Original Article

The International Journal of Robotics Research
2025, Vol. 0(0) 1–30

DOI: 10.1177/02783649241306124
journals.sagepub.com/home/ijr

Asynchronous multi-agent deep reinforcement learning under partial observability

Yuchen Xiao, Weihao Tan, Joshua Hoffman, Tian Xia and Christopher Amato

Abstract

The state-of-the-art multi-agent reinforcement learning (MARL) methods provide promising solutions to a variety of complex problems. Yet, these methods all assume that agents perform primitive actions in a synchronized manner, making them impractical for long-horizon real-world multi-robot tasks that inherently require robots to asynchronously reason about action selection at varying time durations. To solve this problem, we first propose a group of value-based cooperative MARL approaches for asynchronous execution using temporally extended macro-actions*. Here, agents perform asynchronous learning and decision-making with macro-action-value functions in three paradigms: decentralized learning and control, centralized learning and control, and centralized training for decentralized execution (CTDE). Building on the above work, we formulate a set of macro-action-based policy gradient algorithms under the three training paradigms, where agents directly optimize their parameterized policies in an asynchronous manner. We evaluate our methods both in simulation and on real robots over a variety of realistic domains. Empirical results demonstrate the effectiveness of our algorithms for learning high-quality and asynchronous solutions with macro-actions in large multi-agent problems that were previously unsolvable via primitive-action-based approaches. The proposed approaches represent the first general MARL methods for temporally extended actions and serve as the foundation for future methods in the area.*

Keywords

Multi-agent, reinforcement learning, macro-actions

Received 15 October 2023; Revised 30 June 2024; Accepted 20 September 2024

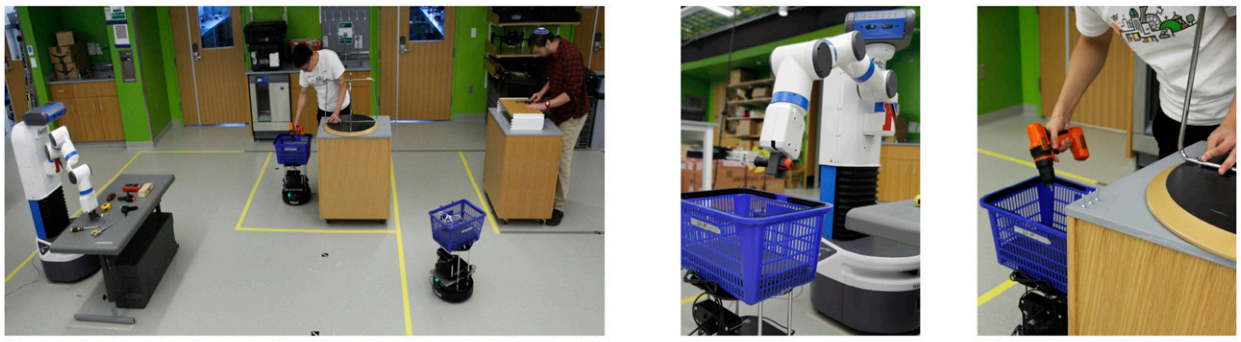
1. Introduction

More and more autonomous robots are (going to be) deployed in a variety of real-world applications. Examples include office service (Ahn et al., 2022), package delivery (Murray and Raj, 2020), and agriculture inspection (Liu et al., 2018), search and rescue (Queralta et al., 2020), autonomous vehicles (Rosenband, 2017; Tang, 2019), sports (Jolly et al., 2007; Liu et al., 2021), and others. For example, consider an assembly warehouse (Figure 1(a)) where a team of autonomous robots is assisting two humans by delivering tools. In order to support humans more efficiently, robots have to be able to predict when each human will need each tool and collaborate with each other to search for tools on a table (Figure 1(a)), pass tools (Figure 1(b)), and deliver them (Figure 1(c)). Performing these high-quality coordination behaviors in large, stochastic, and uncertain environments is challenging for the robots because it requires the robots to operate asynchronously using local information while reasoning about cooperation between teammates.

Multi-agent reinforcement learning (MARL) is a promising framework to generate solutions for these kinds of multi-robot problems. Recently, by leveraging deep neural networks to deal with large state and observation input, deep MARL has solved many challenging multi-agent problems. Unfortunately, the state-of-the-art deep MARL methods (Foerster et al., 2018; Iqbal and Sha, 2019; Lowe et al., 2017; Omidshafiei et al., 2017b; Rashid et al., 2018, 2020; Son et al., 2019; Su et al., 2021; Wang et al., 2021b, 2021c; Wang and Dong, n.d.) struggle to solve large-scale real-world multi-robot problems that involve long-term reasoning and asynchronous behavior, because they

Khoury College of Computer Science, Northeastern University, Boston, MA, USA

Corresponding author:
Yuchen Xiao, Khoury College of Computer Science, Northeastern University, 360 Huntington Ave, Boston, MA 02115, USA.
Email: xiao.yuch@northeastern.edu



(a) Robots serve two humans with tool delivery. (b) Tool Passing. (c) Tool Delivery.

Figure 1. Example of a real-world multi-robot tool delivery task.

were developed for cases where agents synchronously execute primitive actions at every time step.

Temporally-extended actions have been widely used in both learning and planning to improve scalability and reduce complexity in single-robot domains. For example, they have come in the form of motion primitives (Dalal et al., 2021; Stulp and Schaal, 2011), skills (Konidaris et al., 2011, 2018), spatial action maps (Wu et al., 2020), or macro-actions (He et al., 2010; Hsiao et al., 2010; Lee et al., 2021; Theocharous and Kaelbling, 2004). The idea of temporally-extended actions has also been incorporated into multi-agent approaches. In particular, we consider the *Macro-Action Decentralized Partially Observable Markov Decision Process* (MacDec-POMDP) (Amato et al., 2014, 2019).[1]

The MacDec-POMDP is a general model for cooperative multi-agent problems with partial observability and potentially different action durations. As a result, agents can start and end macro-actions at different time steps so decision-making can be asynchronous. MacDec-POMDPs assume the macro-actions are given (like primitive methods assume the primitive actions are given). This is well motivated by the fact that, in real-world multi-robot systems, each robot is already equipped with certain controllers (e.g., a navigation controller, and a manipulation controller) that can be modeled as macro-actions (Amato et al., 2015a; Omidshafiei et al., 2017; Wu et al., 2021a; Xiao and Hoffman, n.d.).

The MacDec-POMDP framework has shown strong scalability with planning-based methods (Amato et al., 2015a, 2015; Hoang et al., 2018; Omidshafiei et al., 2016, 2017). These methods allowed complex solutions to be generated for multi-robot problems ranging from warehouse (Amato et al., 2015b), logistics (Omidshafiei et al., 2016), and aerial delivery (Omidshafiei et al., 2017a). As planning methods, the approaches assume the problem model is known, but what we propose are model-free RL methods.

While several hierarchical multi-agent reinforcement learning (MARL) approaches have been developed, they don't typically address asynchronicity since they assume agents have high-level decisions with the same duration (de Witt et al., 2019; Han et al., 2019; Nachum et al., 2019; Wang et al., 2020b, 2021a; Xu et al., 2021; Yang et al., 2020a). Notably, none of them provides a general formulation for multi-agent reinforcement learning that allows agents to asynchronously learn and execute.

The focus of our proposed algorithms is then on learning asynchronous high-level policies over macro-actions. Our contributions can be categorized into two groups:

- **Value-based frameworks**. We first propose asynchronous value-based macro-action algorithms for the three major classes of MARL: decentralized training and execution, centralized training and execution, and centralized training for decentralized execution (CTDE) (Kraemer and Banerjee, 2016; Oliehoek et al., 2008). These approaches are based on Deep Q-Networks (DQN) (Mnih et al., 2015) but require new buffers and updates to account for the multi-agent asynchronous decision-making.
- **Actor-critic-based frameworks**. Building upon the value-based frameworks, we also develop a set of asynchronous macro-action-based actor-critic methods that generalize their primitive-action counterparts for the three MARL classes. Here, we develop new approaches to deal with asynchronicity. For example, current primitive-action-based multi-agent actor-critic methods typically use a centralized critic to optimize each decentralized actor. However, the asynchronous joint macro-action execution from the centralized perspective could be very different from each agent's decentralized perspective due to varying completion times. To this end, we first present a Naive Independent Actor with Centralized Critic (Naive IACC) method that naively uses a joint macro-action-value function as the critic for each actor's policy gradient estimation (Section 3.6); and then propose a novel Independent Actor with Individual Centralized Critic (Mac-IAICC) method that learns individual critics using centralized information to address the above challenge (Section 3.6.1).

These methods represent all the major classes of MARL algorithms and serve as a foundation for extending

primitive-action methods to the asynchronous case. To our knowledge, this is the first general formalization of macro-action-based multi-agent frameworks under the three state-of-the-art training paradigms that allow multiple agents to asynchronously learn and execute.

We evaluate our proposed frameworks on diverse macro-action-based multi-robot problems: a benchmark Box Pushing domain, a variant of the Overcooked domain (Wu et al., 2021b), and a large warehouse service domain. Experimental results: (1) demonstrate that our methods are able to learn high-quality solutions while primitive-action-based methods cannot; (2) validate the proposed macro-action-based CTDE Q-learning approaches can learn better decentralized policies than fully decentralized learning methods in most of the domains; and (3) show the strength of Mac-IAICC for learning decentralized policies over Naive IAICC and Mac-IAC. Overall, we find that Mac-IAICC is more robust and scalable than other proposed algorithms, achieving the best overall performance in learning decentralized policies, while the utility of value-based approaches is very domain-dependent. Additionally, decentralized policies learned by using Mac-IAICC are successfully deployed on real robots to solve two warehouse tool delivery tasks in an efficient way.

This work extends our earlier conference papers (Amato et al., 2015; Xiao et al., 2022; Xiao and Hoffman, n.d.) in three ways: (1) we present all proposed approaches in a coherent and systematic fashion; (2) we conduct extensive extra simulated experiments to compare the two families of algorithms with a deep analysis of their different pros and cons; and (3) we showcase new real-robot experiments in a warehouse tool delivery scenario, where a team of robots shows more complex and interesting collaborative behaviors.

2. Background

This section introduces the formal definitions of the Dec-POMDP and the MacDec-POMDP and provides an overview of single-agent and multi-agent reinforcement learning algorithms with primitive actions.

2.1. Dec-POMDPs

The decentralized partially observable Markov decision processes (Dec-POMDP) (Oliehoek and Amato, 2016) is a general model for fully cooperative multi-agent tasks, where agents make decisions in a decentralized way based on only local information. A Dec-POMDP is formally defined by a tuple $\langle I, S, A, \Omega, T, O, R, \mathbb{H}, \gamma \rangle$, where I is a set of agents; S is the environmental state space; $A = \times_{i \in I} A_i$ is the joint primitive-action space over each agent's primitive-action set A_i; $\Omega = \times_{i \in I} \Omega_i$ is the joint primitive-observation space over each agent's primitive-observation set Ω_i. At every time step, under a state s, agents synchronously execute a joint primitive-action $\vec{a} = \times_{i \in I} a_i$, each individually selected by an agent using a policy $\pi_i : H_i^A \rightarrow \Delta A_i$, a mapping from local primitive observation-action history H_i^A to primitive-action execution probabilities. The environment then transits to a new state s' according to a state transition function $T(s, \vec{a}, s') = P(s'|s, \vec{a})$. Agents receive a global reward issued by a reward function $R : S \times A \rightarrow \mathbb{R}$, and obtain a joint primitive-observation $\vec{o} = \times_{i \in I} o_i$ drawn from an observation function $O(\vec{o}, \vec{a}, s') = P(\vec{o} \mid \vec{a}, s')$ in state s'. The objective of solution methods is to find a joint policy $\vec{\pi} = \times_i \pi_i$ that optimizes the expected sum of discounted rewards from an initial state s_0:

$$V^{\vec{\pi}}(s_{(0)}) = \mathbb{E}\left[\sum_{t=0}^{\mathbb{H}-1} \gamma^t r(s_t, \vec{a}_t) \mid s_0, \vec{\pi}\right] \tag{1}$$

where $\gamma \in [0, 1]$ is a discount factor, and $\mathbb{H}$ is the number of (primitive) time steps until the problem terminates (the horizon).

2.2. MacDec-POMDPs

The macro-action decentralized partially observable Markov decision process (MacDec-POMDP) (Amato et al., 2014, 2019) incorporates the *option* framework (Sutton et al., 1999) into the Dec-POMDP by defining each agent's macro-action as a tuple $m_i = \langle I_{m_i}, \pi_{m_i}, \beta_{m_i} \rangle$, where the initiation set $I_{m_i} \subset H_i^M$ defines how to initiate a macro-action based on macro-observation-action history H_i^M at the high-level; $\pi_{m_i} : H_i^A \rightarrow \Delta A_i$ is the low-level policy for the execution of a macro-action; and a stochastic termination function $\beta_{m_i} : H_i^A \rightarrow [0, 1]$ determines how to terminate a macro-action based on primitive-observation-action history H_i^A at the low-level. A MacDec-POMDP is thus formally defined by a tuple $\langle I, S, A, M, \Omega, \zeta, T, O, Z, R, \mathbb{H}, \gamma \rangle$, where $I, S, A, \Omega, T, O, R, \mathbb{H}$ and γ remain the same definitions as in the Dec-POMDP; $M = \times_{i \in I} M_i$ is the joint macro-action space over each agent's macro-action space M_i; $\zeta = \times_{i \in I} \zeta_i$ is the joint macro-observation space over each agent's macro-observation space ζ_i; and $Z = \{Z_i\}_{i \in I}$ is a set of macro-observation likelihood models. During execution, each agent independently selects a macro-action m_i using a high-level policy $\Psi_i : H_i^M \rightarrow \Delta M_i$, a mapping from macro-observation-action history to macro-action execution probabilities, and captures a macro-observation $z_i \in \zeta_i$ according to the macro-observation probability function $Z_i(z_i, m_i, s') = P(z_i|m_i, s')$ when the macro-action terminates in a state s'. The objective of solving MacDec-POMDPs with finite horizon is to find a joint high-level policy $\vec{\Psi} = \times_{i \in I} \Psi_i$ that maximizes the value:

$$V^{\vec{\Psi}}(s_{(0)}) = \mathbb{E}\left[\sum_{t=0}^{\mathbb{H}-1} \gamma^t r(s_t, \vec{a}_t) \mid s_0, \vec{\pi}, \vec{\Psi}\right] \tag{2}$$

2.3. Single-agent reinforcement learning

We focus on model-free reinforcement learning (RL), where the agent aims to learn an optimal policy by interacting with the environment without explicit world models (e.g., T, O, and R) as prior knowledge. Model-free RL methods can be categorized into two classes: (a) value-based approaches that learn the values of actions and select actions based on the learned values; and (b) policy gradient approaches that directly learn a parameterized policy to select actions. In this section, we review the representative single-agent deep RL algorithms over the two classes, where deep neural networks are used as function approximators and policies.

2.3.1. DQN, double-DQN, and DRQN. Q-learning (Watkins and Dayan, 1992) is a popular model-free method to optimize a policy π by iteratively updating an action-value function $Q(s, a)$. Deep Q-networks (DQN) (Mnih et al., 2015) extend Q-learning to include a deep neural net as a function approximator. DQN learns $Q_\theta(s, a)$, parameterized with θ, by minimizing the *temporal-difference* (TD) error: $\mathcal{L}(\theta) = \mathbb{E}_{<s,a,s',r>\sim\mathcal{D}}\left[(y - Q_\theta(s,a))^2\right]$, where $y = r + \gamma \arg\max_{a'} Q_{\theta^-}(s', a')$. A target action-value function Q_{θ^-} (θ^- is an outdated copy of θ) and an experience replay buffer $\mathcal{D}$ (Lin, 1992) are implemented for stable learning. In order to deal with the maximum bias, the idea behind Double Q-learning (Hasselt, 2010) is generalized to DQN, called Double DQN, by rewriting the target value calculation as $y = r + \gamma Q_{\theta^-}(s', \arg\max_{a'} Q_\theta(s', a'))$ (Hasselt et al., 2016). In fully observable environments, the action-value function is expressed as $Q(s, a)$, where s represents the state. However, in partially observable environments, the agent cannot directly observe the state. In such cases, Deep Recurrent Q-Networks (DRQN) is proposed to handle single agent tasks with partial observability (Hausknecht and Stone, 2015), where a recurrent layer (LSTM (Hochreiter and Schmidhuber, 1997)) is applied to maintain an internal hidden state from the agent's observation-action history h. The corresponding action-value function $Q_\theta(h, a)$ is then updated by minimizing the following loss: $\mathcal{L}(\theta) = \mathbb{E}_{<o,a,o',r>\sim\mathcal{D}}[(y - Q_\theta(h,a))^2]$, where $y = r + \gamma \max_{a'} Q_{\theta^-}(h', a')$, and $h' = hao'$ represents the next observation-action history. In our work, we incorporate DRQN with Double DQN to learn macro-action-based value functions. This is done for value-based algorithms presented in Section 3.1, Section 3.2, and Section 3.3.

2.3.2. Actor-critic policy gradient. In single-agent reinforcement learning, the *policy gradient theorem* (Sutton et al., 2000) formulates a principled way to optimize a parameterized policy π_θ via gradient ascent on the policy's performance defined as $J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$. In POMDPs, the gradient w.r.t. parameters of an observation-action history-based policy $\pi_\theta(a|h)$ is expressed as: $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a \mid h) Q^{\pi_\theta}(h, a)]$, where h is maintained by an RNN in the policy network (Hausknecht and Stone, 2015). The actor-critic framework (Konda and Tsitsiklis, 2000) learns an on-policy action-value function $Q_\phi^{\pi_\theta}(h, a)$ (critic) via TD learning (Sutton, 1988) to approximate the action-value for the policy (actor) updates. Variance reduction is commonly achieved by training a history-value function $V_{\mathbf{w}}^{\pi_\theta}(h)$ and using it as a baseline (Weaver and Tao, 2001) as well as bootstrapping to estimate the action-value. Accordingly, the policy gradient can be written as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a \mid h) A(h, a)] \tag{3}$$

$$A(h, a) = \left(r + \gamma V_{\mathbf{w}}^{\pi_\theta}(h') - V_{\mathbf{w}}^{\pi_\theta}(h)\right) \tag{4}$$

and r is the immediate reward received by the agent at the corresponding time step.

2.4. Multi-agent reinforcement learning

We consider fully cooperative multi-agent reinforcement learning (MARL), where multiple agents interact with the same environment by perceiving input and selecting actions as well as considering the effect of each other in order to optimize the global return. In this section, we introduce three standard MARL training paradigms: centralized learning, decentralized learning, and centralized training for decentralized execution (CTDE), and we also discuss the corresponding algorithms under each paradigm.

2.4.1. Centralized learning and execution. Perhaps the most straightforward way to solve fully cooperative MARL problems is centralized learning and execution. Specifically, all agents are treated as a single big agent to learn a centralized policy $\pi(\vec{a} \mid \vec{h})$, a mapping from the joint observation-action history space to the joint action space, and all the single-agent RL algorithms can be directly applied here. In theory, with access to the joint information over agents, the centralized policy learned in this training paradigm possesses a convergence guarantee to the globally optimal behavior. However, in practice, this framework suffers from two fundamental challenges: (a) the joint action space exponentially increases with respect to the number of agents, which potentially causes learning to be very slow and likely converges to a local optimum due to approximation error on action-values; and (b) in order to perform centralized control, it requires fast and perfect online communication over agents, which is often impossible to achieve in many real-world settings.

2.4.2. Decentralized learning and execution. Because of the aforementioned issues in the fully centralized case, having a decentralized policy for each agent is preferable, where each agent independently makes decisions based on only local information.

Independent Q-Learning (IQL) (Tan, 1993) is the simplest approach to learn decentralized policies for agents.

It extends Q-learning to multi-agent scenarios by allowing each agent to independently learn its own action-value function. In partially observable environments, each agent i learns a history-action value function as $Q_{\theta_i}(h_i, a_i)$ using DRQN. While decentralized policies can be directly learned in this simple decomposition manner, each independent learner suffers from several innate limitations: the difficulty in achieving efficient credit assignment as each agent maintains Q-values only for individual actions but receives global rewards depending on joint actions; the dilemma of the environmental non-stationarity from a local perspective caused by the existence of other learning agents; and the tendency to settle at a local optimum (a shadowed equilibrium (Fulda and Ventura, 2007)) due to rare information sharing over agents, resulting in agents' local best choices being suboptimal system behavior.

Decentralized Hysteresis DRQN (Dec-HDRQN) (Omidshafiei et al., 2017b) is one representative decentralized learning method to improve solution quality in Dec-POMDPs. It combines Hysteretic Q-learning (Matignon et al., 2007) with DRQN, where each agent uses two learning rates α and β to update its own action-value function. Specifically, α is a normal learning rate when the TD error is positive, and β is a smaller learning rate used otherwise. This facilitates multi-agent learning by making each agent robust against negative updating due to teammates' mistakes. Meanwhile, a new replay buffer called *Concurrent Experience Replay Trajectories* (CERTs) is introduced to assist with the non-stationarity issue, by sampling concurrent experiences for training, which encourages each agent's policy to be optimized in the same direction.

Independent Actor-Critic (IAC) (Foerster et al., 2018) is a straightforward extension of the single-agent A2C to multi-agent cases. Similar to IQL, in this framework, each agent independently optimizes its own actor π_{θ_i} and critic $V_{\mathbf{w}_i}^{\pi_{\theta_i}}$ purely based on local experiences. Accordingly, the independent policy gradient is formulated as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\vec{\pi}_\theta}[\nabla_{\theta_i} \log \pi_{\theta_i}(a_i|h_i) A(a_i, h_i)] \tag{5}$$

where

$$A(a_i, h_i) = r + V_{\mathbf{w}_i}^{\pi_{\theta_i}}(h_i') - V_{\mathbf{w}_i}^{\pi_{\theta_i}}(h_i) \tag{6}$$

and a shared reward r over agents is assigned by the global reward function R. Although IAC may sometimes work in practice, it still suffers from the same inherent issues mentioned above in IQL. Nevertheless, an essential attribute of independent learning is that agents are able to conduct fully online learning.

2.4.3. Centralized training for decentralized execution. In recent years, centralized training for decentralized execution (CTDE) has shown considerable promise in learning high-quality decentralized policies in Dec-POMDPs. To address the main difficulties encountered in independent learning, CTDE provides agents with access to global information during offline training while maintaining decentralized online execution based on local information. This paradigm is potentially more feasible to solve real-world multi-agent tasks, where the policies are first trained in a simulator and then deployed on the real system.

Value Function Factorization is one of the popular CTDE implementations that decouples a joint Q-value function into individual Q-value functions as each agent's policy (Mahajan et al., 2019; Rashid et al., 2018, 2020; Wang et al., 2021c; Wang and Dong, n.d.), naturally avoiding the exponential size of the joint action space. This kind of architecture is scalable to large multi-agent problems in terms of the number of agents. More concretely, each agent optimizes its own Q-net by minimizing the following TD loss of a joint but factored Q-net:

$$\mathcal{L}(\vec{\theta}, \psi) = \mathbb{E}_D\left[\left(y^{tot} - Q_{\vec{\theta},\psi}^{tot}(\mathbf{x}, \vec{h}, \vec{a})\right)^2\right] \tag{7}$$

where

$$Q_{\vec{\theta},\psi}^{tot}(\mathbf{x}, \vec{h}, \vec{a}) = f_\psi\left(\mathbf{x}, \{Q_{\theta_i}(h_i, a_i)\}_{i=1}^N\right) \tag{8}$$

$$y^{tot} = r + \gamma \max_{\vec{a}'} Q_{\vec{\theta}^-,\psi^-}^{tot}(\mathbf{x}', \vec{h}', \vec{a}') \tag{9}$$

and $\mathbf{x}$ represents extra accessible global signals (e.g., the environment state). However, it is important to note that, in these methods, the function f_ψ enforces a particular constraint on the relationship between the centralized Q-values and decentralized Q-values (e.g., a linear summation constraint, a nonlinear monotonic constraint, or other weighted constraints). These constraints actually place different representational limitations on joint Q-value functions, such as the true joint Q-value function of a given domain cannot be represented with these constraints.

Independent Actor with Centralized Critic (IACC) is another widespread exploitation of the CTDE paradigm. The state-of-the-art MARL policy gradient approaches (Du et al., 2019; Foerster et al., 2018; Iqbal and Sha, 2019; Lowe et al., 2017; Su et al., 2021; Wang et al., 2020, 2021b; Yang et al., 2020b; Zhou et al., 2020) utilize IACC in variant ways and have achieved significant successes in solving many challenging multi-agent tasks. The vital idea of the IACC framework is to train a *centralized critic* that is allowed to capture global information, and then use it to direct the optimization of each decentralized actor that conditions on only local information. The resulting policy gradient can be formulated as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\vec{\pi}_\theta}\left[\nabla_{\theta_i} \log \pi_{\theta_i}(a_i|h_i) Q_\phi^{\vec{\pi}_\theta}(\mathbf{x}, \vec{a})\right] \tag{10}$$

where the centralized critic, $Q_\phi^{\vec{\pi}_\theta}(\mathbf{x}, \vec{a})$, is updated in an on-policy learning way by minimizing the following loss:

$$\mathcal{L}(\phi) = \mathbb{E}_{\vec{\pi}_\theta}\left[\left(Q_\phi^{\vec{\pi}_\theta}(\mathbf{x}, \vec{a}) - y\right)^2\right] \tag{11}$$

where

$$y = r + \gamma Q_{\phi^-}^{\vec{\pi}_{\theta^-}}(\mathbf{x}', a'_1, \ldots a'_i \ldots, a'_n)|_{a'_i \sim \pi_{\theta_i^-}(h'_i)} \quad (12)$$

and each agent's target policy $\pi_{\theta_i^-}$ is used to sample the next action to compute the target prediction in order to further stabilize the learning. By accessing all agents' actions, this centralized critic is favored for its stationary learning targets and overcomes the major environmental non-stationary issue in independent learning. Additionally, many variants of global information can be included in $\mathbf{x}$, such as a true environmental state, a joint observation, a joint action-observation history, or even certain mixed combinations, which are helpful in facilitating the update of decentralized policies to optimize global cooperative performance. Apart from these positive effects of using a centralized critic, it certainly introduces extra variance in each agent's decentralized policy gradient estimation depending on other agents' actions (Lyu et al., 2021; Wang et al., 2021b). Therefore, we consider the version of IACC with a joint history-value function as the critic to reduce the variance, and the decentralized policy gradient can be rewritten as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\vec{\pi}_{\vec{\theta}}}[\nabla_{\theta_i} \log \pi_{\theta_i}(a_i \mid h_i) A(\mathbf{x}, \vec{a})] \quad (13)$$

$$A(\mathbf{x}, \vec{a}) = r + \gamma V_{\mathbf{w}}^{\vec{\pi}_{\vec{\theta}}}(\mathbf{x}') - V_{\mathbf{w}}^{\vec{\pi}_{\vec{\theta}}}(\mathbf{x}) \quad (14)$$

3. Approach

Multi-agent deep reinforcement learning with asynchronous decision-making and macro-actions is more challenging as it is difficult to determine *when* to update each agent's policy and *what* information to use. Such asynchronicity over agents motivates a need for a principled way of updating values and maintaining replay buffers. In this section, we first introduce the formulations for learning macro-action-value functions in fully decentralized (Section 3.1) and fully centralized (Section 3.2) manners based on deep Q-networks and show the details of two replay buffers designed for each case accordingly. In order to learn better decentralized policies with asynchronous execution, we then present the first value-based CTDE algorithms with macro-actions (Section 3.3). Although these asynchronous value-based approaches give us the base to learn macro-action value functions, they do not directly extend to the policy gradient case, particularly in the case of centralized training for decentralized execution (CTDE). We thus continue proposing a group of on-policy macro-action-based multi-agent actor-critic methods for decentralized learning (Section 3.4), centralized learning (Section 3.5), and CTDE (Section 3.6).

The algorithms proposed in this paper assume the existence of macro-actions, which can be either pre-defined by humans or learned beforehand (e.g., navigation controller for moving to a specific waypoint and manipulation controller for object pick and place). The algorithms then focus on learning policies over these macro-actions. All proposed algorithms are model-free RL methods, meaning that agents do not have prior knowledge about state space, observation space, transition dynamics, and the reward model. All agents have to directly learn from the reward signals and transitions they experience through trial and error.

3.1. Macro-action-based decentralized Q-learning (Mac-Dec-Q)

In the decentralized case, each agent only has access to its own macro-actions and macro-observations as well as the joint reward at each time step. As a result, there are several choices for how information is maintained. For example, each agent could maintain exactly the information mentioned above (as seen on the left side of Figure 2), the time-step information can be removed (losing the duration information), or some other representation could be used that explicitly calculates time. We choose the middle approach (see Appendix B for a theoretical analysis of this choice). As a result, updates only need to take place for each agent after the completion of its own macro-action, and we introduce a replay buffer based on *Macro-Action Concurrent Experience Reply Trajectories* (Mac-CERTs) visualized in Figure 2.

More concretely, under a macro-action-observation history h_i, each agent independently selects a macro-action m_i using a macro-action-based decentralized policy $\Psi(m_i|h_i)$ and maintains an accumulating reward, $r^c(h_i, m_i, \tau_i) = \sum_{t=t_{m_i}}^{t_{m_i}+\tau_i-1} \gamma^{t-t_{m_i}} r_t$, for the macro-action from its first time-step t_{m_i} to a termination time-step $t_{m_i} + \tau_i - 1$. The agent then obtains a new macro-observation z'_i with the probability $P(z'_i \mid h_i, m_i, \tau_i)$ and results in a new history $h'_i = \langle h_i, m_i, z'_i \rangle$ under the transition model $P(h'_i, \tau_i \mid h_i, m_i)$. Correspondingly, the experience tuple collected by each agent i is represented as $\langle z, m, z', r^c \rangle_i$, where z_i is the macro-observation used for choosing the macro-action m_i. We can write down the Bellman equation for each agent i under a given high-level policy Ψ_i as:

$$Q^{\Psi_i}(h_i, m_i) = \mathbb{E}_{h'_i, \tau_i | h_i, m_i}\left[r^c(h_i, m_i, \tau_i) + \gamma^{\tau_i} V^{\Psi_i}(h'_i)\right] \quad (15)$$

In each training iteration, agents first sample a concurrent mini-batch of sequential experiences (either random traces with the same length or entire episodes) from the replay buffer $\mathcal{D}$. Each sampled sequential experience is further cleaned up by filtering out the experiences when the corresponding macro-action is still executing. This disposal procedure finally results in a mini-batch of 'squeezed' sequential experiences for each agent's training. A specific example is shown in Figure 2.

In this work, we implement Dec-HDRQN with Double Q-learning (Section 2.3.1) to train the decentralized macro-action-value function $Q_{\theta_i}(h_i, m_i)$ (in equation (15)) for each

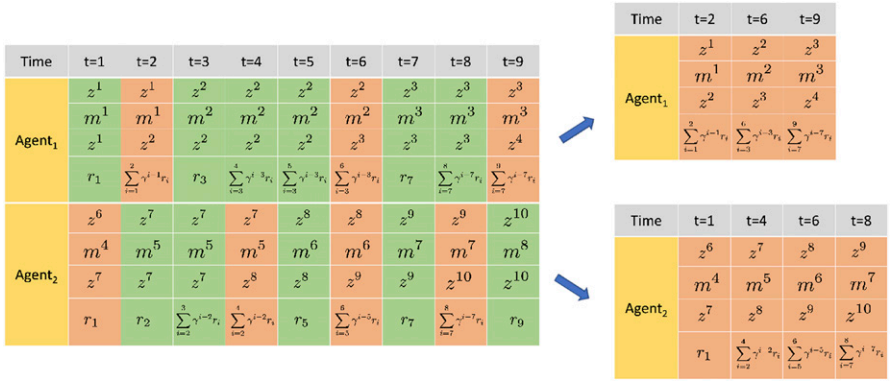


Figure 2. An example of Mac-CERTs. Two agents first sample concurrent trajectories from the replay buffer; the valid experience (when the macro-action terminates, marked as red), is then selected to compose a squeezed sequential experience for each agent. Note that we collect agents' high-level transition tuple at every primitive step, where each agent is allowed to obtain a new macro-observation if and only if the current macro-action terminates, otherwise, the next macro-observation is set as same as the previous one. The superscript is for distinguishing different macro-actions and macro-observations.

agent i. Each agent updates its own macro-action-value function by minimizing the loss:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{<z,m,z',r^c>_i \sim \mathcal{D}}\left[(y_i - Q_{\theta_i}(h_i, m_i))^2\right] \tag{16}$$

$$y_i = r_i^c + \gamma Q_{\theta_i^-}(h_i', \arg\max_{m_i'} Q_{\theta_i}(h_i', m_i')) \tag{17}$$

3.2. Macro-action-based centralized Q-learning (Mac-Cen-Q)

Achieving centralized control in the macro-action setting needs to learn a joint macro-action-value function $Q(\vec{h}, \vec{m})$. This requires a way to correctly accumulate the rewards for each joint macro-action. This is actually more complicated than the decentralized case because there is no obvious update step (i.e., there may never be a time when all agents have terminated their macro-actions together). As a result, we use the idea of updating when *any* agent terminates a macro-action (Amato et al., 2014, 2019) (see Appendix B for a theoretical analysis of this choice). But this makes updating and maintaining a buffer more complicated than in Section 3.1.

In this case, we introduce a centralized replay buffer that we call *Macro-Action Joint Experience Replay Trajectories* (Mac-JERTs). Instead of independently maintaining a cumulative reward for each macro-action, agents share a joint cumulative reward $\vec{r}^c(\vec{h}, \vec{m}, \vec{\tau}) = \sum_{t=t_{\vec{m}}}^{t_{\vec{m}}+\vec{\tau}-1} \gamma^{t-t_{\vec{m}}} r_t$ for each joint macro-action $\vec{m}$, where $t_{\vec{m}}$ is the time-step when a joint macro-action $\vec{m}$ starts, and $t_{\vec{m}} + \vec{\tau} - 1$ is the ending time-step of $\vec{m}$ when *any* agent finishes its macro-action. The agents who just complete their macro-actions can obtain new macro-observations, which leads to a new joint history $\vec{h}' = \langle \vec{h}, \vec{m}, \vec{z}' \rangle$ under the transition model $P(\vec{h}', \vec{\tau} \mid \vec{h}, \vec{m})$. Here, we can write down the Bellman equation under a centralized macro-action-based policy $\mathbf{\Psi}$ as:

$$Q^{\Psi}(\vec{h}, \vec{m}) = \mathbb{E}_{\vec{h}', \vec{\tau} \mid \vec{h}, \vec{m}}\left[\vec{r}^c(\vec{h}, \vec{m}, \vec{\tau}) + \gamma^{\vec{\tau}} V^{\Psi}(\vec{h}')\right] \tag{18}$$

In our work, we use Double-DRQN (DDRQN) to train the centralized macro-action-value function. In each training iteration, a mini-batch of sequential joint experiences is first sampled from Mac-JERTs, and a similar filtering operation, as presented in Section 3.1, is used to obtain the "squeezed" joint experiences (shown in Figure 3). But, in this case, only one joint reward is maintained that accumulates from the selection of any agent's macro-action to the completion of any (possibly other) agent's macro-action.

Using the squeezed joint sequential experiences, the centralized macro-action-value function (in equation (18)), $Q_\phi(\vec{h}, \vec{m})$, is trained end-to-end to minimize the following loss:

$$\mathcal{L}(\phi) = \mathbb{E}_{<\vec{r},\vec{m},\vec{z}',\vec{r}^c> \sim \mathcal{D}}\left[\left(y - Q_\phi\left(\vec{h}, \vec{m}\right)\right)^2\right] \tag{19}$$

$$y = \vec{r}^c + \gamma Q_{\phi^-}(\vec{h}', \arg\max_{\vec{m}'} Q_\phi(\vec{h}', \vec{m}')) \tag{20}$$

The next joint macro-action selection part in equation (20) implies that at the next step, all agents will switch to a new macro-action. However, in reality, only the agents who have terminated their current macro-actions can switch to a new macro-action in the next step. For example, in Figure 3, the first three squeezed sequential experiences show that only one of the agents changes its macro-action per step. Therefore, the more agents that are not switching macro-actions, the less accurate the prediction that equation (20) will make. In order to have a more correct (in terms of matching with the true executions over agents) value estimation for a joint macro-action, here, we propose a *conditional target prediction* as:

$$y_{(t)} = \vec{r}^c + \gamma Q_{\phi^-}(\vec{h}', \arg\max_{\vec{m}'} Q_\phi(\vec{h}', \vec{m}' \mid \vec{m}^{\text{undone}})) \tag{21}$$

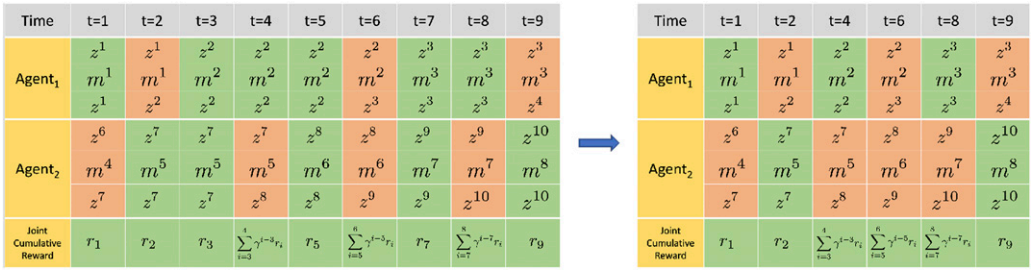


Figure 3. An example of Mac-JERTs. A joint sequential experience is first sampled from the memory buffer, and then, depending on the termination (red) of each joint macro-action, a squeezed sequential experience is generated for the centralized training. Each agent's macro-action, which is responsible for the termination of the joint one, is marked in red. For example, at $t = 1$, agents execute a joint macro-action $\vec{m} = \langle m^1, m^4 \rangle$ for one time step; at $t = 2$, the joint macro-action becomes $\langle m^1, m^5 \rangle$ as Agent_2 finished m^4 at last step and chooses a new macro-action m^5; Agent_1 finished its macro-action m_1 at $t = 2$ and selects a new macro-action m^2 at $t = 3$ so that the joint macro-action switches to $\langle m^2, m^5 \rangle$ which keeps running until the 4th time step. Therefore, the first two joint macro-actions have two single-step rewards, respectively, and the reward of joint macro-action $\langle m^2, m^5 \rangle$ is an accumulative reward over two consecutive time steps.

where $\vec{m}^{\text{undone}}$ is the joint-macro-action over the agents who have not terminated the macro-actions at the current time step and will continue running it at the next step.

3.3. Macro-action-based CTDE Q-learning

In multi-agent environments, decentralized learning causes the environment to be non-stationary from each agent's perspective as other agents' policies change during learning. *Centralized Training for Decentralized Execution* (CTDE) is the most popular framework that can naturally stabilize the learning of decentralized policies by overcoming the environmental non-stationarity issue and promoting global cooperative behavior to be achieved via decentralized execution. VDN (Sunehag et al., 2018) and QMIX (Rashid et al., 2018), the pioneers of CTDE Q-learning, use centralized training by first training a joint, but factored, Q-net and later decomposing it into a decentralized Q-net for each agent to use during execution. Differing from value-factorization-based approaches, in this section, we first propose a new multi-agent Double DQN-based approach, called MacDec-DDRQN, to learn decentralized macro-action-value functions that are trained with a centralized macro-action-value function; and followed by a generalized version of it, called Parallel-MacDec-DDRQN.

3.3.1. MacDec-DDRQN. Double DQN has been implemented in multi-agent domains for learning either centralized or decentralized policies (Simões et al., 2017; Xiao and Hoffman, n.d.; Zheng et al., 2018). However, in the decentralized learning case, each agent independently adopts double Q-learning purely based on its own local information. Learning only from local information often impedes agents from achieving high-quality cooperation.

In order to take advantage of centralized information for learning decentralized action-value functions, we train the centralized macro-action-value function Q_ϕ and each agent's decentralized macro-action-value function Q_{θ_i} simultaneously, and the target value for updating decentralized macro-action-value function Q_{θ_i} is then calculated by using the centralized Q_ϕ for macro-action selection and the decentralized target-net $Q_{\theta_i^-}$ for value estimation.

More concretely, consider a domain with N agents, and both the centralized Q-network Q_ϕ and decentralized Q-networks Q_{θ_i} for each agent i are represented as DRQNs (Hausknecht and Stone, 2015). Here, it is important to note that the centralized Q-network Q_ϕ does not involve any factorization architecture. As a result, Q_ϕ is not restricted by any constraint and it is able to represent the true centralized macro-action values in any given domain.

The experience replay buffer $\mathcal{D}$, a merged version of Mac-CERTs and Mac-JERTs, contains the tuples $\langle \mathbf{z}, \mathbf{m}, \mathbf{z}', \mathbf{r}^{\mathbf{c}}, \vec{r}^{c} \rangle$, where $\mathbf{z} = \{z_0, \ldots, z_N\}$, $\mathbf{m} = \{m_0, \ldots, m_N\}$ and $\mathbf{r}^{\mathbf{c}} = \{r_0^c, \ldots, r_N^c\}$. In each training iteration, agents sample a mini-batch of sequential experiences to first optimize the centralized macro-action-value function Q_ϕ in the way proposed in Section 3.2, and then update each decentralized macro-action-value function by minimizing the squared TD error:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{<\mathbf{z}, \mathbf{m}, \mathbf{z}', \mathbf{r}^{\mathbf{c}}, \vec{r}^{c}> \sim \mathcal{D}} \left[\left(y_i - Q_{\theta_i}(h_i, m_i) \right)^2 \right] \tag{22}$$

where

$$y_i = r_i^c + \gamma Q_{\theta_i^-} \left[h_i', \left[\arg\max_{\mathbf{m}'} Q_\phi(\mathbf{h}', \mathbf{m}') \right] \right] \tag{23}$$

In equation (23), $\left[\arg\max_{\mathbf{m}'} Q_\phi(\mathbf{h}', \mathbf{m}') \right]_i$ implies selecting the joint macro-action with the highest value and then selecting the individual macro-action for agent i. In this updating rule, not only double estimators $Q_{\theta_i^-}$ and Q_ϕ are applied to counteract overestimation on target Q-values, but also a centralized heuristic on action selection is embedded. Now, from each agent's perspective, the target Q-value is calculated by assuming all agents will behave based on the centralized Q-net next step (Equation (23)), in which the provided global info by the centralized Q-net will help each agent to avoid trapping in local optima and also facilitates them to learn cooperation behaviors.

Additionally, similar to the idea of the conditional operation for training centralized joint macro-action-value function discussed in Section 3.2, in order to obtain a more accurate prediction by taking each agent's macro-action executing status into account, equation (23) can be rewritten as:

$$y_i = r_i^c + \gamma Q_{\theta_i^-}\Big[h_i', \big[\arg\max_{\mathbf{m}'} Q_\phi(\mathbf{h}', \mathbf{m}' \mid \mathbf{m}^{\mathbf{undone}})\big]_i\Big] \quad (24)$$

Now, the agent who has not finished the macro-action at the target updating time step is considered to continue running the same macro-action in the target action-value computation.

Algorithm 1 Parallel-MacDec-DDRQN

1: Initialize centralized Q-Networks: Q_ϕ, Q_ϕ^-
2: Initialize decentralized Q-Networks for each agent i: Q_{θ_i}, $Q_{\theta_i}^-$
3: Initialize two parallel environments *cen-env*, *dec-env*
4: Initialize two step counters $t_{\text{cen-env}}$, $t_{\text{dec-env}}$
5: Initialize centralized buffer $\mathcal{D}_{\text{cen}} \leftarrow$ Mac-JERTs
6: Initialize decentralized buffer $\mathcal{D}_{\text{dec}} \leftarrow$ Mac-CERTs
7: Get initial joint-macro-observation $\vec{z}$ for agents in *cen-env*
8: Get initial macro-observation z_i for each agent i in *dec-env*
9: **for** *dec-env-episode* = 1 to M **do**
10: Agents take joint-macro-action with cen-ϵ-greedy using Q_ϕ
11: Store $\langle \vec{z}, \vec{m}, \vec{z}', \vec{r}^c \rangle$ in $\mathcal{D}_{\text{cen}}$
12: $t_{\text{cen-env}} \leftarrow t_{\text{cen-env}} + 1$
13: Each agent i takes macro-action with dec-ϵ-greedy using Q_{θ_i}
14: Store $\langle z_i, m_i, z_i', r_i^c \rangle$ in $\mathcal{D}_{\text{dec}}$
15: $t_{\text{dec-env}} \leftarrow t_{\text{dec-env}} + 1$
16: **if** $t_{\text{dec-env}}$ mod $I_{\text{train}} == 0$ **then**
17: Sample a mini-batch $\mathcal{B}_{\text{cen}}$ of sequential experiences
18: $\langle \vec{z}, \vec{m}, \vec{z}', \vec{r}^c \rangle$ from $\mathcal{D}_{\text{cen}}$
19: Perform a gradient decent step on $\big(y - Q_\phi(\vec{h}, \vec{m})\big)^2_{\mathcal{B}_{\text{cen}}}$, where
20: y is computed as Eq 21.
21: Sample a mini-batch $\mathcal{B}_{\text{dec}}$ of sequential experiences
22: $\langle z_i, m_i, z_i', r_i^c \rangle$ from $\mathcal{D}_{\text{dec}}$ for each agent i
23: Perform a gradient decent step on $\big(y_i - Q_{\theta_i}(h_i, m_i)\big)^2_{\mathcal{B}_{\text{dec}}}$,
24: where y_i is computed as Eq. 24.
25: **if** $t_{\text{dec-env}}$ mod $I_{\text{TargetUpdate}} == 0$ **then**
26: Update centralized target network $\phi^- \leftarrow \phi$
27: Update each agent's decentralized target network $\theta_i^- \leftarrow \theta_i$
28: **if** $t_{\text{cen-env}}$ ==max-episode-length **or** terminal state **then**
29: Reset *cen-env*
30: Get initial joint-macro-observation $\vec{z}$ for agents in *cen-env*
31: **if** $t_{\text{dec-env}}$ ==max-episode-length **or** terminal state **then**
32: Reset *dec-env*
33: Get initial macro-observation z_i for each agent i in *dec-env*

3.3.2. Parallel-MacDec-DDRQN. Exploration is also a difficult problem in multi-agent reinforcement learning. ϵ-greedy exploration has been widely used in many methods such as Q-learning to generate training data (Sutton et al., 1998). In DQN-based methods, as a hyper-parameter, ϵ often acts with a linear decay along with the training steps from 1.0 to a lower value to achieve the trade-off between exploration and exploitation. And, exploration can be done either in a centralized way or in a decentralized way. Centralized exploration may help to choose cooperative actions more often that would have a low probability of being selected from decentralized policies, and decentralized exploration may provide more realistic data that is actually achievable by decentralized policies.

Therefore, in our approach, besides tuning ϵ, we introduce a hyper-selection for performing an ϵ-greedy behavior policy that can perform either centralized exploration based on Q_ϕ or decentralized exploration using each agent's Q_{θ_i}.

However, without having enough knowledge about the properties of a given domain in the very beginning, it is not clear which exploration choice is the best. To cope with this, we propose a more generalized version of MacDec-DDRQN, called *Parallel-MacDec-DDRQN*, summarized in Algorithm 1. The core idea is to have two parallel environments (line 3) with agents, respectively, performing centralized exploring (line 10: cen-ϵ-greedy) and decentralized exploring (line 13: dec-ϵ-greedy) in each. The centralized Q_ϕ is first trained purely using the centralized experiences (line 19), while each agent's decentralized Q_{θ_i} is then optimized using equation (24) with only decentralized experiences (line 23).

3.4. Macro-action-based independent actor-critic

Similar to the idea of IAC with primitive-actions (Section 2.4.2), a straightforward extension is to have each agent independently optimize its own macro-action-based policy (actor) using a local macro-action-value function (critic). Hence, we start with deriving a *macro-action-based policy gradient theorem* in Appendix A.1 by incorporating the general Bellman equation for the state values of a macro-action-based policy (Sutton et al., 1999) into the *policy gradient theorem* in MDPs (Sutton et al., 2000), and then extend it to MacDec-POMDPs so that each agent can have the following policy gradient w.r.t. the parameters of its macro-action-based policy $\Psi_{\theta_i}(m_i|h_i)$ as: $\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\vec{\Psi}_{\vec{\theta}}}\Big[\nabla_{\theta_i} \log \Psi_{\theta_i}(m_i \mid h_i) Q_{\phi_i}^{\Psi_{\theta_i}}(h_i, m_i)\Big]$. During training, each agent accesses its own trajectories and squeezes them in the same way (refer to Figure 2) as the decentralized case (presented in Section 3.1) to train the critic $Q_{\phi_i}^{\Psi_{\theta_i}}(h_i, m_i)$ via on-policy TD learning and perform gradient ascent to update the policy when the agent's macro-action terminates. In our case, we train a local history value function $V_{\mathbf{w}_i}^{\Psi_{\theta_i}}(h_i)$ as each agent's critic and use it as a baseline to achieve variance reduction. The corresponding policy gradient is as follows:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\vec{\Psi}_{\vec{\theta}}}[\nabla_{\theta_i} \log \Psi_{\theta_i}(m_i \mid h_i) A(h_i, m_i)] \quad (25)$$

$$A(h_i, m_i) = r_i^c + \gamma^{\tau_{m_i}} V_{\mathbf{w}_i}^{\Psi_{\theta_i}}(h_i') - V_{\mathbf{w}_i}^{\Psi_{\theta_i}}(h_i) \quad (26)$$

where the cumulative reward r_i^c is w.r.t. the execution of agent i's macro-action m_i.

3.5. Macro-action-based centralized actor-critic

In the fully centralized learning case, we treat all agents as a single joint agent to learn a centralized actor $\Psi_\theta(\vec{m} \mid \vec{h})$ with a centralized critic $Q_\phi^{\Psi_\theta}(\vec{h}, \vec{m})$, and the policy gradient can

be expressed as: $\nabla_\theta J(\theta) = \mathbb{E}_{\Psi_\theta}\Big[\nabla_\theta \log \Psi_\theta(\vec{m} \mid \vec{h}) Q_\phi^{\Psi_\theta}(\vec{h}, \vec{m})\Big]$. Similarly, to achieve a lower variance optimization for the actor, we learn a centralized history value function $V_{\mathbf{w}}^{\Psi_\theta}(\vec{h})$ by minimizing a TD-error loss over joint trajectories that are squeezed w.r.t. each joint macro-action termination (when *any* agent terminates its macro-action, defined in the centralized case in Section 3.2 and visualized in Figure 3). Accordingly, the policy's updates are performed when each joint macro-action is completed by ascending the following gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\Psi_\theta}\Big[\nabla_\theta \log \Psi_\theta(\vec{m} \mid \vec{h}) A(\vec{h}, \vec{m})\Big] \tag{27}$$

$$A(\vec{h}, \vec{m}) = \vec{r}^{\,c} + \gamma^{\vec{\tau}\,\vec{m}} V_{\mathbf{w}}^{\Psi_\theta}(\vec{h}') - V_{\mathbf{w}}^{\Psi_\theta}(\vec{h}) \tag{28}$$

where the cumulative reward $\vec{r}^{\,c}$ is w.r.t. the execution of the joint macro-action $\vec{m}$.

3.6. Macro-action-based independent actor with centralized critic

As mentioned earlier, fully centralized learning requires perfect online communication which is often hard to guarantee, and fully decentralized learning suffers from environmental non-stationarity due to agents' changing policies. In order to learn better decentralized macro-action-based policies, in this section, we propose two macro-action-based actor-critic algorithms using the CTDE paradigm. Typically, the difference between a joint macro-action termination from the centralized perspective and a macro-action termination from each agent's local perspective gives rise to a new challenge: *what kind of centralized critic to learn and how to use it to optimize decentralized policies under such an asymmetric asynchrony from the two perspectives*, which we mainly investigate below.

A naive way of incorporating macro-actions into a CTDE-based actor-critic framework is to directly adapt the idea of the primitive-action-based IACC (Section 2.4.3) to have a shared joint macro-action-value function $Q_\phi^{\vec{\Psi}_{\vec{\theta}}}(\mathbf{x}, \vec{m})$ in each agent's decentralized macro-action-based policy gradient as: $\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\vec{\Psi}_{\vec{\theta}}}\Big[\nabla_{\theta_i} \log \Psi_{\theta_i}(m_i \mid h_i) Q_\phi^{\vec{\Psi}_{\vec{\theta}}}(\mathbf{x}, \vec{m})\Big]$. To reduce variance, with a value function $V_{\mathbf{w}}^{\vec{\Psi}_{\vec{\theta}}}(\mathbf{x})$ as the centralized critic, the policy gradient w.r.t. the parameters of each agent's high-level policy can be rewritten as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\vec{\Psi}_{\vec{\theta}}}[\nabla_{\theta_i} \log \Psi_{\theta_i}(m_i \mid h_i) A(\mathbf{x}, \vec{m})] \tag{29}$$

$$A(\mathbf{x}, \vec{m}) = \vec{r}^{\,c} + \gamma^{\vec{\tau}\,\vec{m}} V_{\mathbf{w}}^{\vec{\Psi}_{\vec{\theta}}}(\mathbf{x}') - V_{\mathbf{w}}^{\vec{\Psi}_{\vec{\theta}}}(\mathbf{x}) \tag{30}$$

Here, the critic is trained in the fully centralized manner described in Section 3.2 while allowing it to access additional global information (e.g., joint macro-observation-action history, ground truth state or both) represented by the symbol $\mathbf{x}$. However, updates of each agent's policy $\Psi_{\theta_i}(m_i \mid h_i)$ only occur at the agent's own macro-action termination time steps rather than depending on joint macro-action terminations in the centralized critic training. In Figure 4, we show an example of the trajectory squeezing process in Naive Mac-IACC.

3.6.1. Independent actor with individual centralized critic (Mac-IAICC). Note that naive Mac-IACC is technically incorrect. The cumulative reward $\vec{r}^{\,c}$ in equation (30) is based on the corresponding joint macro-action's termination that is defined as when *any* agent finishes its own macro-action, which produces two potential issues: (a) $\vec{r}^{\,c} + \gamma^{\vec{\tau}\,\vec{m}} V_{\mathbf{w}}^{\vec{\Psi}_{\vec{\theta}}}(\mathbf{x}')$ may not estimate the value of the macro-action m_i well as the reward does not depend on m_i's termination; (b) from agent i's perspective, its policy gradient estimation may involve higher variance associated with the asynchronous macro-action terminations of other agents.

To tackle the aforementioned issues, we propose to learn a separate centralized critic $V_{\mathbf{w}_i}^{\vec{\Psi}_{\vec{\theta}}}(\mathbf{x}')$ for each agent via TD-learning. In this case, the TD-error for updating $V_{\mathbf{w}_i}^{\vec{\Psi}_{\vec{\theta}}}(\mathbf{x}')$ is computed by using the reward r_i^c that is accumulated purely based on the execution of the agent i's macro-action m_i. With this TD-error estimation, each agent's decentralized macro-action-based policy gradient becomes:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\vec{\Psi}_{\vec{\theta}}}[\nabla_{\theta_i} \log \Psi_{\theta_i}(m_i \mid h_i) A(\mathbf{x}, m_i)] \tag{31}$$

$$A(\mathbf{x}, m_i) = r_i^c + \gamma^{\tau_{m_i}} V_{\mathbf{w}_i}^{\vec{\Psi}_{\vec{\theta}}}(\mathbf{x}') - V_{\mathbf{w}_i}^{\vec{\Psi}_{\vec{\theta}}}(\mathbf{x}) \tag{32}$$

Now, from agent i's perspective, $r_i^c + \gamma^{\tau_{m_i}} V_{\mathbf{w}_i}^{\vec{\Psi}_{\vec{\theta}}}(\mathbf{x}')$ is able to offer a more accurate value prediction for the macro-action m_i, since both the reward, r_i^c and the value function $V_{\mathbf{w}_i}^{\vec{\Psi}_{\vec{\theta}}}(\mathbf{x}')$ depend on agent i's macro-action termination. Also, unlike the case in Naive Mac-IACC, other agents' terminations cannot lead to extra noisy estimated rewards w.r.t. m_i anymore, so the variance on policy gradient estimation gets reduced. Then, updates for both the critic and the actor occur when the corresponding agent's macro-action ends and take advantage of information sharing. Figure 5 shows an example of the trajectory squeezing process in Mac-IAICC.

4. Experiments in simulation

We investigate the performance of our algorithms over a variety of multi-robot problems with macro-actions (Figure 6): Box Pushing (Xiao and Hoffman, n.d.), Overcooked (Wu et al., 2021b), and a larger Warehouse Tool Delivery (Xiao and Hoffman, n.d.) domain. Macro-actions are defined by using prior domain knowledge as they are straightforward in these tasks. Typically, we also include primitive actions in the macro-action set (as one-step macro-actions), which gives agents the chance to learn more complex policies that use both when it is necessary. The horizon of each problem domain and environmental partitions and labels are not known to the agents. We describe

Figure 4. An example of the trajectory squeezing process in Naive Mac-IACC. The joint trajectory is first squeezed depending on joint macro-action termination for training the centralized critic. Then, the trajectory is further squeezed for each agent depending on each agent's own macro-action termination for training the decentralized policy.

the domains' key properties here and have more details in Appendix D.

4.1. Experimental setup

4.1.1. Box pushing (Figure 6(a)). Two robots are in an environment with two small boxes and one large box. The optimal solution is to cooperatively push the big box to the yellow goal area for a terminal reward, but partial observability makes this difficult. Specifically, robots have four primitive-actions: *move forward*, *turn-left*, *turn-right*, and *stay*. In the macro-action case, each robot has three one-step macro-actions: ***Turn-left***, ***Turn-right***, and ***Stay***, as well as three multi-step macro-actions: ***Move-to-small-box(i)*** and ***Move-to-big-box(i)*** navigate the robot to the red spot below the corresponding box and terminate with the robot facing the box; ***Push*** causes the robot to keep moving forward until arriving at the world's boundary (potentially pushing the small box or trying to push the big one). The big box only moves if both robots push it together. Each robot can only observe the status (*empty*, *teammate*, *boundary*, *small*, *or big box*) of the cell in front of it. When any box is pushed to the goal, the team receives a terminal reward (+300 for the big box and +20 for each small box). A penalty is issued when any robot hits the boundary or pushes the big box alone.

4.1.2. Overcooked (Figure 6(b) and (c)). Three robots must learn to cooperatively prepare a lettuce-tomato-onion salad and deliver it to the "star" cell. The challenge is that the salad's recipe (Figure 6(d)) is unknown to the robots. With primitive actions (*move up*, *down*, *left*, *right*, and *stay*), robots can move around and achieve picking, placing, chopping, and delivering by standing next to the corresponding cell and moving against it (e.g., in Figure 6(b), the pink robot can *move right* and then *move up* to pick up the tomato). We describe the major function of macro-actions below and full details (e.g., termination conditions) are included in Appendix D.1. Each robot's macro-action set consists of: (a) five one-step macro-actions that are the same as the primitive ones; (b) ***Chop***, cuts a raw vegetable into pieces when the robot stands next to a cutting board and an unchopped vegetable is on the board, otherwise it does nothing; (c) long-term navigation macro-actions: ***Get-Lettuce***, ***Get-Tomato***, ***Get-Onion***, ***Get-Plate-1/2***, ***Go-Cut-Board-1/2***, and ***Deliver***, which navigate the robot to the location of the corresponding object with various possible terminal effects (e.g., holding a vegetable in hand, placing a chopped vegetable on a plate, arriving at the cell next to a cutting board, delivering an item to the star cell, or immediately terminating when any property condition does not hold, e.g., no path is found or the vegetable/plate is not found); (d) ***Go-Counter*** (only available in Overcook-B, Figure 6(c)), navigates a robot to the center cell in the middle of the map when the cell is not occupied, otherwise, it moves to an adjacent cell. If the robot is holding an object or one is in the cell, the object will be placed or picked up. Each robot only observes the *positions* and *status* of the entities within a 5 × 5 square centered on the robot. The reward involves: +10 for chopping a vegetable into pieces, +200 terminal reward for delivering a lettuce-tomato-onion salad, −5 reward for delivering any wrong item that is then reset to its initial position, and −0.1 for every time step.

4.1.3. Warehouse tool delivery (Figure 6(e)–6(h)). In each workshop (e.g., W-0), a human is working on an assembly task (involving four sub-tasks that each takes a number of time steps to complete) and requires three different tools for future sub-tasks to continue. A robot arm (gray) must find tools for each human on the table (brown) and pass them to mobile robots (green, blue and yellow) who are responsible for delivering tools to humans. Note that, the correct tools needed by each human are unknown to robots, which has to be learned during training in order to perform efficient delivery. A delayed delivery leads to a penalty. We consider variants with two or three mobile robots and two to four humans to examine the scalability of our methods (Figure 6(f)–6(h)).

Figure 5. An example of the trajectory squeezing process in Mac-IAICC: each agent learns an individual centralized critic for the decentralized policy optimization. To better utilize centralized information, each agent's critic should receive all the valid joint macro-observation-action history (when *any* agent terminates its macro-action and obtains a new joint macro-observation). However, the critic's TD updates and the policy's updates still rely on each agent's individual macro-action termination and the accumulative reward at the corresponding time steps. Hence, the trajectory squeezing process for training each critic still depends on joint-macro-action termination but only retaining the accumulative rewards w.r.t. the corresponding agent's macro-action termination for computing the TD loss (the middle part in the above picture). Then, each agent's trajectory is further squeezed depending on its macro-action termination to update the decentralized policy.

We also consider one faster human (orange) to check if robots can prioritize him (Figure 6(g)). Mobile robots have the following macro-actions: ***Go-W(i)***, moves to the waypoint (red) at workshop i; ***Go-TR***, goes to the waypoint at the right side of the tool room (covered by the blue robot in Figure 6(g) and (h)); and ***Get-Tool***, navigates to a pre-allocated waypoint (that is different for each robot to avoid collisions) next to the robot arm and waits there until either receiving a tool or 10 time steps have passed. The robot arm's applicable macro-actions are: ***Search-Tool(i)***, finds tool i and places it in a staging area (containing at most two tools) on the table, and otherwise, it freezes the robot for the amount of time the action would take when the area is fully occupied; ***Pass-to-M(i)***, passes the first staged tool to mobile robot i; and ***Wait-M***, waits for 1 time step. The robot arm only observes the *type* of each tool in the staging area and *which mobile robot* is waiting at the adjacent waypoints. Each mobile robot always knows its *position* and the *type* of tool that it is carrying, and can observe the *number* of tools in the staging area or the *sub-task* a human is working on only when at the tool room or the workshop, respectively. The team receives: +100 for delivering a correct tool to a human on time, −20 for a delayed delivery, −10 for the arm robot running ***Pass-to-M(i)*** without the mobile robot i being next to it, and −1 every time step.

4.2. Results

We evaluate performance of one training trial with a mean discounted return measured by periodically (every 100 episodes) evaluating the learned policies over 10 testing episodes. We plot the average performance of each method over 20 independent trials with one standard error and smooth the curves over 10 neighbors. We also show the optimal expected return in Box Pushing domain as a dash-dot line. To ensure a fair comparison over different approaches, we perform hyper-parameter tuning for methods in each comparison under the same set of hyper-parameters, and then, we choose the best performance of each method depending on its final converged value as the first priority and the sample efficiency as the second. More training details are in Appendix E.

4.2.1. Advantage of learning with macro-actions. We first present a comparison between our macro-action-based methods and the primitive-action-based methods in fully decentralized and fully centralized cases, and we show the results of value-based methods and actor-critic methods in Figures 7 and 8, respectively. The comparisons consider various grid world sizes of the Box Pushing domain and two Overcooked scenarios. The results show significant performance improvements for using macro-actions over primitive actions. More concretely, in the Box Pushing

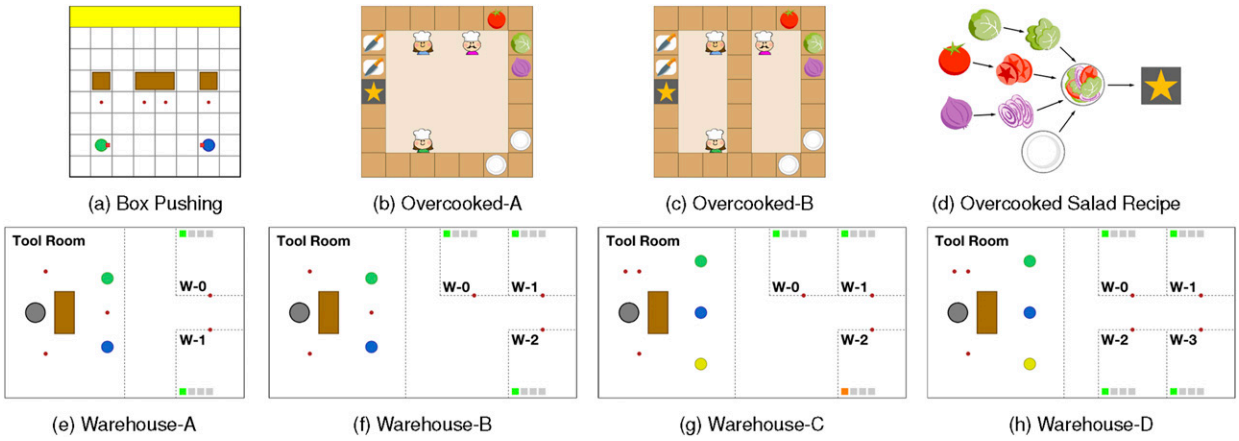


Figure 6. Experimental environments.

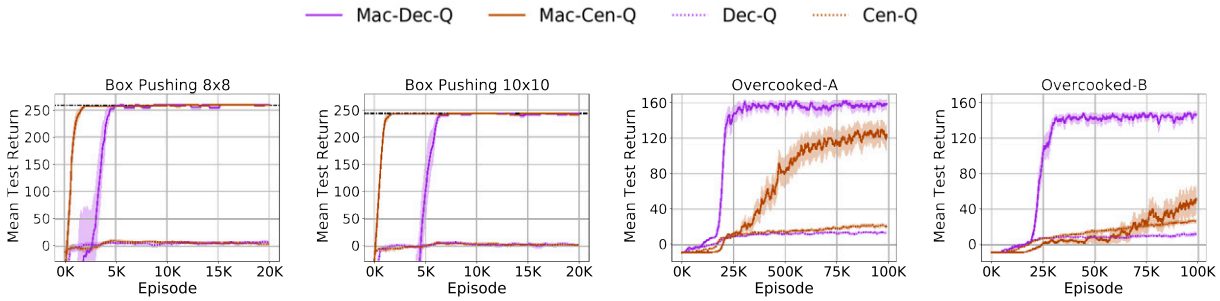


Figure 7. Value-based approaches with macro-actions versus primitive-actions.

domain, reasoning about primitive movements at every time step makes the problem intractable so the robots cannot learn any good behaviors in primitive-action-based approaches other than to keep moving around. Conversely, Mac-Cen-Q (solid brown) and Mac-CAC (solid orange) reach near-optimal performance, enabling the robots to push the big box together. Unlike the centralized critic which can access joint information, even in the macro-action case, it is hard for each robot's on-policy decentralized critic to correctly measure the responsibility for a penalty caused by a teammate pushing the big box alone. Mac-IAC (solid blue) thus converges to a local optimum of pushing two small boxes in order to avoid getting the penalty. Mac-Dec-Q (solid purple) learns slowly at the early stage, but, as an off-policy learning approach, it takes advantage of the replay buffer to re-visit the good experience of pushing the big box and eventually achieves near-optimal performance.

In the Overcooked domain, an efficient solution requires the robots to asynchronously work on independent subtasks (e.g., in scenario A, one robot gets a plate while another two robots pick up and chop vegetables; and in scenario B, the right robot transports items while the left two robots prepare the salad). This large amount of independence explains why Mac-Dec-Q (solid purple) and Mac-IAC (solid blue) can solve the task well. This also indicates that using local information is enough for robots to achieve high-quality behaviors. As a result, Mac-Cen-Q (solid brown) and Mac-CAC (solid orange) learn slower because they must figure out the redundant part of joint information in much larger joint macro-level history and action spaces than the spaces in the decentralized case. The primitive-action-based methods begin to learn but perform poorly in such long-horizon tasks.

4.2.2. Property analysis of macro-action-based CTDE Q-learing. Figure 9 shows the results of three variants of our macro-action-based CTDE Q-learning algorithm, MacDec-DDRQN: (a) with centralized exploration as the default; (b) with decentralized exploration (Dec-Explore); (c) with parallel environments (Parallel) compared with our fully decentralized method (MacDec-Q) and fully centralized method (MacCen-Q).

In the Box Pushing domain, the key sequential cooperative behavior such as going to the big box and pushing it at the same moment is much easier to be generated from the centralized perspective than the decentralized way. Thus, relying on centralized exploration, MacDec-DDRQN achieves near-centralized performance and better sample efficiency than fully decentralized learning, MacDec-Q. The joint Q-value function learned in Dec-Explore is purely based on decentralized data, so it fails to provide better target actions for decentralized policy updates. The failure of Parallel shows that, in this particular domain, having a

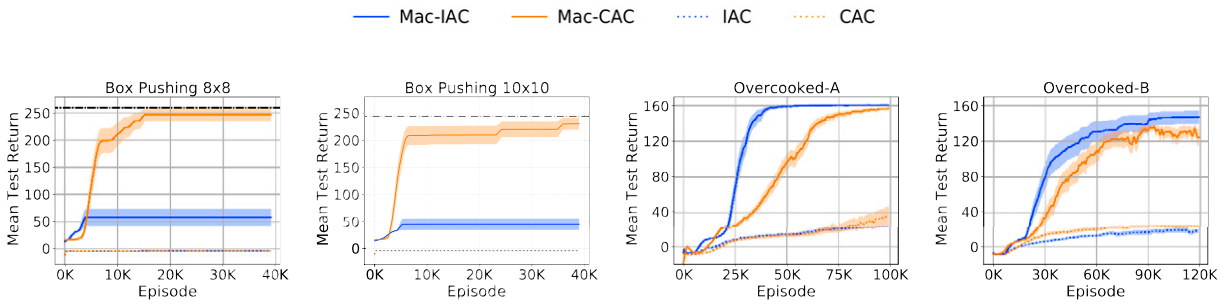


Figure 8. Actor-critic approaches with macro-actions versus primitive-actions.

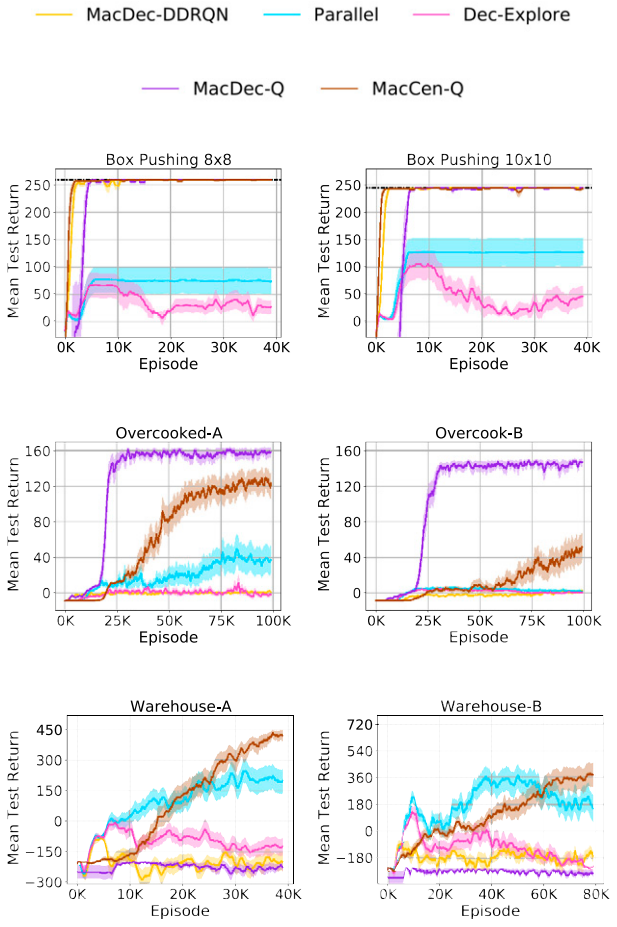


Figure 9. Comparison of value-based methods with macro-actions.

well-trained joint Q-value function from a separate environment but without the centralized cooperative behavior data may hurt the learning on decentralized policies.

Due to the aforementioned attribute of the independence in subtasks solving over agents, in the Overcooked domain, MacDec-Q performs the best while MacCen-Q leans slowly because of the huge joint-macro-action space (15^3). The variants of our CTDE-based approaches, therefore, cannot learn high-quality solutions, as they all rely on the guidance from the trained joint Q-value function (refer to the *argmax* operator in equation (24)).

In the warehouse domain, Mac-Dec-Q cannot solve this problem well due to its natural limitations and the domain's partial observability. In particular, it is difficult for the gray robot (arm) to learn an efficient way to find the correct tools purely based on local information and very delayed rewards that depend on the mobile robots' behaviors. Take advantage of join information over agents, MacCen-Q eventually outperforms all other methods. The Parallel way achieves significant improvement while learning decentralized policies. The failure of Dec-Explore and MacDec-DDRQN with centralized exploration demonstrates the necessity of using centralized data to achieve well-trained joint Q-value functions as well as the importance of having realistic decentralized data for decentralized policy training, which are attained in a Parallel manner.

4.2.3. Advantage of having individual centralized critics. Figure 10 shows the evaluation of our methods in all three domains. As each agent's observation is extremely limited in Box Pushing, we allow centralized critics in both Mac-IAICC and Naive Mac-IACC to access the state (agents' poses and boxes' positions), but use the joint macro-observation-action history in the other two domains.

In the Box Pushing task (the left two at the top row in Figure 10), Naive Mac-IACC (green) can learn policies almost as good as the ones for Mac-IAICC (red) for the smaller domain, but as the grid world size grows, Naive Mac-IACC performs poorly while Mac-IAICC keeps its performance near the centralized approach. From each agent's perspective, the bigger the world size is, the more time steps a macro-action could take, and the less accurate the critic of Naive Mac-IACC becomes since it is trained depending on any agent's macro-action termination. Conversely, Mac-IAICC gives each agent a separate centralized critic trained with the reward associated with its own macro-action execution.

In Overcooked-A (the third one at the top row in Figure 10), as Mac-IAICC's performance is determined by the training of three agents' critics, it learns slower than Naive Mac-IACC in the early stage but converges to a slightly higher value and has better learning stability than Naive Mac-IACC in the end. The result of scenario B (the last one at the top row in Figure 10) shows that Mac-IAICC outperforms other methods in terms of achieving better

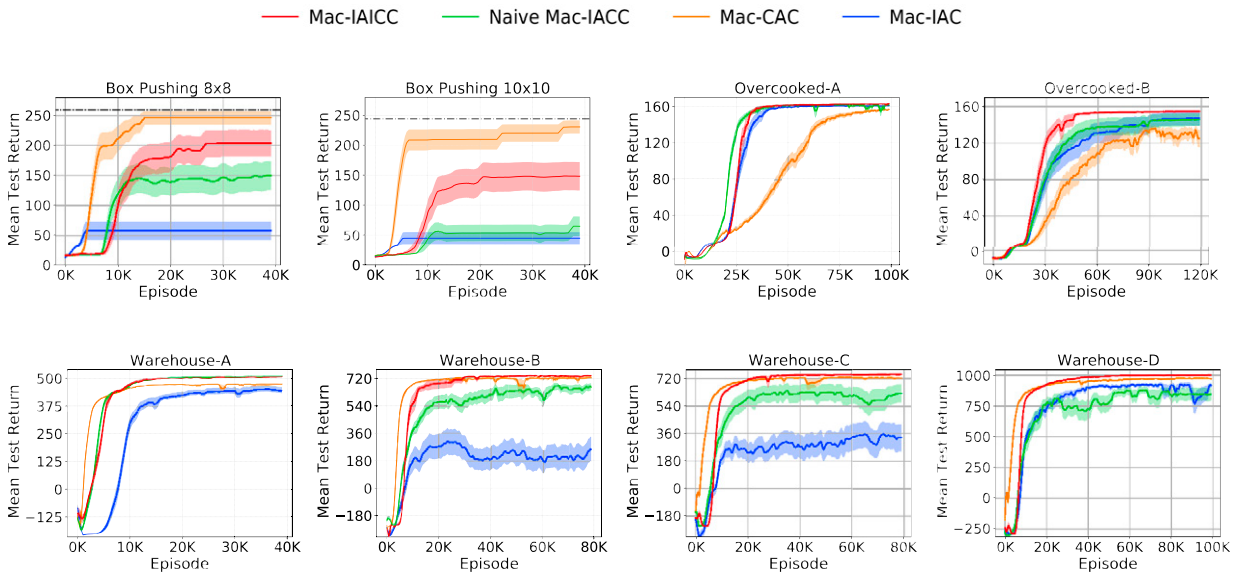


Figure 10. Comparison of macro-action-based asynchronous actor-critic methods.

sample efficiency, a higher final return and a lower variance. The middle wall in scenario B limits each agent's moving space and leads to a higher frequency of macro-action terminations. The shared centralized critic in Naive Mac-IACC thus provides more noisy value estimations for each agent's actions. Because of this, Naive Mac-IACC performs worse with more variance. Mac-IAICC, however, does not get hurt by such environmental dynamics change. Both Mac-CAC and Mac-IAC are not competitive with Mac-IAICC in this domain.

In the Warehouse scenarios (the bottom row in Figure 10), Mac-IAC (blue) performs the worst due to the same reason as MacDec-Q mentioned above. In contrast, in the fully centralized Mac-CAC (orange), both the actor and the critic have global information so it can learn faster in the early training stage. However, Mac-CAC eventually gets stuck at a local optimum in all five scenarios due to the exponential dimensionality of joint history and action spaces over robots. By leveraging the CTDE paradigm, both Mac-IAICC and Naive Mac-IACC perform the best in warehouse A. Yet, the weakness of Naive Mac-IACC is clearly exposed when the problem is scaled up in Warehouse B, C, and D. In these larger cases, the robots' asynchronous macro-action executions (e.g., traveling between rooms) become more complex and cause more mismatching between the termination from each agent's local perspective and the termination from the centralized perspective, and therefore, Naive Mac-IACC's performance significantly deteriorates, even getting worse than Mac-IAC in Warehouse-D. In contrast, Mac-IAICC can maintain its outstanding performance, converging to a higher value with much lower variance, compared to other methods. This outcome confirms not only Mac-IAICC's scalability but also the effectiveness of having an individual critic for each agent to handle variable degrees of asynchronicity in agents' high-level decision-making.

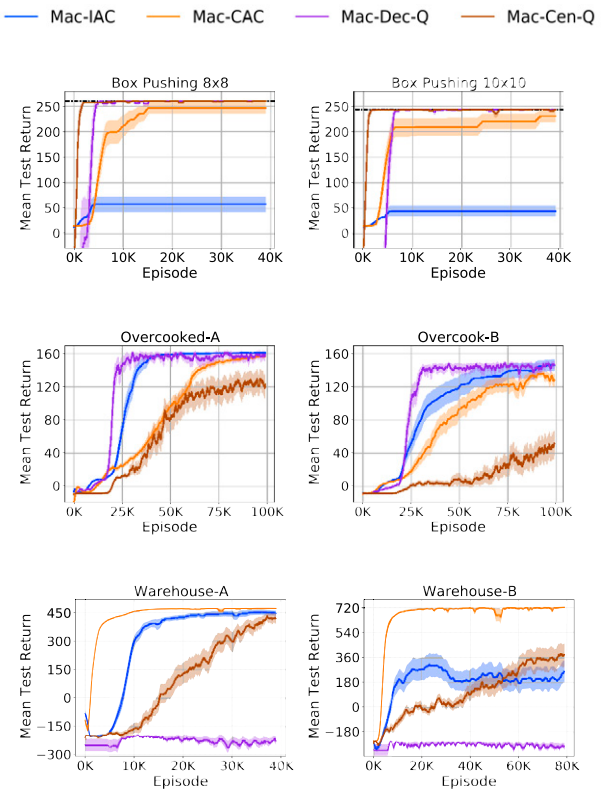


Figure 11. Comparisons of macro-action-based decentralized and centralized methods.

4.2.4. Comparative analysis between actor-critic and value-based approaches in decentralized and centralized training paradigms. Here, we compare our actor-critic methods (Mac-IAC and Mac-CAC) with the value-based approaches (Mac-Dec-Q and Mac-Cen-Q), shown in Figure 11. As aforementioned, the Box Pushing task requires agents to simultaneously reach the big box and push it together. This

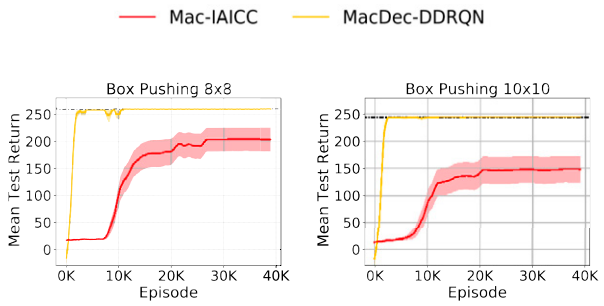


Figure 12. Comparisons of Mac-IAICC and MacDec-DDRQN.

consensus is rarely achieved when agents independently sample actions using stochastic policies in Mac-IAC and it is hard to learn from pure on-policy data. By having a replay buffer, value-based approaches show much stronger sample efficiency than on-policy actor-critic approaches in this domain with a small action space (the first row in Figure 11). Such an advantage is sustained by the decentralized value-based method (Mac-Dec-Q) but gets lost in the centralized one (Mac-Cen-Q) in the Overcooked domains due to a huge joint macro-action space (15^3) (the middle row in Figure 11). On the contrary, our actor-critic methods can scale to large domains and learn high-quality solutions. This is particularly noticeable in Warehouse-A, where the policy gradient methods quickly learn a high-quality policy while the centralized Mac-Cen-Q is slow to learn and the decentralized Mac-Dec-Q is unable to learn. In addition, the stochastic policies in actor-critic methods potentially have a better exploration property so that, in Warehouse domains, Mac-IAC can bypass an obvious local-optima that Mac-Dec-Q falls into, where the robot arm greedily chooses ***Wait-M*** to avoid more penalties.

4.2.5. Comparative analysis between actor-critic and value-based approaches in CTDE paradigm. We also conduct comparisons between our CTDE-based actor-critic method (Mac-IAICC) and our CTDE-base Q-learning methods (MacDec-DDRQN and Parallel-MacDec-DDRQN). In the Box Pushing task, we consider MacDec-DDRQN with a centralized ϵ-greedy policy for exploration, as it performs the best referred to Figure 10. Taking advantage of a replay buffer, MacDec-DDRQN learns much faster than Mac-IAICC (shown in the top row in Figure 12). Although Mac-IAICC possesses a centralized critic to provide a global action-value estimation, its overall performance is still limited by the on-policy data generated only in decentralized execution, where the bigger the world is, the lower the probability for sampling the aforementioned cooperation would be.

In the Overcooked domain, Parallel-MacDec-DDRQN cannot learn any good behaviors, which makes sense as fully decentralized learning has solved this problem very well (as shown in Figure 11). Also, in Figure 11, we have seen that the centralized Q-function learns quite slowly

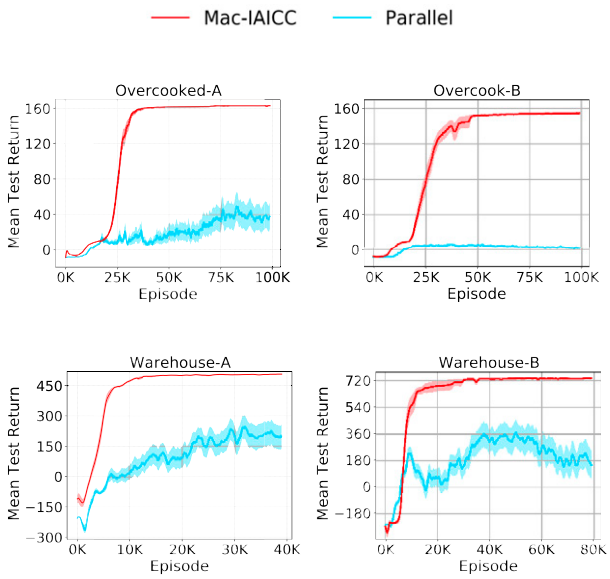


Figure 13. Comparisons of Mac-IAICC and parallel-MacDec-DDRQN.

due to the huge joint macro-action space, so it becomes the bottleneck in Parallel-MacDec-DDRQN such that it cannot offer a good target action for optimizing decentralized action-value functions and hurts the learning (the top row in Figure 13). Mac-IAICC successfully avoids the dilemma of the exponential joint action space by letting each agent learn an individual joint history-value function as the critic.

The results in Figure 10 have proved the necessity of having parallel environments to learn different Q-value functions in solving the warehouse task, we thus consider Parallel-MacDec-DDRQN in two warehouse scenarios and show the comparison with Mac-IAICC in Figure 13. According to the results of Mac-Dec-Q (purple curve) shown in Figure 11, we can conclude that the centralized Q-value function involved in Parallel-MacDec-DDRQN prevents the decentralized policies from a very bad local optimum. But eventually, the learned decentralized policies converge to another local optimum. We suspect the way of using the centralized Q-net to optimize decentralized policies (as in equation (24)) limits the improvement. One hypothesis is that the target action suggested by the centralized Q-net conditioning on joint information actually cannot always be reproduced by the decentralized Q-nets conditioning on only local information. Finally, Mac-IAICC's leading performance over three scenarios further demonstrates its strong scalability to large and long-horizon problems.

5. Experiments on hardware

5.1. Experimental setup

While evaluating the proposed approaches in simulation, we also extend the environment of Warehouse-A

(Figure 6(e)) to a hardware domain. Figure 14 provides an overview of the real-world experimental setup. An open area is divided into regions: a tool room, a corridor, and two workshops, to resemble the configuration shown in Figure 6(e). This mission involves one Fetch Robot (Wise et al., 2016) and two Turtlebots (Koubaa et al., 2016) to cooperatively find and deliver three YCB tools (Calli et al., 2015), in the order: a tape measure, a clamp, and an electric drill, required by each human in order to assemble an IKEA table. This real-world setup mirrors the simulated environment (Figure 6(e)) in a certain ratio in terms of warehouse dimensions and robot execution speeds. In the experiments, we train decentralized policies in the simulator and then deploy them on the real robots under the following two scenarios separately: (A) two humans work at the same speed on their assembly tasks; (B) the human in workshop-0 is faster than the other human on the assembly task.

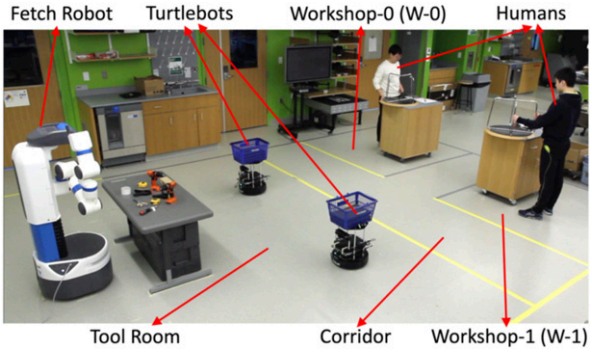


Figure 14. Overview of Warehouse-A hardware domain.

5.2. Results

Figure 15 shows the sequential collaborative behaviors of the robots in one hardware trial under scenario A. Fetch was able to find tools in parallel such that two tape measures (Figure 15(a)), two clamps (Figure 15(b)), and two electric drills, were found instead of finding all three types of tool for one human and then moving on to the other which would result in one of the humans waiting. Fetch's efficiency is also reflected in the behaviors such that it passed a tool to the Turtlebot that arrived first (Figure 15(b)) and continued to find the next tool when there was no Turtlebot waiting beside it (Figure 15(c)). Meanwhile, Turtlebots were clever such that they successfully avoided delayed delivery by sending tools one by one to the nearby workshop (e.g., T-0 focused on W-0 shown in Figures 15(b) and 15(d), and T-1 focused on W-1 shown in Figure 15(c)), rather than waiting for all tools before delivering, traveling a longer distance to serve the human at the diagonal, or prioritizing one of the humans altogether.

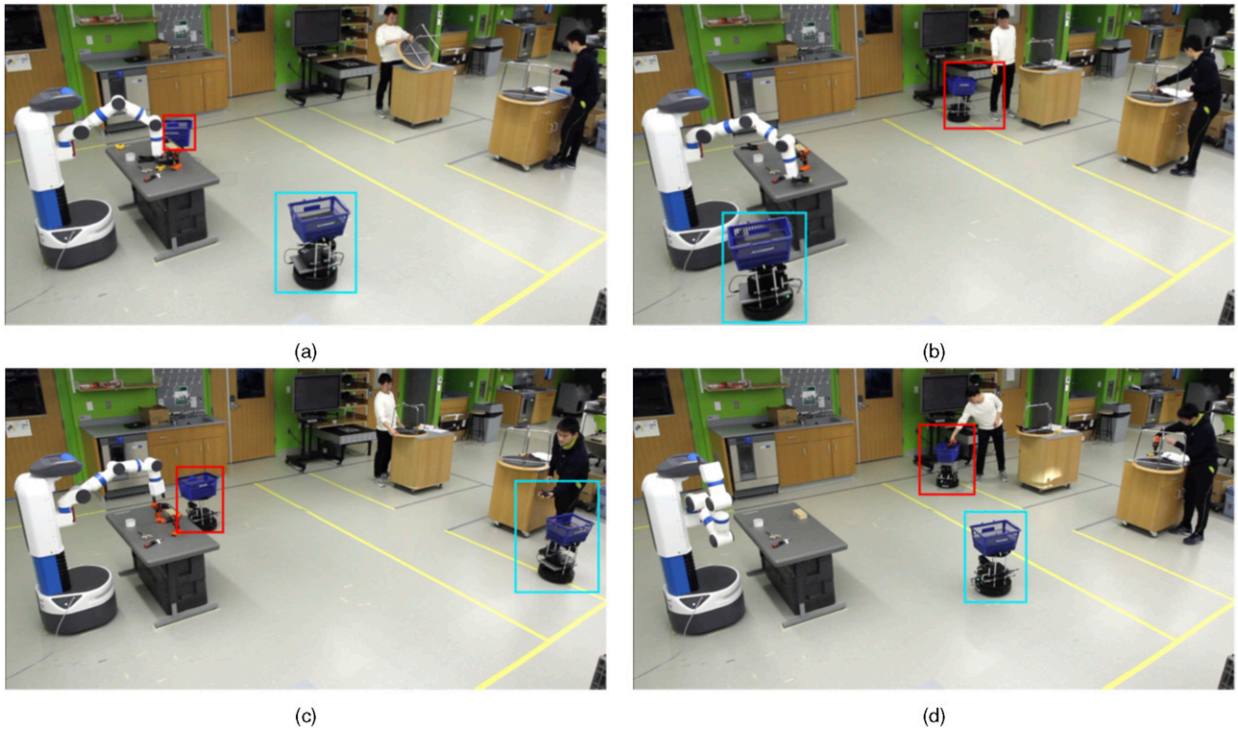


Figure 15. Collaborative behaviors generated by running the decentralized policies learned by Mac-IAICC in scenario A. Turtlebot-0 (T-0) is bounded in red and Turtlebot-1 (T-1) is bounded in blue. (a) After staging a tape measure at the left, Fetch looks for the second one while Turtlebots approach the table; (b) T-0 delivers a tap measure to W-0 and T-1 waits for a clamp from Fetch; (c) T-1 delivers a clamp to W-1, while T-0 carries the other clamp and goes to W-0, and Fetch searches for an electric drill; (d) T-0 delivers an electric drill (the last tool) to W-0 and the entire delivery task is completed.

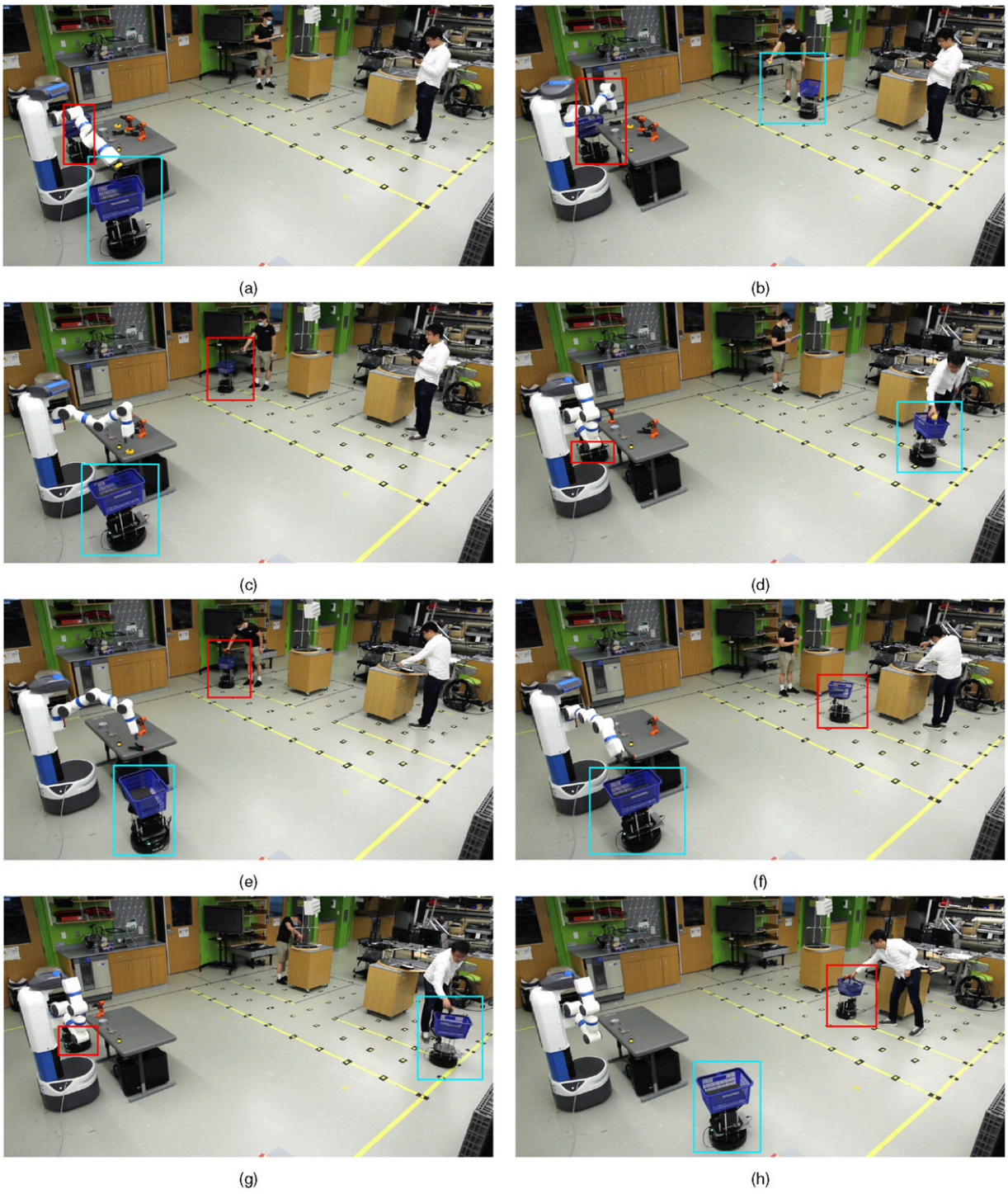


Figure 16. Collaborative behaviors generated by running the decentralized policies learned by Mac-IAICC in scenario B. Turtlebot-0 (T-0) is bounded in red and Turtlebot-1 (T-1) is bounded in blue. (a) Fetch passes a tape measure to T-1 and T-0 is waiting for a tool; (b) T-1 delivers a tape measure to W-0, while Fetch is passing a clamp to T-0; (c) T-0 delivers a clamp to W-0 and Fetch is passing the other tape measure to T-1; (d) T-1 delivers a tape measure to W-1, while T-0 returns the tool room; (e) T-0 sends an electric drill to W-0 and the faster human obtains all required tools, while Fetch finds the other clamp; (f) T-0 arrives at W-1 to observe the slower human's status, and Fetch passes the clamp to T-1; (g) T-1 delivers a clamp to W-1, and T-0 waits beside Fetch; (h) T-0 delivers an electric drill (the last tool) to the slower human and the entire delivery task is completed.

Figure 16 shows more complex and interesting collaborative behaviors under scenario B, where the team of robots prioritized the faster human (with a black shirt) and successfully delivered all tools in time. More concretely, Fetch was smart to successively find a tape measure (Figure 16(a)) and a clamp (Figure 16(b)) for the faster human first, followed by passing one tool to each Turtlebot, which gave Turtlebots a chance to deliver the tools

separately (Figure 16(b) and (c)) rather than letting only one Turtlebot send both tools that would lead to the human pausing there. Interestingly, instead of finding the third tool for the faster human, Fetch realized that it had to find the other tape measure (Figure 16(c)) to avoid a delayed delivery to the slower human, and then T-0 who received the tape measure from Fetch immediately transported it to the slower human (Figure 16(d)). Meanwhile, Fetch had staged the last tool, an electric drill (Figure 16(d)), that the faster human needed, and T-1 carried it to the W-0 at the right time (Figure 16(e)). After observing the faster human had received all necessary tools, T-1 was impressively clever such that it first went to W-1 to check the human's status (Figure 16(f)) and then collaborate with T-0 to assist the slower human together: T-0 conveyed the other clamp (Figure 16(g)) and T-1 eventually delivered the other electric drill (Figure 16(h)).

6. Related work

To scale up learning in MARL problems, hierarchy has been introduced into multi-agent scenarios. One line of hierarchical MARL is still focusing on learning primitive-action-based policy for each agent while leveraging a hierarchical structure to achieve knowledge transfer (Yang et al., 2021), credit assignment (Ahilan and Dayan, 2019), and low-level policy factorization over agent (Vezhnevets et al., 2020). In these works, as the decision-making over agents is still limited at the low-level, none of them has been evaluated in large-scale realistic domains. Instead, by having macro-actions, our methods equip agents with the potential capability of exploiting abstracted skills, sub-task allocation, and problem decomposition via hierarchical decision-making, which is critical for scaling up to real-world multi-robot tasks.

Another line of the research allows agents to learn both a high-level policy and a low-level policy, but the methods either force agents to perform a high-level choice at every time step (de Witt et al., 2019; Han et al., 2019) or require all agents' high-level decisions have the same time duration (Nachum et al., 2019; Wang et al., 2020b, 2021a; Xu et al., 2021; Yang et al., 2020a), where agents are actually synchronized at both levels. In contrast, our frameworks are more general and applicable to real-world multi-robot systems because they allow agents to asynchronously execute at a high level without synchronization or waiting for all agents to terminate.

Recently, some asynchronous hierarchical approaches have been developed. Wu et al., 2021a extend Deep Q-Networks (Mnih et al., 2015) to learn a high-level pixel-wise spatial-action-value map for each agent in a fully decentralized learning way. Our work, however, accepts any representations of high-level actions. Menda et al. (2019) frame multi-agent asynchronous decision-making problems as event-driven processes with one assumption on the acceptable of losing the ability to capture low-level interaction between agents within an event duration and the other on homogeneous agents, but our frameworks rely on the time-driven simulator used for general multi-agent and single-agent RL problems and do not have the above assumptions. Chakravorty et al. (2019) adapt a single-agent *option-critic* framework (Bacon et al., 2017) to multi-agent domains to learn all components (e.g., low-level policy, high-level abstraction, high-level policy) from scratch, but learning at both levels is difficult and the proposed method does not perform well even in small TeamGrid (Maxime and Julien, 2020) scenarios. Yu et al. (2023) directly extend MAPPO (Yu et al., 2022) to Dec-POSMDPs (Omidshafiei et al., 2017a), but the proposed framework is limited to homogeneous robots and they only consider spatial goal positions as macro-actions in navigation domains. Instead, our methods allow heterogeneous robots and accept any type of macro-actions. More important to note is that none of the existing works provides a general asynchronous and hierarchical multi-agent reinforcement learning framework to solve multi-agent problems with macro-actions under partial observability.

7. Conclusion

In this paper, we consider fully cooperative multi-agent systems where agents are allowed to asynchronously execute macro-actions under partial observability. Such asynchronicity matches the nature of real-world multi-robot behavior, and it also raises the key challenge of when to perform updates and what information to maintain in MARL with macro-actions. To address this challenge, we introduce the first formulation and approaches to extend deep Q-nets for learning decentralized and centralized macro-action-value functions, together with two new replay-buffers, Mac-CERTs and Mac-JERTs, to correctly capture agents' sequential macro-action-based experiences for asynchronous policy updates. These two approaches build up the base for developing MARL algorithms with macro-actions. Next, we present MacDec-DDRQN and Parallel-MacDec-DDRQN, the first set of value-based frameworks achieving CTDE with macro-actions, to learn better decentralized policies for solving complex tasks.

Since value-based algorithms do not scale well to large action spaces, we further formulated a set of macro-action-based actor-critic algorithms that allow agents to asynchronously optimize parameterized policies via policy gradients: a decentralized actor-critic method (Mac-IAC), a centralized actor-critic method (Mac-CAC), and two CTDE-based actor-critic methods (Naive Mac-IACC and Mac-IAICC). These are the first approaches to be able to incorporate controllers that may require different amounts of time to complete (macro-actions) in a general asynchronous multi-agent actor-critic framework.

Empirically, our methods are able to learn high-quality macro-action-based policies, allowing agents to perform asynchronous collaborations in a variety of multi-robot domains. Importantly, our most advanced method, Mac-IAICC, demonstrates its strong scalability and efficiency by achieving outstanding performance in long-horizon and large domains over other methods. Additionally, the practicality of Mac-IAICC is validated in a real-world multi-robot setup based on the warehouse domain.

Our formalism and methods open the door for other macro-action-based multi-agent reinforcement learning methods ranging from extensions of other current methods to new approaches and domains. We expect even more scalable learning methods that are feasible and flexible enough in solving realistic multi-robot problems.

Acknowledgments

We thank Dian Wang, Chengguang Xu and Xupeng Zhu for their helps on hardware experiments.

Declaration of conflicting interests

The author(s) declared the following potential conflicts of interest with respect to the research, authorship, and/or publication of this article: neu.edu.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This research is supported in part by the U.S. Office of Naval Research under award number N00014-19-1-2131, Army Research Office award W911NF20-1-0265 and NSF CAREER Award 2044993.

ORCID iD

Yuchen Xiao https://orcid.org/0000-0003-1038-1882

Supplemental Material

Supplemental material for this article is available online.

Note

1. Our approach could also be applied to other models with temporally-extended actions (Omidshafiei et al., 2017a).

References

Ahilan S and Dayan P (2019) Feudal multi-agent hierarchies for cooperative reinforcement learning. arXiv preprint abs/1901.08492.

Ahn M, Brohan A, Brown N, et al. (2022) Do as i can, not as i say: grounding language in robotic affordances.

Amato C, Konidaris GD and Kaelbling LP (2014) Planning with macro-actions in decentralized POMDPs. In: *Proceedings of the international conference on autonomous agents and multiagent systems*. Paris, France, 5-9 May 2014.

Amato C, Konidaris GD, Anders A, et al. (2015a) Policy search for multi-robot coordination under uncertainty. In: *Proceedings of the robotics: science and systems conference*. Rome, Italy, 13–17 July 2015.

Amato C, Konidaris GD, Cruz G, et al. (2015b) Planning for decentralized control of multiple robots under uncertainty. In: Proceedings of the international conference on robotics and automation, Seattle, WA, USA, 26–30 May 2015, pp. 1241–1248.

Amato C, Konidaris G, Kaelbling LP, et al. (2019) Modeling and planning with macro-actions in decentralized pomdps. *Journal of Artificial Intelligence Research* 64: 817–859.

Bacon P, Harb J and Precup O (2017) The option-critic architecture. In: Proceedings of the AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 41726–91734 Feb 2017.

Calli B, Walsman A, Singh A, et al. (2015) Benchmarking in manipulation research: using the Yale-CMU-Berkeley object and model set. *IEEE Robotics Automation Magazine* 22(3): 36–52.

Chakravorty J, Ward PN, Roy J, et al. (2019) Option-critic in cooperative multi-agent systems. arXiv preprint arXiv: 1911.12825.

Cho K, van Merrienboer B, Gülçehre Ç, et al. (2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation. Empirical Methods in Natural Language Processing (EMNLP), 1724–1734.

Dalal M, Pathak D and Salakhutdinov R (2021) Accelerating robotic reinforcement learning via parameterized action primitives. In: Proceedings of the conference on neural information processing systems.

de Witt CS, Foerster J, Farquhar G, et al. (2019) Multi-agent common knowledge reinforcement learning. In: Proceedings of the conference on neural information processing systems. Vancouver, BC, Canada, 8–14 Dec, 2019.

Du Y, Han L, Fang M, et al. (2019) Liir: learning individual intrinsic reward in multi-agent reinforcement learning. In: Proceedings of the conference on neural information processing systems. Vancouver, BC, Canda, 8–14 Dec 2019.

Foerster J, Farquhar G, Afouras T, et al. (2018) Counterfactual multi-agent policy gradients. In: Proceedings of the AAAI conference on artificial intelligence.

Fulda N and Ventura D (2007) Predicting and preventing coordination problems in cooperative q-learning systems. In: Proceedings of the international joint conference on artificial intelligence. Hyderabad, India, 6780–12785 Jan 2007.

Han D, Böhmer W, Wooldridge MJ, et al. (2019) Multi-agent hierarchical reinforcement learning with dynamic termination. In: *PRICAI (2), Lecture Notes in Computer Science*. Berlin: Springer, Vol. 11671, pp. 80–92.

Hasselt HV (2010) Double q-learning. In: *Proceedings of the conference on neural information processing systems*. Vancouver, BC, Canada, 62613–112621 Dec 2010.

Hasselt HV, Guez A and Silver D (2016) Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI conference on artificial intelligence. pp. 2094–2100.

Hausknecht M and Stone P (2015) Deep recurrent q-learning for partially observable mdps. In: AAAI fall symposium on

sequential decision making for intelligent agents (AAAI-SDMIA15).

He R, Bachrach A and Roy N (2010) Efficient planning under uncertainty for a target-tracking micro-aerial vehicle. In: Proceedings of the international conference on robotics and automation, Anchorage, AK, USA, 03–07 May 2010.

Hoang TN, Xiao Y, Sivakumar K, et al. (2018) Near-optimal adversarial policy switching for decentralized asynchronous multi-agent systems. In: Proceedings of the international conference on robotics and automation, Brisbane, QLD, Australia, 21–25 May 2018.

Hochreiter S and Schmidhuber J (1997) Long short-term memory. *Neural Computation* 9(8): 1735–1780.

Hsiao K, Kaelbling LP and Lozano-Perez T (2010) Task-driven tactile exploration. In: *Proceedings of the robotics: science and systems conference*. Zaragoza, Spain, 27–30 June 2010.

Iqbal S and Sha F (2019) Actor-attention-critic for multi-agent reinforcement learning. *Proceedings of the International Conference on Machine Learning* 97: 2961–2970.

Jolly K, Ravindran K, Vijayakumar R, et al. (2007) Intelligent decision making in multi-agent robot soccer system through compounded artificial neural networks. *Robotics and Autonomous Systems* 55(7): 589–596.

Konda VR and Tsitsiklis JN (2000) Actor-critic algorithms. In: Proceedings of the conference on neural information processing systems. pp. 1008–1014.

Konidaris GD, Kuindersma S, Grupen RA, et al. (2011) Autonomous skill acquisition on a mobile manipulator. *Proceedings of the AAAI Conference on Artificial Intelligence* 25(1): 1468–1473.

Konidaris GD, Kaelbling LP and Lozano-Pérez T (2018) From skills to symbols: learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research* 61: 215–289.

Koubaa A, Sriti MF, Javed Y, et al. (2016) Turtlebot at office: a service-oriented software architecture for personal assistant robots using ROS. 2016 international conference on autonomous robot systems and competitions (ICARSC), Braganca, 04–06 May 2016, pp. 270–276.

Kraemer L and Banerjee B (2016) Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing* 190: 82–94.

Lee Y, Cai P and Hsu D (2021) MAGIC: learning macro-actions for online POMDP planning. In: *Proceedings of the robotics: science and systems conference*. 12–16 July 2021.

Lin LJ (1992) Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning* 8(3): 293–321.

Liu X, Chen S, Aditya S, et al. (2018) Robust fruit counting: combining deep learning, tracking, and structure from motion. In: Proceedings of IEEE/RSJ international conference on intelligent robots and systems, Madrid, Spain, 01–05 October 2018, pp. 1045–1052.

Liu S, Lever G, Wang Z, et al. (2021) From motor control to team play in simulated humanoid football abs/2105.12196.

Lowe R, Wu Y, Tamar A, et al. (2017) Multi-agent actor-critic for mixed cooperative-competitive environments. Proceedings of the conference on neural information processing systems. Long Beach, CA, USA, 4–9 Dec 2017.

Lyu X, Xiao Y, Daley B, et al. (2021) Contrasting centralized and decentralized critics in multi-agent reinforcement learning. In: Proceedings of the international conference on autonomous agents and multiagent systems. 3–7 May 2021.

Mahajan A, Rashid T, Samvelyan M, et al. (2019) Maven: multi-agent variational exploration. In: *Proceedings of the conference on neural information processing systems*. Vancouver, BC, Canada, 87611–147622 Dec 2019.

Matignon L, Laurent GJ and Fort-Piat NL (2007) Hysteretic q-learning : an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In: Proceedings of IEEE/RSJ international conference on intelligent robots and systems, San Diego, CA, USA, 29 October 2007–02 November 2007, pp. 64–69.

Maxime CB and Julien R (2020) Teamgrid. https://github.com/mila-iqia/teamgrid.

Menda K, Chen Y, Grana J, et al. (2019) Deep reinforcement learning for event-driven multi-agent decision processes. *IEEE Trans. Intell. Transp. Syst* 20(4): 1259–1268.

Mnih V, Kavukcuoglu K, Silver D, et al. (2015) Human-level control through deep reinforcement learning. *Nature* 518: 529–533.

Murray CC and Raj R (2020) The multiple flying sidekicks traveling salesman problem: parcel delivery with multiple drones. *Transportation Research Part C: Emerging Technologies* 110: 368–398.

Nachum O, Ahn M, Ponte H, et al. (2019) Multi-agent manipulation via locomotion using hierarchical sim2real. In: Proceedings of the conference on robot learning. Osaka, Japan, 30 Oct - 1 Jan 2019.

Oliehoek FA and Amato C (2016) *A Concise Introduction to Decentralized POMDPs*. Incorporated: Springer Publishing Company.

Oliehoek FA, Spaan MTJ and Vlassis NA (2008) Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research* 32: 289–353.

Omidshafiei S, Agha-mohammadi A, Amato C, et al. (2016) Graph-based cross entropy method for solving multi-robot decentralized POMDPs. In: Proceedings of the international conference on robotics and automation, Stockholm, Sweden, 16–21 May 2016.

Omidshafiei S, Agha-mohammadi A, Amato C, et al. (2017a) Decentralized control of multi-robot partially observable markov decision processes using belief space macro-actions. *The International Journal of Robotics Research* 36(2): 231–258.

Omidshafiei S, Pazis J, Amato C, et al. (2017b) Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In: Proceedings of the international conference on machine learning, Sydney, Australia, 62681–112690 Aug 2017.

Queralta JP, Taipalmaa J, Can Pullinen B, et al. (2020) Collaborative multi-robot search and rescue: planning, coordination, perception, and active vision. *IEEE Access* 8: 191617–191643.

Rashid T, Samvelyan M, de Witt CS, et al. (2018) Qmix: monotonic value function factorisation for deep multi-agent reinforcement learning. In: Proceedings of the International Conference on Machine Learning. Stockholm, Sweden, 10–15 July 2018.

Rashid T, Farquhar G, Peng B, et al. (2020) Weighted qmix: expanding monotonic value function factorisation. In: Proceedings of the conference on neural information processing systems. 6–12 Dec 2020.

Rosenband DL (2017) Inside waymo's self-driving car: my favorite transistors. 2017 symposium on VLSI circuits, Kyoto, Japan, 05–08 June 2017, pp. C20–C22.

Simões D, Lau N and Reis LP (2017) Multi-agent double deep Q-networks *Progress in Artificial Intelligence*. Cham: Springer International Publishing, 123–134.

Son K, Kim D, Kang WJ, et al. (2019) QTRAN: learning to factorize with transformation for cooperative multi-agent reinforcement learning. In: Proceedings of the international conference on machine learning. Long Beach, CA, 10–15 June 2019.

Stulp F and Schaal S (2011) Hierarchical reinforcement learning with movement primitives. In: 11th IEEE-RAS international conference on humanoid robots, Bled, Slovenia, 26–28 October 2011.

Su J, Adams S and Beling PA (2021) Value-decomposition multi-agent actor-critics. In: Proceedings of the AAAI conference on artificial intelligence.

Sunehag P, Lever G, Gruslys A, et al. (2018) Value-decomposition networks for cooperative multi-agent learning based on team reward. In: *Proceedings of the international conference on autonomous agents and multiagent systems*. Stockholm, Sweden, pp. 102085–152087 July 2018.

Sutton RS (1988) Learning to predict by the methods of temporal differences. *Machine Learning* 3: 9–44.

Sutton RS, Precup D and Singh S (1998) Intra-option learning about temporally abstract actions. In: Proceedings of the International Conference on Machine Learning. Madison, Wisconsin, USA, 24–27 July 1998.

Sutton R, Precup D and Singh S (1999) Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112: 181–211.

Sutton RS, McAllester DA, Singh SP, et al. (2000) Policy gradient methods for reinforcement learning with function approximation. In: *Advances in Neural Information Processing Systems*, 1057–1063.

Tan M (1993) Multi-agent reinforcement learning: independent vs. cooperative agents. In: *Proceedings of the international conference on machine learning*. Amherst, MA, USA, 27330–29337 June 1993.

Tang YC (2019) Towards learning multi-agent negotiations via self-play. In: Autonomous driving workshop, IEEE international conference on computer vision, Seoul, Republic of Korea, 27–28 October 2019.

Theocharous G and Kaelbling L (2004) Approximate planning in pomdps with macro-actions. In: *Advances in Neural Information Processing Systems*.

Vezhnevets AS, Wu Y, Leblond R, et al. (2020) Options as responses: grounding behavioural hierarchies in multi-agent rl. In: Proceedings of the international conference on machine learning. 12–18 July 2020.

Wang J, Ren Z, Liu T, et al. (2021c) Qplex: duplex dueling multi-agent q-learning. In: Proceedings of the international conference on learning representations. 3–7 May 2021.

Wang J, Zhang Y, Kim TK, et al. (2020a) Shapley q-value: a local reward approach to solve global reward games. In: Proceedings of the AAAI conference on artificial intelligence.

Wang T and Dong H, and Victor Lesser CZ (2020c) Roma: multi-agent reinforcement learning with emergent roles. In: Proceedings of the international conference on machine learning. 12–18 July 2020.

Wang T, Gupta T, Mahajan A, et al. (2021a) Rode: learning roles to decompose multi-agent tasks. In: Proceedings of the international conference on learning representations. 3–7 May 2021.

Wang Y, Han B, Wang T, et al. (2021b) DOP: off-policy multi-agent decomposed policy gradients. In: Proceedings of the international conference on learning representations. 3–7 May 2021.

Wang RE, Kew JC, Lee D, et al. (2020b) Model-based reinforcement learning for decentralized multiagent rendezvous. In: Proceedings of the conference on robot learning. 16–18 Nov 2020.

Watkins CJCH and Dayan P (1992) Q-learning. *Machine Learning* 8(3): 279–292.

Weaver L and Tao N (2001) The optimal reward baseline for gradient-based reinforcement learning. In: Proceedings of the conference on uncertainty in artificial intelligence. Morgan Kaufmann, pp. 538–545.

Wise M, Ferguson M, King D, et al. (2016) Fetch & freight: standard platforms for service robot applications. In: Workshop on autonomous mobile service robots, international joint conference on artificial intelligence.

Wu J, Sun X, Zeng A, et al. (2020) Spatial action maps for mobile manipulation. In: *Proceedings of the robotics: science and systems conference*. 12–16 July 2020.

Wu J, Sun X, Zeng A, et al. (2021a) Spatial intention maps for multi-agent mobile manipulation. In: Proceedings of the international conference on robotics and automation. Xi'an, China, 30 May – 5 June 2021.

Wu SA, Wang RE, Evans JA, et al. (2021b) Too many cooks: coordinating multi-agent collaboration through inverse planning. *Topics in Cognitive Science*.

Xiao Y, Hoffman J and Amato C (2019a) Macro-action-based deep multi-agent reinforcement learning. In: Proceedings of the conference on robot learning. Osaka, Japan, 30 Oct – 1 Nov 2019.

Xiao Y, Katt S, ten Pas A, et al. (2019b) Online planning for target object search in clutter under partial observability. In: Proceedings of the international conference on robotics and automation, Montreal, QC, Canada, 20–24 May 2019.

Xiao Y, Tan W and Amato C (2022) - *9* 28. (accessed Nov).
Xu Z, Bai Y, Zhang B, et al. (2021) HAVEN: hierarchical cooperative multi-agent reinforcement learning with dual coordination mechanism. arXiv preprint abs/2110.07246.
Yang J, Borovikov I and Zha H (2020a) Hierarchical cooperative multi-agent reinforcement learning with skill discovery. In: Proceedings of the international conference on autonomous agents and multiagent systems. 9–13 May 2020.
Yang J, Nakhaei A, Isele D, et al. (2020b) *Cm3: cooperative multi-goal multi-stage multi-agent reinforcement learning*. In: Proceedings of The International Conference On Learning Representations. Addis Ababa, Ethiopia, 26–30 April 2020.
Yang T, Wang W, Tang H, et al. (2021) An efficient transfer learining framework for multiagent reinforcement learining. In: Proceedings of the conference on neural information processing systems. 6–14 Dec 2021.
Yu C, Velu A, Vinitsky E, et al. (2022) - *9* 28. (accessed Nov).
Yu C, Yang X, Gao J, et al. (2023) Asynchronous multi-agent reinforcement learning for efficient real-time multi-robot cooperative exploration. In: *Proceedings of the international conference on autonomous agents and multiagent systems*. London, UK, 29 May – 2 June 2023.
Zheng Y, Meng Z, Hao J, et al. (2018) Weighted double deep multiagent reinforcement learning in stochastic cooperative environments. In: Pacific rim international conference on artificial intelligence. pp. 421–429.
Zhou M, Liu Z, Sui P, et al. (2020) Learning implicit credit assignment for cooperative multi-agent reinforcement learning. In: Proceedings of the conference on neural information processing systems. 6–12 Dec 2020.

Appendix

A. Appendix

A.1. Macro-action-based policy gradient theorem. As POMDPs can always be transformed to history-based MDPs, we can directly adapt the general Bellman equation for the state values of a hierarchical policy (Sutton et al., 1999) to a macro-action-based POMDP by replacing the state s with a history h as follows (for keeping the notation simple, we use τ to represent the number of time steps taken by the corresponding macro-action m, and we use h to represent macro-observation-action history):

$$V^{\Psi}(h) = \sum_{m} \Psi(m|h) Q^{\Psi}(h,m) \tag{33}$$

$$Q^{\Psi}(h,m) = r^{c}(h,m) + \sum_{h'} P(h'|h,m) V^{\Psi}(h') \tag{34}$$

where

$$r^{c}(h,m) = \mathbb{E}_{\tau\sim\beta_m, s_{t_m}|h}\left[\sum_{t=t_m}^{t_m+\tau-1} \gamma^{t} r_t\right] \tag{35}$$

$$P(h'|h,m) = P(z'|h,m) = \sum_{\tau=1}^{\infty} \gamma^{\tau} P(z',\tau|h,m) \tag{36}$$

$$= \sum_{\tau=1}^{\infty} \gamma^{\tau} P(\tau|h,m) P(z'|h,m,\tau) \tag{37}$$

$$= \sum_{\tau=1}^{\infty} \gamma^{\tau} P(\tau|h,m) P(z'|h,m,\tau) \tag{38}$$

$$= \mathbb{E}_{\tau\sim\beta_m}\left[\gamma^{\tau}\mathbb{E}_{s|h}\left[\mathbb{E}_{s'|s,m,\tau}[P(z'|m,s')]\right]\right] \tag{39}$$

Next, we follow the proof of the policy gradient theorem (Sutton et al., 2000):

$$\nabla_{\theta} V^{\Psi_\theta}(h) = \nabla_{\theta}\left[\sum_{m} \Psi_{\theta}(m|h) Q^{\Psi_\theta}(h,m)\right] \tag{40}$$

$$= \sum_{m}\Big[\nabla_{\theta}\Psi_{\theta}(m|h) Q^{\Psi_\theta}(h,m) + \Psi_{\theta}(m|h)\nabla_{\theta} Q^{\Psi_\theta}(h,m)\Big] \tag{41}$$

$$= \sum_{m}\Big[\nabla_{\theta}\Psi_{\theta}(m|h) Q^{\Psi_\theta}(h,m) + \Psi_{\theta}(m|h)\nabla_{\theta}\big(r^{c}(h,m) + \sum_{h'} P(h'|h,m) V^{\Psi_\theta}(h')\big)\Big] \tag{42}$$

$$= \sum_{m}\Big[\nabla_{\theta}\Psi_{\theta}(m|h) Q^{\Psi_\theta}(h,m) + \Psi_{\theta}(m|h)\sum_{h'} P(h'|h,m)\nabla_{\theta} V^{\Psi_\theta}(h')\Big] \tag{43}$$

$$= \sum_{\widehat{h}\in H}\sum_{k=0}^{\infty} P(h\to\widehat{h},k,\Psi_\theta) \sum_{m}\nabla_{\theta}\Psi_{\theta}(m|\widehat{h}) Q^{\Psi_\theta}(\widehat{h},m)\,(\text{afterrepeatedunrolling}) \tag{44}$$

Then, we can have:

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} V^{\Psi_\theta}(h_0) \tag{45}$$

$$= \sum_{h\in H}\sum_{k=0}^{\infty} P(h_0\to h,k,\Psi_\theta)\sum_{m}\nabla_{\theta}\Psi_{\theta}(m|h) Q^{\Psi_\theta}(h,m) \tag{46}$$

$$= \sum_{h}\rho^{\Psi_\theta}(h)\sum_{m}\nabla_{\theta}\Psi_{\theta}(m|h) Q^{\Psi_\theta}(h,m) \tag{47}$$

$$= \sum_{h}\rho^{\Psi_\theta}(h)\sum_{m}\Psi_{\theta}(m|h)\nabla_{\theta}\log\Psi_{\theta}(m|h) Q^{\Psi_\theta}(h,m) \tag{48}$$

$$= \mathbb{E}_{h\sim\rho^{\Psi_\theta}, m\sim\Psi_\theta}\left[\nabla_\theta \log \Psi_\theta(m|h) Q^{\Psi_\theta}(h,m)\right] \tag{49}$$

B. Theoretical analysis on trajectory squeezing

We provide a theoretical analysis about the idea of squeezing sequential experience based on the termination of macro-actions and the macro-action-based policy optimization in our proposed methods. Let us use the example given in Amato et al. (2019) and perform a further derivation. We consider a two-agent joint macro-action-based policy $\vec{\Psi}$ and agents begins with macro-actions $\langle m_1, m_2\rangle$ at state s, we can have

$$Q^{\vec{\Psi}}(m_1,m_2,s) = \sum_{o_1,o_2} O(o_1,o_2,a_1,a_2,s) \times \sum_{a_1,a_2} \pi_{m_1}(a_1|o_1)\pi_{m_2}(a_2|o_2)\Big[R(a_1,a_2,s) \tag{50}$$

$$+\gamma \sum_{s'} T(s',a_1,a_2,s) \sum_{o_1',o_2'} O(o_1',o_2',a_1,a_2,s') \tag{51}$$

$$\Big((1-\beta_{m_1}(o_1'))(1-\beta_{m_2}(o_2'))Q^{\vec{\Psi}}(m_1,m_2,s') \quad \text{(neither terminate)} \tag{52}$$

$$+\beta_{m_1}(o_1')\beta_{m_2}(o_2') \sum_{m_1',m_2'} \Psi_1(m_1'|o_1')\Psi_2(m_2'|o_2')Q^{\vec{\Psi}} \quad (m_1',m_2',s') \text{ (both terminate)} \tag{53}$$

$$+\beta_{m_1}(o_1')\big(1-\beta_{m_2}(o_2')\big)\sum_{m_1'}\Psi_1(m_1'|o_1')Q^{\vec{\Psi}}(m_1',m_2,s') \quad \text{(agent 1 terminates)} \tag{54}$$

$$+(1-\beta_{m_1}(o_1'))\beta_{m_2}(o_2')\sum_{m_2'}\Psi_2(m_2'|o_2')Q^{\vec{\Psi}}(m_1,m_2',s') \quad \text{(agent 2 terminates)}\Big)\Big] \tag{55}$$

For simplicity, the above example uses single observation instead of observation-action history, which can be easily extended to the history case. If we continue unrolling the "neither terminate" component, like $Q^{\vec{\Psi}}(m_1,m_2,s')$ in equation (2), we will eventually have

$$Q^{\vec{\Psi}}(m_1,m_2,s) = R(m_1,m_2,s) + \sum_{s',o_1',o_2'} p^{m_1,m_2}_{s,o_1,o_2,s',o_1',o_2'} \sum_{m_1',m_2'} \Psi_1(m_1'|o_1')\Psi_2(m_2'|o_2')Q^{\vec{\Psi}}(m_1',m_2',s') \tag{56}$$

where

$$R(m_1,m_2,s) = \mathbf{E}[R_1 + \gamma R_2 + \cdots + \gamma^{\tau-1}R_\tau \mid S_0 = s, A_{1(0:\tau-1)}\sim\pi_{m_1}, A_{2(0:\tau-1)}\sim\pi_{m_2}] \tag{57}$$

$$p^{m_1,m_2}_{s,o_1,o_2,s',o_1',o_2'} = \sum_{\tau=1}^{\infty}\gamma^\tau p(s',\tau)O(o_1',o_2',s') \tag{58}$$

where $p(s',\tau$ is the probability that the joint macro-action terminates in s' after k steps.

Therefore, according to equation (57), we choose to squeeze experience in both decentralized and centralized learning approaches; according to equations (52)–(55), we define termination of a joint macro-action as when any agent's macro-action is completed; according to equation (56), the single agent optimal value functions and optimal Bellman equations to options and to policies over options. Sutton et al. (1999) can be easily extended to both decentralized policy and centralized policy optimization in our cases.

Note that, the algorithms proposed in this paper do not explicitly keep the time information, but during learning, the observations and rewards are sampled with the probability in the above equations. Thus, we are still evaluating policies correctly. It would be an interesting future research direction to develop methods that remember the time information, which would end up with a richer policy class and could potentially do better than our squeezing approach. Future methods could keep track of times steps or macro-observations at each step but this would also be more complicated. Our approach is correct (i.e., it can learn accurate values for macro-action policies), but other choices are possible that use more (or less) information.

C. Asynchronous actor-critic algorithms

In this section, we present the pseudo code of each proposed macro-action-based actor-critic algorithm. We describe all methods in the on-policy learning manner while off-policy learning can be achieved by applying importance sampling weights and not resetting the buffer.

Macro-Action-Based Independent Actor-Critic (Mac-IAC):

Algorithm 2 Mac-IAC

Initialize a decentralized policy network for each agent i: Ψ_{θ_i}
Initialize decentralized critic networks for each agent i: $V_{\mathbf{w}_i}^{\Psi_{\theta_i}}$, $V_{\mathbf{w}_i^-}^{\Psi_{\theta_i}}$
Initialize a buffer $\mathcal{D}$
for *episode* = 1 to M **do**
  $t = 0$
  Reset env
  while not reaching a terminal state **and** $t < \mathbb{H}$ **do**
    $t \leftarrow t + 1$
    for each agent i **do**
      if the macro-action m_i is terminated **then**
        $m_i \sim \Psi_{\theta_i}(\cdot \mid h_i; \epsilon)$
      else
        Continue running current macro-action m_i
    for each agent i **do**
      Get cumulative reward r_i^c, next macro-observation z_i'
      Collect $\langle z_i, m_i, z_i', r_i^c \rangle$ into the buffer $\mathcal{D}$
  if *episode* mod $I_{\text{train}} = 0$ **then**
    for each agent i **do**
      Squeeze agent i's trajectories in the buffer $\mathcal{D}$
      Perform a gradient decent step on $L(\mathbf{w}_i) = \big(y - V_{\mathbf{w}_i}^{\Psi_{\theta_i}}(h_i)\big)_{\mathcal{D}}^2$, where $y = r_i^c + \gamma^{\tau_{m_i}} V_{\mathbf{w}_i^-}^{\Psi_{\theta_i}}(h_i')$
      Perform a gradient ascent on:
      $\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\vec{\Psi}_{\vec{\theta}}}\Big[\nabla_{\theta_i} \log \Psi_{\theta_i}(m_i|h_i)\big(r_i^c + \gamma^{\tau_{m_i}} V_{\mathbf{w}_i^-}^{\Psi_{\theta_i}}(h_i') - V_{\mathbf{w}_i}^{\Psi_{\theta_i}}(h_i)\big)\Big]$
    Reset buffer $\mathcal{D}$
  if *episode* mod $I_{\text{TargetUpdate}} = 0$ **then**
    for each agent i **do**
      Update the critic target network $\mathbf{w}_i^- \leftarrow \mathbf{w}_i$

Macro-Action-Based Centralized Actor-Critic (Mac-CAC):

Algorithm 3 Mac-CAC

Initialize a centralized policy network: Ψ_θ
Initialize centralized critic networks: $V_{\mathbf{w}}^{\Psi_\theta}$, $V_{\mathbf{w}^-}^{\Psi_\theta}$
Initialize a centralized buffer $\mathcal{D} \leftarrow$ Mac-JERTs,
for *episode* = 1 to M **do**
  $t = 0$
  Reset env
  while not reaching a terminal state **and** $t < \mathbb{H}$ **do**
    $t \leftarrow t + 1$
    if the joint macro-action $\vec{m}$ is terminated **then**
      $\vec{m} \sim \Psi_\theta(\cdot \mid \vec{h}, \vec{m}^{\text{undone}}; \epsilon)$
    else
      Continue running current joint macro-action $\vec{m}$
    Get a joint cumulative reward $\vec{r}^c$, next joint macro-observation $\vec{z}'$
    Collect $\langle \vec{z}, \vec{m}, \vec{z}', \vec{r}^c \rangle$ into the buffer $\mathcal{D}$
  if *episode* mod $I_{\text{train}} = 0$ **then**
    Squeeze joint macro-level trajectories in the buffer $\mathcal{D}$ according to joint macro-action terminations
    Perform a gradient decent step on $L(\mathbf{w}) = \big(y - V_{\mathbf{w}}^{\Psi_\theta}(\vec{h})\big)_{\mathcal{D}}^2$, where $y = \vec{r}^c + \gamma^{\vec{\tau}_{\vec{m}}} V_{\mathbf{w}^-}^{\Psi_\theta}(\vec{h}')$
    Perform a gradient ascent on $\nabla_\theta J(\theta) = \mathbb{E}_{\Psi_\theta}\Big[\nabla_\theta \log \Psi_\theta(\vec{m} \mid \vec{h})\big(\vec{r}^c + \gamma^{\vec{\tau}_{\vec{m}}} V_{\mathbf{w}^-}^{\Psi_\theta}(\vec{h}') - V_{\mathbf{w}}^{\Psi_\theta}(\vec{h})\big)\Big]$
    Reset buffer $\mathcal{D}$
  if *episode* mod $I_{\text{TargetUpdate}} = 0$ **then**
    Update the critic target network $\mathbf{w}^- \leftarrow \mathbf{w}$

where $\vec{m}^{\text{undone}}$ is the sub-joint-macro-action over the agents who have not terminated their macro-actions and will continue running.

Naive Mac-IACC:

In the pseudo code of Naive Mac-IACC presented below, we assume the accessible centralized information $\mathbf{x}$ is joint macro-observation-action history in the centralized critic.

Algorithm 4 Naive Mac-IACC

Initialize a decentralized policy network for each agent i: Ψ_{θ_i}
Initialize centralized critic networks: $V_{\mathbf{w}}^{\vec{\Psi}_{\vec{\theta}}}$, $V_{\mathbf{w}^-}^{\vec{\Psi}_{\vec{\theta}}}$
Initialize a decentralized buffer $\mathcal{D} \leftarrow$ Mac-JERTs,
for *episode* = 1 to M **do**
  $t = 0$
  Reset env
  while not reaching a terminal state **and** $t < \mathbb{H}$ **do**
    $t \leftarrow t + 1$
    for each agent i **do**
      if the macro-action m_i is terminated **then**
        $m_i \sim \Psi_{\theta_i}(\cdot \mid h_i; \epsilon)$
      else
        Continue running current macro-action m_i
    Get a reward $\vec{r}^c$ accumulated based on current joint macro-action termination
    Get next joint macro-observations $\vec{z}'$
    Collect $\langle \vec{z}, \vec{m}, \vec{z}', \vec{r}^c \rangle$ into the buffer $\mathcal{D}$
  if *episode* mod $I_{\text{train}} = 0$ **then**
    Squeeze joint macro-level trajectories in the buffer $\mathcal{D}$ according to joint macro-action terminations
    Perform a gradient decent step on $L(\mathbf{w}) = \big(y - V_{\mathbf{w}}^{\vec{\Psi}_{\vec{\theta}}}(\vec{h})\big)^2_{\mathcal{D}}$, where $y = \vec{r}^c + \gamma^{\vec{\tau}_{\vec{m}}} V_{\mathbf{w}^-}^{\vec{\Psi}_{\vec{\theta}}}(\vec{h}')$
    for each agent i **do**
      Squeeze agent i's trajectories in the buffer $\mathcal{D}$ according to its own macro-action terminations
      Perform a gradient ascent on:
      $\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\vec{\Psi}_{\vec{\theta}}}\Big[\nabla_{\theta_i} \log \Psi_{\theta_i}(m_i|h_i)\big(\vec{r}^c + \gamma^{\vec{\tau}_{\vec{m}}} V_{\mathbf{w}^-}^{\vec{\Psi}_{\vec{\theta}}}(\vec{h}') - V_{\mathbf{w}}^{\vec{\Psi}_{\vec{\theta}}}(\vec{h})\big)\Big]$
    Reset buffer $\mathcal{D}$
  if *episode* mod $I_{\text{TargetUpdate}} = 0$ **then**
    Update the critic target network $\mathbf{w}^- \leftarrow \mathbf{w}$

Macro-Action-Based Independent Actor with Individual Centralized Critic (Mac-IAICC):

In the pseudo code of Mac-IAICC presented below, we assume the accessible centralized information **x** is joint macro-observation-action history in the centralized critic.

Algorithm 5 Mac-IAICC

Initialize a decentralized policy network for each agent i: Ψ_{θ_i}
Initialize centralized critic networks for each agent i: $V_{\mathbf{w}_i}^{\vec{\Psi}_{\vec{\theta}}}$, $V_{\mathbf{w}_i^-}^{\vec{\Psi}_{\vec{\theta}}}$
Initialize a decentralized buffer $\mathcal{D}$
for *episode* = 1 to M **do**
  $t = 0$
  Reset env
  while not reaching a terminal state **and** $t < \mathbb{H}$ **do**
    $t \leftarrow t + 1$
    for each agent i **do**
      if the macro-action m_i is terminated **then**
        $m_i \sim \Psi_{\theta_i}(\cdot \mid h_i; \epsilon)$
      else
        Continue running current macro-action m_i
    for each agent i **do**
      Get a reward r_i^c accumulated based on agent i's macro-action termination
    Get next joint macro-observations $\vec{z}'$
    Collect $\langle \vec{z}, \vec{m}, \vec{z}', \{r_1^c, \ldots, r_n^c\} \rangle$ into the buffer $\mathcal{D}$
  if *episode* mod $I_{\text{train}} = 0$ **then**
    for each agent i **do**
      Squeeze trajectories in the buffer $\mathcal{D}$ according to joint macro-action terminations
      Compute the TD-error of each time step in the squeezed experiences:
      $L(\mathbf{w}_i) = \big(y - V_{\mathbf{w}_i}^{\vec{\Psi}_{\vec{\theta}}}(\vec{h})\big)^2_{\mathcal{D}}$, where $y = r_i^c + \gamma^{\tau_{m_i}} V_{\mathbf{w}_i^-}^{\vec{\Psi}_{\vec{\theta}}}(\vec{h}')$
      Perform a gradient descent only over the TD-errors when agent i's macro-action is terminated
      Squeeze agent i's trajectories in the buffer $\mathcal{D}$ according to its own macro-action terminations
      Perform a gradient ascent on:
      $\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\vec{\Psi}_{\vec{\theta}}}\Big[\nabla_{\theta_i} \log \Psi_{\theta_i}(m_i|h_i)\big(r_i^c + \gamma^{\tau_{m_i}} V_{\mathbf{w}_i^-}^{\vec{\Psi}_{\vec{\theta}}}(\vec{h}') - V_{\mathbf{w}_i}^{\vec{\Psi}_{\vec{\theta}}}(\vec{h})\big)\Big]$
    Reset buffer $\mathcal{D}$
  if *episode* mod $I_{\text{TargetUpdate}} = 0$ **then**
    for each agent i **do**
      Update the critic target network $\mathbf{w}_i^- \leftarrow \mathbf{w}_i$

D. Domain descriptions

D.1. Box Pushing. **Goal**. The objective of the two robots is to learn collaboratively push the middle big box to the goal area at the top rather than pushing (Figure 17) a small box on each own.

State Space (*S*). The global state information consists of the position and orientation of each robot and each box's position in a grid world.

Primitive-Action Space (*A*). *Move forward*, *turn-left*, *turn-right*, and *stay*.

Macro-Action Space (*M*).

• One-step macro-actions: ***Turn-left***, ***Turn-right***, and ***Stay***.

• Multi-step macro-actions: ***Move-to-small-box(i)*** that navigates the robot to the red spot below the corresponding small box and terminates with robot facing the box; ***Move-to-big-box(i)*** that navigates the robot to a red spot below the big box and terminate with robot facing the big box; ***Push*** that operates the robot to keep moving forward and terminate while arriving the world's boundary, touching the big box along or pushing a small box to the goal.

Observation Space (*O* and *Z*). In both the primitive-observation and macro-observation, each robot is only allowed to capture one of five states of the cell in front of it: *empty*, *teammate*, *boundary*, *small box*, *big box*.

Dynamics (*T*). The transition in this task is deterministic. Boxes can only be moved toward the north when the robot faces the box and moves forward. The small box can be moved by a single robot while the big box requires two robots to move it together.

Rewards (*R*). The team receives +300 for pushing big box to the goal area and +20 for pushing a small box to the goal area. A penalty −10 is issued when any robot hits the boundary or pushes the big box on its own.

Episode Termination. Each episode terminates when any box is pushed to the goal area, or when 100 time steps have elapsed.

D.2. Overcooked. **Goal**. Three robots need to learn cooperating with each other to prepare a Tomato-Lettuce-Onion salad and deliver it to the "star" counter cell as soon as possible. The challenge is that the recipe of making a tomato-lettuce-onion salad is unknown to robots. Robots have to learn the correct procedure in terms of picking up raw vegetables, chopping, and merging in a plate before delivering.

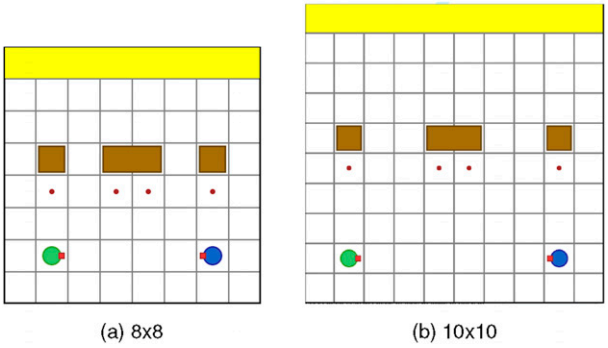


Figure 17. Experimental environments.

State Space (*S*). The environment is a 7 × 7 grid world involving three robots, one tomato, one lettuce, one onion, two plates, two cutting boards, and one delivery cell. The global state information consists of the positions of each robot and above items, and the status of each vegetable: chopped, unchopped, or the progress under chopping.

Primitive-Action Space (*A*). Each robot has five primitive-actions: *up*, *down*, *left*, *right* and *stay*. Robots can move around and achieve picking, placing, chopping and delivering by standing next to the corresponding cell and moving against it (e.g., in Figure 18(a), the pink robot can *move right* and then *move up* to pick up the tomato).

Macro-Action Space (*M*). Here, we first describe the main function of each macro-action and then list the corresponding termination conditions.

- Five one-step macro-actions that are the same as the primitive ones;
- ***Chop***, cuts a raw vegetable into pieces (taking three time steps) when the robot stands next to a cutting board and an unchopped vegetable is on the board, otherwise it does nothing; and it terminates when:
  - The vegetable on the cutting board has been chopped into pieces;
  - The robot is not next to a cutting board;
  - There is no unchopped vegetable on the cutting board;
  - The robot holds something in hand.
- ***Get-Lettuce***, ***Get-Tomato***, and ***Get-Onion***, navigate the robot to the latest observed position of the vegetable, and pick the vegetable up if it is there; otherwise, the robot moves to check the initial position of the vegetable. The corresponding termination conditions are listed below:
  - The robot successfully picks up a chopped or unchopped vegetable;
  - The robot observes the target vegetable is held by another robot or itself;
  - The robot is holding something else in hand;
  - The robot's path to the vegetable is blocked by another robot;
  - The robot does not find the vegetable either at the latest observed location or the initial location;
  - The robot attempts to enter the same cell with another robot, but has a lower priority than another robot.
- ***Get-Plate-1/2***, navigates the robot to the latest observed position of the plate, and picks the vegetable up if it is there; otherwise, the robot moves to check the initial position of the vegetable. The corresponding termination conditions are listed below:
  - The robot successfully picks up a plate;
  - The robot observes the target plate is held by another agent or itself;
  - The robot is holding something else in hand;

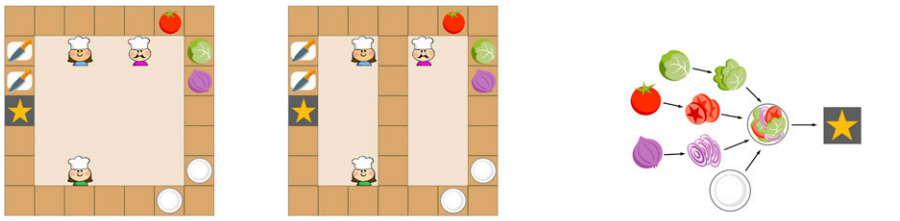


Figure 18. Experimental environments.

- The robot's path to the plate is blocked by another robot;
- The robot does not find the plate either at the latest observed location or at the initial location;
- The robot attempts to enter the same cell with another robot but has a lower priority than another robot.

- ***Go-Cut-Board-1/2***, navigates the robot to the corresponding cutting board with the following termination conditions:
  - The robot stops in front of the corresponding cutting board, and places an in-hand item on it if the cutting board is not occupied;
  - If any other robot is using the target cutting board, the robot stops next to the teammate;
  - The robot attempts to enter the same cell with another robot but has a lower priority than another robot.
- ***Go-Counter*** (only available in Overcook-B, Figure 6(c)), navigates the robot to the center cell in the middle of the map when the cell is not occupied, otherwise it moves to an adjacent cell. If the robot is holding an object the object will be placed. If an object is in the cell, the object will be picked up.
- ***Deliver***, navigates the robot to the "star" cell for delivering with several possible termination conditions:
  - The robot places the in-hand item on the cell if it is holding any item;
  - If any other robot is standing in front of the "star" cell, the robot stops next to the teammate;
  - The robot attempts to enter the same cell with another robot, but has a lower priority than another robot.

Observation Space (*O* and *Z*): The macro-observation space for each robot is the same as the primitive observation space. Robots are only allowed to observe the *positions* and *status* of the entities within a 5 × 5 view centered on the robot. The initial positions of all the items are known to robots.

Dynamics (*T*): The transition in this task is deterministic. If a robot delivers any wrong item, the item will be reset to its initial position. From the low-level perspective, to chop a vegetable into pieces on a cutting board, the robot needs to stand next to the cutting board and execute *left* three times. Only the chopped vegetable can be put on a plate.

Reward (*R*): +10 for chopping a vegetable, +200 terminal reward for delivering a tomato-lettuce-onion salad, −5 for delivering any wrong entity, and −0.1 for every time step.

Episode Termination: Each episode terminates either when robots successfully deliver a tomato-lettuce-onion salad or reach the maximal time steps, 200.

D.3. Warehouse tool delivery. In this Warehouse Tool Delivery domain, we consider four different scenarios shown in Figure 19 with many variants in terms of both the number of robots and the number of humans as well as having faster human (orange) in the environment.

Goal. Under all scenarios, in each workshop, a human is working on an assembly task involving four subtasks to be finished (each subtask takes amount of primitive time steps). At the beginning, each human has already got the tool for the first subtask and immediately starts. In order to continue, the human needs a particular tool for each following subtask. In the scenarios, humans either work in the same speed (Figures 19(a), 19(b), 19(d)) or have one of them working faster (the orange one in Figures 19(c) and 19(e)). A team of robots includes a robot arm (gray) with the duty of finding tools for each human on the table (brown) and passing them to mobile robots (green, blue and yellow) who are responsible for delivering tools to the humans. The objective of the robots is to assist the humans to finish their assembly tasks as soon as possible by finding and delivering the correct tools in the proper order. To make this problem more challenging, the correct tools needed by each human are unknown to robots, which has to be learned during training in order to perform timely delivery without letting humans wait.

State Space (*S*). The environment is either a 5 × 7 (Figures 19(a) and (e)) or a 5 × 9 (Figures 19(b)–(d)) continuous space. A global state consists of the 2D position of each mobile robot, the execution status of the arm robot's current macro-action (e.g., how many steps are left for completing the macro-action, but in real-world, this should be the angle and speed of each arm's joint), the subtask each human is working with a percentage indicating the progress of the subtask, and the position of each tool (either on the brown table or carried by a mobile robot). The initial state of every episode is deterministic as shown in Figure 19, where humans always start from the first step.

Macro-Action Space (*M*).

The available macro-actions for each mobile robot include:

- ***Go-W(i)*** navigates to the red waypoint at the corresponding workshop;

Figure 19. Experimental environments.

Table 1. The Number of Time Steps Taken by Each Human on Each Subtask in Scenarios.

Scenarios	Warehouse-A	Warehouse-B	Warehouse-C	Warehouse-D
Human-0	[27,20,20,20]	[40,40,40,40]	[38,38,38,38]	[40,40,40,40]
Human-1	[27,20,20,20]	[40,40,40,40]	[38,38,38,38]	[40,40,40,40]
Human-2	N/A	[40,40,40,40]	[27,27,27,27]	[40,40,40,40]
Human-3	N/A	N/A	N/A	[40,40,40,40]

• ***Go-TR*** navigates to the red waypoint (covered by the blue robot in Figures 19(c) and (d)) at the right side of the tool room;

• ***Get-Tool*** navigates to a pre-allocated waypoint besides the arm robot and waits over there until either 10 time steps have passed or received a tool from the gray robot.

The available macro-actions for the arm robot include:

• ***Search-Tool(i)*** takes 6 time steps to find tool *i* and place it in a staging area (containing at most two tools) when the area is not fully occupied, otherwise freezes the robot for the same amount of time;

• ***Pass-to-M(i)*** takes 4 time steps to pass the first found tool to a mobile robot from the staging area;

• ***Wait-M*** takes 1 time step to wait for mobile robots coming.

Macro-Observation Space (*Z*).

The arm robot's macro-observation includes the information about *the type* of each tool in the staging area and *which mobile robot* is waiting beside.

Each mobile robot always observes its own *position* and *the type* of each tool carried by itself, while observing *the number* of tools in the staging area or *the subtask* a human working on only when locating at the tool room or the workshop, respectively.

Dynamics (*T*). Transitions are deterministic. Each mobile robot moves in a fixed velocity 0.8 and is only allowed to receive tools from the arm robot rather than from humans. Note that each human is only allowed to possess the tool for the next subtask from a mobile robot when the robot locates at the corresponding workshop and carries the correct tool. Humans are not allowed to pass tools back to mobile robots. There are enough tools for humans on the table in the tool room, such that the number of each type of tool exactly matches the number of humans in the environment. Humans cannot start the next subtask without obtaining the correct tool. Humans' dynamics on their tasks are shown in Table 1.

Rewards (*R*). The team receives a +100 reward when a correct tool is delivered to a human in time while getting

Table 2. Number of Neurons on Each Layer in Networks for All Methods in Domains.

Domain	Box pushing		Overcooked		Warehouse	
Actor & Critic & Q-network	Dec	Cen	Dec	Cen	Dec	Cen
MLP-1	32	32	32	128	32	32
MLP-2	32	32	32	128	32	32
GRU	32	64	32	64	32	64
MLP-3	32	32	32	64	32	32

an extra −20 penalty for a delayed delivery such that the human has paused over there. A −10 reward occurs when the gray robot does ***Pass-to-M(i)*** but the mobile robot *i* is not next to it, and a −1 reward is issued every time step.

Episode Termination. Each episode terminates when all humans obtain all the correct tools for all subtasks, otherwise, the episode will run until the maximal time steps (200 for Warehouse-A and E, 250 for Warehouse-B and C, 300 for Warehouse-D).

E. Training details

Our results are generated by running on a cluster of computer nodes under "CentOS Linux" operating system. We use the CPUs including "Dual Intel Xeon E5-2650," "Dual Intel Xeon E5-2680 v2," "Dual Intel Xeon E5-2690 v3."

E.1. Network architecture. For all domains, all methods apply the same neural network architecture for both actor & critic network and Q-network. Each of them consists of two fully connected (FC) layers with Leaky-Relu activation function, one GRU layer (Cho et al., 2014) and one more FC layer followed by an output layer. The number of neurons in each layer for Decentralized (Dec) or Centralized (Cen) actor, critic and Q-network are shown in Table 2. Empirical experiments show that centralized actor and critic usually need more neurons to deal with larger joint macro-observation and macro-action spaces.