

UNMAS: Multi-Agent Reinforcement Learning for Unshaped Cooperative Scenarios

Jiajun Chai, *Student Member, IEEE*, Weifan Li, *Student Member, IEEE*, Yuanheng Zhu, *Senior Member, IEEE*, and Dongbin Zhao, *Fellow, IEEE* Zhe Ma, Kewu Sun, Jishi Yu Ding

Abstract—Multi-agent reinforcement learning methods such as VDN, QMIX, and QTRAN that adopt centralized training with decentralized execution (CTDE) framework have shown promising results in cooperation and competition. However, in some multi-agent scenarios, the number of agents and the size of action set actually vary over time. We call these *unshaped scenarios*, and the methods mentioned above fail in performing satisfyingly. In this paper, we propose a new method called Unshaped Networks for Multi-Agent Systems (UNMAS) that adapts to the number and size changes in multi-agent systems. We propose the self-weighting mixing network to factorize the joint action-value. Its adaption to the change in agent number is attributed to the nonlinear mapping from each-agent Q value to the joint action-value with individual weights. Besides, in order to address the change in action set, each agent constructs an individual action-value network that is composed of two streams to evaluate the constant environment-oriented subset and the varying unit-oriented subset. We evaluate UNMAS on various StarCraft II micro-management scenarios and compare the results with several state-of-the-art MARL algorithms. The superiority of UNMAS is demonstrated by its highest winning rates especially on the most difficult scenario 3s5z_vs_3s6z. The agents learn to perform effectively cooperative behaviors while other MARL algorithms fail in. Animated demonstrations and source code are provided in <https://sites.google.com/view/unmas>.

Index Terms—multi-agent, reinforcement learning, StarCraft II, centralized training with decentralized execution.

I. INTRODUCTION

MULTI-AGENT reinforcement learning (MARL) employs reinforcement learning to solve the multi-agent system problems. With cooperative multi-agent systems playing an increasingly important role, such as controlling robot swarms with limited sensing capabilities [1], [2], mastering build order production [3] and micro-management task in real-time strategy (RTS) games [4]–[6], dispatching ride requests [7], autonomous driving [8]–[10], and so on, MARL attracts the attention of many researchers. MARL algorithms face the problem of huge action space, non-stationary environment, and global exploration. Although many algorithms have been proposed to solve these problems, it is still an open problem.

To address cooperative tasks, one class of MARL methods is independent learning that allows the agents to learn

independently [11]–[13]. Independent learning suffers from the non-stationarity because other agents are also impacting the environment. Another class is centralized learning that takes the multi-agent system as a whole [14]–[16]. *Centralized training with decentralized execution* (CTDE) is a compromise between independent and centralized learning [5], [17], [18]. It provides local autonomy to agents by decentralized execution and mitigates the problem of the non-stationary environment by centralized training. There are many methods adopting CTDE framework [19]–[22], including QMIX [5], VDN [18], QTRAN [23], and so on.

Unshaped scenario is defined as the scenario where the number of units and the size of action set change over time, which is a common multi-agent scenario. [24] solves the problem of formation control in face of an uncertain number of obstacles. DyAN [25] reconstructs the agent observation into information related to environment and units, and ASN [26] categorizes the agent action set considering the semantics between actions. Although the mentioned methods adapt to unshaped scenarios, there is still room for improvement.

StarCraft II micro-management is a widely used experimental environment to test MARL methods [6]. It is actually an unshaped scenario, in which the death of agents in combat leads to the change in their number. In addition, since the enemy can also be killed, the size of attack action subset of each unit changes as well. However, the existing methods [5], [23] ignore the variation of action set and still provide action-values for invalid actions. This may cause miscalculation of the action-value among all agents. In addition, as their joint action-value functions still collect the meaningless action-values of dead agents, the joint action-value may also be miscalculated.

A. Contribution

In this paper, we focus on the unshaped cooperative multi-agent scenarios, in which the number of agents and the size of action set change over time. Our contribution to the uncertain challenge of MARL is twofold. (i) The self-weighting mixing network is proposed with the network input dimension adapting with the number of agents, so that the joint action-value is estimated more accurately. (ii) The action set of an agent is divided into two subsets: *environment-oriented* subset and *unit-oriented* subset. We use two network streams, the *environment-oriented stream* and *unit-oriented stream*, to calculate the Q values of corresponding actions. Finally, we introduce the training algorithm of UNMAS and conduct experiments to compare with several other MARL algorithms

J. Chai, W. Li, Y. Zhu, and D. Zhao are with the State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China, and are also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China. Z. Ma, K. Sun, and J. Ding are with Xlab, the second academy of CASIC, Beijing 100854, China.

J. Chai and W. Li contribute equally to this paper.

in the StarCraft II micro-management scenarios. The results show that UNMAS achieves the highest winning rates on a wide variety of StarCraft II micro-management maps.

B. Related Work

Research on independent learning begins with the work of [27], in which the author tries to execute Q-learning independently for each agent to achieve cooperation. With the use of deep learning, IQL [12] introduces Deep Q-Network (DQN) [28] into MARL to cope with high-dimensional observations. Some other research that tackles the multi-agent problem with independent learning can be found in [11], [13], [29], [30]. DyAN [25] divides the observation of the agents into environment-oriented and unit-oriented subsets, and employs a GNN to adapt to the change in the number of agents in the expansion from small-scale scenarios to large-scale scenarios. However, independent learning suffers from the non-stationarity of environment, which leads to the difficulty of learning convergence [20].

Centralized learning treats the multi-agent system as a whole. Grid-Net [31] employs an encoder-decoder architecture to generate actions for each grid agent. [32] and [33] propose methods for the multi-agent system in the framework of actor-critic. [34] formulates the problem of multitask into a multi-agent system. However, the centralized learning method is hard to be scaled to larger scenarios, so the CTDE framework becomes popular as a compromise between independent and centralized learning. One spectrum of the CTDE method is the actor-critic method. MADDPG [35], which is developed from DDPG [36], provides a centralized critic for the whole system and a group of decentralized actors for each agent. COMA [14] computes a counterfactual baseline to marginalize out a single agent's action while the other agents' actions are fixed. COMA also provides a centralized critic to estimate the joint Q-function.

The other spectrum is the value-based method. The challenge for value-based methods in the framework of CTDE is how to factorize the joint action-value correctly [20]. VDN [18] factorizes the joint action-value by summing up the individual action-values of each agent. QMIX [5] combines the individual action-values in a non-linear way by a mixing network whose weights and biases are generated according to the multi-agent global state. ASN [26] improves the individual action-value network in VDN and QMIX by considering action semantics between agents. QTRAN [23] guarantees more general factorization by factorizing the joint action-value with a transformation. Q-DPP [20] introduces DPP into MARL tasks to increase the diversity of agents' behaviors. ROMA [19] and RODE [37] allow agents with a similar role to share similar behaviors. Qatten [38] designs a multi-head attention network to approximate joint action-value function. However, such methods do not consider the problem of unshaped scenarios. They assume that the number of agents and the size of action set are fixed, and this assumption limits the applications of these methods.

C. Organization

This paper is organized as follows. Section II introduces the background knowledge about multi-agent reinforcement learning and factorization of the joint action-value. Section III describes UNMAS from three aspects: joint action-value function, individual action-value function, and training algorithm. Section IV shows the experiments and results, and analyzes the learned strategies. Finally, Section V gives the conclusion.

II. BACKGROUND

A. Multi-Agent Reinforcement Learning

We consider a fully cooperative multi-agent task with partially observable environment, in which agents observe and take actions individually. This task is also called the *decentralized partially observable Markov decision process* (Dec-POMDP) [39]. It can be defined as a tuple $\mathcal{U} = \{\mathbb{D}, \mathbb{S}, \mathbb{U}, \mathbb{T}, \mathbb{O}, R, \gamma\}$. $\mathbb{D} = \{1, \dots, n\}$ is the set of agents, the number of which is n . The Dec-POMDP extends POMDP by introducing the set of joint actions \mathbb{U} and joint observations \mathbb{O} . The multi-agent system takes the joint action $\mathbf{u}_t = \{u_{1,t}, \dots, u_{n,t}\}$ according to the joint observation $\mathbf{o}_t = \{o_{1,t}, \dots, o_{n,t}\}$ and gets the immediate reward r_t from environment according to the function $R : \mathbb{S} \times \mathbb{U} \rightarrow \mathbb{R}$. Then, the global state of multi-agent system $s_t \subseteq \mathbb{S}$ is produced according to the transition function \mathbb{T} , which specifies $\Pr(s_{t+1}|s_t, \mathbf{u}_t)$. Finally, γ in tuple \mathcal{U} denotes the discount factor of *discounted cumulative reward*: $G_t = \sum_{j=0}^{\infty} \gamma^j r_{t+j}$.

In the Dec-POMDP, we consider a joint policy π , which is composed of the policies $\pi_i(u_{i,t}|o_{i,t})$ of every agent $i \in \mathbb{D}$. The joint policy has a joint action-value function: $\mathbf{Q}(\mathbf{o}_t, \mathbf{u}_t) = \mathbb{E}_{\mathbf{o}_{t+1:\infty}, \mathbf{u}_{t+1:\infty}}[G_t | \mathbf{o}_t, \mathbf{u}_t]$. The purpose of fully cooperative multi-agent task is to maximize this return.

B. Factorization of Joint Action-Value Function

As mentioned before, value-based methods with CTDE framework need to find an efficient and adaptable way to factorize the joint action-value. A common requirement in the field of CTDE is the *Individual Global Max* (IGM) condition.

Definition 1. For a multi-agent system with n agents, if the optimal joint action is equivalent to the set of agents' actions that make the individual action-value functions get the maximum values, the system is said to satisfy the IGM condition. This statement is formulated as follows:

$$\arg \max_{\mathbf{u}_t} \mathbf{Q}(\mathbf{o}_t, \mathbf{u}_t) = \begin{pmatrix} \arg \max_{u_{1,t}} Q_1(o_{1,t}, u_{1,t}) \\ \vdots \\ \arg \max_{u_{n,t}} Q_n(o_{n,t}, u_{n,t}) \end{pmatrix}. \quad (1)$$

Since the IGM condition is difficult to be verified in practice, the following monotonicity condition is mostly used as its substitute:

$$\frac{\partial \mathbf{Q}(\mathbf{o}_t, \mathbf{u}_t)}{\partial Q_i(o_{i,t}, u_{i,t})} \geq 0, \quad \forall i \in \mathbb{D}. \quad (2)$$

If $\mathbf{Q}(\mathbf{o}_t, \mathbf{u}_t)$ is factorized monotonically as (2), then this way of factorization meets the IGM condition. Under the IGM condition, maximizing the joint action-value is equivalent to

maximizing the action-value of each agent. Thus, the multi-agent system pursues the same goal as every agent and achieves cooperation.

We propose UNMAS in the next section to adapt to the number and size changes in the unshaped scenario.

III. UNSHAPED NETWORKS FOR MULTI-AGENT SYSTEMS

In this section, we propose a new method called Unshaped Networks for Multi-Agent Systems (UNMAS), aiming at helping agents adapt to the change in the unshaped scenario. UNMAS uses the self-weighting mixing network, which is adaptive to the size of input, to factorize the joint action-value. The individual action-value network of agent is specially designed with two network streams to evaluate the actions in the *environment-oriented* subset and *unit-oriented* subset separately. The size of unit-oriented subset is also unshaped.

A. Self-weighting mixing network

Fig. 1 presents the self-weighting mixing network, which approximates the joint action-value function. Since UNMAS adopts the CTDE framework, the joint action-value function is only used in training. The weights and biases of the self-weighting mixing network are produced by a group of hyper networks represented by the yellow blocks in the architecture, and the input of the hyper networks is the global state s_t .

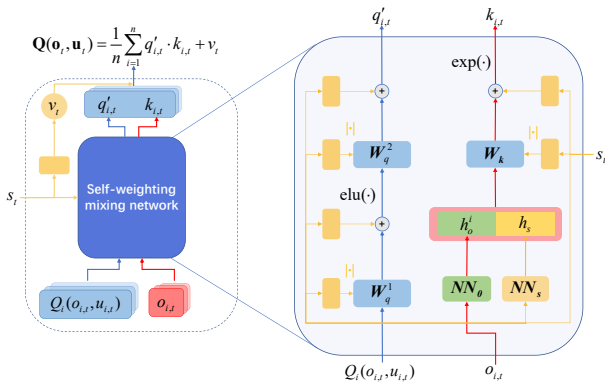


Fig. 1. Self-weighting mixing network architecture. Specifically, the right side of the diagram shows the details of the self-weighting mixing network. Given the observation and action-value of each agent i , it provides $q'_{i,t}$ and $k_{i,t}$ to calculate the joint action-value $\mathbf{Q}(\mathbf{o}_t, \mathbf{u}_t)$ with its bias v_t on the left side.

Instead of following the original definition of the joint action-value function that takes joint observations and joint actions as input, CTDE decomposes it as a mapping from the individual action-values of each agent. Such way greatly decouples the complicated interplay between different agents.

The biggest difference between UNMAS and other CTDE methods lies in the ways of dealing with the input size of joint action-value function. In the other CTDE methods, the input size of joint action-value function is determined beforehand and keeps constant during the whole training and execution phase. Since they still provide action-values for dead agents, the joint action-value may be miscalculated and reduce the performance of agent policies. However, the joint action-value

function represented by self-weighting mixing network can change its input size with the number of agents. It outputs two scalars $q_{i,t}$ and $k_{i,t}$ for each agent:

$$\begin{aligned} q'_{i,t} &= W_q^2 \cdot \text{elu}(W_q^1 \cdot Q_i(o_{i,t}, u_{i,t}; \theta_i) + b_q^1) + b_q^2 \\ k_{i,t} &= W_k \cdot [h_o^i, h_s] + b_k, \end{aligned} \quad (3)$$

where $q'_{i,t}$ is the result after nonlinear mapping by self-weighting mixing network, and $k_{i,t}$ is the individual weight, which describes the contribution of agent i to the joint action-value. By dynamically evaluating the contribution of each agent according to $k_{i,t}$, the estimation of the joint action-value could be more accurate. Thus, the cooperation of the multi-agent system becomes better. Elu [40] is a nonlinear activation function, which is also used in the mixing network of QMIX. $Q_i(o_{i,t}, u_{i,t}; \theta_i)$ is the action-value of agent i , which is estimated by the *individual action-value network*. W_q^1 , W_q^2 , and W_k are weights of the self-weighting mixing network as shown in Fig. 1, and b_q^1 , b_q^2 , and b_k are the corresponding biases. h_o^i and h_s are outputs of networks NN_o and NN_s , which take observation $o_{i,t}$ of agent i and the global state s_t as input, respectively. Finally, the concatenation of h_o^i and h_s is taken as the input of $[W_k, b_k]$ to calculate the weight $k_{i,t}$.

By employing self-weighting mixing network to evaluate the contribution of each agent, UNMAS can adapt to the change in the number of agents in training and improve the accuracy of the estimation of joint action-value. Taking the StarCraft II micro-management scenario as an example, the number of agents decreases due to the death caused by enemies attack. The weights and biases of the self-weighting mixing network are obtained through the global state. Therefore, even if the $q_{i,t}$ of each agent do not take the information of other agents into account, the self-weighting mixing network can still estimate the joint action-value accurately. The joint action-value function is formulated as follows:

$$\mathbf{Q}(\mathbf{o}_t, \mathbf{u}_t; \theta_{joint}) = \frac{1}{n} \sum_{i=1}^n q'_{i,t} \cdot k_{i,t} + v_t, \quad (4)$$

where n is the number of agents in the current timestep t , and v_t is a scalar output from the hyper network. It is a bias of the joint action-value, which is generated from a fully-connected network with s_t as input. The ablation result of QMIX [5] shows that adding v_t to the joint action-value is able to achieve better results, which is also confirmed in our ablation experiments. In (4), in order to adapt to the change in the number of agents, we use n to average the weighted sum $\sum_{i=1}^n q'_{i,t} \cdot k_{i,t}$. Besides, the weights of self-weighting mixing network take the *absolute* values to make sure that the factorization meets the IGM condition. The proof that the self-weighting mixing network meets the IGM condition is provided in Theorem 1.

Theorem 1. *In a fully cooperative task, if letting the self-weighting mixing network shown in Fig. 1 represent the joint action-value function, then the factorization process meets the IGM condition.*

Proof. The joint action-value estimated by the self-weighting mixing network is written as follows:

$$\begin{aligned} \mathbf{Q}(\mathbf{o}_t, \mathbf{u}_t; \boldsymbol{\theta}_{joint}) &= \frac{1}{n} \sum_{i=1}^n q'_{i,t} \cdot k_{i,t} + v_t \\ &= \frac{1}{n} \sum_{i=1}^n [W_q^2 \cdot \text{elu}(W_q^1 \cdot Q_i + b_q^1) + b_q^2] \cdot \\ &\quad \exp(W_k \cdot [h_o^i, h_s] + b_k) + v_t. \end{aligned} \quad (5)$$

The partial derivative of the joint action-value to agent action-value is shown in (6), where all elements are non-negative.

$$\frac{\partial \mathbf{Q}}{\partial Q_i} = \frac{2}{n} \cdot W_q^2 \cdot \frac{\partial \text{elu}(W_q^1 \cdot Q_i + b_q^1)}{\partial Q_i} \cdot \exp(W_k \cdot [h_o^i, h_s] + b_k), \forall i \in \mathbb{D}. \quad (6)$$

Since the parameter α of elu function is set to 1, the derivative $\partial \text{elu}(W_q^1 \cdot Q_i + b_q^1) / \partial Q_i$ shown in (6) can be written as follows:

$$\frac{\partial \text{elu}(W_q^1 \cdot Q_i + b_q^1)}{\partial Q_i} = \begin{cases} W_q^1 \cdot e^{W_q^1 \cdot Q_i + b_q^1}, & W_q^1 \cdot Q_i + b_q^1 \geq 0 \\ W_q^1, & \text{otherwise} \end{cases} \quad (7)$$

As described above, the weights of self-weighting mixing network take absolute values, so the partial derivative $\partial \mathbf{Q} / \partial Q_i$ is also non-negative. According to the monotonicity condition, the factorization meets the IGM condition. \square

Through the special design of network structure and averaging weighted sum with n , self-weighting mixing network can adapt to the change in the number of agents. In the next subsection, we introduce the architecture of the individual action-value network, which can adapt to the change in the size of action set.

B. Individual action-value network

The individual action-value network approximates the action-value function of each agent, and it shares the parameters among agents. In the framework of CTDE, an agent takes an action according to its local observation without communicating with the other agents.

In order to help agents adapt to the change in the size of action set, we divide the action set into two subsets. One is the environment-oriented subset, and the other is the unit-oriented subset. The first action subset represents the interactions between the agent and the environment, and keeps constant during the whole running phase. The second represents the interactions between the agent and other agents, and its size varies with the number of agents in the current environment. Taking StarCraft II micro-management scenario as an example, the environment-oriented actions include *stop* and four *movement* actions in the directions of *up*, *down*, *left*, and *right*, while the unit-oriented action is the attack action aiming at an enemy unit.

Based on the division of the action set, we propose the individual action-value network as shown in Fig. 2. Two network streams are constructed for the two action subsets. The *environment-oriented stream* takes the *environment-oriented observation* as input. It contains three *Fully-Connected* (FC)

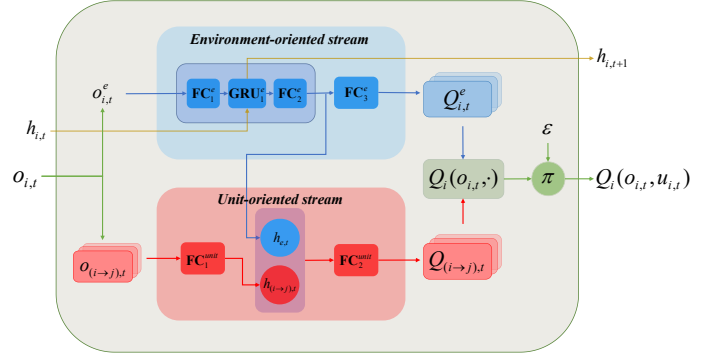


Fig. 2. Individual action-value network architecture. Specifically, it divides the action set of an agent into two subsets: environment-oriented and unit-oriented, and evaluates them through two separate streams.

layers \mathbf{FC}_j^e , $j = 1, 2, 3$, and one *Gated Recurrent Unit* (GRU) [41] layer \mathbf{GRU}^e .

The GRU layer, employed to solve the problem of POMDP, can better estimate the current state. At each timestep t , the GRU layer additionally inputs a hidden state $h_{i,t}$, which contains historical information and outputs the next hidden state. Its outputs are the Q values of the environment-oriented actions as follows:

$$Q_{i,t}^e(o_{i,t}^e, h_{i,t}, \cdot), h_{i,t+1} = \mathbf{NN}_i^e(o_{i,t}^e, h_{i,t}) \quad (8)$$

where $Q_{i,t}^e$ are the Q values of agent i executing the environment-oriented actions. $o_{i,t}^e$ is the environment-oriented observation, which describes the information observed by agent i from the environment. $h_{i,t+1}$ is the hidden state of the next timestep $t+1$. \mathbf{NN}_i^e is the short form of the environment-oriented stream.

The *unit-oriented stream* takes the *unit-oriented observations* as input. Its outputs are the Q values of the unit-oriented actions. The unit-oriented observation $o_{(i->j),t}$ differs from the environment-oriented observation in the fact that it describes the information from agent i to the target unit j . Taking the StarCraft II micro-management scenario as an example, if there are 3 enemies on the map at timestep t , the unit-oriented stream takes 3 observations in terms of these enemies to evaluate the Q values of attacking them. Due to the number of enemies changes over time in unshaped scenarios, the number of attack targets also changes. Besides, it should be noted that if the agent has a healing action, the target units here will be other agents, not enemies. Furthermore, we concatenate the first layer's output of two streams to form a new vector, and feed it into the second layer of the unit-oriented stream. The Q values of the unit-oriented actions are calculated as follows:

$$\begin{aligned} \text{vector} &= [h_{e,t}, h_{(i->j),t}] \\ Q_{(i->j),t}(o_{(i->j),t}, u_{(i->j),t}) &= \mathbf{FC}_2^{\text{unit}}(\text{vector}) \end{aligned} \quad (9)$$

where *vector* is the concatenation of $h_{e,t}$ and $h_{(i->j),t}$, which are the output of the first part of environment-oriented stream and unit-oriented stream. $Q_{(i->j),t}(o_{(i->j),t}, u_{(i->j),t})$ is the Q value of agent i executing the unit-oriented action on target unit j . $\mathbf{FC}_2^{\text{unit}}$ is the second layer of the unit-oriented stream.

Due to the concat operation, the output of the unit-oriented stream also contains the historical information provided by $h_{i,t}$, so there is no need to provide a GRU layer for it. For the sake of brevity, the environment-oriented observations and unit-oriented observations of agent i are simplified into one variable $o_{i,t} = \{o_{i,t}^e, o_{(i \rightarrow j),t}\}$. Finally, we concatenate all $Q_{i,t}^e$ and $Q_{(i \rightarrow j),t}$ to form the individual action-value function $Q_i(o_{i,t}, \cdot)$.

It should be noted that the individual action-value networks of other methods like QMIX fail in adapting to the change in the size of action set. They use a single network to evaluate the Q values of all kinds of actions, which include both valid and invalid actions.

Algorithm 1 Unshaped Networks for Multi-Agent Systems

- 1: Initialize the replay buffer D and exploration rate ϵ ;
 - 2: Initialize θ_{joint} and θ_i with random parameters;
 - 3: Initialize the parameters of target networks θ_i^- and θ_{joint}^- :
 $\theta_i^- = \theta_i, \theta_{joint}^- = \theta_{joint}$;
 - 4: **for** $episode = 1$ **to** M **do**
 - 5: **for** $t = 0, \dots, T$ **do**
 - 6: Collect the global state $s_{i,t}$;
 - 7: **for** each agent i **do**
 - 8: Collect the local observation $o_{i,t}$;
 - 9: Choose a random action with probability ϵ ;
 - 10: Otherwise, estimate action-values $Q_i(o_{i,t}, \cdot; \theta_i)$
 - for each agent with the individual action-value network and choose greedy action following (10);
 - 11: **end for**
 - 12: Execute the joint action \mathbf{u}_t and collect the next joint observation \mathbf{o}_{t+1} , next state s_{t+1} , and reward r_{t+1} ;
 - 13: Store the transition $(\mathbf{o}_t, \mathbf{u}_t, r_{t+1}, \mathbf{o}_{t+1})$ into D ;
 - 14: Store the state transition (s_t, s_{t+1}) into D ;
 - 15: **end for**
 - 16: **if** replay buffer D is full **then**
 - 17: Sample b minibatch from D ;
 - 18: Estimate joint action-value $\mathbf{Q}(\mathbf{o}_k, \mathbf{u}_k; \theta_{joint})$ as (4) by the self-weighting mixing network;
 - 19: Calculate the *update target* y_k^{joint} in (12) and update the networks by the loss $\mathcal{L}(\theta)$ in (11);
 - 20: **end if**
 - 21: Replace target networks with *target update interval*;
 - 22: Decay the exploration rate ϵ ;
 - 23: **end for**
-

C. Training algorithm of UNMAS

In this subsection, we introduce the training algorithm of UNMAS. The detail is shown in Algorithm 1. The agents take ϵ -greedy policy to explore the environment in training, that is, choose a random action with probability ϵ , otherwise choose the action with the highest Q value:

$$u_{i,t} = \arg \max_u Q_i(o_{i,t}, u; \theta_i), \quad (10)$$

where the exploration rate ϵ decays along the training steps. The transitions and rewards generated by the interaction between the multi-agent system and the environment are stored into the replay buffer D .

When the replay buffer D is full, we sample a minibatch from replay buffer D to update the networks. They are trained by minimizing the loss:

$$\mathcal{L}(\theta) = \frac{1}{b} \sum_{k=1}^b [(y_k^{joint} - \mathbf{Q}(\mathbf{o}_k, \mathbf{u}_k; \theta_{joint}))^2], \quad (11)$$

where b is the size of minibatch. $\theta = \{\theta_{joint}, \theta_i\}$ is the parameter of all networks involved in training. θ_{joint} and θ_i are the parameters of self-weighting mixing network and individual action-value network respectively. y_k^{joint} is the *joint update target* of the multi-agent system, which is defined as follows:

$$\begin{aligned} y_k^{joint} &= r_{k+1} + \gamma \mathbf{Q}(\mathbf{o}_{k+1}, \bar{\mathbf{u}}_{k+1}; \theta_{joint}^-) \\ \bar{\mathbf{u}}_{k+1} &= [\arg \max_{u_{i,k+1}} Q_i(o_{i,k+1}, u_{i,k+1}; \theta_i^-)], \end{aligned} \quad (12)$$

where the target action $\bar{\mathbf{u}}_{k+1}$ is obtained by maximizing the individual action-value function of agents. θ_{joint}^- and θ_i^- are parameters of the target network for the self-weighting mixing network and the individual action-value network. Throughout the training phase, they are replaced by the current parameters every fixed episodes, which is defined as *target update interval*.

The multi-agent system is treated as a whole in the training phase, so the self-weighting mixing network and individual action-value network can also be seen as one neural network, which possesses the parameter θ . At each iteration, θ is updated by the gradient that is obtained by minimizing the loss function in (11). Since the self-weighting mixing network only appears in the training phase, this learning law indicates that the role of the self-weighting mixing network and its parameter is to help the update of the individual action-value network.

IV. EXPERIMENTS ON STARCRAFT II MICRO-MANAGEMENT

A. Experimental Setup

The micro-management scenario is an important aspect of StarCraft II. It requires the player to control a group of agents to win a combat against enemies. The micro-management scenario with a decentralized setting is a proper experimental scenario to test MARL methods because the agents are partially observable and can perform decentralized actions. We use StarCraft Multi-Agent Challenge (SMAC) [42] as the environment and evaluate our method on various maps.

We consider combats with two groups of units, one of which is using UNMAS and the other is the enemy. This scenario is an unshaped scenario, as the number of agents and enemies will decrease due to the attack. In this scenario, each agent is partially observable, which means that it could only get the information within its sight range. The observation mainly contains the following attributes for both allies and enemies: *distance, relative_x, relative_y, health, shield, and unit_type*. The global state mainly contains the above information for all units, both agents and enemies. Details of the observation and state are provided in the Appendix A. Each agent has the following actions: *move[direction], stop, and attack[enemy_id]*. Each enemy is controlled by the built-in StarCraft II AI, which



Fig. 3. The experimental scenarios to test the methods.

uses hand-crafted scripts. We set the difficulty of built-in AI as “*very difficult*”.

We take the scenario in Fig. 3(a) as an example to show more details about the uncertainty in StarCraft II micro-management. In this scenario, there are three agents and three enemies at the beginning. Each agent receives two kinds of observations. One kind is environment-oriented represented by a tensor of length 42. It contains the information of agent self (8), agent allies and enemies ($2 * 5 + 3 * 5$), and the one-hot coding of agent last action (9). The other kind is unit-oriented represented by a collection of 3 tensors of length 5. Each tensor describes the information of an enemy (5) observed by this agent. Then, agents select actions from their action sets according to these observations. Their action sets contain the following actions: *move[up]*, *move[down]*, *move[left]*, *move[right]*, *stop*, *attack[enemy_0]*, *attack[enemy_1]*, and *attack[enemy_2]*.

Each agent evaluates actions by the individual action-value network described before. This network takes environment-oriented observations (42) and unit-oriented observations ($3 * 5$) as input, and outputs the values of actions ($6 + 3$). The values of 6 environment-oriented actions and 3 unit-oriented actions are computed by the separate streams. If an enemy dies in combats, the size of unit-oriented observations will become $2 * 5$, and thus the unit-oriented stream just outputs the values of attack actions towards the remaining two enemy units. In this way, the output dimension of the individual action-value network becomes $6 + 2$.

During training, UNMAS samples from the replay buffer and uses the self-weighting mixing network shown in Fig. 1 to compute the joint action-value of the multi-agent system. The input dimension of this network is 3, which corresponds to the number of agents on the map. Furthermore, if an agent dies later in combat, the individual action-value network provides only the action-values of the remaining two agents. Therefore, the input dimension of self-weighting mixing network

becomes 2.

At each timestep, agents perform their actions in the decentralized way and receive a global reward from the environment. SMAC provides positive rewards by default, that is, the damage done by agents to enemy units. In addition, the positive (+10) reward is also provided after an enemy is killed. We evaluate the method by running 32 episodes every 10000 timesteps to get the *test winning rates*. The agents use ϵ -greedy to choose their action and turn it off in testing. The exploration rate ϵ decays from 1 to 0.05.

We compare UNMAS with VDN [18], QMIX [5], QTRAN [23], and ASN [26]. VDN factorizes the joint action-value $\mathbf{Q}(\mathbf{o}_t, \mathbf{u}_t)$ by the summation of the individual action-value function $Q_i(o_{i,t}, u_{i,t}; \theta_i)$:

$$\mathbf{Q}(\mathbf{o}_t, \mathbf{u}_t) = \sum_{i=1}^n Q_i(o_{i,t}, u_{i,t}; \theta_i) \quad (13)$$

where θ_i is the parameter of the network representing the individual action-value function. QMIX employs a mixing network to represent the joint action-value function. This network takes the action-value $Q_i(o_{i,t}, u_{i,t}; \theta_i)$ of all agents as input and the joint action-value $\mathbf{Q}(\mathbf{o}_t, \mathbf{u}_t; \theta_{joint})$ as output, where θ_{joint} is its parameter. By being taken absolute values, the weights of the mixing network keep non-negative. QTRAN factorizes the joint action-value by designing a special learning objective. Therefore, it can transform the original joint action-value function into a new, easily factorizable one with the same optimal actions in both functions. ASN considers the action semantics between agents to compute the action-values and reconstruct the observation of agent i at timestep t as follows: $o_{i,t} = \{o_{i,t}^{env}, m_{i,t}, o_{i,t}^1, \dots, o_{i,t}^{i-1}, o_{i,t}^{i+1}, o_{i,t}^n\}$, where $o_{i,t}^{env}$ is the observation about environment, $m_{i,t}$ is the private information about agent i itself, and $o_{i,t}^j$ is the observation of agent i to other agent j .

Experiments are performed on the following symmetric maps, where agents and enemies have the same numbers

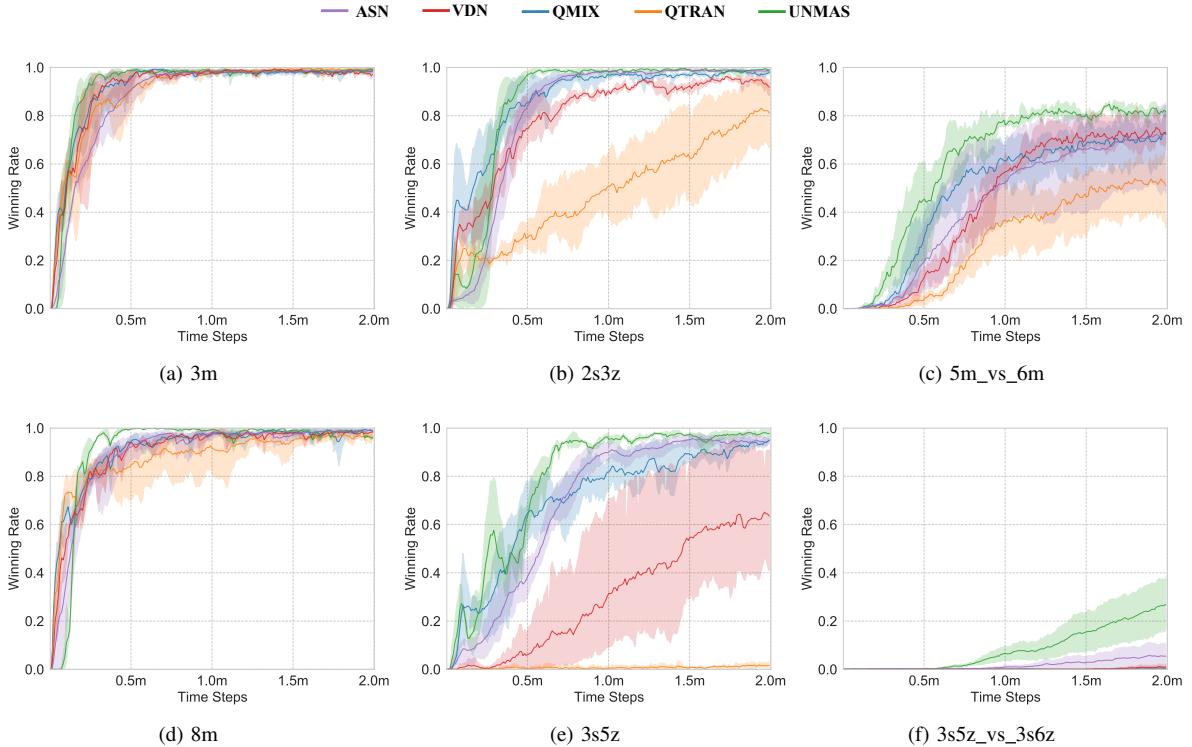


Fig. 4. Winning rates for ASN, VDN, QMIX, QTRAN and UNMAS. UNMAS achieves the highest winning rate in all scenarios, especially the most difficult scenario $3s5z_vs_3s6z$.

and types: 3 Marines ($3m$), 8 Marines ($8m$), 2 Stalkers and 3 Zealots ($2s3z$), 3 Stalkers and 5 Zealots ($3s5z$). Besides, the asymmetric maps including 5 Marines against 6 Marines ($5m_vs_6m$), 3 Stalkers and 5 Zealots against 3 Stalkers and 6 Zealots ($3s5z_vs_3s6z$) are also chosen for comparison. The description of these units is provided in Table I.

TABLE I
DESCRIPTION OF STARCRRAFT II UNITS.

Unit	Race	Type
Marine	Terran	Long-range
Stalker	Protoss	Long-range
Zealot	Protoss	Short-range

These maps are provided by SMAC and could test the performance of UNMAS on StarCraft II micro-management scenario. VDN [18], QMIX [5], and QTRAN [23] are commonly used for comparison in CTDE methods and have already been implemented in SMAC. The code of ASN [26] is also open-source. Besides, the detailed implementation of networks and hyper parameters are provided in Appendix Table B.1.

B. Main Results

We apply these methods on the maps, and choose the test winning rate as the comparison metric. Each experiment is repeated three times for average results, and the learning curves are shown in Fig. 4.

1) *Symmetric Scenarios*: The maps with symmetric setting include: $3m$, $8m$, $2s3z$ and $3s5z$.

a) *Homogeneous Scenarios*: $3m$ and $8m$. In these scenarios, there is only one type of unit: Marine. Therefore, the individual action-value network only needs to represent the policy of one type of unit. As shown in Fig. 4(a) and Fig. 4(d), all of the methods quickly achieve a winning rate close to 100%. The curve of UNMAS starts rising slowly, but still achieves a higher winning rate faster than other methods.

b) *Heterogeneous Scenarios*: $2s3z$ and $3s5z$. In these scenarios, there are two types of units: Zealot and Stalker, so the individual action-value network needs to represent the policies of different types of units. Different types of units have different roles in combat, so heterogeneous scenarios are more difficult than homogeneous scenarios. As shown in Fig. 4(b) and Fig. 4(e), UNMAS achieves the highest winning rate on these maps, especially $3s5z$. The $3s5z$ map is more difficult than $2s3z$, because the increase in the number of units makes the multi-agent system harder to control. In the $3s5z$ experiment, VDN and ASN can not perform well, and QTRAN fails to win. Only UNMAS and QMIX can win combats stably.

2) *Asymmetric Scenarios*: The maps with asymmetric setting include: $5m_vs_6m$ and $3s5z_vs_3s6z$. It means that the number of enemies is greater than that of own agents.

a) *Homogeneous Scenarios*: $5m_vs_6m$. In this scenario, 5 Marines controlled by the algorithms have to combat against 6 enemy Marines. Even if there is only one difference in the number, the control difficulty in order to win

TABLE II
THE TEST WINNING RATES OF VDN, QMIX, QTRAN, ASN, UNMAS, AND OTHER STATE-OF-THE-ART METHODS.

Map	VDN†	QMIX†	QTRAN†	ASN†	RODE†	QPD‡	MADDPG‡	Qatten‡	QPLEX‡	UNMAS
3m	98	99	99	98	99	92	98	-	-	99
8m	97	98	97	98	98	93	98	-	-	97
2s3z	92	98	81	99	99	99	94	96	97	96
3s5z	63	95	2	95	92	91	72	94	92	98
5m_vs_6m	73	72	55	72	75	-	-	74	-	82
3s5z_vs_3s6z	1	1	0	5	1	10	-	16	-	28

† Obtained by the experiments that we conduct.

‡ Obtained by the results provided in literature.

increases considerably. Therefore, the policies of agents are required to be more elaborate in order to win. A more elaborate policy means that the agents make fewer mistakes. As shown in Fig. 4(c), no method achieves 100% winning rate. Among the tested methods, UNMAS achieves the highest winning rate, which means that the multi-agent system using UNMAS is more possible to achieve cooperation and choose correct actions.

- b) *Heterogeneous Scenarios: 3s5z_vs_3s6z*. It is an asymmetric and heterogeneous scenario, the most difficult of all. With an extra Zealot, the enemy is better able to block our Zealots and attack our important *damage dealer* Stalkers. Therefore, the multi-agent system not only needs to ensure the stability of its policy but also explores a better policy to win. As shown in Fig. 4(f), UNMAS achieves the highest winning rate.

Throughout the plot in Fig. 4, it is shown that UNMAS achieves uniform convergence on all experimental maps, and is especially superior on the most difficult scenario 3s5z_vs_3s6z. The final winning rates of different methods are listed in Table II. Compared with QMIX and other methods, we provide a more flexible network structure to represent the joint action-value function, so it is more suitable for training on unshaped scenarios. As for the design of the individual action-value network, ASN uses the GRU layer in unit-oriented stream, while we remove it. Since we are considering a Dec-POMDP especially with limited sight range, a unit wandering close to an agent’s sight range will cause it to receive the unit’s observations intermittently. Therefore, it may lead to a wrong computation of the hidden state for GRU layer and have a negative effect on the strategy of agent.

Besides, for other state-of-the-art MARL methods, due to the limits of reproduction, we only compare with the results provided in their literature, and show them in Table II. MADDPG [35] provides a centralized critic for the whole system and a group of decentralized actors for each agent. Qatten [38] designs a multi-head attention network to approximate joint action-value function. RODE [37] allows agents with a similar role to share similar behaviors. QPD [22] employs integrated gradients method to factorize the joint action-value of multi-agent system into individual action-values of agents. QPLEX [21] takes a duplex dueling network to factorize the joint action-value.

The results shown in Table II are the average winning

rates after 2 million timesteps. In the article of RODE, the experiment on most maps is the result of training 5 million timesteps, we run RODE on each map and test its winning rate at 2 million timesteps. Since MADDPG does not experiment on StarCraft II micro-management scenarios, we compare with the results provided by [43], in which the authors implement several MARL methods. Although these methods achieve similar performance to us in symmetric maps, UNMAS still has obvious advantage in the most difficult scenario 3s5z_vs_3s6z. These results indicate that UNMAS is still competitive among those state-of-the-art methods.

C. Ablation Results

In order to investigate the effect of (i) the weight $k_{i,t}$, (ii) the value v_t in the self-weighting mixing network, and (iii) the concat operation in the individual action-value network, we conduct ablation experiments on three maps: 3s5z, 5m_vs_6m and 3s5z_vs_3s6z.

- 1) *The effect of weight $k_{i,t}$* . We remove the weight $k_{i,t}$ in the self-weighting mixing network and replace the joint action-value with the following equation:

$$\mathbf{Q}(\mathbf{o}_t, \mathbf{u}_t; \boldsymbol{\theta}_{joint}) = \frac{1}{n} \sum_{i=1}^n q'_{i,t} + v_t, \quad (14)$$

which denotes that the weight $k_{i,t}$ of each agent is set to 1 in any case. We refer to this method as UNMAS-ADD. Fig. 5 shows that the winning rate of self-weighting mixing network decreases under the condition of fixed weights $k_{i,t}$. One possible explanation is that because the weights are fixed, the self-weighting mixing network cannot correctly estimate the true joint action-value based on the contribution of each agent to the joint action-value. In order to illustrate the effect of $k_{i,t}$ specifically, we provide two screenshots of the combat on the 3s5z_vs_3s6z map and mark the weight ratio $k_{i,t} / \sum_j k_{j,t}$ of each agent as shown in Fig. 6. The screenshots indicate that agents have different contribution to the joint action-value. When there is no close combat between two sides as shown in Fig. 6(a), the long-range unit Stalker is more important. Its average weight of 0.186 is larger than Zealot’s 0.073. Once they are fighting hand-to-hand as shown in Fig. 6(b), the short-range unit Zealot becomes vital. Its average weight of 0.120 is larger than Zealot’s

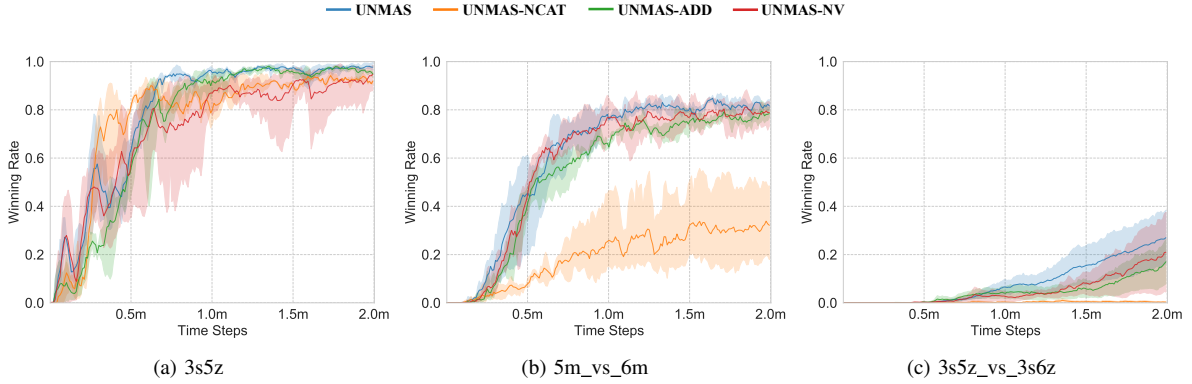


Fig. 5. Winning rates of ablation experiments. The ablation results demonstrate the effect of three elements.

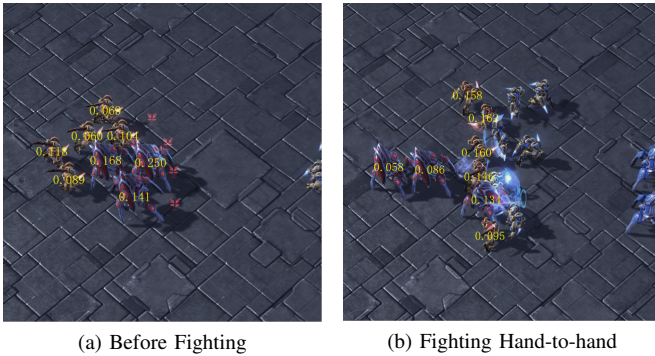


Fig. 6. Two screenshots of the combat on the 3s5z_vs_3s6z map. The weight ratio $k_{i,t}/\sum_j k_{j,t}$ is marked on the position of each agent i .

0.092. Besides, within the same type of units, agents get larger weights if there are more enemies around them. Taking the Zealots in Fig. 6(b) as an example, the unit surrounded by enemies has larger weight (0.162) than the one that is alone (0.095).

- 2) *The effect of value v_t .* In the second experiment, we remove the value v_t in the joint action-value. Therefore, the joint action-value can be calculated by the following formula:

$$\mathbf{Q}(\mathbf{o}_t, \mathbf{u}_t; \boldsymbol{\theta}_{joint}) = \frac{1}{n} \sum_{i=1}^n q'_{i,t} \cdot k_{i,t}. \quad (15)$$

We refer to this method as UNMAS-NV. Fig. 5 shows that the introduction of v_t increases the efficiency of approximation, which leads to higher winning rate. It indicates that the lack of a bias term makes it more difficult for the self-weighting mixing network to approximate the joint action-value function.

- 3) *The effect of the concat operation.* In the third experiment, we remove the concat operation in the individual action-value network. It means that the Q values used to evaluate unit-oriented actions only depend on the observations of target units rather than the observation history. We refer to this method as UNMAS-NCAT. Fig. 5 shows that the concat operation is critical to the performance of the agent. The winning rate of UNMAS-NCAT in ablation experiments is much lower than UNMAS.

According to the ablation experiments, we figure out the importance of the weight $k_{i,t}$, value v_t , and the concat operation. The reason that UNMAS achieves the highest winning rate becomes clear. In the factorization of joint action-value, UNMAS provides the weights for each alive agent and ignores the agents that are killed by enemies, which leads to a more proper factorization. The bias term also helps approximate the joint action-value function. In estimating the action-value of an agent, UNMAS evaluates the actions in two different subsets respectively with the help of the concat operation, making the evaluation more accurate.

D. Strategy Analysis

To understand what UNMAS has learned to achieve the performance, we analyze the strategies of agents using UNMAS according to the combat replay in this subsection.

- 1) *Homogeneous Scenarios: 3m, 8m and 5m_vs_6m.* In these scenarios, the combat is conducted between Marines. Next is the analysis of the strategies executed by the multi-agent system.

- a) *More to Fight Less:* The multi-agent system adjusts the formation to form a situation where more agents attack fewer enemies in the combat. It is a basic strategy for micro-management scenario. Taking an example, as shown in Fig. 7(a), agents try to concentrate on attacking an enemy Marine. The red line indicates that the agent is attacking. By adopting this strategy, the multi-agent system is able to eliminate the enemies as quickly as possible to reduce the damage caused by them.
- b) *Damage Sharing:* The agents with higher health share the damage for agents with lower health. Since only alive agents can cause damage to enemies, it is important to ensure the survival of agents who is in danger. As shown in Fig. 7(b), when the health of the agent is low, it retreats a distance from the enemies to allow the other agents to share the damage. The white line indicates that the agent is moving. At the same time, the agents around it step forward in the enemy's direction to ensure successful damage sharing. By adopting this strategy, the agents are able to survive longer to maximize the damage they cause.

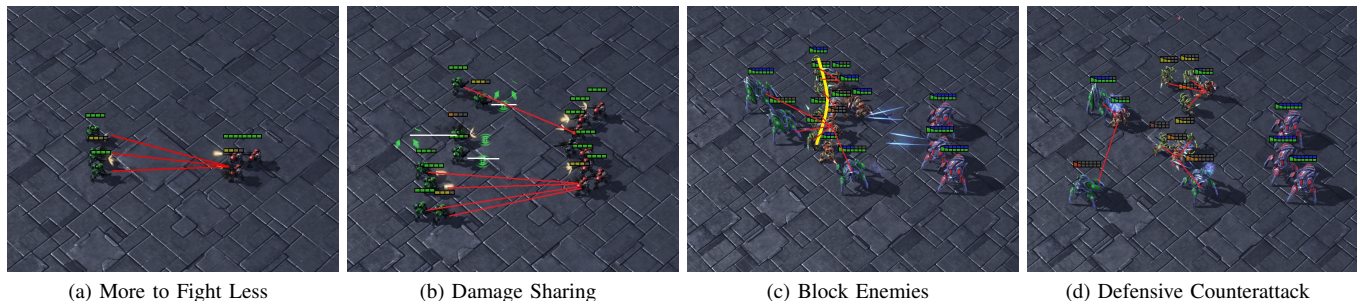


Fig. 7. Strategies learned by the agents using UNMAS. There are two basic strategies *more to fight less* and *damage sharing*, and two advanced strategies *block enemies* and *defensive counterattack* to address more difficult scenarios.

In the homogeneous scenarios, the multi-agent system performs the above strategies to defeat the enemies. Although other methods like QMIX can also learn these strategies, the agents using UNMAS make fewer mistakes.

2) *Heterogeneous Scenarios*: $2s3z$, $3s5z$ and the most difficult map $3s5z_vs_3s6z$. In these scenarios, the combat is conducted between two types of units: Zealot and Stalker. The former is short-range and the latter is long-range. For scenarios where units have different roles, the multi-agent system not only needs to execute the strategies analyzed above but also needs new strategies to accommodate the change. Next is the analysis of the strategies executed by the multi-agent system.

- a) *Block Enemies*: The Zealots try to block the enemy Zealots to protect the Stalkers. Since the Stalker is a long-range unit, it is able to cause damage to the enemies without being attacked. As shown in Fig. 7(c), the multi-agent system using UNMAS learns the strategy of placing the Zealots between Stalkers and enemies to ensure the safety of Stalkers. The yellow line is the defensive line formed by Zealots. Through this strategy, the multi-agent system can maximize its damage as much as possible.
- b) *Defensive Counterattack*: Agents focus on eliminating enemy Zealots who break through the defensive line as a defense, and then continue their previous attack as a counterattack. In the actual combat, the defensive line may not completely separate the enemies from Stalkers. As shown in Fig. 7(d), the multi-agent system using UNMAS learns to attack the enemy who breaks through the defensive line. They also perform a *more to fight less* strategy to eliminate enemies as quickly as possible. In addition, sometimes one Stalker chooses to attack the enemies at the edge of the battlefield to disperse pressure from other allies.

It should be noted that, in the heterogeneous scenarios, the multi-agent system not only learns the strategies mentioned above, but also learns *more to fight less* and *damage sharing* as in homogeneous scenarios. Other methods all fail in $3s5z_vs_3s6z$ because they just perform the *more to fight less* strategy as in homogeneous scenarios, while UNMAS achieves 28% winning rate because of the above strategies.

V. CONCLUSION

This paper proposes UNMAS, a novel multi-agent reinforcement learning method that is more adaptable to the unshaped scenario, in which the number of agents and the size of action set change over time. UNMAS factorizes the joint action-value by mapping Q values nonlinearly and calculating its weights in the joint action-value for each agent. We theoretically analyze that the value factorization of UNMAS meets the IGM condition. In order to adapt to the change in the size of action set, the individual action-value network uses two network streams to evaluate the actions in *environment-oriented* subset and *unit-oriented* subset.

We compare UNMAS with VDN, QMIX, QTRAN, and ASN in StarCraft II micro-management scenario. Our results show that UNMAS achieves state-of-the-art performance among tested algorithms and also learns effective strategies in the most difficult scenario $3s5z_vs_3s6z$ that other algorithms fail in.

ACKNOWLEDGMENT

The authors would like to thank Professor Hongbin Sun of Xi'an Jiaotong University for his suggestions.

APPENDIX A ENVIRONMENTAL SETTINGS

We use SMAC as the experimental environment. Alive agents obtain their local observations from environment and execute their actions. The observation provided by SMAC consists of the information about agent self, other agents, and the enemies within sight range. In detail, the information contains the following elements:

- 1) *Distance*: the distance between agent and other units;
- 2) *Relative coordinates*: the relative coordinates of x and y between agent and other units;
- 3) *Health*: the health percentage of agent and other units;
- 4) *Shield*: the shield percentage of agent and other units;
- 5) *Unit type*: the one-hot coding for the unit type of agent and other units;
- 6) *Last action*: the action executed by agent at last timestep;
- 7) *Agent Index*: the index used to distinguish agents.

Similarly, the global state also provides the above information. However, the difference is that the relative information takes the center point of the map as the reference point. The

information included in the state contains all the alive agents on the map instead of those only within the *sight range*.

APPENDIX B PARAMETER SETTINGS

In the experiments, all methods adopt the *same* hyper parameters, which are shown in Table B.1, and are the default values in PyMARL.

TABLE B.1
HYPER-PARAMETERS OF EXPERIMENTAL METHODS, INCLUDING ASN,
VDN, QMIX, QTRAN, RODE, AND UNMAS.

Setting	Name	Value
Training Settings	Size of Replay buffer D	5000 episodes
	Batch size b	32 episodes
	Testing interval	10000 timesteps
	Target update interval	200 episodes
	Maximum timesteps	2 million timesteps
	Exploration rate ϵ	1.0 to 0.05
	Discount factor γ	0.99
Network Settings	Self-weighting mixing network unit	32
	Hyper network unit	64
	GRU layer unit	64
	Optimizer	RMSProp
	RMSProp α_R	0.99
	RMSProp ϵ_R	0.00001
	Learning rate α	0.0005

The hypernetwork of UNMAS, which is used to calculate the weights and biases of self-weighting mixing network, adopts the same settings as QMIX. v_t , the final element of self-weighting mixing network, is the output of a network with two layers and one ReLU activation. Other parameters related to the networks are also shown in Table B.1.

REFERENCES

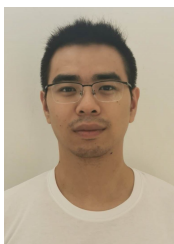
- [1] Y. Cao, W. Yu, W. Ren, and G. Chen, "An overview of recent progress in the study of distributed multi-agent coordination," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 427–438, 2013.
- [2] W. Jiang, G. Feng, S. Qin, T. S. P. Yum, and G. Cao, "Multi-agent reinforcement learning for efficient content caching in mobile D2D networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 3, pp. 1610–1622, 2019.
- [3] Z. Tang, D. Zhao, Y. Zhu, and P. Guo, "Reinforcement learning for build-order production in StarCraft II," in *International Conference on Information Science and Technology*, 2018, pp. 153–158.
- [4] K. Shao, Y. Zhu, and D. Zhao, "StarCraft micromanagement with reinforcement learning and curriculum transfer learning," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 3, no. 1, pp. 73–84, 2019.
- [5] 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 *International Conference on Machine Learning*, 2018, pp. 4295–4304.
- [6] Z. Tang, K. Shao, Y. Zhu, D. Li, D. Zhao, and T. Huang, "A review of computational intelligence for StarCraft AI," in *2018 IEEE Symposium Series on Computational Intelligence*, 2018, pp. 1167–1173.
- [7] M. Li, Z. Qin, Y. Jiao, Y. Yang, J. Wang, C. Wang, G. Wu, and J. Ye, "Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning," in *The World Wide Web Conference*, 2019, pp. 983–994.
- [8] Y. Zhu, D. Zhao, and Z. Zhong, "Adaptive optimal control of heterogeneous CACC system with uncertain dynamics," *IEEE Transactions on Control Systems Technology*, vol. 27, no. 4, pp. 1772–1779, 2019.
- [9] L. Liang, H. Ye, and G. Y. Li, "Spectrum sharing in vehicular networks based on multi-agent reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 10, pp. 2282–2292, 2019.
- [10] H. Li, Q. Zhang, and D. Zhao, "Deep reinforcement learning-based automatic exploration for navigation in unknown environment," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 6, pp. 2064–2076, 2020.
- [11] G. Palmer, K. Tuyls, D. Bloembergen, and R. Savani, "Lenient multi-agent deep reinforcement learning," in *International Foundation for Autonomous Agents and Multiagent Systems*, 2018, pp. 443–451.
- [12] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PLoS One*, vol. 12, no. 4, pp. 1–15, 2017.
- [13] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *International Conference on Autonomous Agents and Multiagent Systems*, 2017, pp. 66–83.
- [14] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018.
- [15] K. Shao, Y. Zhu, and D. Zhao, "Cooperative reinforcement learning for multiple units combat in StarCraft," in *2017 IEEE Symposium Series on Computational Intelligence*, 2017, pp. 1–6.
- [16] Z. Zhang, D. Zhao, J. Gao, D. Wang, and Y. Dai, "FMRQ—A multiagent reinforcement learning algorithm for fully cooperative tasks," *IEEE Transactions on Cybernetics*, vol. 47, no. 6, pp. 1367–1379, 2017.
- [17] Q. Zhang, D. Zhao, and F. L. Lewis, "Model-free reinforcement learning for fully cooperative multi-agent graphical games," in *International Joint Conference on Neural Networks*, 2018, pp. 1–6.
- [18] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. F. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls *et al.*, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *International Foundation for Autonomous Agents and Multiagent Systems*, 2018, pp. 2085–2087.
- [19] T. Wang, H. Dong, V. Lesser, and C. Zhang, "ROMA: Multi-agent reinforcement learning with emergent roles," in *Proceedings of the 37th International Conference on Machine Learning*, 2020.
- [20] Y. Yang, Y. Wen, L. Chen, J. Wang, K. Shao, D. Mguni, and W. Zhang, "Multi-agent determinantal Q-learning," *arXiv preprint arXiv:2006.01482*, 2020.
- [21] J. Wang, Z. Ren, T. Liu, Y. Yu, and C. Zhang, "QPLEX: Duplex dueling multi-agent Q-learning," *arXiv preprint arXiv:2008.01062*, 2020.
- [22] Y. Yang, J. Hao, G. Chen, H. Tang, Y. Chen, Y. Hu, C. Fan, and Z. Wei, "Q-value path decomposition for deep multiagent reinforcement learning," *arXiv preprint arXiv:2002.03950*, 2020.
- [23] 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 *International Conference on Machine Learning*, 2019, pp. 5887–5896.
- [24] Z. Sui, Z. Pu, J. Yi, and S. Wu, "Formation control with collision avoidance through deep reinforcement learning using model-guided demonstration," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.
- [25] W. Wang, T. Yang, Y. Liu, J. Hao, X. Hao, Y. Hu, Y. Chen, C. Fan, and Y. Gao, "From few to more: Large-scale dynamic multiagent curriculum learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 7293–7300.
- [26] —, "Action semantics network: Considering the effects of actions in multiagent systems," in *International Conference on Learning Representations*, 2020.
- [27] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the International Conference on Machine Learning*, 1993, pp. 330–337.
- [28] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [29] M. Zhou, Y. Chen, Y. Wen, Y. Yang, Y. Su, W. Zhang, D. Zhang, and J. Wang, "Factorized Q-learning for large-scale multi-agent systems," in *Proceedings of the First International Conference on Distributed Artificial Intelligence*, 2019, pp. 1–7.
- [30] Y. Zhu and D. Zhao, "Online minimax Q network learning for two-player zero-sum markov games," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.
- [31] L. Han, P. Sun, Y. Du, J. Xiong, Q. Wang, X. Sun, H. Liu, and T. Zhang, "Grid-wise control for multi-agent reinforcement learning in video game ai," in *International Conference on Machine Learning*, 2019, pp. 2576–2585.

- [32] P. Peng, Y. Wen, Y. Yang, Q. Yuan, Z. Tang, H. Long, and J. Wang, "Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play StarCraft combat games," *arXiv preprint arXiv:1703.10069*, 2017.
- [33] J. Qin, M. Li, Y. Shi, Q. Ma, and W. X. Zheng, "Optimal synchronization control of multiagent systems with input saturation via off-policy reinforcement learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 1, pp. 85–96, 2019.
- [34] C. Sun, W. Liu, and L. Dong, "Reinforcement learning with task decomposition for cooperative multiagent systems," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.
- [35] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 6382–6393.
- [36] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations*, 2016.
- [37] T. Wang, T. Gupta, A. Mahajan, B. Peng, S. Whiteson, and C. Zhang, "RODE: Learning roles to decompose multi-agent tasks," in *International Conference on Learning Representations*, 2021.
- [38] Y. Yang, J. Hao, B. Liao, K. Shao, G. Chen, W. Liu, and H. Tang, "Qatten: A general framework for cooperative multiagent reinforcement learning," *arXiv preprint arXiv:2002.03939*, 2020.
- [39] F. A. Oliehoek, C. Amato *et al.*, *A Concise Introduction to Decentralized POMDPs*. Springer, 2016, vol. 1.
- [40] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units," in *International Conference on Learning Representations*, 2016.
- [41] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," in *Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014.
- [42] M. Samvelyan, T. Rashid, C. Schroeder de Witt, G. Farquhar, N. Nardelli, T. G. Rudner, C.-M. Hung, P. H. Torr, J. Foerster, and S. Whiteson, "The StarCraft multi-agent challenge," in *Proceedings of the 18th International Conference on Autonomous Agents and Multi-Agent Systems*, 2019, pp. 2186–2188.
- [43] G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht, "Comparative evaluation of multi-agent deep reinforcement learning algorithms," *arXiv preprint arXiv:2006.07869*, 2020.



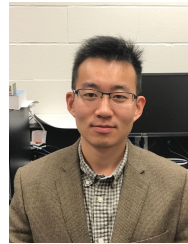
Jiajun Chai received the B.S degree from the Faculty of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an, China, in 2020. He is currently pursuing the Ph.D. degree with the State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing, China.

His current research interests include multi-agent reinforcement learning, deep learning, and game AI.



Weifan Li received the M.E degree in the Mechanical engineering and automation, Fuzhou University, Fuzhou, China, in 2018. He is currently pursuing the Ph.D. degree in control theory and control engineering at the State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing, China.

His current research interest is deep reinforcement learning.



omatic driving, and game intelligence.

Yuanheng Zhu (M'15) received the B.S. degree in automation from Nanjing University, Nanjing, China, in 2010, and the Ph.D. degree in control theory and control engineering from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2015. He is currently an Associate Professor with the State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences. His research interests include optimal control, adaptive dynamic programming, reinforcement learning, au-



Dongbin Zhao (M'06-SM'10-F'20) received the B.S., M.S., Ph.D. degrees from Harbin Institute of Technology, Harbin, China, in 1994, 1996, and 2000 respectively. He is now a professor with Institute of Automation, Chinese Academy of Sciences, and also with the University of Chinese Academy of Sciences, China. He has published 6 books, and over 100 international journal papers. His current research interests are in the area of deep reinforcement learning, computational intelligence, autonomous driving, game artificial intelligence, robotics, etc.

Dr. Zhao serves as the Associate Editor of IEEE Transactions on Neural Networks and Learning Systems, IEEE Transactions on Cybernetics, IEEE Transactions on Artificial Intelligence, etc. He is the chair of Distinguished Lecture Program of IEEE Computational Intelligence Society (CIS). He is involved in organizing many international conferences. He is an IEEE Fellow.



Zhe Ma has received his Ph.D. degree. He is currently a researcher at X-Lab in the Second Academy of CASIC.

His current research interests include artificial intelligence and SoS.



Kewu Sun has received her B.S. and M.S. degree. She is currently a senior engineer at X-Lab in the Second Academy of CASIC.

Her current research interests include multi-agent reinforcement learning and SoS.



Jishi Yu Ding received the B.S. degree from Beijing Jiaotong University in 2015 and Ph.D. degree from Tsinghua University in 2020. He is currently an engineer at X-Lab in The Second Academy of CASIC.

His current research interest is multi-agent reinforcement learning.