Learning Multi-Agent Cooperation via Considering Actions of Teammates

Shanqi Liu, Weiwei Liu, Wenzhou Chen, Guanzhong Tian, *Member, IEEE*, Jun Chen, Yao Tong, Junjie Cao, and Yong Liu

***Abstract*— Recently value-based centralized training with decentralized execution (CTDE) multi-agent reinforcement learning (MARL) methods have achieved excellent performance in cooperative tasks. However, the most representative method among these methods, Q-network MIXing (QMIX), restricts the joint action Q values to be a monotonic mixing of each agent's utilities. Furthermore, current methods cannot generalize to unseen environments or different agent configurations, which is known as ad hoc team play situation. In this work, we propose a novel Q values decomposition that considers both the return of an agent acting on its own and cooperating with other observable agents to address the nonmonotonic problem. Based on the decomposition, we propose a greedy action searching method that can improve exploration and is not affected by changes in observable agents or changes in the order of agents' actions. In this way, our method can adapt to ad hoc team play situation. Furthermore, we utilize an auxiliary loss related to environmental cognition consistency and a modified prioritized experience replay (PER) buffer to assist training. Our extensive experimental results show that our method achieves significant performance improvements in both challenging monotonic and nonmonotonic domains, and can handle the ad hoc team play situation perfectly.**

***Index Terms*— Ad hoc team play, cooperative game, multi-agent reinforcement learning (MARL), nonmonotonic, value decomposition.**

Nomenclature

(o_i, a_i)	Observation and action of agent i.
$(o_j^{\text{joint}}, u_j)$	Local joint observation and joint action of all agents in team j.
N, M_j, G	Number of total agents, agents in team j, and total teams.
$r_{\text{tot}}(s, u)$	Total reward from the environment under state s and joint action u.
$r_a(o_i, a_i)$	Reward that is only related to each agent itself.
$r_c(o_j^{\text{joint}}, u_j)$	Reward related to cooperative tasks of each team.
$Q_{\text{tot}}(s, u)$	Agent i's expected future return of $r_{\text{tot}}(s, u)$.
$Q_a^i(o_i, a_i)$	Agent i's expected future return of $r_a(o_i, a_i)$.
$Q_{ctot}^j(o_j^{\text{joint}}, u_j)$	Team j's expected future return of $r_c(o_j^{\text{joint}}, u_j)$.
$Q_c^i(o_j^{\text{joint}}, u_j)$	Agent i's (in team j) expected future return of $r_c(o_j^{\text{joint}}, u_j)$.
a_k^*	Cooperative action that agent i believes agent k in the team will do.
(a_i^*, a_i')	Optimal action and suboptimal action of agent i.
$\hat{Q}_c^i((o_i, a_i), a_k^*)$	Agent i's decomposition of $Q_c^i(o_j^{\text{joint}}, u_j^*)$.

Manuscript received 21 January 2022; revised 30 October 2022 and 2 February 2023; accepted 23 March 2023. Date of publication 18 April 2023; date of current version 6 August 2024. This work was supported in part by the Autonomous Intelligent Unmanned Systems through NSFC under Grant 62088101 and in part by the National Natural Science Foundation of China under Grant U21A20484. *(Corresponding authors: Junjie Cao; Yong Liu.)*

Shanqi Liu, Weiwei Liu, Jun Chen, Junjie Cao, and Yong Liu are with the State Key Laboratory of Industrial Control Technology, Institute of Cyber-Systems and Control, Zhejiang University, Hangzhou, Zhejiang 310027, China (e-mail: shanqiliu@zju.edu.cn; 11932061@zju.edu.cn; junc@zju.edu.cn; cjunjie@zju.edu.cn; yongliu@iipc.zju.edu.cn).

Wenzhou Chen is with the College of Computer Science, Hangzhou Dianzi University, Hangzhou 310005, China (e-mail: wenzhouchen@zju.edu.cn).

Guanzhong Tian is with the Ningbo Innovation Center, Zhejiang University, Hangzhou, Zhejiang 310027, China (e-mail: gztian@zju.edu.cn).

Yao Tong is with the Northwest Institute of Mechanical and Electrical Engineering and the Northwest Electromechanical Engineering Research Institute, Xianyang 712000, China (e-mail: yaotong1991@gmail.com).

Digital Object Identifier 10.1109/TNNLS.2023.3262921

I. Introduction

Many critical tasks like autonomous driving [1], multi-agent cooperative tracking [2], manipulation control [3], and swarms [4] involve multiple agents acting in the same environment. Recently, cooperative multi-agent reinforcement learning (MARL) has been primarily used to learn good behaviors in such tasks from agents' experiences. The most popular methods in MARL are centralized training with decentralized execution (CTDE) methods [5], [6], which can deal with practical communication constraints and an exponentially growing large joint action space. In CTDE methods, the state-of-the-art methods are a class of methods that learn factored value functions, including value decomposition network (VDN) [7], QMIX [8], etc. One of the most representative methods among these CTDE methods is QMIX, which represents the optimal joint action value function using a restricted monotonic mixing function of each agent utilities. This restricted function allows for efficient maximization during training and easy decentralization of the learned policy [9]. However, two key challenges still stand between CTDE methods and these real-world applications. First, QMIX cannot always represent the true optimal value function, as the monotonicity constraint restricts QMIX to suboptimal value approximations in the nonmonotonic environments. This means it cannot represent the value functions of an agent's optimal action depending on other agents' actions. Second, current methods usually fail when environments have different team sizes and configurations at test time, which is known as ad hoc team play situation [10]. The reason is that agents tend to learn a fixed policy in current methods. Instead, in the ad hoc team play MARL setting, agents must assess and adapt to others' capabilities to behave optimally.

In this work, we address these two challenges by proposing a novel MARL method to cooperative tasks. The insight behind our method is that humans can interpret the actions of others and act in a way that is informative when their actions are being observed by others, which is referred to as the theory of mind (ToM) [11]. In the setting of QMIX class methods, the main reason for monotonicity constraint value decomposition is that each agent acts independently instead of cooperating with others at each time step. However, if all agents can consider actions of other potentially cooperative agents, then agents can interpret the actions and act more cooperatively.

Based on these intuitions, we propose a novel Q values decomposition that considers both the return of an agent acting independently and cooperating with other observable agents. In this decomposition, our method expands the original observation of each agent. The expanded observation consists of the local observation and observable agents' actions. In this decomposition, agents consider others' actions and take the optimal action for cooperation instead of always acting independently. However, there are still two problems. First, if all agents need others' actions to act, the agent who takes action first would not be able to access others' actions. Additionally, neural networks cannot handle the expanded observation as they have the fixed input dimensions, and the number of observable agents is dynamically changing. To address these two problems, we propose a greedy action searching method to search for all agents' cooperative actions. In our method, we first make each agent takes action based on the optimistic belief that all observable agents will cooperate with him to solve the first problem. Second, we further decompose the Q values to obtain a network model that can flexibly handle the varying dimension observation. We indicate that the greedy action searching method can improve the exploration of cooperative behaviors and has a better representation of the optimal policy. Moreover, this structure can naturally adapt to the changes in total agents' numbers and learn a flexible policy to ad hoc team play MARL situation. Besides, we utilize an auxiliary loss related to environmental cognition consistency to assist training. Finally, we use the prioritized experience replay (PER) buffer to focus on samples that contribute more to the tasks. In summary, our method has three contributions as follows.

1) We propose a novel Q values decomposition considering others' actions to address the nonmonotonic problem.
2) We propose a greedy action searching method that can handle the varying dimension observation and adapt our method to the ad hoc scenarios.
3) We use an auxiliary loss and PER which are both proposed based on the unique structure of our method to assist training.

We conduct extensive experiments on both monotonic tasks' environment StarCraft multi-agent challenge (SMAC) [12] and nonmonotonic tasks' environment MAgent [13]. The results demonstrate that our method can efficiently solve the nonmonotonic tasks that current methods struggle with and can also perform well in monotonic tasks. Our additional analysis experiments also demonstrate the ability of our method to adapt to ad hoc team play situation.

II. Related Work

The progresses of deep reinforcement learning give rise to an increasing effort of designing general-purpose deep MARL methods for complex multi-agent environments, including counterfactual multi-agent policy gradients (COMA) [14], multi-agent deep deterministic policy gradient (MADDPG) [15], population-based training (PBT) [16], multi-actor-attention-critic (MAAC) [17], asynchronous experience replay learning (AERL) [18], etc. In this article, we focus on fully cooperative environments. The mainstream methods are CTDE [19] methods. In CTDE [19], [20] extend independent Q-Learning [21] to use deep Q-network (DQN) to learn Q values for each agent independently. VDNs [7], which learns the joint-action Q values by factoring them as the sum of each agent's Q values. QMIX [8] extends VDN to allow the joint action Q values to be a monotonic combination of each agent's Q values that can vary depending on the state. However, the monotonic constraints on the joint action values introduced by QMIX and similar QMIX-class methods result in provably poor exploration and relative overgeneralization [22]. To address this problem, duPLEX dueling multi-agent Q-learning (QPLEX) [23] and Q-learning with Transition-based Auxiliary Tasks for Multi-Agent Cooperation (QTRAN) [24] aim to learn value functions with complete expressiveness capacity. However, they are reported to perform poorly when used in practice because learning the complete expressiveness is impractical in complicated MARL tasks due to the challenging exploration in large joint action space [25], [26]. Other methods attempt to solve the problem in many different ways, such as multi-agent variational exploration (MAVEN) [27] hybridizes value and policy-based methods by introducing a latent space for hierarchical control. This allows MAVEN to achieve committed, temporally extended exploration. Weighted QMIX [9] is based on QMIX and rectifies the suboptimally by introducing a weighting into the projection to place more importance on the better joint actions. Furthermore, reinforcement learning with task decomposition (RLTD) [28] learns to decompose the holistic reward signal for each agent into multiple parts according to the subtasks. Unshaped networks for multi-agent systems (UNMAS) [29] adapt to the number and size changes in multi-agent cooperative tasks.

However, all these methods cannot estimate the value of actions considering the changes in other agents' actions, which is essential for cooperation in nonmonotonic environments. Our method is different from these methods because our method learns a policy which considers other agents' actions and can enable the policy to explore more about the optimal joint actions to tackle the nonmonotonic problem. Furthermore, our method can adapt to different environment settings that other methods cannot.

TABLE I
PAYOFF MATRIX (LEFT) AND INCORRECT PAYOFF LEARNED BY QMIX (RIGHT)

8	-12	-12
-12	0	0
-12	0	0

-12	-12	-12
-12	0	0
-12	0	0

Our work is also related to PER [30] in this work, which is a common method to promote sample efficiency in single-agent environments. Lastly, our work is related to the problem of ad hoc team play [10] in multi-agent cooperative environments. Past works [31], [32], [33] usually require strong domain knowledge or sophisticated online learning at test time. This is infeasible in complex real-world situations [34]. In contrast, we solve the problem in a more general way that does not need any adjustments during test time.

III. Analysis of Nonmonotonicity

In this section, we present the limitation of QMIX or similar algorithms like VDN that they cannot represent the true optimal action-value function in some cases. Based on the analysis in [9] and [27], these methods would underestimate the optimal joint action value in nonmonotonic environments. Intuitively, monotonicity implies that the optimal action of agent i does not depend on the other agents' actions. However, agents are usually obliged to cooperate with others' actions to solve the task in fully cooperative environments. For example, the payoff matrix in Table I [24] produces a value function for which QMIX's approximation (right) does not result in the correct argmax (left). In such a case, the policy of QMIX will not be able to achieve cooperation.

IV. Method

In this section, we propose a novel Q values decomposition and a greedy action searching method to make all agents consider taking cooperative actions instead of focusing on the optimal action of themselves and tackle the nonmonotonic problem. Based on this, we propose an auxiliary loss and a PER buffer to assist training.

A. Basic Setting

A fully cooperative multi-agent sequential decision-making task can be described as a decentralized partially observable Markov decision process (Dec-POMDP), which is defined by a set of states S describing the possible configurations of all N agents, a set of possible actions $A_1, \ldots, A_N$, and a set of possible observations $O_1, \ldots, O_N$. At each time step, each agent chooses an action $a \in A \equiv 1, \ldots, k$, forming a joint action $u \in U \equiv U_n$. The joint action u produces the next state by a transition function $P : S \times U \rightarrow S$. The observation of each agent is updated by an observation function $O_p : S \rightarrow O$. All agents share the same reward $r : S \times U \rightarrow R$ and with a joint value function $Q_\pi = E_{s_{t+1}:\infty, a_{t+1}:\infty}[R_t | s_t, u_t]$ where $R_t = \sum_{j=0}^{\infty} \gamma^j r_{t+j}$ is the discounted return. Furthermore, to clearly explain the notations used in the article, we list all main function notations in Nomenclature.

B. Q Values Decomposition

In MARL setting, the main shortcoming of Q values decomposition is that each agent acts independently, not cooperating with others at each time step. Therefore, if all agents can consider others' actions at all times, they will interpret the actions and act more cooperatively.

To achieve this, we propose a novel Q values decomposition that considers both the return of the agent acting independently and the return of cooperating with others. Commonly, in fully cooperative situations, an agent gets rewards from two sides: the reward that the agent can gain by acting on its own and the reward gained by the whole team when all agents in the team are taking the optimal cooperative actions. Here, we define the collection of agents who can potentially cooperate with each other as a team. Specifically, following the decentralized execution principle, each agent should only be able to cooperate with the agents who can be observed by themselves. The reason is that the decentralized policy can only take an action according to information within the local observation. If an agent needs to cooperate with other agents who are not in its observation, we must introduce the communication method to transfer the necessary messages. Otherwise, the agents cannot even figure out which agents can potentially achieve cooperation, which will make the example of cooperation irregular from the view of local observation and the cooperative policy become difficult to learn. Therefore, the team can be viewed as the collections of agents who can observe each other, as shown in Fig. 1. The detailed division process of teams is included in Algorithm 1, where the set $\mathbf{T_i}$ represents each team and we finally have the set $\mathbf{C}$ as the collection of all teams in the environment.

Algorithm 1 Division of Teams
```
1: Initialize all agents view range $\mathbf{V}$ according to environment settings
2: **while** not end of episode **do**
3:     Initialize an empty set $\mathbf{C}$
4:     **for** i = 1 to N **do**
5:         Initialize an empty set $\mathbf{T_i}$
6:         **for** j = 1 to N and j != i **do**
7:             Calculate the distances between agent $\mathbf{A_i}$ and agent $\mathbf{A_j}$ as $\mathbf{D_j}$
8:             **if** $\mathbf{D_j} < \mathbf{V_i}$ **then**
9:                 $\mathbf{T_i} = \mathbf{T_i} + \mathbf{A_j}$
10:    **for** i = 1 to N **do**
11:        $\mathbf{C} = \mathbf{C} + \mathbf{T_i}$
```

Therefore, we have

$$r_{\text{tot}}(s, u) = \sum_{i=0}^{N} (r_a(o_i, a_i)) + \sum_{j=0}^{G} \left(r_c\left(o_j^{\text{joint}}, u_j\right)\right) \tag{1}$$

where r_{tot} means the total return from the environment under state s and joint action u. r_a is the return reward that is only related to each agent itself, o_i and a_i is the observation and action of each agent. r_c is the return of each small team, o_j^{joint} stands for the local joint observation of all agents in team j, and u_j is the joint action of the team. N means the total number of agents in the environment and G means the number of teams in the environment. However, we cannot directly calculate the reward decomposition from the environment. Therefore, we propose our value decomposition method to

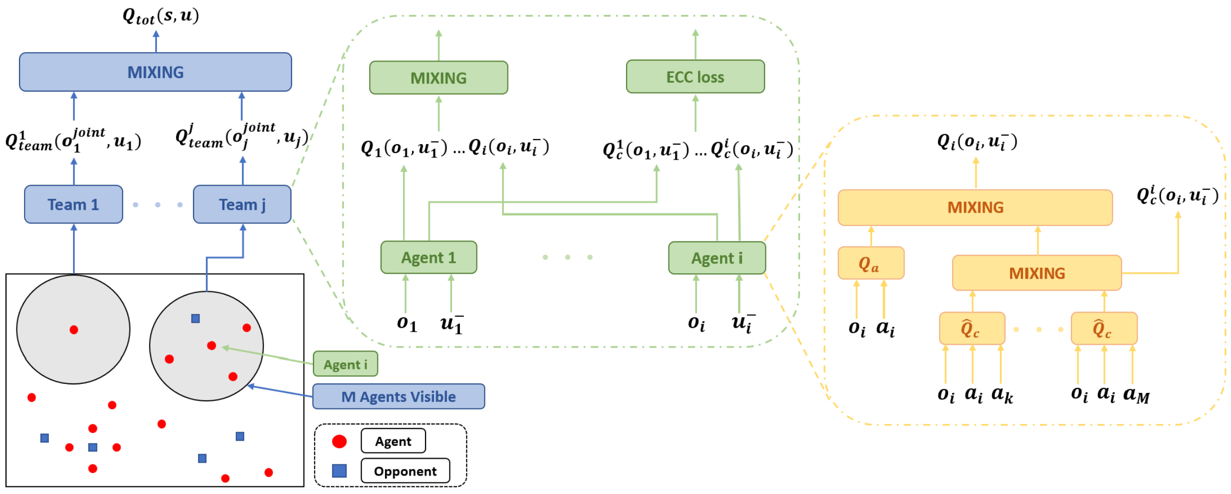


Fig. 1. Architecture of our method. Left: The architecture for total Q values. Each agent belongs to a team composed of all agents whom the agent can observe. If there are no agents in the local view, the team consists of a single agent. Teams with the same agent composition are considered as the same team. Middle: How the Q values of team j and the ECC loss are computed. Right: The architecture used for Q values of a single agent.

approximate the global Q value $Q_{\text{tot}}(s,u)$ which is similar to other CTDE methods.

Based on the decomposition of the reward, we have

$$\begin{aligned} Q_{\text{tot}}(s,u) &= \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t r_{\text{tot}}(s_t,u_t) \mid \pi\right] \\ &= \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t \sum_{i=0}^{N}(r_a(o_i,a_i)) \mid \pi\right] \\ &\quad + \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t \sum_{j=0}^{G}(r_c(o_j^{\text{joint}},u_j)) \mid \pi\right]. \end{aligned} \tag{2}$$

To decompose $Q_{\text{tot}}(s,u)$, we consider decomposing both items in (2) separately. If we have $Q_a^i(o_i,a_i)$ stands for the agent i's expected future return of $r_a(o_i,a_i)$, we can transfer the first item in (2) item into

$$\begin{aligned} \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t \sum_{i=0}^{N}(r_a(o_i,a_i)) \mid \pi\right] &= \sum_{i=0}^{N} Q_a^i(s,u) \\ &= \sum_{i=0}^{N} Q_a^i(o_i,a_i). \end{aligned} \tag{3}$$

The reason behind this is that for each agent, the Q value $Q_a^i(s,u)$ of agent i means the expected return of acting independently, which is only related to the agent's own observation and action (o_i,a_i). A more intuitively understanding is that when all agents have no interaction with each others, the $r_c(o_j^{\text{joint}},u_j)$ is all zeros. $r_{\text{tot}}(s,u)$ is the sum of $r_a(o_i,a_i)$, so the $Q_{\text{tot}}(s,u)$ equals to the sum of all agents' $Q_a^i(o_i,a_i)$.

The second item in (2) is related to the cooperative reward return. We have $Q_c^i(o_j^{\text{joint}},u_j)$ means the agent i (in team j)'s expected future return of $r_c(o_j^{\text{joint}},u_j)$. First, we know that the cooperative return is shared among all agents who participate in the cooperation. In this way, the $Q_c^i(o_j^{\text{joint}},u_j)$ of each agent in the same team j should be equal, and each of them can represent the total Q values of team j noted as $Q_{\text{ctot}}^j(o_j^{\text{joint}},u_j)$. We have the unbiased estimate Q values of all G teams in the environment by computing the average of $Q_c^i(o_j^{\text{joint}},u_j)$ in each team

$$\sum_{j=0}^{G} Q_{\text{ctot}}^j\left(o_j^{\text{joint}},u_j\right) = \sum_{j=0}^{G}\left(\frac{1}{M_j}\cdot\sum_{i=0}^{M_j} Q_c^i\left(o_j^{\text{joint}},u_j\right)\right) \tag{4}$$

where M_j is the number of agents in team j. Additionally, each team's return is only related to the team's own state and action (o_j^{joint},u_j), we have (5) similar to (3)

$$\sum_{j=0}^{G} Q_{\text{ctot}}^j(s,u) = \sum_{j=0}^{G} Q_{\text{ctot}}^j\left(o_j^{\text{joint}},u_j\right). \tag{5}$$

Then we bring (4) and (5) into (2)'s second item, we have (6)

$$\begin{aligned} &\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t \sum_{j=0}^{G}\left(r_c\left(o_j^{\text{joint}},u_j\right)\right) \mid \pi\right] \\ &\quad = \sum_{j=0}^{G} Q_{\text{ctot}}^j(s,u) \\ &\quad = \sum_{j=0}^{G}\left(\frac{1}{M_j}\cdot\sum_{i=0}^{M_j} Q_c^i\left(o_j^{\text{joint}},u_j\right)\right). \end{aligned} \tag{6}$$

Finally, we have the decomposition of $Q_{\text{tot}}(s,u)$ as

$$Q_{\text{tot}}(s,u) = \sum_{i=0}^{N} Q_a^i(o_i,a_i) + \sum_{j=0}^{G}\left(\frac{1}{M_j}\cdot\sum_{i=0}^{M_j} Q_c^i\left(o_j^{\text{joint}},u_j\right)\right). \tag{7}$$

C. Greedy Action Searching Policy Based on Optimistic Belief

The proposed Q values decomposition function overcomes the nonmonotonic limitation of past methods like QMIX by considering the behavior of others. However, it is still infeasible to learn the proper Q values function. First, each team's agent number is dynamically changing no matter in an ad hoc team play environment or an environment with fixed agents number. The value function network with a fixed dimension of input cannot handle such a situation. Furthermore, the u_j in $Q_c^i(o_j^{\text{joint}}, u_j)$ is the joint action of all agents in team j which makes the potential action space grow exponentially as the number of agents in the team grows. These make the learning of $Q_c^i(o_j^{\text{joint}}, u_j)$ inconvenient.

To overcome the uncertainty of the number of agents and enable decentralized execution, we consider finding an alternative form of the $Q_c^i(o_j^{\text{joint}}, u_j)$ function of agent i. Fortunately, as $Q_c^i(o_j^{\text{joint}}, u_j)$ is only related to agents in team j, we can decompose it within the team. First, when an agent considers cooperating with others, what affects the agent's policy is the actions that other agents are taking, not the observations that other agents see. The reason is that all potential cooperative agents are in the agent's field of view, so the agent's observation includes all necessary observation information about other agents which is needed for cooperation. Therefore, since $Q_c^i(o_j^{\text{joint}}, u_j)$ is the value function of agent i, we can simplify it by replacing o_j^{joint} with o_i. In this way, we can have the fixed size of local observation input no matter how the team changes. Second, we notice that we only care about finding $a_i \sim \arg\max(Q_a^i(o_i, a_i) + Q_c^i(o_i, u_j^-, a_i))$ where u_j^- is the joint action of other agents. Naturally, we consider taking u_j^- as the input observation instead of the output actions. We have

$$Q_c^i\left(o_j^{\text{joint}}, u_j\right) \approx Q_c^i(o_i, u_j) = Q_c^i\left(\left(o_i, u_j^-\right), a_i\right). \tag{8}$$

However, agents are unable to select actions without knowledge of u_j^- which are inaccessible during execution. A simple idea to address this is that we fix the order in which all agents take actions. The agent who acts first cannot access any others' actions, and the last one can observe all other's actions. It is clear that this method relies on the predefined order of actions and cannot get an accurate value function. Moreover, accessing others' actions requires communication between agents which prevents decentralized execution. To solve this problem, we propose a greedy action searching method based on optimistic belief assuming that for any agent, all the agent's teammates will adopt the optimal actions to cooperate with the agent. In other words, the greedy action searching method does not need the true u_j^- of agent i. It optimistically uses the u_j^{-*} that can lead to the maximization value of a_i. Therefore, our method still enables decentralized execution. In this way, the greedy action searching policy of agent i can be represented as

$$\pi_i(a_i|o_i) = \underset{a_i \in A}{\arg\max}\left(Q_a^i(o_i, a_i) + Q_c^i\left(\left(o_i, u_j^{-*}\right), a_i\right)\right). \tag{9}$$

The policy represented by greedy action searching method has many advantages. First, we find that the greedy action searching method can promote cooperation by increasing exploration and jumping out of the suboptimal policy. Because the suboptimal policy is often a policy where the agent is worried that other agents will not cooperate with itself and take negative action (i.e., staying still) to ensure that it will not be punished. Using the greedy action searching method can increase the possibility of taking the cooperative actions. Thereby, improving the exploration of cooperative behaviors which helps to converge to the optimal policy.

Furthermore, we have

Theorem 1:

$$\begin{aligned} &Q_c^i\left(\left(o_i, u_j^{-*}\right), a_i^*\right) - Q_c^i\left(\left(o_i, u_j^{-*}\right), a_i'\right) \\ &\qquad \geq Q_c^i\left(\left(o_i, u_j^-\right), a_i^*\right) - Q_c^i\left(\left(o_i, u_j^-\right), a_i'\right) \end{aligned} \tag{10}$$

where u_j^{-*} is the optimal joint actions of other agents in team j, a_i^* is the optimal action, and a_i' is other suboptimal action of agent i.

Proof: Since we know the greedy action searching policy can be view as

$$\pi_i(a_i|o_i) = \underset{a_i \in A}{\arg\max}\left(Q_a^i(o_i, a_i) + Q_c^i\left(\left(o_i, u_j^{-*}\right), a_i\right)\right) \tag{11}$$

where the u_j^{-*} are the optimal joint actions of other agents in team j. The original policy that uses u_j^- instead of u_j^{-*} is

$$\pi_i^o(a_i|o_i) = \underset{a_i \in A}{\arg\max}\left(Q_a^i(o_i, a_i) + Q_c^i\left(\left(o_i, u_j^-\right), a_i\right)\right) \tag{12}$$

where the u_j^- is the ground true joint actions that interact with the environment of other agents in team j. For the same a_i, we have

$$\begin{aligned} \Delta Q_i(o_i, a_i) &= \left(Q_a^i(o_i, a_i) + Q_c^i\left(\left(o_i, u_j^{-*}\right), a_i\right)\right. \\ &\quad \left. - \left(Q_a^i(o_i, a_i) + Q_c^i\left(\left(o_i, u_j^-\right), a_i\right)\right)\right. \\ &= Q_c^i\left(\left(o_i, u_j^{-*}\right), a_i\right) - Q_c^i\left(\left(o_i, u_j^-\right), a_i\right). \end{aligned} \tag{13}$$

Let a_i^* be the optimal action that we want to find. We take a_i^* into the (13)

$$\Delta Q_i\left(o_i, a_i^*\right) = Q_c^i\left(\left(o_i, u_j^{-*}\right), a_i^*\right) - Q_c^i\left(\left(o_i, u_j^-\right), a_i^*\right). \tag{14}$$

Since we know that only when a_i equals a_i^*, the team can complete the cooperative task. Any other a_i' will lead to a suboptimal state. So, we have

$$Q_c^i\left(\left(o_i, u_j^{-*}\right), a_i^*\right)\right) \geq Q_c^i\left(\left(o_i, u_j^{-*'}\right), a_i'\right)\right) \geq Q_c^i\left(\left(o_i, u_j^{-*'}\right), a_i^*\right)\right) \tag{15}$$

where $u_j^{-*'}$ is u_j^{-*} under a suboptimal a_i'. Thus, we have

$$\Delta Q_i\left(o_i, a_i^*\right) = Q_c^i\left(\left(o_i, u_j^{-*}\right), a_i^*\right) - Q_c^i\left(\left(o_i, u_j^-\right), a_i^*\right) \geq 0. \tag{16}$$

Then we take the suboptimal a_i' in (13)

$$\Delta Q_i\left(o_i, a_i'\right) = Q_c^i\left(\left(o_i, u_j^{-*}\right), a_i'\right) - Q_c^i\left(\left(o_i, u_j^-\right), a_i'\right). \tag{17}$$

Similarly, we have

$$Q_c^i\left(\left(o_i, u_j^{-*'}\right), a_i'\right)\right) \geq Q_c^i\left(\left(o_i, u_j^{-*}\right), a_i'\right)\right). \tag{18}$$

We find that we have

$$\begin{aligned} \Delta Q_i\left(o_i, a_i'\right) &= Q_c^i\left(\left(o_i, u_j^{-*}\right), a_i'\right) - Q_c^i\left(\left(o_i, u_j^-\right), a_i'\right) \leq 0 \\ \Delta Q_i\left(o_i, a_i^*\right) &= Q_c^i\left(\left(o_i, u_j^{-*}\right), a_i^*\right) - Q_c^i\left(\left(o_i, u_j^-\right), a_i^*\right) \geq 0. \end{aligned} \tag{19}$$

In other words

$$\begin{aligned} &Q_c^i\left(\left(o_i, u_j^{-*}\right), a_i^*\right) - Q_c^i\left(\left(o_i, u_j^{-*}\right), a_i'\right) \\ &\qquad \geq Q_c^i\left(\left(o_i, u_j^-\right), a_i^*\right) - Q_c^i\left(\left(o_i, u_j^-\right), a_i'\right). \end{aligned} \tag{20}$$

This completes the proof. □

Theorem 1 indicates the excellent property of the greedy action searching policy that it has a better representation of the optimal cooperative policy by increasing the distinction between optimal and suboptimal actions. As when taking the optimal action, it has higher Q values and when taking the suboptimal action, it has lower Q values. This can also prove that the greedy action searching policy has no bias compared to the typical greedy policy. Thus, we can promise

$$a_i^* = \arg\max_{a_i \in A} \left(Q_a^i(o_i, a_i) + Q_c^i\left(\left(o_i, u_j^{-*}\right), a_i\right)\right). \quad (21)$$

However, we still have to go through all possible combinations of u_j^- to find u_j^{-*} which leads to the maximization of $Q_c^i((o_i, u_j^-), a_i)$ using (9). The time complexity is $O(N^2)$ which is hard computationally. Moreover, using u_j^{-*} still faces the problem that the number of observable agents is dynamically changing. Fortunately, due to the property of Q_c^i, we can view the function in a decomposition form. First, we have

Assumption 1:

$$Q_c^i\left(\left(o_i, a_i\right), u_j^{-*}\right) \approx \sum_{k=0, k\neq i}^{M_j} \left(\hat{Q}_c^i\left(\left(o_i, a_i\right), a_k^*\right)\right) \quad (22)$$

where $\hat{Q}_c^i((o_i, a_i), a_k^*)$ is the decomposition of $Q_c^i(o_j^{\text{joint}}, u_j)$, a_k^* means the optimal cooperative action that agent i believes agent k in the team will do. And M_j is the number of agents in team j.

Analysis: Let $r_j((o_i, a_i), u_j^{-*})$ stands for the cooperative reward given (o_i, a_i) in team j. We find $r_j((o_i, a_i), u_j^{-*})$ can decompose into

$$r_j\left(\left(o_i, a_i\right), u_j^{-*}\right) = r_j^0\left(\left(o_i, a_i\right), a_0^*\right) + \cdots + r_j^k\left(\left(o_i, a_i\right), a_k^*\right) + \cdots + r_j^{M_j}\left(\left(o_i, a_i\right), a_{M_j}^*\right) \quad (23)$$

where $r_j^k((o_i, a_i), a_k^*)$ can be viewed as a reward that measures the contribution of agent k to cooperative tasks under (o_i, a_i). Similarly, this reward decomposition is actually an intermediate result of the Q value decomposition process. It is not necessary to directly calculate the reward decomposition from the environment. We still use a value decomposition to approximate the Q values. Because the $r_j((o_i, a_i), u_j^{-*})$ is the return reward that given only when the specific cooperative task is finished. If a_i is not a_i^* that can finish the task, $r_j((o_i, a_i), u_j^{-*})$ should equals zero as the reward of each agent acting independently has already been represented by $r_a(o_k, a_k)$ of each agent. And each item $r_j^k((o_i, a_i), a_k^*)$ in the decomposition should equal zero as well. When a_i is a_i^*, we assume that each agent contributes equally to the task, so the $r_j((o_i, a_i), u_j^{-*})$ can be decomposed no matter the cooperation is achieved or not. In this way, we have

$$\begin{aligned} Q_c^i\left(\left(o_i, a_i\right), u_j^{-*}\right) &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_j\left(\left(o_i, a_i\right), u_j^{-*}\right) \mid \pi\right] \\ &= \mathbb{E}\left[\sum_{k=0}^{M_j} \sum_{t=0}^{\infty} \gamma^t \left(r_j^k\left(\left(o_i, a_i\right), a_k^*\right)\right) \mid \pi\right] \\ &= \sum_{k=0}^{M_j} \left(\hat{Q}_c^i\left(\left(o_i, a_i\right), u_j^{-*}\right)\right). \end{aligned} \quad (24)$$

Furthermore, the action-value function $\hat{Q}_c^i((o_i, a_i), u_j^{-*})$ is the expected future return of $r_j^k((o_i, a_i), a_k^*)$, which could be expected to depend more strongly on actions a_k^* under observation (o_i, a_i). In this way, we have

$$\sum_{k=0}^{M_j} \left(\hat{Q}_c^i\left(\left(o_i, a_i\right), u_j^{-*}\right)\right) \approx \sum_{k=0}^{M_j} \left(\hat{Q}_c^i\left(\left(o_i, a_i\right), a_k^*\right)\right). \quad (25)$$

Thus

$$Q_c^i\left(\left(o_i, a_i\right), u_j^{-*}\right) \approx \sum_{k=0}^{M_j} \left(\hat{Q}_c^i\left(\left(o_i, a_i\right), a_k^*\right)\right). \quad (26)$$

This completes the analysis. □

According to Assumption 1, we have

$$\begin{aligned} Q_c^i\left(\left(o_i, u_j^{-*}\right), a_i\right) &= Q_c^i\left(\left(o_i, a_i\right), u_j^{-*}\right) \\ &\approx \sum_{k=0, k\neq i}^{M_j} \left(\hat{Q}_c^i\left(\left(o_i, a_i\right), a_k^*\right)\right) \\ &= \sum_{k=0, k\neq i}^{M_j} \left(\hat{Q}_c^i\left(\left(o_i, a_k^*\right), a_i\right)\right). \end{aligned} \quad (27)$$

Intuitively, the decomposition indicates that when searching for a_k^* of agent k in the team, we only need to consider agent k cooperating with agent i to find the optimal cooperative action a_k^*. We believe this assumption may have some limitations like the approximation can be inaccurate when the value function is not well trained, which may add randomness during the training process. However, this assumption is correct on the whole, and it has been proved by both above analysis and experiments.

Now the policy can be represented as

$$\pi\left(a_i | o_i\right) = \arg\max_{a_i \in A} \left(Q_a^i(o_i, a_i) + \sum_{k=0, k\neq i}^{M_j} \left(\hat{Q}_c^i\left(\left(o_i, a_k^*\right), a_i\right)\right)\right). \quad (28)$$

Theorem 2: The time complexity of finding the maximization of

$$Q_a^i(o_i, a_i) + \sum_{k=0, k\neq i}^{M_j} \left(\hat{Q}_c^i\left(\left(o_i, a_k^*\right), a_i\right)\right) \quad (29)$$

is $O(N)$.

Proposition 1: Let $\{Y_1, \ldots, Y_n\}$ be estimates of $\{X_1, \ldots, X_n\}$ that are conditionally unbiased in that $E(Y_i | X_1, \ldots, X_n) = X_i$ for all i. Let i^* denote the alternative with the maximal estimated value $Y_{i^*} = \max(Y_1, \ldots, Y_n)$. Then

$$E(X_{i^*} - Y_{i^*}) \leq 0. \quad (30)$$

If X_{i^*} is the maximal in $\{X_1, \ldots, X_n\}$ then $E(X_{i^*} - Y_{i^*}) = 0$. This result has been proven in previous work [35].

Proof: We want to find a_i^* by searching for the a_k^* that lead to the maximization of

$$Q_i(o_i, a_i) = Q_a^i(o_i, a_i) + \sum_{k=0}^{M_j} \left(\hat{Q}_c^i\left(\left(o_i, a_k^*\right), a_i\right)\right). \quad (31)$$

According to proposition 1, for each $a_i \in A$, we have

$$\begin{aligned}\max_{a_k\in A}(Q_i(o_i,a_i)) &= \max_{a_k\in A}\left(Q_a^i(o_i,a_i)+\sum_{k=0}^{M_j}\left(\hat{Q}_c^i((o_i,a_k),a_i)\right)\right)\\ &\le \max_{a_k\in A}\left(Q_a^i(o_i,a_i)\right)\\ &\quad+\sum_{k=0}^{M_j}\left(\max_{a_k\in A}\left(\hat{Q}_c^i((o_i,a_k),a_i)\right)\right)\\ &= \sum_{k=0}^{M_j}\left(\max_{a_k\in A}\left(\hat{Q}_c^i((o_i,a_k),a_i)\right)\right). \qquad (32)\end{aligned}$$

Thus, the searching time of finding the maximization of each a_i is $O(M) \times O(A)$, where M is the average number of agents in the team and A is the dimension of action space. As we want to find the maximization of all a_i, we need to go through all a_i, so the total searching time is $O(M) \times O(A^2)$. The A is a constant number, so the total time complex is $O(N)$. This completes the proof. □

Theorem 2 indicates that we can use such a decomposition to find the optimal action within in linear computation time. In this way, we manage to have a model with flexible input dimensions and reduce the searching time complexity from $O(N^2)$ to $O(N)$. Finally, we have the decomposition of Q_{tot}

$$\begin{aligned}Q_{\mathrm{tot}}(s,u) \approx \sum_{j=0}^{G}\left(\frac{1}{M_j}\cdot\sum_{i=0}^{M_j}\sum_{k=0,k\ne i}^{M_j}\left(\hat{Q}_c^i\left(\left(o_i,a_k^*\right),a_i\right)\right)\right)\\ +\sum_{i=0}^{N}Q_a^i(o_i,a_i). \qquad (33)\end{aligned}$$

D. Environmental Cognition Consistency Loss

As our method is based on Q-Learning, our method can be trained end-to-end to minimize the following loss:

$$\begin{aligned}\mathcal{L}_Q(\theta) = \mathbb{E}_\pi\Bigg[&\Big(Q(s_t,u_t;\theta)\\ &-\Big[r(s_t,u_t)+\gamma\max_{u_{t+1}}Q(s_{t+1},u_{t+1};\theta')\Big]\Big)^2\Bigg] \qquad (34)\end{aligned}$$

where t is the time step and θ' means the parameter of the target network. Notable, although we use the searching method to find u_j^* when interacting with the environment, we use the true u_j when updating the Q values function. Because all transitions have been stored in the replay buffer, there is no problem accessing others' actions. We can use the actual joint actions that lead to the transitions to train a precise Q values function. In such a case, the optimized policy is not the behavior policy. We usually need to use importance sampling to update the Q values functions. However, we prove that the off-policy updating process of our method does not need the importance sampling.

Proof: The update of our Q values function can be viewed as

$$Q(s_t,u_t) \leftarrow r(s_t,u_t)+\gamma\max_{u_{t+1}}Q(s_{t+1},u_{t+1}) \qquad (35)$$

where the u_t is from the greedy action searching policy

$$\begin{aligned}\pi(u_t|s_t) \leftarrow \Big(&\arg\max_{a_1\in A}\left(Q_a^1(o_1,a_1)+Q_c^1\left(\left(o_1,u_1^{-*}\right),a_1\right)\right)\\ &\ldots\arg\max_{a_i\in A}\left(Q_a^i(o_i,a_i)+Q_c^i\left(\left(o_i,u_j^{-*}\right),a_i\right)\right)\ldots\Big) \qquad (36)\end{aligned}$$

and u_{t+1} is the greedy action chosen by the optimized policy

$$\begin{aligned}&\pi^{\mathrm{op}}(u_{t+1}|s_{t+1})\\ &\leftarrow \Big(\arg\max_{a_1\in A}\left(Q_a^1(o_1,a_1)+Q_c^1\left(\left(o_1,u_1^{-}\right),a_1\right)\right)\\ &\qquad\ldots\arg\max_{a_i\in A}\left(Q_a^i(o_i,a_i)+Q_c^i\left(\left(o_i,u_j^{-}\right),a_i\right)\right)\ldots\Big). \qquad (37)\end{aligned}$$

We find that although our method uses a different policy to interact with the environment, the $r(s_t,u_t)$ is not influenced by the interacting policy like the Q-Learning method. Furthermore, since we use one-step updating, the target Q values function's action is chosen by maximizing the optimized policy Q values function. There is no bias when updating the Q values. In fact, we also use the ϵ-greedy method in the greedy action searching policy. However, this has no influence on the updating for the same reason. This completes the proof. □

Moreover, the decomposition so far does not promise the $Q_c^i(o_j^{\mathrm{joint}},u_j)$ of agents in the same team j is equal. To ensure this, we propose an environmental cognition consistency loss (ECC loss) as an auxiliary loss to promote training process. The auxiliary loss can be viewed as

$$\begin{aligned}\mathcal{L}_e(\theta) = \mathbb{E}_\pi\Bigg[\sum_{j=0}^{G}\Bigg(\frac{1}{M_j}\sum_{i=0}^{M_j}\Bigg(&M_j\cdot Q_c^i(o_j^{\mathrm{joint}},u_j)\\ &-\sum_{k=0}^{M_j}\left(Q_c^k\left(o_j^{\mathrm{joint}},u_j\right)\right)\Bigg)^2\Bigg)\Bigg]. \qquad (38)\end{aligned}$$

Notable, we use $Q_c^i(o_j^{\mathrm{joint}},u_j)$ in (38) just for conciseness of expression, we use the decomposition form of $Q_c^i(o_j^{\mathrm{joint}},u_j)$ in actual calculation. By minimizing auxiliary loss, we can make the difference of each $Q_c^i(o_j^{\mathrm{joint}},u_j)$ zero in the same team. And since we know the auxiliary loss is related to cooperation, we make it equals to zero when no successful cooperation event is detected during the episode. These events are easy to detect because they are consistent with the task to be completed. Thus, we have the overall loss objective

$$\mathcal{L}(\theta) = \mathcal{L}_Q(\theta)+\lambda\cdot\mathcal{L}_e(\theta) \qquad (39)$$

where λ is positive multipliers to control the optimization ratio of these two losses.

E. PER With ECC Loss

A few works use PER buffer [30] to improve the sample efficiency of the learning process. However, the original PER buffer is designed for single-agent reinforcement learning.

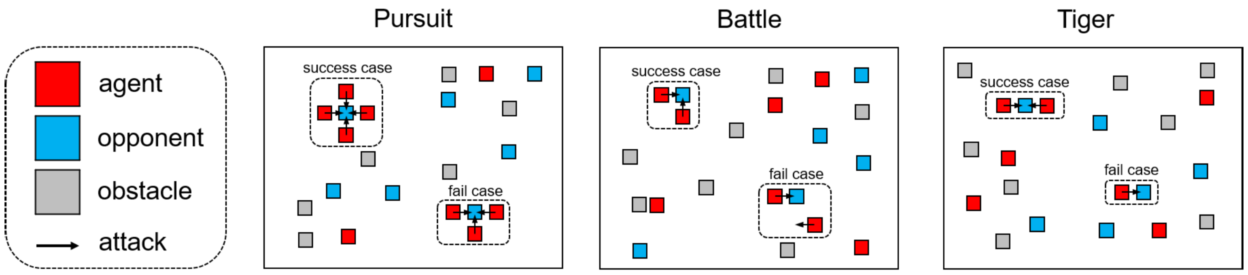


Fig. 2. Illustrations of the experimental environments. Left: Pursuit scenario where agents need to surround the preys and attack it together (upper). A fail case is attacking the preys while not surrounding it (bottom). Middle: Battle scenario where agents need to form a group to attack enemy and kill it (upper). Fail cases are attacking enemy alone but not kill it or attacking when no enemy around (bottom). Right: Tiger scenario where tigers need to attack deer together but not kill it (upper). A fail case is that the tiger attacks deer on its own (bottom).

TABLE II

SETTINGS OF SCENARIOS FROM EASY TO HARD

	Pursuit easy	Pursuit hard	Battle easy	Battle hard	Tiger easy	Tiger hard
Agents number	8	8	5	5	10	5
Map size	180 × 180	270 × 270	80 × 80	100 × 100	40 × 40	50 × 50
Wall number	20	20	20	200	40	40

It uses temporal difference (TD) error as the priority of sampling which is not suitable for a MARL situation. In the fully cooperative settings, we would like to make the learning process focus on the transitions related to collaboration. However, TD error cannot distinguish the transition is related to collaboration or not, as it is only determined by training times.

To overcome this, we propose a novel PER buffer that uses our ECC loss as the priority instead of TD-error. The ECC loss is closely related to the cooperation process, which can guide the agents to learn more about cooperation.

V. EXPERIMENTS

A. Environments and Training Settings

For the experiments, we evaluate all methods in two different kinds of environments with monotonic tasks and nonmonotonic tasks. We choose SMAC environment as the monotonic tasks environment as the optimal action of each agent does not depend on other agents' actions [25], [36]. In this environment, methods such as VDN or QMIX do not suffer from the monotonic restriction in these tasks and can represent the optimal policy.

However, in nonmonotonic tasks where the optimal action of each agent depends on other agents' actions, these methods fail to learn the optimal policy. On the contrary, our method can solve the nonmonotonic tasks efficiently. Therefore, we compare our method with other methods in nonmonotonic tasks to present the ability of our method to tackle the monotonic restriction. For the nonmonotonic tasks' experiments, we adopt a grid-world platform MAgent [13]. In MAgent, each agent corresponds to one or multiple grids and has a local observation that contains a square view centered at the agent and a feature vector including coordinates, health point (HP) and ID of agents nearby, and the agent's last action. The discrete actions are moving, staying, attacking, etc. To verify the effectiveness and versatility of our method, we designed easy and hard scenarios for all three tasks, namely *pursuit*, *battle*, and *tiger*. Each task has a unique mission, as shown in Fig. 2, which requires cooperation between agents to complete. There are the detailed settings of these scenarios, as shown in Table II.

The global state that used by reward-optimized decision-making (RODE), QMIX, and MAVEN is a mini map of the global information in MAgent and the default global state in SMAC. The opponent's policies used in experiments are randomly escaping policy in *pursuit* and *tiger* and pretrained policy in *battle*. For view range, we set the view ranges of agents to five grids in *pursuit*, four grids in *tiger*, and six grids in *battle*, and we use the default view range setting in SMAC. We run all the experiments three times with different random seeds. We set the discount factor as 0.99 and use the RMSprop optimizer with a learning rate of $5e-4$. The ϵ-greedy is used for exploration with ϵ annealed linearly from 1.0 to 0.05 in 700 k steps. The batch size is 32 and updating the target every 200 episodes. The length of each episode is limited to 350 steps in MAgent. The λ that is used to control the optimization ratio of the ECC loss is 0.05. Specially, RODE involves extra parameters, including number of roles is 3 and role interval is 5. All experiments are carried out on the same computer, equipped with an Intel i7-7700 K, 32 GB RAM, and an NVIDIA GTX1080Ti. The system is Ubuntu 18.04 and the framework is PyTorch.

In the experiments, we compare our method with VDN [7], QMIX [8], MAVEN [27], RODE [37], and QTRAN [24]. We use a network structure consists of two CNN layers and two hidden layers in MAgent and use a network structure consists of three multi-layer perceptron (MLP) layers with 64 units in SMAC for our method, VDN, and QMIX. Specifically, the kernel size of CNN layer is three and each hidden layer has 512 units with ReLU nonlinearities. The network of RODE, MAVEN, and QTRAN use the same LSTM network, which consists of a recurrent layer composed of a GRU with a 512-D hidden state in MAgent and 64-D hidden state in SMAC, with one fully connected layer before and two after. The rest hyperparameters are the same as the former ones. Notable, RODE, QMIX, and MAVEN have access to the global state while other methods just utilize the local observations.

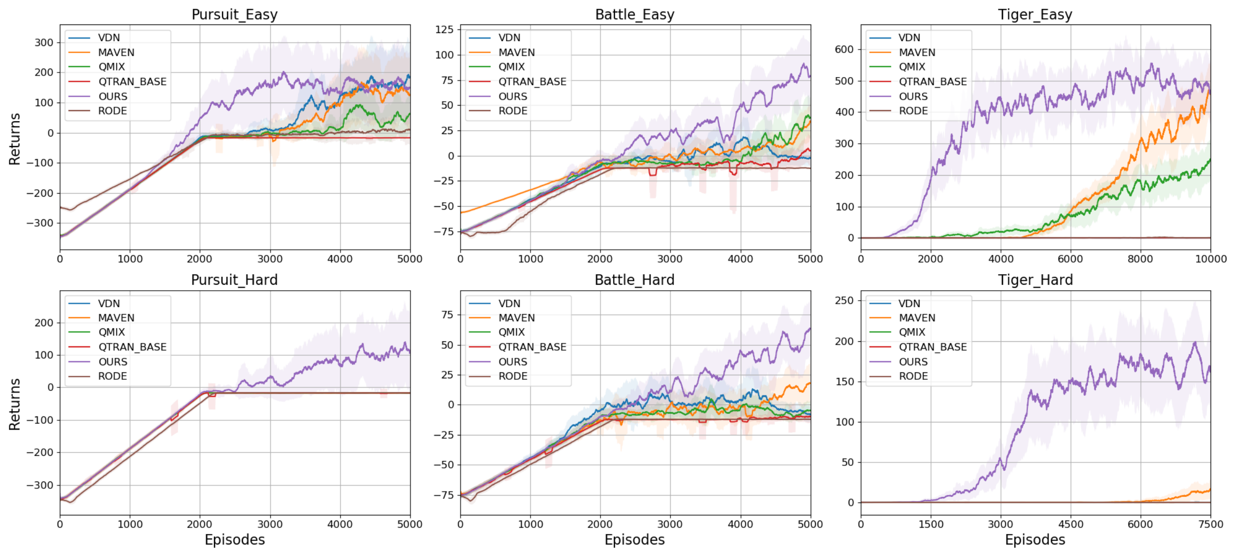


Fig. 3. Experiments in three MAgent domain. Upper left: Results in easy pursuit scenario. Upper middle: Results in easy battle scenario. Upper right: Results in easy tiger scenario. Lower left: Results in hard pursuit scenario. Lower middle: Results in hard battle scenario. Lower right: Results in hard tiger scenario.

B. Performance on MAgent Scenarios

We show the results of six different scenarios from easy to hard in Fig. 3. In *pursuit*, agents need to learn to surround and attack preys. The easy scenario has a smaller map size that the hunters can encounter the preys more frequently and easily surround them. On the contrary, the hard scenario has a vast map that the preys have more space to escape, which means the hunter must cooperate tightly to surround preys. The results show that our method outperform all comparison methods significantly. We find that some comparison methods can eventually catch up with our method with a longer learning process in the easy scenario. However, all these methods cannot exert excellent performance in the hard scenario. The reason is that the effective attack after rounding up is challenging to achieve in hard scenarios. These methods not considering teammates' actions tend to converge to a local optimal policy, which never attacks to avoid the punishment of ineffective attacks. However, such a policy is also impossible to execute an effective attack. On the opposite, our method can increase the exploration of cooperative behaviors and converge to the cooperative policy in the hard scenario.

In *battle*, agents learn to fight against enemies who have superior abilities than the agents. The superior enemies used in the experiment are agents pretrained by VDN. To keep the balance of battlefield, after the death of an agent or enemy, we will add a new agent or enemy at a random location. The easy scenario has fewer walls and a smaller map size making it easier for agents to gather as a group or focus fire. As estimated in the easy scenario, we find that only our method, QMIX and MAVEN learn the proper policy to gather as a group and focus fire. Meanwhile, all other methods learn suboptimal policies such as escaping from the enemies and rarely firing back. When it comes to the hard scenario, the result shows that our method still converges to the optimal policy even if the walls obstruct gathering and focusing fire. However, it is hard to escape from enemies with too many walls blocking the ways. Thus, these suboptimal policies have a lower reward in the hard scenario.

In *tiger*, the task of agents is to pincer deer together, which is similar to *pursuit*. However, agents in *tiger* need to learn to let the deer escape, not kill the deer immediately as the deer can recover HP and tigers can attack deer after the recovery to get higher rewards. The easy scenario has more agents and a smaller map size that agents can switch attack targets conveniently. The results show that our method learns the optimal policy in both easy and hard scenarios. Meanwhile, only QMIX and MAVEN learn a suboptimal policy in the easy scenario and all the comparing methods fail in hard scenarios. We believe that QMIX and MAVEN have a better performance than other compared methods is because they have access to the global state information. Finally, we find RODE has poor performance in these scenarios. We believe this is because the RODE has more hyperparameters like role numbers, etc. We did not find the feasible parameters for the current environment, thus causing an undesirable performance.

In general, our method has achieved better performance in six scenarios of three environments than all other methods, including RODE, QMIX, and MAVEN, which can use global state information. These results prove the effectiveness of our method in nonmonotonic multi-agent cooperative scenarios.

C. Performance on SMAC Scenarios

We also conduct several experiments in the SMAC environment scenarios to demonstrate that our method can also work in monotonic tasks. In SMAC, all maps have classified as easy, hard, and super hard. We use two hard scenarios *3s_vs_4z* and *5m_vs_6m* as well as a super hard scenario *corridor* for experiments. The results are shown in Fig. 4. The results demonstrate that our method still has relatively better performance in SMAC scenarios. In the easiest scenario *3s_vs_4z*, the result shows that VDN and QMIX have faster converge speed, which is because they have suitable simpler structures for easy tasks and do not face the nonmonotonic issue in SMAC scenarios. However, our method has superior performance in all other two harder scenarios. This indicates

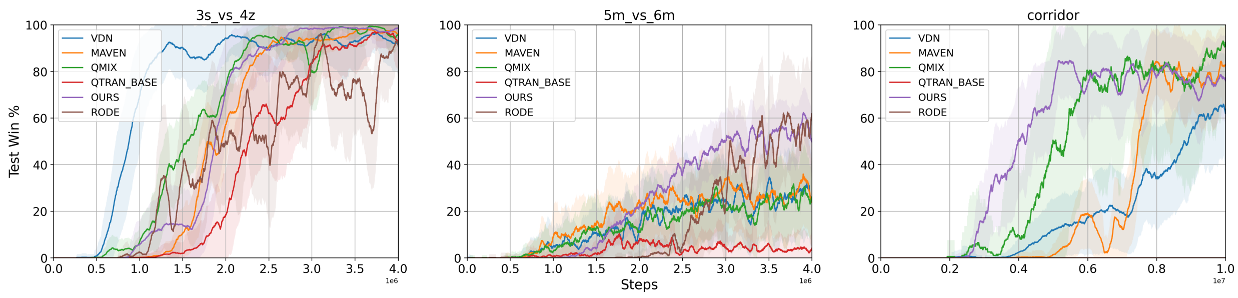


Fig. 4. Results of SMAC experiments. Left: Results in *3s_vs_4z* scenario. Middle: Results in *5m_vs_6m* scenario. Right: Results in *corridor* scenario.

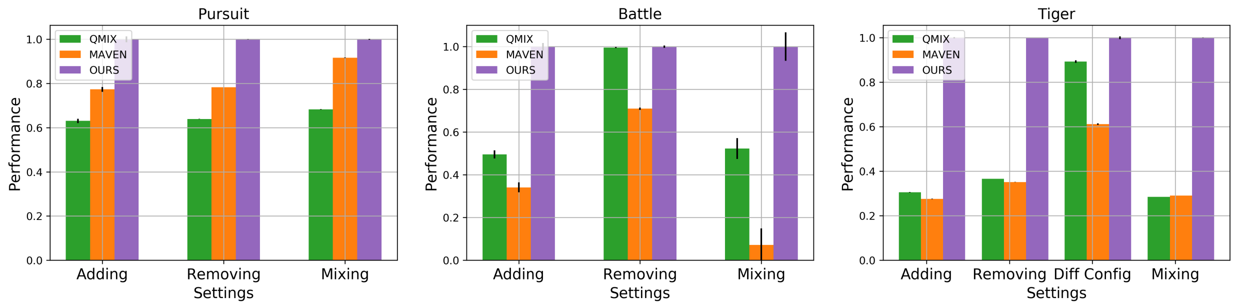


Fig. 5. Results of ad hoc team play settings in all three environments. Left: Results in easy pursuit scenario. Middle: Results in easy battle scenario. Right: Results in easy tiger scenario. For a clear demonstration, we normalize the y-axis coordinates and set the performance of our method to 1.0.

TABLE III
SETTINGS OF AD HOC TEAM PLAY

	Pursuit	Battle	Tiger
Adding agents number	+8	+2	+5
Removing agents number	-4	-2	-5
Changing map size map size	+90 × +90	-	+40 × +40
Changing wall number	-	+180	-
Changing agent configuration	-	-	+1 attack point

that our method can apply to monotonic tasks and can solve the nonmonotonic tasks that other methods struggle with.

D. Ad Hoc Team Play Experiments

Now we demonstrate that our method is robust to the changing settings during test time. We test the ad hoc team play in changing the following settings: adding or removing agents, changing environment settings such as map size or the number of walls, and changing the configuration of agents such as adding attacking damage. The adding, removing, and changing agents' configuration scenarios mean only adding, removing agents, or changing agents' configuration. Particularly, the mixing settings mean changing the environment setting as well as adding or removing agents. The mixing setting of *pursuit* is the combination of adding agents (+8) and changing map size (+90). In *battle*, the mixing setting is the combination of removing agents (−2) and changing wall number (+180). The mixing setting of *tiger* is the combination of adding agents (+3) and changing map size (+40). We also test a setting that changes configuration of the agent in *tiger*. We add one point to the attack point in this setting. The detailed implementations of all ad hoc team play settings are shown in Table III. The "-" in the table means do not change the original parameters. The adding or removing agents' settings are directly implemented as described in the table. We estimate the final model's performance in the ad hoc team plays settings as the results. All tested models are well trained in three easy scenarios. All methods are tested 50 episodes in each setting with different random seeds and plot the mean/std in the figure.

From Fig. 5, we can see that our method can generalize better to the ad hoc team play test settings in all three environments. In *pursuit*, our method outperforms other methods largely no matter adding or removing agents. The result can prove that our method has learned to cooperate flexibly, as no matter how teammates change, our method can still produce the optimal action to complete the task. However, when testing in a larger map with more agents, our method has a lower advantage than the comparison method. We believe this is because the preys are too sparse on the large map that agents has to spend more steps to surround them, which leads to lower returns. In *battle*, the results are similar that our method can handle ad hoc team play settings perfectly. However, we notice that MAVEN fails in the setting which changes the number of walls in the environment. We believe this is because MAVEN needs to generate z variables based on the global state at the beginning of the episode to control the behavior mode of the agents. Thus, the changing in environment static items (e.t., walls) can affect the policy and lead to poor performance. We add a unique setting (Diff config in Fig. 5) that changes the attacking damage point of agents in *tiger*. We notice that our method fails to outperform other methods a lot in this setting compared to other three settings. The reason is that our method learns a policy that attacks a deer and then stops the attack for a few steps. However, adding attacking damage point can kill the deer faster and agents have to chase other deer again. The chasing process gets more challenging as fewer deer leaves and causes a drop in our method's performance. The result shows the limitation of our method that our method can only

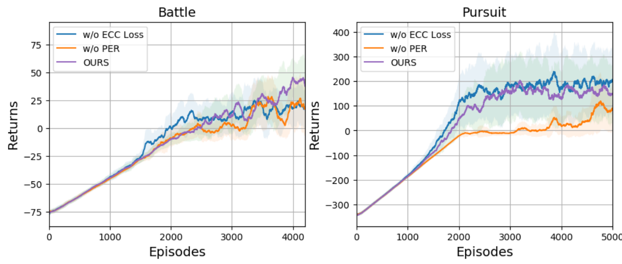


Fig. 6. Ablation experiments on training without theECC Loss and without the PER. Left: Results in easy battle scenario. Right: Results in easy pursuit scenario.

handle the ad hoc team play setting in team level but not in the agent level.

E. Ablations

We conduct several ablations on the *pursuit* and *battle* scenarios. We first consider training without the ECC loss. Furthermore, we try to train without the novel PER buffer. The ablations results are shown in Fig. 6. The experiment shows that the PER buffer improves the sample efficiency as the policy trained with PER buffer learns faster in both scenarios. Furthermore, the ECC loss promotes the training process, as the policy trained without ECC loss has a more unsatisfactory performance in *battle*. However, we notice that the ECC loss has no significant improvement in *pursuit*. We believe it is likely that the task is simpler in *pursuit* that the cooperation requires a fixed number of agents (i.e., four agents to surround). Thus, the bias of the learned value function may not affect the final performance.

VI. Conclusion

In this work, we propose a novel Q values decomposition that considers both the return of an agent acting independently and cooperating with other observable agents to address the nonmonotonic problem of current value-based CTDE methods. We propose a greedy action searching method that can deal with the dynamically changing number of observable agents and the agents' pending order of actions to find cooperative actions. In this way, our method can naturally adapt to various agents' numbers. Moreover, we utilize an auxiliary loss related to environmental cognition consistency and a modified PER buffer to assist training. We test our method in both monotonic and nonmonotonic tasks. Specifically, we test our method in three challenging nonmonotonic MAgent domains with six scenarios from easy to hard and in monotonic SMAC tasks. Additionally, we conduct experiments in different ad hoc team play settings. The experimental results show that our method achieves significant performance improvements in all domains, especially the nonmonotonic domains that current methods struggle with, and generalizes in the ad hoc team play settings.

References

[1] Y. Cao, W. Yu, W. Ren, and G. Chen, "An overview of recent progress in the study of distributed multi-agent coordination," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 427–438, Feb. 2012.

[2] G. Chen and Y.-D. Song, "Cooperative tracking control of nonlinear multiagent systems using self-structuring neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 8, pp. 1496–1507, Aug. 2014.

[3] Y. Li, C. Yang, W. Yan, R. Cui, and A. Annamalai, "Admittance-based adaptive cooperative control for multiple manipulators with output constraints," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 12, pp. 3621–3632, Dec. 2019.

[4] M. Hüttenrauch, A. Šošić, and G. Neumann, "Guided deep reinforcement learning for swarm systems," 2017, *arXiv:1709.06011*.

[5] J. Jiang, C. Dun, T. Huang, and Z. Lu, "Graph convolutional reinforcement learning," 2018, *arXiv:1810.09202*.

[6] T. Wang, H. Dong, V. Lesser, and C. Zhang, "ROMA: Multi-agent reinforcement learning with emergent roles," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 1–12.

[7] P. Sunehag et al., "Value-decomposition networks for cooperative multi-agent learning," 2017, *arXiv:1706.05296*.

[8] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4295–4304.

[9] S. Whiteson, "Weighted QMIX: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 10199–10210.

[10] P. Stone, G. Kaminka, S. Kraus, and J. Rosenschein, "Ad hoc autonomous agent teams: Collaboration without pre-coordination," in *Proc. AAAI Conf. Artif. Intell.*, vol. 24, no. 1, 2010, pp. 1504–1509.

[11] C. L. Baker, J. Jara-Ettinger, R. Saxe, and J. B. Tenenbaum, "Rational quantitative attribution of beliefs, desires and percepts in human mentalizing," *Nature Hum. Behav.*, vol. 1, no. 4, pp. 1–10, Mar. 2017.

[12] M. Samvelyan et al., "The StarCraft multi-agent challenge," 2019, *arXiv:1902.04043*.

[13] L. Zheng et al., "MAgent: A many-agent reinforcement learning platform for artificial collective intelligence," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, no. 1, 2018, pp. 8222–8223.

[14] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proc. AAAI Conf. Artif. Intell.*, 2018, vol. 32, no. 1, pp. 1–9.

[15] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," 2017, *arXiv:1706.02275*.

[16] M. Jaderberg et al., "Human-level performance in 3D multiplayer games with population-based reinforcement learning," *Science*, vol. 364, no. 6443, pp. 859–865, May 2019.

[17] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 2961–2970.

[18] Z. Pu, H. Wang, Z. Liu, J. Yi, and S. Wu, "Attention enhanced reinforcement learning for multi agent cooperation," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Feb. 18, 2022, doi: 10.1109/TNNLS.2022.3146858.

[19] L. Kraemer and B. Banerjee, "Multi-agent reinforcement learning as a rehearsal for decentralized planning," *Neurocomputing*, vol. 190, pp. 82–94, May 2016.

[20] A. Tampuu et al., "Multiagent cooperation and competition with deep reinforcement learning," *PLoS ONE*, vol. 12, no. 4, Apr. 2017, Art. no. e0172395.

[21] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proc. 10th Int. Conf. Mach. Learn.*, 1993, pp. 330–337.

[22] L. Panait, S. Luke, and R. P. Wiegand, "Biasing coevolutionary search for optimal multiagent behaviors," *IEEE Trans. Evol. Comput.*, vol. 10, no. 6, pp. 629–645, Dec. 2006.

[23] J. Wang, Z. Ren, T. Liu, Y. Yu, and C. Zhang, "QPLEX: Duplex dueling multi-agent Q-learning," 2020, *arXiv:2008.01062*.

[24] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi, "QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5887–5896.

[25] T. Gupta, A. Mahajan, B. Peng, W. Böhmer, and S. Whiteson, "UneVEn: Universal value exploration for multi-agent reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 3930–3941.

[26] L. Wan, Z. Liu, X. Chen, H. Wang, and X. Lan, "Greedy-based value representation for optimal coordination in multi-agent reinforcement learning," 2021, *arXiv:2112.04454*.

[27] A. Mahajan, T. Rashid, M. Samvelyan, and S. Whiteson, "MAVEN: Multi-agent variational exploration," 2019, *arXiv:1910.07483*.

[28] C. Sun, W. Liu, and L. Dong, "Reinforcement learning with task decomposition for cooperative multiagent systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 5, pp. 2054–2065, May 2021.

[29] J. Chai et al., "UNMAS: Multiagent reinforcement learning for unshaped cooperative scenarios," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 4, pp. 1–12, Apr. 2023.

[30] D. Horgan et al., "Distributed prioritized experience replay," 2018, *arXiv:1803.00933*.

[31] S. Barrett, P. Stone, and S. Kraus, "Empirical evaluation of ad hoc teamwork in the pursuit domain," in *Proc. AAMAS*, 2011, pp. 567–574.

[32] D. Chakraborty and P. Stone, "Cooperating with a Markovian ad hoc teammate," in *Proc. AAMAS*, 2013, pp. 1085–1092.

[33] M. Woodward, C. Finn, and K. Hausman, "Learning to interactively learn and assist," in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, no. 3, pp. 2535–2543.

[34] T. Zhang et al., "Multi-agent collaboration via reward attribution decomposition," 2020, *arXiv:2010.08531*.

[35] J. E. Smith and R. L. Winkler, "The optimizer's curse: Skepticism and postdecision surprise in decision analysis," *Manage. Sci.*, vol. 52, no. 3, pp. 311–322, Mar. 2006.

[36] J. Hu, S. Jiang, S. A. Harding, H. Wu, and S.-W. Liao, "Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning," 2021, *arXiv:2102.03479*.

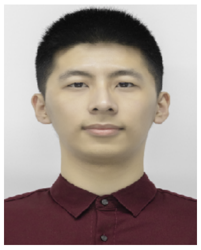
[37] T. Wang, T. Gupta, A. Mahajan, B. Peng, S. Whiteson, and C. Zhang, "RODE: Learning roles to decompose multi-agent tasks," 2020, *arXiv:2010.01523*.

Shanqi Liu received the B.S. degree in control science and engineering from Zhejiang University, Hangzhou, China, in 2019, where he is currently pursuing the Ph.D. degree with the Department of Control Science and Engineering, Institute of Cyber Systems and Control.

His research interests include reinforcement learning and robotics.

Weiwei Liu is currently pursuing the Ph.D. degree with the Advanced Perception on Robotics and Intelligent Learning Laboratory, College of Control Science and Engineering, Zhejiang University, Hangzhou, China.

At present, his main research directions are artificial intelligence and decision-making and control.



Wenzhou Chen received the B.S. degree in automation from the Zhejiang University of Technology, Hangzhou, China in 2017, and the Ph.D. degree from the Department of Control Science and Engineering, Institute of Cyber Systems and Control, Zhejiang University, Hangzhou, in 2022.

He is currently a Lecturer with the College of Computer Science, Hangzhou Dianzi University, Hangzhou. His current research interests include industrial robot, deep reinforcement learning, multimodal learning, and robot decision-making.

Guanzhong Tian (Member, IEEE) received the B.S. degree from the Harbin Institute of Technology, Harbin, China, in 2010, and the Ph.D. degree from Zhejiang University, Hangzhou, China, in 2021.

He is currently a Research Associate with the Ningbo Research Institute, Zhejiang University. His research interests include computer vision, model compression, and embedded AI.

Jun Chen received the M.S. degree in automation from Zhejiang University, Hangzhou, China, in 2020, where he is currently pursuing the Ph.D. degree with the Department of Control Science and Engineering, Institute of Cyber Systems and Control.

His research interests include neural network quantization and deep learning.

Yao Tong received the master's degree from Chinese Academy of Sciences, in 2018.

She is currently an Engineer. Her research interests include command and control and intelligent decision-making.

Junjie Cao received the M.S. and Ph.D. degrees in mechanical engineering (mechatronics) and control science and engineering from Zhejiang University, Hangzhou, Zhejiang, China, in 2017 and 2021, respectively.

He is currently working with the College of Control Science and Engineering, Zhejiang University. His current research interests include machine learning, sequential decision making, and robotics.

Yong Liu received the B.S. degree in computer science and engineering and the Ph.D. degree in computer science from Zhejiang University, Hangzhou, China, in 2001 and 2007, respectively.

He is currently a Professor with the Department of Control Science and Engineering, Institute of Cyber Systems and Control, Zhejiang University. He has published more than 30 research articles in machine learning, computer vision, information fusion, and robotics. His research interests include machine learning, robotics vision, information processing, and granular computing.