Reinforcement Learning With Task Decomposition for Cooperative Multiagent Systems

Changyin Sun, *Senior Member, IEEE*, Wenzhang Liu, and Lu Dong, *Member, IEEE*

***Abstract*— In this article, we study cooperative multiagent systems (MASs) with multiple tasks by using reinforcement learning (RL)-based algorithms. The target for a single-agent RL system is represented by its scalar reward signals. However, for an MAS with multiple cooperative tasks, the holistic reward signal consists of multiple parts to represent the tasks, which makes the problem complicated. Existing multiagent RL algorithms search distributed policies with holistic reward signals directly, making it difficult to obtain an optimal policy for each task. This article provides efficient learning-based algorithms such that each agent can learn a joint optimal policy to accomplish these multiple tasks cooperatively with other agents. The main idea of the algorithms is to decompose the holistic reward signal for each agent into multiple parts according to the subtasks, and then the proposed algorithms learn multiple value functions with the decomposed reward signals and update the policy with the sum of distributed value functions. In addition, this article presents a theoretical analysis of the proposed approach. Finally, the simulation results for both discrete decision-making and continuous control problems have demonstrated the effectiveness of the proposed algorithms.**

***Index Terms*— Cooperative multiagent system (MAS), multitask, reinforcement learning (RL), task decomposition.**

I. Introduction

MULTIAGENT systems (MASs) are a set of distributed artificial intelligent system (DAIS) that deals with many complex, multitask, and strongly coupled problems, e.g., multimanipulator control [1], multiagent cooperative tracking [2], [3], and multiagent navigation [4], which cannot be handled by a single-agent independently. Different from the single-agent scenario, if there is more than one agent in a control system, the relationship between each agent, such as cooperation and competition, under limited communication is studied. It will bring more difficulties when the connection topology between agents is complex [5]–[11]. In an MAS, the agents take actions through distributed policies in a common environment and communicate with each other to get further information and then influence the environment together (see Fig. 1). Consider

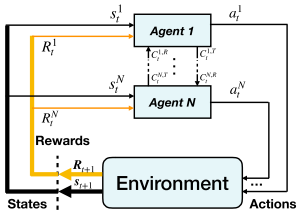


Fig. 1. Agent–environment interaction process for N-agent RL system. $C_t^{i,R}$ and $C_t^{i,T}$ represent the received information and transmitted information of agent i at time step t (for $i = 1, \ldots, N$), respectively.

an application example. Suppose that a control center receives N calls for the emergency ambulance at the same time. Then, it needs to decide how to allocate N ambulances at different places to these destinations as soon as possible. The problem can be formulated as a multiagent navigation problem with target assignment and path planning for all ambulances. For each ambulance, it has to get the knowledge about *where to go* and *how to go*. The former is a path planning problem, and the latter is a navigation problem. This article aims to study how to learn the subtasks separately instead of a holistic task to make the whole training process more efficient.

The multiagent reinforcement learning (MARL) is an extension from single-agent reinforcement learning (RL) to DAIS. In RL, the agent aims to learn an optimal policy such that it can get as much as accumulated numerical reward from the environment by machine–environment interacting. RL is built on the Markov decision process (MDP) assumption, and a key idea of RL is the use of value functions to organize and structure the search for good policies [12]. The last few decades have seen great progress in RL, as well as machine learning and artificial intelligence. A common important reason for that is the great improvement of computing ability and big data. Along with the development of deep learning (DL) technology, Mnih *et al.* [13] have published a deep Q-learning algorithm with deep Q-network (DQN) that can let the machine play Atari games, and it achieved performance comparable with the professional human level. DQN can solve many high-dimensional, complex problems, but its capability for continuous control problems is limited. Hence, Lillicrap *et al.* [15] extended deterministic policy gradient (DPG) algorithm in [14] and proposed deep DPG (DDPG) algorithm to solve large scale and continuous control problems. DDPG is built on the actor–critic architecture that consists of an actor network and a critic network [16]. For a cooperative MAS, these algorithms can help each agent understand the environment and interact with each other under the designed reward function.

Although those RL algorithms can solve many complex control problems, they cannot always be applied to MAS directly. For each agent in MARL, it might treat others

Manuscript received May 8, 2019; revised November 19, 2019 and March 6, 2020; accepted May 16, 2020. This work was supported by the National Natural Science Foundation of China under Grant 61921004, Grant U1713209, Grant 61803085, and Grant 62041301. *(Corresponding author: Changyin Sun.)*

Changyin Sun and Wenzhang Liu are with the School of Automation, Southeast University, Nanjing 210096, China, and also with the Key Laboratory of Measurement and Control of Complex Systems of Engineering, Ministry of Education, Southeast University, Nanjing 210096, China (e-mail: cysun@seu.edu.cn; wzliu@seu.edu.cn).

Lu Dong is with the College of Electronics and Information Engineering, Tongji University, Shanghai 201804, China (e-mail: ldong@tongji.edu.cn).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2020.2996209

as a part of the environment and learn the optimal policy based on its observation from this environment (see Fig. 1). Thus, any change in one agent's policy may influence the performance of other agents' policies. A widely used approach to extend the RL from single-agent scenarios to MAS is centralized learning and decentralized execution, in which all agents are trained simultaneously. For example, Lowe *et al.* [6] adopted a framework with centralized training and decentralized execution to make the agents act in a mixed cooperative–competitive complex environment. Foerster *et al.* [17] proposed counterfactual multiagent (COMA) policy gradient to improve the learning performance with centralized critic and decentralized actors. However, these algorithms that obtain a single numerical feedback reward from the environment do not optimize each subtask individually in MAS with multiple tasks. To make it learn each subtask more efficiently in MAS, in this article, we separately optimize multiple value functions for these subtasks and jointly find an optimal policy for each agent. In addition, the experiment results empirically demonstrate the performance of our proposed algorithms. The main contributions of this article are listed as follows.

1) We apply the idea of task decomposition to multiagent Q-learning (MAQ) algorithms for cooperative MAS with multiple tasks, where each agent learns multiple Q-functions through value decomposition and jointly improves the performance of the policy with these Q-functions. The MAQ with task decomposition (MAQ-TDec) algorithm is introduced to improve the performance of the original MAQ and get almost the same performance with the state-of-the-art algorithms. In addition, we apply the same idea to the compared state-of-the-art algorithms and get significant improvements in the performance.
2) We also consider cooperative MAS in continuous control scenarios and propose an algorithm called multiagent DPG with task decomposition (MADPG-TDec) that consists of a common actor network and distributed critic networks for each agent via value decomposition. The critic mechanism is distributed such that it aims at each subtask separately rather than optimizing a holistic objective function directly.

The rest of this article is organized as follows. The related works about MARL and multitask learning methods are introduced in Section II. Section III is about the research background and problem formulation. The details about our proposed algorithms are provided in Section IV. In Section V, we show the simulation results to verify the proposed algorithms. Finally, the conclusion is drawn in Section VI.

II. Related Work

The research on MARL aims to learn optimal distributed policies for multiple agents such that these agents can cooperatively work together [6], [11] or compete to achieve a Nash equilibrium [5], [6], [9]. In this article, we aim at MARL for cooperative tasks.

A direct approach of applying single-agent RL into MARL is independent learning, e.g., the independent Q-learning algorithm in [18] and independent DQN in [19]. The agents in independent learning algorithms do not share information with others and are easy to overfit and find the local optimal policies. A popular framework for MARL is based on centralized training and decentralized execution, in which agent collects information of other agents during training time and executes actions individually according to its local observation. Lowe *et al.* [6] proposed a multiagent deep DPG (MADDPG) algorithm in which the centralized critic and deterministic policies for each agent are adopted. However, the input space of the critic grows linearly with the number of agents [6], which will be computationally expensive. Also, the COMA policy gradients algorithm in [17] used a global centralized critic to estimate a counterfactual advantage for decentralized policies.

In the fully centralized and decentralized approaches, Sunehag *et al.* [20] found the problem of spurious rewards and a phenomenon called "lazy agent." They addressed these problems by decomposing the team value function into agentwise value functions with value decomposition networks (VDNs) and learning the sum of these value functions with the joint team reward. However, the full factorization of VDN is not necessary to extract decentralized policies that are fully consistent with their centralized counterparts [21]. To ensure the consistency that greedy policy with holistic Q value yields the same results as a set of individual greedy policies with agentwise Q values, Rashid *et al.* [21] proposed a monotonic value function factorization method called QMIX. QMIX is trained end-to-end by using a mixing network that combines the agentwise Q values into a holistic one.

In this article, we aim at multitask MARL (MT-MARL), where agents learn to solve multiple tasks cooperatively. The task of an RL system is described in its numerical reward signal that is designed previously. In the single-agent scenarios, Marthi [22] studied the automatic shaping and the decomposition of reward functions to speed up the learning process. The proposed method first learned the shape of the original reward function given by the environment and then decomposed the reward function according to the shape it learned. Van Seijen *et al.* [23] studied the decomposed reward functions for multitask RL. They first decomposed the reward function given by the environment manually. Then, they learned an optimal policy with multiple Q values, which resulted in speeding up the learning process and increasing accuracy. To achieve a unified policy for multiple tasks in multiagent scenarios, Omidshafiei *et al.* [24] first learned task-specific policies individually, and then, they composed these policies into one policy by using policy distillation technology (the method that we labeled as "Distillation"), which performs well across multiple related tasks.

Unlike the distillation method that contains two phases, the proposed algorithms in this article learn a unified optimal policy for each agent directly with multiple value functions. The value decomposition applied here is based on task decomposition where taskwise value functions are learned separately, which is different from the agentwise value decomposition methods used in VDN and QMIX [20], [21]. An advantage of the task decomposition methods is that they can avoid task-specific local optimal policies by using these multiple value functions.

III. Background and Problem Formulation

In this section, we first introduce the background of MDP and RL. Then, we provide the problem formulation about MARL and that in multitask scenario.

A. Markov Decision Process and Reinforcement Learning

MDP is a set of problems in which the historical state and decision data have no effect on current state and decision-making. An MDP can be described as a tuple $< \mathcal{S}, \mathcal{A}, P, R, \gamma >$, where $\mathcal{S}$ and $\mathcal{A}$ denote the entire state space and action space, respectively. $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is the state transition probability, and for a model free problem, P is unknown prior knowledge. R denotes the reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$, which is a bounded function that returns the immediate step reward to agent. It is the only feedback signal when the agent is interacting with environment. $\gamma \in (0, 1)$ is the discount factor that determines the current value of future rewards [12].

To interact with environment, the agent follows a policy π and chooses an available action $a_t \in \mathcal{A}$ conditional on current state s_t. Then, s_t will be transmitted to another state s_{t+1} with an immediate reward R_{t+1} from environment. The policy π could be a distribution form with respect to available action set, $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ or a deterministic function from state set to action set, $\pi : \mathcal{S} \mapsto \mathcal{A}$.

When agent keeps interacting with the environment following a policy π, we will get a time series of states, actions and rewards that form a trajectory: $\tau = \{s_0, a_0, s_1, R_1, \ldots, s_t, a_t, s_{t+1}, R_{t+1}, \ldots\}$, and we call $< s_t, a_t, s_{t+1}, R_{t+1} >$ as a specific transition from trajectory τ.

The essential goal of RL is finding an optimal policy π^* to maximize the expected discounted total reward $\mathbb{E}[G_t] = \mathbb{E}[\sum_{l=0}^{\infty} \gamma^l R_{t+l+1}]$, where $G_t = \sum_{l=0}^{\infty} \gamma^l R_{t+l+1}$ is the discounted return and l is the time step count factor.

To maximize $\mathbb{E}[G_t]$, a value function is defined to measure the performance of the policy at a specific state (and an action). Formally, the state value function with discount factor γ under π is defined as

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{l=0}^{\infty} \gamma^l R_{t+l+1} | s_t = s\right]. \tag{1}$$

It represents the expected G_t that the agent will get in the future if it keeps following policy π since $s_t = s$. Thus, we can evaluate the quality of state s under policy π. The equivalent formation called the Bellman equation is

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[R_t + \gamma V^{\pi}\left(s'\right) | s_t = s\right]. \tag{2}$$

The state–action value function, which is also called Q value function, can be defined based on the state value function conditioned on a specified action a, that is

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{l=0}^{\infty} \gamma^l R_{t+l+1} | s_t = s, a_t = a\right] \tag{3}$$

and we can also get its Bellman equation form as

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[R_{t+1} + \gamma \sum_{s', a'} p\left(s' | s, a\right) Q^{\pi}\left(s', a'\right) | s, a\right] \tag{4}$$

where $p(s'|s, a)$ is the one-step transmission probability, and it will be ignored if the system is deterministic. For value-based algorithms, the key idea is optimizing the Q value function and then improving the policy by the optimized Q value. The optimized Q value function satisfies

$$Q^*(s, a) = \mathbb{E}\left[R(s, a) + \gamma \max_{a'} Q^*\left(s', a'\right)\right]. \tag{5}$$

There are many algorithms designed to optimize these values and are clustered as value-based algorithms, such as Q-learning [25]–[28] and Sarsa [12], and policy-based algorithms, such as policy gradient [29] and DPG [14], [15].

B. Multiagent Reinforcement Learning

An MARL system is built on MDP that can be described as $< \mathcal{S}^1, \ldots, \mathcal{S}^{N_a}, \mathcal{A}^1, \ldots, \mathcal{A}^{N_a}, P, R^1, \ldots, R^{N_a}, \gamma >$, in which N_a is the agents' number. $\mathcal{S}^i$ and $\mathcal{A}^i$ denote the feasible state space and action space, respectively, for agent i [30]. In some specific scenario, such as heterogeneous MAS [31], the state space and action space for each agent are different, whereas, in this article, we suppose that all agents share a common state space $\mathcal{S}$ and a common action space $\mathcal{A}$. P is state transition probability that satisfies the Markov Property. $R^i : \mathcal{S} \times \mathcal{A}^1 \times \cdots \mathcal{A}^{N_a} \times \mathcal{S} \mapsto \mathbb{R}$ denotes the reward function for agent i in which other agents' actions are also considered as its input. In MAS, π^i is the policy for agent i, and $\boldsymbol{\pi} = (\pi^1, \ldots, \pi^{N_a})$ is the joint policy for all agents. It should be noted that any change of agent i's policy π^i may influence the performance of the other agents, for generally $P(s'|s, a, \boldsymbol{\pi}) \neq P(s'|s, a, \boldsymbol{\pi}')$ if $\boldsymbol{\pi}' \neq \boldsymbol{\pi}$.

The Q value for agent i is defined as $Q^i(s, a^i, a^{-i}) = \mathbb{E}_{\boldsymbol{\pi}}[\sum_{l=0}^{\infty} \gamma^l R^i_{t+l+1} | s^i_t = s, a^i_t = a^i, a^{-i}_t = a^{-i}]$, where s^i_t is its observation at time step t, a^i is the action, a^{-i} is other agents' joint action, and R^i_t is its immediate reward at t step. When we update a Q value for one agent, we will assume that the other agents' policies are fixed and that will simplify the problem.

For continuous control problems with parameterized policies $\pi^i_{\theta}, i = 1, \ldots, N_a$, the targets are optimizing the objective functions for these agents that are defined as $\mathcal{J}^i(\pi^i_{\theta}) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R^i_{t+1} | s^i_0, \pi^i_{\theta}, \pi^{-i}_{\theta}], i = 1, \ldots, N_a$, where π^{-i}_{θ} is the joint policy of other agents.

C. Multitask MARL

In practice, an MAS usually contains multiple tasks, and the rewards for these tasks compose a holistic reward function for each agent. Most of the existing MARL algorithms treat these multiple tasks together as a holistic task and optimize the corresponding objective function. However, it is usually hard to find global optimal policies for an MARL system, especially by using neural network-based approximators. If an algorithm gets a suboptimal solution for the holistic objective function, it naturally cannot guarantee the optimality for each subobjective. Thus, we consider multiobjective optimality for MT-MARL via task decomposition.

For a cooperative MARL with M tasks and N_a agents, the optimization problems can be expressed as

$$\max_{\pi^i} \mathcal{J}^i\left(\pi^i\right), i = 1, \ldots, N_a \tag{6}$$

where $\mathcal{J}^i(\pi^i) = [\mathcal{J}_1^i(\pi^i), \ldots, \mathcal{J}_M^i(\pi^i)]$. $\mathcal{J}_k^i(\pi^i)$ is agent i's objective function for the kth task, $k = 1, \ldots, M$. Generally, the optimal policies for problem (6) that maximize each subobjective simultaneously does not always exist [32], but one can get a Pareto optimal policy π^{i*} [32]–[35].

Definition 1 (Pareto Domination): For the optimization problem of agent i in (6), a policy π_1^i is said to Pareto dominate π_2^i, that is $\pi_1^i \succeq \pi_2^i$, if $\mathcal{J}_k^i(\pi_1^i) \geq \mathcal{J}_k^i(\pi_2^i) \forall k = 1, \ldots, M$.

Definition 2 (Pareto Optimal): For the optimization problem of agent i in (6), a policy π^{i*} is called Pareto optimal if $\pi^{i*} \succeq \pi^i \forall \pi^i \in \Pi^i$, where Π^i is the set of all feasible policies for agent i.

In a general multiobjective optimization problem, the Pareto optimal solution is not unique. We call the set of all Pareto optimal solutions as the Pareto set ($\mathcal{P}_\pi = \{\pi^* \mid \pi^* \text{ is Pareto optimal}\}$) and the corresponding objective results as the Pareto frontier ($\mathcal{P}_\mathcal{J} = \{\mathcal{J}(\pi^*) | \pi^* \in \mathcal{P}_\pi\}$).

IV. Task Decomposition-Based Learning Approach

In this section, we consider multiagent with multiple cooperative tasks and propose task decomposition-based methods for both discrete and continuous control problems that make the learning process more effective. As introduced in [36], we can initially decompose the holistic task into several subtasks from the knowledge contained in the reward function. Here, we first introduce the value decomposition method and then propose two algorithms: MAQ-TDec and MADPG-TDec. We also analyze the convergence and superiority of the proposed algorithms.

A. Value Decomposition

For a given MDP with multiple tasks, the reward for each subtask composes the holistic reward function. Most of the existing algorithms optimize the holistic value function to achieve these tasks. Here, we first study how the value function can be decomposed as subvalue functions, and then, we give the condition to decompose the holistic reward function into subreward functions such that it can optimize the corresponding subobjective functions separately.

Theorem 1: If the reward function for an MDP, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$, can be decomposed into M subreward functions, i.e., $R(s, a, s') = \sum_{k=1}^M r_k(s, a, s')$, then its state value function and Q value function following a policy π can also be decomposed as $V^\pi(s) = \sum_{k=1}^M V_k^\pi(s)$ and $Q^\pi(s, a) = \sum_{k=1}^M Q_k^\pi(s, a)$ [23].

Proof: In (1), $V^\pi(s)$ is defined as

$$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{l=0}^{\infty} \gamma^l R_{t+l+1} | s_t = s\right].$$

Then, according to the reward decomposition formation, we can decompose the state value function as

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi\left[\sum_{l=0}^{\infty} \gamma^l \sum_{k=1}^{M} r_{t+l+1,k} | s_t = s\right] \\ &= \mathbb{E}_\pi\left[\sum_{k=1}^{M} \gamma^l \sum_{l=0}^{\infty} r_{t+l+1,k} | s_t = s\right] \\ &= \sum_{k=1}^{M} \mathbb{E}_\pi\left[\gamma^l \sum_{l=0}^{\infty} r_{t+l+1,k} | s_t = s\right] \\ &= \sum_{k=1}^{M} V_k^\pi(s). \end{aligned} \tag{7}$$

In the same way, we can also get

$$Q^\pi(s, a) = \sum_{k=1}^{M} Q_k^\pi(s, a). \tag{8}$$

□

Since the value function is decomposed into several subfunctions, we next consider that if these subfunctions can be optimized separately. In fact, even though the value functions are decomposed by (7) and (8), they share a common policy π, which, sometimes, cannot achieve the optimalities for these subfunctions simultaneously. That means a common policy may only be able to optimize part of these subtasks.

In general, the optimal policy to maximize a given objective function is not unique, especially for high dimensions and complex problems. We denote Π^* as the optimal policy set for an MDP with multiple tasks, and $\Pi_k^*, k = 1, \ldots, M$, are optimal policy sets for the subtasks.

Assumption 1: In an MDP with M tasks, the reward function can be decomposed as $R(s, a, s') = \sum_{k=1}^M r_k(s, a, s')$ such that $\bigcap_{k=1}^M \Pi_k^* \neq \emptyset$ holds, and there is no temporal relations between these M tasks.

With Assumption 1, the availability of the task decomposition-based learning methods is guaranteed. It shows that the arbitrary form of decomposition for the holistic reward function cannot be applied to the decomposed training. The main reason for that is the multiple tasks for a system must be achieved under a common policy. In addition, we only study the tasks without temporal relations, which means that the decomposed parts never consider the priority of subtasks. Sometimes, the subtasks decomposed from the original whole task are invisible and unrealistic, but we can still decompose it and train it individually to get an equivalent optimal solution. Next, we propose two algorithms based on this assumption.

B. Multiagent Q-Learning With Task Decomposition

MAQ aims to learn an optimal Q-function for each agent such that it can use it to search optimal policy. In a cooperative scenario, the Q value of agent i can be defined as $Q^i(s, a^i, a^{-i}) = \mathbb{E}[\sum_{l=0}^\infty \gamma^l R_{t+l+1}^i | s_t^i = s, a_t^i = a^i, a_t^{-i} = a^{-i}]$. We update it by

$$\begin{aligned} &Q^i\left(s_t^i, a_t^i, a_t^{-i}\right) \\ &:= (1-\alpha) Q^i\left(s_t^i, a_t^i, a_t^{-i}\right) \\ &\quad + \alpha\left[r_{t+1}^i + \gamma \max_{a^i, a^{-i} \in \mathcal{A}} Q^i\left(s_{t+1}^i, a^i, a^{-i}\right)\right], \quad i = 1, \ldots, N_a \end{aligned} \tag{9}$$

where α is the learning rate, γ is a discount factor, and N_a is the number of agents. Equation (9) is the updating rule for MAQ with cooperative tasks.

Remark 1: The cooperative task mentioned in this article means that agents share the same reward function, and the learning goal is to maximize the common discounted return [8], [37]. Here, we use agent-specific notations r_t^i to represent the reward signals for agent i under its local observation and action at step t. They still share a common reward function.

Given that the reward signal for agent i is decomposed as $R_t^i = \sum_{k=1}^M r_{t,k}^i$, which satisfies Assumption 1, its Q value can be decomposed by Theorem 1 as

$$Q^i\left(s, a^i, a^{-i}\right) = \sum_{k=1}^{M} Q_k^i\left(s, a^i, a^{-i}\right) \tag{10}$$

where

$$Q_k^i\left(s, a^i, a^{-i}\right) = \mathbb{E}\left[\sum_{l=0}^{\infty} \gamma^l r_{t+l+1,k}^i | s_t^i = s, a_t^i = a^i, a_t^{-i} = a^{-i}\right] \tag{11}$$

is the sub-Q value for task k.

With these sub-Q values, the ϵ-greedy policy for agent i, which is applied to explore the environment during training, is defined as

$$\pi_\epsilon^i(s) = \begin{cases} \arg\max\limits_{a^i, a^{-i} \in \mathcal{A}} \sum\limits_{k=1}^{M} Q_k^i\left(s, a^i, a^{-i}\right), & \text{with probability 1-}\epsilon \\ \text{random action from } \mathcal{A}, & \text{with probability } \epsilon. \end{cases}$$

Different from traditional updating rule, such as (9), we update the holistic Q-function for agent i by optimizing its sub-Q functions separately by using

$$\begin{aligned} &Q_k^i\left(s_t^i, a_t^i, a_t^{-i}\right) \\ &:= (1-\alpha) Q_k^i\left(s_t^i, a_t^i, a_t^{-i}\right) \\ &\quad +\alpha\left[r_{t+1,k}^i + \gamma \max_{a^i, a^{-i} \in \mathcal{A}} Q_k^i\left(s_{t+1}^i, a^i, a^{-i}\right)\right], \quad k = 1, \ldots, M. \end{aligned} \tag{12}$$

Theorem 2: For MARL with M cooperative tasks and discrete actions, the updating rule (12) will converge to optimal solution and perform better compared with (9) when Assumption 1 holds.

Proof: It is assumed that Q^{i*} is the optimal value function for agent i, and then, it satisfies $Q^{i*} = \sum_{k=1}^M Q_k^{i*}$. We initialize agent i's Q value as $Q^{i,0}(s, a^i, a^{-i}) = \sum_{k=1}^M Q_k^{i,0}(s, a^i, a^{-i}) \forall s \in \mathcal{S}$ and $\forall a^i, a^{-i} \in \mathcal{A}$. Then, we iteratively update it by (12) until N_e steps and get $Q_k^{i,N_e}, k = 1, \ldots, M$. By defining a transform for the updating rules as $\mathcal{T}_k$, we have

$$\begin{aligned} &\left(\mathcal{T}_k Q_k^{i,N_e}\right)\left(s, a^i, a^{-i}\right) \\ &\quad = (1-\alpha) Q_k^{i,N_e}\left(s, a^i, a^{-i}\right) \\ &\qquad +\alpha\left[r_{t+1,k}^i + \gamma \max Q_k^{i,N_e}\left(s', \cdot, \cdot\right)\right], \quad k = 1, \ldots, M. \end{aligned} \tag{13}$$

With the norm $\|\cdot\|_{\sup}$, $\mathcal{T}_k$ is a contraction mapping [12]. Therefore

$$\begin{aligned} &\left\| Q_k^{i,N_e} - Q_k^{i*} \right\|_{\sup} \\ &\quad = \left\| \mathcal{T}_k Q_k^{i,N_e-1} - \mathcal{T}_k Q_k^{i*} \right\|_{\sup} \\ &\quad = \left\| (1-\alpha)\left(Q_k^{i,N_e-1} - Q_k^{i*}\right) \right. \\ &\qquad \left. +\alpha\gamma\left[\max Q_k^{i,N_e-1}\left(s', \cdot, \cdot\right) - \max Q_k^{i*}\left(s', \cdot, \cdot\right)\right] \right\|_{\sup} \\ &\quad \le (1-\alpha+\alpha\gamma)\left\| Q_k^{i,N_e-1} - Q_k^{i*} \right\|_{\sup} \\ &\quad \cdots \\ &\quad \le (1-\alpha+\alpha\gamma)^{N_e}\left\| Q_k^{i,0} - Q_k^{i*} \right\|_{\sup}. \end{aligned} \tag{14}$$

Then, we can get the holistic inequation as

$$\begin{aligned} \left\| Q^{i,N_e} - Q^{i*} \right\|_{\sup} &= \left\| \sum_{k=1}^{M} Q_k^{i,N_e} - \sum_{k=1}^{M} Q_k^{i*} \right\|_{\sup} \\ &= \left\| \sum_{k=1}^{M} \left(Q_k^{i,N_e} - Q_k^{i*}\right) \right\|_{\sup}. \end{aligned} \tag{15}$$

By applying the subadditivity of norm $\|\cdot\|_{\sup}$, we have

$$\begin{aligned} \left\| Q^{i,N_e} - Q^{i*} \right\|_{\sup} &\le \sum_{k=1}^{M} \left\| Q_k^{i,N_e} - Q_k^{i*} \right\|_{\sup} \\ &\le (1-\alpha+\alpha\gamma)^{N_e} \sum_{k}^{M} \left\| Q_k^{i,0} - Q_k^{i*} \right\|_{\sup}. \end{aligned} \tag{16}$$

Given that the $\alpha, \gamma \in (0, 1)$, and $\|Q_k^{i,0} - Q_k^{i*}\|_{\sup}$ is a constant value, we can get $\lim_{N_e \to \infty} (1-\alpha+\alpha\gamma)^{N_e} = 0$. Hence

$$\lim_{N_e \to \infty} \left\| Q^{i,N_e} - Q^{i*} \right\|_{\sup} = 0. \tag{17}$$

Furthermore, by using update rule (12), we have

$$\begin{aligned} &Q^{i,N_e+1}\left(s, a^i, a^{-i}\right) \\ &= \sum_{k=1}^{M} \left(\mathcal{T}_k Q^{i,N_e}\right)\left(s, a^i, a^{-i}\right) \\ &= (1-\alpha) Q^{i,N_e}\left(s, a^i, a^{-i}\right) \\ &\quad +\alpha\left[R_{t+1}^i + \gamma \sum_{k=1}^{M} \max Q_k^{i,N_e}\left(s', \cdot, \cdot\right)\right] \\ &\ge (1-\alpha) Q^{i,N_e}\left(s, a^i, a^{-i}\right) \\ &\quad +\alpha\left[R_{t+1}^i + \gamma \max \sum_{k=1}^{M} Q_k^{i,N_e}\left(s', \cdot, \cdot\right)\right] \\ &= (1-\alpha) Q^{i,N_e}\left(s, a^i, a^{-i}\right) + \alpha\left[R_{t+1}^i + \gamma \max Q^{i,N_e}\left(s', \cdot, \cdot\right)\right]. \end{aligned} \tag{18}$$

Equation (18) shows that the updating rule (12) can perform better than the original rule (9), and the gap between them will be larger along with the increase of the training step. Thus, the decomposition training method can not only converge to optimal solution but also perform better than the original method. □

We can apply neural networks to approximate these Q-functions for agent i by minimizing the losses defined according to the updating rule (12)

$$\mathcal{L}_k^i\left(\theta_{q,k}^i\right) = \mathbb{E}\left[\left(Q_k^i\left(s_t^i, a_t^i, a_t^{-i}; \theta_{q,k}^i\right) - y_{t,k}^i\right)^2\right], \quad k = 1, \ldots, M \tag{19}$$

Algorithm 1 MAQ-TDec

Input: Agent number N_a, task number M, episode number N_e, max trajectory length T, mini-batch size N_{batch}, learning rate α, discount factor γ, greedy probability ϵ, target network update period C.

Initialize: Random parameters for Q-networks, empty replay buffer $\mathcal{D}$.

for $episode = 1$ to N_e **do**
  Reset environment and get initialized states $\{s_0^i\}_{i=1}^{N_a}$.
  for $t = 0$ to $T-1$ **do**
    Get actions: $a_t^i = \pi_\epsilon^i(s_t^i), i = 1, \ldots, N_a$.
    Execute $\{a_t^i\}_{i=1}^{N_a}$ simultaneously and get $\{s_{t+1}^i\}_{i=1}^{N_a}$, $\{R_{t+1}^i\}_{i=1}^{N_a}$, where $R_{t+1}^i = [r_{t+1,1}^i, \ldots, r_{t+1,M}^i]$.
    Store $\{< s_t^i, a_t^i, s_{t+1}^i >, R_{t+1}^i\}_{i=1}^{N_a}$ into $\mathcal{D}$.
    Sample N_{batch} transitions from $\mathcal{D}$
    for $i = 1$ to N_a, $k = 1$ to M **do**
      Update Q-networks by minimizing mean of (19).
    end for
    Every C steps update target networks.
    $s_t^i \leftarrow s_{t+1}^i, i = 1, \ldots, N_a$.
  end for
end for

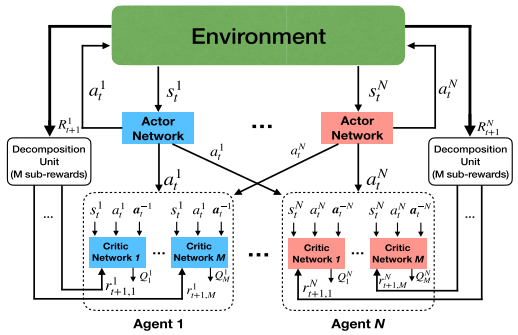


Fig. 2. Actor–critic framework of our proposed MADPG-TDec algorithm.

where $y_{t,k}^i$ denotes the target sub-Q value

$$y_{t,k}^i = r_{t+1,k}^i + \gamma \max_{a^i, a^{-i} \in \mathcal{A}} Q_k^i\left(s_{t+1}^i, a^i, a^{-i}; \theta_{q,k}^{i,\text{tar}}\right) \quad (20)$$

and $\theta_{q,k}^i$ and $\theta_{q,k}^{i,\text{tar}}$ are the parameters of the evaluate network and target network [13]. The $\theta_{q,k}^{i,\text{tar}}$ are periodically copied from $\theta_{q,k}^i$.

The pseudocode of the MAQ-TDec is given in Algorithm 1. Here, we use neural networks to map state–action pair onto Q value, and they can also be replaced by other function approximators.

C. Multiagent DPG With Task Decomposition

For the MARL system with N_a agents and continuous action spaces, we extend the DPG algorithm in [14] with actor–critic architecture to multitask scenarios. The proposed algorithm learns multiple critics for the subtasks and jointly finds an optimal policy of each agent for its multiple objective functions. The algorithm architecture is shown in Fig. 2.

The actor–critic framework is applied to find optimal distributed policies for each agent, in which actor and critic are approximated by neural networks. However, different from the MADDPG in [6] and [38], the input of agent i's critic network includes its local observation, the action of itself, and the actions of other agents. The local observations of other agents are ignored during training and executing times because the input space will be larger when the number of agents grows, which will end up with computationally expensive. During the training step, each agent treats others as a part of the environment and assumes that the policies for other agents are fixed at that time step.

Remark 2: The actor network is the parameterized policy that takes as input the state and output the actions. The critic network is an approximator used to evaluate the expected return that takes as input the state and the selected actions, i.e., the critic network is applied to evaluate the actual Q values. For the architecture shown in Fig. 2, we apply distributed agentwise actors to approximate a joint policy. The critic networks are used to evaluate the behavior of the actors on each subtask.

The holistic objective function of agent i for a given policy π^i can be defined as

$$\mathcal{J}^i\left(\pi^i\right) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1}^i | s_0^i, \pi^i, \pi^{-i}\right] \quad (21)$$

where π^i and π^{-i} are policy of agent i and the joint policy of other agents, respectively. We first modify the DPG algorithm [14] into multiagent scenarios and then extend it to MT-MARL through task decomposition.

Lemma 1 (Multiagent DPG): For MARL with N_a agents, let Θ_π and Θ_q be parameter space for actor and critic, respectively. A policy π^i and a Q-function Q^i are parameterized by θ_π^i and θ_q^i, respectively, where $\theta_\pi^i \in \Theta_\pi$ and $\theta_q^i \in \Theta_q$. $\mathcal{J}^i(\pi^i)$ is the averaged return defined by (21). Then, the gradient of $\mathcal{J}^i(\pi^i)$ with respect to θ_π^i is given by

$$\nabla_{\theta_\pi^i} \mathcal{J}^i\left(\pi^i\right) = \mathbb{E}_{s\sim d_{\pi^i}}\left[\nabla_{\theta_\pi^i} \pi^i(s) \nabla_{a^i} Q^i\left(s, a^i, a^{-i}\right)|_{a^i=\pi^i(s)}\right] \quad (22)$$

where $d_{\pi^i}(s)$ denotes the stationary state distribution for agent i.

Proof: From [14], for a single-agent RL with continuous action space, the gradient of the objective function with respect to the policy's parameters is

$$\begin{aligned}\nabla_\theta \mathcal{J}(\pi) &= \int_{s\in\mathcal{S}} d_\pi(s) \nabla_\theta \pi_\theta(s) \nabla_a Q^\pi(s,a)|_{a=\pi_\theta(s)} ds \\ &= \mathbb{E}_{s\sim d_\pi}\left[\nabla_\theta \pi_\theta(s) \nabla_a Q^\pi(s,a)|_{a=\pi_\theta(s)}\right]. \end{aligned} \quad (23)$$

For multiple agents, we allow the independence between agents' training processes, and these agents share their actions to each other. Thus, we extend (23) to multiagent scenario

$$\begin{aligned}&\nabla_{\theta_\pi^i} \mathcal{J}^i\left(\pi^i\right) \\ &= \int_{s\in\mathcal{S}} d_{\pi^i}(s) \nabla_{\theta_\pi^i} \pi^i(s) \nabla_{a^i} Q^i\left(s, a^i, a^{-i}\right)|_{a^i=\pi^i(s)} ds \\ &= \mathbb{E}_{s\sim d_{\pi^i}}\left[\nabla_{\theta_\pi^i} \pi^i(s) \nabla_{a^i} Q^i\left(s, a^i, a^{-i}\right)|_{a^i=\pi^i(s)}\right]. \end{aligned} \quad (24)$$

□

The critic for agent i in Lemma 1 is defined as $Q^i(s_t^i, a_t^i, a_t^{-i}) = \mathbb{E}[\sum_{l=0}^{\infty} \gamma^l R_{t+l+1}^i | s_t^i, a_t^i, a_t^{-i}]$ and the Bellman equation is

$$\begin{aligned}&Q^i\left(s_t^i, a_t^i, a_t^{-i}\right) \\ &= \mathbb{E}\left[R_{t+1}^i + \gamma\, Q^i\left(s_{t+1}^i, \pi^i\left(s_{t+1}^i\right), a_{t+1}^{-i}\right) | s_t^i, a_t^i, a_t^{-i}\right] \end{aligned} \quad (25)$$

where $i = 1, \ldots, N_a$, and a_{t+1}^{-i} are other agents' joint actions on time step $t+1$ calculated by their own actors separately. When we update parameters for agent i's critic, the policies for other agents are treated as fixed. Thus, the expectation in (25) depends only on the environment and other agents' current policies. θ_q^i in Lemma 1 are optimized by minimizing the loss

$$\mathcal{L}^i\left(\theta_q^i\right) = \mathbb{E}\left[\left(Q^i\left(s_t^i, a_t^i, a_t^{-i}\right) - y_t^i\right)^2\right] \tag{26}$$

where y_t^i denotes the target Q value

$$y_t^i = R_{t+1}^i + \gamma\, Q^i\left(s_{t+1}^i, \pi^i\left(s_{t+1}^i\right), a_{t+1}^{-i}\right). \tag{27}$$

For multiple tasks scenario, by following the reward decomposition in Theorem 1, (21) can be decomposed as $\mathcal{J}^i(\pi^i) = \sum_{k=1}^M \mathcal{J}_k^i(\pi^i)$, where

$$\mathcal{J}_k^i\left(\pi^i\right) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1,k}^i | s_0^i, \pi^i, \pi^{-i}\right], \quad k = 1, \ldots, M. \tag{28}$$

The state–action value function for agent i can also be decomposed as $Q^i(s^i, a^i, a^{-i}) = \sum_{k=1}^M Q_k^i(s^i, a^i, a^{-i})$, where

$$\begin{aligned} &Q_k^i\left(s_t^i, a_t^i, a_t^{-i}\right) \\ &\quad = \mathbb{E}\left[\sum_{l=t}^{\infty} \gamma^l r_{t+l+1,k}^i | s_t^i, a_t^i, a_t^{-i}\right] \\ &\quad = \mathbb{E}\left[r_{t+1,k}^i + \gamma\, Q_k^i\left(s_{t+1}^i, \pi^i\left(s_{t+1}^i\right), a_{t+1}^{-i}\right) | s_t^i, a_t^i, a_t^{-i}\right]. \end{aligned} \tag{29}$$

Theorem 3 (MADPG-TDec): For MARL with M multiple cooperative tasks, the reward function for each agent can be decomposed as $R_t^i = \sum_{k=1}^M r_{t,k}^i$. Then, its policy gradient is given by

$$\begin{aligned} \nabla_{\theta_\pi^i} \mathcal{J}^i\left(\pi^i\right) &= \sum_{k=1}^M \nabla_{\theta_\pi^i} \mathcal{J}_k^i\left(\pi^i\right) \\ &= \sum_{k=1}^M \mathbb{E}_{s\sim d_{\pi^i}}\left[\nabla_{\theta_\pi^i} \pi^i(s) \nabla_{a^i} Q_k^i\left(s, a^i, a^{-i}\right)|_{a^i=\pi^i(s)}\right]. \end{aligned} \tag{30}$$

Proof: By using the finitely additive for the gradient operators, the proof of Theorem 3 can be achieved straightforwardly with Lemma 1. □

With the policy gradient given by (30), we can update the parameters of the actor network by gradient ascent

$$\theta_\pi^i := \theta_\pi^i + \alpha_a \nabla_{\theta_\pi^i} \mathcal{J}^i\left(\pi^i\right) \tag{31}$$

where α_a is the learning rate for the actor networks.

The parameters of subcritics for agent i are updated by minimizing the losses

$$\mathcal{L}_k^i\left(\theta_{q,k}^i\right) = \mathbb{E}\left[\left(Q_k^i\left(s_t^i, a_t^i, a_t^{-i}\right) - y_{t,k}^i\right)^2\right], \quad k = 1, \ldots, M \tag{32}$$

where $y_{t,k}^i$ denotes the target sub-Q value

$$y_{t,k}^i = r_{t+1,k}^i + \gamma\, Q_k^i\left(s_{t+1}^i, \pi^i\left(s_{t+1}^i\right), a_{t+1}^{-i}\right). \tag{33}$$

It seems that the policy gradient in (30) still suffers from the issue of symmetry breaking or equilibrium selection, where the increase and decrease of different sub-Q values happen and result in average gradient applied in (30) and prevent the policy from either increasing or decreasing. When that issue happens and the solution is not a global optimal policy, we can conclude that the objective $\mathcal{J}^i(\pi^i)$ that we get fails to well evaluate the performance of policy π^i. In this case, if we decompose the critic into multiple subcritics and optimize them individually with task-specific rewards by minimizing (32), then we can get a better objective $\mathcal{J}^i(\pi^i)$ and the corresponding policy gradient $\nabla_{\theta_\pi^i} \mathcal{J}^i(\pi^i)$ to evaluate the policy and update the parameters of the actor network, respectively. Hence, that issue can be effectively alleviated.

Algorithm 2 MADPG-TDec

Input: Agent number N_a, task number M, network architecture, episode number N_e, max trajectory length T, mini-batch size N_{batch}, soft update factor τ, discount factor γ.

Initialize: Networks' parameters, replay buffer $\mathcal{D}$, the target networks are initialized by the same parameters with current networks.

for $episode = 1$ to N_e **do**
  Reset environment and get initialized states $\{s_0^i\}_{i=1}^{N_a}$.
  for $t = 0$ to $T-1$ **do**
    Get actions: $a_t^i = \pi^i(s_t^i), i = 1, \ldots, N_a$.
    Execute $\{a_t^i\}_{i=1}^{N_a}$, then get $\{s_{t+1}^i\}_{i=1}^{N_a}$ and $\{R_{t+1}^i\}_{i=1}^{N_a}$, where $R_{t+1}^i = [r_{t+1,1}^i, \ldots, r_{t+1,M}^i]$.
    Store $\{< s_t^i, a_t^i, s_{t+1}^i, R_{t+1}^i >\}_{i=1}^{N_a}$ into $\mathcal{D}$.
    Sample N_{batch} transitions as a mini-batch data in $\mathcal{D}$.
    for $i = 1$ to N_a **do**
      for $k = 1$ to M **do**
        Update critic networks by minimizing (32) with the sampled data.
      end for
      Update actor networks using gradient ascent with the sampled data by (30) and (31).
    end for
    Update target networks:

$$\theta_{q,k}^{i\,\prime} \leftarrow \tau\theta_{q,k}^i + (1-\tau)\theta_{q,k}^{i\,\prime},$$
$$\theta_\pi^{i\,\prime} \leftarrow \tau\theta_\pi^i + (1-\tau)\theta_\pi^{i\,\prime}.$$

    $s_t^i \leftarrow s_{t+1}^i, i = 1, \ldots, N_a$.
  end for
end for

Equations (30) and (32) are used to iteratively update the actor and critic in MASs with multiple tasks. The pseudocode for the training process is given in Algorithm 2. To stabilize the training process, we apply target neural networks with a soft updating rate τ, which follows the settings in [6] and [15].

Corollary 1: For a multiagent MDP with N_a agents and M tasks, the reward function of agent i is decomposed as $R_t^i = \sum_{k=1}^M r_{t,k}^i$, and the holistic objective function is decomposed as $\mathcal{J}^i(\pi^i) = \sum_{k=1}^M \mathcal{J}_k^i(\pi^i)$. Then, the optimization for $\mathcal{J}^i(\pi^i)$ is equivalent to optimize $\mathcal{J}_k^i(\pi^i), k = 1, \ldots, M$ separately if and only if Assumption 1 holds.

Proof: According to the definition of $\mathcal{J}^i(\pi^i)$ in (21), if the optimization for $\mathcal{J}^i(\pi^i)$ is equivalent to optimize $\mathcal{J}_k^i(\pi^i)$

separately, that means $\max_\pi \mathcal{J}^i(\pi) \Leftrightarrow \max_{\pi_k} \mathcal{J}^i_k(\pi_k), k = 1, \ldots, M$, that is

$$\max_\pi \sum_{k=1}^{M} \mathcal{J}^i_k(\pi) = \sum_{k=1}^{M} \max_{\pi_k} \mathcal{J}^i_k(\pi_k), \quad k = 1, \ldots, M. \tag{34}$$

Suppose that $\pi^{i*} \in \Pi^{i*}$ and $\pi^{i*}_k \in \Pi^{i*}_k, k = 1, \ldots, M$, and then, we can get $\mathcal{J}^i_k(\pi^{i*}) = \mathcal{J}^i_k(\pi^{i*}_k)$ according to (34); hence, $\pi^{i*} \in \Pi^{i*}_k \forall k = 1, \ldots, M$, and then, $\bigcap_{k=1}^{M} \mathbf{\Pi}^{i*}_k \neq \emptyset$.

Conversely, according to Assumption 1, if $\bigcap_{k=1}^{M} \Pi^{i*}_k \neq \emptyset$ and $\forall \pi^{i*} \in \bigcap_{k=1}^{M} \Pi^{i*}_k$, then $\mathcal{J}^i_k(\pi^{i*}) = \max_\pi \mathcal{J}^i_k(\pi)$. By objective decomposition

$$\mathcal{J}^i\left(\pi^{i*}\right) = \sum_{k=1}^{M} \mathcal{J}^i_k\left(\pi^{i*}\right) = \sum_{k=1}^{M} \max_\pi \mathcal{J}^i_k(\pi) = \max_\pi \mathcal{J}^i(\pi). \tag{35}$$

Therefore, $\max_\pi \mathcal{J}^i(\pi) \Leftrightarrow \max_{\pi_k} \mathcal{J}^i_k(\pi_k), k = 1, \ldots, M$, i.e., the optimization for $\mathcal{J}^i(\pi)$ is equivalent to optimize $\mathcal{J}^i_k(\pi) \forall k = 1, \ldots, M$ separately. □

In addition, the MARL system with multiple tasks aims to achieve a policy that satisfies Pareto optimality, i.e., the policy that maximizes the subobjectives simultaneously should Pareto dominate other policies.

Corollary 2: An equivalent optimal solution π^{i*} in Corollary 1 for a MAS with M tasks is the Pareto optimal.

Proof: Since policy π^{i*} is an optimal solution in Corollary 1, then, for agent i's objective function, $\mathcal{J}^i_k(\pi^{i*}) = \max_\pi \mathcal{J}^i_k(\pi) \geq \mathcal{J}^i_k(\pi^i)\ \forall \pi^i \in \Pi^i, k = 1, \ldots, M$. Then, according to Definition 1, $\pi^{i*} \succeq \pi^i \forall \pi^i \in \Pi^i$. Thus, π^{i*} is a Pareto optimal solution according to Definition 2, i.e., $\pi^{i*} \in \mathcal{P}_{\pi^i}$. □

From the abovementioned analysis, we can conclude that the optimal policy found by MADPG-TDec is equivalent to the original MADPG algorithm. The difference between these two algorithms is critic networks and policy updating. The proposed MADPG-TDec algorithm uses multiple critics with respect to the task-specific rewards to calculate the gradient for the objective function, whereas the MADGP algorithm uses a single critic with respect to the holistic reward. For some complex control problems with function approximators, such as multilayer neural networks, it is difficult to guarantee that the policy found by the MADPG algorithm is global optimal for all tasks. However, for MADPG-TDec, the optimality of each task could be guaranteed better than MADPG.

Remark 3: Both MAQ-TDec and MADPG-TDec algorithms assume the prior knowledge of the number of subtasks and the knowledge of task rewards that are decomposed from the holistic reward by task decomposition. Without the prior knowledge of these, it is impossible to decompose the holistic reward from the environment as specific numerous parts, and we will not know how many sub-Q functions and subcritics are needed for value decomposition and which subreward should be used to approximate them. Hence, that is also the key reason why task decomposition is helpful in learning.

When the number of subcritics of MADPG-TDec does not match the number of subtasks, the sum of these subcritics may be unable to represent the holistic value function. Denote K as the number of subcritics and M as the number of subtasks. If $K < M$, we can remove the redundant critics to guarantee that the sum of subcritic is equivalent to the holistic one. If $K > M$, we can recompose the extra subrewards to match the subcritics such that the holistic task is decomposed into these subrewards and trained by these subcritic networks.

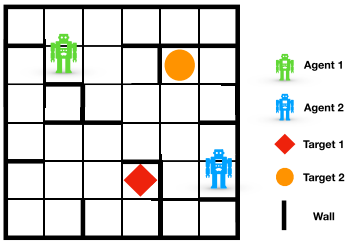


Fig. 3. Multiagent treasure hunting tasks.

In summary, the task decomposition-based learning method can be applied to many cooperative MASs with multiple tasks. We present the pseudocodes in Algorithms 1 and 2 for discrete decision-making problems and continuous control problems, respectively.

V. Simulation Results

In this section, to verify the theoretical analysis in Section IV, we evaluate the performance of proposed algorithms on both discrete and continuous control problems in different multiagent simulation environments. Our simulations run at a desktop with Intel Core i7-7700k CPU@4.20 GHz under Ubuntu 16.04 Operation System, and we apply the Tensorflow open-source toolbox [39] to realize the proposed algorithms.

A. Multiagent Treasure Hunting

We first apply Algorithm 1 (MAQ-TDec) to the multiagent treasure hunting problem with discrete action spaces. As shown in Fig. 3, there are two agents and two targets in a 6×6 grid world. These two agents aim to find the targets cooperatively without conflict. During the training period, these agents are initialized at different positions randomly.

Agent can observe the position of itself and that of other agent as its state information and takes actions from a discrete action set: $\mathcal{A} = \{\text{up, down, left, right, wait}\}$. Each agent would be rewarded with $+5$ if it finds any of these two targets. It would be rewarded with -1 if it finds nothing. If it is obstructed by the wall or collides with the other agent, it would get another -1 reward. Thus, the reward of agent i can be designed as

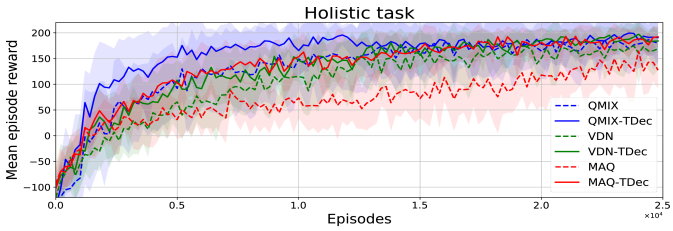
$$R^i_t = r^i_{t,\text{target1}} + r^i_{t,\text{target2}} + r^i_{t,\text{penalty}} \tag{36}$$

where $r^i_{t,\text{target1}} = \{-1, +5\}$, $r^i_{t,\text{target2}} = \{-1, +5\}$m and $r^{(i)}_{t,\text{penalty}} = \{-1, 0\}$. The tasks are represented by $r^i_{t,\text{target1}}$ and $r^i_{t,\text{target2}}$, and our decomposition training method is based on this decomposition formation, i.e., the reward function for agent i can be decomposed as: $R^i_t = r^i_{t,1} + r^i_{t,2}$, where $r^i_{t,1} = r^i_{t,\text{target1}} + 0.5 \times r^i_{t,\text{penalty}}$ and $r^i_{t,2} = r^i_{t,\text{target2}} + 0.5 \times r^i_{t,\text{penalty}}$.

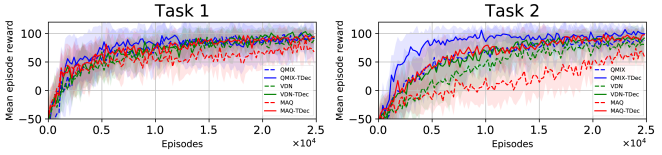
All training parameters of the compared algorithms are set according to Table I. We use two-layer MLP to approximate the Q value functions for these algorithms. The experiment

TABLE I
PARAMETER SETTINGS FOR SIMULATION *A*

Parameter	Value
Discount factor (γ)	0.99
Learning rate (α)	0.001
Batch size	64
Replay buffer size	100,000
Initialized greedy rate	0.1
Episode number (N_e)	25,000
Max trajectory length (T)	20
Hidden layer units (layer 1)	16
Hidden layer units (layer 2)	16
Agent number (N_a)	2
Task number (M)	2



(a)



(b)

Fig. 4. Team performance on simulation *A* of (a) holistic task and (b) specific subtasks.

results are shown in Fig. 4, in which the average performance of each algorithm is tested over ten trials at every 50 episodes during training time. For each trial, we calculate the accumulated rewards as the empirical performance at that training episode. For the compared Distillation, VDN, and QMIX algorithms, we use the same architecture in [20], [24], and [21]. The deep recurrent Q-network (DRQN) used in [20], [21], and [24] is replaced as the same network of MLP in Table I, which will make their performance more comparable. For the Distillation algorithm in [24], we train it in two different phases. In the first phase, the agents learn task-specific policies with taskwise rewards individually, and in the second phase, these policies are composed as one with distillation technology according to [24]. Thus, the increased episode rewards during the training time are not comparable, and we only show its final performance with others.

Fig. 4 shows the comparative results of the mean episode reward over 15 experiments. In addition, we choose the trained model for each algorithm that performs best among these 15 experiments and compare their success rate and average accumulated return over 1000 trials in Table II and Fig. 5, respectively. It can be found that the MAQ-TDec outperforms MAQ and can get almost the same performance with the VDN and QMIX. From Table II and Fig. 5, we also find that the VDN and QMIX outperform the proposed MAQ-TDec in some instances. To verify the effectiveness of task decomposition, we extend the VDN to VDN-TDec and QMIX to QMIX-TDec by decomposing their Q values in the same way, which results in significant improvements in the performance. Fig. 4(b) shows the task-specific performance of the team, and we find, especially for task 2, that the performance improvement achieved by the task decomposition-based algorithms (MAQ-TDec, VDN-TDec, and QMIX-TDec) is obvious. As a result, it is the task decomposition in the MAQ-TDec algorithm that improves the performance of the original MAQ algorithm. Furthermore, we find that the Distillation algorithm cannot perform well in this environment because the policies learned in the first phase are task-specific and local optimal. It is hard to learn a global optimal policy by data regression in the second phase, while the proposed task decomposition algorithm can avoid these task-specific local optimal policies compared with the Distillation algorithm.

B. Multiagent Navigation

We apply multiagent simulation environment in [6] to study the multiagent cooperative navigation problems. Consider that N_a agents (noted as $A_1, \ldots, A_{N_a}$) and M targets (noted as $T_1, \ldots, T_M$) are randomly initialized on different positions. The holistic task of the system is N_a agents reaching the M landmarks without collision (see Fig. 6) in a minimum time. To simplify the problem, we set $N_a = M$. The observation for each agent includes the information about the position of itself, the relative positions of targets and the other agents, and its physical movement velocity. The action for each agent consists of two components, which represents the forces in the horizontal and vertical directions, respectively. In each direction, the action is continuous and ranges from -1 to 1. To avoid the conflict that multiple agents choose the same target, the reward function should be designed properly for the holistic task.

In Fig. 6, there are three agents in different positions, and their tasks are navigating to three different targets. The system will reward an agent if any target is getting closer to the nearest agent from it. Thus, the holistic reward function for agent i is designed as

$$R_t^i = r_{t,1}^i + r_{t,2}^i + r_{t,3}^i \tag{37}$$

where

$$\begin{cases} r_{t,1}^i = -\min\{\text{dis}(T_1, A_1), \text{dis}(T_1, A_2), \text{dis}(T_1, A_3)\} + \rho_t^i \\ r_{t,2}^i = -\min\{\text{dis}(T_2, A_1), \text{dis}(T_2, A_2), \text{dis}(T_2, A_3)\} + \rho_t^i \\ r_{t,3}^i = -\min\{\text{dis}(T_3, A_1), \text{dis}(T_3, A_2), \text{dis}(T_3, A_3)\} + \rho_t^i \end{cases} \tag{38}$$

and $\rho_t^i = \{-1, 0\}$. If agent i collides with other agents, it will be flicked away with -1 penalty, i.e., $\rho_t^i = -1$; otherwise, $\rho_t^i = 0$. In (38), $\text{dis}(T_k, A_i)$ denotes the Euclidean distance from T_k to A_i.

We use two-layer neural networks for both actors and critics, and each layer's activation function is RELU [40]. Following the parameter settings in Table III, the experiment results are shown in Fig. 7. To compare with the performance of VDN and QMIX, we discretize the continuous actions from $[-1, 1]$ to $\{-1, 0, 1\}$ at each moving direction. Then, we apply the architecture of VDN and QMIX with the same neural network and

TABLE II

SUCCESS RATE FOR SIMULATION *A* WITH THE BEST PERFORMANCE

Methods	**MAQ-TDec**	MAQ	**VDN-TDec**	VDN	**QMIX-TDec**	QMIX	Distillation
Success rate (%)	**98.4**	96.4	**99.4**	99.2	**99.9**	99.0	90.10

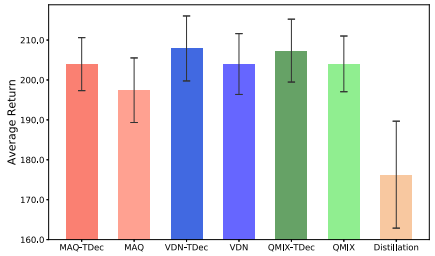


Fig. 5. Testing average return for the trained models in simulation *A* that gets the best performance.

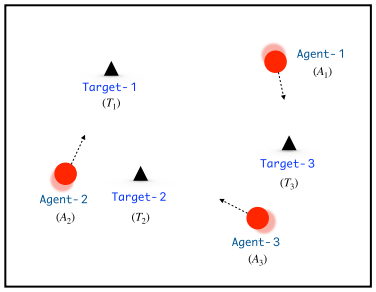


Fig. 6. Multiagent navigation problem: the holistic task for these three agents is navigating to the three targets cooperatively.

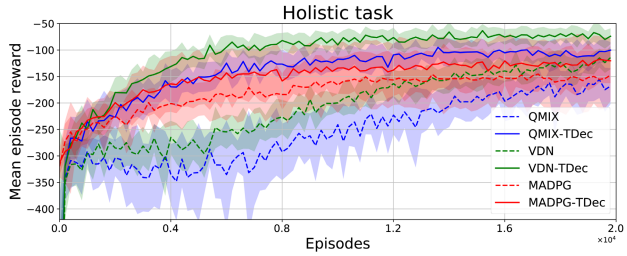
TABLE III

PARAMETER SETTINGS FOR SIMULATION *B*

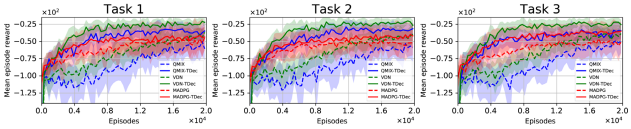
Parameter	Value
Discount factor (γ)	0.95
Learning rate (critic, α_c)	0.001
Learning rate (actor, α_a)	0.01
Soft update factor (τ)	0.001
Batch size (N_{batch})	64
Replay buffer size (M)	100,000
Initialize exploration variance (σ)	0.3
Hidden layer units (layer 1)	32
Hidden layer units (layer 2)	32
Episode number (N_e)	20,000
Max trajectory length (T)	50

take the sum of agentwise rewards as the joint team reward to learn the Q values. We also test the performance of VDN-TDec and QMIX-TDec by using the task-specific rewards, such as simulation *A*. The performance of the training models is tested at every 100 episodes, and we average the results over ten trials with maximum trajectory length as 50. Then, we get an average performance over 15 experiments with the same parameter settings but different initialized network parameters. Then, we choose the model of each algorithm that performs best and show their average accumulated return over 1000 trials in Fig. 8. In this scenario, the Distillation algorithm is hard to get comparable performance, so we have not shown its performance in Fig. 8 (as well as in the following figures).

The MADPG-TDec algorithm searches the optimal policy by three subcritics. That means, for each agent, the parameters for the actor network are updated with three critic networks with respect to the three tasks individually. While the MADPG algorithm searches the optimal policy by a single critic



(a)



(b)

Fig. 7. Team performance on simulation *B* of (a) holistic task and (b) specific subtasks.

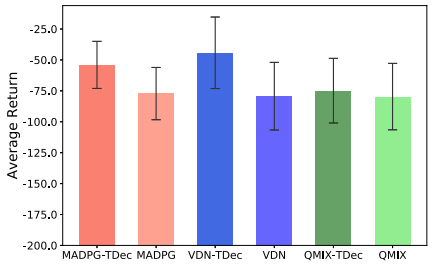


Fig. 8. Testing average return for the trained models in simulation *B* that gets the best performance.

network with respect to the holistic reward signals in (37). As shown in Figs. 7 and 8, with the same parameter settings in Table III, the proposed MADPG-TDec algorithm outperforms the MADPG, VDN, and QMIX. Fig. 7(a) and (b) shows the holistic performance and task-specific performance of the team for compared algorithms, respectively. When we apply the idea of task decomposition to VDN and QMIX, both of VDN-TDec and QMIX-TDec achieve significant improvement of the performance and even outperform MADPG-TDec. It is easy to conclude that the idea of task decomposition is useful to improve the performance of the existing algorithms.

C. Dynamic Targets Multiagent Navigation

In this case, we consider a dynamic multitarget navigation problem with four agents and four targets, which enlarges the search space and increases the difficulty of the experiment. We extend the multiagent navigation environment in [6] to a dynamic target scenario, which is shown in Fig. 9, where each target performs a full elastic collision with a specific speed in a rectangular room. Each agent can observe the position of itself, the relative positions of targets and other agents, and the velocities of itself and the moving targets. The holistic task for multiple agents is navigating to these moving targets. The reward setting is similar to (37) and (38).

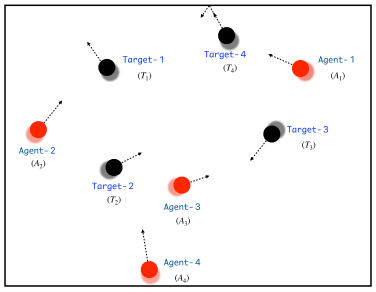


Fig. 9. Multiagent navigation problem with dynamic targets: the holistic task for these four agents is navigating to the four moving targets cooperatively.

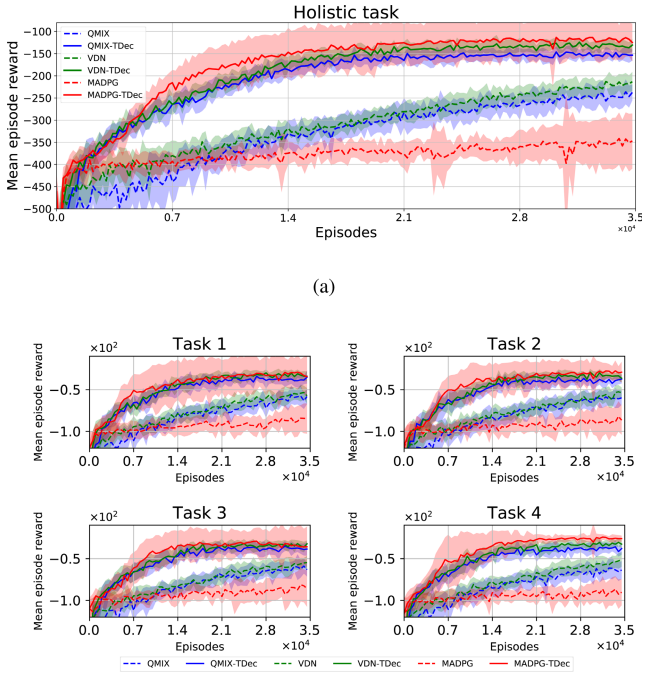
TABLE IV
PARAMETER SETTINGS FOR SIMULATION *C*

Parameter	Value
Discount factor (γ)	0.95
Learning rate (critic, α_c)	0.001
Learning rate (actor, α_a)	0.001
Soft update factor (τ)	0.001
Batch size (N_{batch})	256
Replay buffer size (M)	200,000
Initialize exploration variance (σ)	0.2
Hidden layer units (layer 1)	64
Hidden layer units (layer 2)	64
Episode number (N_e)	35,000
Max trajectory length (T)	50

The parameter settings for this experiment are listed in Table IV. Two-layer neural networks are used for the actors and critics for both MADPG-TDec and MADPG algorithms, and the activation function for each hidden layer is also RELU. In this scenario, the MADPG-TDec algorithm searches the optimal policy for each agent with four different subcritic networks using the decomposed reward signals separately. The original MADPG algorithm still searches the optimal policy with one critic network with respect to the holistic reward signals.

We test the training performance in the same way as simulation *B* and get the experiment curves in Fig. 10. It is found that the performance for the MADPG algorithm without reward decomposition units grows slowly after 2000 episodes. While, for the MADPG-TDec algorithm, it grows faster than MADPG after 3000 episodes and converges to an optimal solution for each task. For MADPG-TDec, it deals with every single task separately, and the corresponding subcritic network is easy to converge. We test the performance of VDN and QMIX in Fig. 10(a) and (b) in the same way in simulation *B*. We also find that VDN-TDec and QMIX-TDec can get significant improvement compared with VDN and QMIX, respectively. Then, we choose the trained model that performs best among the 15 experiments for each approach and get the average test performance of them over 1000 trials in Fig. 11. It is obviously found that the MADPG-TDec algorithm gets the best performance compared with others, and all the task decomposition-based approaches get a significant improvement of the performance compared with the corresponding baselines.

In conclusion, these simulation results have verified the effectiveness of the task decomposition in the proposed algorithms. The results compared with the state-of-the-art



(a)

(b)

Fig. 10. Team performance on simulation *C* of (a) holistic task and (b) specific subtasks.

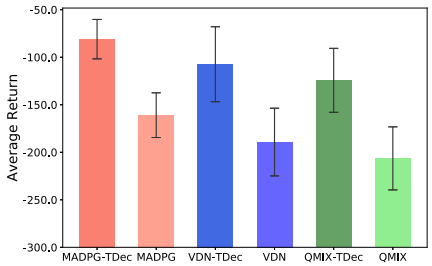


Fig. 11. Testing average return for the trained models in simulation *C* that get the best performance.

methods have also demonstrated the theoretical analysis in Section IV empirically.

VI. CONCLUSION

This article proposes the task decomposition method for cooperative MASs with multiple tasks. The theoretical analysis for the proposed method ensures convergence and superiority compared with the traditional methods without the task decomposition unit. We apply the task decomposition to the traditional algorithms to learn an optimal policy for each agent with discrete and continuous action spaces. The key idea for the proposed algorithms is decomposing the holistic reward signal for each agent into several subsignals with respect to subtasks and solving the subtasks separately to achieve an optimized solution. Finally, three different cooperative multiagent problems are tested to verify the effectiveness of the task decomposition. The simulation experiments show the consensus results with the theoretical analysis.

REFERENCES

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

[2] M. Khalili, X. Zhang, Y. Cao, M. M. Polycarpou, and T. Parisini, "Distributed fault-tolerant control of multiagent systems: An adaptive learning approach," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 2, pp. 420–432, Feb. 2020.

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

[4] J. Godoy, T. Chen, S. J. Guy, I. Karamouzas, and M. Gini, "ALAN: Adaptive learning for multi-agent navigation," *Auto. Robots*, vol. 42, no. 8, pp. 1543–1562, Dec. 2018.

[5] H. M. Schwartz, *Multi-Agent Machine Learning: A Reinforcement Approach*. Hoboken, NJ, USA: Wiley, 2014.

[6] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6379–6390.

[7] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proc. IEEE*, vol. 95, no. 1, pp. 215–233, Jan. 2007.

[8] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 38, no. 2, pp. 156–172, Mar. 2008.

[9] Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge, U.K.: Cambridge Univ. Press, 2008.

[10] D. Bloembergen, K. Tuyls, D. Hennes, and M. Kaisers, "Evolutionary dynamics of multi-agent learning: A survey," *J. Artif. Intell. Res.*, vol. 53, pp. 659–697, Aug. 2015.

[11] L. Zheng *et al.*, "MAgent: A many-agent reinforcement learning platform for artificial collective intelligence," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 8222–8223.

[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[13] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[14] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. A. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. 31th Int. Conf. Mach. Learn.*, 2014, pp. 387–395.

[15] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*. [Online]. Available: https://arxiv.org/abs/1509.02971

[16] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1008–1014.

[17] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 2974–2982.

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

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

[20] P. Sunehag *et al.*, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *Proc. 17th Int. Conf. Auto. Agents MultiAgent Syst.*, 2018, pp. 2085–2087.

[21] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. N. Foerster, and S. Whiteson, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 4292–4301.

[22] B. Marthi, "Automatic shaping and decomposition of reward functions," in *Proc. 24th Int. Conf. Mach. Learn.*, 2007, pp. 601–608.

[23] H. Van Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes, and J. Tsang, "Hybrid reward architecture for reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5392–5402.

[24] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian, "Deep decentralized multi-task multi-agent reinforcement learning under partial observability," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 2681–2690.

[25] C. J. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[26] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[27] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. 33nd Int. Conf. Mach. Learn.*, 2016, pp. 1995–2003.

[28] H. V. Hasselt, "Double Q-learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2010, pp. 2613–2621.

[29] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1057–1063.

[30] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Proc. Mach. Learn.* Amsterdam, The Netherlands: Elsevier, 1994, pp. 157–163.

[31] S. Zuo, Y. Song, F. L. Lewis, and A. Davoudi, "Optimal robust output containment of unknown heterogeneous multiagent system using off-policy reinforcement learning," *IEEE Trans. Cybern.*, vol. 48, no. 11, pp. 3197–3207, Oct. 2017.

[32] V. G. Lopez and F. L. Lewis, "Dynamic multiobjective control for continuous-time systems using reinforcement learning," *IEEE Trans. Autom. Control*, vol. 64, no. 7, pp. 2869–2874, Jul. 2019.

[33] K. Van Moffaert and A. Nowé, "Multi-objective reinforcement learning using sets of Pareto dominating policies," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3483–3512, 2014.

[34] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 525–536.

[35] C. Liu, X. Xu, and D. Hu, "Multiobjective reinforcement learning: A comprehensive overview," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 45, no. 3, pp. 385–398, Mar. 2015.

[36] J. Karlsson, "Task decomposition in reinforcement learning," in *Proc. AAAI Spring Symp. Goal-Driven Learn.* Stanford, CA, USA, 1994, pp. 46–53.

[37] F. A. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs* (Springer Briefs in Intelligent Systems). Cham, Switzerland: Springer, 2016.

[38] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Basar, "Fully decentralized multi-agent reinforcement learning with networked agents," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 5867–5876.

[39] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th Symp. Oper. Syst. Design Implement.*, 2016, pp. 265–283.

[40] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn.*, 2010, pp. 807–814.

Changyin Sun (Senior Member, IEEE) received the B.S. degree in applied mathematics from the College of Mathematics, Sichuan University, Chengdu, China, in 1996, and the M.S. and the Ph.D. degrees in electrical engineering from Southeast University, Nanjing, China, in 2001 and 2004, respectively.

He is currently a Professor with the School of Automation, Southeast University. His current research interests include intelligent control, flight control, and optimal theory.

Dr. Sun is also an Associate Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, *Neural Processing Letters*, and the IEEE/CAA JOURNAL OF AUTOMATICA SINICA.

Wenzhang Liu received the B.S. degree in engineering from Jilin University, Changchun, China, in 2016. He is currently pursuing the Ph.D. degree in control science and engineering with the School of Automation, Southeast University, Nanjing, China.

His research interests include machine learning, deep reinforcement learning, optimal control, and multiagent cooperative control.

Lu Dong (Member, IEEE) received the B.S. degree from the School of Physics, Southeast University, Nanjing, China, and the Ph.D. degree from the School of Automation, Southeast University, Nanjing, in 2012 and 2017, respectively.

She is currently an Associate Professor with the College of Electronics and Information Engineering, Tongji University, Shanghai, China. Her current research interests include adaptive dynamic programming, event-triggered control, and nonlinear system control and optimization.