Electronic Research Archive

http://www.aimspress.com/journal/era

ERA, 32(2): 779–798.
DOI: 10.3934/era.2024037
Received: 12 November 2023
Revised: 13 December 2023
Accepted: 02 January 2024
Published: 11 January 2024

Research article

Learning cooperative strategies in StarCraft through role-based monotonic value function factorization

Kun Han[1], Feng Jiang[1,2,*], Haiqi Zhu[1], Mengxuan Shao[1] and Ruyu Yan[3]

[1] Faculty of Computing, Harbin Institute of Technology, Harbin 150000, China
[2] School of Medicine and Health, Harbin Institute of Technology, Harbin 150000, China
[3] School of Management, Harbin Institute of Technology, Harbin 150000, China

* **Correspondence:** Email: fjiang@hit.edu.cn.

Abstract: StarCraft is a popular real-time strategy game that has been widely used as a research platform for artificial intelligence. Micromanagement refers to the process of making each unit perform appropriate actions separately, depending on the current state in the the multi-agent system comprising all of the units, i.e., the fine-grained control of individual units for common benefit. Therefore, cooperation between different units is crucially important to improve the joint strategy. We have selected multi-agent deep reinforcement learning to tackle the problem of micromanagement. In this paper, we propose a method for learning cooperative strategies in StarCraft based on role-based montonic value function factorization (RoMIX). RoMIX learns roles based on the potential impact of each agent on the multi-agent task; it then represents the action value of a role in a mixed way based on monotonic value function factorization. The final value is calculated by accumulating the action value of all roles. The role-based learning improves the cooperation between agents on the team, allowing them to learn the joint strategy more quickly and efficiently. In addition, RoMIX can also reduce storage resources to a certain extent. Experiments show that RoMIX can not only solve easy tasks, but it can also learn better cooperation strategies for more complex and difficult tasks.

Keywords: Q-learning; multi-agent reinforcement learning; machine learning; artificial intelligence; StarCraft multi-agent challenge

1. Introduction

Starcraft is a real-time strategy game released by Blizzard Entertainment. Compared to traditional games, StarCraft has a large observable space dimension and complex strategy combinations. It not only attracts a group of professional esports players, it is also regarded as a form of highly valuable human-machine competition in the current field of artificial intelligence [1]. StarCraft has two main

gameplay components: macromanagement and micromanagement [2]. Macromanagement requires a comprehensive consideration of resources, economy, and military capabilities by determining what combat units to build or produce to win the entire competition. Micromanagement is another important subproblem in StarCraft, and it mainly considers the fine-grained control of individual units. Additionally, it focuses on decentralized micromanagement challenges, where each unit makes corresponding action decisions based on its own partial observation states [3].

In the micromanagement, the task goal is to enhance the combat capabilities of each unit in a team by seeking ways to join other allies and realize the ultimate team victory [4]. The main difficulty of micromanagement is determining how to coordinate all units in a team to learn the best joint strategy based on their current local observation state and determine the welfare based on the best joint strategy. Therefore, the micromanagement of StarCraft can be considered as a fully cooperative multi-agent learning problem.

In a fully cooperative problem, all agents collaborate with each other to achieve a specific shared goal by jointly learning the action strategy. Cooperation is important and widespread, as it is seen as an important foundation for human evolution [5]. The problem of multi-agent cooperation is also widely present in practical applications, such as unmanned aerial vehicles [6], wireless communications [7], load frequency control [8], urban traffic light control [9], and so on.

Multi-agent reinforcement learning (MARL) has achieved great success as a tool to solve cooperative decision-making problems with multiple agents [10, 11]. It is worth considering that, for the micromanagement task of StarCraft, the cooperative strategy is difficult to learn due to the problems caused by the partial observability, communication constraints, and non-stationary learning process of the agents. Therefore, the MARL method based on the framework of centralized training with decentralized execution (CTDE) is widely used [12–15]. CTDE combines the advantages of centralized learning [16] and independent learning [17] paradigms, allowing agents to fully utilize global information during learning and only use local information during decision-making. CTDE frameworks include two training methods: on-policy and off-policy learning. In contrast to on-policy learning, off-policy learning can store the generated experience data in the replay buffer and be utilized multiple times, making it sample-efficient [18]. However, on-policy learning allows the generated experience data to be used only once during training, making it sample-inefficient. Actor–critic based methods such as counterfactual multi-agent (COMA) policy gradients [12] and multi-agent proximal policy optimization [13] require on-policy learning, which can be sample-inefficient. In addition, as the number of agents increases, it is impractical to train the fully centralized critic. The value function factorization [14] methods that combine deep Q-networks (DQNs) [18] and actor–critic methodology require off-policy learning, thus avoiding the previously mentioned problems. Furthermore, QMIX [15] violates the linear constraint and utilizes additional state information to further enhance learning. Although the QMIX algorithm can already solve most cooperative multi-agent problems, it also has a limitation: all agents learn equally during the training process, which makes QMIX inefficient on some complex tasks. Particularly, when the task is hard, it is difficult to learn feasible and effective joint strategies.

In current research, there is an increasing number of large-scale data analysis or complex multi-agent decision-making tasks, such as gesture recognition based on multiple sensor signals [19], wearable systems based on multimodal data [20], and large-scale collaborative control [21]. For these tasks, role-based strategies can divide the agents based on their different attributes, and then learn

local decisions or subtask strategies in each role set. Unlike integrated multi-agent tasks, it potentially decomposes tasks and reduces difficulty. Role-based learning is an effective means to solve large-scale or difficult multi-agent tasks.

Therefore, to solve complex and difficult domain-specific problems, especially hard or superhard maps in the micromanagement of StarCraft, we propose a novel method to learn cooperative strategies based on role-based monotonic value function factorization (RoMIX). RoMIX decomposes tasks through the implementation of role mechanisms to reduce learning difficulty, with the aim of efficiently learning complex problems. More specifically, RoMIX is a two-stage learning method. In the first stage, a set of roles is created based on the different concerns of the agents regarding the task; this stage serves to indirectly decompose the task based on the role mechanism. In the second stage, the action values of all agents in each role subset are monotonically mixed. The monotonic mixing method restricts the non-negative correlation between each individual value and the total mixed value. And, the value obtained via this monotonic mixing method represents the role value of the corresponding subtask. Finally, the value of the task is represented by role values.

In addition, we believe that RoMIX has satisfactory decision-making ability for difficult tasks, and it is meaningful to integrate it into specific multi-agent learning fields such as multi-robot collaboration, multi-sensor systems, drone swarm combat, etc. RoMIX can improve decision-making efficiency and ability for hard tasks, as well as facilitate further learning of effective and complex cooperative strategies. However, the challenges of integrating RoMIX into existing multi-agent learning cannot be ignored, such as adaptability and security issues. RoMIX has been designed for collaborative, complex task-type problems, and the natural gap between it and practical multi-agent learning systems is security. Therefore, integrating RoMIX into multi-agent learning systems is both an opportunity and a challenge.

We evaluate RoMIX on StarCraft decentralized micromanagement tasks and use the StarCraft Multi-Agent Challenge (SMAC) [2] environment as the testbed, which has become a common-used benchmark for evaluating state-of-the-art MARL approaches. We selected nine maps from three different levels of difficulty for experiments, and we verified that the RoMIX can learn effective joint strategies on fully cooperative multi-agent tasks, especially for complex and difficult task scenarios. Moreover, it can also reduce storage resources to a certain extent.

2. Related work

For cooperative multi-agent tasks, most classic algorithms are based on Q-learning methods, which extend Q-learning to multi-agent environments. Wang and De Silva [22] proposed the team Q-learning algorithm for multi-agent cooperative tasks and theoretically proved its convergence to the optimal strategy. Galindo-Serrano and Giupponi [23] proposed a real-time MARL distributed Q-learning method to manage the interference generated by agents by directly interacting with the surrounding environment through distributed means. In addition, Wang and Sandholm [24] proposed the optimal adaptive learning (OAL) based on biased action selection and incomplete historical sampling, and they demonstrated that this method converges to the optimal Nash equilibrium in team games with multiple Nash equilibria. Furthermore, Arslan and Yüksel [25] proposed a decentralized Q-learning algorithm for stochastic games based on OAL, supplementing the proof that the algorithm converges stably to the optimal Nash equilibrium in a large number of stochastic games.

StarCraft is a real-time game that is renowned for its complexity, intricate strategy combinations, and vast observation space. In the StarCraft environment, agents interact with the game by observing states, receiving events, and executing actions. There are many factors to consider for this complex game task, such as strategy diversity, team composition and quantity, macro and micro-management, communication and collaboration between agents, large-scale spatial processing, robustness, adaptability, and training efficiency. Therefore, there are many studies on StarCraft environmental tasks.

Value decomposition is a research focus for MARL, and it is widely applied for the baseline tasks of Starcraft. Its essence is an extension of Q-learning. Value-decomposition networks (VDNs) [14] constitute the first attempt of value decomposition in Starcraft. It assumes that the value function can be decomposed, requiring the joint action value to be the linear sum of the action values of each individual agent. But, the premise of the linear assumption also causes VDNs to have many shortcomings. One is that the constraint on the value function is too hard, which harmfully limits the complex representation of action values between the total and decentralized individuals, and it is only applicable to some small-scale scenarios. Another shortcoming is that during training, any additional information available is completely ignored and the best strategy can only be learned in some simple environments. To improve the representation of value functions and fully utilize the global information during training, Rashid et al. [15] proposed QMIX, which trains decentralized strategies in an end-to-end way. Due to the inability of VDNs to represent complex association between individuals and the total, QMIX uses neural networks with strong approximation capabilities for representation. In addition, QMIX utilizes additional global state information when approximating the total action value, taking fully advantage of centralized training.

The QTRAN [26] algorithm further combines the advantages of VDNs and QMIX. QTRAN requires the assumption that directly using neural networks to approximate joint functions is relatively difficult, so it entails a two-step approximation: first, the VDN method is used to obtain the added joint value function as an approximation of the joint value function; then, it fits the difference between the added joint value function and the joint value function. Mahajan et al. [27] proposed a new hierarchical learning algorithm, multi-agent variational exploration, based on the QMIX framework to improve the efficiency of multi-agent exploration. Yang et al. [28] proposed a value function mixture based on the multi-head attention mechanism to approximate joint value functions without introducing additional assumptions and constraints. Furthermore, Khan et al. [29] proposed TransMix by combining Fastformer [30] attention to effectively decompose value functions. In addition, weighted QMIX (WQMIX) [31] enhances the importance of the optimal joint action by weighting the loss function of QMIX with centrally-weighted and optimistically-weighted QMIX, thereby promoting the convergence of the optimal joint strategy. Duplex dueling multi-agent Q-learning (QPLEX) [32] guarantees compliance to the principle of monotonicity by introducing an advanced based function that is realized with a duplex dueling architecture [33].

There are also some algorithms based on the actor–critic framework that have been proposed. Foerster et al. [12] proposed COMA policy gradients based on the counterfactual baseline by using a completely centralized learning approach. Iqbal and Sha [34] proposed the multi–actor–attention–critic methodology based on the CTDE framework, combined with an attention mechanism and the counterfactual baseline of COMA, to further improve the reliability allocation problem. Zhang et al. [35] proposed the CollaQ for the dynamic allocation of rewards through the

application of Taylor expansion, with the aim of allowing agents to automatically adapt to changes in the environment and other agents without retraining. In addition, task decomposition is also an effective method. Liu et al. [36] modeled the relationships between agents based on graph neural networks and proposed a two-stage attention network (G2ANet) to decompose the graph into a set of sub-graphs for learning, thereby accelerating strategy learning. And, the RODE [37] algorithm decomposes complex multi-agent tasks into simpler subtasks by learning different roles to improve learning efficiency. Although neither Zhao and Lv [38] nor Wang et al. [39] are studying the StarCraft environment, their respective studies on the robustness of complex multi-agent systems and heterogeneous multi-agent systems provide great inspiration.

3. Proposed method

In this section, we introduce our cooperative strategy learning methodology based on role-based monotonic value function factorization, i.e., ROMIX. We consider that fully cooperative MARL tasks in the SMAC are typically modeled as a decentralized partially observable Markov decision process [40], described by the tuple $G =< N, S, U, P, r, \Omega, O, n, \gamma >$. Here, N is the finite set of n agents, and S describes the true or global state in the environment. For every time step, each agent $i \in N \equiv \{1, ..., n\}$ selects an action $u_i \in U$, giving rise to a joint action vector $\mathbf{u} \in \mathbf{U} \in U^n$. When the joint action $\mathbf{u}$ is executed, it leads to a transition to the next state s' according to the transition function $P(s'|s, \mathbf{u}) : S \times \mathbf{U} \times S \to [0, 1]$. Each agent shares the same joint reward function $r(s, \mathbf{u}) : S \times \mathbf{U} \to \mathbb{R}$ with the discount factor $\gamma \in [0, 1)$. For a partially observable process, each agent i receives an individual local observation $o_i \in \Omega$ based on the observation function $O(o_i|s, u_i)$. The action observation history of agent i is denoted by $\tau_i \in T \equiv (\Omega \times U)^*$, on which it conditions its decentralized stochastic policy $\pi_i(u_i|\tau_i) : T \times U \to [0, 1]$.

RoMIX learns cooperative strategies by decomposing multi-agent cooperative tasks into a set of subtasks. Different subtasks are matched with different roles, and agents in the same role share their learning to jointly learn the role strategy of the subtask. RoMIX learns cooperative strategies by implementing an end-to-end model in two stages, as shown in Figure 1.

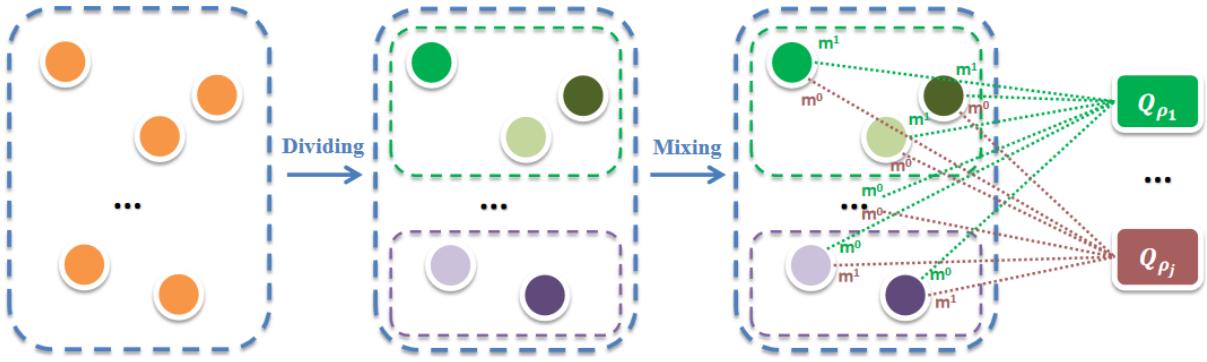


Figure 1. Role-based monotonic value function factorization.

In the first stage, for a cooperative multi-agent task $G =< N, S, U, P, r, \Omega, O, n, \gamma >$, a set of roles Φ is created based on the different concerns of the agents regarding the task G. Each role $\rho_j \in \Phi$ is

associated with the corresponding subtask g_j, where the subtask $g_j :=< N_j, S, U, P, r, \Omega_j, O, \gamma >\in G$, with $N_j \subset N$, $\cup_j N_j = N$, and $N_j \cap N_k = \varnothing$, $j \neq k$. After completing the first stage, each agent only belongs to a specific role set.

In the second stage, the expected global return Q^{Φ} of task G is maximized by learning an optimal set of $\Phi^{\star}$:

$$Q^{\Phi}(s_t, \mathbf{u}_t) = \mathbb{E}_{s_{t+1:\infty}, \mathbf{u}_{t+1:\infty}}\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} | s_t, \mathbf{u}_t, \Phi\right] \tag{3.1}$$

where Q^{Φ} is composed of Q^{ρ_j}, and Q^{ρ_j} is a mixed representation of the individual action values of each agent in ρ_j based on monotonic value function factorization. The following sections will provide a detailed introduction to the two stages.

3.1. Action representation learning and role set creation

Learning how to come up with a set of roles to effectively decompose a task is an important stage. Instead of traditional methods based on prior knowledge, we were inspired by RODE [37] to decompose a task based on action representation. Specifically, we consider that each agent's focus on a task is associated with its own action attributes, which allows each role to focus on a set of actions. Learning roles in this way has better generalization, as the decomposition of a task is not limited by changes in the number of agents. For example, for homogeneous and symmetrical SMAC task scenarios, the set of learned roles still has strong usability when the numbers of enemies and allies changes equally. Therefore, it is a practicable and effective way to decompose a task based on the actions of agents in cooperative multi-agent problems.

Both supervised learning and unsupervised learning are used in the role-learning stage. Supervised learning encodes action representations, while unsupervised learning classifies action representation encoding, as shown in Figure 2.

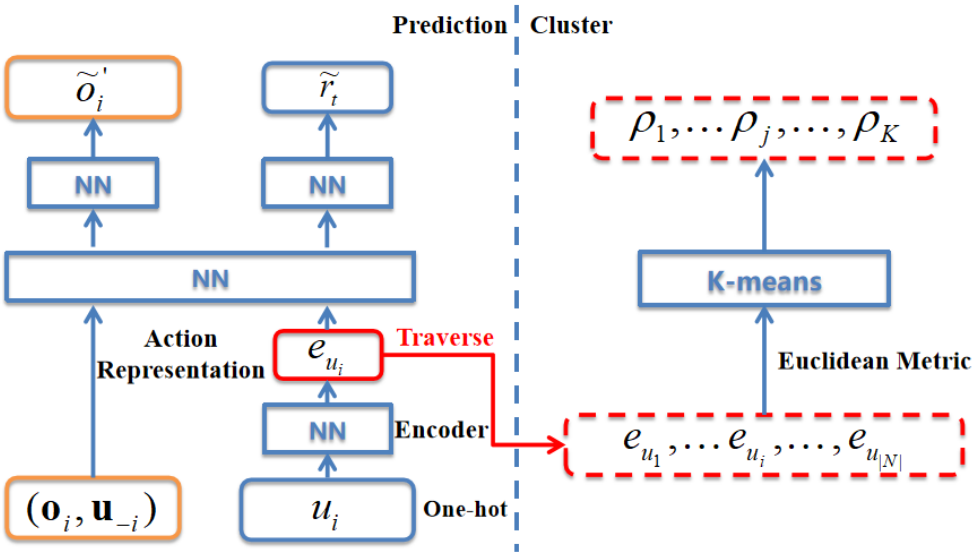


Figure 2. The architecture for action representation learning and character set creation.

For the supervised learning shown in the left part of Figure 2, a feed forward neural network is used for prediction, with the aim of encoding the actions of agent i. Specifically, the local observation o_i and action u_i of agent i are used as inputs to predict the local observation $\widetilde{o'}$ of agent i at the (next) $t+1$ time step as well as the global reward $\widetilde{r_t}$ at the (current) t time step. In addition, to avoid non-stationary effects caused by the actions of other agents on the prediction, it is necessary to input the action vector $\mathbf{u}_{-i}$ of other agents. For agent i, the action-encoder module maps a one-hot action u_i with dimension $\mathbb{R}^{|U|}$ to a representation encoding e_{u_i} of the action with dimension $\mathbb{R}^d$, where d is the network output dimension of the action-encoder module.

The loss function of the supervised learning prediction model is given by

$$\mathcal{L}(\theta_e, \xi_p) = \mathbb{E}_{(\mathbf{o},\mathbf{u},r,\mathbf{o}')\sim\mathcal{D}}\left[\sum_i \|p_o(e_{u_i}, o_i, \mathbf{u}_{-i}) - o_i'\|_2^2 + \lambda_p \sum_i (p_r(e_{u_i}, o_i, \mathbf{u}_{-i}) - r)^2\right] \tag{3.2}$$

where p_o and p_r denote the observation predictor and reward predictor of agent i, respectively, and both are parameterized by ξ_p. λ_p is a hyperparameter that is used to adjust the scale of the predictors, and $\mathcal{D}$ is the replay buffer.

For the unsupervised learning shown in the right part of Figure 2, we use k-means clustering to cluster a set of roles Φ based on Euclidean distance for actions with dimension $\mathbb{R}^{|d|}$, where the total number of roles is K, i.e., $\rho_j \in \Phi$, $j = 1, ..., K$. Finally, each agent i is assigned a unique and deterministic role representation ρ_j based on the action representation. In practice, K is a hyperparameter that is applied to set micromanagement tasks in StarCraft, representing the dimensional size of the role set.

3.2. Mixing action values by role-based monotonic value function factorization

In a multi-agent task, we perform a mixed representation of individual Q_i values and the overall Q_{tot} value at two levels: intra-role set value mixing and inter-role set value mixing. The complete architecture of RoMIX is shown in Figure 3, where (a) is the role-based mixing architecture, (b) is the overall architecture of RoMIX, and (c) is the network structure of a single agent. Inspired by QMIX [15], we propose RoMIX, which implicitly decomposes a task by combining role mechanisms, with the aim of representing Q values more effectively. And, then, we will provide a detailed illustration of our RoMIX.

We use the deep recurrent Q-networks (DRQN) [41] based on gate recurrent unit (GRU) to represent each agent i, which evaluates individual $Q_i(\tau^i, u^i)$ by using the observation $o_t^i \in \Omega$ at time step t and the action $u_{t-1}^i \in U$ at time $t-1$ as inputs, as shown in Figure 3(c). In practice, the first layer in the agent network is a fully connected layer with 64 neurons. After learning in the GRU network, the output dimension of the second fully connected network layer is $\mathbb{R}^{|U|}$. It is worth noting that agent i selects actions by following the ε-greedy policy in the next time step, and the minimum ε is 0.02.

As shown in the Element-wise role matching module in Figure 3(a), the action values of each agent are divided into corresponding role sets by this module. Specifically, we design action-value masks based on the action sets of each role set. For the action-value vector $Q_1(\tau^1, u_t^1), ..., Q_N(\tau^N, u_t^N)$ of all agents, we filter it element-wise by multiplying the action-value vector with the masks corresponding to each role, retaining the action values of the agents required in the action-value vector.

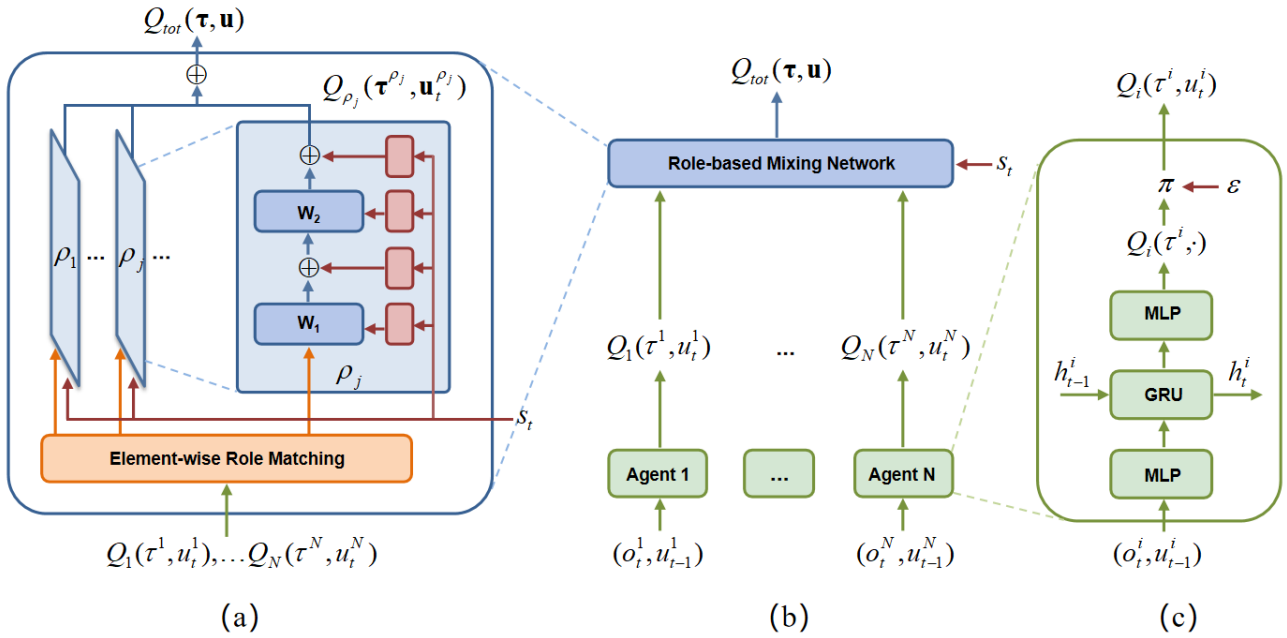


Figure 3. Complete architecture of RoMIX.

For all agents in the role set ρ_j, we mix their Q_i by applying monotonic value function factorization to represent Q_{ρ_j}. Specifically, Q_{ρ_j} represents the total utility Q_i of all agents in ρ_j. To ensure consistency and monotonicity between Q_{ρ_j} and Q_i of each agent, the individual global max (IGM) principle has been incorporated; it restrictively ensures that the global argmax operation performed on Q_{ρ_j} produces the same result as the argmax operation performed on each Q_i:

$$\arg\max_{\mathbf{u}^{\rho_j}} Q_{\rho_j}(\tau^{\rho_j},\mathbf{u}^{\rho_j}) = \begin{pmatrix} \arg\max_{u^1} Q_1(\tau^1,u^1) \\ \vdots \\ \arg\max_{u^{|\rho_j|}} Q_{|\rho_j|}(\tau^{|\rho_j|},u^{|\rho_j|}) \end{pmatrix}, \forall j \in \{1,2,...,K\} \tag{3.3}$$

where $|\rho_j|$ is a scalar representing the total number of all agents in the role set ρ_j; K is also a scalar representing the total number of role sets. The vectors τ^{ρ_j} and $\mathbf{u}^{\rho_j}$ represent the joint action observation history and joint action of all agents in the role set ρ_j, respectively.

The IGM allows each agent i to perform decentralized execution only by selecting a greedy action related to its own Q_i. Furthermore, the following intuitive and sufficient but unnecessary constraint can further generalize Eq (3.3) to more general Q_{ρ_j} and each Q_i monotonicity:

$$\frac{\partial Q_{\rho_j}}{\partial Q_i} \geq 0, \forall i \in \{1,2,...,|\rho_j|\} \tag{3.4}$$

Therefore, in the role set ρ_j, we learn the non-negative correlation between the joint action value Q_{ρ_j} and the individual action value Q_i by combining the role-based mixing structure ρ_j-mixing, and incorporate global state information during the training process, as shown in Figure 3(a). Unlike linear addition, ρ_j-mixing uses a neural network to integrate the action values of each individual. By following the monotonicity constraint in Eq (3.4), the joint Q_{ρ_j} is maximized when the individual Q_i is maximized, that is, the optimal joint action selection and optimal joint strategy are equivalent.

In practice, we perform a mixed representation of individual Q_i and the final total joint action value Q_{tot} at two levels: intra-role set Q_i mixing and inter role set Q_{ρ_j} mixing. For the intra-role set mixing, we use extra hypernetworks to learn the non-negative weights and biases of a ρ_j-mixing network based on the global state S to represent the Q_{ρ_j} in a mixed way and consequently follow the monotonicity of the Q_{ρ_j} and the Q_i of each agent i. Specifically, each ρ_j-mixing network is a fully connected network containing a hidden layer with a number of neurons H (such as $H = 32$), whose input and output are a joint action-value vector $\mathbf{Q} \in \mathbb{R}^{|N|}$ and a scalar Q_{ρ_j}, respectively. Therefore, the ρ_j-mixing network contains two sets of weights and biases: $w_1 \in \mathbb{R}^{N\times H}$ and b_1, and $w_2 \in \mathbb{R}^{|H|}$ and b_2. We use hypernetworks with the global state S as inputs to produce these two sets of weights and biases, which are followed by an absolute activation function to ensure consistency in monotonicity. K ρ_j-mixing networks are used to mix the action values of agents in K role sets. We use the role-selector layer to solve the problem of variable ρ_j-mixing input. Specifically, in the role-selector layer, the role-matching mask vector is element-wise multiplied with the joint action value vector, where the length of the role-matching mask vector is $\mathbb{R}^{|N|}$. Subsequently, for the inter-role set mixing, we refer to the linear sum method in the VDN [14] to calculate the total value. We also accumulate each Q_{ρ_j} obtained from ρ_j-mixing to calculate the final total joint action value Q_{tot}:

$$Q_{tot}(\tau, \mathbf{u}) = \sum_{j=1}^{K} Q_{\rho_j}(\tau^{\rho_j}, \mathbf{u}^{\rho_j}; \xi_{\rho_j}) \tag{3.5}$$

where ξ_{ρ_j} represents the ρ_j-mixing network parameters corresponding to ρ_j.

In addition, during the training process, RoMIX updates by referring to the standard DQN [18] loss to minimize TD loss, as follows:

$$\mathcal{L}(\theta, \xi) = \sum_{i=1}^{b} (y_i^{tot} - Q_{tot}(\tau, \mathbf{u}, s; \theta, \xi)) \tag{3.6}$$

where θ denotes the parameters of all agent networks, ξ denotes the parameters of all ρ-mixing networks, b is the batch size of transitions sampled from the replay buffer $\mathcal{D}$, $y^{tot} = r + \gamma \max_{\mathbf{u}'} Q_{tot}(\tau', \mathbf{u}', s'; \theta^-, \xi^-)$, and both θ^- and ξ^- are the parameters of the target network, as in the DQN.

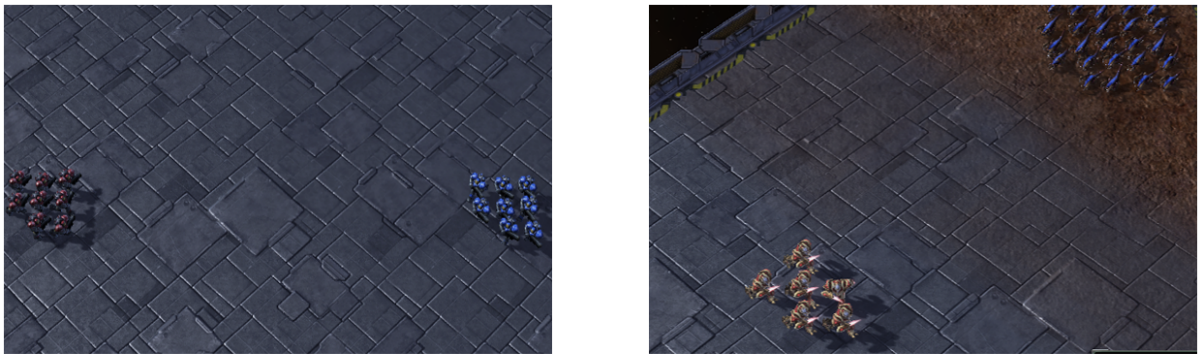
4. Experiments

We chose to evaluate RoMIX on the StarCraft II decentralized micromanagement tasks. And, in this experimental section, we clarify the settings and present sufficient validation and detailed analysis for task scenarios at three difficulty levels.

4.1. Experimental settings

We chose to use the SMAC environment as the testbed, as it has become a common-used benchmark for evaluating cooperative MARL methods. In SMAC tasks, allied units learn decentralized cooperative strategies through algorithms. The strategies of enemy units are manually coded by human experts, with a total of 10 difficulties. In our experiment, we chose the default difficulty of 7 (sc_pb.VeryHard). For example, Figure 4 shows two scenarios: (a) eight Marine allies

are fighting against eight Marine enemies. The name of this map is 8m, which is a homogeneous and symmetric easy scenario. (b) It is a heterogeneous and asymmetric superhard scenario where six Zealot allies engage 24 Zergling enemies.



(a) 8 Marines' map (b) Corridors' map

Figure 4. Decentralized unit micromanagement in StarCraft.

Table 1. Nine maps with three difficulty level scenarios.

Map	Ally unit	Enemy unit	Type	Difficulty level
8m	8 Marines	8 Marines	homogeneous symmetric	easy
2s3z	2 Stalkers, 3 Zealots	2 Stalkers, 3 Zealots	heterogeneous symmetric	easy
2s_vs_1sc	2 Stalkers	1 Spine Crawler	heterogeneous asymmetric	easy
5m_vs_6m	5 Marines	6 Marines	homogeneous asymmetric	hard
3s_vs_5z	3 Stalkers	5 Zealots	heterogeneous asymmetric	hard
2c_vs_64zg	2 Colossi	64 Zerglings	heterogeneous asymmetric	hard
3s5z_vs_3s6z	3 Stalkers, 5 Zealots	3 Stalkers, 6 Zealots	heterogeneous asymmetric	superhard
6h_vs_8z	6 Hydralisks	8 Zealots	heterogeneous asymmetric	superhard
corridor	6 Zealots	24 Zerglings	heterogeneous asymmetric	superhard

To fully validate our method, we selected nine maps for the experiment, with three difficulty level scenarios. The specific settings and detailed descriptions of the maps are shown in Table 1. And, from the table, it can be seen that homogeneity, symmetry, and the scale of the agents collectively affect the difficulty of the task.

The proposed method was trained by using a PC configured with i7-10700, GeForce GTX 1080Ti 11GB, 32GB RAM, 250GB SSD, and 2TB HDD. Our training time ranged from about 8 hours to 72 hours on these maps, as based on the number of agents, map features, and total training setps of each map. For some easy scenarios, the number of total training steps was set as 1.5 million or 2 million in our experiment. However, for some difficult scenarios, the number of total steps was set as 4 million or 5 million, as shorter steps cannot be fully trained. And, we evaluated the model every 5×10^3 steps. Generally, the size of the replay buffer $\mathcal{D}$ was applied as 5×10^3. Due to the hardware conditions, for some large-scale scenarios, the size in all our experiments and comparative experiments was set to 3×10^3 or 2.5×10^3. In addition, for the other hyperparameters, the settings were as follows: batch size = 8, learning rate of 5×10^{-4} with the RMSprop optimization, the number of role sets $K = 2$, and

$\lambda_p = 10$. We applied the ϵ-greedy policy for exploration. The starting exploration rate was set to 1, and the end exploration rate was 0.02.

4.2. Validation

4.2.1. Easy scenarios

For easy task scenarios, we chose three scenarios: 8m, 2s_vs_1sc, and 2s3z. Because the total number of multi-agent tasks was not large and the task difficulty was relatively easy, we set the training steps to 1.5 million. In addition, we chose VDN, QTRAN, G2ANet, and QMIX as the comparative methods for all experiments. The above comparative experiments are all based on the CTDE learning paradigm. We evaluated the mean test win rate across 32 runs per 5000 steps for each method on selected maps, and we independently ran each method three times to obtain the experimental results. The experimental results are shown in Figure 5.

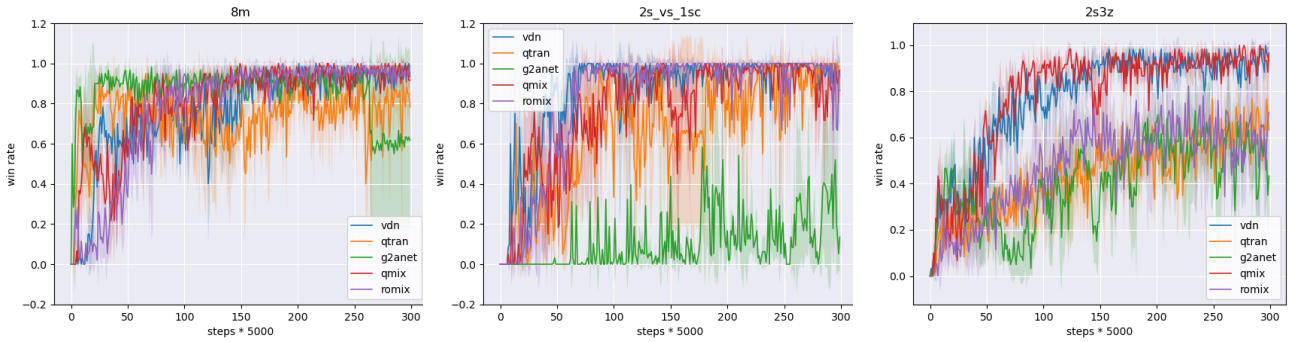


Figure 5. Win rates for easy map scenarios.

For a clearer comparison, we shall provide the corresponding statistical test results. Specifically, in each task scenario, we applied the median winning rate of the last 50×5000 steps as the evaluation metric. The winning percentages for easy scenarios are shown in Table 2. The 8m scenario was a homogeneous and symmetrical task, so the learning speeds were similar for all algorithms. The winning rate for the G2ANet algorithm dropped sharply to 61.67% in the end, while the VDN, QMIX, and RoMIX algorithms all achieved high winning rates of no less than 95%. However, it is worth noting that the QTRAN method did not perform well enough, which also proves that the QTRAN method is excellent in theory but less effective in practice. For the asymmetric and heterogeneous small-scale task of the 2s_vs_1sc scenario, with the exception of G2ANet, the accuracy and learning speeds of all value function decomposition algorithms were similar.

As the number and types of agents increase, tasks become increasingly challenging. In the case of the 2s3z scenario, both VDN and QMIX learned faster and achieved high winning rates of 93.33 and 95.00%, respectively, because simpler methods are faster and more effective for easy and small-scale task scenarios. It is worth mentioning that, as the training step size increased, our RoMIX method was able to ultimately achieve the same winning rate of 93.75%, as shown in Figure 6. This is because the RoMIX network has a larger scale; furthermore, although it may be slower to learn parameters for easy tasks, the upper performance limit of the model is not low.

Table 2. Win percentages for easy scenarios.

Scenario	VDN	QTRAN	G2ANet	QMIX	RoMIX
8m	95.00	80.00	61.67	96.67	95.83
2c_vs_1sc	98.33	95.00	17.71	94.17	98.96
2s3z	93.33	63.33	53.33	95.00	59.90

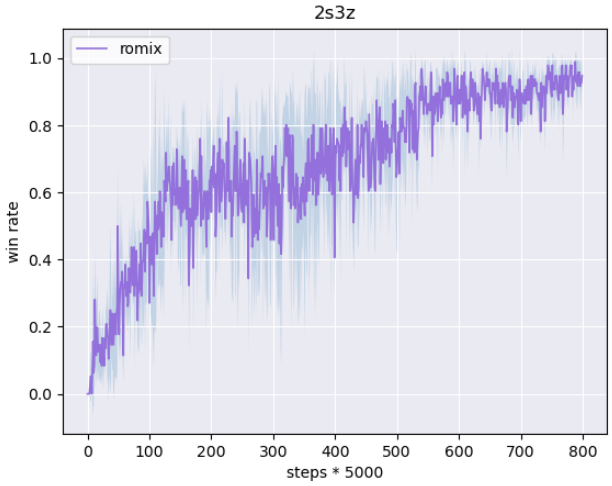


Figure 6. The win rate for 2s3z scenarios with 4 million training steps.

4.2.2. Hard scenarios

For the learning of cooperative strategies on hard tasks, we chose three scenarios: 3s_vs_5z, 2c_vs_64zg, and 5m_vs_6m. The experimental results are shown in Figure 7 and Table 3. By combining the table and the figure for analysis, we can clearly see that, for the three difficult task scenarios, due to the G2ANet method based on the actor–critic framework having the lowest-valued experimental results and almost all winning rates equaling zero, this section of the discussion only focuses on the four methods based on value function decomposition. First, in the case of the 3s_vs_5z map, RoMIX consistently outperformed other methods and achieved the highest winning rate of 86.88%. And, as the winning rate increased, the variance also rapidly converged, and the experimental results tended to be stable. According to the results, the difficulty of the 3s_vs_5z map is lower than that of the two other maps. Subsequently, we will analyze the other two task scenarios separately.

Table 3. Win percentages for hard scenarios.

Scenario	VDN	QTRAN	G2ANet	QMIX	RoMIX
3s_vs_5z	53.13	80.00	0.00	72.92	86.88
2c_vs_64zg	0.00	0.00	2.10	0.00	11.46
5m_vs_6m	26.04	54.17	0.00	34.38	46.88

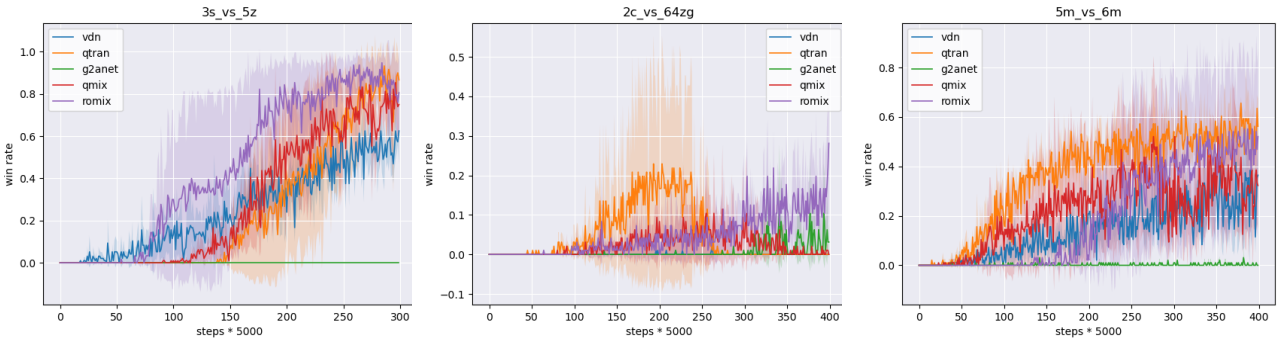


Figure 7. Win rates for hard map scenarios.

For the 2c_vs_64zg map, we set the replay buffer $\mathcal{D}$ size to 2.5×10^3. The two main problems of the 2c_vs_64zg map are extreme asymmetry and the large number of agents. Due to these two problems, the agents cannot learn effective cooperative strategies. According to Table 3, with the exception of RoMIX, the winning rates of all methods were almost zero. For the sake of comparison, we can further analyze based on the reward of the task, as shown in Figure 8. And, the reward results for VDN, QTRAN, G2ANet, QMIX, and RoMIX were 13.79, 10.58, 13.27, 11.13, and 17.34, respectively. It is obvious that RoMIX learns better cooperative strategies than other methods, although it was difficult for all methods to achieve an ultimate win.

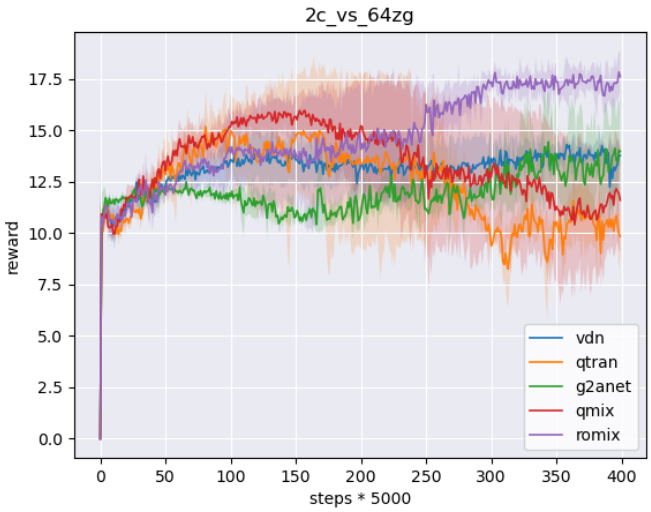


Figure 8. The reward of training on the 2c_vs_64zg scenario.

Finally, for the 5m_vs_6m scenario, when the training step size was 2 million, the results indicated that QTRAN performs better than RoMIX. The winning rate in terms of training steps was higher than that for VDN and QMIX, and the winning rates for VDN and QMIX tended to flatten or decrease. Then, only considering QTRAN and RoMIX, when we extended the training steps to 4 million, the winning rates for QTRAN and RoMIX were 56.25 and 68.75%, respectively. The results are shown in

Figure 9. Therefore, we can clearly see that our RoMIX performs better than QTRAN and can learn better joint strategies.

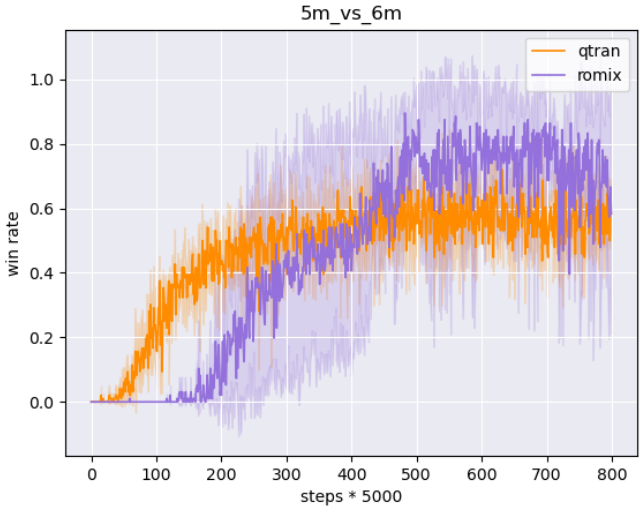


Figure 9. The win rate for the 5m_vs_6m scenario with 4 million training steps.

4.2.3. Superhard scenarios

For superhard task scenarios, in the cases of the maps 6h_vs_8z and 3s5z_vs_3s6z, we set the size of the replay buffer $\mathcal{D}$ to 3×10^3. For the Corridors map, we set the size of $\mathcal{D}$ to 2.5×10^3. The above settings and operations are due to the large number of agents in these three maps, as it was necessary to consider the limitations of computing resources. To control the influence of variables, we applied the same settings and operations for the other comparison methods; we obtained the experimental results as shown in Figure 10 and Table 4.

By combining the table and the figure for analysis, we can clearly see that the winning rates for the 6h_vs_8z and Corridor maps were significantly higher, which indicates that RoMIX has a significant effect on the process of solving complex and difficult problems. In addition, RoMIX has fewer requirements for the size of the replay buffer than other methods, which also indicates that RoMIX can save computational resources. QMIX only learned effective strategies for the 6h_vs_8z map and achieved a winning rate of 48.96%, while QTRAN only learned effective strategies for the Corridor map and achieved a winning rate of 22.40%. However, RoMIX learned effective strategies for both maps and achieved winning rates of 55.21 and 68.75%, respectively, demonstrating stronger generalization ability. It was difficult for the simple VDN method to learn effective cooperative strategies on superhard tasks.

For the 3s5z_vs_3s6z scenario, we also analyzed the task rewards because all methods have low winning rates, as shown in Figure 11. The reward results for VDN, QTRAN, G2ANet, QMIX, and RoMIX were 10.00, 13.61, 11.11, 13.56, and 15.82. By comparing the reward results, we can know that, for complex problems, the performance of the simple VDN is similar to that of G2ANet, while the performance levels of QMIX and QTRAN are similar; moreover, the joint strategies learned by RoMIX are more effective.

Table 4. Win percentages for superhard scenarios.

Scenario	VDN	QTRAN	G2ANet	QMIX	RoMIX
6h_vs_8z	1.04	0.00	0.00	48.96	55.21
Corridor	0.00	22.40	0.00	0.00	68.75
3s5z_vs_3s6z	0.00	0.00	0.00	0.00	2.08

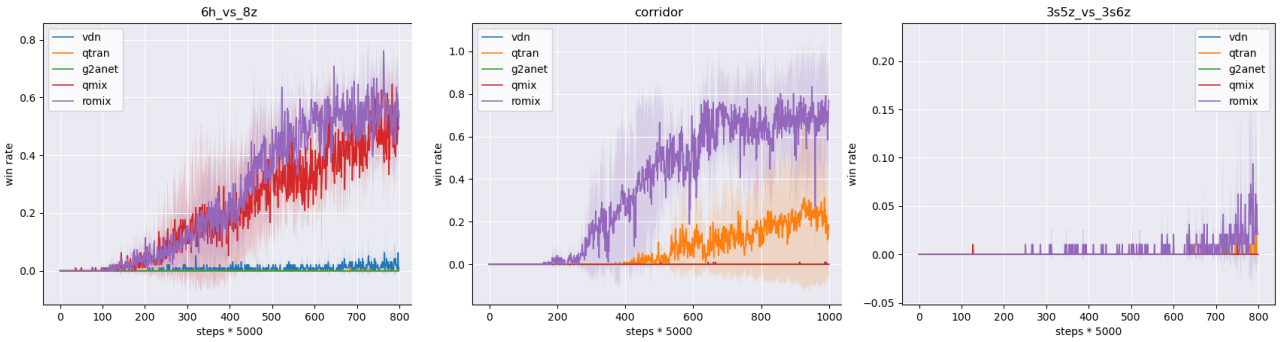


Figure 10. Win rates for superhard map scenarios.

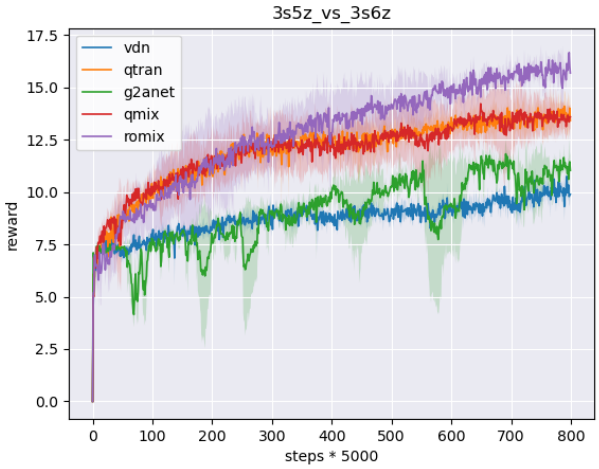


Figure 11. The reward of training on the 3s5z_vs_3s6z scenario.

4.2.4. Ablation study

To investigate whether different role sets will affect strategy learning, we selected one map from each of the three different difficulty-level scenarios: 8m, 5m_vs_6m, and 6h_vs_8z. The experimental settings for each map were the same as before. It is worth noting that the settings of RoMIX-r entailed randomly selecting role sets, which caused it to be different from RoMIX but have the same number of sets as RoMIX. In addition, we also included QMIX as a comparative method because QMIX can be

seen as a special case in RoMIX when the number of sets is 1. The experimental results are shown in Figure 12.

For the easy task of the 8m map, the winning rates for QMIX, RoMIX-r, and RoMIX were 96.67, 96.88, and 96.33%, respectively. It can be seen that the role selection has no effect on the learning strategies. However, the impact is gradually amplified as the difficulty of the task increases. For the superhard task 6h_vs_8z, the winning rates were 48.96, 15.10, and 55.21%, respectively. It can be seen that randomly selecting roles makes it almost impossible to learn effective strategies. In addition, for the 6h_vs_8z map, we suspect that the low winning rate was caused by the random initial set of roles, as there were almost zero wins in two of the three independent experiments. The ablation study clearly validates the impact of role integration on learning strategies and further demonstrates the effectiveness of role selection.

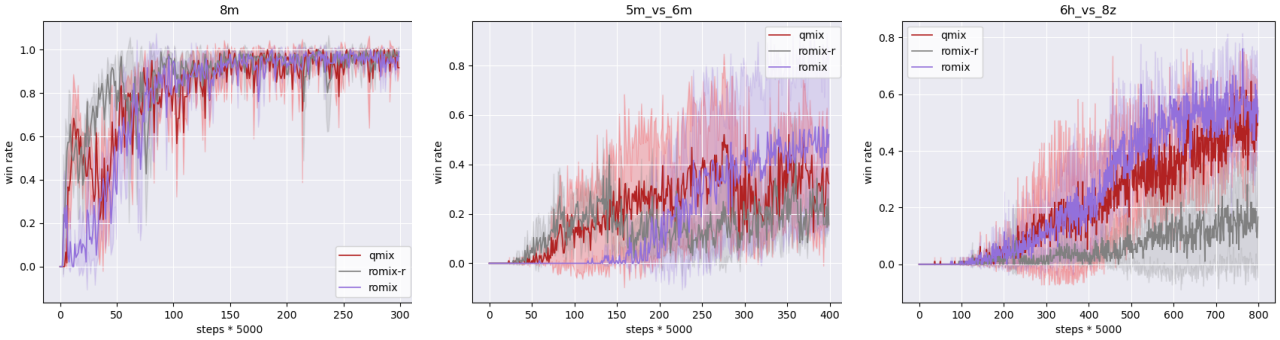


Figure 12. Win rates for the ablation study.

5. Conclusions

This paper presented RoMIX, a MARL method based on value function factorization, which combines role mechanisms for the end-to-end learning of decentralized policies in a centralized setting. RoMIX utilizes action representation to indirectly decompose tasks, and it is combined with a monotonicity constraint to allow for learning role-based joint action-value functions for a subtask. By aggregating the subtasks of all roles, RoMIX learns a rich and effective joint action-value function. Our specific domain for the task was StarCraft, and detailed experimental validations were provided for its decentralized unit micromanagement tasks. The experimental results indicate that RoMIX can not only solve easy scenarios, they also exhibit excellent performance on more complex and difficult tasks. Moreover, it can also reduce storage resources to a certain extent.

In future work, we will focus on the current limitations and further research, particularly, automatic role matching and role scalability problems. For role-based learning, how to match roles is a challenge. And, to be practical, it is crucial to automatically learn an appropriate set of roles. In addition, the StarCraft environment poses a serious challenge to adaptability and reliability due to its dynamism and the diversity and complexity of agents. For our RoMIX, its fixed length mask causes it to lack good scalability and flexibility. Therefore, future work will focus on seeking adaptive role set learning methods to automate role construction, as well as combining attention mechanisms with other variable length masks to further improve scalability. Finally, integrating RoMIX into practical multi-agent

learning system applications with similar special fields is significant, such as drone swarm combat or multi-sensor systems, as it will greatly improve the learning ability of cooperative strategies in complex environments.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This work was funded by the National Natural Science Foundation of China (No. 62076080), the Natural Science Foundation of Chongqing (No. CSTB2022NSCQ-MSX0922), and the Postdoctoral Science Foundation of Heilongjiang Province of China (No. LBH-Z22175).

Conflict of interest

The authors declare there is no conflicts of interest.

References

1. O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, et al., Grandmaster level in StarCraft II using multi-agent reinforcement learning, *Nature*, **575** (2019), 350–354. https://doi.org/10.1038/s41586-019-1724-z
2. M. Samvelyan, T. Rashid, C. S. De Witt, G. Farquhar, N. Nardelli, T. G. Rudner, et al., The starcraft multi-agent challenge, preprint, arXiv:1902.04043.
3. W. Huang, Q. Yin, J. Zhang, K. Huang, Learning Macromanagement in Starcraft by Deep Reinforcement Learning, *Sensors*, **21** (2021), 3332. https://doi.org/10.3390/s21103332
4. M. Kim, J. Oh, Y. Lee, J. Kim, S. Kim, S. Chong, et al., The StarCraft multi-agent exploration challenges: Learning multi-stage tasks and environmental factors without precise reward functions, *IEEE Access*, **11** (2023), 37854–37868. http://doi.org/10.1109/ACCESS.2023.3266652
5. A. Dafoe, E. Hughes, Y. Bachrach, T. Collins, K. R. McKee, J. Z. Leibo, et al., Open problems in cooperative AI, preprint, arXiv:2012.08630.
6. Y. Zhang, Z. Mou, F. Gao, J. Jiang, R. Ding, Z. Han, UAV-enabled secure communications by multi-agent deep reinforcement learning, *IEEE Trans. Veh. Technol.*, **69** (2020), 11599–11611. http://doi.org/10.1109/TVT.2020.3014788
7. A. Feriani, E. Hossain, Single and multi-agent deep reinforcement learning for AI-enabled wireless networks: A tutorial, *IEEE Commun. Surv. Tutorials*, **23** (2021), 1226–1252. http://doi.org/10.1109/COMST.2021.3063822
8. Z. Yan, Y. Xu, A multi-agent deep reinforcement learning method for cooperative load frequency control of a multi-area power system, *IEEE Trans. Power Syst.*, **35** (2020), 4599–4608. http://doi.org/10.1109/TPWRS.2020.2999890

9. T. Wu, P. Zhou, K. Liu, Y. Yuan, X. Wang, H. Huang, et al., Multi-agent deep reinforcement learning for urban traffic light control in vehicular networks, *IEEE Trans. Veh. Technol.*, **69** (2020), 8243–8256. http://doi.org/10.1109/TVT.2020.2997896
10. P. Hernandez-Leal, B. Kartal, M. E. Taylor, A survey and critique of multiagent deep reinforcement learning, *Auton. Agent. Multi-Agent Syst.*, **33** (2019), 750–797. https://doi.org/10.1007/s10458-019-09421-1
11. T. T. Nguyen, N. D. Nguyen, S. Nahavandi, Deep reinforcement learning for multiagent systems: A Review of challenges, solutions, and applications, *IEEE Trans. Cybern.*, **50** (2020), 3826–3839. https://doi.org/10.1109/TCYB.2020.2977374
12. J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, S. Whiteson, Counterfactual multi-agent policy gradients, in *Thirty-Second AAAI Conference on Artificial Intelligence*, **32** (2018), 2974–2982. https://doi.org/10.1609/aaai.v32i1.11794
13. C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, et al., The surprising effectiveness of PPO in cooperative multi-agent games, preprint, arXiv:2103.01955.
14. P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, et al., Value-decomposition networks for cooperative multi-agent learning, preprint, arXiv:1706.05296.
15. T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, S. Whiteson, Monotonic value function factorisation for deep multi-agent reinforcement learning, preprint, arXiv:1803.11485.
16. C. Claus, C. Boutilier, The dynamics of reinforcement learning in cooperative multiagent systems, in *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, (1998), 746–752.
17. M. Tan, Multi-agent reinforcement learning: Independent vs. cooperative agents, in *Proceedings of the Tenth International Conference on Machine Learning*, (1993), 330–337. https://doi.org/10.1016/B978-1-55860-307-3.50049-6
18. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, et al., Playing atari with deep reinforcement learning, preprint, arXiv:1312.5602.
19. W. Qi, S. E. Ovur, Z. Li, A. Marzullo, R. Song, Multi-sensor guided hand gesture recognition for a teleoperated robot using a recurrent neural network, *IEEE Robot. Autom. Lett.*, **6** (2021), 6039–6045. https://doi.org/10.1109/LRA.2021.3089999
20. W. Qi, A. Aliverti, A multimodal wearable system for continuous and real-time breathing pattern monitoring during daily activity, *IEEE J. Biomed. Health Inf.*, **24** (2020), 2199–2207. https://doi.org/10.1109/JBHI.2019.2963048
21. Q. Fu, T. Qiu, J. Yi, Z. Pu, S. Wu, Concentration network for reinforcement learning of large-scale multi-agent systems, in *AAAI Technical Track on Multiagent Systems*, (2022), 9341–9349. https://doi.org/10.1609/aaai.v36i9.21165
22. Y. Wang, C. W. De Silva, Multi-robot box-pushing: Single-agent Q-Learning vs. team Q-Learning, in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (2006), 3694–3699. https://doi.org/10.1109/IROS.2006.281729
23. A. Galindo-Serrano, L. Giupponi, Distributed Q-learning for aggregated interference control in cognitive radio networks, *IEEE Trans. Veh. Technol.*, **59** (2010), 1823–1834. http://doi.org/10.1109/TVT.2010.2043124

24. X. Wang, T, Sandholm, Reinforcement learning to play an optimal Nash equilibrium in team Markov games, in *Advances in Neural Information Processing Systems*, (2002).

25. G. Arslan, S. Yüksel, Decentralized Q-learning for stochastic teams and games, *IEEE Trans. Autom. Control*, **62** (2017), 1545–1558. http://doi.org/10.1109/TAC.2016.2598476

26. K. Son, D. Kim, W. J. Kang, D. E. Hostallero, Y. Yi, QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning, in *Proceedings of the 36th International Conference on Machine Learning*, (2019), 5887–5896.

27. A. Mahajan, T. Rashid, M. Samvelyan, S.Whiteson, MAVEN: Multi-agent variational exploration, in *33rd Conference on Neural Information Processing Systems*, (2019), 1–12.

28. Y. Yang, J. Hao, B. Liao, K. Shao, G. Chen, W. Liu, et al., Qatten: A general framework for cooperative multiagent reinforcement learning, preprint, arXiv:2002.03939.

29. M. J. Khan, S. H. Ahmed, G. Sukthankar, Transformer-based value function decomposition for cooperative multi-agent reinforcement learning in StarCraft, in *Eighteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, **18** (2022), 113–119. https://doi.org/10.1609/aiide.v18i1.21954

30. C. Wu, F. Wu, T. Qi, Y. Huang, X. Xie, Fastformer: Additive attention can be all you need, preprint, arXiv:2108.09084.

31. T. Rashid, G. Farquhar, B. Peng, S. Whiteson, Weighted QMIX: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning, in *34th Conference on Neural Information Processing Systems*, (2020), 1–12.

32. J. Wang, Z. Ren, T. Liu, Y. Yu, C. Zhang, QPLEX: Duplex dueling multi-agent Q-learning, preprint, arXiv:2008.01062.

33. Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, N. Freitas, Dueling network architectures for deep reinforcement learning, in *Proceedings of The 33rd International Conference on Machine Learning*, (2016), 199–2003.

34. S. Iqbal, F. Sha, Actor-attention-critic for multi-agent reinforcement learning, in *Proceedings of the 36th International Conference on Machine Learning*, (2019), 2961–2970.

35. T. Zhang, H. Xu, X. Wang, Y. Wu, K. Keutzer, J. E. Gonzalez, et al., Multi-agent collaboration via reward attribution decomposition, preprint, arXiv:2010.08531.

36. Y. Liu, W. Wang, Y. Hu, J. Hao, X. Chen, Y. Gao, Multi-agent game abstraction via graph attention neural network, in *Proceedings of the AAAI Conference on Artificial Intelligence*, **34** (2020), 7211–7218. https://doi.org/10.1609/aaai.v34i05.6211

37. T. Wang, T. Gupta, A. Mahajan, B. Peng, S. Whiteson, C. Zhang, RODE: Learning roles to decompose multi-agent tasks, preprint, arXiv:2010.01523.

38. J. Zhao, Y. Lv, Output-feedback Robust Tracking Control of Uncertain Systems via Adaptive Learning, *Int. J. Control Autom. Syst.*, **21** (2023), 1108–1118. https://doi.org/10.1007/s12555-021-0882-6

39. Y. Wang, Z. Liu, J. Xu, W. Yan, Heterogeneous network representation learning approach for ethereum identity identification, *IEEE Trans. Comput. Soc. Syst.*, **10** (2023), 890–899. https://doi.org/10.1109/TCSS.2022.3164719

40. F. A. Oliehoek, C. Amato, *A Concise Introduction to Decentralized POMDPs* , Springer, Cham, 2016. https://doi.org/10.1007/978-3-319-28929-8

41. M. Hausknecht, P. Stone, Deep recurrent Q-learning for partially observable MDPs, preprint, arXiv:1507.06527.

AIMS Press