Neural Computing and Applications (2023) 35:19847–19863
https://doi.org/10.1007/s00521-023-08785-6

ORIGINAL ARTICLE

Expert demonstrations guide reward decomposition for multi-agent cooperation

Liu Weiwei[1,2,4] · Jing Wei[2] · Liu Shanqi[1] · Ruan Yudi[1] · Zhang Kexin[1] · Yang Jiang[3] · Liu Yong[4]

Received: 7 December 2022 / Accepted: 12 June 2023 / Published online: 10 July 2023

Abstract

Humans are able to achieve good teamwork through collaboration, since the contributions of the actions from human team members are properly understood by each individual. Therefore, reasonable credit assignment is crucial for multi-agent cooperation. Although existing work uses value decomposition algorithms to mitigate the credit assignment problem, since they decompose the global value function at multi-agents' local value function level, the overall evaluation of the value function can easily lead to approximation errors. Moreover, such strategies are vulnerable to sparse reward scenarios. In this paper, we propose to use expert demonstrations to guide the team reward decomposition at each time step, rather than value decomposition. The proposed method computes the reward ratio of each agent according to the similarity between the state-action pair of the agent and the expert demonstrations. In addition, under this setting, each agent can independently train its value function and evaluate its behavior, which makes the algorithm highly robust to team rewards. Moreover, the proposed method constrains the policy to collect data with similar distribution to the expert data during the exploration, which makes policy update more robust. We conduct extensive experiments to validate our proposed method in various MARL environments, the results show that our algorithm outperforms the state-of-the-art algorithms in most scenarios; our method is robust to various reward functions; and the trajectories by our policy is closer to that of the expert policy.

Keywords Multi-agent reinforcement learning · Expert demonstrations · Multi-agent systems · Reward decomposition · Inverse reinforcement learning

1 Introduction

Cooperative multi-agent reinforcement learning (MARL) [1] is widely used for multi-agent systems (MAS) on collaborative tasks [2, 3]. Existing cooperative MARL algorithms often use only a single team reward to evaluate the actions of the agents. However, since all agents get the same global team reward, the MARL algorithms may fail to evaluate the agents' behaviors properly, and update their policies incorrectly. Therefore, the key challenge to promote effective cooperation in MAS is correctly assigning rewards to each agent, which is known as the multi-agent credit assignment problem [4].

Existing work proposes various value decomposition methods to solve the multi-agent credit assignment problem. As shown in Fig. 1, implicit value decomposition (IVD) [5] evaluates actions of a single agent using a global evaluation network trained by global state-action pairs, which prompts the evaluation of the agents' contribution to the team. In addition, as shown in Fig. 1b, VDN [6] directly performs the explicit value decomposition (EVD) operation by a summation of the local value function, which simply maps the single-agent value function to the team

✉ Liu Yong
yongliu@iipc.zju.edu.cn

[1] Huzhou Institute of Zhejiang University, Huzhou 313002, China

[2] Department of Autonomous Driving Lab, Alibaba DAMO Academy, Hangzhou, China

[3] China Research and Development Academy of Machinery Equipment, Beijing, China

[4] The Advanced Perception on Robotics and Intelligent Learning Lab, College of Control Science and Enginneering, Zhejiang University, Hangzhou, China

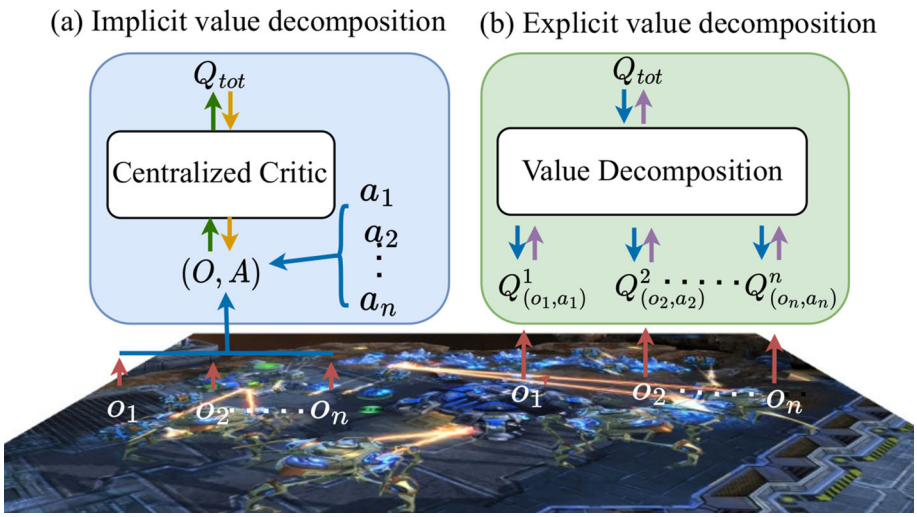


Fig. 1 Two frameworks of value decomposition algorithm

value function. VDN often fails to reflect the nonlinear contribution made by each agent to the team. Subsequently, QMIX [7] uses a neural network to nonlinearly map a global value function from the local value function of each agent. However, the global state is still required as the auxiliary input for QMIX.

Agents share the same reward function, leading them to use the same reward signal to evaluate the agents' behavior, even if they vary greatly. The original purpose of value decomposition is to correctly evaluate the agent's policy at the value function level. Although the aforementioned value decomposition methods achieve good performance on some tasks, they still suffer from several shortcomings. First, the individual agent still cannot obtain the reward signal of a single-step state-action pair with existing algorithms. In other words, the decomposed local value function only averagely evaluates the agent behavior, which may trap the algorithm in a local optimum [7, 8]. Second, EVD is difficult to train the algorithm due to its many constraints and difficult calculations [9]. Third, the policies obtained by the value decomposition algorithms do not consider the gap with the real-world data, which may lead to difficulty in generalization [9, 10].

Similar to the inverse reinforcement learning algorithm [11, 12], we explore the guiding role of the expert policy in rewards and propose a novel multi-agent credit assignment method. Precisely, the similarity between the agent and the expert policies is measured by the concept of occupancy measure [13]. The higher the similarity, the higher the reward, and vice versa. Decompose team rewards through expert demonstrations, reward each agent at each time step, let them accurately evaluate the value of their state actions, and no longer need to decompose the global value function. Our main contributions are as follows:

- First, to the best of our knowledge, this is the first time to use expert demonstrations to decompose the team reward function at each time step, and to solve the credit assignment problem from the reward function level, rather than value function level.
- Second, the algorithm enables the agent to have a high ability to explore the environment, making it robust to the reward function. Since the expert demonstration guides the reward decomposition, even when the reward is sparse or the task is difficult, the agent still performs actions more similar to the expert behavior, allowing it to explore efficiently. This reduces the difficulty of RL reward setting and performs outstandingly under challenging tasks.
- Third, the distribution of the trajectories collected by our policy is similar to that of expert demonstrations, because experts assign higher rewards to similar agent behaviors, guiding agents to imitate experts. This can guide the agent to produce realistic behavior when the expert demonstrations come from the real world. Furthermore, we also conduct extensive experiments to validate the algorithm and reach the state-of-the-art (SOTA) level.

2 Related work

It is intuitive to directly extend the single-agent algorithms [14, 15] to the multi-agent collaborative tasks for MAS. In such a setting, each agent is controlled by a neural network model and collects samples to train the model. Although these algorithms perform well in some environments [16] for fully cooperative tasks [17], they do not perform value decomposition [6] operations and rely only on team rewards to update the policies, which may lead to credit

assignment problems [18]. Additionally, these methods do not consider the mutual interference among the actions of each agent (non-stationarity of the multi-agent environment), so the policies trained in this setting are difficult to converge in some scenarios [19].

Many MARL algorithms have been proposed to alleviate the non-stationarity of the multi-agent environment and solve the credit assignment problem. We can classify existing MARL methods into two categories according to their approach to training the agents' local action-value function: explicit and implicit value decomposition (EVD and IVD). It is worth noting that these methods use centralized training [20] to address the non-stationarity problem in multi-agent environments.

2.1 Explicit value decomposition

For the credit assignment problem, EVD decomposes the joint value function of the multi-agent system into a specific combination of individual action-value functions through certain constraints. VDN [6] centrally trains the joint value function, which is equal to the sum of the local value functions of each agent, realizes value decomposition, and alleviates the multi-agent credit assignment problem. However, the simple summation limits the local value function's ability to evaluate the state-action pros and cons of the agent. It is also unable to represent complicated relationships between heterogeneous agents. QMIX [7] extends this simple summation to a nonlinear monotonic functions. However, it is still challenging to accurately approximate the global value function directly according to the local value function. MAVEN [21] introduces a latent space of hierarchical control, and the shared latent variable controls the action of each agent. This variable is controlled by a hierarchical policy, which improves the exploration ability of the algorithm. QTRAN [22] uses the summation method to obtain the approximate value of the joint value function and then fits the difference between the approximate value and the actual joint value function.

2.2 Implicit value decomposition

IVD centrally trains the global value network corresponding to each agent and evaluates their state action. Since the global value function of each agent only evaluates its policy, at this time, the global information belongs to the auxiliary input of the value network to help the agent to act based on the global situation. We think that this global action implicitly acquires its local value function, enabling value decomposition. The existing methods are as follows: MADDPG [5] is extended use of DDPG [23] in the multi-agent field. It focuses on training a joint evaluation network and does not explicitly decompose team rewards, and each agent has its independent policy network. The COMA [24] framework is the same as MADDPG, but COMA uses the GRU [25] network to better deal with the problem that the agent only has a partial view. Its core lies in the introduction of a counterfactual baseline function. The problem of multi-agent credit allocation is solved by comparing the team rewards obtained by the agent's current policy and the default policy decision-making. Similar to MADDPG, MAPPO [26] extends PPO [27] algorithm to a multi-agent setting and achieves reliable performance.

Different from the value function decomposition method, we are the first to propose using expert samples to guide the decomposition of the team reward function. The algorithm improves the exploration ability of the agent and increases the robustness to different reward functions. Its trained policy generates state-action pairs that are closer to expert behavior.

3 Background

3.1 Preliminaries

In this paper, we consider a decentralized partially observable Markov decision process (dec-POMDP) [28] with shared rewards extended to the field of multi-agents. Dec-POMDP is described by a tuple $\{\mathcal{S}, \mathcal{A}, \gamma, \mathcal{P}, r\}$, where $\mathcal{S} = \{S^1, \ldots, S^N\}$ and $\mathcal{A} = \{A^1, \ldots, A^N\}$ denote the space of the joint states and joint actions of N agents, respectively. $S^i \in S$ and $A^i \in A$ represent the state and action space of agent $i \in N$, respectively. The agent we studied has only partial observation ability of the environment, $S \times N \rightarrow p(S)$ represents the probability distribution of the joint state. $\gamma \in (0, 1)$ is the discount factor, which indicates the degree of influence of future rewards on the agent's current action choices. At each time step, agent i selects action a_i based on the current state s_i, and his policy is $\pi_i : S_i \times A_i \rightarrow [0, 1]$. $\mathcal{P}(S' \mid S, A)$ is the transition probability of taking actions $\{a^1, \ldots, a^N\} \in A$ in state $\{s_1, \ldots, s_N\} \in S$ to reach next state $\{s'_1, \ldots, s'_N\} \in S'$. All agents share a reward function $r(\vec{s}, \vec{a}) : S \times A \rightarrow \mathbb{R}$, that is, all agents get the same reward at each time step. All agents cooperate in order to obtain the greatest team return $J = \mathbb{E}_{a_1 \sim \pi_1, \ldots, a_n \sim \pi_n, S \sim \mathcal{P}} \Sigma_{t=0}^{T} \gamma^t r_t(\vec{s}, \vec{a})$, where T is the time horizon, this is used to evaluate the pros and cons of joint policies. Similarly, the joint state value function of all agents is $V_\pi(\vec{s}) = \mathbb{E}_\pi[r(\cdot, \cdot) \mid \vec{s}_0 = \vec{s}]$, and joint state-action value function is $Q_\pi(\vec{s}, \vec{a}) = \mathbb{E}_\pi[r(\cdot, \cdot) \mid \vec{s}_0 = \vec{s}, \vec{a}_0 = \vec{a}]$.

3.2 Inverse reinforcement learning

The objective of reinforcement learning [29] is to maximize the cumulative reward. In contrast, the purpose of inverse reinforcement learning (IRL) [11, 12] is to learn a reward function that motivates the agent to generate such a trajectory through a known behavioral trajectory. Therefore, existing work [30, 31] often combines the two. In areas where it is difficult to quantify the reward function, such as autonomous driving [32], learning the reward function behind the expert's behavior is conducive to algorithm convergence and makes the agent behave like an expert.

Feature matching [33] is the most common IRL algorithm, which directly extracts features from expert trajectories, measures the feature loss of the trajectory generated by the current policy and the known expert trajectories to update the reward function [34, 35]. The disadvantage of feature matching is that there may be multiple different but reasonable reward functions for the same expert trajectory, and the two cannot be matched [36]. Maximum entropy (MaxEnt) IRL [37] employs the maximum entropy principle [38] to eliminate this matching ambiguity, which is based on the assumption that the policy of expert trajectories to generate their feature expectations is the optimal trajectory. The maximum causal entropy IRL [39] is to propose the causal policy model, which considers that the choice of action is related to all previous actions and states. Due to the complexity of traditional IRL implementation, GAIL [40] uses the GAN [41] idea to use the data distribution generated by the current policy and the existing expert sample distribution as a loss, and directly learn the policy from the expert trajectory. MAGAIL [42] is an extension of GAIL in multi-agent.

3.3 Occupancy measure

[13] proposed to measure the distribution of state-action pairs in the agent's exploration trajectory when executing policy π [13, 43]. This is the key to the following analysis.

Definition 3.1 (Occupancy measure) An agent is given a stable policy π, let $\rho_\pi(s) : \mathcal{S} \to \mathbb{R}$ and $\rho_\pi(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denote the density of the state distribution and the state-action joint distribution of the agent under the policy π.

$$\begin{aligned} \rho_\pi(s) &\triangleq \sum_{t=0}^{\infty} \gamma^t P(s_t = s \mid \pi), \\ \rho_\pi(s,a) &\triangleq \rho_\pi(s)\pi(s,a). \end{aligned} \tag{1}$$

$\rho_\pi(s,a)$ is the distribution of state-action pairs, also called the occupancy measure of policy π. There is a one-to-one correspondence between $\pi \in \prod$ and $\rho \in \mathcal{D}$.

4 Methods

Humans can accomplish challenging collaborative tasks through teamwork. First, each player on the team will independently determine the expected reward for acting in the current state at each moment. For example, the reward for scoring a goal off the dribble is more remarkable when a goal is not scored. However, in fully cooperative tasks, since the agents share the same reward function, which is not conducive to the agent evaluating its behavior. To this end, each agent should be assigned an appropriate reward according to the state and behavior at each time step. Second, taking autonomous driving as an example, the behavior of the agent trained in the simulation does not necessarily conform to real driving habits, which leads to generalization problems when the simulation algorithm is deployed in the real world. Therefore, agents in simulation should behave similarly to real experts to facilitate the gap between virtual and real.

This work uses expert demonstrations to decompose the reward at each time step. Similar to GAIL [40] and MAGAIL [42], our algorithm measures the similarity between the agent and expert strategies by occupancy measure and learns reward decomposition strategies. As shown in Fig. 2, we train the discriminator using expert demonstrations and RL buffer experience. Specifically, when using an expert state-action pair as input, the discriminator outputs the square of the difference between the probability and 1 as a loss function. When using the agent state-action pair as input, the discriminator outputs the square of the difference between the probability and 0 as the loss function. These two loss functions are used to train the discriminator to distinguish between expert and agent strategies. Finally, with each agent state-action pair as input, the discriminator output is used as the basis for reward distribution. That is, the higher the discriminator output score, the closer the agent behaves to the expert, and the higher the reward it gets. Form adversarial training as a whole. Notably, if the agent is isomorphic, there is only one discriminator. Otherwise, each proxy corresponds to a discriminator. In general, our algorithm provides each agent with a reward signal for each step and guides the agent to behave more like an expert.

4.1 Decompose the rewards of each time step

Assumption 1 The global reward of a multi-agent system is equal to the sum of the rewards received by each agent.

$$r(\mathbf{s}_t, \mathbf{a}_t) = \sum_{i=0}^{N} r_i(o_t^i, a_t^i), \tag{2}$$

Among them, $\mathbf{s}_t$ denotes the global agent state, $\mathbf{a}_t$ denotes the joint actions of all agents, and o_t^i and a_t^i denote the state

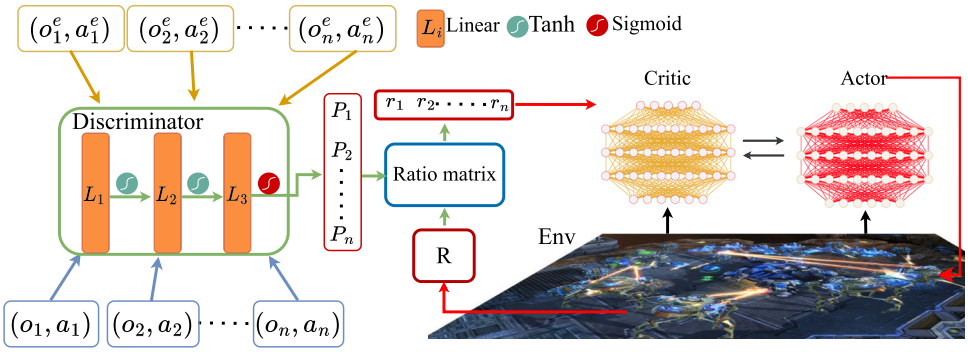


Fig. 2 Expert demonstrations guide reward decomposition framework diagram. On the left is the training of discriminator, and on the right is the use of the discriminator to decompose the global reward. In the figure, o_i^e and a_i^e, respectively, represent the observation and action of the expert agent i. R represents the global shared reward. $r_1, \cdots, r_n$ represents the respective reward of each agent

and actions of agent i at time t. To simplify the derivation process, the subscript time t of the state and action is omitted below.

First of all, we can know from Assumption 1:

$$\begin{cases} r_i(o_i, a_i) = \alpha_i R(\mathbf{s}, \mathbf{a}) \\ \sum_i^n \alpha_i = 1 \end{cases} \tag{3}$$

Here R is the team reward of all agents in a time step, r_i is the reward of agent i, and α_i is the team reward coefficient of agent i, representing the contribution to teamwork, α is a function of the agent, global state-action pairs, $\alpha = \alpha(o_i, a_i; \mathbf{s}, \mathbf{a})$. So far, we only need to obtain the team reward coefficient of each agent to decompose the team reward.

We know that the objective of inverse reinforcement learning is to learn a reward function that motivates the agent to generate such a trajectory based on the known expert trajectory. Conversely, if the two policies are the same, then they get the same reward. From the definition of occupancy, we know that if the agent's policy is consistent with the expert's policy, the occupancy of its policy should also be consistent. So, we get

$$\mathbb{E}[V^{\pi}(s_0)] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid \pi\right] \approx \mathbb{E}[V^{\pi_E}(s_0)], \tag{4}$$

and

$$\frac{R'(\mathbf{s}_t, \mathbf{a}_t)}{R_E(\mathbf{s}_t, \mathbf{a}_t)} = \zeta_{\text{tot}}\left(\frac{\rho_\pi}{\rho_{\pi_E}}\right). \tag{5}$$

Similarly, for agent i:

$$\frac{r_i'(s_t, a_t)}{r_i^e(s_t, a_t)} = \zeta_i\left(\frac{\rho_{\pi_i}}{\rho_{\pi_i^e}}\right), \tag{6}$$

where ζ_{tot} and ζ_i are monotonically increasing functions. The occupancy can only be estimated from expert demonstrations. $R'(\mathbf{s}_t, \mathbf{a}_t)$ and $r_i'(s_t, a_t)$ represent the estimated team reward and the reward of agent i. Because the reward of the expert demonstrations at each moment is determined, the reward of the agent at each moment is only related to the similarity of occupancy.

$$\begin{cases} r_i' = \xi_i\left(\frac{\rho_{\pi_i}}{\rho_{\pi_i^e}}\right) \\ R' = \xi_{\text{tot}}\left(\frac{\rho_\pi}{\rho_{\pi_E}}\right) \end{cases} \tag{7}$$

Since $\frac{r_i'}{R'} = \frac{r_i}{R}$, and from equation (7), we can get

$$r_i = \frac{\xi_i\left(\frac{\rho_{\pi_i}}{\rho_{\pi_i^e}}\right)}{\xi_{\text{tot}}\left(\frac{\rho_\pi}{\rho_{\pi_E}}\right)} R = \alpha_i R, \tag{8}$$

and the coefficient meet this condition:

$$\sum_i^n \frac{\xi_i\left(\frac{\rho_{\pi_i}}{\rho_{\pi_i^e}}\right)}{\xi_{\text{tot}}\left(\frac{\rho_\pi}{\rho_{\pi_E}}\right)} = 1. \tag{9}$$

4.2 Reward distribution coefficient

First, it should be noted that we do not directly use $\zeta_i\left(\frac{\rho_{\pi_i}}{\rho_{\pi_i^e}}\right) r_i^e(s_t)$ as the reward of agent i. Because the cost of obtaining expert demonstrations is high, it is challenging to design appropriate expert rewards in practice. So, we use very few expert demonstrations, which do not include expert rewards. Second, from equation (3), we know that in a multi-agent system, the reward of each agent is constrained by the team reward, so we use equation (8) to decompose the reward. So far, to decompose the reward at each time step, we only need to obtain the similarity

between the agent's and the expert's policies. Here, we introduce maximum causal entropy IRL (MCEIRL) [39], GAIL [40], and MAGAIL [42] algorithms to solve this problem. Among them, MAGAIL is an extension of GAIL in the field of multi-agents. GAIL first proposes to train discriminators to measure the similarity between agents and expert strategies.

Maximum causal entropy IRL looks for a cost function $c \in \mathcal{C}$ that assigns low costs to expert policies and high costs to other policies.

$$\text{maximize}(\min -\mathrm{H}(\pi)+\mathrm{E}_{\pi}[\mathrm{c}(\mathrm{s},\mathrm{a})])-\mathrm{E}_{\pi_{\mathrm{E}}}[\mathrm{c}(\mathrm{s},\mathrm{a})] \tag{10}$$

Reinforcement learning, on the other hand, maps the learned cost function to a high-entropy policy that minimizes the expected cumulative cost, enabling the imitation of experts.

$$RL(c)=\text{argmin}-\mathrm{H}(\pi)+\mathrm{E}_{\pi}[\mathrm{c}(\mathrm{s},\mathrm{a})] \tag{11}$$

$H(\pi)\triangleq E_{\pi}[-\log\pi(\mathrm{a}\mid \mathrm{s})]$ is a γ discounted cumulative causal entropy [39], it is used to prevent the algorithm from overfitting. According to the definition of occupancy, the expected cost function $E_{\pi}[c(s,a)]$ can be written as follows:

$$E_{\pi}[c(s,a)]=\sum_{s,\mathrm{a}}\rho_{\pi}(s,a)c(s,a) \tag{12}$$

Define an IRL primitive process that finds a cost function such that the expert cost is lower than all other policies, the cost function is regularized by ψ:

$$\begin{aligned} IRL_{\psi}(\pi_E)=&\,\text{argmax}-\psi(\mathrm{c})+(\min-\mathrm{H}(\pi)\\ &+E_{\pi}[c(s,a)])-E_{\pi_E}[c(s,a)]\end{aligned} \tag{13}$$

For each agent, the imitation of an expert can be defined as an occupancy matching problem. That is, the state-action distribution of the learned policy is consistent with the expert.

$$RL\circ IRL_{\psi}(\pi_E)=\text{argmin}_{\pi\in\prod}-\mathrm{H}(\pi)+\psi^{*}(\rho_{\pi}-\rho_{\pi_E}), \tag{14}$$

The above equation can be interpreted as a measure of the similarity of the occupancy between the single-agent's policy and the expert's policy. The optimal negative log loss for this binary classification problem is as follows:

$$\psi^{*}=\max_{D\in(0,1)^{S\times\mathcal{A}}}\mathbb{E}_{\pi_E}[\log(D(s,a))]+\mathbb{E}_{\pi}[\log(1-D(s,a))], \tag{15}$$

The role of this network is to act as a discriminator to identify whether it is an expert or an agent. On the other hand, the agent's policy uses $log(D(s, a))$ as the reward function, which makes the discriminator mistakenly consider it as an expert demonstration. At this point, for agent i, we have

$$r_i=\xi_i\left(\frac{\rho_{\pi_i}}{\rho_{\pi_i^e}}\right)=\log(D_i(s_i,a_i)). \tag{16}$$

Similarly, for multi-agents [42], assume that $\psi(R)=\Sigma_{i=1}^{n}\psi_i(r_i)$, then

$$MARL\circ MAIRL_{\psi}(\pi_E)=arg\min_{\pi\in\Pi}\sum_{i=1}^{n}\psi_i^{*}(\rho_{\pi_{i,E_{-i}}}-\rho_{\pi_E}) \tag{17}$$

where $\pi_{i,E_{-1}}$ denotes π_i for agent i and $\pi_{E_{-i}}$ for other agents. For agent i, the discriminator needs to be trained by expert demonstrations and the agents' sample data, the discriminator map it to 1 and 0 to distinguish between the two. Instead, we use $\Sigma_{i=1}^{n}\log D_{\omega_i}(\mathbf{s},a_i)$ as the reward function of the agents, which, in turn, helps the agents fool the discriminator. We optimize the following goals:

$$\begin{aligned}\mathcal{L}_{MD}=&\,\theta\,min\,\omega\,max\left[\mathbb{E}_{\pi_\theta}\left[\sum_{i=1}^{n}\log\mathrm{D}_{\omega_i}(\mathbf{s},\mathrm{a_i})\right]\right.\\ &\left.+\mathbb{E}_{\pi_E}\left[\sum_{i=1}^{n}\log(1-\mathrm{D}_{\omega_i}(\mathbf{s},\mathrm{a_i}))\right]\right]\end{aligned} \tag{18}$$

For this multi-agent system, we have a reward function

$$R=\xi_{tot}\left(\frac{\rho_{\pi}}{\rho_{\pi_E}}\right)=\sum_{i=1}^{n}\log(\mathrm{D_i}(\mathbf{s},\mathrm{a_i})). \tag{19}$$

As a result, the joint reward for each time step can be decomposed as follows:

$$r_i=\frac{\log(\mathrm{D_i}(\mathrm{s},\mathrm{a_i}))}{\Sigma_{i=1}^{n}\log(\mathrm{D_i}(\mathrm{s},\mathrm{a_i}))}R \tag{20}$$

The above equation expresses that the contribution of the agent to team and the distribution of the reward would increase, when the state action of the agent is close to the expert's policy. Note that after decomposing the team reward, we use the PPO [27] algorithm to train the policy for each agent, and the objective function is as follows:

$$\begin{aligned}J_{\theta_k}(\theta)\approx&\,\Sigma_{(s_t,a_t)}\min\left(\frac{p_\theta(a_t,s_t)}{p_{\theta_k}(a_t,s_t)}A_{\theta_k}(a_t\mid s_t),\right.\\ &\left.cilp\left(\frac{p_\theta(a_t,s_t)}{p_{\theta_k}(a_t,s_t)},1-\varepsilon,1+\varepsilon\right)A_{\theta_k}(a_t\mid s_t)\right)\end{aligned} \tag{21}$$

here:

$$\begin{aligned}A(s_t,a_t)=&\,Q(s_t,a_t)-V(s_t)\\ =&\,E_{s_{t+1}\backsim p(s_{t+1}|s_t,a_t)}[r(s_t)+\gamma V^{\pi}(s_{t+1})-V^{\pi}(s_t)]\end{aligned} \tag{22}$$

5 Experiment

5.1 Experimental environments and settings

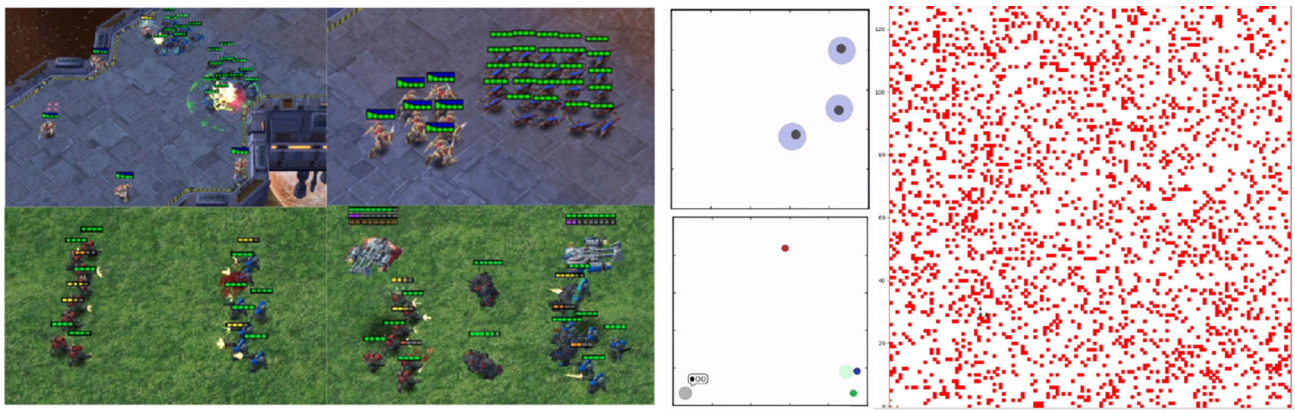
In this section, we benchmark our method with other multi-agent reinforcement learning algorithms on the Starcraft Micromanagement Challenge [44] (SMAC), the Multi-agent Particle-world Environment [5] (MPE), and the Grid World [45]. The experimental scene is shown in Fig. 3. The baseline algorithms include VDN [6], QMIX [7], COMA [24], QTRAN [22], MAVEN [21], MAVEN+BC [46], MAPPO [26], MAGAIL [42], IPPO [47], RODE [48], and MADDPG [5]. For fair comparison, we run all algorithms on the same computer, with an Intel i7-8700K CPU, 16-GB RAM, and an NVIDIA GTX1080Ti GPU. For the hyperparameters: The learning rate is 1e-4, the discount factor γ is 0.99.

We first conduct experiments in two public multi-agent environments (Starcraft II SMAC and MPE). These are designed with many small maps, and agents can cooperate fully or compete with each other. Finally, we conducted experiments in the grid world. It is a formation walking environment. The agent is required maintain the formation as much as possible, bypass obstacles, and reach the destination. Since we can entirely manually design the rewards for this environment, we use it to verify the algorithm's robustness to the handcrafted rewards. It should be noted that each expert demonstration only includes state-action pairs of all agents in one step, without their rewards. The number of expert demonstrations used in different experimental scenarios is shown in Table 1. These demonstrations are used by MAVEN+BC, MAGAIL, and our proposed algorithm. The algorithms used to collect expert demonstrations in different environments are different. In StarCraft and MPE environments, we use the converged MAPPO to collect expert demonstrations. We also add filtering operations to discard demonstrations with low-performing behaviors. In grid world environment, enhanced collision-based search algorithm (ECBS) [49] is used to collect expert demonstrations.

5.2 Results in SMAC environment

We benchmark our algorithm with several existing algorithms such as VDN, QMIX, COMA, MAVEN, QTRAN, MAVEN+BC, and RODE, in SMAC environment. Table 2 compares the results when all the algorithms converge, while five random seeds are used for each algorithm. Under each random seed, all algorithms except RODE are evaluated 32 times after convergence to calculate their average winning rate. The resultof RODE is directly obtained from the paper [48]. First, Table 2 shows our algorithm outperforms other methods in most scenarios, i.e., achieving the highest winning rate in 20 of the 23 maps. In addition, in the two maps "so many baneling, 25 m," the gap between our algorithm and the others with the highest winning rate is very small. In our algorithm's worst-performing scenario, "5 m versus 6 m" map, the winning rate is higher than the baseline algorithms except RODE.

Furthermore, as the difficulty of maps increases, the success rate of baseline algorithms, including RODE, declines. Because, at this time, the set team reward is difficult to guide the agent to conduct effective exploration, resulting in poor performance of the baseline algorithm. Nevertheless, our algorithm is less affected since the expert demonstrations guide the team reward is appropriately decomposed. Even if the team reward is invalid, our algorithm encourages the agent to take actions similar to the expert demonstrations through appropriate reward



(a) Starcraft II (b) MPE (c) Grid World

Fig. 3 Three experimental environments. **a** so many banes, corridor, 8 M and MMM in Starcraft II; **b** spread and communication in MPE; and **c** a map of size 128 with an obstacle density of 0.15 in grid world, where red squares represent obstacle

Table 1 The number of expert demonstrations used in different experimental scenarios

MPE					
Communication	1000	Spread	1000		
Starcraft II					
1c3s5z	862	2c versus 64zg	877	2 m versus 1z	1136
2s3z	960	Bane versus bane	1020	25 m	1097
3 s versus 3z	1109	3 s versus 4z	1623	3 s versus 5z	1738
5 m versus 6 m	845	6 h versus 8z	840	8 m	1044
8 m versus 9 m	701	3 m	804	MMM	1288
2 s versus 1sc	825	corridor	875	MMM2	1064
So many baneling	1127	3s5z	1581	3s5z versus 3s6z	1031
27 m versus 30 m	1924	10 m versus 11 m	937		
Grid world					
32-0.05	64	32-0.15	64	64-0.05	128
64-0.15	128	128-0.05	256	128-0.15	256
256-0.05	512	256-0.15	512	512-0.05	1024
512-0.15	1024				

Table 2 The percentage of victory achieved by different algorithms in 23 scenarios in the Starcraft II

Maps	Difficult	VDN	QMIX	COMA	QTRAN	MAVEN	MAVEN+BC	RODE	Ours
3 m	Easy	79.38%	**100%**	3.75%	**100%**	**100%**	**100%**	–	**100%**
2s3z	Easy	32.5%	88.75%	0.0%	98.75%	94.38%	98.13%	**100%**	**100%**
3 s versus 3z	Easy	56.25%	**100%**	0.0%	**100%**	**100%**	**100%**	–	**100%**
MMM	Easy	51.25%	98.13%	58.16%	93.13%	99.38%	89.37%	–	**100%**
3 s versus 4z	Easy	0.0%	83.75%	0.0%	99.38%	**100%**	96.25%	–	**100%**
2 m versus 1z	Easy	10.63%	**100%**	1.25%	**100%**	**100%**	**100%**	–	**100%**
Bane versus bane	Easy	64.36%	77.5%	75%	96.25%	88.13%	98.75%	**100%**	**100%**
1c3s5z	Easy	5.63%	20.64%	59.38%	94.33%	96.88%	96.88%	**100%**	**100%**
So many baneling	Easy	25.94%	99.38%	42.81%	**100%**	95.94%	92.5%	–	99.75%
8 m	Easy	93.13%	98.75%	1.25%	**100%**	98.13%	99.38%	–	**100%**
2 s versus 1sc	Easy	0.63%	61.25%	1.25%	**100%**	91.88%	**100%**	**100%**	**100%**
5 m versus 6 m	Hard	55%	63.75%	1.88%	44.38%	64.38%	67.5%	71.6%	68.75%
10 m versus 11 m	Hard	0.0%	3.44%	2.19%	5.63%	7.5%	83.75%	95.6%	**98.75%**
2c versus 64zg	Hard	1.88%	70.63%	95.63%	93.75%	86.88%	86.25%	96%	**96.88%**
3s5z	Hard	66.25%	74.37%	17.5%	41.87%	38.12%	16.88%	93.75%	**99.38%**
3 s versus 5z	Hard	0.0%	56.25%	0.0%	96.88%	98.13%	72.5%	78.9%	**100%**
25 m	Hard	87.96%	93.6%	8.91%	96.87%	92.19%	0.0%	–	96.88%
8 m versus 9 m	Hard	19.38%	23.13%	10.0%	85.0%	76.88%	41.88%	–	**93.75%**
Corridor	Super hard	0.0%	80%	0.0%	71.88%	0.0%	87.5%	90.6%	**100%**
3s5z versus 3s6z	Super hard	11.88%	79.13%	0.0%	0.0%	0.0%	0.0%	84.25%	**100%**
6 h versus 8z	Super hard	0.0%	3.13%	0.0%	11.25%	0.63%	8.13%	59.6%	**100%**
MMM2	Super hard	41.25%	27.5%	0.63%	14.38%	5.63%	46.25%	86.45%	**96.88%**
27 m versus 30 m	Super hard	10.62%	8.75%	0.63%	9.37%	1.25%	0.0%	96.05%	**100%**

Highlight the highest-performing sections. For example the highest success rate, the smallest loss, the shortest time, etc

decomposition and guides the agent to take the correct action and conduct effective exploration. Compared with the MAVEN+BC algorithm, although MAVEN+BC is beneficial to solve the cold start problem of the algorithm because the action clone executes the sequence of behavior, this also creates a cumulative error. Finally, the difference between the agent's behavior and the optimal

behavior becomes larger and larger. However, our algorithm decomposes the reward through the similarity with the expert demonstrations to obtain the corresponding value function to evaluate the policy, avoiding the accumulation of errors caused by direct behavioral cloning.

Finally, Table 2 and Fig. 4a–g show that, compared with such value decomposition algorithms as VDN and QMIX, our algorithm performance is minimally affected by the number of agents. This also shows that the expert demonstrations-guided reward decomposition helps improve the algorithm's robustness to the number of agents. However, the convergence difficulty of the traditional decomposition algorithm, such as VDN and QMIX, is positively related to the number of agents. Figure 4h–o compares the rewards obtained by different algorithms and then reflects the convergence speed of the algorithms. It can be seen from the figure that even if we are an online algorithm, its convergence speed is still highly competitive. Figure 5 further shows the relationship between the convergence degree of the algorithm and the time. Compared with the baseline algorithm, our algorithm takes less time to reach convergence.

5.3 Results in MPE

We test algorithms on two cooperative tasks in the MPE environment, including the cooperative communication and the cooperative navigation (spread) task, and comparison algorithms include MADDPG, MAPPO, and MAGAIL. It should be noted that the observation of each agent includes the positions of other agents and obstacles, and this observation is essentially equivalent to the global state.

For detailed descriptions of tasks in different scenarios in the MPE environment.[1]

Figure 6 shows the experimental results averaged with five random seeds. The results comparison indicates that our method has achieved the SOTA level performance in all scenarios. Moreover, even though our method is an online RL algorithm, the convergence ratio in the communication tasks is faster than the off-policy MADDPG. In addition, our method outperforms MAGAIL in both convergence ratio and total rewards, because estimating rewards directly from MAGAIL with minimal demonstrations is challenging. The performance of our method is similar to MAPPO in MPE environment; however, we show in later section that our algorithm is more robust to different reward design compared to MAPPO.

[1] MPE:https://github.com/openai/multiagent-particle-envs.

5.4 Results in grid world

Grid world simulation[2] is used for additional benchmark. The grid world environment can set arbitrary map size and obstacle density. In this paper, we use map size $\{32, 64, 128, 256, 512\}$ and the obstacle density $\{0.05, 0.15\}$ (The map is shown in Fig. 3c). In this simulation, the field of view for the agents is 9 grids, which formulates a partially observable Markov decision process. The starting point of each map is at the upper right, and the goal point is at the lower left. Three agents must avoid obstacles while maintaining the formation as much as possible to reach the destination. During training, 100 maps of each type will be randomly generated to prevent the algorithm from overfitting. In order to verify the algorithm's robustness to the reward, we designed multiple reward functions and compared the algorithm's performance in the grid world.

First, as shown in Table 3, our algorithm has achieved the best performance in most cases. It is noted that the performance of the proposed method is comparable to the baseline algorithm, only when the tasks are relatively simple, especially when the map is smaller with fewer obstacles, and the path that the agent needs to detect is short. For most of the difficult tasks, our algorithm significantly outperforms the baseline algorithms, this validates the effectiveness of our proposed method using the guidance from expert data to train the agent.

Moreover, Fig. 7 indicates that our algorithm is robust to variation of reward functions. We vary the reward function with increasing the penalty for formation loss. It is observed that the reward and loss of MAPPO still converge reasonably fast. However, the agents trained by MAPPO stop walking to avoid losing formation and failing to reach their destination. In contrast, our algorithm could maintain the formation to reach the destination, with little performance variation. This is because the single-step reward is decomposed under the guidance of experts. Even if each agent cannot obtain an accurate reward signal, the reward ratio of each agent can still prompt the agent to act similarly to the expert.

5.5 Analysis on data distribution and expert data

The experimental results in Fig. 8 show the distribution difference between state-action pairs collected by different algorithms and that from expert data. It is observed that the state-action pairs produced by our algorithm are closer to

[2] Grid world for planning: https://github.com/whoenig/libMultiRobotPlanning.

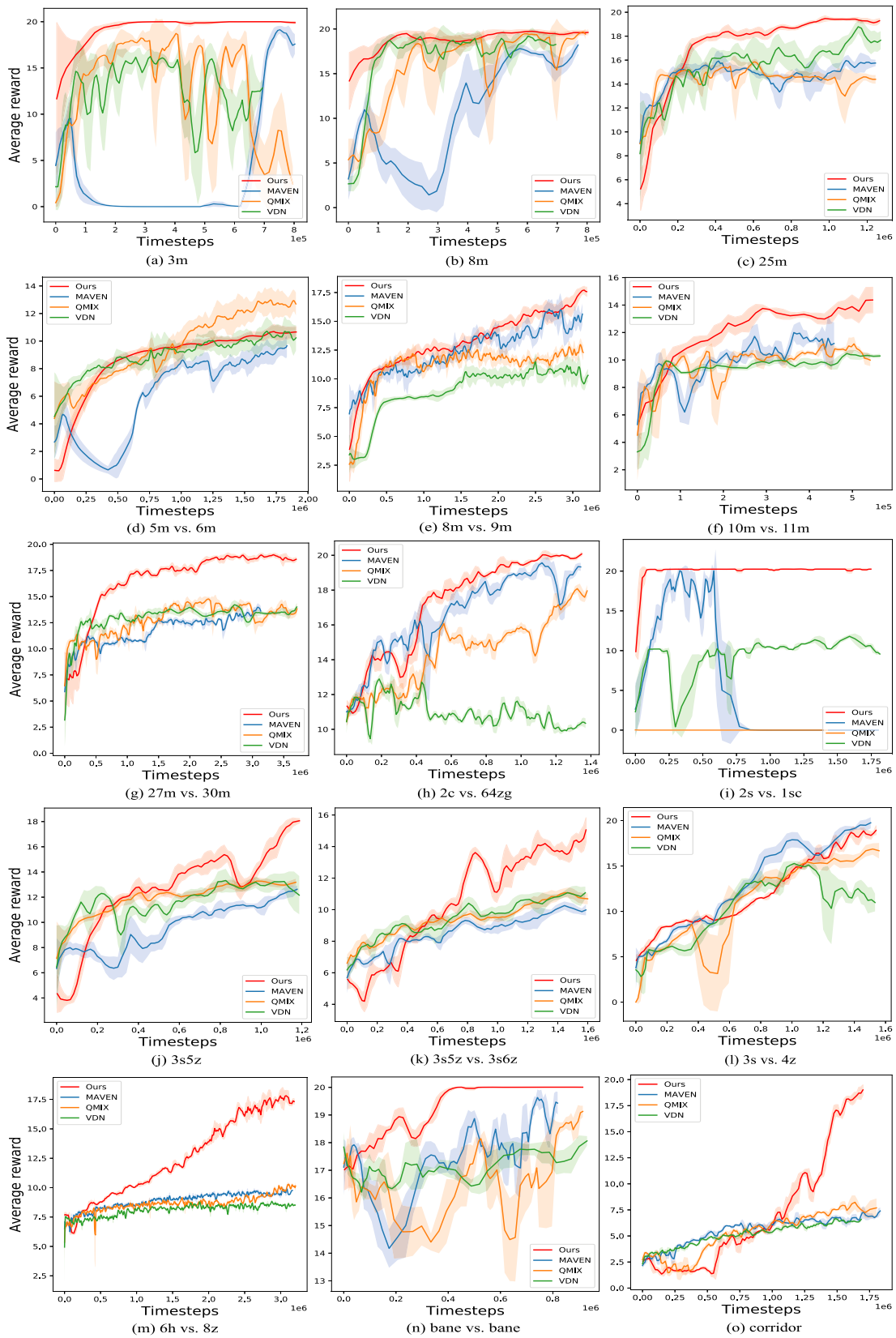


Fig. 4 Compare the average round rewards of different algorithms

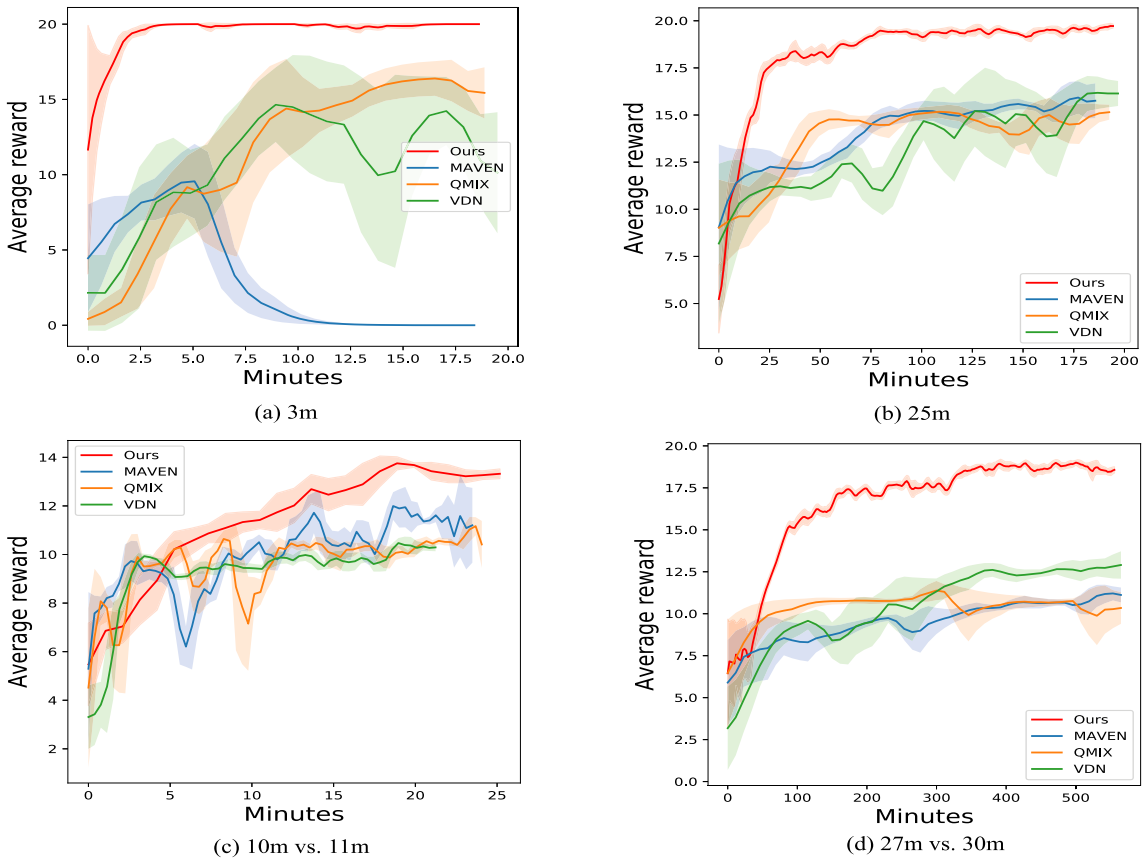


Fig. 5 Convergence speed comparison of different algorithms

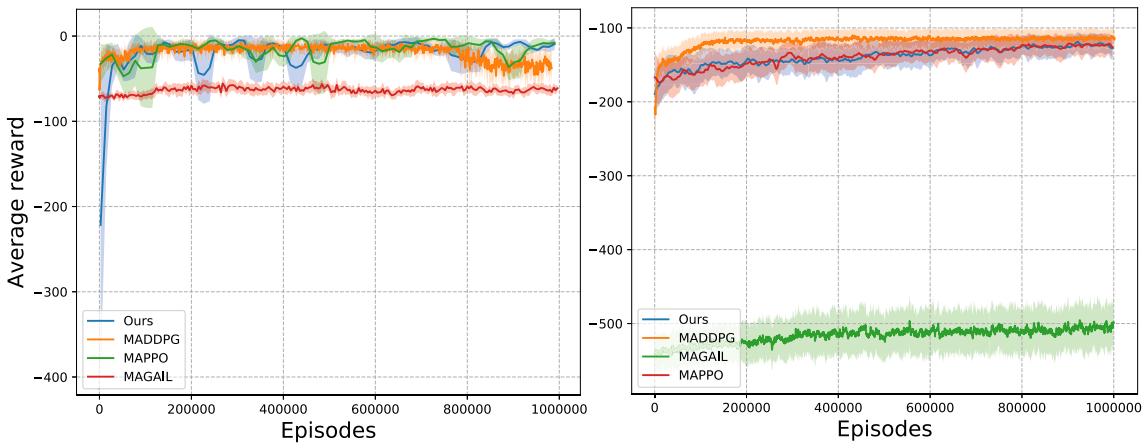


Fig. 6 Comparison of rewards obtained by different algorithms in different scenarios in MPE

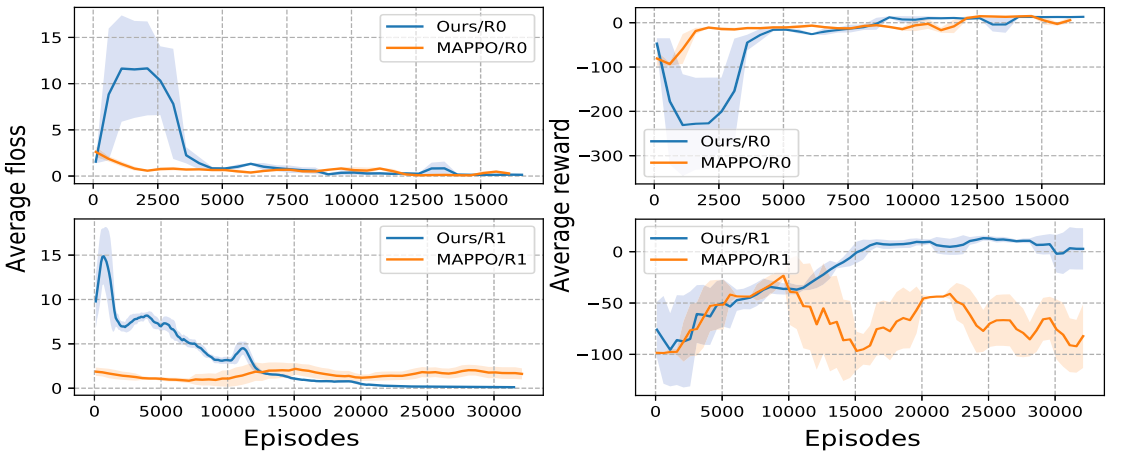
the expert demonstrations, compared to QMIX. In addition, the results indicate that our algorithm produces similar behavioral trajectories when expert demonstrations are given by algorithms valid in the real world. Thus, our algorithm would be easier to generalize to real-world applications.

Figure 9 shows the effect of different numbers of expert demonstrations on our algorithm. When the number of expert demonstrations is too small, it is difficult to provide adequate guidance to our algorithm. We argue that if the demonstration number is too small, it covers fewer situations, the agents may encounter. Therefore, no guidance can be made. In the figure, when the number of expert

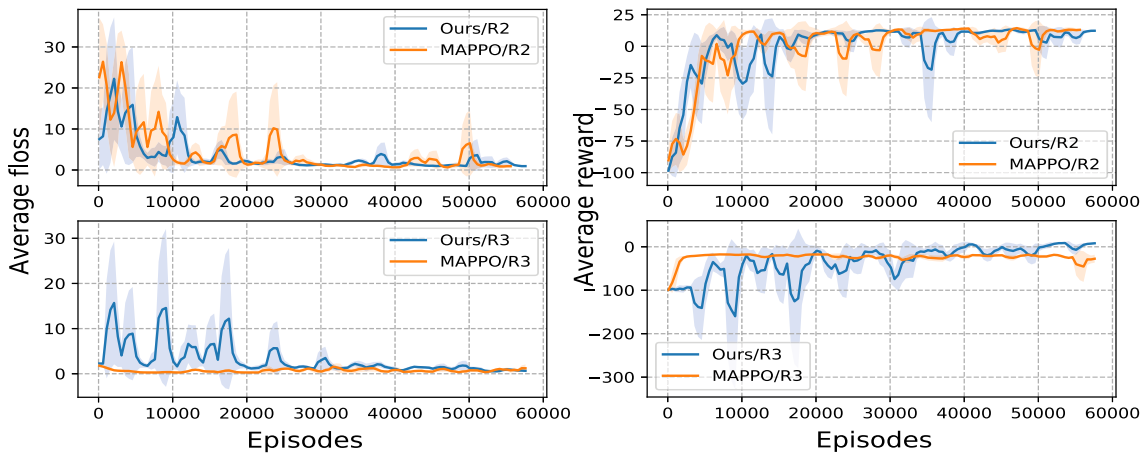
Table 3 The experimental results of different maps in the grid world environment: D represents the density of obstacles on the map, and '–' represents that the algorithm training fails to converge, and the results are unavailable

Environment setting			Makespan			Formation loss			Success rate			Step reward		
Map size	Agnet num	D	Ours	IPPO	MAPPO	Ours	IPPO	MAPPO	Ours	IPPO	MAPPO	Ours	IPPO	MAPPO
32 × 32	3	0.05	**58**	60.4	58	0.11	0.194	**0.08**	**100%**	100%	100%	**16.279**	14.724	16.046
	3	0.15	**62**	66.8	62.9	**1**	1.68	3.11	**100%**	80%	90%	**10.73**	−2.66	7.7
64 ×64	3	0.05	**126**	129.2	128	0.59	0.692	**0.326**	**100%**	100%	100%	**13.06**	12.03	−5.89
	3	0.15	129.2	139.2	**128**	1.574	1.49	**1.17**	**100%**	80%	100%	9.39	5.91	**10.75**
128 × 128	3	0.05	**255.4**	258.4	256.8	**0.63**	0.66	0.752	**100%**	100%	100%	**12.85**	12.44	12.26
	3	0.15	**256.4**	309.4	283.2	**0.92**	2.83	1.498	**100%**	60%	80%	**10.95**	1.35	5.88
256 × 256	3	0.05	**515.6**	619.4	–	**1.02**	10.072	–	**100%**	60%	–	**0.524**	−2.09	–
	3	0.15	**508**	–	–	**0.87**	–	–	**100%**	–	–	**14.21**	–	–
512 ×512	3	0.05	**1042**	–	–	**1.3**	–	–	**100%**	–	–	**13.58**	–	–
	3	0.15	**1236**	–	–	**3.246**	–	–	**60%**	–	–	**−9.94**	–	–

Highlight the highest-performing sections. For example the highest success rate, the smallest loss, the shortest time, etc



(a) Map size 32, obstacle density 0.05



(b) Map size 64, obstacle density 0.15

Fig. 7 Comparison of the algorithms' robustness to reward function: R0, R1, R2, and R3 represent different reward functions. R1 is relative to R0, and R3 is relative to R2, adding a penalty for formation loss in the reward function. Average floss refers to the average formation loss of multiple agents

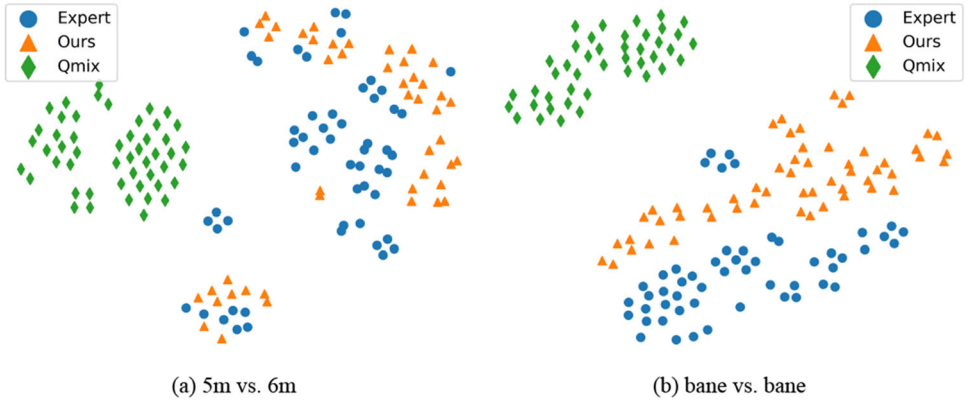


Fig. 8 Scatter plot of samples collected by different algorithms and expert demonstrations. Here, we use t-SNE [50] to cluster the state-action pair data generated by different algorithms

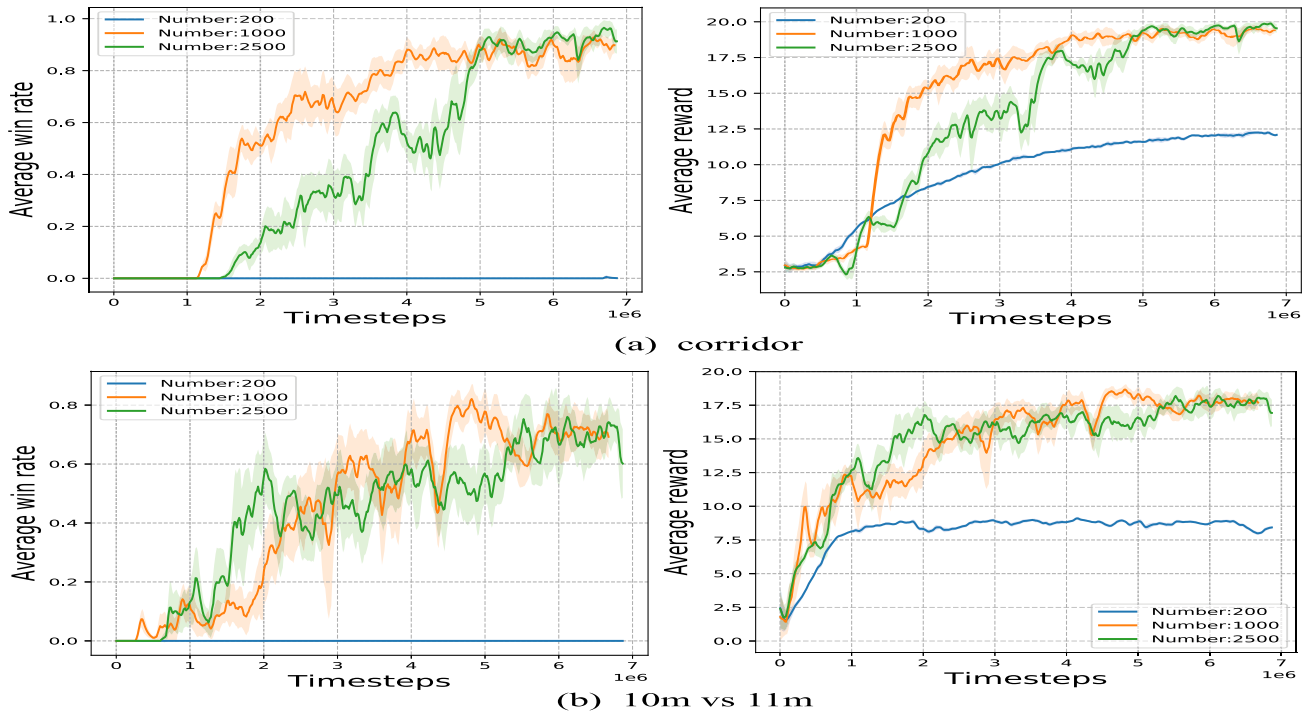


Fig. 9 The effect of different numbers of expert demonstrations on the algorithm in the corridor and 10-m versus 11-m map

demonstrations is 1000 or 2500, after convergence, there is not much difference in the performance of the algorithm. At this time, increasing the number of demonstrations has little impact on the algorithm.

6 Conclusions

In this work, we propose to use expert demonstrations to guide the reward decomposition at each time step, to address the credit assignment problem in MAS. We have conducted extensive experiments to validate that the proposed algorithm achieves or outperforms the SOTA level MARL methods. Compared to the EVD algorithms, such as QMIX and VDN, the convergence of our algorithm is scalable to the number of agents. In addition, our algorithm is more robust to different reward functions than MAPPO. Furthermore, our algorithm significantly outperforms imitation learning algorithms such as MAGAIL and MAVEN+BC with the same number of expert demonstrations, because we only estimate the reward distribution ratio, rather than directly estimating the reward of each agent. Compared to the baseline algorithms, since the expert data indirectly guide the policy update, the data distribution generated by the policy is closer to the expert demonstrations distribution.

Appendix

Experimental environment details

Starcraft II

SMAC consists of a set of StarCraft II micro-scenarios designed to evaluate the ability of agents to learn to coordinate to solve complex tasks. These carefully crafted scenarios require learning one or more micromanagement techniques to defeat the enemy. Each scene is a confrontation between two armies. The initial location, number, and unit types of each army vary by scenario, as does the presence of high ground or impassable terrain. A trained algorithm controls one side, and the other side consists of enemy units controlled by the built-in game AI using carefully designed non-learning heuristics. At the start of each turn, the game AI instructs its units to attack the allied agent using its scripted strategy. An episode ends when all units of either army die or a pre-specified time limit is reached (in which case, the game is considered a defeat for the allied agent). The goal of each scenario is to maximize the winning rate of the learned policy.

Agent Observations At each time step, the agent receives local observations drawn within its field of view. This contains map information within a circular area around it. From each agent's point of view, line-of-sight bounds make the environment partially observable. Agents can observe other agents if they are alive and within line of sight. Therefore, agents cannot determine whether their teammates are far away or dead. The feature vector observed by each agent contains the following attributes for friendly and enemy units in view: distance, relative x, relative y, health, shield, and unit type. Additionally, the agent can access the last actions of allied units in view. Finally, the agent can observe surrounding terrain features;

The global state is only available to the agent during training and contains information about all units on the map. Specifically, the state vector includes the coordinates of all agents relative to the map's center, and the unit features present in the observation. Finally, the last actions of all agents are attached to the central state.

Action Space The discrete set of actions the agent is allowed to take include move[direction] (four directions: north, south, east, or west), attack[enemy-id], stop, and no-op. A dead agent can only take no-ops, but a live agent cannot. Medivacs must use the heal[agent-id] action instead of attack[enemy-id] as healing units. The maximum number of actions an agent can perform is between 7 and 70, depending on the scene.

Reward The overall goal is to achieve the highest win rate in each combat scenario. We provide options for sparse rewards, which will cause the environment to return rewards of only +1 (for wins) and -1 (for losses). However, we also provide a default setting for the shape reward signal calculated based on the health damage, the agent does and receives, some positive (negative) bonus after killing an enemy (allied) unit, and/or positive (negative) rewards for winning (losing) a battle.

For more specific details about the SMAC environment see the link: https://github.com/deepmind/pysc2; https://github.com/oxwhirl/smac.

MPE

MPE is a time-discrete and space-continuous two-dimensional multi-agent environment developed by OpenAI. The environment completes a series of tasks by controlling the movement of different character particles in two-dimensional space. At present, it is widely used in the simulation verification of various MARL algorithms. As shown in Fig. 3b, we use spread and communication maps in this paper.

Spread The number of agents and landmarks in the spread map is the same, where black is the landmark and blue is the agent. The agent sees the landmark location and is rewarded based on its proximity to the landmark, and the agents need to negotiate to avoid the same landmark as their goal.

Communication There are two agents in the communication map and three landmarks of different colors. But one of the agents is a non-moving "speaker" (gray) (able to see the other agent's goal), and the other agent is a listener (cannot speak, but must navigate to the correct landmark and its goal subject to change). The gray agent sends a message to another agent, telling it the location of the target and helping it to complete the task.

Since the state and action space of each environment are inconsistent, please refer to the link for details of the MPE environment: https://github.com/openai/multiagent-particle-envs.

Grid world

As shown in Fig. 3c, grid world is a discretized environment, and the number of agents in this environment is always 3. The agent walks from the upper right corner to the lower corner target position in maps of different sizes. Its state space is a square field of vision nine grids away from the agent's position, within which obstacles and other agents can be observed; the action space is five actions, including up, down, left, right, and stop. The round ends when the agents reach the target position or hit an object (obstacle or agent). For more specific details about the

SMAC environment see the link: https://github.com/whoenig/libMultiRobotPlanning.

Baseline algorithm

The baseline algorithms we use include VDN [6], QMIX [7], QTRAN [22], MAVEN [21], MAVEN+BC [46], COMA [24], MAPPO [26], IPPO [47], MAGAIL [42], RODE [48], and MADDPG [5]. VDN, QMIX, QTRAN, and MAVEN are all explicit value decomposition algorithms, but they use different decomposition mechanisms. The VDN fits the joint action-value function in a summation fashion. QMIX uses a family of nonlinear functions for nonlinear fitting, which is bound by monotonicity. Based on VDN and QMIX algorithms, QTRAN proposes a more general value decomposition method to decompose any decomposable task successfully. The MAVEN algorithm improves the QMIX algorithm, which overcomes the problem that QMIX cannot perform effective exploration due to monotonic constraints. Specifically, it introduces a hierarchically controlled latent space, mixing value-based and policy-based methods, so that the behavior of each value-based agent depends on shared latent variables, and hierarchical policies control latent variables. MAVEN+BC introduces the behavior cloning (BC) [46] algorithm based on the MAVEN algorithm, which combines it with imitation learning. Note that BC directly models expert behavior through state-to-action supervision signals. MAVEN+BC will serve as an important baseline for comparison with our algorithm. COMA measures how much an agent's behavior contributes to the team via a counterfactual baseline. MAPPO is an extension of the PPO algorithm in the field of multi-agents. IPPO sets each agent as an independent learner. MAGAIL is an algorithm that combines imitation learning and inverse reinforcement learning. Similar to us, it learns expert strategies through the similarity between agent and expert behaviors. Note that MAVEN+BC and MAGAIL use the same amount of data as our expert demonstrations. RODE is a role-based multi-agent reinforcement learning method that uses a role selector to decompose the joint action space into a limited action space corresponding to different roles. The problem of too sizeable joint action state space in a multi-agent environment can be solved by dividing roles and layering. The MADDPG algorithm is an extension of the DDPG algorithm in the multi-agent field. The core is that each agent obtains all agent information to train its action-value function. We think that it is an implicit value decomposition algorithm.

Additional experimental hyperparameters

Table 4 shows the general hyperparameters, and Tables 5, 6, and 7 are the parameters fine-tuned according to the specific environment. In the following tables, "Max

Table 5 The hyperparameters used by our algorithm in the Starcraft II environment

Hyperparameters	Value
Num envs	8
Num GRU layers	1
RNN hidden state dim	64
fc layer dim	64
Num fc	2
Num fc after	1
Buffer length	400

Table 4 Generic hyperparameters

Common hyperparameters	Value
Value loss	Huber loss
GAE lambda	0.95
Max clipped value loss	0.2
Gradient clip norm	10
Use reward normalization	True
Use feature normalization	True
Network initialization	Orthogonal
Optimizer	Adam
Huber delta	10
Optimizer epsilon	1e-5
Weight decay	0
Gradient clip norm	10
Batch size	Num envs × buffer length × num agents
Mini batch size	Batch size/mini-batch
Gamma	0.99

Table 6 The hyperparameters used by our algorithm in the MPE environment

Hyperparameters	Value
Num envs	128
Num GRU layers	1
RNN hidden state dim	64
fc layer dim	64
Num fc	2
Num fc after	1
Buffer length	25

Table 7 The hyperparameters used by our algorithm in the grid world environment

Hyperparameters	Value
Num envs	100
Num GRU layers	1
RNN hidden state dim	64
fc layer dim	64
Num fc	2
Num fc after	1
Buffer length	Map size $\times$ 1.5

clipped value loss" refers to the value-clipping term in the value loss. "Huber delta" refers to the delta parameter in the Huber loss function. "Gamma" refers to the discount factor. "Num fc" refers to the number of linear layers in the MLP, and its dimension is represented by "fc layer dim." "Num layer after" indicates the number of linear layers after RNN.

Funding This work was supported by the National Defense Basic Research Program (JCKY2021204B051).

Data Availability The datasets generated during and/or analyzed during the current study are not publicly available due to REASON(S) WHY DATA ARE NOT PUBLIC but are available from the corresponding author on reasonable request.

Declarations

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

References

1. Ferber J, Weiss G (1999) Multi-agent systems: an introduction to distributed artificial intelligence vol. 1. Addison-Wesley Reading, ???
2. Hönig W, Kiesel S, Tinka A, Durham JW, Ayanian N (2019) Persistent and robust execution of mapf schedules in warehouses. IEEE Robot Autom Lett 4(2):1125–1131
3. Pendleton SD, Andersen H, Du X, Shen X, Meghjani M, Eng YH, Rus D, Ang MH (2017) Perception, planning, control, and coordination for autonomous vehicles. Machines 5(1):6
4. Sutton RS (1984) Temporal credit assignment in reinforcement learning. PhD thesis, University of Massachusetts Amherst
5. Lowe R, Wu Y, Tamar A, Harb J, Abbeel P, Mordatch I (2017) Multi-agent actor-critic for mixed cooperative-competitive environments. Neural Inform Process Syst (NIPS)
6. Sunehag P, Lever G, Gruslys A, Czarnecki WM, Zambaldi V, Jaderberg M, Lanctot M, Sonnerat N, Leibo JZ, Tuyls K (2018) Value-decomposition networks for cooperative multi-agent learning based on team reward. In: Proceedings of the 17th international conference on autonomous agents and multiagent systems (AAMAS'18), vol. 3, pp. 2085–2087. Assoc Computing Machinery
7. Rashid T, Samvelyan M, Schroeder C, Farquhar G, Foerster J, Whiteson S (2018) Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In: International Conference on Machine Learning, pp. 4295–4304. PMLR
8. Dou Z, Kuba JG, Yang Y (2022) Understanding value decomposition algorithms in deep cooperative multi-agent reinforcement learning. arXiv preprint arXiv:2202.04868
9. Gronauer S, Diepold K (2022) Multi-agent deep reinforcement learning: a survey. Artif Intell Rev, 1–49
10. De Hauwere Y-M, Vrancx P, Nowé A (2010) Generalized learning automata for multi-agent reinforcement learning. AI Commun 23(4):311–324
11. Ng AY, Russell S (2000) Algorithms for inverse reinforcement learning. In: Icml, vol. 1, p. 2
12. Arora S, Doshi P (2021) A survey of inverse reinforcement learning: challenges, methods and progress. Artif Intell 297:103500
13. Puterman ML (1995) Markov decision processes: discrete stochastic dynamic programming. J Oper Res Soc 46(6):792–792
14. Tan M (1993) Multi-agent reinforcement learning: independent vs. cooperative agents. In: Proceedings of the 10th international conference on machine learning, pp. 330–337
15. Tampuu A, Matiisen T, Kodelja D, Kuzovkin I, Korjus K, Aru J, Aru J, Vicente R (2017) Multiagent cooperation and competition with deep reinforcement learning. PLoS ONE 12(4):0172395
16. Lee KM, Subramanian SG, Crowley M (2021) Investigation of independent reinforcement learning algorithms in multi-agent environments. arXiv preprint arXiv:2111.01100
17. Canese L, Cardarilli GC, Di Nunzio L, Fazzolari R, Giardino D, Re M, Spanò S (2021) Multi-agent reinforcement learning: a review of challenges and applications. Appl Sci 11(11):4948
18. Zhang K, Yang Z, Başar T (2021) Multi-agent reinforcement learning: a selective overview of theories and algorithms. Handbook of Reinforcement Learning and Control, 321–384
19. Du W, Ding S (2021) A survey on multi-agent deep reinforcement learning: from the perspective of challenges and applications. Artif Intell Rev 54(5):3215–3238
20. Claus C, Boutilier C (1998) The dynamics of reinforcement learning in cooperative multiagent systems. AAAI/IAAI 1998(746–752):2
21. Mahajan A, Rashid T, Samvelyan M, Whiteson S (2019) Maven: multi-agent variational exploration. In: Proceedings of the 33rd international conference on neural information processing systems, pp. 7613–7624
22. Son K, Kim D, Kang WJ, Hostallero DE, Yi Y (2019) Qtran: learning to factorize with transformation for cooperative multi-agent reinforcement learning. In: International conference on machine learning, pp. 5887–5896. PMLR

23. Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2016) Continuous control with deep reinforcement learning. In: ICLR (Poster)
24. Foerster J, Farquhar G, Afouras T, Nardelli N, Whiteson S (2018) Counterfactual multi-agent policy gradients. In: Proceedings of the AAAI conference on artificial intelligence, vol. 32
25. Chung J, Gulcehre C, Cho K, Bengio Y (2014) Empirical evaluation of gated recurrent neural networks on sequence modeling. In: NIPS 2014 workshop on deep learning, December 2014
26. Yu C, Velu A, Vinitsky E, Wang Y, Bayen A, Wu Y (2021) The surprising effectiveness of mappo in cooperative, multi-agent games. arXiv preprint arXiv:2103.01955
27. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347
28. Littman ML (1994) Markov games as a framework for multi-agent reinforcement learning. In: Machine learning proceedings 1994, pp. 157–163. Elsevier, ???
29. Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. J Artif Intell Res 4:237–285
30. Hadfield-Menell D, Russell SJ, Abbeel P, Dragan A (2016) Cooperative inverse reinforcement learning. Adv Neural Inform Process Syst, 29
31. You C, Lu J, Filev D, Tsiotras P (2019) Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning. Robot Auton Syst 114:1–18
32. Wu P, Jia X, Chen L, Yan J, Li H, Qiao Y (2022) Trajectory-guided control prediction for end-to-end autonomous driving: a simple yet strong baseline. arXiv preprint arXiv:2206.08129
33. Baumberg A (2000) Reliable feature matching across widely separated views. In: Proceedings IEEE conference on computer vision and pattern recognition. CVPR 2000 (Cat. No. PR00662), vol. 1, pp. 774–781. IEEE
34. Abbeel P, Ng AY (2004) Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the 21st international conference on machine learning, p. 1
35. Ratliff ND, Bagnell JA, Zinkevich MA (2006) Maximum margin planning. In: Proceedings of the 23rd International Conference on Machine Learning, pp. 729–736
36. Herman M, Gindele T, Wagner J, Schmitt F, Burgard W (2016) Inverse reinforcement learning with simultaneous estimation of rewards and dynamics. In: Artificial intelligence and statistics, pp. 102–110 . PMLR
37. Ziebart BD, Maas AL, Bagnell JA, Dey AK (2008) Maximum entropy inverse reinforcement learning. In: Aaai, vol. 8, pp. 1433–1438. Chicago, IL, USA
38. Guiasu S, Shenitzer A (1985) The principle of maximum entropy. Math Intell 7(1):42–48
39. Bloem M, Bambos N (2014) Infinite time horizon maximum causal entropy inverse reinforcement learning. In: 53rd IEEE conference on decision and control, pp. 4911–4916. IEEE
40. Ho J, Ermon S (2016) Generative adversarial imitation learning. Adv Neural Inf Process Syst 29:4565–4573
41. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2020) Generative adversarial networks. Commun ACM 63(11):139–144
42. Song J, Ren H, Sadigh D, Ermon S (2018) Multi-agent generative adversarial imitation learning. arXiv preprint arXiv:1807.09936
43. Syed U, Bowling M, Schapire RE (2008) Apprenticeship learning using linear programming. In: Proceedings of the 25th international conference on machine learning, pp. 1032–1039
44. Samvelyan M, Rashid T, de Witt CS, Farquhar G, Nardelli N, Rudner TGJ, Hung C-M, Torr PHS, Foerster J, Whiteson S (2019) The StarCraft multi-agent challenge. CoRR **abs/1902.04043**
45. Liu S, Wen L, Cui J, Yang X, Cao J, Liu Y (2020) Moving forward in formation: a decentralized hierarchical learning approach to multi-agent moving together. arXiv preprint arXiv:2011.02373
46. Codevilla F, Santana E, López AM, Gaidon A (2019) Exploring the limitations of behavior cloning for autonomous driving. In: Proceedings of the IEEE/CVF international conference on computer vision, pp. 9329–9338
47. de Witt CS, Gupta T, Makoviichuk D, Makoviychuk V, Torr PH, Sun M, Whiteson S (2020) Is independent learning all you need in the starcraft multi-agent challenge? arXiv preprint arXiv:2011.09533
48. Wang T, Gupta T, Peng B, Mahajan A, Whiteson S, Zhang C (2021) Rode: learning roles to decompose multi- agent tasks. In: Proceedings of the international conference on learning representations . OpenReview
49. Barer M, Sharon G, Stern R, Felner A (2014) Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In: Seventh annual symposium on combinatorial search
50. Van der Maaten L, Hinton G (2008) Visualizing data using t-sne. J Mach Learn Res 9(11)