IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

Self-Organised Sequential Multi-Agent Reinforcement Learning for Closely Cooperation Tasks

Hao Fu,Mingyu You*, Hongjun Zhou and Bin He

***Abstract*—Cooperative tasks are common in multi-agent systems, with closely cooperative tasks being a special case of this, where a change in the state of the environment requires multiple agents to perform a specific operation at the same time. Take a box-pushing task as an example, the box is heavy and requires multiple agents to push it simultaneously. Optimal actions in a closely cooperation task are correlated with the actions of other agents, which makes the individual optimal action potentially inconsistent with the group optimal action, which leads to more non-globally optimal Nash equilibrium policies in the problem. This makes it easier for the policy learned by reinforcement learning to fall into these locally optimal policies. In this paper, we propose a self-organised sequential multi-agent reinforcement learning algorithm (SOS-MARL). We propose sequential decision-making to change the optimization objective of the agent's policy so that the learned policy tends to group optimal policies. And propose an automatic grouping mechanism to make the policy smoother for training and reasoning in large-scale agent environments. We decompose the joint action value factorization outside the group into a combination of each group action value, thus guiding the agents to improve their group policies in a fine-grained manner. We deployed scenarios in both simulated and real environments and compared SOS-MARL with various classical MARL algorithms on box-pushing tasks, demonstrating the state-of-the-art of our method.**

***Index Terms*—MARL, Closely Collaborative Tasks, Sequential Decision, self-Organised Group.**

I. Introduction

REINFORCEMENT Learning techniques allow agents to learn skills autonomously and show great potential in real-world scenario applications, such as gaming and robot control. However, the ability of a single agent is limited, and complex tasks usually require the cooperation of multiple agents to complete them [1]. MARL(Multi-Agent Reinforcement Learning) technology extends reinforcement learning on multi-agent systems [2]. Cooperative tasks are common in multi-agent systems, which refer to all agents having the same goal, and are widely found in life. We consider a special case in cooperative tasks where multiple agents are required to perform specific actions simultaneously, which we call closely cooperative tasks.

Taking the task of box pushing as an example, suppose that the box is heavy for the agent and that multiple agents are required to push the box simultaneously to make the box move. The actions of a single agent will not only fail to push the box but will even lead to more serious consequences, such as the box tipping over and the agent being damaged [3]. These closely cooperative tasks bring new challenges to multi-agent reinforcement learning. The locally optimal action of an agent may not be the globally optimal action. The optimal action of an agent at each moment depends not only on the state of the environment and other agents but also on the upcoming actions of other agents. For the box-pushing task, the agent's optimal action is related to the action the cooperating agents took. The policies learned by the agents are likely to fall into a local optimum, especially if the penalty for failure is large. For example, both agents eventually choose not to push the box to avoid damage.

The goal of reinforcement learning is that the agent learns the optimal action in the current state and eventually reaches a Nash equilibrium [4]. The difference between individual and group optimal actions for agents leads to more Nash equilibria. This makes it easier for the learned policy to be oriented towards a non-optimal policy. Some reinforcement learning methods attempt to promote cooperative behavior by communicating information between agents [5]. In some methods, information contains the actions the agent is expected to take. Richer observations can improve the efficiency of the agent's policy learning, trying to jump out of the locally optimal during many explorations but failing to solve the problem fundamentally. We propose to convert the parallel decision structure of a multi-agent system into a sequential decision structure. The agents take the actions sequentially, and each agent knows the actions selected by previous agents. The optimization objective of the agent is also converted to the optimal policy given the known actions of other agents. At this point, the final agent of the decision sequence can better learn the group optimal action under the known group action and gradually optimize the other agents' policies, which ultimately makes the cluster of agents know the globally optimal joint action.

However, including all agents in sequence decision-making is inappropriate, especially in large-scale multi-agent environments. The first problem brought about is the impact of efficiency. The time complexity of sequence decision-making

Manuscript received November 20, 2024; Revised January 14, 2025; Accepted March 11, 2025.

This paper was recommended for publication by Editor Aleksandra Faust upon evaluation of the Associate Editor and Reviewers' comments.

This work was supported in part by the National Natural Science Foundation of China under Grant No. 62073244 and 61825303, 62088101.

H. Fu, H. Zhou, M. You, and B. He are with the College of Electronic and Information Engineering, Shanghai Research Institute for Intelligent Autonomous Systems, Tongji University, State Key Laboratory of Intelligent Autonomous Systems, Frontiers Science Center for Intelligent Autonomous Systems, Shanghai Key Laboratory of Intelligent Autonomous Systems.

*Corresponding author: Mingyu You. (e-mail: myyou@tongji.edu.cn)

Digital Object Identifier (DOI): see top of this page.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

grows linearly with the number of agents. Furthermore, in large-scale environments, close cooperation behavior also usually only occurs between neighboring agents, and the inclusion of unrelated agents in sequential decision-making is hardly likely to enhance group cooperation positively. All these problems limit the length of the sequence.

To solve the above problems, we propose a self-organized sequential multi-agent reinforcement learning(SOS-MARL) method. Taking inspiration from natural language processing, we propose a sequential decision-making method that allows an agent to autoregressively execute actions sequentially based on environmental observations as well as the joint actions of the agents before the sequence. We propose a recursive reward decomposition method that recursively decomposes from the value of group actions to obtain the value of each agent's action. In the training process, we autonomously learn the degree of action closeness between agents and propose an automatic grouping mechanism to divide the agents that may have close cooperation into a group. Factor-weighted group action values are fused among groups to obtain the global joint action value.

We test our algorithm in multiple map-scale box-pushing scenarios with different numbers of agents and boxes. We have also deployed box-pushing scenarios in real environments to validate the effectiveness of our algorithms. Experimental results show that the policy learned by our proposed method outperforms the existing MARL algorithm regarding task completion rate. The task completion rate improved by an average of 36% compared to various currently popular MARL algorithms with different map sizes and agent numbers. Moreover, our grouping method also exhibits better generalization. Policy trained in a small range of environments can be better migrated to scenarios with a larger range and number of agents.

Our main contributions are as follows:

- We propose a MARL algorithm called SOS-MARL for closely cooperative tasks, which uses a sequential decision-making method that allows an agent to autoregressively execute actions sequentially.
- We propose an automatic grouping mechanism to divide the agents with close cooperation into the same group.
- We empirically demonstrate that our SOS-MARL outperforms other state-of-the-art methods in simulation box-pushing tasks and one real-world task.

II. Related works

The cooperation task is a classical task for multi-agent systems, and several MARL algorithms have been proposed to try to solve it [6]. CTDE(Centralized Training with Decentralized Execution) is the dominant framework for MARL algorithms. Most subsequent MARL algorithms follow this framework, such as MADDPG [7], and MAPPO [8], our method also follows that framework.

Value function factorization under the CTDE paradigm is a popular approach to MARL [9]. Most existing methods learn flat value factorization by treating agents as independent factors and directly estimating the joint action value from local utilities. The earlier work, VDN [10], learns a linear decomposition into a sum of local utilities used for greedy action selection. QMIX [11] learns a non-linear mixing network with the global state and enlarges the functions the mixing network can represent, but it still faces the monotonicity constraint. QTRAN [12] further improves the expressivity by proposing the Individual-Global-Max (IGM) principle between individual utilities and the global action value. Subsequent works follow the IGM principle and further boost performance by encouraging exploration, enhancing the expressiveness of the mixing network, preventing sub-optimal convergences, and integrating functional modules. However, under the close cooperation task, the locally optimal action of an agent is subject to changes due to the actions of other agents. This violates the IGM principle, leading to a series of decomposition methods represented by QMIX, which performs poorly on our closely cooperation task. Therefore, we propose a sequential decision-making that fundamentally solves the problem of ill-defined optimal actions.

This paper explores automatic group division for multi-agent teams to realize group-wise learning. Early related works [13] predefine specific responsibilities to each agent based on goal, visibility, capability, or search. Some other [14] efforts achieve implicit grouping by task allocation. These methods only address tasks with a clear structure and require domain knowledge or apriori settings. Another class of approaches focuses on individuality or role learning. Among them, ROMA [15] learns dynamic roles that depend on the context agents observe. RODE [16] decomposes the joint action spaces and integrates the action effects into the role policies to boost learning. A similar work, VAST [17], also studies the impact of subgroups on value factorization but still requires a priori of group number. We propose an automatic grouping mechanism that enables agents to automatically determine whether or not close cooperation will occur between agents during reinforcement learning, and use this as a basis for automatically grouping agents together.

III. Methods

We consider a multi-agent fully cooperative task where each agent has the same capabilities and targets. There are n agents $\{a_1, a_2, \cdots, a_n\}$ in the environment. We use the POMDP setting [18], the agents are unable to directly observe the global state $S = \{s_1, \cdots, s_n\}$ and can only obtain local observations $o_i \in O$ of the surrounding environment. The environment updates the state S based on the joint actions U and provides a joint reward R.

We classify agents that may closely cooperate actions into a group. The environment contains m groups and $1 \leq m \leq n$. Each agent belongs to only one group, and each group i consists of k agents $\{a_{i_1}, \cdots, a_{i_k}\}$, where $0 \leq k \leq n$. We propose a sequential decision-making method where the agent of group i selects actions $u_{i_j} \sim \pi_i\left(o_{group_i}, \left(\varnothing, u_{i_1}, \cdots, u_{i_{j-1}}\right)\right)$ k times based on the group joint observation and actions of previous agents, which eventually combine to group joint action $U_i = \{u_{i_1}, \cdots, u_{i_k}\}$. A joint action $U = \{U_i, U_2, \cdots, U_m\}$ consists of groups of joint actions.

We propose self-organized sequential multi-agent reinforcement learning(SOS-MARL) for closely cooperation tasks. As shown in Fig. 1, our approach is divided into two steps. The first

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

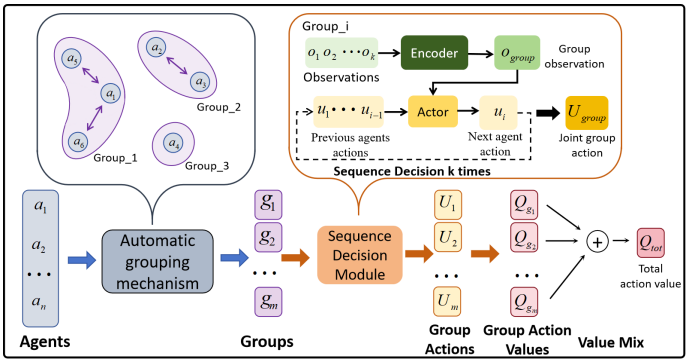


Fig. 1: The framework of our SOS-MARL. Our method is divided into two parts, automatic grouping and action decision. Firstly, the agent automatically groups based on historical observations to get m local groups. Afterward, the joint group action is obtained by autoregressive sequential decision-making within the group. The joint value is factorised outside the group to get the group action value.

step is to group the agents. We design an automatic grouping mechanism whereby the agents a_i determine the score of their influence on the state of the environment σ_i, and the score of possible closely cooperation with other agents $\omega_{i,j}$, based on observations of themselves and their surroundings. Based on these scores, the agents are grouped and ranked. The second step is to take action. According to the grouping result in the previous step, firstly, within the group, the agents select the action $u_i \sim \pi\left(o_{group}, u_{1:i-1}\right)$ according to the policy network in turn, and finally get the group joint action U_{group_i}, after that, the group action valueQ_{group_i} is evaluated and mixed to the global joint action Q_{tot} using mix net.

A. *Sequential Decision Model*

We propose a sequential reinforcement learning approach in which agents no longer make decisions in parallel but in order. After each agent's action is selected, it is not executed, but all the agents are informed after the sequence. By the time all the agents' actions have been selected, the group of agents performs joint actions in synchronization, interacting with the environment, updating their state, and being rewarded by the environment. This allows for sequential decision-making without changing the environmental settings.

The first agent a_1 acts as the leader and takes actions based solely on observation o, while the subsequent agents $\{a_2 \cdots a_k\}$ make decisions sequentially based on their observations and the actions chosen by the preceding agents. The leader's decision is crucial, as selecting the optimal action for the entire group based solely on its observation is challenging because each agent has only partial observation of the environment. Therefore, the policy network $\pi_i\left(u_i \left| o_{group}, u_{1:i-1}\right.\right)$ selects actions sequentially based on the joint observations of the entire group o_{group} and the actions of the other agents $u_{1:i-1}$. We use the transformer [19] architecture to design our autoregressively policy network.

Each agent first encodes the surrounding environment into a fixed-length observation code based on the observation coding network. An agent's observation is an 11×11 grid area centered on itself. We divide the agent's local observations into two parts, observations of the environment and observations of the operating objects. Observations of the environment contain the positions of walls and other agents in the environment relative to themselves. Observations of the objects consist of the positions of x boxes around them. We encode the $x+1$ observations into fixed-length vectors by CNN convolution, after which the vectors are fused by graph attention network(GAT) [20] based on the attention mechanism to obtain a fixed-length observation encoding.

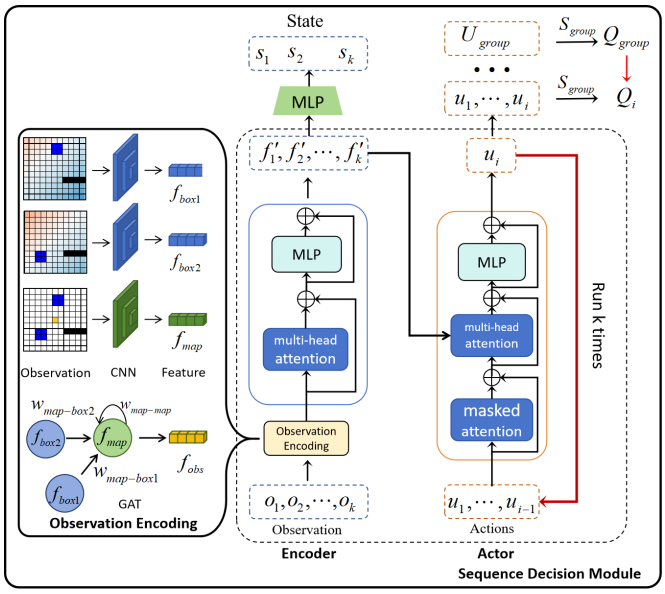


Fig. 2: Sequential decision module. Agents first encode observations of their surroundings into fixed-length vectors through an observation-encoding network and arrange them into a sequence in order within the group. Observations within the group are fused through attention. Using an autoregressive structure, each agent selects its action based on the current group's already determined actions and the group's joint observation, repeats k times until all agents have selected an action.

We use the encoder architecture of the transformer to fuse the observations around the agents; first, the encoder encodes the observations of the agents in the group, as a sequence, into the encoding network. Similar to the standard transformer, the encoder network has multiple computational modules, each of which contains the self-attention mechanism, MLP, and residual connectivity. The output fused high-level information $\{f'_1, f'_2, \cdots, f'_m\}$, which not only characterizes the information of its agent but also fuses the information of the observations within the group. We add additional supervisory information to this high-level information, and the fused observations should be closer to the state encoding.

$$L\left(\phi\right) = \frac{1}{T}\sum_{i=1}^{k}\sum_{t=0}^{T-1}\left[s_i - s'_i\right]^2 \tag{1}$$

Afterward, we use actor to generate the actions within the group from the autoregression. Each module uses a masked

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

self-attention mechanism to ensure that each agent can only obtain information such as the actions of the agent before the sequence, ensuring that the serialization is up to date. Afterward, the attention of the observation representation and the action representation was calculated, and finally, the current agent's action was output. After several autoregressions, the joint action of the group is finally output. We use clipping's PPO [21] to calculate the loss.

$$L(\theta) = -\frac{1}{Tn}\sum_{m=1}^{k}\sum_{t=0}^{T-1}\min\left(\mathrm{r}_t^m(\theta)A_t^m, \mathrm{clip}\left(\mathrm{r}_t^m(\theta), 1\pm\epsilon\right)A_t^m\right) \quad (2)$$

where $r_t^m = \pi_\theta\left(a_t^m \left| o, a_t^{1:m-1}\right.\right) / \pi_{\theta_{old}}\left(a_t^m \left| o, a_t^{1:m-1}\right.\right)$ indicates the degree of change in the policy, $A_t^m\left(o, a_{1:m}\right) = Q_t^m\left(o, a_{1:m}\right) - V_t\left(o, a_{1:m-1}\right)$ is the advantage function, and $V\left(o, a_{1:m-1}\right) = \sum_{a_m \in A} \pi\left(a_m \left| o, a_{1:m-1}\right.\right) Q\left(o, a_{1:m}\right)$ is the value function.

B. Self-Organised Groups Module

In this section, we propose an automatic grouping mechanism to classify agents that are likely to cooperate closely into the same group. In large-scale scenarios, each group contains only a limited number of agents, thus avoiding long decision sequences within the group. We supervise the learning of the grouping mechanism based on the influence of states between agents and the rewards provided by the environment.

We divide the state of the environment S into two parts, S_{agent} describing the state of the agents, including the position and velocity of the agents, and S_{env} describing the state of the environment, including the obstacles in the environment, the state of the objects to be operated and so on. In closely cooperation tasks, the actions of a single agent can usually change the state of itself, but not necessarily the state of the environment. For example, in the task of pushing a box, the movement action of an agent can change its own position, but to push the position of the box, multiple agents are needed to push it cooperatively. We collect a large number of state transfer pairs $\{S, u, S_{t+1}\}$ during reinforcement learning training. The global state S of the environment is migrated to the state of the next moment S_{t+1} under the joint actions U of the agents. At the same time, we incorporate the process of exploration in the simulation environment. For an agent a_i, assume that the actions of other agents in the environment remain unchanged, denoted as u^{-a_i}, change the actions of the agent u_i, and record the change of the state of the environment of the agent, denoted as $\{S, [u^{-a_i}, u_i], S'_{t+1}\}$.

We need to get two scores, firstly whether the current state of each agent affects the environment, called affect score σ_i. The second is whether the current agent's action will be coupled with other agents, and if so, with which ones, called related score $w_{i,j}$. The scoring networks are all composed of simple RNN and MLP networks. We use Euclidean distance to determine the difference in states $d\left(S, S'\right) = \sqrt{\left(S_i - S'_i\right)^2}$. When the amount of change in the environment state is less than a threshold when an agent tries to take a different action, we consider that the agent's current state cannot affect the environment. At the same time, based on the collected historical trajectories $\left\{s_1^i, s_2^i, \cdots, s_n^i\right\}$ of the i-th agent, an additional influence factor value, which is assigned before the historical trajectories $\sigma\left(s_k^i\right) = \gamma^{t-k}\sigma\left(s_t^i\right)$, $0 \le k < t$. We compute the related scores between the agents based on the similarity of their states when they perform counterfactual reasoning. The values obtained will be used as the ground truth of the network. All networks use MSE losses.

All the agents that satisfy the conditions $w_{i,j} > \tau$ and $d\left(s_i, s_j\right) < \upsilon$ are constructed as a group. The ordering within the group is determined by σ_i. The process of self-organizing grouping is shown in Fig. 3.

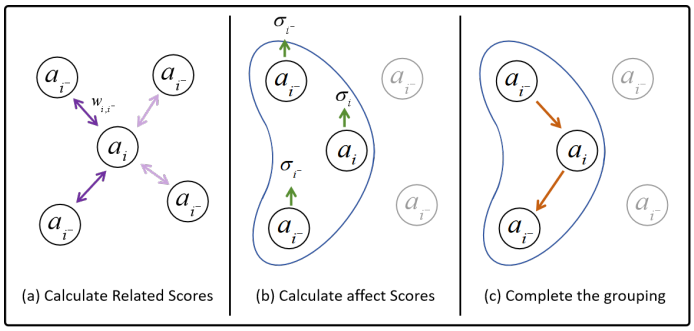


Fig. 3: Self-Organised Groups Module. (a): Calculating related scores w with neighbouring agents. (b): Calculating affect scores σ. (c): The grouping relationship is determined by w, and the group's decision order is determined by σ.

We use a mixing net to fuse the action values of each group. As shown in Fig. 1 value mix, there is no interference in actions between groups, so the basic QMIX can be used to fuse the values of actions between groups. We estimate the global action-value Q_{tot} by mixing all the Q_g. Concretely, the two layers of the total mixing network are generated by two hypernetworks, respectively taking group states S_g and the full state s as inputs. Likewise, S_g and s are deeply involved in the gradients, promoting inter-group cooperation. The architecture of the total mixing network is akin to the group mixing network and is omitted. The TD-loss of the estimated Q_{tot} is:

$$\mathcal{L}_{TD}(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{\mathbf{u}'} \bar{Q}^{tot}\left(s', \mathbf{u}'\right) - Q^{tot}(s, \mathbf{u})\right)^2\right] \quad (3)$$

Where Q_{tot} is a target network with periodic updates. The two-layer mixing structure of the group mixing network estimates the action-value Q_{tot} by w_1 which decides the group weights and w_2 which carries status information.

Our complete training process is shown in Algorithm 1. We not only optimize the policy and evaluate the network during the reinforcement learning training process. It also collects state transfer pairs during the training process. By analyzing the state transfer, we get the degree of state-action coupling between the agents, and judge the possibility of closely cooperation between the agents to train the scoring network and achieve the grouping of the agents. The group joint action values are decomposed from global joint action values using the mixing network. and the action advantage of each agent is estimated from the group action value using clipping PPO losses.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

Algorithm 1 SOS-MARL Training Process

Require: E: Environment, A: The set of agents T: Maximum episode length
Ensure: π_g: Sequential actor-net, AS: Affect scoring network, RS: related scoring network
Initiate Group critic-net C_g, sequential actor-net π_g with ϕ_g, θ_g and Group mix-net M_g with ϕ_m. Initiate scoring network A_S and R_S. Initiate the memory buffer D
repeat
  for episode $t = 1, 2, \cdots, T$ **do**
    for each agent $a_i \in A$ **do**
      get the affect score $\sigma_i = A\_S(o_i)$
      for each agent $a_j \in A$, $i \neq j$ **do**
        get the related score $w_{i,j} = R\_S(o_i, o_j)$
      end for
    end for
    Grouping the agents according to σ and w
    for each group **do**
      Autoregressive generation of joint group actions
      Predicting the value of joint group actions
    end for
    Mixing to the value of global joint actions
    $O_{t+1}, R_{t+1} \sim E(S_t, A_t)$
  end for
  $\tau^i = \{O_t, O_{t+1}, S_t, S_{t+1}, A_t, R_t, d\}_{t=1}^{T}, \tau^i \rightarrow D$
  Update the scoring network.
  Update the Critic Network
  Update the actor networks π_g
until reaching maximum training steps;

IV. Experiments

In this section, we compare our approach to many popular MARL algorithms on the closely cooperative task. We show that our method can effectively learn cooperative policy and the learned policy can be directly applied to more agent environments. We have developed simulation environments for box-pushing tasks with different map scales, numbers of agents, and numbers of boxes to validate our algorithm's effectiveness. Additionally, we have deployed a real-world box-pushing scenario using cars as agents to push large, heavy objects to specified locations.

A. Experimental Setting

We conduct experiments in both simulated and real-world environments. We set up a grid map with a certain number of boxes and agents randomly placed on it. As shown in Fig. 4(b), we use a Randomized Prim's Algorithm [22] for the simulation environment to generate the maps. The map of the real-world environment is designed manually. We use the Dijkstra [23] algorithm to calculate the distance from the goal point to all locations on the map to generate a potential energy map. The potential energy map is shown in Fig. 4(a).

The boxes occupy a 2x2 grid, while the agents occupy a single grid. The agents' task is to push all the boxes to designated goals. Boxes will be removed from the map when pushed to the goals. The environmental state S includes the position of the agents and the boxes. Each agent has five actions, up, down, left, right, and stop. Walls, boxes, or other agents obstruct agents' movements. The box will be pushed only if two agents push it simultaneously in the same direction. Agents only have limited observations about the surrounding scene and can exchange information with others within their observation range.

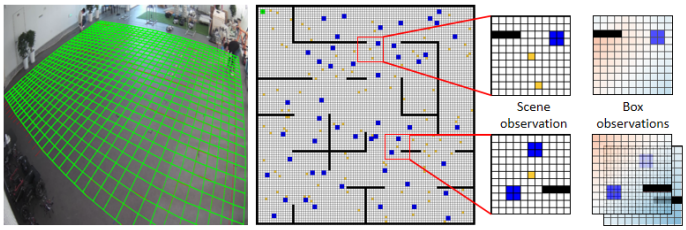


Fig. 4: Examples of experiment maps. (a):Grid maps in real environments. (b):Example of simulation environment. (c):Observation of agents.

The environment provides rewards based on changes of the boxes. The agents receive a small reward of 1 for each box that moves closer to the target point, a small penalty of -0.2 for each box that moves away from the target point, and a substantial reward of 10 for each box that reaches the goal.

B. Simulation experiments

We first validate the effectiveness of our approach in a relatively simple environment. We create a map scene of size 30x30 with several boxes and agents. We consider two kinds of application scenarios: the normal cooperation scenario and the close cooperation scenario. The difference between them is that in the normal cooperation scenario, a single agent can also push the crate and receive a reward. In contrast, in the close cooperation scenario, a single agent pushing the crate not only fails to push the crate but also gets a -0.5 penalty.

TABLE I: Comparison of closely cooperative task with normal cooperation task

	cooperation task			close cooperation task		
	2box,4agent	4box,8agent	6box,12agent	2box,4agent	4box,8agent	6box,12agent
QMIX	100%	100%	90%	90%	80%	45%
MAPPO	100%	100%	100%	100%	95%	85%
SoS-MARL	100%	100%	100%	100%	100%	100%

We validate this simple task using QMIX, MAPPO, and our SOS-MARL. The numbers in the table represent the success rate of the task which means all boxes are pushed to the specified end position. As can be seen from the table, the strategies learned by classical reinforcement learning algorithms such as QMIX and MAPPO can maintain a high success rate in ordinary cooperative tasks. In the setting of 6 boxes and 12 agents, MAPPO can still maintain a 100% success rate. However, once switching to the strong cooperation task, the policy performance of QMIX decreases rapidly, and it is difficult to learn a robust policy. The performance of the policy learned by MAPPO also appears to be degraded. In contrast, our method can learn the correct cooperation policy and complete the box-pushing task in both the normal and closely cooperation tasks.

Afterward, we compare with more reinforcement learning algorithms in more complex closely cooperative tasks. We set

up four scenarios with varying difficulty levels. We set the reward -0.5 for a single agent pushing the box. We compare this with some multi-agent reinforcement learning methods, value decomposition methods including QMIX, and MAPPO, role-based methods including ROMA and grouping methods SOG, GoMARL. All methods are based on the same observation encoding network inputs.

TABLE II: Performance in a 30x30 simulation environment

	2box,4agent		4box,10agent		10box,20agent		15box,30agent		20box,40agent	
	S-rate	R-rate	S-rate	R-rate	S-rate	R-rate	S-rate	R-rate	S-rate	R-rate
QMIX	90%	95%	80%	95.0%	20%	56.5%	0%	21.3%	0%	18.5%
MAPPO	100%	100%	95%	98.8%	45%	88.0%	10%	67.3%	0%	28.6%
ROMA	100%	100%	90%	96.3%	25%	62.0%	0%	25.7%	0%	16.3%
SOG	95%	97.5%	95%	98.8%	55%	89.5%	35%	71.7%	0%	33.8%
GoMARL	100%	100%	100%	100%	80%	91.5%	40%	79.7%	0%	35.9%
SoS-MARL	100%	100%	100%	100%	95%	99.0%	95%	98.3%	42.5%	76.1%

In the table. II, S-rate stands for success rate which means the percentage of all boxes successfully pushed to the goal and R-rate stands for reach rate which means the percentage of boxes successfully pushed to the goal. As can be seen from the table, our method gets the highest success rate regardless of the difficulty of the environment. In the 15box, 30agent environment, only our method can maintain more than 95% success rate and more than 98% reach rate. QMIX performs poorly on our task and the success rate in the 15box, 30agent environment is 0%. MAPPO are classical MARL algorithms. As the number of boxes increases, their success rate rapidly decreases. ROMA is a classical role-based multi-agent reinforcement learning algorithm that performs well in a simple environment, achieving a 100% success rate in 2box, 4agent. However, as the number of agents increases, its performance is not as good as algorithms with groups. SOG and GoMARL are classical group-based reinforcement learning algorithms, and their performance is better, maintaining a 40% success rate in the 15box, 30agent environment. In the more complex 20box, 40agent scenario, the success rate of our method also saw a significant decrease. However, at this point, other state-of-the-art MARL algorithms were unable to complete the task, with their success rate dropping to 0. Despite the decrease in success rate, our method still maintains an advantage in terms of reach rate, successfully pushing as many boxes as possible to the targets.

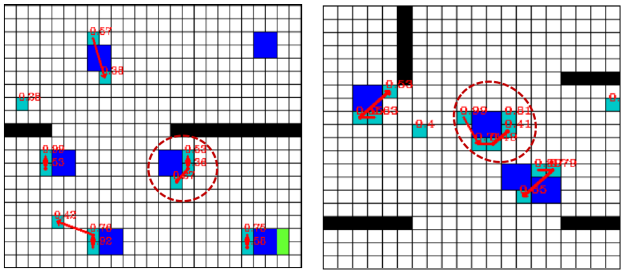
Afterward, we compare the generalizability of policies learned by different methods. We directly apply the policy to larger-scale maps. We use two different scale maps: 100x100, and 300x300. 50, 300, and 500 boxes were randomly placed in the environment with 100, 600, and 1000 agents respectively.

The results are shown in the table below. Here, we compare only the three relatively well-performing algorithms in small environments. We tested each 20 times in environments of varying difficulty. Due to the large differences in map scales, we have different maximum running step sizes T, which are 2000, 5000, and 7000, respectively.

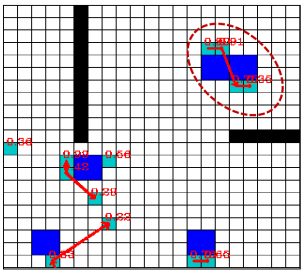
In the table. III, 100 and 300 represent the grid map size for the three environments. The generalization of our method is significantly higher than other algorithms. However, the success rate of our method is shallow at 0% in the maximum 300x300 environment. That's because the map scale is so large, and the number of agents in the scene is so high that it's easy for a large number of crates to squeeze into one place, making it difficult to handle. But from an arrival rate standpoint. It can maintain more than 98% reach rate in the 100x100 environments, and even in the most complex 500box,1000agent|300 task, it can retain more than 48.6% reach rate. In comparison, other baseline methods cannot even maintain 20%.

TABLE III: Performance of algorithms for generalisation to larger scale simulation environments

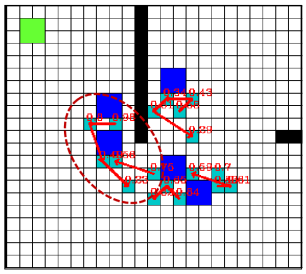
	50box,100agent|100		300box,600agent|300		500box,1000agent|300	
	S-rate	R-rate	S-rate	R-rate	S-rate	R-rate
MAPPO	10%	26.7%	0%	19.6%	0%	18.2%
ROMA	0%	21.8%	0%	17.2%	0%	12.9%
GoMARL	0%	43.8%	0%	31.5%	0%	21.8%
SoS-MARL	85%	98.2%	30%	90.1%	0%	48.6%



(a) Grouping results



(b) Under groupings



(c) Over groupings

Fig. 5: Visualization of grouping results. Agents connected by a red line represent the same group. The number next to the agent represents the affect score.

The Fig. 5 shows the results of our agent grouping. The blue 2*2 grid in the figure represents boxes, while the light blue individual grids represent agents. Agents connected by red lines are in the same group, and the arrows' direction indicates the agents' order.

As shown in Fig. 5(a), the agents that collaborate to push the same box are classified into one group because pushing the box together is a closely cooperating task requiring multiple agents' collaboration. However, our grouping still has some flaws. Fig. 5(b) demonstrates an under-grouping situation, where groups of four agents pushing different boxes are divided into one whole due to their close position. Fig. 5(c) demonstrates a case of over-grouping, where combinations pushing boxes are incorrectly divided. Such grouping is more likely to occur when there is a high density of agents. We will discuss the implications of these situations later in the ablation experiment.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

C. Grouping Methods Experiment

We validate the effectiveness of our grouping method. We consider two cases: the first Single-group is the single group, where all agents in the environment are treated as a single group. The second case is Rand-group(random group), where the output of the scoring network is set to a random number $(0-1)$ to ensure consistency in subsequent processes. We measure the differences in these approaches from two perspectives: execution efficiency and success rate.

TABLE IV: Impact of different grouping methods on performance

	5box,10agent|30		10box,20agent|30		50box,100agent|100	
	S-rate	R-rate	S-rate	R-rate	S-rate	R-rate
Single-group	100%	100%	75%	88.5%	0%	25.2%
Rand-group	90%	97.4%	90%	95.5%	65%	88.5%
Auto-group	100%	100%	95%	99.0%	85%	98.2%

Firstly, modeling all agents as a single organization does not yield satisfactory results. As the number of agents increases, the drawbacks of this approach become more pronounced. Information from agents located too far away is irrelevant for individual agents. This irrelevant information can introduce noise, interfering with the agents' decision-making processes and adversely impacting their action choices. Furthermore, this centralized organizational approach reduces execution efficiency, as the time required for agents to make decisions increases directly to the total number of agents. The second approach, which involves random grouping, improves performance in scenarios with fewer agents. However, this arbitrary grouping policy poses the risk of separating coupled agents into different groups, which can impede effective communication and coordination between them. Although our automatic grouping method is similar to random grouping in terms of execution efficiency, our method achieves a higher success rate across scenarios of different scales. This suggests that our grouping method is more effective and better suited for various environments.

TABLE V: Inference efficiency of different grouping methods

	5box,10agent	10box,20agent	50box,100agent
Single-group	19.7ms	25.3ms	89.6ms
No-group	17.3ms	17.6ms	18.1ms
Auto-group	18.0ms	19.3ms	22.5ms

Next, we tested on a machine with a 13900K CPU and RTX 3090 GPU to compare the impact of inference time on different grouping schemes.

No-group means that all agents each become a group independently, at which point there is no longer a sequenced inference structure. As can be seen from the table above, our method only improves about 21% in inference time from the benchmark N-group, this is because we divide the agents into several smaller groups and only the actor-network repeats the inference, and the inference time of our grouping method does not grow significantly with the number of agents compared to the S-group.

D. Real-world Experiments

We also validate the effectiveness of our algorithm in a real environment. We deployed the experimental scene in an area of about 10m x 10m, the grid scale was set to 30cm, and the size of the grid map was about 35x35. We placed eight cars as agents in the scene, which were built based on the differential speed structure, and the cars communicate with each other using Wi-Fi. The task setup is similar to the simulation scene, requiring multiple agents to cooperate to push the boxes in the scene to the target points.

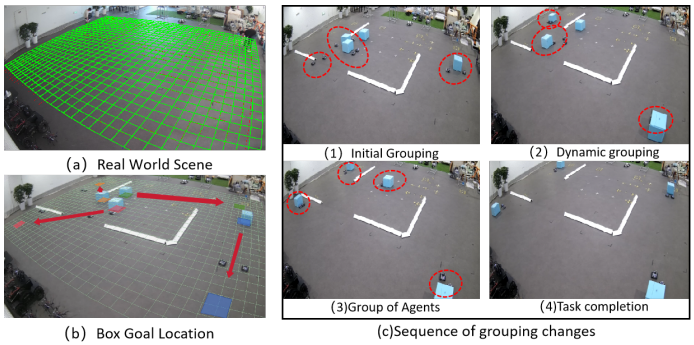


Fig. 6: Real scenario experiments. (a): Real scenario. (b): Box goal location. (c): Multi-agent decision-making process, where agents spontaneously form groups, each pushing a box. Red circles represent real-time dynamic grouping results.

To reduce the difference between the simulation environment and the real environment, we use a central system to capture the positions of all agents and boxes in the environment and provide local observation information to each car. The system only collects the state of the scene and does not participate in any decision-making. This central system mainly consists of Vicon, an infrared-based positioning device. We have placed several markers on the cars and boxes to facilitate tracking by the system. Each car obtains top-level control policies based on local observations using our policy network, determining the direction of movement. A PID controller handles lower-level motor control.

Fig. 6(a) shows our real experimental scenario, with the green lines delineating the locations of the grids. The agents circled in red are in the same group. Fig. 6(c) shows that our method can also complete the box-pushing task in the real environment. Eight carts formed themselves into four teams, each team pushing a box and finally achieving the task of pushing all the boxes to the target.

The number of agents in the real experimental scenario is small, so some reinforcement learning methods also achieve a high success rate. We compared the efficiency of task completion with the number of steps required to complete the task under the same initialization conditions, that is the same number of agents and box placement.

The numbers in the table represent the step lengths needed to complete the task. As can be seen from the table above, our method completed the task in all 5 experiments and required the lowest average steps. In experiments of run 1 and 2, the advantage of our method is not obvious when the box placement is scattered and it is easier for the agents to reach cooperative

TABLE VI. Comparison of task completion steps in real scenarios

	run-1	run-2	run-3	run-4	run-5	AVG
GoMARL	101	99	105	115	182	120.4
MAPPO	93	89	117	143	227	133.8
SoS-MARL	97	81	121	107	147	110.6

behavior. However, In the experiment of run 5, when the boxes are placed densely, the task completion efficiency of our method is higher than the other methods.

V. Conclusion

This paper introduces SOS-MARL, an auto-grouping multi-agent reinforcement learning (MARL) method for closely cooperative tasks. In this approach, agents automatically learn a grouping policy during training, effectively organizing agents with similar cooperative behavior into the same group. A sequential decision-making method is proposed to address the instability often encountered in reinforcement learning when agents engage in close cooperation. Experimental results demonstrate that SOS-MARL successfully learns appropriate grouping policies and effectively accomplishes closely cooperative tasks.

However, the current work has some limitations. The thresholds for grouping are manually selected, which can vary across scenarios of different sizes and may lead to over-grouping. Additionally, the policy struggles in scenarios involving large-scale congestion of boxes, where agents find it difficult to coordinate consistent action intentions due to excessive agent accumulation. Future work will focus on the following directions: (1) Enhancing information exchange between groups to foster cross-group collaboration at a macro level; (2) Differentiating cooperative policies among various types of agents to improve the efficiency of collaborative behaviors.

References

[1] Rafael Figueiredo Prudencio, Marcos ROA Maximo, and Esther Luna Colombini. A survey on offline reinforcement learning: Taxonomy, review, and open problems. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[2] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, pages 321–384, 2021.

[3] Hyun-Rok Lee and Taesik Lee. Improved cooperative multi-agent reinforcement learning algorithm augmented by mixing demonstrations from centralized policy. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1089–1098, 2019.

[4] Marco A Wiering and Martijn Van Otterlo. Reinforcement learning. *Adaptation, learning, and optimization*, 12(3):729, 2012.

[5] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

[6] Afshin Oroojlooy and Davood Hajinezhad. A review of cooperative multi-agent deep reinforcement learning. *Applied Intelligence*, 53(11):13677–13722, 2023.

[7] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[8] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in neural information processing systems*, 35:24611–24624, 2022.

[9] Hanhan Zhou, Tian Lan, and Vaneet Aggarwal. Value functions factorization with latent state information sharing in decentralized multi-agent policy gradients. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 7(5):1351–1361, 2023.

[10] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[11] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51, 2020.

[12] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International conference on machine learning*, pages 5887–5896. PMLR, 2019.

[13] Francisco Martinez-Gil, Miguel Lozano, and Fernando Fernández. Marl-ped: A multi-agent reinforcement learning based framework to simulate pedestrian groups. *Simulation Modelling Practice and Theory*, 47:259–275, 2014.

[14] Jianye Hao, Ho-Fung Leung, and Zhong Ming. Multiagent reinforcement social learning toward coordination in cooperative multiagent systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 9(4):1–20, 2014.

[15] Tonghan Wang, Heng Dong, Victor Lesser, and Chongjie Zhang. Roma: Multi-agent reinforcement learning with emergent roles. In *International Conference on Machine Learning*, pages 9876–9886. PMLR, 2020.

[16] T Wang, T Gupta, B Peng, A Mahajan, S Whiteson, and C Zhang. Rode: learning roles to decompose multi- agent tasks. In *Proceedings of the International Conference on Learning Representations*. OpenReview, 2021.

[17] Thomy Phan, Fabian Ritz, Lenz Belzner, Philipp Altmann, Thomas Gabor, and Claudia Linnhoff-Popien. Vast: Value function factorization with variable agent sub-teams. *Advances in neural information processing systems*, 34:24018–24032, 2021.

[18] Shen Guicheng and Wang Yang. Review on dec-pomdp model for marl algorithms. In *Smart Communications, Intelligent Algorithms and Interactive Methods: Proceedings of 4th International Conference on Wireless Communications and Applications (ICWCA 2020)*, pages 29–35. Springer, 2022.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[20] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[22] Arindam Dey and Anita Pal. Prim's algorithm for solving minimum spanning tree problem in fuzzy environment. *Annals of Fuzzy Mathematics and Informatics*, 12(3):419–430, 2016.

[23] Rolf H Möhring, Heiko Schilling, Birk Schütz, Dorothea Wagner, and Thomas Willhalm. Partitioning graphs to speedup dijkstra's algorithm. *Journal of Experimental Algorithmics (JEA)*, 11:2–8, 2007.