# Structured Cooperative Reinforcement Learning with Time-varying Composite Action Space

Wenhao Li, Xiangfeng Wang, Bo Jin, Dijun Luo, and Hongyuan Zha

**Abstract**—In recent years, reinforcement learning has achieved excellent results in low-dimensional static action spaces such as games and simple robotics. However, the action space is usually composite, composed of multiple sub-action with different functions, and time-varying for practical tasks. The existing sub-actions might be temporarily invalid due to the external environment, while unseen sub-actions can be added to the current system. To solve the robustness and transferability problems in time-varying composite action spaces, we propose a structured cooperative reinforcement learning algorithm based on the centralized critic and decentralized actor framework, called SCORE. We model the single-agent problem with composite action space as a fully cooperative partially observable stochastic game and further employ a graph attention network to capture the dependencies between heterogeneous sub-actions. To promote tighter cooperation between the decomposed heterogeneous agents, SCORE introduces a hierarchical variational autoencoder, which maps the heterogeneous sub-action space into a common latent action space. We also incorporate an implicit credit assignment structure into the SCORE to overcome the multi-agent credit assignment problem in the fully cooperative partially observable stochastic game. Performance experiments on the proof-of-concept task and precision agriculture task show that SCORE has significant advantages in robustness and transferability for time-varying composite action space.

**Index Terms**—Cooperative Multi-Agent Reinforcement Learning, Composite Action Space, Time-varying Action Space.

## 1 Introduction

In recent years, deep reinforcement learning (DRL) has been successfully applied to many fields like computer games [1, 2], simple robotic control [3, 4], autonomous vehicles [5], image processing [6, 7], etc. In general, the action spaces of these applications are static and homogeneous, while very few are heterogeneous but with a simple structure. For example, in Atari games [8], the action spaces mostly only include moving up/down/left/right, plus firing and jumping; in simple robot control tasks, all robot joints are homogeneous and controlled by a fixed-length parameter vector; in image processing tasks, actions are generally defined as pixel-wise value changing, homogeneous for all pixels in the image. Besides, the action spaces of the above tasks are all static, predefined, and will not change with the execution of the task.

However, for many realistic tasks, the action spaces are usually *time-varying* and *composite*. The *time-varying* means the action space will change over time, and the *composite* means there are multiple heterogeneous sub-action spaces with different function, as shown in Figure 1. For instance, in the precision agriculture problem, on the one hand, the agent (control center) needs to determine the composite actions of the greenhouse, including temperature, humidity, light intensity, carbon dioxide concentration, etc. Different physical controllers control these heterogeneous sub-actions. As a result, the dimensions of these sub-actions controllable parameters and the types of sub-action values (continuous or discrete values) may be different. On the other hand, some controllers will be temporarily invalid in the actual deployment due to poor communication networks, device aging, abnormal weather, or sudden natural disasters. Also, with the iteratively updating of the intelligent greenhouse, new controllers will gradually be added to the precision agriculture task. The increasing or decreasing of the controllers are common in real-world scenes, which leads to the time-varying and composite action space in the corresponding RL problem. In addition to precision agriculture scenarios, such action spaces are also widely arisen in complex robot control, industrial production, and drone planning. The various modules of modular robots, the robot arms in the automated assembly line, and the drones with different functions exhibit prominent composable and time-varying characteristics of the action space.

To the best of our knowledge, few works can handle time-varying and composite action space simultaneously. Some early works [9, 10, 11] modeled the composite action space problem as a general high-dimensional action selection problem. However, they usually ignore the heterogeneity of the action space. Zhang et al. [12] proposed an LSTM-based autoregressive structure to learn in the composite action space, in which the LSTM structure requires predefined dependencies between heterogeneous actions. Further, the parameterized action space problem can be regarded as a particular case of the composite action space problem, while the number of sub-action spaces equals one. A series

- W. Li, X. Wang and B. Jin are with the School of Computer Science and Technology and the MOE Key Laboratory for Advanced Theory and Application in Statistics and Data Science, East China Normal University, Shanghai 200062, China. Email: 52194501026@stu.ecnu.edu.cn, {xfwang, bjin}@cs.ecnu.edu.cn.
- D. Luo is with Tencent AI Lab, Tencent Inc, Shenzhen 518000, China. Email: dijunluo@tencent.com.
- H. Zha is with School of Data Science and AIRS, The Chinese University of Hong Kong, Shenzhen 518172, China. Email: zhahy@cuhk.edu.cn.

*Manuscript received November 16, 2020; revised May 03, 2021; accepted July 28, 2021. Date of publication xxxxx; date of current version xxxx.*
*(Corresponding authors: Xiangfeng Wang and Bo Jin)*
*Recommended for acceptance by xxxxx.*
*Digital Object Identifier no. xxxxx.*

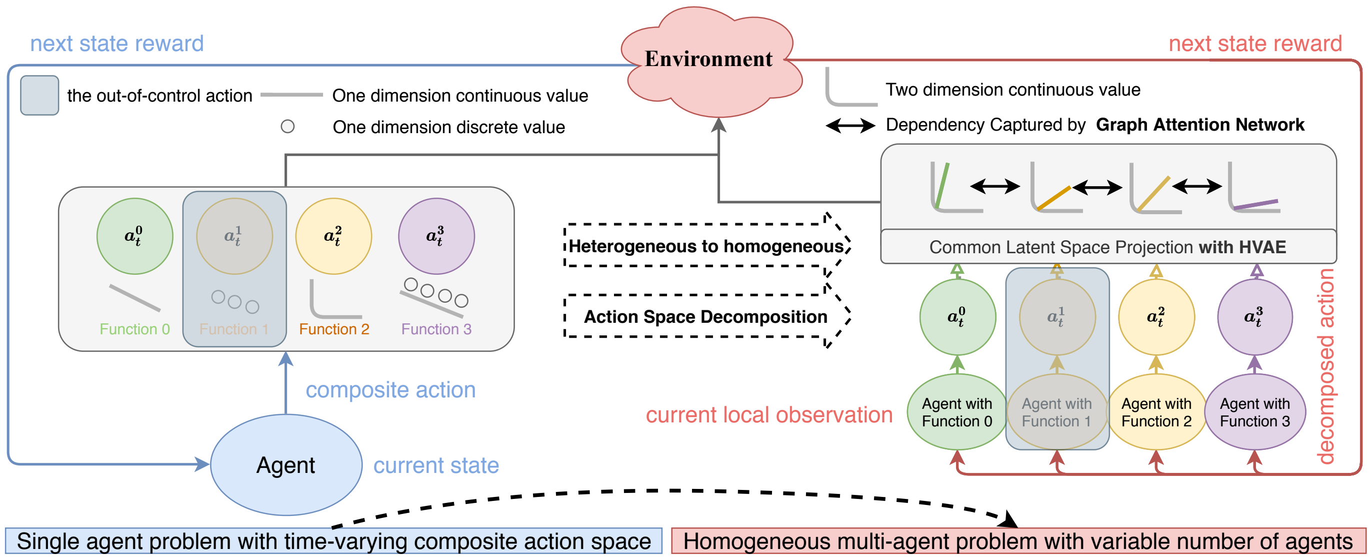

next state reward
Environment
next state reward
the out-of-control action
One dimension continuous value
One dimension discrete value
Two dimension continuous value
Dependency Captured by Graph Attention Network
$a_t^0$
$a_t^1$
$a_t^2$
$a_t^3$
Function 0
Function 1
Function 2
Function 3
Heterogeneous to homogeneous
Action Space Decomposition
Common Latent Space Projection with HVAE
Agent with Function 0
Agent with Function 1
Agent with Function 2
Agent with Function 3
decomposed action
composite action
current local observation
Agent
current state
Single agent problem with time-varying composite action space
Homogeneous multi-agent problem with variable number of agents

Fig. 1. The conceptual diagram of time-varying composite action space and structured cooperative reinforcement learning algorithm (SCORE). **Left:** The single-agent problem with time-varying composite sub-action space. Sub-action spaces with different functions are generally heterogeneous, with different semantics and data structures. For example, the four functional sub-action spaces in the figure are one-dimensional continuous space, one-dimensional discrete space, two-dimensional continuous space, and a mixed space of one-dimensional continuous space and one-dimensional discrete space. At each time step, the agent needs to determine all sub-actions based on the policy jointly. **Right:** The homogeneous multi-agent problem with the variable number of agents after decomposition. Different sub-action spaces are considered independently and regarded as agents. All agents have the same local observations and shared team rewards. This paper proposes the SCORE algorithm based on graph attention network (GAT) and hierarchical variational autoencoder (HVAE) to solve this multi-agent problem. After decomposing the composite actions, HVAE maps the sub-action spaces with different functions to the common latent space into a more straightforward homogeneity problem. At the same time, GAT is used to deal with the variable number of agents and to capture the dependencies between sub-action spaces with different functions.

of works for parameterized action space problems are discussed in recent years [13, 14, 15, 16]. However, all the works mentioned above only consider the static action spaces. There also are some recent works focus on transferring reinforcement learning to unseen actions [17, 18, 19, 20, 21, 22]. In these works, the number of sub-action spaces is assumed to be one. As a result, transfer learning can be achieved by directly measuring the similarity between actions or capturing the structure of the action spaces. However, similarity measuring and structure capturing become extremely difficult for tasks with composite action spaces, such as precision agriculture. Specifically, it is impractical to apply the above ideas to the composite action space directly. Naturally, sub-action spaces with different functions in the composite action space have mutual influences, which bring new challenges to transfer to unseen actions.

In this paper, decoupling is introduced as the core idea to solve the time-varying composite action space problem. We model the single-agent reinforcement learning problem with time-varying composite action space as a fully cooperative partially observable stochastic game (POSG) with variable number of agents and propose the **S**tructured **CO**operation **RE**inforcement Learning (SCORE) algorithm. The sub-action space of each function is regarded as an independent agent[1]. SCORE adopts the centralized critic and decentralized actor (CCDA) framework [23] and introduces the graph neural network (GNN) as the central critic. However, the environment nonstationarity problem is introduced when we consider the sub-action space of each function independently. More critically, this will ignore the dependency between different sub-action spaces. For example, in the precision agriculture task, when the temperature rises, enzyme activity also increases; if the light intensity is increased simultaneously, the intensity of photosynthesis will get increased and promote the growth of crops (see Figure 2 for intuitive explanation). If we can explicitly model these dependencies, the multi-agent system can explore more efficiently, which will improve the final algorithm performance [24]. Based on this idea, SCORE first introduced the hierarchical variational autoencoder (HVAE)) [25] to map the sub-action space of each function to

1. In this paper, we will no longer distinguish between the two concepts of sub-action space and agent.

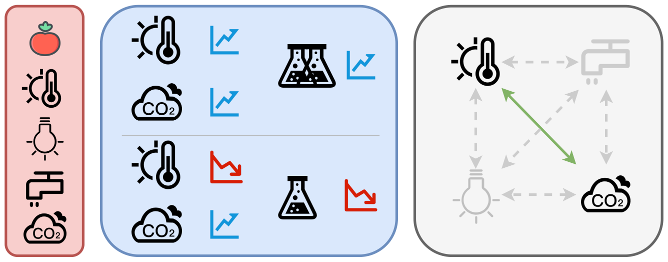


Fig. 2. We choose the tomato planting task to show the dependence between different sub-action spaces in the precision agriculture scene. **Left:** The tomato planting task contains four sub-action spaces with different functions, from top to bottom: temperature, light, irrigation, and carbon dioxide. **Middle:** As the carbon dioxide concentration and temperature increase, the activity of crop enzymes will increase, which will increase crop yields. If the temperature drops while the carbon dioxide concentration increases, it will reduce the enzyme activity and reduce the crop yield. **Right:** To increase crop yield, the correlation constraint between the carbon dioxide sub-action space and the illumination sub-action space should be positive.

a common latent action space; the attention mechanism [26] is then introduced to the centralized critics based on GNN to form a graph attention network (GAT) [27] based CCDA framework. At this time, SCORE can model the correlation between the sub-actions of each function projected to the common latent space by using the attention weight. Finally, modeling the time-varying composite action space problem as a fully cooperative POSG will also introduce the multi-agent credit assignment problem. For this reason, SCORE additionally introduces an implicit credit assignment structure in the CCDA framework. The conceptual diagram is shown in Figure 1, and the main contributions are summarized as follows:

- We reformulate the single-agent reinforcement learning problem with time-varying composite action space as a fully cooperative partially observable stochastic game with the variable number of agents and propose a novel graph-attention-network-based MARL algorithm, SCORE.
- We incorporate the hierarchical variational autoencoder architecture into the MARL algorithm to better capture the correlation between actions and improve transferability to unseen sub-action space.
- We organically combine the implicit credit assignment structure with the SCORE algorithm, which effectively solves the multi-agent credit assignment problem in fully cooperative POSG.
- Both the proof-of-concept task and experiments on the agricultural simulator show that the proposed SCORE algorithm can exceed the performance of the baselines both in robustness and transferability in the time-varying composite action space.

This paper is organized as follows. We introduce the related works of composite action space, action relationships, and unseen actions transfer learning in Section 2. We provide the background material on cooperative POSG and graph attention network in Section 3. We present our structured cooperative reinforcement learning algorithms (SCORE) in Section 4. Extensive experiments are presented in Section 5, and we conclude this paper in Section 6.

## 2 Related Works

### 2.1 AI for Agriculture

The United Nations estimates that the world needs to feed two billion additional people by 2050 [28], which presents a significant challenge to the global agriculture capacity. To improve agriculture is to address both today's and tomorrow's problems. AI has lots of potential in this domain, and we follow the agriculture supply chain to introduce these works, similar as Shi et al. [29]. The progress and commercialization of UAVs, satellite images, and IoT promoting the so-called "precision agriculture". With them, it is possible to address the various aspects of crop management, such as crop planning, maintenance, yield prediction, mitigating crop disease, and agricultural information gathering Shi et al. [29]. Lücken and Brunelli [30] uses multi-objective evolutionary algorithms to determine the optimal crop to grow based on the soil information. Other works explore selecting the best time to sow based on additional factors such as weather information [31]. Holman et al. [32] propose using the Gaussian Process to estimate crop evapotranspiration information with data from ordinary weather stations for optimizing irrigation plans. You et al. [33] not only use deep learning models to predict the crop yield from satellite images to help farmer plan and the government determine agriculture policy but also design a compact representation to address the scarcity of labeled data. Quinn et al. [34] propose several approaches to address problems related to crop disease, including using Gaussian Process ordinal regression to estimate disease distribution, using a model for survey planning, etc. A mobile app based on Quinn et al. [34] is now available [35]. Some works focus on agriculture information gathering efficiency [33, 36] and price movement prediction in agriculture produce market [37].

Crop planting is the first part of the agricultural supply chain. However, to the best of our knowledge, few works directly control crop planting strategies, so-called "fine-grained precision agriculture". This paper models the fine-grained precision agriculture problem as a single-agent reinforcement learning problem with time-varying composite action space and proposes a robust and transferable reinforcement learning algorithm SCORE. The simulated experiment for the tomato planting task proves that it can effectively promote crop growth.

### 2.2 Composite Action Space

Similar to the composite action space concept, the parameterized action space is a discrete action space with some continuous parameters. One straightforward approach is to discretize the continuous part of the action space directly and turn it into a sizeable discrete set (for example, with the tile coding approach [38]). This trivial method loses the advantages of continuous action space for fine-grained control and often ends up with a vast discrete action space. Another direction is to convert the discrete action selection into a continuous space. Hausknecht and Stone [39] used an actor-network to output a value for each of the discrete actions, concatenated with all continuous parameters, discrete action is chosen to be the one with the maximum output value. Masson et al. [13] focused on learning an action-selection policy given fixed-parameter selection and proposed the framework called Q-PAMDP, which alternately learns the discrete action selection with Q-learning and updated parameter-selection policies with policy search methods. Wei et al. [14] proposed a hierarchical approach for RL in parameterized action space where the parameter policy is conditioned on the discrete action policy and used TRPO and Stochastic Value Gradient [40] to train such an architecture. Xiong et al. [15] proposed the parameterized deep Q-networks (P-DQN) algorithm. P-DQN has one network to select the continuous parameters for all discrete actions. Another network takes the state and the chosen continuous parameters as input and outputs the Q-values for all discrete actions.

There are only a few works focus on the composite action space problem. Fan et al. [16] proposed an actor-critic-based approach H-PPO, which consists of multiple parallel sub-actor networks to decompose the structured action space into simpler action spaces with a shared critic network to guide the training of all sub-actor networks. Zhang et al. [12] proposed an LSTM-based auto-regressive structure to learning in the composite action space. In each timestep, each sub-action is determined by all previously decided sub-actions. However, all the works mentioned above only consider the static action spaces.

## 2.3 Learning Relationships Between Actions

He et al. [41] introduces a novel architecture for reinforcement learning with deep neural networks designed to handle state and action spaces characterized by natural language, as found in text-based games. Termed a deep reinforcement relevance network, the architecture represents action and state spaces with separate embedding vectors, which are combined with an interaction function to approximate the Q-function in reinforcement learning. Wang and Yu [42] explores the multi-action relationship in reinforcement learning and proposes to learn the multi-action relationship by enforcing a regularization term capturing the relationship. They incorporate the regularization term into the least-squared policy-iteration and the temporal-difference methods, resulting in efficiently solvable convex learning objectives. Tennenholtz and Mannor [43] introduces a general framework for learning context-based action representation. Representing actions in a vector space help RL achieve better performance by grouping similar actions and utilizing relationships between different actions. Kim et al. [24] proposes an exploration method that constructs embedding representation of states and actions that do not rely on generative decoding of full observation but extracts predictive signals that can guide exploration based on forwarding prediction in the representation space. These works only consider one sub-action space.

## 2.4 Generalize to Unseen Actions

For a time-varying action space, unseen actions will be added to the previous action space in addition to some sub-actions will get temporarily invalid. Jain et al. [20] proposes a method for reinforcement learning with unseen actions. The actions available during training (known actions) are a subset of all the actions available during evaluation (known and unknown actions). The method can choose unknown actions during evaluation through an embedding space over the actions, which defines a distance between actions. Farquhar et al. [17] uses a curriculum of progressively growing action spaces to accelerate learning. This approach uses off-policy RL to estimate optimal value functions for multiple action spaces simultaneously and efficiently transfers data, value estimation, and state representations from restricted action spaces to the full task. Chen et al. [18] proposes a transfer learning method to learn action embedding for discrete actions in RL from generated trajectories without prior knowledge and then leverage it to transfer policy across tasks with different state spaces and/or different discrete action space. Chandak et al. [21] presents a two-stage algorithm to tackle the problem that the action set whose size changes over time in lifelong learning. This algorithm breaks the problem into two sub-problems that can be solved iteratively: inferring the underlying, unknown structure in the space of actions and optimizing a policy that leverages this structure. Annie et al. [19], Fang et al. [22] aims to solve the problem of learning a model that can perform complex tasks and transfers to previously unseen objects. Annie et al. [19] approaches this problem by training a model with both a visual and physical understanding of multi-object interactions. They combine diverse demonstration data with self-supervised interaction data, aiming to leverage the interaction data to build transferable models and the demonstration data to guide the model-based RL planner to solve complex tasks. Fang et al. [22] proposes a method to jointly optimize both the transferable tool-choosing policy and the manipulation policy for the chosen tool. The training process of the model is based on large-scale simulated self-supervision with procedurally generated tool objects. However, this part is the same as the previous part, without considering the composite action space. For composite action space, similarity measuring and structure capturing become extremely difficult.

## 2.5 Multi-Agent Reinforcement Learning with Graph Neural Network

Graph neural networks (GNNs) are influential in extracting relations among entities and handling the variable number of agents, with emerging applications in MARL. RFM [44] designs an auxiliary action prediction task (predict other agents' actions) with graph networks [45], which can help agents learn interpretable intermediate representations. MAGNet [46] uses heuristic rules to learn the relevant graph to help actors and critics learning. DGN [47] learns the GCN together with the relation kernel by minimizing the TD error, which can be applied to dynamic multi-agent RL problems. HAMA [48] adopts a hierarchical graph attention network based on a pre-defined hierarchical graph to help agents capture interrelations. The pre-defined and fixed group scheme used in HAMA limits its adaptability in dynamic scenarios. PIC [49] introduce the graph neural network into the MADDPG [50] algorithm to overcome the permutation variant problem and propose an invariant permutation critic, which yields identical output irrespective of the agent permutation. To simplify the learning process and get better asymptotic performance, GA-AC [51] model the relationship between agents by a complete graph and propose a novel game abstraction mechanism based on a two-stage attention network and graph neural network.

**Remarks.** Although the above methods are not specifically proposed to solve the variable number of agents, they can naturally handle a variable number of agents due to the characteristics of graph neural networks. However, in the problems dealt with by these methods, agents are generally homogeneous. Even if they are heterogeneous, they only need to complete different tasks in the environment, and the action space is the same. The above two cases allow these methods to directly use graph neural networks to perform end-to-end black-box optimization, which can learn the correlation between agents. However, this does not apply to heterogeneous sub-action spaces in the composite action space. The heterogeneity of these sub-action spaces makes it impossible to capture the correlation between them by simply using graph neural network modeling. This requires some additional representation learning techniques, such as the hierarchical variational autoencoder based on unsupervised learning introduced by the SCORE algorithm. Although the MARL methods mentioned above can handle the variable number of agents, they cannot transfer to scenarios with agents not seen during training. The SCORE algorithm can quickly transfer to tasks with unseen sub-action spaces through fine-tuning a small number of samples after the algorithm is well-trained.

## 3 Preliminaries

### 3.1 Fully Cooperative Partially Observable Stochastic Game

Partially observable stochastic game (POSG) [52] is denoted as a tuple based on Markov Game as follows:

$$\left\langle \mathcal{X}, \mathcal{S}, \left\{\mathcal{A}^i\right\}_{i=1}^n, \left\{\mathcal{O}^i\right\}_{i=1}^n, \mathcal{P}, \mathcal{E}, \left\{\mathcal{C}^i\right\}_{i=1}^n \right\rangle,$$

where $n$ is the total number of agents; $\mathcal{X}$ represents the agent space; $\mathcal{S}$ is a finite set of states; $\mathcal{A}^i$ is a finite action set of agent $i$; $\boldsymbol{\mathcal{A}} = \mathcal{A}^1 \times \mathcal{A}^2 \times \cdots \times \mathcal{A}^n$ is the finite set of joint actions; $\mathcal{P}(s'|s, \boldsymbol{a})$ is the Markovian state transition probability function; $\mathcal{O}^i$ is a finite observation set of agent $i$; $\boldsymbol{\mathcal{O}} = \mathcal{O}^1 \times \mathcal{O}^2 \times \cdots \times \mathcal{O}^n$ is the finite set of joint observations, $\mathcal{E}(\boldsymbol{o}|s)$ is Markovian observation emission probability function; $\mathcal{C}^i : \mathcal{S} \times \boldsymbol{\mathcal{A}} \times \mathcal{S} \rightarrow \mathbb{R}$ denotes the cost function of agent $i$.

The game in POSG unfolds over a finite or infinite sequence of stages (or timesteps), where the number of stages is called *horizon*. In this paper, we consider the finite horizon problem. The objective for each agent is to minimize the expected cumulative cost received during the game. For a cooperative POSG, we quote the definition in [53],

$$\forall x \in \mathcal{X}, \forall x' \in \mathcal{X}\backslash\{x\}, \forall \pi_x \in \Pi_x, \forall \pi_{x'} \in \Pi_{x'}, \frac{\partial \mathcal{C}^{x'}}{\partial \mathcal{C}^x} \geqslant 0,$$

where $x$ and $x'$ are a pair of agents in agent space $\mathcal{X}$; $\pi_x$ and $\pi_{x'}$ are the corresponding policies in the policy space $\Pi_x$ and $\Pi_{x'}$ separately. Intuitively, this definition means that there is no conflict of interest for any pair of agents. The most common example of cooperative POSG is the fully cooperative POSG (also called decentralized partially observable Markov decision process, Dec-POMDP), that all the agents share the same global cost at each stage, and $\mathcal{C}^1 = \mathcal{C}^2 = \cdots = \mathcal{C}^n$.

In this paper, we model the time-varying composite action space problem as a fully cooperative POSG. Each agent completes a common task based on the shared local observation and reward function. Without loss of generality, the optimization goal of the fully cooperative POSG problem is defined as follows

$$\min_{\Psi} \sum_{i=1}^{n} \sum_{t=0}^{T-1} \mathbf{E}_{s \sim d_{\Psi}, \boldsymbol{o} \sim \mathcal{E}, a \sim \boldsymbol{\pi}_{\Psi}} \left[c_{t+1}^i\right], \tag{1}$$

where $\Psi := \{\psi^i\}_{i=1}^n$ denotes the parameters of the approximated policy function of all agents and $\boldsymbol{\pi}_{\Psi} := \prod_{i=1}^n \pi_{\psi^i}^i$ represents the factorizable joint policy of all agents. Note $d_{\Psi}$ is the stationary state distribution with respect to $\boldsymbol{\pi}_{\Psi}$.

### 3.2 Graph Neural Networks and Graph Attention Networks

Graph neural networks(GNNs) [54, 55] is the targeted neural network for graph data. Many machine learning problems are established with a natural graph structure, while the GNNs have been broadly adopted. Many variants of GNNs have been proposed, and we here take the popular GraphNets [45] as the main framework. Graph data is often defined as a tuple $\mathcal{G} = (u, \mathcal{V}, \mathcal{E})$, where $u$ denotes the global attribute; $\mathcal{V} = \{v_i\}_{i=1:N^v}$ and $\mathcal{E} = \{(e_k, r_k, s_k)\}_{k=1:N^e}$ denote the node feature set and the edge feature set ($e_k$ is the edge feature; $r_k$ and $s_k$ are the receiver's index and sender's index).

GraphNets treat GNNs as the combination of multiple "graph network(GN)" blocks. The GN blocks contain three updating and aggregation functions, which operate on different levels (node, edge, and global graph level). $\rho^{e\rightarrow v}$, $\rho^{e\rightarrow u}$, $\rho^{v\rightarrow u}$ are denoted as the aggregation functions for "edge to node", "edge to global" and "node to global" respectively. Different from $\bar{e}' = \rho^{e\rightarrow u}, \bar{v}' = \rho^{v\rightarrow u}$ that aggregate across all edges and nodes, $\bar{e_i}' = \rho^{e\rightarrow u}(E_i')$ aggregates the received/sent edges only for node $i$. For invariant input permutation, the aggregation function can be mean, element-wise summation, attention, etc. The updating functions $\phi^e(e_k, v_{r_k}, v_{s_k}, u)$, $\phi^v(\bar{e_i}', v_i, u)$, $\phi^u(\bar{e}', \bar{v}', u)$ update edges, nodes and global features respectively. The typical GN blocks keep the following update scheme

$$\phi^e \rightarrow \rho^{e\rightarrow v} \rightarrow \phi^v \rightarrow \rho^{e\rightarrow u} \rightarrow \rho^{v\rightarrow u} \rightarrow \phi^u.$$

The graph attention network (GAT) used in this paper is a particular case of a graph neural network whose aggregation function is the attention operator:

$$\mathbf{x}_i' = \alpha_{i,i}\Theta\mathbf{x}_i + \sum_{j\in\mathcal{N}(i)} \alpha_{i,j}\Theta\mathbf{x}_j,$$

where $\mathbf{x}_{i,j}$ means arbitrary feature vector, $\Theta$ is a shared weight matrix parameterizes a linear transformation, and the attention coefficients $\alpha_{i,j}$ are computed as:

$$\alpha_{i,j} = \frac{\exp\left(\text{Leaky ReLU}\left(\mathbf{a}^\top \left[\Theta\mathbf{x}_i \| \Theta\mathbf{x}_j\right]\right)\right)}{\sum_{k\in\mathcal{N}(i)\cup\{i\}} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top \left[\Theta\mathbf{x}_i \| \Theta\mathbf{x}_k\right]\right)\right)}.$$

## 4 The SCORE Algorithmic Framework

In the following, we will propose our SCORE algorithm formally. The SCORE algorithm can be considered as a two-stage structure framework, i.e., *action space representation learning stage* and *robust and transferable policy learning stage*. The main idea of these two stages are summarized first, and the detailed discussions are presented as follows:

1. *Action space representations learning*. We use an unsupervised-representation-learning-based hierarchical variational autoencoder (HVAE) to encode each sub-action space into a latent representation (embedding). This representation expresses the sub-action space properties present in the set of diverse observations collected in the environment by randomly sampled sub-actions. In the following policy learning stage, we use the action encoder in HVAE, take the sub-action space embedding as the conditional input, project each sub-action space into a commonly hidden space, and transform the heterogeneous

problem into a more straightforward homogeneous problem.

2. *Robust and transferable policy learning*. We propose a flexible graph-attention-network-based centralized critic and decentralized actor architecture to incorporate sub-action representations as inputs and model the dependencies between sub-actions, which can be trained through multi-agent reinforcement learning (Section 4.2). We provide a training procedure to control overfitting to the training sub-action set, making the policy better adapt to the structure of time-varying composite action space and transfer better to unseen sub-actions. We also incorporate an implicit credit assignment structure to overcome the multi-agent credit assignment problem.

## 4.1 Action Space Representation Learning

Since each sub-action space in the time-varying composite action space impacts different aspects of the environment, it is difficult to directly model the dependencies between sub-actions and transfer the algorithm to unseen sub-actions. For example, in precision agriculture tasks, the sub-action for temperature adjusts the room temperature of the greenhouse. In contrast, the carbon dioxide controller adjusts the concentration of carbon dioxide in the greenhouse. Without prior knowledge of experts, it is difficult for us to directly model the dependency between these two sub-actions from the data sampled by the agents.

However, the impact of these sub-actions on different aspects of the environment is ultimately reflected in the crop yield. Therefore, we can make a reasonable assumption: each sub-action space affects common latent variables related to crop yield, such as enzyme activity[2]. Suppose we project each sub-action space into the same latent action space. In that case, we can more easily capture the dependencies between each sub-action and can also transfer to unseen sub-actions more naturally.

The critical challenge is how to learn the representation of the entire sub-action space. Similar with Jain et al. [20], we use the observations sampled by different sub-actions to encode the entire sub-action space. It is worth mentioning that this is also one of the main differences between us and Jain et al. [20]. The latter is to encode each action in one action space separately, while we are encoding the entire sub-action space.

To encoding the entire sub-action space, we employ the hierarchical variational autoencoder (HVAE) architecture [25]. We first use a behavior policy (random policy in this paper) to collect observations for each sub-action space. Specifically, to encode the sub-action space more thoroughly, behavior policy needs to collect observation samples with sufficient diversity. For each sub-action space, the behavior policy will first fix other sub-actions to corresponding random values and randomly sample the currently processed sub-action and collect a batch of observation samples. The above process is repeated many times, and finally, all the collected observation samples are summarized. The obtained set is the observation sample set corresponding to the current sub-action space.

2. This is an assumption on the task of precision agriculture, and other scenarios should have similar assumptions.

Then HVAE encodes the observation sample set corresponding to each sub-action space into a single sub-action space embedding. Finally, the sub-action space embedding and a specific sub-action belonging to this sub-action space are used to condition the VAE of the observation corresponding to the specific sub-action. Here, we use sub-action space embedding together with a specific sub-action as the input of VAE is to get the projection of the sub-action in the common latent action space. We can capture the dependency between different sub-actions through independent transition data sampled by the agents only by getting the projection of the sub-action in the latent space instead of the projection in the original space. To enhance the representation ability of action space embedding, similar with [56], we add a self-supervised auxiliary task based on the original HVAE. In addition to outputting reconstructed observations, the decoder also predicts observations at future moments. The schematic diagram of the above process is shown in Figure 3.

Formally, for each sub-action space $\mathcal{A}_i \in \mathcal{A}$, HVAE encodes its associated observation sample set $\{o_{i,j}^k\} \in \mathcal{O}_i$ into a embedding $c_i$ by mean-pooling over the individual observations $o_{i,j}^k$. For the fixed sub-actions $\mathcal{A}_{-i}$, a total of $k$ fixed random values $\boldsymbol{a}_{-i}^k$ are included. Each random value $\boldsymbol{a}_{-i}^k$ corresponds to $j$ randomly sampled sub-actions $a_{i,j}^k$ of the current sub-action space $i$, so the observation sample set $\{o_{i,j}^k\}$ contains a total of $k \times j$ observation samples. We refer to this action space encoder as action space representation module $q_\phi(c_i|\mathcal{O}_i)$. The sub-action space embedding $c_i$ sampled from the action space representation module and sub-action $a_{i,j}^k$ is used to condition the action encoder $q_\psi(z_{i,j}^k|o_{i,j}^k, a_{i,j}^k, c_i)$ and action decoder $p_u(o_{i,j}^k|z_{i,j}^k, c_i)$ for each individual observation $o_{i,j}^k \in \mathcal{O}_i$. In addition, there also is a observation predictor $p_s([o_{i,j}^k]_t|z_{i,j}^k, c_i)$ where $[o_{i,j}^k]_t$ represents future $t$ timesteps' observations after $o_{i,j}^k$. LSTM is used as the observation predictor $p_s([o_{i,j}^k]_t|z_{i,j}^k, c_i)$ to predict the future trajectory given the current observation $o_{i,j}^k$. The entire HVAE framework is trained with the following three-part losses: 1) the reconstruction loss of all individual observation $o_{i,j}^k$; 2) the KL-divergence regularization of action encoder $q_\phi$ and $q_\psi$ with their respective prior distribution $p(c)$ and $p(z|c_i)$; 3) the prediction loss of all individual observation $o_{i,j}^k$. The final training objective requires maximizing the Evidence Lower-Bound (ELBO):

$$\mathcal{L}_e = \sum_{\mathcal{O}\in O} \Bigg\{ \mathbb{E}_{q_\phi(c|\mathcal{O})} \Bigg[ \sum_{o\in\mathcal{O}} \mathbb{E}_{q_\psi(z|o,a,c)} \log p_u(o \mid z, c) + \sum_{o\in\mathcal{O}} \mathbb{E}_{q_\psi(z|o,a,c)} \log p_s([o] \mid z, c) - D_{KL}\left(q_\psi \| p(z \mid c)\right)\Bigg] - D_{KL}\left(q_\phi \| p(c)\right)\Bigg\} \tag{2}$$

## 4.2 Robust and Transferable Policy Learning

In this section, we use the CCDA framework to learn the factorizable joint policy. With the sub-action space embedding and action encoder learned in the previous section, we can easily project different sub-actions into the same latent action space and use the graph attention network to capture the dependencies between different sub-actions. So that the exploration will be more efficient and the joint policy will be better approximated. The network architecture is shown in Figure 4.

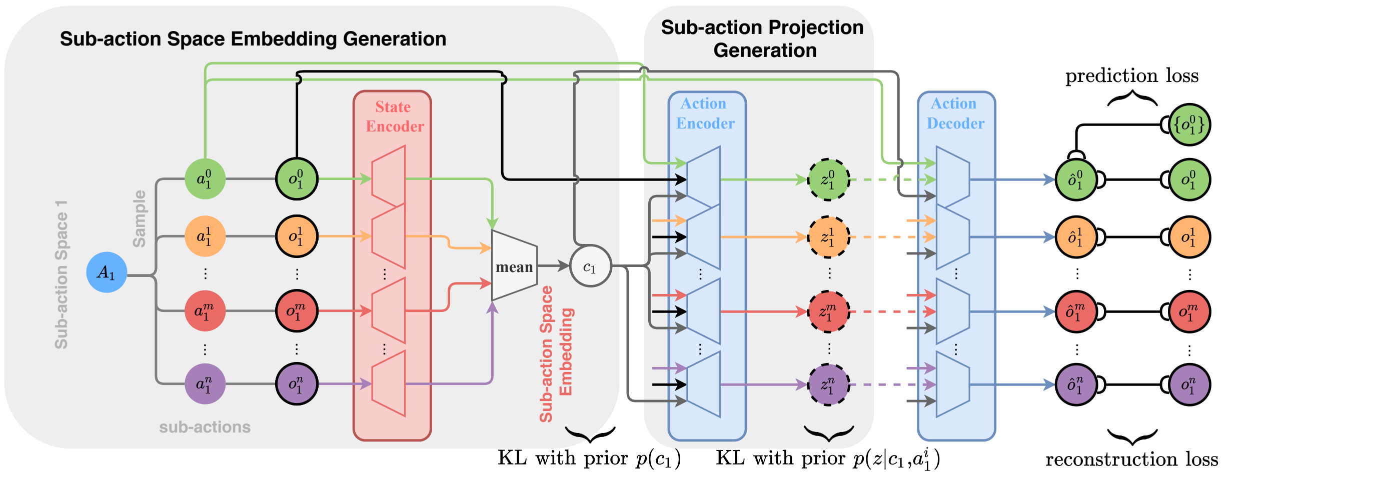


Fig. 3. The action space representation learning stage based on hierarchical variational autoencoder. For the sake of clarity, only the training process of the embedding of one sub-action space is shown here. For each sub-action space, SCORE will first fix the others sub-actions and then sample a batch of local observations from the environment by randomly sample a batch of current sub-action. Then, the local observation set processed by the state encoder is mean-pooled to obtain the sub-action space embedding distribution. The latter action encoder and action decoder constitute a standard conditional variational autoencoder. The input is the sub-action space embedding (sample from the embedding distribution), the local observation, and the sub-action at a certain timestep. The output is the reconstructed local observation. Besides enhancing the representation ability of the sub-action space embedding, we introduce a self-supervised auxiliary task to predict the local observation sequence in the future.

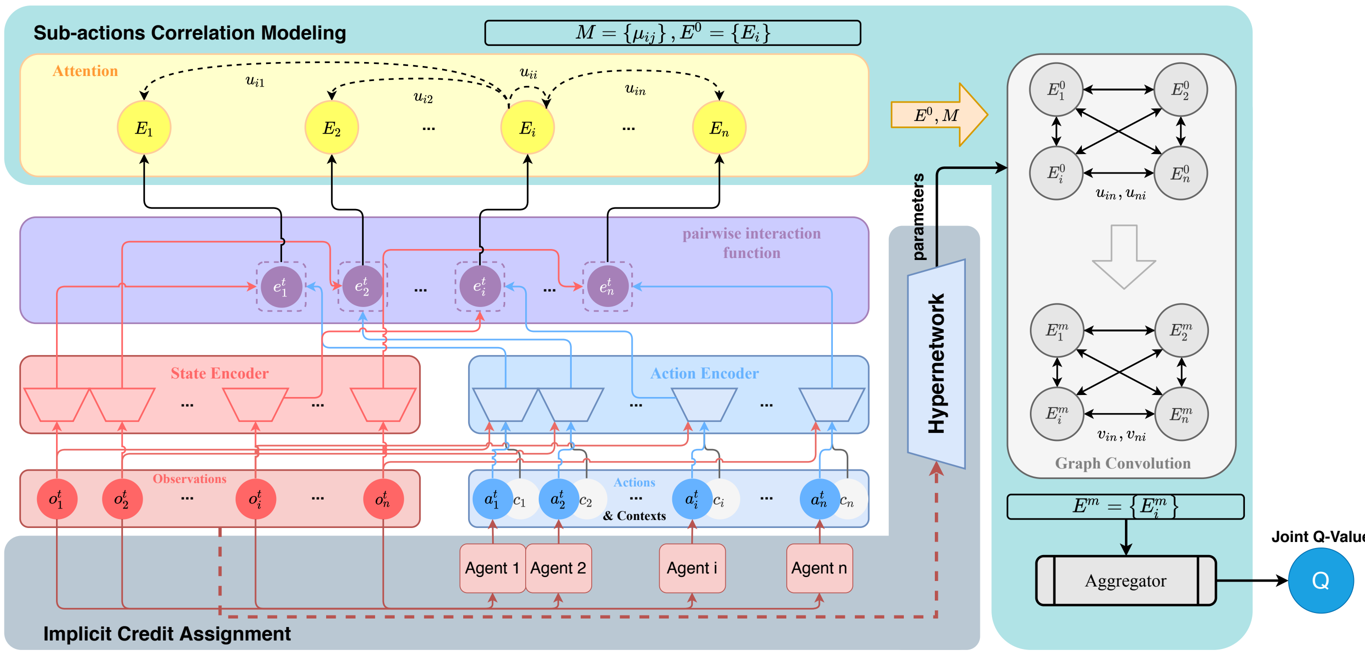


Fig. 4. The graph-attention-network-based centralized critic and decentralized actor architecture for robust and transferable policy learning. At each timestep, each sub-action output by different agents will be input into the action encoder together with each sub-action space embedding, and the projected sub-action representation is obtained. After passing the state-action pairing interaction function and state embedding, the obtained agent representation will pass through the graph attention network. By fully capturing the correlation between different sub-actions, the joint Q value function is finally obtained. The decentralized actor and centralized critic based on hypernetwork form an implicit credit allocation structure similar to Zhou et al. [57].

The centralized critic includes four sub-modules: state encoder $q_\phi$, action encoder $q_\psi$, pairwise interaction module $p_w$ and graph attention module $g_\Theta$. The state encoder and the action encoder are the action space representation module and the action encoder in the previous section. The parameters of these two modules are not updated by backpropagation. At each timestep $t$, the state encoder $q_\phi$ and the action encoder $q_\psi$ receive the local observation $o_i^t$ and the sub-action output $a_i^t$ by the current policy network of each agent $i$ as inputs. At the same time, the sub-action space embedding $c_i$ corresponding to each agent $i$ will also be input into the action encoder:

$$e_{o,i}^t = q_\phi(o_i^t), \quad e_{a,i}^t = q_\psi(o_i^t, a_i^t, c_i).$$

Next, the local observations embedding $e_{o,i}^t$ of each agent $i$ and the projection of the sub-action $e_{a,i}^t$ in the common latent space will be input into the pairwise interaction module $p_w$ together to form the embedding of each agent $i$ at the current timestep $t$:

$$e_i^t = p_w(e_{o,i}^t, e_{a,i}^t).$$

We then compute the attention weights from agent $i$ to $j$ using these embedding as:

$$\alpha_{i,j} = \frac{\exp\left(\text{Leaky ReLU}\left(\mathbf{a}^{(1)\top}\left[\theta^{(1)}e_i^t \| \theta^{(1)}e_j^t\right]\right)\right)}{\sum_{k\in\mathcal{N}(i)\cup\{i\}}\exp\left(\text{LeakyReLU}\left(\mathbf{a}^{(1)\top}\left[\boldsymbol{\theta}^{(1)}e_i^t \| \theta^{(1)}e_k^t\right]\right)\right)},$$

where the graph attention module is parameterized by $\Theta^{(1)} := \{\mathbf{a}^{(1)}, \theta^{(1)}\}$, and $[\cdot\|\cdot]$ represents the concatenate operation. The updated agent embedding is a weighted summation with the attention coefficient as the weight:

$$e_i^{t(1)} = \alpha_{i,i}\theta^{(1)}e_i^t + \sum_{k\in\mathcal{N}(i)} \alpha_{i,j}\theta^{(1)}e_k^t.$$

Above graph attention convolution operation is performed $m$ times and we denote the output of $m$th layer $e_i^{t(m)}$. Finally, we mean-pool all updated embeddings in the graph and get the joint Q-value of all agents:

$$e^t = \text{ Mean }\left(e_i^{t(m)}\right), \quad Q_t(\{o_i^t\}, \{a_i^t\}) = f_\xi(e^t).$$

In a fully cooperative POSG, all agents share the same reward function, and the multi-agent credit assignment problem will be introduced [58]. Although there are some works [59, 60, 61] that try to solve this problem and have achieved good results, these algorithms have their limitations. None of them can effectively deal with variable numbers of agents and composite action spaces. To solve this problem, we borrowed the idea of Zhou et al. [57] to implicitly achieves credit assignment among agents. Since the centralized critic is based on graph attention networks, it can naturally handle the variable number of agents. At the same time, the CCDA framework is based on the SAC [62] algorithm, so the heterogeneous composite action space is no longer a problem[3].

Specifically, similar to Rashid et al. [60], we use the current state $s^t$ and the hypernetwork [64] to generate the parameters $\{\Theta^{(l)}\}_{l=1}^m$ of the graph attention network. Formally, for layer $l$ of the graph attention network, we have $\Theta^{(l)} = \text{ MLP }(s_t)$. See Algorithm 1 for the complete pseudocode. It is worth noting that, to enhance the robustness of the algorithm, that is, to effectively cope with the situation that some sub-action spaces are randomly temporarily invalid, we randomly mask some sub-actions during training. The above trick is also called DropNode, and it has been proven effective in GNN training [65].

**Algorithm 1:** The SCORE algorithm.

```
Input: initial policy parameter $\varepsilon_i$ for each agent $i$,
       $Q$-function $Q_1, Q_2$, empty replay buffer $\mathcal{D}$;
1  Set target $Q$-function $\bar{Q}_1, \bar{Q}_2$ equal to main
   $Q$-function; while not convergence do
2      Observe local observation $o_i$ and select action
        $a_i \sim \pi_\varepsilon(\cdot \mid o_i)$ for each agent $i$;
3      Concatenate all action $a = [a_1\| \cdots \|a_n]$ and
        execute $a$ in the environment;
4      Observe next local observation $o_i^{'}$ of each agent $i$,
        global reward $r$, and global done signal $d$;
5      Store $(\{o_i\}, \{a_i\}, r, \{o_i^{'}\}, d)$ in replay buffer $\mathcal{D}$;
6      if $d$ is terminal then
7          Reset the environment;
8      end
9      if it's time to update then
10         for j in range(however many updates) do
11             Randomly sample a batch of transitions,
                $\mathcal{B} = \{(\{o_i\}, \{a_i\}, r, \{o_i^{'}\}, d)\}$ from $\mathcal{D}$;
                Compute targets for the $Q$ function:
                  $y(r, \{o_i'\}, d) =$
                  $r + \gamma(1-d)\left(\min_{j=1,2}\bar{Q}_j(\{o_i'\}, \{\tilde{a}_i'\})\right.$
                  $\left.-\alpha\sum_{i=1}^n \log\pi_{\varepsilon_i}(\tilde{a}_i' \mid o_i')\right), \quad \tilde{a}_i' \sim \pi_{\varepsilon_i}(\cdot \mid o_i')$
12             Update $Q$-functions by one step of
                gradient descent using
                $\nabla\frac{1}{|\mathcal{B}|}\sum_{\mathcal{B}}\left(Q_{j=\{1,2\}}(\{o_i\}, \{a_i\}) - y(r, \{o_i'\}, d)\right)^2;$
13             Update policy for each agent by one step
                of gradient ascent using
                $\nabla_{\varepsilon_i}\frac{1}{|\mathcal{B}|}\sum_{\mathcal{B}}\left(\min_{j=1,2}Q_j(\{o_i\}, \{\tilde{a}_{\varepsilon_i}(o_i)\})\right.$
                      $\left.-\alpha\log\sum_i \pi_{\varepsilon_i}(\tilde{a}_{\varepsilon_i}(o_i) \mid o_i)\right),$
14             where $\tilde{a}_{\varepsilon_i}(o_i)$ is a sample from $\pi_{\varepsilon_i}(\cdot \mid o_i)$
                which is differentiable w.r.t. $\varepsilon_i$ via the
                reparametrization trick (for continuous
                action space) or gumbel-softmax trick
                (for discrete action space);
15             Update target networks with
                $\bar{Q}_{j=\{1,2\}} \leftarrow \rho\bar{Q}_{j=\{1,2\}} + (1-\rho)Q_{j=\{1,2\}}.$
16         end
17     end
18 end
```

## 4.3 Transfer to Unseen Actions

The SCORE algorithm can be easily transferred to unseen sub-actions, as shown in Figure 5. For example, for precision agriculture tasks, with the development of intelligent greenhouse technology, new controllers will be added to the original control system. First, we use the action space representation module to get the embedding of the unseen sub-actions. Next, because the centralized critic based on the graph attention network can handle a variable number of agents, the unseen sub-actions can be naturally added to the original centralized critics. In this way, we do not need to retrain the entire actor-critic network, but only to fine-tune the pairwise interaction module $p_w$ and the graph attention module $g_\Theta$, and then train the actor-networks corresponding to the unseen sub-actions.

3. For discrete action space, Gumbel-softmax trick can be used, similar to the MAAC [63] algorithm.

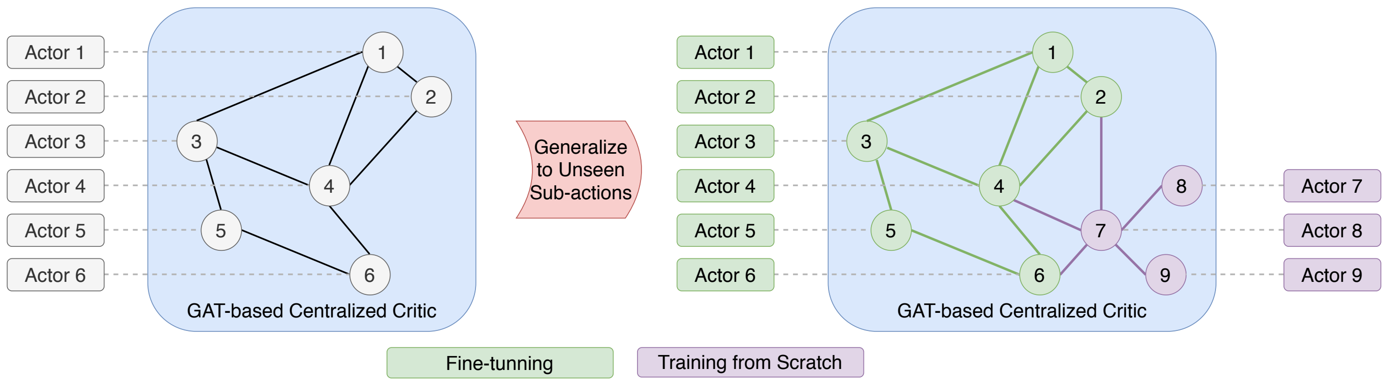


Fig. 5. How SCORE transfers to the unseen sub-action spaces. Since the centralized critic based on the graph attention network can handle any number of agents, we only need to represent the unseen sub-action spaces through the action space representation learning module and then fine-tune the centralized critic. Of course, we also need to train the actor-networks corresponding to the unseen sub-action spaces from scratch.

## 5 Experiments

### 5.1 Baselines

Since this paper transforms the single-agent problem into a multi-problem, to verify the effectiveness of the SCORE framework, we choose the PPO algorithm and the SAC algorithm as the single-agent baselines. In terms of multi-agent algorithms, considering the robustness of independent learning algorithms, we choose an independent learning algorithm based on SAC, ISAC, as one of the baselines. Besides, we choose the MADDPG [50] algorithm based on the CCDA framework as another baseline. To verify the effect of the attention mechanism in the graph attention network, we also compared the MAAC [66] algorithm that also uses the attention mechanism in the critic. Then, to verify the role of the credit assignment module in SCORE and the effectiveness of solving the team reward problem, we also compared the QMIX [60] algorithm. Since neither of these two algorithms can handle environments with continuous action spaces, we have divided a reasonable range in each sub-action space based on the output of the policy after our method converges. Next, we discretize the continuous sub-action space within this reasonable range to adapt to the algorithm settings of QMIX and MAAC. Finally, due to the similarity between the composite action space and the parameterized action space, we also choose the H-PPO [16] algorithm as one of the baselines.

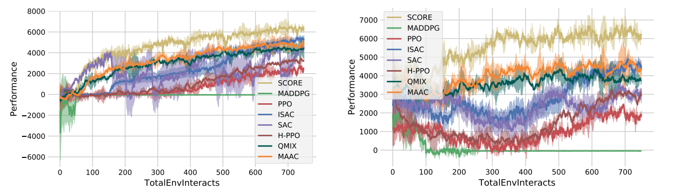
### 5.2 Proof-of-Concept Task: Spider

#### 5.2.1 Environment Details

To verify the efficiency of the SCORE algorithm, we first designed a simple proof-of-concept simulation environment. Specifically, we designed the Spider environment based on the Ant environment of Mujoco. Compared with the four identical legs in the Ant environment, the Spider environment contains two changes: 1) The Spider has a variable length of its four legs; 2) Spider adds four additional legs based on the Ant, two of which contain two joints (three parts), and the other two legs have no joints. Two two-joint legs and two zero-joint legs are symmetrically located in four directions of the body. Since the length of the single-joint legs is different, the legs on both sides of each leg are also different. In summary, we consider each leg in the Spider as a different sub-action space.

#### 5.2.2 Effectiveness

We first compare the performance of different algorithms in fully sub-action spaces. Figure 6(a) shows the average episode reward of each algorithm in the test environment under the fully 8 sub-action spaces. It can be seen from the figure that the performance of the two single-agent benchmark algorithms is not satisfactory. The learning speed of the PPO and H-PPO algorithm is too slow, while the SAC algorithm has poor stability. This directly reflects that it is unreasonable to model the time-varying composite action space as a single agent problem. ISAC, MAAC, and QMIX show the best performance except for SCORE, and the learning process was also very stable. This sufficiently shows that introduce decoupling to solve the time-varying composite action space problem is very effective. Although the MAAC algorithm also uses centralized critics, it uses the attention mechanism to effectively filter useless information, thus approximating the effect of decoupling. The MADDPG algorithm performed the worst among all algorithms and did not learn meaningful policies. After analysis, we believe that the poor performance of MADDPG is caused by the combination of centralized critics, deterministic policies, and the credit assignment problem. Although MADDPG uses the observations of all agents as input, the same is true for the PPO algorithm and the single agent algorithms such as SAC, and their performance is better than MADDPG. This is because the single-agent algorithm does not have a credit assignment problem. Further, MADDPG increases the exploratory nature of the algorithm by adding noise to the strategy. This method is more unstable and challenging to fine-tune. MAAC [66] replaces the DDPG algorithm with the SAC algorithm for learning stochastic policies and introduces counterfactual baselines to solve the credit assignment problem, which significantly improves the performance of the MADDPG algorithm. Moreover, in the experiment of MAAC, the MADDPG algorithm also appeared similar to the experiment in this paper. It can be seen from the results that the MAAC algorithm and the QMIX algorithm are significantly better than MADDPG. The reason why the QMIX algorithm works better is that



(a) Fully sub-action spaces. (b) Unseen sub-action spaces.

Fig. 6. Performance comparison of fully sub-action spaces and unseen sub-action spaces in Spider environment. The vertical axis represents the average episode return in 10 different test environments. The horizontal axis represents the total number of times the training phase interacts with the environment in millions of units. All algorithms are tested under three different random seeds, and the shaded area represents the standard deviation.

it implicitly solves the credit assignment problem [67] and achieves better exploration. The SCORE algorithm has a noticeable performance gap compared with ISAC, MAAC, and QMIX. This shows that it is essential to model the dependency relationship between the sub-action spaces after decoupling.

#### 5.2.3 *Robustness*

Next, we compare the robustness of each algorithm in the time-varying sub-action space of the Spider environment. To verify the stability of different algorithms, we randomly interfere with the part of the sub-action spaces in the fully sub-action spaces and then directly test the performance of the policies trained in the fully sub-action spaces. The 5 graphs in Figure 7 show the performance of different algorithms in the test environment. The abscissa represents the number of disturbing sub-action spaces, ranging from 1 to 7. The disturbed sub-action spaces are independently and randomly selected from all sub-action spaces at each timestep. To perform sufficient verification, we set 3 interference modes for the failed sub-action spaces: all zeros padding, random padding, and lasted padding (padding with the sub-action value at the previous time step). It can be seen from the figures that the performance of the baselines either drops sharply with the increase of the number of failed sub-action spaces or only maintains at a low level with minor fluctuations. In contrast, the SCORE algorithm is excellent for stability. From the comparison results of the three interference modes, it can be seen that the effect of all-zero padding is the best, the variance of lasted padding is the largest, and then the performance of random padding is the worst. All zeros padding is reasonable in practical tasks. Since the range of each sub-action in the Spider environment is $[-1, 1]$, 0 is a moderate value. In actual tasks, such as precision agriculture scenarios, when the controller is damaged, the output control signal is also a moderate default value.

#### 5.2.4 *Transferability*

Finally, we compare the transferability of each algorithm in the unseen sub-action spaces. We first remove 2 sub-action spaces from the entire composite action space so that all algorithms are trained in partially composite action space containing only 6 sub-action spaces. Next, we put the trained policies in the entire composite action space for finetuning. For ISAC, since each sub-action space is completely independent, only two additional agents need to be added when transferring to the unseen sub-action space. The policies of the original agents can be directly inherited. MAAC and QMIX algorithms show similar performance to ISAC. For SCORE, the graph-attention-network-based critic can handle a varying number of agents, although it is a centralized one. However, for MADDPG and other single-agent baselines, it cannot be directly transferred to the unseen sub-action spaces. We did not let these algorithms train from scratch in the unseen sub-action spaces to ensure a fair comparison. Specifically, we design the network structure of these baselines according to the number of the sub-action spaces in the entire composite action space but freeze the parameters of 2 sub-action spaces. Figure 6(b) shows the performance comparison during the finetuning process. It can be seen from the figure that all algorithms have an average reward greater than 0 at the beginning of the finetuning. This shows that the policies in the partially sub-action spaces are still effective in the unseen sub-action space. However, with the progress of the finetuning, except for the SCORE algorithm, the remaining baselines only converged slowly after a significant performance decline. This shows that directly migrating the original policies to the unseen sub-action spaces is not an efficient solution.

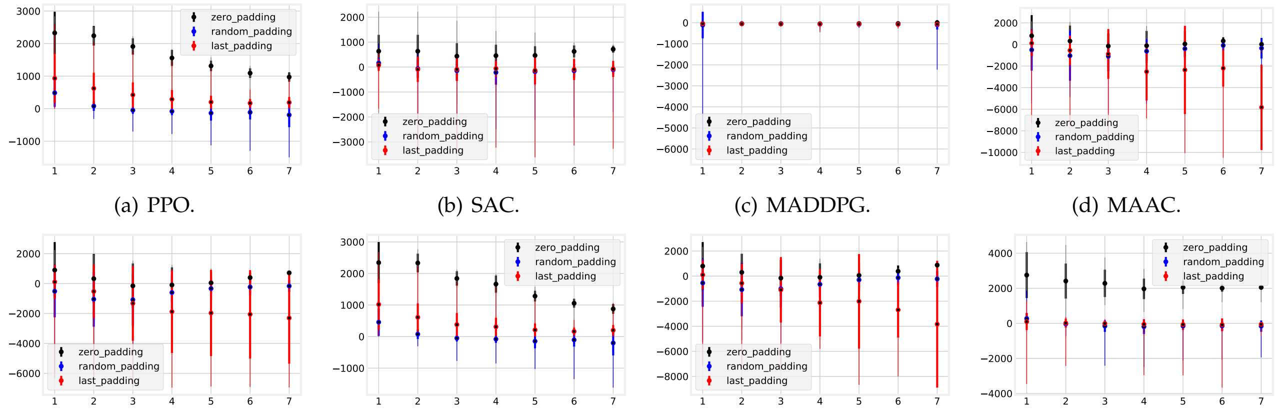
### 5.3 Precision Agriculture Task: Simulated Greenhouse

To verify the robustness and transferability of the SCORE algorithm on precision agriculture tasks, we conducted performance experiments on the simulated greenhouse environment.

#### 5.3.1 *Environment Details*

Since the agricultural tasks covered by the simulated greenhouse are very extensive, we selected one of the subtasks, the tomato simulator, for follow-up experiments to verify the SCORE algorithm more clearly. Tomato simulator can accurately simulate the changes in tomato growth in various aspects under different weather conditions. Table 1 shows the state space, action space, and reward function of the tomato simulator in detail. Specifically, the tomato simulator simulates the growth status of tomatoes under different planting strategies in an intelligent greenhouse for more than 160 days[4]. The weather status of each day is different and determined, read from a fixed database. The state information of each simulated day includes two parts: digital weather data and tomato status data. The digital weather data includes data from two consecutive simulated days: the 24-hour light intensity, outdoor temperature, outdoor humidity, outdoor wind speed, and virtual atmospheric temperature of the current simulated day, as well as the 24-hour temperature, humidity, carbon dioxide concentration, and light intensity in the greenhouse of the previous simulated day. Plant status data includes the number of planting days, watering volume, drainage volume, leaf area index, plant weight, number of plant branches and

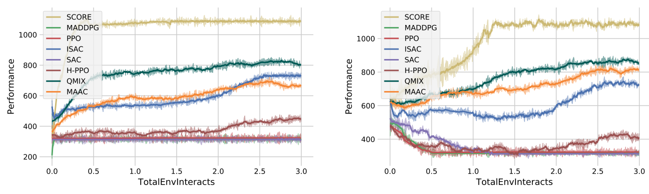
4. From late December to the end of May of the following year, a planting cycle of tomatoes is covered. Note that the normal growth cycle of tomatoes is about $2-3$ months. Since planting starts in winter, crops grow slowly, so the simulated planting cycle in the tomato simulator is longer.



(a) PPO. (b) SAC. (c) MADDPG. (d) MAAC.

(e) ISAC. (f) H-PPO. (g) QMIX. (h) SCORE.

Fig. 7. Comparison of the robustness of different algorithms in the Spider environment. The above results are obtained by conducting $10$ experiments with different random seeds in the test environment after the algorithm training converges. The dot represents the average episode return, the thick vertical line represents the standard deviation, and the thin vertical line represents the maximum and minimum values. The horizontal axis indicates the number of temporarily invalid sub-actions, from $1$ to $7$. The specific failure mechanism is as follows: at each timestep in an episode, a certain number of sub-actions will be randomly taken to invalidate. There are three failure modes, default value completion (black), random value completion (blue), and sub-action value completion of the previous timestep (red).



(a) Fully sub-action spaces. (b) Unseen sub-action spaces.

Fig. 8. Performance comparison of fully sub-action spaces and unseen sub-action spaces in tomato simulator. The vertical axis represents the average episode return in $10$ different test environments. The horizontal axis represents the total number of times the training phase interacts with the environment in millions of units. All algorithms are tested under $3$ different random seeds, and the shaded area represents the standard deviation.

leaves, cumulative tomato wet weight, cumulative tomato dry weight, and cumulative tomato number corresponding to the simulated day. All state information has a total of 223 dimensions and has been normalized to $[-1, 1]$ in advance. The control strategy includes the 24-hour greenhouse temperature, carbon dioxide concentration, light duration and end time, and irrigation start and end time for a total of 52 dimensions on each simulated day. We model the control strategy as a composite action space containing $4$ sub-action spaces: temperature sub-action (24-dimensional continuous value, range from $13$ to $25$), carbon dioxide sub-action (24-dimensional continuous value, range from $400$ to $700$), light sub-action (2-dimensional continuous value, range from $0$ to $24$), and irrigation sub-action (2-dimensional continuous value, range from $0$ to $24$). The reward (continuous value, range from $0$ to $30$) for each simulated day is represented by cumulative tomatoes wet weight.

### 5.3.2 *Effectiveness*

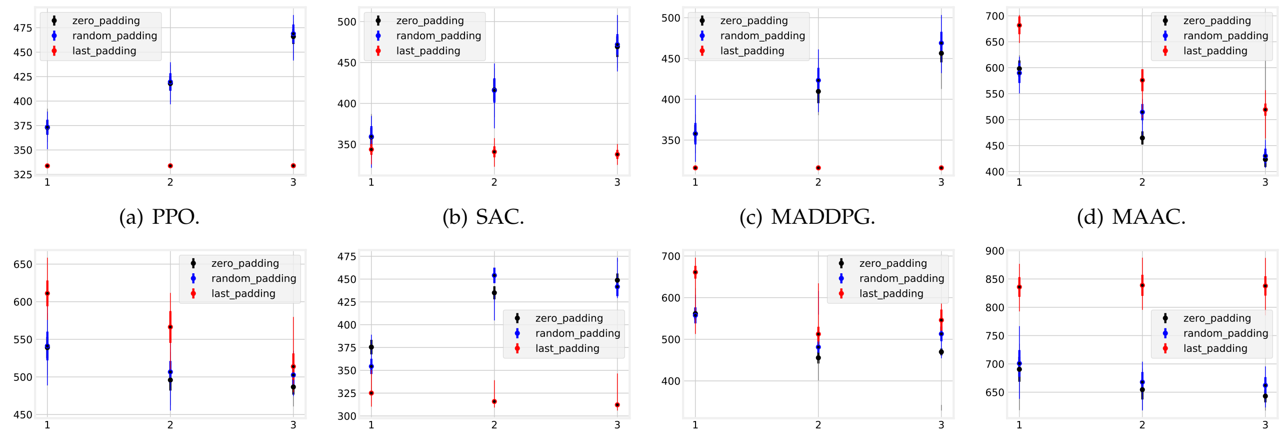
Like the Spider environment, we first compare the average episode rewards of each algorithm in the entire composite action space containing $4$ sub-action spaces. As can be seen from Figure 8(a), the three single-agent RL algorithms, PPO and SAC, H-PPO, and the MARL algorithm, MADDPG, cannot learn meaningful planting policies. This is similar to the experimental results obtained in the Spider environment. Although the tomato simulator has only $4$ sub-action spaces, the overall action vector length is $52$, making the tomato simulator much more complex than the Spider environment. The shortcomings of joint training of single-agent RL algorithms and the centralized critic of MADDPG have become more apparent. The ISAC, MAAC and QMIX algorithm still achieved good results in the tomato simulator, but its convergence speed is too slow. In contrast, the SCORE algorithm far surpasses the ISAC algorithm in convergence speed or final performance. This means that explicitly modeling the dependencies between the sub-action spaces can improve the exploration efficiency of the algorithm, thereby improving the convergence speed and final performance.

### 5.3.3 *Robustness*

Next, we analyze the robustness of each algorithm in the tomato simulator. The way the experimental results are obtained is similar to the Spider environment. The difference is the way of zero-padding. We no longer use $0.0$ for padding but use a moderate default value for each sub-action space. For example, for the sub-action space for temperature, the value range is $13.0$ to $25.0$, and we take $19.0$ as the default value. Through Figure 9, we found some interesting phenomena. For the two single-agent RL algorithms and the MARL algorithm MADDPG in the first row, as the number of temporarily invalid sub-action spaces gradually increases, the result of using zero-padding or random padding is gradually rising. The results obtained using the last padding have been maintained at a superficial level. In contrast, the two algorithms in the second row show that the results obtained by using the last padding are generally better than the other two padding methods. Combining the results of Figure 8(a) and the characteristics of the tomato simulator, we believe that the following reasons lead to the above results. First, because planting is an RL task with more delayed reward characteristics, the results obtained by

TABLE 1
The detailed information of tomato simulator.

| Elements | Description | | | Remarks | |
|---|---|---|---|---|---|
| **Observation** | Digital weather data | Previous simulated day | Temperature, humidity, carbon dioxide concentration, and light intensity in the greenhouse | Per hour | 223 dimensions, normalized to $[-1, 1]$ in advance |
| | | Current simulated day | Light intensity, outdoor temperature, outdoor humidity, outdoor wind speed, and virtual atmospheric temperature | Per hour | |
| | Crops status data | The number of planting days, watering volume, drainage volume, leaf area index, plant weight, number of plant branches and leaves, cumulative tomatoes wet weight, cumulative tomatoes dry weight, and cumulative tomatoes number | | Per day | |
| **Composite Action Space** | Sub-action for temperature | Continuous value | Range: $[13.0, 25.0]$ | Per hour | 52 dimensions |
| | Sub-action for carbon dioxide | Continous value | Range: $[400.0, 700.0]$ | Per hour | |
| | Sub-action for illumination | Discrete value | Start and end time, Range: $[0, 23]$ | Per day | |
| | Sub-action for irrigation | Discrete value | Start and end time, Range: $[0, 23]$ | Per day | |
| **Reward** | cumulative wet weight | Continuous value | Range: $[0.0, 30.0]$ | Per day | 1 dimension |



(a) PPO. (b) SAC. (c) MADDPG. (d) MAAC.

(e) ISAC. (f) H-PPO. (g) QMIX. (h) SCORE.

Fig. 9. Comparison of the robustness of different algorithms in tomato simulator. The above results are obtained by conducting $10$ experiments with different random seeds in the test environment after the algorithm training converges. The dot represents the average episode return, the thick vertical line represents the standard deviation, and the thin vertical line represents the maximum and minimum values. The horizontal axis indicates the number of temporarily invalid sub-action spaces, from $1$ to $3$. The specific failure mechanism is as follows: at each timestep in an episode, a certain number of sub-action spaces will be randomly taken to invalidate. There are three failure modes, default value completion (black), random value completion (blue), and action value completion of the previous timestep (red).

using the last padding method should generally be better than the other two padding methods; secondly, according to the results in Figure 8(a), the three algorithms in the first row have not learned an effective planting strategy, which makes the random strategy or the default strategy will achieve better results, so the result of the last padding is the worse than other two padding methods. Finally, for the three algorithms in the first row, as the number of temporarily invalid sub-action spaces increase, the tomato simulator becomes simpler, so the performance will gradually become better. Finally, we focus on comparing the experimental results of ISAC, MAAC, QMIX and SCORE. It can be seen from Figure 9(e) and Figure 9(h) that as the number of temporarily invalid sub-action spaces increases, the results obtained by ISAC, MAAC and QMIX using the last padding show a significant drop. In contrast, the drop in SCORE is tiny. This shows that the structure of SCORE makes it have excellent algorithm robustness.

### 5.3.4 *Transferability*

Finally, we compared the transferability of each algorithm in the unseen sub-action space under the tomato simulator. We first set the 2 sub-action spaces for temperature and irrigation to fixed default values and train each algorithm to converge under this setting; then transfer the trained algorithms to the fully sub-action spaces for finetuning. The transferring and finetuning method is consistent with the Spider environment. It can be seen from Figure 8(b) that the three single-agent RL algorithms PPO, SAC, H-PPO and the MADDPG algorithm achieve better performance at the beginning of the finetuning process than directly training in the entire action space. This is because the complexity of the tomato simulator with only 2 sub-action spaces is significantly reduced compared to the entire composite action space, and the fixed default values are also a good planting strategy. However, as the finetuning process continues, the performance of these three algorithms gradually declines and finally converges to performance similar to that of the entire action space. In the process of finetuning the ISAC, MAAC and QMIX algorithm, similar to the Spider environment, the performance drops first and then rises slowly. The SCORE algorithm can quickly converge with a higher performance start point. These phenomena once again show that directly transfer the trained policies to the unseen sub-action space is not a reasonable solution.

#### 5.3.5 *More Analysis*

To explore the reasons for the excellent performance of the SCORE algorithm in more depth, we compare the correlation between each sub-action space output by different algorithms. Specifically, we train each algorithm in the fully sub-action space to converge and then sample a composite action trajectory in the test environment. Each timestep in the trajectory contains 4 sub-actions. Since light and irrigation actions indicate the start and end time, while temperature and carbon dioxide indicate the specific values of each hour of the simulation day, we choose temperature and carbon dioxide as the discussion objects to better visualize the correlation. It can be seen from the Figure 10 that the temperature and carbon dioxide action output by the SCORE algorithm have a certain correlation, while the different sub-actions output by other algorithms are relatively independent. In addition, the sub-actions output by the SCORE algorithm and the ISAC algorithm are more concentrated, and have obvious changes over time.

## 5.4 Ablation Study

In this section, we perform an ablation analysis on the critical components of the SCORE algorithm. First, in order to explore the impact of sub-action space relationship modeling on the performance of the algorithm, we replaced the aggregation function of the graph neural network from the attention layer (AL) with the mean function (MF) and the sum function (SF); secondly, in order to verify the influence of hypernetwork (HN) on the multi-agent credit assignment, we removed the hypernetwork, and the parameters of the GNN network are directly updated according to the gradient; finally, to verify the effect of the action space representation learning on the relationship modeling of the sub-action spaces, we removed the action space representation learning (ASRL) stage. The action encoder and state encoder are trained from scratch. According to the above notation, the notation of the SCORE algorithm can be named as ASRL + AL + HN; The baseline using the mean function as the aggregate function can be named ASRL + MF + HN. The notation of other baselines is similar. We train all baselines in the fully sub-action spaces and the unseen sub-action spaces of the two environments to converge and then calculate the mean and variance of experiments under 10 different random seeds in the test environment. We first analyze the transferability of all algorithms. The final results are shown in Table 2.

It can be seen from the table that the impact of the HN module on the final performance is smaller than that of the ASRL module. This shows that the problem of multi-agent credit assignment in the Spider task and tomato planting task is not severe, and the excellent performance of the ISAC algorithm in these two tasks also proves this. After removing the ASRL module, the transferability of the SCORE algorithm in the unseen sub-action space is generally worse than the final performance in the fully sub-action space. This shows that the action space representation learning has a positive effect on improving the transferability of the algorithm. Without the HN module and the ASRL module, the SCORE algorithm can be regarded as modeling the central critic of the MADDPG algorithm with GNN. As can be seen from the table, the algorithm's performance is greatly improved compared to the MADDPG algorithm. This shows the importance of modeling the dependencies between the sub-action spaces after decomposing the composite action space.

Finally, we analyze the robustness of each algorithm in the time-varying action space. We use the converged performance of each algorithm in the fully sub-action spaces of the tomato simulator as a baseline and get the histogram shown in Figure 11. It can be seen from the figure that the action space representation learning plays a vital role in the robustness of the algorithm. After removing the ASRL module, the different aggregation functions of the graph neural network have little effect on the algorithm's robustness.

To analyze why the ASRL module affects the performance more deeply, we visualized the intermediate network result, that is, the agent embedding of the AL+HN algorithm and the ASRL+AL+HN (SCORE) algorithm. The results are shown in the Figure 12. It can be seen from the figure that the ASRL module can learn better action embedding and agent embedding so that the subsequent modules can better model the correlation between each sub-action space.

In addition to the methods described in the experimental part of this paper, we also conducted experiments with two additional methods, i.e., the mask training mechanism and the penalty training mechanism, to solve the problem that the baselines cannot be extended to the new action space. Specifically, the mask training mechanism is not much different from our current training mechanism. The difference is that at this time, the parameters of the output head corresponding to the redundant sub-action spaces are no longer frozen; The penalty training mechanism is based on the mask training mechanism, adding a penalty term, which calculates the deviation of the sub-actions output by these redundant output heads from the default sub-actions. The transfer results under new sub-action spaces are shown in the Table 3.

## 6 Conclusion

This paper proposes a structured cooperative reinforcement learning algorithm, SCORE, based on centralized critic and decentralized actor framework, aiming at the time-varying composite action space problem. SCORE introduces the idea of decoupling and decomposes the single-agent RL problem with time-varying composite action space into a variable

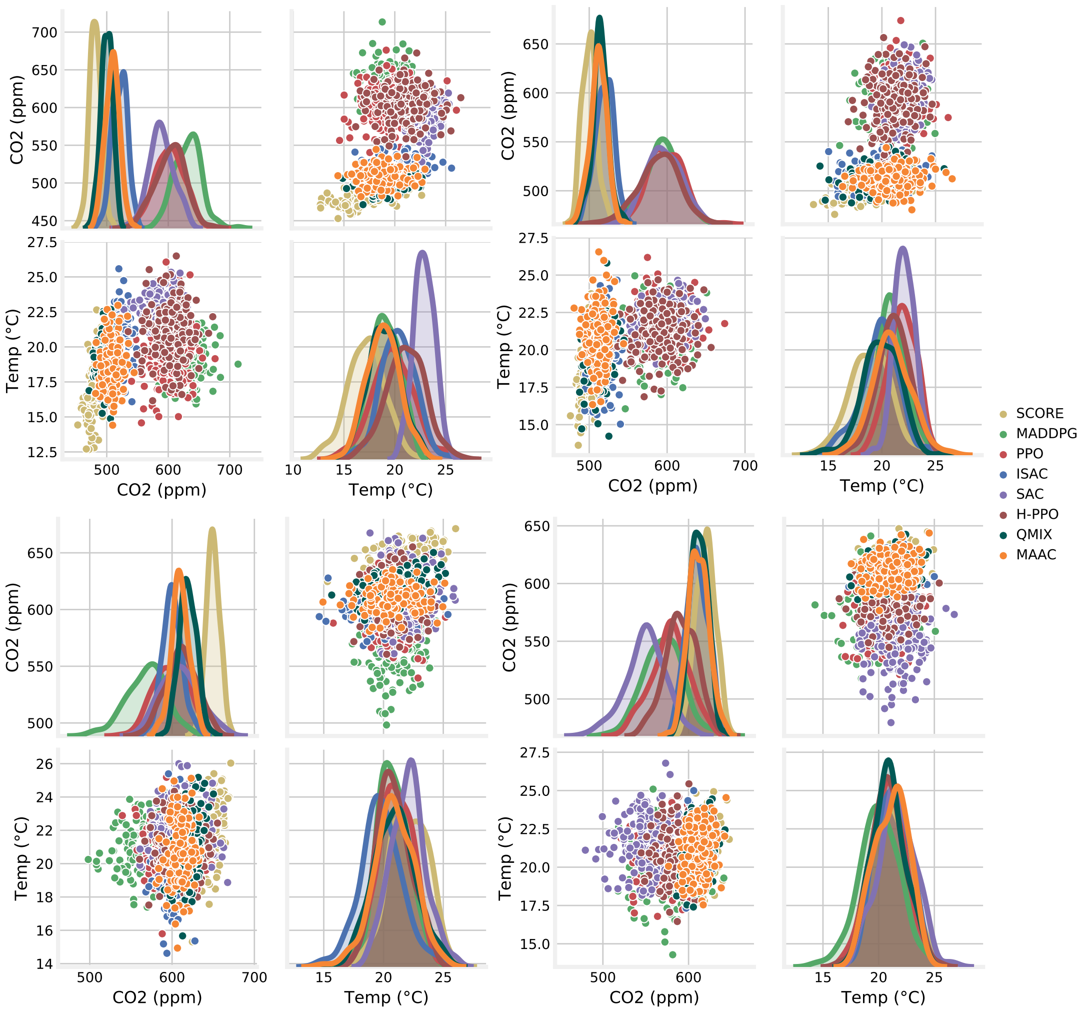


Fig. 10. The pairwise relationships of sub-actions for carbon dioxide and temperature in tomato simulator. These $2$ sub-action spaces are both $24$-dimensional, representing the hourly temperature and carbon dioxide concentration in $24$ hours of a simulated day. We select the temperature and carbon dioxide values at $0:00$ (upper left), $06:00$ (upper right), $12:00$ (lower left) and $18:00$ (lower right) in a whole sub-action trajectory. This action trajectory is sampled in the test environment after the algorithm training converges in the fully sub-action space of the tomato simulator.

TABLE 2
The ablation studies of the variants of SCORE algorithm. ASRL means action space representation learning stage; HN means hypernetwork in the policy learning stage; MF means we replace attention layer with mean function to be the aggregation function in GNN; SF means we replace attention layer with summation function to be the aggregation function in GNN; AL means use attention layer as the aggregation function in GNN. The mean and standard deviation are obtained under $10$ different random seeds in the test environment.

| | Spider | | | Tomato Simulator | | |
|---|---|---|---|---|---|---|
| | Fully Sub-Action Spaces | Unseen Sub-Action Spaces | | Fully Sub-Action Spaces | Unseen Sub-Action Spaces | |
| | Converge | Initial | Converge | Converge | Initial | Converge |
| **MF** | 4256(±93) | 2545(±205) | 4078(±265) | 524(±9) | 429(±13) | 481(±6) |
| **SF** | 4235(±95) | 2487(±227) | 3975(±283) | 537(±9) | 438(±15) | 490(±14) |
| **AL** | 4519(±112) | 2632(±335) | 4379(±145) | 552(±21) | 442(±6) | 512(±20) |
| **MF+HN** | 4796(±78) | 2786(±285) | 4568(±63) | 625(±13) | 486(±31) | 583(±25) |
| **SF+HN** | 4820(±203) | 2762(±321) | 4728(±262) | 629(±12) | 534(±22) | 576(±12) |
| **AL+HN** | 4887(±176) | 2711(±324) | 4792(±268) | 622(±12) | 519(±12) | 580(±12) |
| **ASRL+MF** | 5411(±181) | 2880(±245) | 5345(±131) | 752(±8) | 548(±13) | 767(±6) |
| **ASRL+SF** | 5200(±260) | 2737(±223) | 5220(±220) | 755(±15) | 529(±12) | 720(±31) |
| **ASRL+AL** | 5783(±69) | 2855(±326) | 5742(±103) | 836(±24) | 552(±30) | 856(±31) |
| **ASRL+MF+HN** | 5523(±230) | 3176(±310) | 5328(±160) | 854(±8) | 623(±35) | 823(±15) |
| **ASRL+SF+HN** | 5498(±154) | 3108(±286) | 5345(±165) | 867(±12) | 612(±43) | 833(±22) |
| **ASRL+AL+HN (SCORE)** | **6252**(±187) | **3365**(±329) | **6296**(±257) | **1125**(±9) | **648**(±35) | **1143**(±12) |

TABLE 3
Different baselines use different methods to transfer to the unseen sub-action spaces.

| | Spider | | Tomato Simulator | |
|---|---|---|---|---|
| | Unseen Sub-Action Spaces | | Unseen Sub-Action Spaces | |
| | Initial | Converge | Initial | Converge |
| **PPO** | 1267($\pm$982) | 1962($\pm$234) | 443($\pm$12) | 313($\pm$2) |
| **PPO-mask** | 1223($\pm$912) | 2007($\pm$208) | 458($\pm$9) | 308($\pm$2) |
| **PPO-penalty** | 1185($\pm$897) | 1786($\pm$208) | 415($\pm$10) | 313($\pm$1) |
| **SAC** | 2526($\pm$539) | 3143($\pm$156) | 512($\pm$10) | 312($\pm$2) |
| **SAC-mask** | 2550($\pm$554) | 3187($\pm$128) | 500($\pm$12) | 311($\pm$2) |
| **SAC-penalty** | 2399($\pm$508) | 2855($\pm$132) | 487($\pm$10) | 313($\pm$2) |
| **MADDPG** | 1634($\pm$535) | 0($\pm$0) | 485($\pm$10) | 312($\pm$2) |
| **MADDPG-mask** | 1588($\pm$587) | 0($\pm$0) | 483($\pm$8) | 311($\pm$2) |
| **MADDPG-penalty** | 1503($\pm$569) | 0($\pm$0) | 454($\pm$10) | 311($\pm$2) |
| **H-PPO** | 2834($\pm$227) | 3052($\pm$112) | 465($\pm$12) | 408($\pm$5) |
| **H-PPO-mask** | 2789($\pm$235) | 3145($\pm$98) | 423($\pm$12) | 418($\pm$4) |
| **H-PPO-penalty** | 2624($\pm$227) | 2958($\pm$62) | 406($\pm$9) | 368($\pm$3) |
| **QMIX** | 3250($\pm$187) | 3956($\pm$50) | 621($\pm$5) | 823($\pm$3) |
| **QMIX-mask** | 3329($\pm$156) | 3888($\pm$76) | 601($\pm$3) | 800($\pm$5) |
| **QMIX-penalty** | 3032($\pm$187) | 3662($\pm$88) | 584($\pm$3) | 767($\pm$5) |
| **MAAC** | 3189($\pm$202) | 4348($\pm$83) | 625($\pm$7) | 805($\pm$5) |
| **MAAC-mask** | 3103($\pm$206) | 4267($\pm$92) | 654($\pm$4) | 824($\pm$2) |
| **MAAC-penalty** | 2889($\pm$336) | 4022($\pm$84) | 584($\pm$3) | 752($\pm$4) |

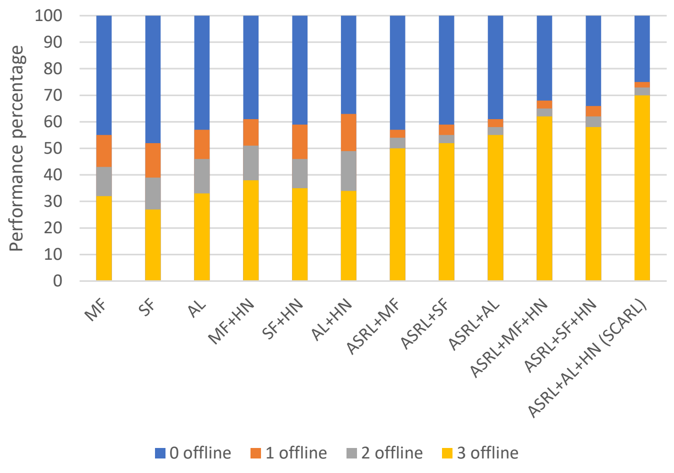


Fig. 11. The robustness of ablation algorithms in tomato simulator. ASRL means action space representation learning stage; HN means hyper-network in the policy learning stage; MF means we replace attention layer with mean function to be the aggregation function in GNN; SF means we replace attention layer with summation function to be the aggregation function in GNN; AL means use attention layer as the aggregation function in GNN. We use the converged performance of each algorithm in the fully sub-action spaces of the tomato simulator as a baseline ($100\%$). All results are obtained under $10$ different random seeds in the test environment. The longer the different color segments, the more performance degradation, representing the worse robustness. Here the "offline" means temporarily invalid.

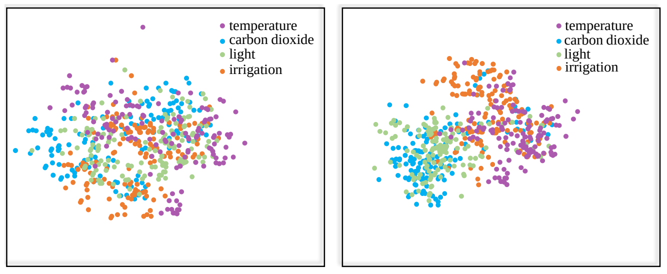


Fig. 12. Agent embedding visualization. The **Left** figure shows the visualization result of the **AL+HN** algorithm agent embeddings; the **Right** figure shows the visualization result of the **ASRL+AL+HN (SCORE)** algorithm agent embedding. The above two-dimensional image is obtained by reducing the dimension of the agent embedding through the t-SNE algorithm, and different colors represent different sub-action spaces. Each point represents a timestep in a trajectory sampled by the converged algorithm in the test environment.

number of heterogeneous MARL problems. In order to deal with the variable number of multi-agent scenarios, the nonstationary environment problem and multi-agent credit allocation problem introduced after decomposition, SCORE introduces a centralized critic based on the graph attention network and an implicit credit assignment structure; Simultaneously, to better handle the unseen sub-action spaces and capture the dependencies between the agents, we also introduce the hierarchical variational autoencoder, a feature extractor based on unsupervised learning.

Experiments in the proof-of-concept task and precision agriculture task show that the SCORE algorithm is significantly better than the single-agent and multi-agent baselines in algorithm robustness and transferability. The ablation analysis shows that after the single-agent problem is decomposed into multi-agent problems, the use of graph attention networks to capture the dependencies between agents explicitly impacts the algorithm's performance. Besides, the hierarchical variational autoencoder based on unsupervised learning can play a positive role in the robustness and transferability of the algorithm.

## ACKNOWLEDGMENTS

This work was supported in part by National Key Research and Development Program of China (No.

2020AAA0107400), NSFC (No. 12071145), Shanghai Municipal Science and Technology Major Project (No. 2021SHZDZX0102), STCSM (No. 18DZ2270700 and 20511101100), the Open Research Projects of Zhejiang Lab (NO.2021KE0AB03) and a grant from Shenzhen Institute of Artificial Intelligence and Robotics for Society.

## References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[2] Y. Hua, X. Wang, B. Jin, W. Li, J. Yan, X. He, and H. Zha, "Hyper-meta reinforcement learning with sparse reward," *Arxiv:2002.04238*, 2020.

[3] V. H. Pong, M. Dalal, S. Lin, A. Nair, S. Bahl, and S. Levine, "Skew-fit: State-covering self-supervised reinforcement learning," in *ICML*, 2020.

[4] H. Zhu, J. Yu, A. Gupta, D. Shah, K. Hartikainen, A. Singh, V. Kumar, and S. Levine, "The ingredients of real-world robotic reinforcement learning," in *ICLR*, 2020.

[5] A. Amini, I. Gilitschenski, J. Phillips, J. Moseyko, R. Banerjee, S. Karaman, and D. Rus, "Learning robust control policies for end-to-end autonomous driving from data-driven simulation," *IEEE Robotics and Automation Letters*, vol. 5, pp. 1143–1150, 2020.

[6] R. Furuta, N. Inoue, and T. Yamasaki, "Fully convolutional network with multi-step reinforcement learning for image processing," in *AAAI*, 2019.

[7] X. Liao, W. Li, Q. Xu, X. Wang, B. Jin, X. Zhang, Y. Zhang, Y. Wang, and Y. Zhang, "Iteratively-refined interactive 3D medical image segmentation with multi-agent reinforcement learning," in *CVPR*, 2020.

[8] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents (extended abstract)," *J. Artif. Intell. Res.*, vol. 47, pp. 253–279, 2013.

[9] J. Pazis and M. G. Lagoudakis, "Reinforcement learning in multidimensional continuous action spaces," *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 97–104, 2011.

[10] J. Pazis and R. Parr, "Generalized value functions for large action sets," in *ICML*, 2011.

[11] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, "Deep reinforcement learning in large discrete action spaces," *Arxiv:1512.07679*, 2015.

[12] Y. Zhang, Q. H. Vuong, K. Song, X. Gong, and K. W. Ross, "Efficient entropy for policy gradient with multidimensional action space," *Arxiv*, vol. abs/1806.00589, 2018.

[13] W. Masson, P. Ranchod, and G. Konidaris, "Reinforcement learning with parameterized actions," in *AAAI*, 2016.

[14] E. Wei, D. Wicke, and S. Luke, "Hierarchical approaches for reinforcement learning in parameterized action space," *Arxiv*, vol. abs/1810.09656, 2018.

[15] J. Xiong, Q. Wang, Z. Yang, P. Sun, L. Han, Y. Zheng, H. Fu, T. Zhang, J. Liu, and H. Liu, "Parametrized deep Q-networks learning: Reinforcement learning with discrete-continuous hybrid action space," *Arxiv*, vol. abs/1810.06394, 2018.

[16] Z. Fan, R. Su, W. Zhang, and Y. Yu, "Hybrid actor-critic reinforcement learning in parameterized action space," in *IJCAI*, 2019.

[17] G. Farquhar, L. Gustafson, Z. Lin, S. Whiteson, N. Usunier, and G. Synnaeve, "Growing action spaces," *Arxiv:1906.12266*, 2019.

[18] Y. Chen, Y. Chen, Y. Yang, Y. Li, J. Yin, and C. Fan, "Learning action-transferable policy with action embedding," *Arxiv:1909.02291*, 2019.

[19] X. Annie, E. Frederik, L. Sergey, and F. Chelsea, "Improvisation through physical understanding: Using novel objects as tools with visual foresight," in *RSS*, 2019.

[20] A. Jain, A. Szot, and J. Lim, "Generalization to new actions in reinforcement learning," in *ICML*, 2020.

[21] Y. Chandak, G. Theocharous, C. Nota, and P. S. Thomas, "Lifelong learning with a changing action set," in *AAAI*, 2020.

[22] K. Fang, Y. Zhu, A. Garg, A. Kurenkov, V. Mehta, L. Fei-Fei, and S. Savarese, "Learning task-oriented grasping for tool manipulation from simulated self-supervision," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 202–216, 2020.

[23] W. Li, B. Jin, X. Wang, J. Yan, and H. Zha, "F2A2: Flexible fully-decentralized approximate actor-critic for cooperative multi-agent reinforcement learning," *Arxiv:2004.11145*, 2020.

[24] H. Kim, J. Kim, Y. Jeong, S. Levine, and H. O. Song, "EMI: Exploration with mutual information," in *ICML*, 2019.

[25] H. Edwards and A. Storkey, "Towards a neural statistician," in *ICLR*, 2017.

[26] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *Arxiv:1409.0473*, 2014.

[27] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[28] United Nations, "World population prospects: The 2015 revision," 2015.

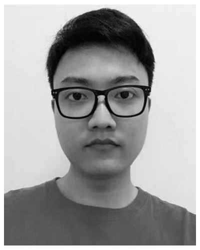
[29] Z. R. Shi, C. Wang, and F. Fang, "Artificial intelligence for social good: A survey," *Arxiv:2001.01818*, 2020.

[30] C. V. Lücken and R. Brunelli, "Crops selection for optimal soil planning using multiobjective evolutionary algorithms," in *AAAI*, 2008.

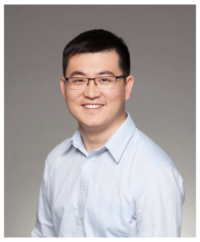
[31] ICRISAT, "Microsoft and icrisat's intelligent cloud pilot for agriculture in andhra pradesh increase crop yield for farmers," 2017. [Online]. Available: https://www.icrisat.org/microsoft-andicrisats-intelligent-cloud-pilot-for-agriculture-in-andhrapradesh-increase-crop-yield-for-farmers/

[32] D. Holman, M. Sridharan, P. Gowda, D. Porter, T. Marek, T. Howell, and J. Moorhead, "Estimating reference evapotranspiration for irrigation management

in the texas high plains," in *IJCAI*, 2013.
[33] J. You, X. Li, M. Low, D. Lobell, and S. Ermon, "Deep gaussian process for crop yield prediction based on remote sensing data," in *AAAI*, 2017.
[34] J. A. Quinn, K. Leyton-Brown, and E. Mwebaze, "Modeling and monitoring crop disease in developing countries," in *AAAI*, 2011.
[35] mCrops, 2016. [Online]. Available: http://34.242.164.142/mcrops/
[36] A. Jain, Z. Kapetanovic, A. Kumar, V. N. Swamy, R. Patil, D. Vasisht, R. Sharma, M. Swaminathan, R. Chandra, A. Badam *et al.*, "Low-cost aerial imaging for small holder farmers," in *COMPASS*, 2019.
[37] W. Ma, K. Nowocin, N. Marathe, and G. H. Chen, "An interpretable produce price forecasting system for small and marginal farmers in india using collaborative filtering and adaptive nearest neighbors," in *ICTD*, 2019.
[38] A. A. Sherstov and P. Stone, "Function approximation via tile coding: Automating parameter choice," in *SARA*, 2005.
[39] M. J. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," in *ICLR*, 2016.
[40] N. M. O. Heess, G. Wayne, D. Silver, T. P. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," in *NeurIPS*, 2015.
[41] J. He, J. Chen, X. He, J. Gao, L. Li, L. Deng, and M. Ostendorf, "Deep reinforcement learning with a natural language action space," in *ACL*, 2016.
[42] H. Wang and Y. Yu, "Exploring multi-action relationship in reinforcement learning," in *PRICAI*, 2016.
[43] G. Tennenholtz and S. Mannor, "The natural language of actions," in *ICML*, 2019.
[44] A. Tacchetti, H. F. Song, P. A. M. Mediano, V. Zambaldi, J. Kramár, N. C. Rabinowitz, T. Graepel, M. Botvinick, and P. W. Battaglia, "Relational forward models for multi-agent learning," in *ICLR*, 2019.
[45] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *Arxiv:1806.01261*, 2018.
[46] A. Malysheva, T. T. Sung, C.-B. Sohn, D. Kudenko, and A. Shpilman, "Deep multi-agent reinforcement learning with relevance graphs," *Arxiv:1811.12557*, 2018.
[47] J. Jiang, C. Dun, T. Huang, and Z. Lu, "Graph convolutional reinforcement learning," in *ICLR*, 2020.
[48] H. Ryu, H. Shin, and J. Park, "Multi-agent actor-critic with hierarchical graph attention network," in *AAAI*, 2020.
[49] I.-J. Liu, R. A. Yeh, and A. G. Schwing, "Pic: permutation invariant critic for multi-agent deep reinforcement learning," in *CoRL*, 2020.
[50] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *ICML*, 2017.
[51] Y. Liu, W. Wang, Y. Hu, J. Hao, X. Chen, and Y. Gao, "Multi-agent game abstraction via graph attention neural network," in *AAAI*, 2020.
[52] E. A. Hansen, D. S. Bernstein, and S. Zilberstein, "Dynamic programming for partially observable stochastic games," in *AAAI*, 2004.
[53] Y. Song, J. Wang, T. Lukasiewicz, Z. Xu, M. Xu, Z. Ding, and L. Wu, "Arena: A general evaluation platform and building toolkit for multi-agent intelligence," in *AAAI*, 2020.
[54] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "Computational capabilities of graph neural networks," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 81–102, 2008.
[55] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks," in *ICLR*, 2016.
[56] Z. Ding, Y. Xu, W. Xu, G. Parmar, Y. Yang, M. Welling, and Z. Tu, "Guided variational autoencoder for disentanglement learning," in *CVPR*, 2020.
[57] M. Zhou, Z. Liu, P. Sui, Y. Li, and Y. Chung, "Learning implicit credit assignment for multi-agent actor-critic," in *NeurIPS*, 2020.
[58] Y.-H. Chang, T. Ho, and L. P. Kaelbling, "All learning is local: Multi-agent learning in global reward games," in *NeurIPS*, 2004.
[59] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls *et al.*, "Value-decomposition networks for cooperative multi-agent learning," *Arxiv:1706.05296*, 2017.
[60] 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 *ICML*, 2018.
[61] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," *Arxiv:1705.08926*, 2017.
[62] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, 2018.
[63] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," in *ICML*, 2019.
[64] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," in *ICLR*, 2017.
[65] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *ICLR*, 2019.
[66] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," in *ICML*, 2019.
[67] J. Wang, Z. Ren, B. Han, and C. Zhang, "Towards understanding linear value decomposition in cooperative multi-agent q-learning," *ArXiv*, 2020.



**Wenhao Li** received the B.S. degree in computer science and technology from Lanzhou University, Lanzhou, China, in 2016, and the M.S. degree in software engineering from East China Normal University, Shanghai, China, in 2019. He is currently pursuing the Ph.D. degree with East China Normal University and Tongji University, Shanghai. His main research direction is multi-agent deep reinforcement learning, computer vision, and multi-agent pathfinding.



**Xiangfeng Wang** received the B.S. degree in mathematics and applied mathematics and the Ph.D. degree in computational mathematics from Nanjing University, Nanjing, China, in 2009 and 2014, respectively. He is currently an Associate Professor with the School of Computer Science and Technology and the MOE Key Laboratory for Advanced Theory and Application in Statistics and Data Science, East China Normal University, Shanghai, China. His research interests lie in the areas of distributed optimization, multi-agent reinforcement learning and trustworthy machine learning.

**Bo Jin** received the B.S. degree in electronic engineering and the Ph.D. degree in control theory and control engineering from Shanghai Jiao Tong University, Shanghai, China, in 2004 and 2014, respectively. He is currently an Associate Professor with the School of Computer Science and Technology and the MOE Key Laboratory for Advanced Theory and Application in Statistics and Data Science, East China Normal University, Shanghai, and the Shanghai Institute of Intelligent Science and Technology, Tongji University, Shanghai. His major research interests include machine learning, online learning, reinforcement learning, computer vision, and their applications.

**Dijun Luo** received the B.S. degree in Biomedical Engineering and the M.S. degree in Control Theory and Engineering from Zhejiang University in 2004 and 2006, respectively. He received the Ph.D. degree in computer science and technology from University of Texas at Arlington in 2012. He is currently a researcher of Machine Learning Center, Tencent AI Lab. He mainly leads the company's research in the cutting-edge field of the combination of AI and agriculture, aiming to use AI technology to increase food production and solve global food shortages. His major research interests include machine learning, data mining, computer vision, and bioinformatics.

**Hongyuan Zha** received his B.S. degree in Mathematics from Fudan University in 1984, and his Ph.D. in Scientific Computing from Stanford University in 1993. He was a faculty member of College of Computing at Georgia Institute of Technology from 2006 to 2020, and the Department of Computer Science and Engineering at Pennsylvania State University from 1992 to 2006. He also worked at Inktomi Corporation from 1999 to 2001. He is currently the Presidential Chair Professor of the Chinese University of Hong Kong, Shenzhen and the Executive Dean of the School of Data Science. He has published over 300 papers in top journals and conferences in computer science and other related fields. According to Google Scholar, as of April 2021, he has been cited for over 25,100 times and his H-index is 79. Besides, he has won many prominent academic awards including Leslie Fox Prize (second prize,1991) awarded by the Institute of Mathematics and Applications (IMA), the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2011) Best Student Paper Award (as advising Professor), the 26th NeurIPS Outstanding Paper Award (2013). His current research interest lies in machine learning and its applications.