefore, the corresponding parameter θ can be updated through gradient descend, as shown in the following equation.

$$\theta = \theta - \eta\frac{\partial}{\partial\theta}L\left(\theta\right), \tag{18}$$

where η is the learning rate.

On the other hand, the learning for policy network is based on policy gradient, and the goal is to maximize the overall evaluation of actions in strategies generated by the policy network, shown as follows,

$$J\left(\widehat{\theta}\right) = \underset{s\sim D}{E}\left[Q\left(s,P\left(s\middle|\widehat{\theta}\right)\middle|\theta\right)\right]. \tag{19}$$

In DDPG, the policy is deterministic, and the gradient in (19) can be represented as

$$\nabla_{\widehat{\theta}}J\left(\widehat{\theta}\right) = \underset{s\sim D}{E}\left[\nabla_{\theta}P\left(s\middle|\widehat{\theta}\right)\nabla_{a}Q\left(s,a\middle|\theta\right)|_{a=P(s|\widehat{\theta})}\right]. \tag{20}$$

Then, the parameter $\widehat{\theta}$ in policy network can be updated through gradient ascend in (21).

$$\widehat{\theta} = \widehat{\theta}+\eta\frac{\partial}{\partial\widehat{\theta}}J\left(\widehat{\theta}\right). \tag{21}$$

The detailed learning processes can be found in Algorithm 1 in B.

3.2. Multi-agent model and learning processes

Single agent reinforcement learning cannot effectively utilize the global state and action information of other agents, namely the Q network in DDPG can only react according to single agent's state and action, which leads to the inaccurate evaluation given by the Q network. To solve this problem, the MADDPG (Lowe et al., 2017) algorithm had been proposed.

MADDPG is an extension of DDPG in multi-agent environment. The policy networks of MADDPG and DDPG are the same, which consider single agent's state information. The difference between them is that the Q network in MADDPG takes the actions and states of other agents into account, and the difference is only reflected in the training phase. In the testing phase, the execution process of their policy networks are exactly the same since the Q networks are no longer needed. The comparison between MADDPG and DDPG in network structure is shown in Fig. 3.

From Fig. 3, we can see the difference between MADDPG and DDPG. In the following subsections, we will also take the attack agent as an example, and briefly introduce the network structure and learning processe of MADDPG.

3.2.1. The network structure of MADDPG

In MADDPG, there is a policy network and a Q network for attack agent at_i. The definition of policy network is consistent with that of DDPG, as shown in (11). However, the Q network is updated as

$$q_{s_{at,i,t},a_{at,i,t}} = Q_{at,i}\left(s_{at,i,t},a_{at,i,t},ats_{at,i,t},dfs_{at,i,t},ata_{at,i,t},dfa_{at,i,t}\middle|\widehat{\theta}_{at,i}\right). \tag{22}$$

Compared with (12), (22) additionally takes into account the set of states of all other attack agents $ats_{at,i,t}$ and the set of actions $ata_{at,i,t}$ (excluding at_i), as well as the set of states of all defense agents $dfs_{at,i,t}$ and the set of actions $dfa_{at,i,t}$. In particular, $ats_{at,i,t}$ is the connection vector of state feature vectors of all attackers except at_i, and $ata_{at,i,t}$ is the connection vector of 5-dimensional action vectors of all attackers except at_i. $dfs_{at,i,t}$ and $dfa_{at,i,t}$ are defined similarly.

The Q network and policy network in MADDPG also utilize MLP, as shown in Fig. 4.

3.2.2. Learning processes of MADDPG

The learning processes of MADDPG are similar to that of DDPG. For the Q network, the corresponding objective function is

$$L\left(\theta\right) = \underset{\left(s,a,other\_s\_a,r,s^{'}\right)\sim D}{E}\left[\left(Q(s,a,other\_s\_a|\theta)-\left(r+Q(s^{'},a^{'},other\_s^{'}\_a^{'}|\theta^{'})\right)\right)^{2}\right], \tag{23}$$

where $other\_s\_a$ is the states and actions of other attackers and defenders. For the states and actions at the next time instant $other\_s^{'}\_a^{'}$, the states $other\_s^{'}$ are from the replay buffer, and the related actions $other\_a^{'}$ are obtained through the delayed policy network. The update of gradient is shown in (18).

For the policy network, the corresponding objective function is

$$J\left(\widehat{\theta}\right) = \underset{s,other\_s\_a\sim D}{E}\left[Q\left(s,P\left(s\middle|\widehat{\theta}\right),other\_s\_a\middle|\theta\right)\right], \tag{24}$$

and the gradient can be calculated as follows.

$$\nabla_{\widehat{\theta}}J\left(\widehat{\theta}\right) = \underset{s,other\_s\_a\sim D}{E}\left[\nabla_{\theta}P\left(s\middle|\widehat{\theta}\right)\nabla_{a}Q\left(s,a,other\_s\_a\middle|\theta\right)|_{a=P(s|\widehat{\theta})}\right]. \tag{25}$$

Then, the final update of gradient can also be calculated by (21).

The overall processes of multi-agent defense and attack using

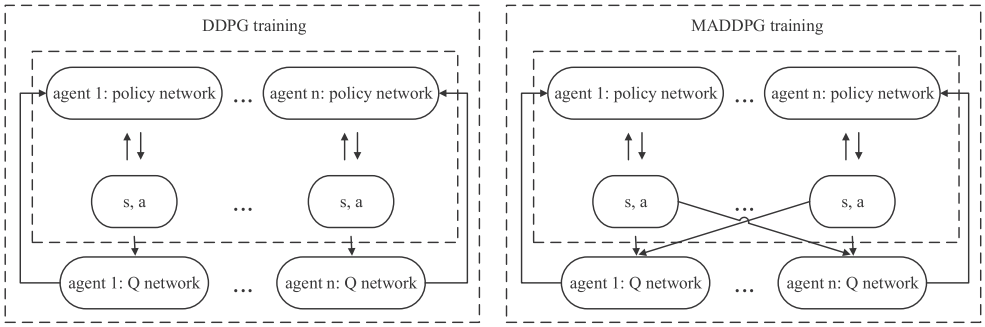

Fig. 3. The difference between MADDPG training and DDPG training.

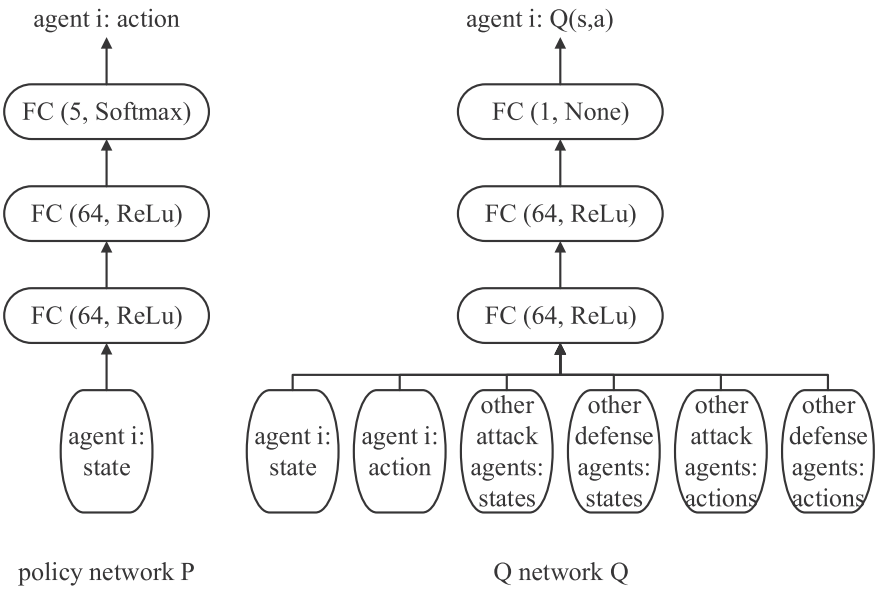

Fig. 4. The architecture of policy network and Q network in MADDPG.

MADDPG are shown in Algorithm 2 in C.

4. Simulation results and analysis

In this section, we conduct a series of simulations to analyze the effects of the applied DRL algorithms in multi-agent defense and attack problem. We also give the environment model settings, evaluation criteria, parameter settings, statistical analysis, and results of the considered issue. The simulations are all carried out in Python 3 under Linux.

4.1. Environment model settings

The simulated environment is a two-dimensional environment with N attackers, M defenders and one defense zone. The environment is constructed in a bounded coordinate system with the upper and lower bounds of X and Y axes being 1 and -1, respectively. The attackers and defenders both have a radius of 0.08, the radius of defense zone is 0.1, and its coordinate is $(0,0)$. In order to reasonably simulate the process of defense and attack, we specifically set that the distances between the initial positions of attackers and the defense zone are more than 0.9, and the distances between the initial positions of defenders and the defense zone are less than 0.4.

The maximum speeds of the attackers and defenders should not exceed 0.4 and 1.0 respectively, and the time unit in environment is 0.1. Note that the defenders' maximum speed is higher than that of attackers, the reason is that there are more attackers than defenders under our situations, and it is almost impossible for the defenders to successively intercept all attackers without the speed advantage. In addition, if an attacker succeeds in attacking the defense zone or is captured by a defender, the attacker will stop and cannot be captured by other defenders again. Finally, the interaction is terminated when there is no more attackers or the maximum number of interactions is reached.

For the defenders, we adopt DRL model to train them, while for the attackers, they can utilize DRL model to learn or just act according to the given rules. For the DRL model controlled agents, their motions are variable, and the accelerations of agents are given by the learning model, but the agents' maximum speed can not be exceeded. For the rule-based attackers, the following simple but effective rules are adopted in this paper: 1) the speed direction of each agent points directly to the defense zone to ensure that it moves along the shortest path towards its goal; 2) the attackers move with a uniform speed, and their speed is the maximum allowed speed.

4.2. Evaluation criteria and parameter settings

In our simulations, three different evaluation criteria are used to demonstrate the performance of the model, which are the average attack hit rate, the average attack success rate, and the average agent curriculum reward, respectively.

The average attack hit rate measures the quality of attacks, and it represents the percentage of N attackers that can hit the defense zone in a confrontation averagely. Obviously, for the attackers, the higher the average hit rate is, the better. The specific calculation of the average attack hit rate is shown as follows.

$$MHR = \frac{1}{|B|}\sum_{i=1}^{|B|}\frac{H_i}{N}. \tag{26}$$

where B represents a set that stores $|B|$ confrontations, H_i represents the number of attackers that successfully hit the defense zone in the i-th confrontation in B.

The average attack success rate also evaluates the quality of attacks, and it represents the number of attackers that can hit the defense zone in a confrontation averagely. For this criterion, as long as the attack hit rate is greater than 0, the attack is considered successful. Similarly, for the attackers, the higher the average attack success rate is, the better. What's more, the average attack success rate is often higher than the average attack hit rate, and we define it as follows,

$$MSR = \frac{1}{|B|}\sum_{i=1}^{|B|} I_i, \tag{27}$$

where I_i is denoted as

$$I_i = \begin{cases} 1, & if \quad H_i > 0, \\ 0, & if \quad H_i = 0. \end{cases} \tag{28}$$

Although the average attack hit rate and the average attack success rate can directly reflect the results of the confrontation, they cannot reflect whether the model is actually optimized. In order to clearly show the optimization process of the model, it is necessary to consider the accumulative reward in a confrontation averagely, since the goal of reinforcement learning is to maximize the accumulative reward. For both the attackers and defenders, the performance of effective training is the

increase in the accumulative reward. The definition of the average accumulative reward is shown as follows.

$$MAR = \frac{1}{|B|}\sum_{i=1}^{|B|}\sum_{j=1}^{E_i}\sum_{g\in TA} r\left(k_g, z_g, j\right), \tag{29}$$

where E_i represents the i-th interaction in B, and the length of the interaction is $|E_i|$. TA represents the set of agents that need to be trained (exclusive of the rule-based agents). k_g stands for the type of agent g, and z_g represents the serial number of the agent under this type.

Note that no additional test sets are needed in this paper, since every confrontation environment in the training phase is randomly generated, and there is almost no repetition. Therefore, there is no repeated training data which will induce the overfitting phenomenon. Thus, the results obtained during training are equivalent to the results obtained on other test sets.

The related parameter settings of the environment, the learning procedure and the neural networks can be found in Table 1.

In Table 1, the maximum interaction length refers to the maximum steps in a confrontation, the total interaction number refers to the confrontation number for training the agents. Our entire experiment was repeated for six times. In each experiment, 200,000 steps of training were performed, and at every 1000 steps, we performed a test. Specifically, the test result is the average value of the results of the previous 1000 steps. We finally report the average results of the six experiments. What's more, we perform the grid search for the hyperparameters of ANN under the scenario, 3 attackers vs. 2 defenders. Specifically, the numbers of layers are $[3, 4, 5]$ and the numbers of cells are $[16, 32, 64]$. But we use the default learning rate (0.01) in Adam optimizer.

4.3. Baseline methods

To further study the effects of different DRL algorithms for solving the multi-agent defense and attack problems, we compare the DDPG-based and the MADDPG-based models with the following baseline algorithms.

Independent Q-learning (IQL) (Tampuu et al., 2017) turns a multi-agent learning problem into a collection of simultaneous single-agent learning problems, and each agent learns an individual action-value function through deep Q-learning independently.

Value Decomposition Networks (VDN) (Sunehag et al., 2018) decomposes a central state-action value function into a sum of individual state-action value functions through value decomposition networks based on the individual observations and actions.

Actor Crtic (AC) (Mnih et al., 2016) combines the strong points of actor-only and critic-only methods. The critic network learns a value function through approximation, and then the value function is used to update the actor network in the direction of performance improvement.

4.4. Multi-agent defense and attack with rule-based attack agents

In the multi-agent defense and attack with rule-based attack agents problem, the defenders make decisions according to the learning model, and the actions of attackers are given through rules.

We consider four different confrontation environments, which are 2 defenders versus 2 attackers, 2 defenders versus 3 attackers, 3 defenders versus 5 attackers, and 4 defenders versus 6 attackers, respectively. In each environment, we compare the performance of defenders under AC, VDN, IQL, DDPG, and MADDPG training models.

In order to more intuitively observe the processes of confrontation, we firstly give a confrontation example of 5 rule-based attackers versus 3 MADDPG-based defenders, as shown in Fig. 5. The confrontation processes are split into 8 frames. Frame 1 gives the initial positions of both attackers and defenders (randomly generated), and the attackers go straight to the defense zone while the defenders take actions according to the MADDPG learning model. In frame 3, attacker 4 is defended by defender 2, then it stops (becomes hollow circle afterwards) and the defender continues to search for other attackers. From frame 4 to frame 7, attackers $2, 3, 5$ and 1 are successfully defended accordingly. Frame 8 shows the final locations of defenders.

Further, we analyze the three evaluation criteria under different scenarios. The average accumulated reward (MAR) of different training models is shown in Fig. 6.

From Fig. 6 we can see that through training, the average accumulative rewards gradually increase for all the DRL models except AC. The increase in rewards implies the effective training of defense agents under different scenarios with rule-based attack agents. While for AC, since it is sensitive to the environments, and there are almost no identical environments in the training phase, thus AC cannot learn effectively. What's more, by comparing the four sub-figures above, we can see that MADDPG achieves the best learning results among all methods. With the increasing number of agents in the scenarios, the gaps between MADDPG and other methods become more and more obvious, and this also reveals that the MADDPG-based learning model can take advantage of other agents' information to make better decisions.

From Fig. 7 we can see that, with increasing steps, the average hit rate of attack agents descends, which means the defense success rate of defense agents increases. Through learning, the defenders can effectively protect the defense zone from the attack of the rule-based attackers. Besides, the performance of MADDPG is better than that of other DRL models, and this is especially obvious under the complex scenario. In Fig. 7a and b, the performance lines of DDPG and MADDPG are very close, however, we can see significant difference in the statistical data in Table 2.

Seen from Fig. 8, the average attack success rate decreases with increasing training steps. The MSR under MADDPG trained defenders is lower than that under other DRL models trained defenders. This also implies that the MADDPG trained defenders performs better under all scenarios, and the advantage becomes more and more obvious as the environment becomes more complex. The detailed difference can be seen in Table 2.

To clearly see the performance of different learning algorithms under different scenarios, Table 2 gives the final results of the criteria under four test scenarios.

From Table 2, we can obtain the following conclusions. (1) MADDPG can consistently outperform other methods under different scenarios, which verifies the superiority of MADDPG. Additionally, the results of t-test indicate that the improvement obtained by MADDPG is significant. (2) AC is more sensitive than other methods, since its action selection process is stochastic in contrast with deterministic action selection in DDPG or MADDPG. As a consequence, the final results of AC are very poor. (3) VDN can achieve promising result in the early stage of learning. However, MADDPG and DDPG outperform VDN as the training steps increase. (4) MADDPG is generally better than DDPG, which reveals that explicitly considering other agents' information in the neural network architecture or in the learning algorithm can further improve the performance. (5) In our environment, a scenario with more attackers is more complicated than a scenario with fewer attackers. As the

Table 1
Parameter settings of the model.

Parameter type	Parameter name	Parameter value
Environment	Maximum interaction length	40
	Total interaction number	200,000
	Test Frequency	1000
Learning	Learning rate	0.01
	Discount factor	0.95
	Batch size	1024
	Optimizer	Adam
Network	Network layer	5
	Activation function	ReLu
	Number of hidden layer neurons	64

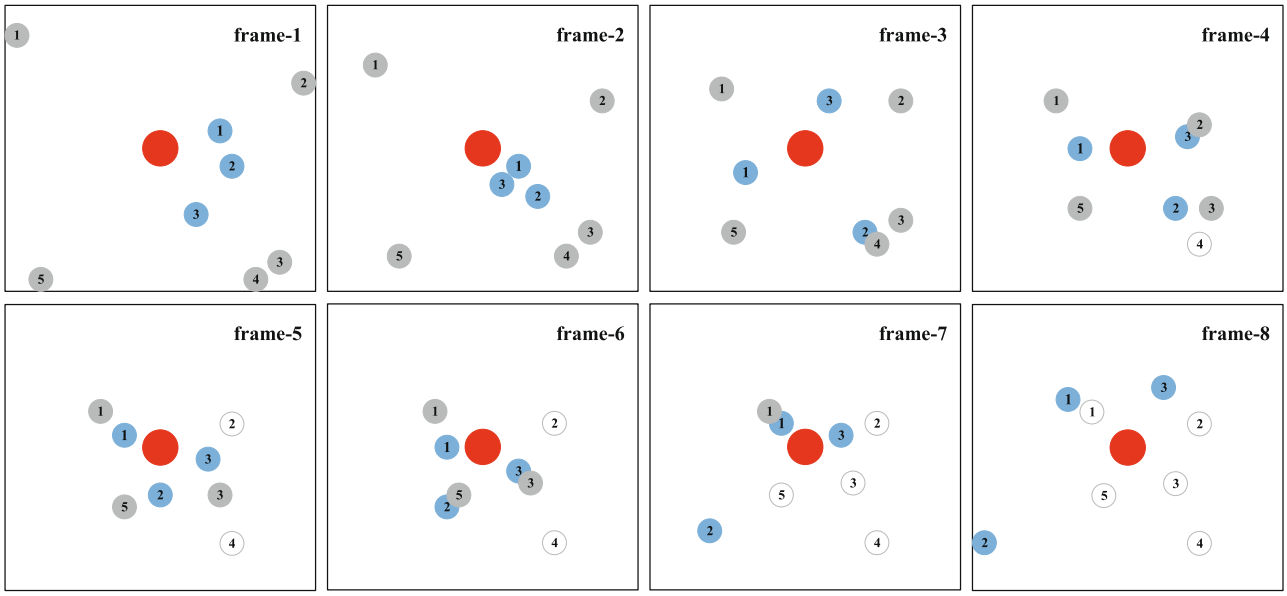

Fig. 5. The processes of confrontation under 5 rule-based attackers and 3 MADDPG-based defenders.

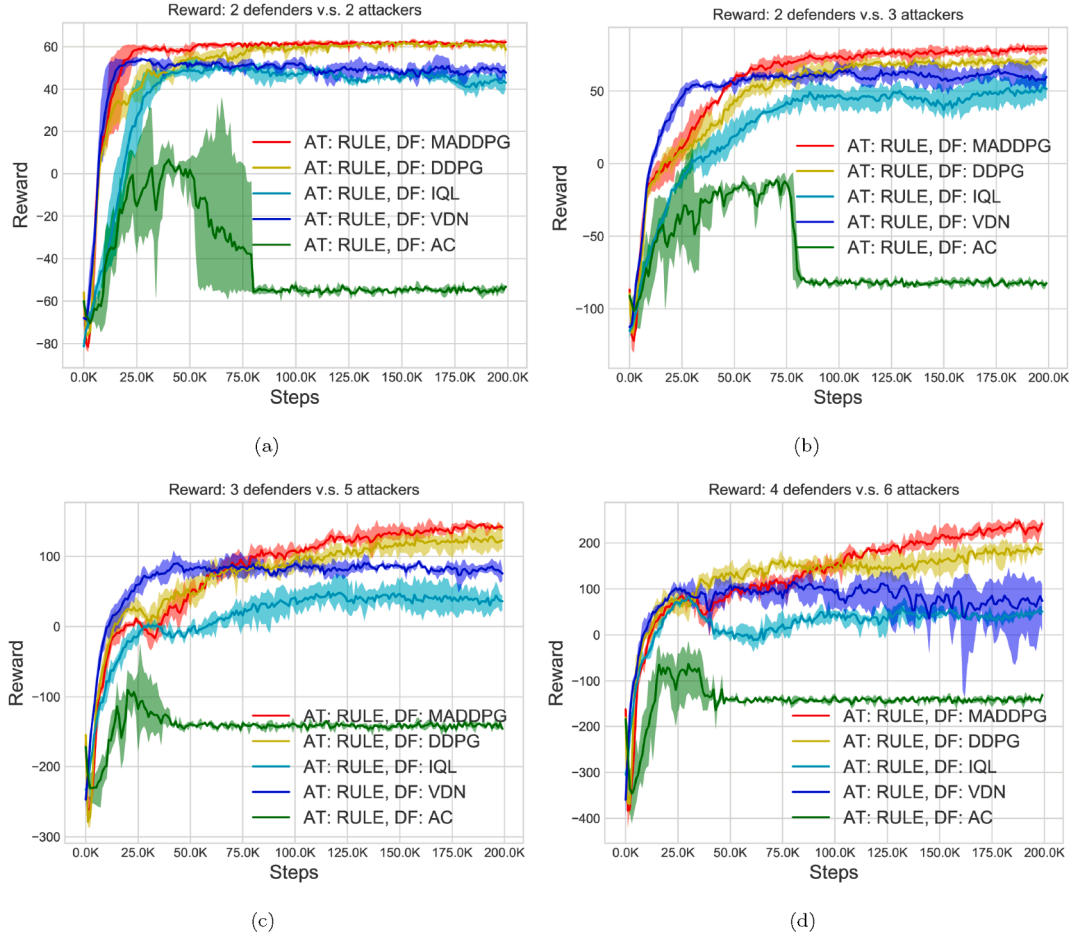

Fig. 6. Comparisons of the average accumulative reward (MAR) under different scenarios. (a) 2 defenders versus 2 attackers; (b) 2 defenders versus 3 attackers; (c) 3 defenders versus 5 attackers; (d) 4 defenders versus 6 attackers.

environment becomes more and more complex, the rule-based attacker's success rate and hit rate continue to increase against our DRL-based defender, and all compared methods suffer from this problem. This drawback is due to two main reasons. (a) The complexity of the environment itself. Apparently, with more attackers, it becomes more difficult for the defenders to successfully defend all of them. However, our learning model-based defenders are still able to constrain the hit rate of a single attacker to a relatively low level. For example, the hit rates of

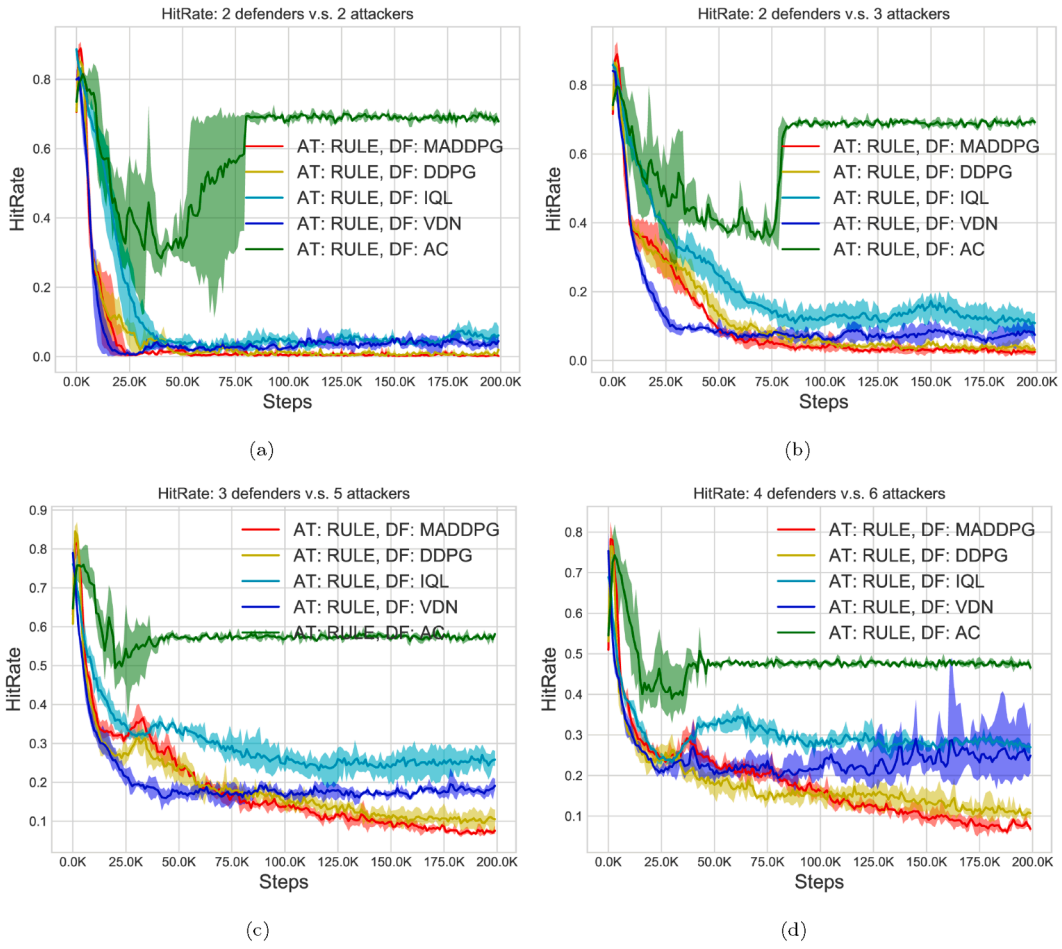

Fig. 7. Comparisons of the average attack hit rate (MHR) under different scenarios. (a) 2 defenders versus 2 attackers; (b) 2 defenders versus 3 attackers; (c) 3 defenders versus 5 attackers; (d) 4 defenders versus 6 attackers.

Table 2
Results of the multi-agent defense and attack under rule-based attack agents. "*" indicates that p-value is less than 0.05 for significance test (two-tailed t-test) over the best baseline.

At Num	Df Num	At Pol	Df Pol	MAR	MHR	MSR
2	2	Rule	AC	-53.23 ± 0.52	$67.86 \pm 0.62\%$	$87.87 \pm 1.18\%$
2	2	Rule	IQL	43.13 ± 2.32	$6.13 \pm 1.81\%$	$10.83 \pm 3.11\%$
2	2	Rule	VDN	47.91 ± 3.18	$4.47 \pm 1.18\%$	$8.43 \pm 3.47\%$
2	2	Rule	DDPG	58.64 ± 3.27	$1.97 \pm 0.21\%$	$3.86 \pm 3.27\%$
2	2	Rule	MADDPG	$62.24 \pm 0.95^{*}$	$0.16 \pm 0.02\%^{*}$	$0.33 \pm 0.40\%^{*}$
3	2	Rule	AC	-82.46 ± 1.27	$69.21 \pm 0.64\%$	$95.66 \pm 0.31\%$
3	2	Rule	IQL	55.82 ± 5.35	$9.15 \pm 1.58\%$	$23.03 \pm 3.82\%$
3	2	Rule	VDN	59.74 ± 4.85	$10.88 \pm 2.36\%$	$26.40 \pm 4.94\%$
3	2	Rule	DDPG	71.22 ± 1.02	$3.57 \pm 0.34\%$	$8.86 \pm 0.85\%$
3	2	Rule	MADDPG	$79.36 \pm 2.74^{*}$	$2.32 \pm 0.31\%^{*}$	$6.16 \pm 0.65\%^{*}$
5	3	Rule	AC	-146.36 ± 2.16	$58.10 \pm 0.03\%$	$97.50 \pm 0.43\%$
5	3	Rule	IQL	36.12 ± 9.37	$25.84 \pm 1.60\%$	$72.37 \pm 1.92\%$
5	3	Rule	VDN	75.63 ± 7.40	$19.09 \pm 1.30\%$	$60.73 \pm 2.24\%$
5	3	Rule	DDPG	122.59 ± 15.64	$10.52 \pm 2.54\%$	$40.09 \pm 8.35\%$
5	3	Rule	MADDPG	$140.96 \pm 3.33^{*}$	$7.54 \pm 0.91\%^{*}$	$30.66 \pm 1.99\%^{*}$
6	4	Rule	AC	-130.77 ± 5.01	$46.61 \pm 0.68\%$	$94.66 \pm 0.73\%$
6	4	Rule	IQL	50.95 ± 7.76	$26.97 \pm 1.08\%$	$79.06 \pm 0.73\%$
6	4	Rule	VDN	74.10 ± 47.01	$24.92 \pm 5.71\%$	$75.20 \pm 7.17\%$
6	4	Rule	DDPG	185.84 ± 10.73	$10.71 \pm 1.13\%$	$45.66 \pm 3.94\%$
6	4	Rule	MADDPG	$242.06 \pm 9.74^{*}$	$6.81 \pm 0.67\%^{*}$	$31.80 \pm 1.52\%^{*}$

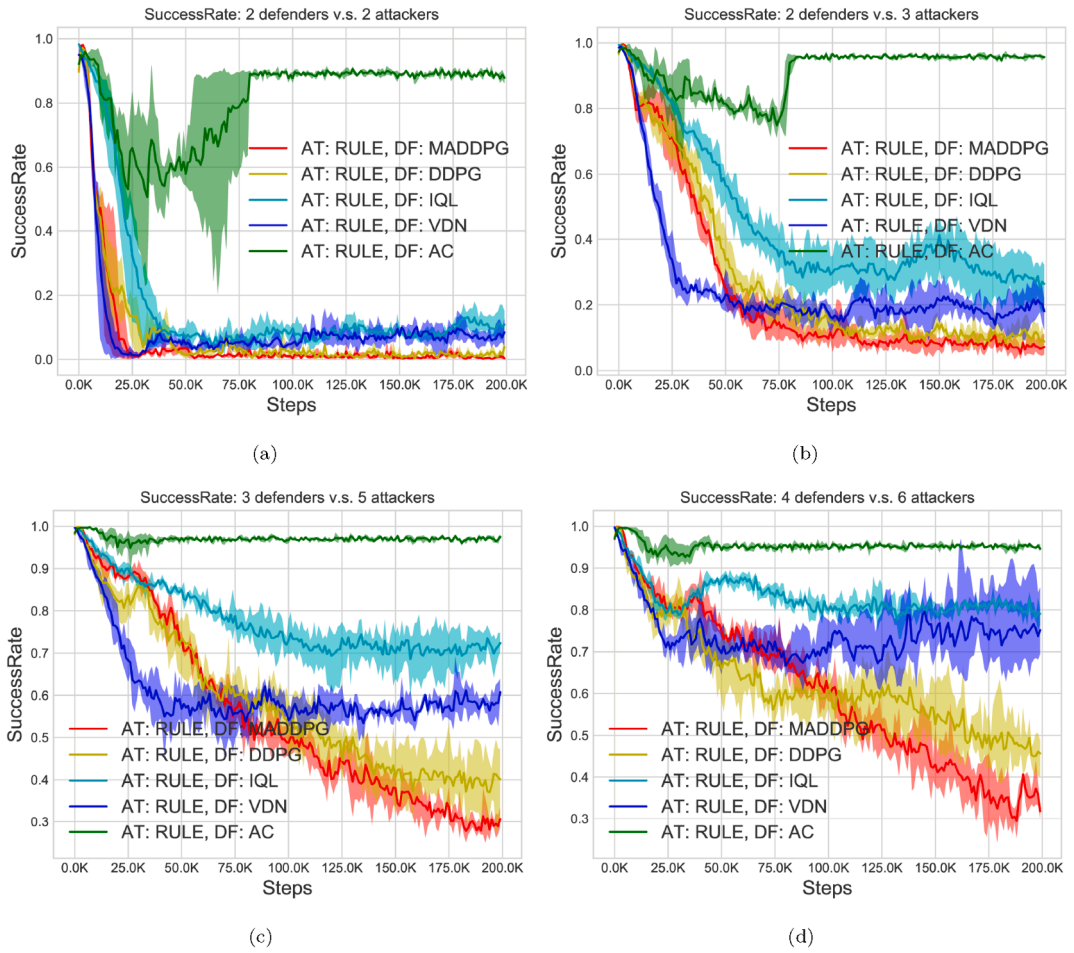

Fig. 8. Comparisons of the average attack success rate (MSR) under different scenarios. (a) 2 defenders versus 2 attackers; (b) 2 defenders versus 3 attackers; (c) 3 defenders versus 5 attackers; (d) 4 defenders versus 6 attackers.

MADDPG for 3 attacks v.s. 2 defenders, 5 attacks v.s. 3 defenders, and 6 attacks v.s. 4 defenders are 2.32%, 7.54%, and 6.81%, while the correlated success rates are 6.16%, 30.66%, and 31.80%. (b) The other reason is the agent scalability problem for the deep reinforcement learning models. As the number of agent increases, the conducted policy should collaborate with more agents, and the state space for the model grows exponentially, which is also a problem faced by most DRL models (Li et al., 2019; Wang et al., 2020; Liu et al., 2020). As a result, it becomes

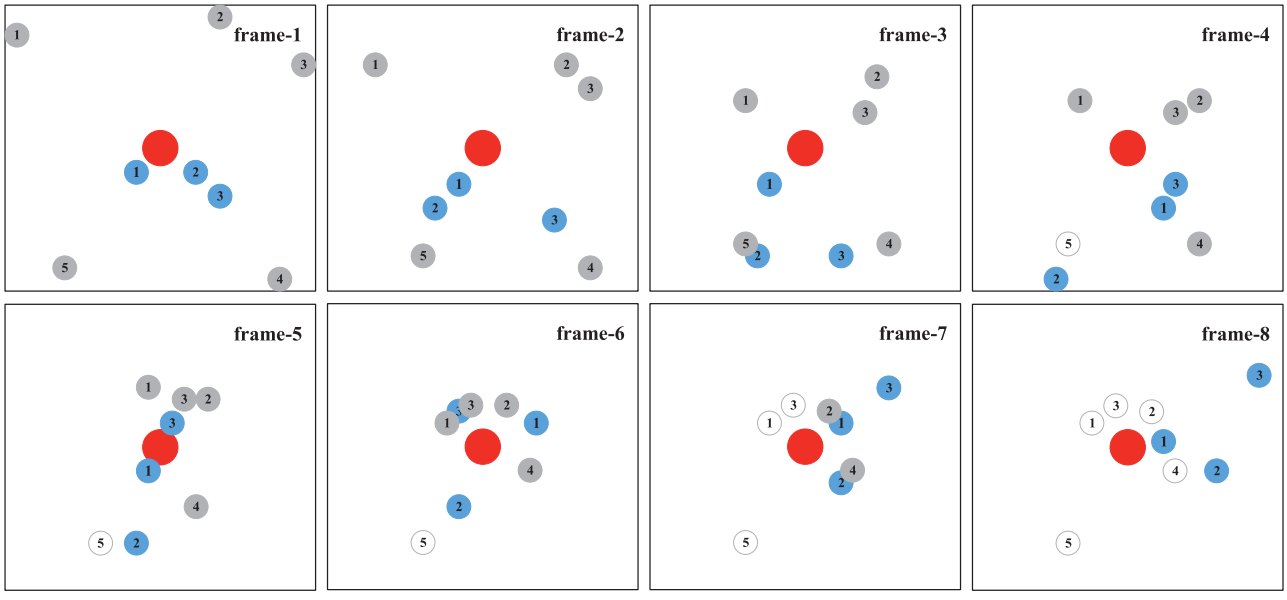

Fig. 9. The processes of confrontation under 5 MADDPG-based attackers and 3 MADDPG-based defenders.

more difficult for the models to learn to acquire promising results.

4.5. Multi-agent defense and attack with DRL-based attack agents

In multi-agent defense and attack with DRL-based attack agents problem, the actions of both attackers and defenders are given through DRL algorithms. In this subsection, we also investigate four scenarios, which are 2 defenders versus 2 attackers, 2 defenders versus 3 attackers, 3 defenders versus 5 attackers, and 4 defenders versus 6 attackers, respectively. In each scenario, we consider both the agents trained with DDPG or MADDPG.

Fig. 9 gives an example of the confrontation processes of 5 attackers versus 3 defenders both trained with MADDPG. In frame 1, the attackers and defenders are randomly generated in the environment, and they both move with variable speeds. In frame 3, attacker 5 is successfully defended by defender 2, and becomes hollow afterwards. The defenders keep moving until defender 3 meets attacker 1 and 3, and stops them simultaneously, as shown in frame 6. Attackers 2 and 4 are respectively defended by defender 1 and 2 in frame 7. Frame 8 shows the final locations of the defend agents.

Further, the results of average accumulated reward (MAR) under different situations and different models are shown in Fig. 10.

To clearly see the performance comparisons between the rule-based attackers and intelligent attackers, the average accumulative reward lines of rule-based attackers are also given in Fig. 10. The increasing MAR explains the learning effects of DDPG and MADDPG algorithms. Compared with the reward curves of rule-based attackers, the MAR of DRL-based attackers shows slight vibrations. Such as in the confrontation of 5 attackers versus 3 defenders, the MADDPG algorithm achieves the best results in the middle of training, but begins to show drastic fluctuations in the following interactions, and even leads to the result that DDPG surpasses MADDPG. This phenomenon is mainly due to the mutual exclusion between the attackers and defenders, and the more the number of agents, the more obvious this kind of mutual exclusion. Moreover, as the number of agents in the environment increases, insufficient training will also result in the fluctuations in the average accumulative reward. Besides, the MAR values under rule-based situations are larger than that under DRL-based situations, which illustrates that under complex environment, the competition tends to be more intense, and the overall MAR values will be smaller.

Then, we compare the MHR and MSR between the rule-based attack agents situations and the DRL-based attack agents situations, as shown in Figs. 11 and 12, respectively.

From the comparison results of MHR and MSR, we can obtain that when confronting the same defenders, the attackers' MHR and MSR after learning through DDPG or MADDPG are higher than those obtained through rule-based attackers. This shows that through learning, the attackers become more competitive, and can better achieve the attack tasks. Meanwhile, the curves of intelligent attackers versus intelligent defenders change a lot, which also illustrate the mutual confrontation and game between them.

In addition, in order to clearly show the final effects of different DRL training models, we give the statistical results under different situations in Table 3. The environment situations are 2 defenders versus 2 attackers, 2 defenders versus 3 attackers, 3 defenders versus 2 attackers, and 4 defenders versus 6 attackers, respectively.

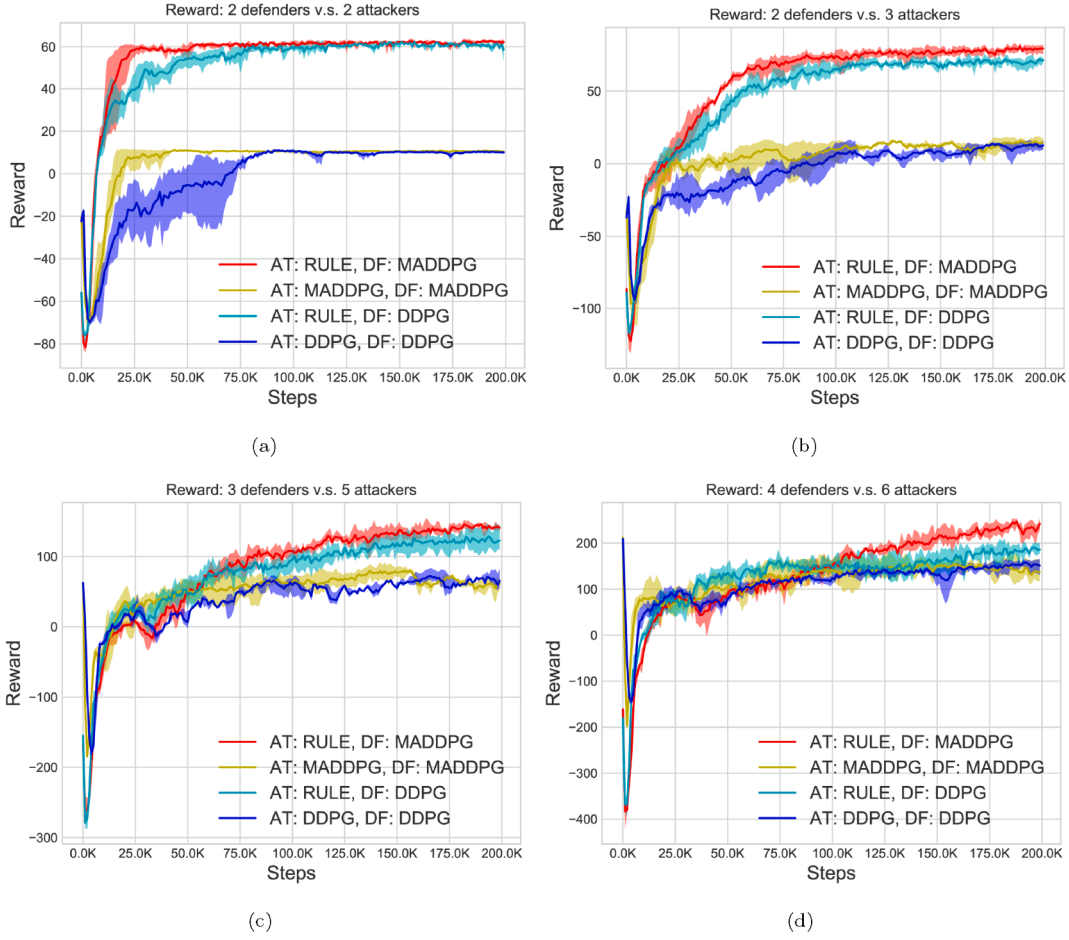

Fig. 10. Comparisons of the average accumulative reward (MAR) under different scenarios and different models. (a) 2 defenders versus 2 attackers; (b) 2 defenders versus 3 attackers; (c) 3 defenders versus 5 attackers; (d) 4 defenders versus 6 attackers.

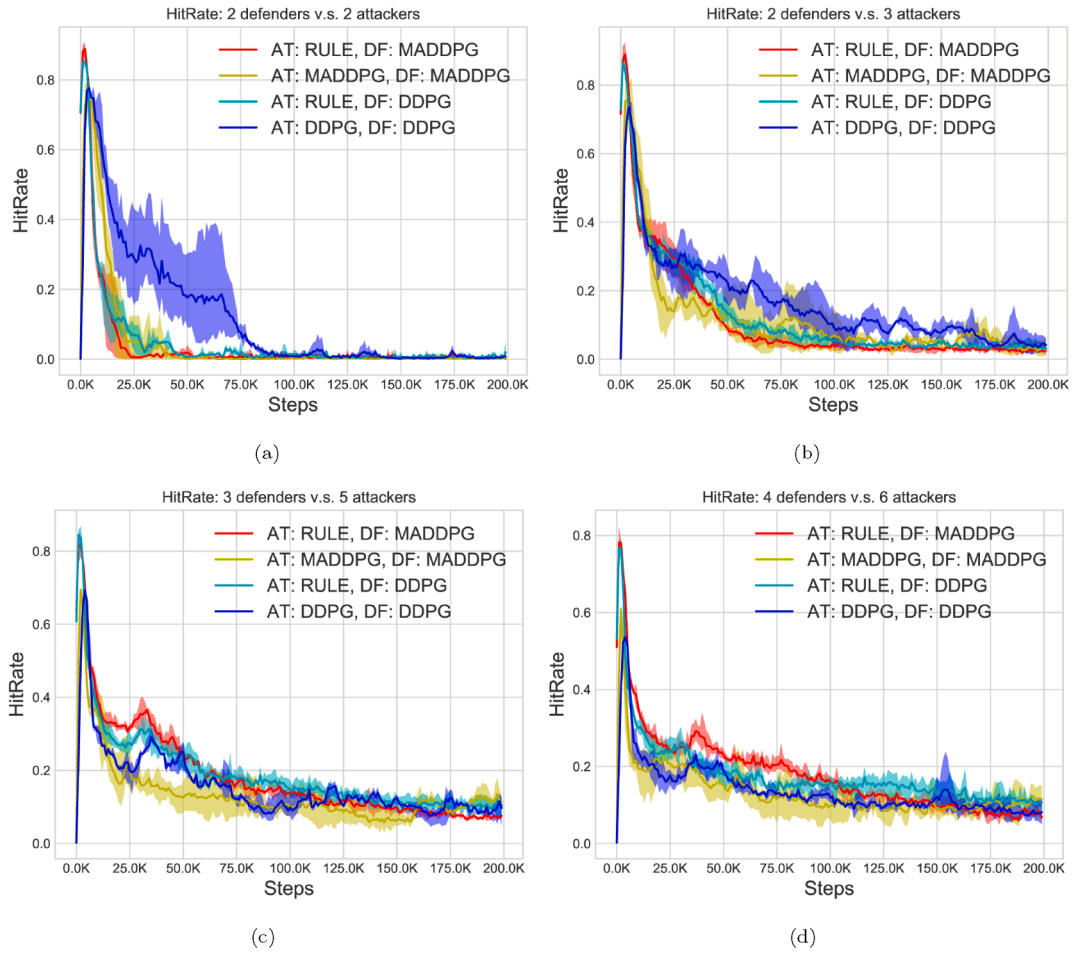

Fig. 11. Comparisons of the average attack hit rate (MHR) under different scenarios and different models. (a) 2 defenders versus 2 attackers; (b) 2 defenders versus 3 attackers; (c) 3 defenders versus 5 attackers; (d) 4 defenders versus 6 attackers.

Seen from Table 3, with the environment becomes complex, the MHR and MSR of intelligent attackers increase gradually, which also reflects the effects of intelligence. What's more, the attack agents with MADDPG training perform better, which displays the advantage of mastering other agents' information in the same team.

Moreover, in Table 3, the t-test is for evaluating the effectiveness of applying MADDPG or DDPG to the attack agents. Different from the experiments in Table 2, here we perform the t-test between the DRL-based attackers and the rule-based attackers. Under the scenarios with DRL-based attackers, both the defenders and attackers are intelligent, the agents of the same type cooperate with each other, and simultaneously they compete with the agents of the opponent type. For example, we evaluate the p-values under "5 rule-based attackers against 3 MADDPG-based defenders" and "5 MADDPG-based attackers against 3 MADDPG-based defenders" situations. Also, we evaluate the p-values for "5 rule-based attackers against 3 DDPG-based defenders" and "5 DDPG-based attackers against 3 DDPG-based defenders" scenarios. We found that the associated application of MADDPG to the attack and defense agents is significant under the scenarios with more agents, e.g., 5 vs. 3 and 6 vs. 4. However, DDPG is not significant for all the considered scenarios. These results also demonstrate that MADDPG is more effective than DDPG in the hybrid cooperation-competition problems.

5. Conclusion and future work

In this paper, we dealt with multi-agent defense and attack problem using deep reinforcement learning algorithms. The defense and attack problem are divided into two categories according to the types of attack agents, namely multi-agent defense and attack with rule-based attackers and multi-agent defense and attack with DRL-based attackers. In both cases, the behavior of defense agents are given through deep reinforcement learning models. We improved the continuous state space, continuous action space and reward function in DDPG and MADDPG algorithms to better solve the considered problem. The network structures and learning methods of DDPG and MADDPG during training are introduced in detail. A series of experiments and comparisons are also conducted to demonstrate the effectiveness of DRL training, and the performance of MADDPG is better than that of other employed DRL models.

However, this paper is mainly aimed at a small number of attack agents and defense agents. Besides, only the attack agents, defense agents and defense zone are considered in the simulated environment. In future work, we will focus on more practical environments containing static and dynamic obstacles, as well as the situations with dynamic changes of agents' numbers (scalability).

CRediT authorship contribution statement

Liwei Huang: Conceptualization, Methodology, Software, Visualization, Writing - original draft, Writing - review & editing. **Mingsheng Fu:** Conceptualization, Funding acquisition, Methodology, Software, Supervision, Writing - review & editing. **Hong Qu:** Conceptualization, Funding acquisition, Resources, Writing - review & editing. **Siying Wang:** Methodology, Visualization, Writing - review & editing. **Shangqian Hu:** Methodology, Visualization, Writing - review & editing.

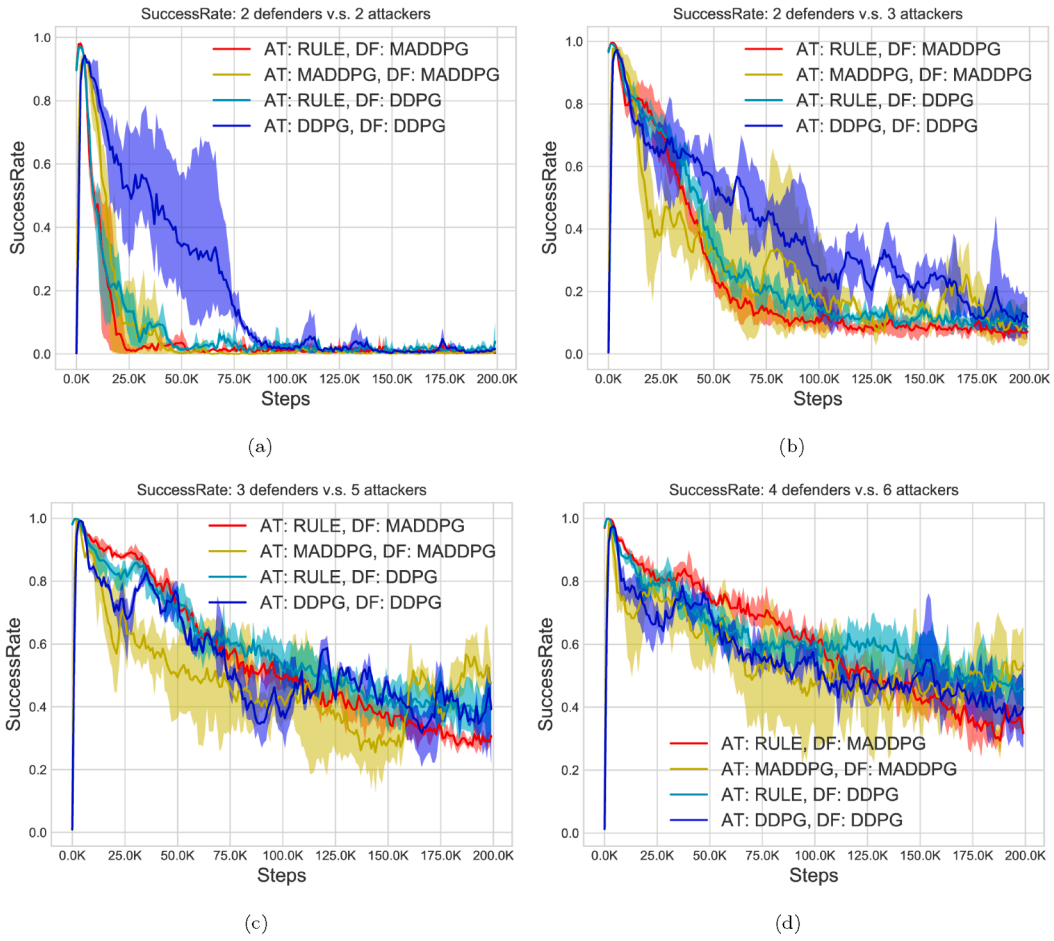

Fig. 12. Comparisons of the average attack success rate (MSR) under different scenarios and different models. (a) 2 defenders versus 2 attackers; (b) 2 defenders versus 3 attackers; (c) 3 defenders versus 5 attackers; (d) 4 defenders versus 6 attackers.

Table 3
Results of the multi-agent defense and attack under DRL-based attack agents. "*" indicates that p-value is less than 0.05 for significance test (two-tailed t-test) over the best baseline.

At Num	Df Num	At Pol	Df Pol	MAR	MHR	MSR
2	2	Rule	DDPG	58.64 ± 3.27	$1.97 \pm 0.21\%$	$3.86 \pm 3.27\%$
2	2	DDPG	DDPG	$10.07 \pm 0.59^{*}$	$0.83 \pm 0.22\%$	$1.63 \pm 0.47\%$
3	2	Rule	DDPG	71.22 ± 1.02	$3.57 \pm 0.34\%$	$8.86 \pm 0.85\%$
3	2	DDPG	DDPG	$12.53 \pm 2.61^{*}$	$4.11 \pm 1.89\%$	$11.83 \pm 5.45\%$
5	3	Rule	DDPG	122.59 ± 15.64	$10.52 \pm 2.54\%$	$40.09 \pm 8.35\%$
5	3	DDPG	DDPG	$65.42 \pm 12.27^{*}$	$9.67 \pm 3.31\%$	$39.26 \pm 12.06\%$
6	4	Rule	DDPG	185.84 ± 10.73	$10.71 \pm 1.13\%$	$45.66 \pm 3.94\%$
6	4	DDPG	DDPG	$150.99 \pm 15.36^{*}$	$8.17 \pm 2.70\%$	$45.67 \pm 3.94\%$
2	2	Rule	MADDPG	62.24 ± 0.95	$0.16 \pm 0.02\%$	$0.33 \pm 0.40\%$
2	2	MADDPG	MADDPG	$10.47 \pm 0.62^{*}$	$0.16 \pm 0.20\%$	$0.76 \pm 1.01\%$
3	2	Rule	MADDPG	79.36 ± 2.74	$2.32 \pm 0.31\%$	$6.16 \pm 0.65\%$
3	2	MADDPG	MADDPG	$14.29 \pm 2.56^{*}$	$2.87 \pm 1.92\%$	$8.40 \pm 5.71\%$
5	3	Rule	MADDPG	140.96 ± 3.33	$7.54 \pm 0.91\%$	$30.66 \pm 1.99\%$
5	3	MADDPG	MADDPG	$61.47 \pm 6.39^{*}$	$11.43 \pm 1.20\%^{*}$	$47.60 \pm 4.38\%^{*}$
6	4	Rule	MADDPG	242.06 ± 9.74	$6.81 \pm 0.67\%$	$31.80 \pm 1.52\%$
6	4	MADDPG	MADDPG	$136.85 \pm 18.84^{*}$	$11.51 \pm 2.78\%^{*}$	$53.20 \pm 13.05\%^{*}$

Declaration of Competing Interest

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

Acknowledgements

This work was supported by the China Postdoctoral Science Foundation under Grant 2020M673182, and National Science Foundation of China under Grant 61976043.

Appendix A. Detailed state features of attack and defense agents

In our environment, there is one defense zone dz, and we always let it be the center of the environment at each time instant, and thus the coordinate of defense zone P_{dz} is $(0,0)$. As a consequence, we only need to know the approximate distance between the agent and the defense zone dz, and the distance between the agent and the other agents, as well as the relative directions.

A.1. State features of attack agent

Let the i-th attack agent in AT be $at_i \in AT$, we can get the relative coordinate between at_i and dz, as well as the attacker's velocity, which are $(x_{at,i}, y_{at,i})$ and $(vx_{at,i}, vy_{at,i})$ respectively.

More specifically, self feature vector $self_{at,i,t}$ is represented as

$$self_{at,i,t} = \left[x_{at,i}, y_{at,i}, vx_{at,i}, vy_{at,i}\right]. \tag{30}$$

The feature vector of other attackers $ats_{at,i,t}$ includes $N-1$ sub-vectors, which are displayed as follows,

$$ats_{at,i,t} = \left[\widehat{s}_{at,i,0,t}, \ldots, \widehat{s}_{at,i,N-1,t}\right], \tag{31}$$

where $\widehat{s}_{at,i,l,t}, l \neq i$ represents the relative states between attacker at_i and other attacker at_l, which is denoted as

$$\widehat{s}_{at,i,l,t} = \left[x_{at,i} - x_{at,l}, y_{at,i} - y_{at,l}, vx_{at,l}, vy_{at,l}\right]. \tag{32}$$

The feature vector of defenders $dfs_{at,i,t}$ includes M sub-vectors, which are displayed as

$$dfs_{at,i,t} = \left[\overline{s}_{at,i,0,t}, \ldots, \overline{s}_{at,i,M,t}\right], \tag{33}$$

where $\overline{s}_{at,i,l,t}$ represents the relative states between attacker at_i and defender df_l, which is expressed as follows,

$$\overline{s}_{at,i,0,t} = \left[x_{at,i} - x_{df,l}, y_{at,i} - y_{df,l}, vx_{df,l}, vy_{df,l}\right]. \tag{34}$$

A.2. State features of defense agent

Similarly, for the j-th defense agent $df_j \in DF$, we can get the relative coordinate between df_j and dz, as well as the defender's velocity in last time instant, which are $(x_{df,j}, y_{df,j})$ and $(vx_{df,j}, vy_{df,j})$, respectively.

More specifically, the feature vector $self_{df,j,t}$ is represented as

$$self_{df,j,t} = \left[x_{df,j}, y_{df,j}, vx_{df,j}, vy_{df,j}\right]. \tag{35}$$

In feature vector $ats_{df,j,t}$, there are N sub-vectors, which are represented as

$$ats_{df,j,t} = \left[\widehat{s}_{df,j,0,t}, \ldots, \widehat{s}_{df,j,N,t}\right], \tag{36}$$

where $\widehat{s}_{df,j,l,t}$ stands for the relative states between defender df_j and attacker at_l, which is

$$\widehat{s}_{df,j,l,t} = \left[x_{df,j} - x_{at,l}, y_{df,j} - y_{at,l}, vx_{at,l}, vy_{at,l}\right]. \tag{37}$$

In feature vector $dfs_{df,j,t}, l \neq j$, there are $M-1$ sub-vectors, which are displayed as

$$dfs_{df,j,t} = \left[\overline{s}_{df,j,0,t}, \ldots, \overline{s}_{df,j,M-1,t}\right], \tag{38}$$

where $\overline{s}_{df,j,0,t}$ stands for the relative states between defender df_j and other defender df_l, which is

$$\overline{s}_{df,j,0,t} = \left[x_{df,j} - x_{df,l}, y_{df,j} - y_{df,l}, vx_{df,l}, vy_{df,l}\right]. \tag{39}$$

Algorithm 1: Multi-agent Defense and Attack Using DDPG

Require: Initial policy network parameter $\widehat{\theta}$, Q network parameter θ, and replay buffer D.

1: Assign target network parameters $\widehat{\theta}' \leftarrow \widehat{\theta}$, $\theta' \leftarrow \theta$.
2: **while** not stop **do**

(*continued on next page*)

(*continued*)

3: **for** $t = 1 \sim max\_episode\_length$ **do**
4: Obtain current environment state s, and obtain action a according to policy network P.
5: $a = P(s|\widehat{\theta}) + \varepsilon$, $\varepsilon \sim N$ (N is noise).
6: Perform action a, obtain new state $s^{'}$, reward r, and add $(s, a, r, s^{'})$ into replay buffer D.
7: **if** $s^{'}$ is the terminate state **then**
8: Reset environment.
9: break
10: **end if**
11: **end for**
12: Randomly select b samples from D, and add them into B.
13: Calculate Q value $y(r, s^{'}) = r + \gamma Q(s^{'}, P(s^{'}|\widehat{\theta}^{'})|\theta^{'})$.
14: Update Q network parameter.
15: $\theta \leftarrow \theta - \nabla_{\theta} \frac{1}{|B|} \sum_{(s,a,r,s^{'}) \in B} (Q(s, a|\theta) - y(r, s^{'}))^2$.
16: Update policy network parameter.
17: $\widehat{\theta} \leftarrow \widehat{\theta} + \nabla_{\widehat{\theta}} \frac{1}{|B|} \sum_{s \in B} Q(s, P(s|\widehat{\theta})|\theta)$.
18: Update target network parameters.
19: $\widehat{\theta}^{'} \leftarrow \rho\widehat{\theta}^{'} + (1-\rho)\widehat{\theta}$.
20: $\theta^{'} \leftarrow \rho\theta^{'} + (1-\rho)\theta$.
21: **end while**

Appendix B. DDPG for multi-agent defense and attack

The details of using DDPG in this paper is given as follows.

In (17), we adopt different parameters θ and $\theta^{'}$ for different states. In the training process, we don't consider $\theta^{'}$, and only update θ. But at regular intervals, we will use current θ to update the value of $\theta^{'}$. By doing so, $r + \gamma Q(s^{'}, a^{'}|\theta^{'})$ can be treated as a constant label, and this is convenient for training. What's more, since DDPG is a reinforcement learning method with deterministic policy, $a^{'}$ is unique, and $a^{'} = P(s|\widehat{\theta}^{'})$ instead of choosing an action with maximum Q value according to the Q network. Similarly, the policy network also has delayed network parameters $\widehat{\theta}^{'}$, and a replay buffer D which stores $(s, a, r, s^{'})$ obtained by the interactions of agents. During training, we randomly take K sets from D each time.

The overall processes of multi-agent defense and attack using DDPG are shown in Algorithm 1.

Appendix C. MADDPG for multi-agent defense and attack

The learning algorithm of MADDPG is similar to that of DDPG. For the Q network in MADDPG, the input also takes other agents' information into consideration for the global critic, but the parameter update formula stays the same, also as shown in (18). Specifically, for the policy gradient in (25), the actions of current agent need to be calculated in real time through the policy network, the rest of actions are obtained through the repay buffer, and the final update of gradient is shown in (21).

The overall processes of multi-agent defense and attack using MADDPG are shown in Algorithm 2.

Algorithm 2: Multi-agent defense and attack using MADDPG

Require: Number of agents M, policy network $\widehat{\theta}_i$, Q network parameter θ_i, $i = 1$ *to* M.
1: Assign target network parameters
2: **for** $i = 1$ *to* M **do**
3: $\widehat{\theta}_i^{'} \leftarrow \widehat{\theta}_i$, $\theta_i^{'} \leftarrow \theta_i$.
4: **end for**
5: **while** not stop **do**
6: **for** $t = 1 \sim max\_episode\_length$ **do**
7: **for** $i = 1$ *to* M **do**
8: Obtain current environment state s_i, and obtain action a_i according to policy network P.
9: $a_i = P(s_i|\widehat{\theta}_i) + \varepsilon$, $\varepsilon \sim N$.
10: **end for**
11: Perform actions $(a_1, a_2, \ldots, a_M)$, obtain new states $(s_1^{'}, s_2^{'}, \ldots, s_M^{'})$, rewards $(r_1, r_2, \ldots, r_M)$, and add $(s_1, a_1, r_1, s_1^{'}, \ldots, s_M, a_M, r_M, s_M^{'})$ into replay buffer D.
12: **if** there exists terminate state in $(s_1^{'}, s_2^{'}, \ldots, s_M^{'})$ **then**
13: Reset environment.
14: break
15: **end if**
16: **end for**
17: Randomly select b samples from D, and add them into B.
18: **for** $i = 1$ *to* M **do**
19: Calculate Q value $y_i = r_i + \gamma Q(s_i^{'}, a_i^{'}, other\_s_i^{'} \_ a_i^{'}|\theta_i^{'})$.
20: Update Q network parameter.

(*continued on next page*)

(*continued*)

21: $\theta \leftarrow \theta - \nabla_\theta \frac{1}{|B|} \sum_{(s_i, a_i, other\_s_i\_a_i) \in B} (Q(s_i, a_i, other\_s_i\_a_i|\theta_i) - y_i)^2$.

22: Update policy network parameter.

23: $\widehat{\theta} \leftarrow \widehat{\theta} + \nabla_{\widehat{\theta}} \frac{1}{|B|} \sum_{(s_i, other\_s_i\_a_i) \in B} Q(s_i, P(s_i|\widehat{\theta}_i), other\_s_i\_a_i|\theta_i)$.

24: **end for**

25: **for** $i = 1$ *to* M **do**

26: Update target network parameters.

27: $\widehat{\theta}_i^{'} \leftarrow \rho \widehat{\theta}_i^{'} + (1-\rho)\widehat{\theta}_i$.

28: $\theta_i^{'} \leftarrow \rho \theta_i^{'} + (1-\rho)\theta_i$.

29: **end for**

30: **end while**

References

Abdoos, M., Mozayani, N., & Bazzan, A. L. (2011). Traffic light control in non-stationary environments based on multi agent q-learning. In *2011 14th international IEEE conference on intelligent transportation systems (ITSC)* (pp. 1580–1585). https://doi.org/10.1109/ITSC.2011.6083114

Bu, L., Babu, R., De Schutter, B., et al. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics Part C (Applications and Reviews), 38*, 156–172. https://doi.org/10.1109/TSMCC.2007.913919

Chen, Y. F., Liu, M., Everett, M., & How, J. P. (2017). Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)* (pp. 285–292). https://doi.org/10.1109/ICRA.2017.7989037

D'Andrea, R. & Murray, R. M. (2003). The roboflag competition. In Proceedings of the 2003 American control conference, 2003 (Vol. 1, pp. 650–655). DOI: 10.1109/ACC.2003.1239093.

Das, P. K., Behera, H. S., Das, S., Tripathy, H. K., Panigrahi, B. K., & Pradhan, S. (2016). A hybrid improved pso-dv algorithm for multi-robot path planning in a clutter environment. *Neurocomputing, 207*, 735–753. https://doi.org/10.1016/j.neucom.2016.05.057

Earl, M. G. & D'Andrea, R. (2002a). Modeling and control of a multi-agent system using mixed integer linear programming. In Proceedings of the 41st IEEE conference on decision and control, 2002. (Vol. 1, pp. 107–111). DOI: 10.1109/CDC.2002.1184476.

Earl, M. G. & D'Andrea, R. (2002b). A study in cooperative control: The roboflag drill. In Proceedings of the 2002 American control conference (IEEE Cat. No. CH37301) (Vol. 3, pp. 1811–1812). DOI: 10.1109/ACC.2002.1023829.

Earl, M. G., & D'Andrea, R. (2007). A decomposition approach to multi-vehicle cooperative control. *Robotics and Autonomous Systems, 55*, 276–291. https://doi.org/10.1016/j.robot.2006.11.002

Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N., & Whiteson, S. (2018). Counterfactual multi-agent policy gradients. *Thirty-second AAAI conference on artificial.* intelligence.

Galstyan, A., Czajkowski, K., & Lerman, K. (2005). Resource allocation in the grid with learning agents. *Journal of Grid Computing, 3*, 91–100. https://doi.org/10.1007/s10723-005-9003-7

Goldman, C. V., & Zilberstein, S. (2004). Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research, 22*, 143–174.

Huang, L., Qu, H., Ji, P., Liu, X., & Fan, Z. (2016). A novel coordinated path planning method using k-degree smoothing for multi-uavs. *Applied Soft Computing, 48*, 182–192. https://doi.org/10.1016/j.asoc.2016.06.046

Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaneda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., et al. (2019). Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science, 364*, 859–865. https://doi.org/10.1126/science.aau6249

Kantaros, Y., & Zavlanos, M. M. (2016). Global planning for multi-robot communication networks in complex environments. *IEEE Transactions on Robotics, 32*, 1045–1061. https://doi.org/10.1109/TRO.2016.2593045

Kim, G., & Chung, W. (2006). Tripodal schematic control architecture for integration of multi-functional indoor service robots. *IEEE Transactions on Industrial Electronics, 53*, 1723–1736. https://doi.org/10.1109/TIE.2006.881956

Li, S., Wu, Y., Cui, X., Dong, H., Fang, F. & Russell, S. (2019). Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In Proceedings of the AAAI conference on artificial intelligence (Vol. 33, pp. 4213–4220).

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *Computer Science, 8*, A187. https://doi.org/10.1016/S1098-3015(10)67722-4

Littman, M. L. (2015). Reinforcement learning improves behaviour from evaluative feedback. *Nature, 521*, 445. https://doi.org/10.1038/nature14540

Liu, Y., Wang, W., Hu, Y., Hao, J., Chen, X. & Gao, Y. (2020). Multi-agent game abstraction via graph attention neural network. In AAAI (pp. 7211–7218).

Lowe, R., Wu, Y., 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).

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

Park, K.-H., Kim, Y.-J., & Kim, J.-H. (2001). Modular q-learning based multi-agent cooperation for robot soccer. *Robotics and Autonomous Systems, 35*, 109–122. https://doi.org/10.1016/S0921-8890(01)00114-2

Pendharkar, P. C., & Cusatis, P. (2018). Trading financial indices with reinforcement learning agents. *Expert Systems with Applications, 103*, 1–13. https://doi.org/10.1016/j.eswa.2018.02.032

Qu, H., Xing, K., & Alexander, T. (2013). An improved genetic algorithm with co-evolutionary strategy for global path planning of multiple mobile robots. *Neurocomputing, 120*, 509–517. https://doi.org/10.1016/j.neucom.2013.04.020

Robinson, D. R., Mar, R. T., Estabridis, K., & Hewer, G. (2018). An efficient algorithm for optimal trajectory generation for heterogeneous multi-agent systems in non-convex environments. *IEEE Robotics and Automation Letters, 3*, 1215–1222. https://doi.org/10.1109/LRA.2018.2794582

Silva, M. A. L., de Souza, S. R., Souza, M. J. F., & Bazzan, A. L. C. (2019). A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems. *Expert Systems with Applications, 131*, 148–171. https://doi.org/10.1016/j.eswa.2019.04.056

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V. F., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K. & et al. (2018). Value-decomposition networks for cooperative multi-agent learning based on team reward. In AAMAS (pp. 2085–2087).

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press.

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., & Vicente, R. (2017). Multiagent cooperation and competition with deep reinforcement learning. *PloS One, 12*, Article e0172395.

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

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature, 575*, 350–354. https://doi.org/10.1038/s41586-019-1724-z

Wang, W., Yang, T., Liu, Y., Hao, J., Hao, X., Hu, Y., Chen, Y., Fan, C. & Gao, Y. (2020). From few to more: Large-scale dynamic multiagent curriculum learning. In AAAI (pp. 7293–7300).

Yi, X., Zhu, A., Yang, S. X., & Luo, C. (2017). A bio-inspired approach to task assignment of swarm robots in 3-d dynamic environments. *IEEE Transactions on Cybernetics, 47*, 974–983. https://doi.org/10.1109/TCYB.2016.2535153

Yu, C., Zhang, M., Ren, F., & Tan, G. (2015). Multiagent learning of coordination in loosely coupled multiagent systems. *IEEE Transactions on Cybernetics, 45*, 2853–2867. https://doi.org/10.1109/TCYB.2014.2387277