QDN: An Efficient Value Decomposition Method for Cooperative Multi-agent Deep Reinforcement Learning

Zaipeng Xie[1,2],Yufeng Zhang[1,2], Pengfei Shao[1,2], and Weiyi Zhao[3]
[1]Key Laboratory of Water Big Data Technology of Ministry of Water Resources, Hohai University, Nanjing, China
[2]College of Computer and Information, Hohai University, Nanjing, China
[3]The University of Hong Kong, Hong Kong, China
Email: {zaipengxie, yufengzhang, pengfeishao}@hhu.edu.cn, wyzhao99@connect.hku.hk

***Abstract*—Multi-agent systems have recently received significant attention from researchers in many scientific fields. The value factorization method is popular for scaling up cooperative reinforcement learning in multi-agent environments. However, the approximation of the joint value function may introduce a significant disparity between the estimated and actual joint reward value function, leading to a local optimum for cooperative multi-agent deep reinforcement learning. In addition, as the number of agents increases, the input space grows exponentially, negatively impacting the convergence performance of multi-agent algorithms. This work proposes an efficient multi-agent reinforcement learning algorithm, QDN, to enhance the convergence performance in cooperative multi-agent tasks. The proposed QDN scheme utilizes a competitive network to enable the agents to learn the value of the environmental state without the influence of actions. Hence, the error between the estimated joint reward value function and the actual joint reward value function can be significantly reduced, preventing the emergence of sub-optimal actions. Meanwhile, the proposed QDN algorithm utilizes the parametric noise on the network weights to introduce randomness in the network's weights so that the agents can explore the environments and states effectively, thereby improving the convergence performance of the QDN algorithm. We evaluate the proposed QDN scheme using the SMAC challenges with various map difficulties. Experimental results show that the QDN algorithm excels in the convergence speed and the success rate in all scenarios compared to some state-of-the-art methods. Further experiments using four additional multi-agent tasks demonstrate that the QDN algorithm is robust in various multi-agent tasks and can significantly improve the training convergence performance compared with the state-of-the-art methods.**

***Index Terms*—Deep reinforcement learning, cooperative multi-agent systems, convergence performance**

I. Introduction

Multi-agent systems (MAS) [1] have received major attention from researchers in many scientific fields in recent years. Many real-world problems can be solved using cooperation among multiple agents, such as autonomous driving [2], traffic signal control [3], and distributed decision-making [4]. These agents can often make independent decisions and cooperate to form a cooperative multi-agent system [5]. It is desired to focus on helpful information in the environment to quickly get valuable samples and thus obtain positive reward feedback under limited time and resources. However, the number of states and the action spaces of agents increases exponentially with the expanding scale in cooperative multi-agent scenarios and hence leads to dimensional disaster [6]. However, many methods, such as the value function approximation methods [7], the state clustering method [8], and the hierarchical reinforcement learning [9], can be used to mitigate the dimensional disaster problem to some extent. In these methods, experience is usually buffered to instruct the agent to select the best actions in a specific state. Because the agents interact with the environment for experience acquisition, the exploration to accumulate experience can lead to an increased convergence time. Therefore, the number of state spaces and the selection of suboptimal actions [10] are the key factors in the convergence speed of the cooperative multi-agent system.

Learning in cooperative MAS can be difficult [11] due to the existence of partial observability and the local viewpoints of agents. A Partially Observable Markov Decision Process (POMDP) [12] may capture the dynamics of many real-world environments by explicitly acknowledging that the sensations received by the agents are only partial glimpses of the underlying system state. POMDP can be described as a 6-tuple (S, A, P, R, Ω, O), where S, A, P and R, are the states, actions, transitions, and rewards; and Ω is a set of all possible observational vectors of o. This observation is generated from the underlying system state according to the probability distribution $o \sim O(s)$. In the general case, estimating a Q-value from observation can be inefficient because $Q(o, a|\theta) \neq Q(s, a|\theta)$. In the POMDP settings, the agent usually only gets noisy and incomplete observations about the true nature of the state. An optimal action choice cannot be made based on its current observations. Consequently, the agent may be unable to choose the optimal action, eventually leading to a slower convergence.

This work introduces a novel reinforcement learning scheme, Q-Decomposition Network (QDN), to mitigate the local optima problem and improve the convergence speed.

The corresponding author is Zaipeng Xie. This work is supported by The Belt and Road Special Foundation of the State Key Laboratory of Hydrology-Water Resources and Hydraulic Engineering under Grant 2021490811, and the National Natural Science Foundation of China under Grant No. 61872171.

The proposed QDN scheme utilizes a competitive network to enable the agents to learn the value of the environmental state without the influence of actions. Hence, the error between the estimated joint reward value function and the actual joint reward value function can be significantly reduced, preventing the emergence of sub-optimal actions. Meanwhile, the proposed QDN algorithm utilizes the parametric noise on the network weights to randomize the network's weights so that the agents can explore the environments and states effectively, thereby improving the convergence performance of the QDN algorithm. Our contributions can be summarized as follows:

- We propose an improved multi-agent reinforcement learning algorithm based on the Q-Decomposition Network (QDN). Our algorithm utilizes the competitive network structure to enable agents to learn the value of the environmental state. The error between the estimated joint reward value function and the actual joint reward value function can be reduced to prevent the agent from getting suboptimal rewards.
- Our proposed QDN algorithm uses a network structure called Stochastic_Net that can randomly perturb the state information at the early stage of training. Furthermore, Stochastic_Net takes the global state as an input to mitigate the partial observability and maximize the rewards.
- We choose SMAC as the evaluation environment for the proposed QDN algorithm by varying the difficulty of the SMAC scenarios. The experimental results show that the QDN algorithm excels in the convergence speed and success rate in all scenarios compared to some state-of-the-art methods. We extend our algorithm for four additional multi-agent tasks, and the experimental results demonstrate that the QDN algorithm can achieve a decent performance in various multi-agent tasks.

II. Related Work

Multi-agent deep reinforcement learning frameworks [13] can tackle high-dimensional state and action spaces in MAS tasks. However, the error between the estimated and actual action values may lead to the agents falling into the local optima [14]. Hence, several methods [15]–[20] have been proposed to mitigate the problems.

Hasselt et al. [15] propose a Double Deep Q-learning (DDQN) algorithm to decouple the action selection and evaluation using different value functions, thereby reducing the overestimation of the Q value by DDQN during the training process. Schaul et al. [16] propose a Prioritized Replay Deep Q-learning algorithm (DQN) that adds an experience buffer ordered by experience priority. The experience buffer can assign different weights to each experience sample. The agent can learn from the essential experience samples to speed up training and obtain rewards quickly. However, the experience samples are collected at fixed intervals, and much historical information is ignored. Sunehag et al. [17] propose the Value-Decomposition Network (VDN) algorithm to decompose the joint action-value function into a simple summation of the action-value function of each agent. Rashid et al. [18] propose the QMIX algorithm to remove the restriction on the representation of the centralized joint action-value function in the VDN algorithm. However, the monotonicity constraint imposed by the QMIX algorithm is a sufficient but unnecessary condition. There are some scenarios where the value function may not be accurately fitted by QMIX [21]. Son et al. [19] propose the QTRAN algorithm to relax further the restriction on the cumulative or monotonic relationship between joint action-value function and individual action-value function in the VDN and QMIX algorithm. However, the QTRAN algorithm may not satisfy the exact Individual-Global-Max (IGM) consistency [19] due to approximations. Yang et al. [20] propose the Qatten algorithm to simplify the joint action-value function to a low-order linear combination of individual action-value functions. They use a multi-head attention mechanism neural network to approximate the joint action-value function.

In recent years, the Centralized Training with Decentralized Execution (CTDE) framework [17] has been widely used in many multi-agent reinforcement learning frameworks. CTDE allows each agent to learn the action value function in a decentralized manner, and then individual actions can be centralized to fit the joint action value function. As the size of the multi-agent system scales up, the computational complexity of the joint action value function grows exponentially, which may prolong the convergence time of the algorithm. Lowe et al. [22] propose a multi-agent deep deterministic policy gradient algorithm, where each agent has its Actor-network, Critic network, and reward function. However, the input space grows exponentially when the number of agents increases. Iqbal et al. [23] propose a multi-actor-attention-critic algorithm by introducing an attention-based mechanism to allow agents to focus on other agents to learn the Critic network selectively. However, in simple multi-agent tasks, the attention mechanism sometimes prolongs the convergence time of the algorithm. Therefore, it is necessary to explore the value decomposition method under the CTDE framework for improving the convergence performance of cooperative MAS.

III. Motivations

The existing value decomposition algorithms [17]–[19] usually employ approximations for their joint reward value function and thus may not accurately apprehend the relationship between the joint reward value function and the local reward value function. Consequently, agents may fall into the local optima in a complex multi-agent scenario, increasing exploration time. In some sparse rewards tasks, the exploration spaces may grow exponentially [24] when the number of agents scales up, leading to a significant increase in the convergence time performance. In general, the convergence time performance is impacted by the local optima and the scale of the state space.

A. Effect of local optimum on the convergence speed

The VDN [17] method is one of the first algorithms that propose the decomposition of the joint reward value function to solve the convergence speed problem shown in Eq.1.

$$Q_{tot}(s,a) = \sum_{i=1}^{n} Q_i(s_i,a_i), \tag{1}$$

where $Q_{tot}(s,a)$ is the joint reward value function, s is the global state, a is the selected joint action, i is the ordinal agent number, and n is the number of all agents. The VDN algorithm approximates the local reward value function $Q_i(s_i,a_i)$ of a single agent to the joint reward value function $Q_{tot}(s,a)$. Then the algorithm updates $Q_{tot}(s,a)$ using DQN, and after that, the gradient is passed backward through $Q_{tot}(s,a)$ to $Q_i(s_i,a_i)$ to achieve a global perspective to update the agent based on the local information-based $Q_i(s_i,a_i)$. However, VDN cannot represent the non-summing relation $Q_{tot}(s,a)$ well.

The QMIX [18] algorithm takes advantage of centralized training by capturing the additional available state information during the training. QMIX assumes that the relationship between $Q_{tot}(s,a)$ and $Q_i(s_i,a_i)$ is monotonically increasing because the ultimate goal is to learn a higher $Q_{tot}(s,a)$. All parameters in the QMIX neural network are restricted to be non-negative, thus satisfying the following equation:

$$\frac{\partial Q_{tot}(s,a)}{\partial Q_i(s_i,a_i)} \geq 0, \forall i \in [1,n] \tag{2}$$

However, the relationship between $Q_{tot}(s,a)$ and $Q_i(s_i,a_i)$ can be limited due to these assumptions. The approximate $Q_{tot}(s,a)$ can differ significantly from the actual value. Thus it may not be able to fit complex MAS tasks and lead to a degraded convergence performance.

The authors of REFIL [25] propose an attention-based Q-value hybrid network in QMIX to generate a random mask group of agents. REFIL utilizes a generalized form to describe the relationship between the joint reward value and the local reward value function without imposing additional assumptions and constraints. It can promote generalization across tasks by breaking value function predictions into reusable components.

QTRAN [19] removes the restriction between the joint reward value function and the local reward value function so that $[Q_i(s_i,a_i)]_{i=1}^{n}$ satisfies the Individual-Global-Max for $Q_{tot}(s,a)$ as long as the individual optimal action and optimal joint action are guaranteed to be the same. Therefore, the relationship for $Q_{tot}(s,a)$ is not necessarily required, and thus QTRAN can be applied to more general complex multi-agent scenarios. Hence, $Q_{tot}(s,a)$ can be obtained by cumulative approximation. Since the cumulative $Q_{tot}(s,a)$ and the real $Q_{tot}(s,a)$ are much different, the compensation term $V(s)$ is introduced [19] to offset the gap between them. However, QTRAN cannot prevent the agents from being stuck in local optima.

In summary, the joint reward value function estimation can be inefficient in the state-of-the-art MADRL methods. In more general and complex scenarios, the agents may easily fall into local optima; hence, obtaining a fast convergence rate in MADRL is still challenging.

B. Effect of state spaces on the convergence speed

Some value decomposition algorithms [26] rely on random perturbation of the agent's strategy. However, due to the randomness, the perturbation-based exploration can be inefficient [26] in a complex multi-agent environment. Figure 1 shows an example that illustrates the effect of the number of state spaces on the value-decomposition-based convergence speed, where six state spaces are presented, and the agent explores from state space 1 to state space 6 using the ϵ-greedy algorithm. The

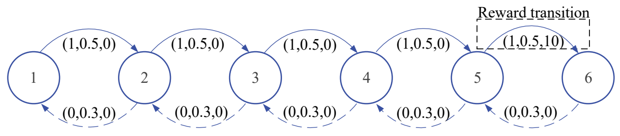

Fig. 1. Example diagram of the state transition process

triad represents: (direction, probability, reward). The direction can be 0 or 1, indicating the left and right direction; the probability indicates the possibility of taking actions, and the reward describes the expected reward value. If the agent takes the same action with the same reward without constraint, it can quickly learn the action from state 1 to state 6. However, only state 5 to state 6 can obtain the reward value in Fig. 1. Therefore, the probability that the agent reaches state 6 and obtains the reward is $P = 0.3^m \times 0.5^n$ (m,n are the number of times moving left and right). As the number of spaces explored by the agents grows, the probability of reaching the goal may go to zero and the convergence performance of the MADRL algorithm can be negatively impacted. Hence, exploring methods to improve the convergence performance of the MADRL algorithm is still an open research topic.

IV. Method

We propose an improved multi-agent reinforcement learning scheme, Q-Decomposition Network (QDN), to improve the convergence performance of MAS. This algorithm uses random noise and competitive networks for value function estimation to improve the convergence performance of the value decomposition algorithm in large-scale cooperative MAS tasks. Our scheme mainly consists of three neural networks, namely, the Agent_Net, the Stochastic_Net, and the Competitive_Net, and each neural network is built in a modular way.

Figure 2 illustrates the flow of the proposed QDN scheme. The Agent_Net decouples the environmental state assessment and action value assessment and converts the input information, including observed value o_i^t of the agent at moment t and action a_i^{t-1} at moment $t-1$, into the corresponding value function $Q_i(\tau_i)$.

The Stochastic_Net is a random noise neural network that can perturb the state information explored by the agents to avoid reward sparsity. The Stochastic_Net combines $Q_i(\tau_i)$ with global state S_t perturbed by a noise parameter as the weight, then it outputs the agent's reward value function $Q_i(\tau)$ based on the global information τ.

The Competitive_Net can facilitate agents learning the environmental state's value by decomposing the joint reward value function into the state value function and the dominance value function. Using Competitive_Net can reduce the error between the estimated value and actual value and prevent

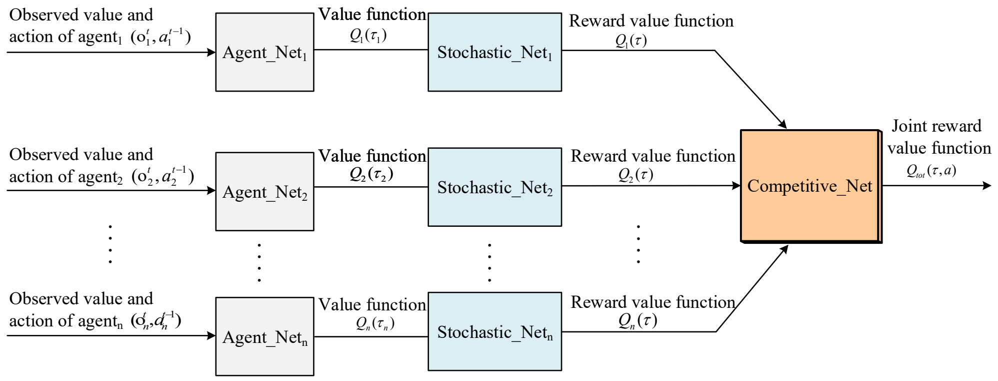
Observed value and action of agent$_1$ (o_1^t, a_1^{t-1})
Agent_Net$_1$
Value function $Q_1(\tau_1)$
Stochastic_Net$_1$
Reward value function $Q_1(\tau)$
Observed value and action of agent$_2$ (o_2^t, a_2^{t-1})
Agent_Net$_2$
Value function $Q_2(\tau_2)$
Stochastic_Net$_2$
Reward value function $Q_2(\tau)$
Competitive_Net
Joint reward value function $Q_{tot}(\tau, a)$
Observed value and action of agent$_n$ (o_n^t, a_n^{t-1})
Agent_Net$_n$
Value function $Q_n(\tau_n)$
Stochastic_Net$_n$
Reward value function $Q_n(\tau)$

Fig. 2. Algorithmic flowchart of the proposed QDN algorithm.

the agents from choosing actions with suboptimal rewards. The Competitive_Net also translates the IGM consistency into an easy-to-implement dominance function by taking range constraints, thus facilitating the learning of value functions. Algorithm 1 presents the detailed procedure of the proposed QDN scheme.

Algorithm 1: QDN algorithm

```
Input: Agent number $i$, observation value $o_i^t$ at
       moment $t$, action $a_i^{t-1}$ executed at previous
       moment $t-1$, global state $S_t$, experience
       buffer size $d$, training rounds $m$, update cycle
       $T$ of Target_Agent_Net
Output: Joint reward value function $Q_{tot}$
for $episode = 1$ to $m$ do
    for $timestep = t-1$ to $t$ do
        Send $o_i^t$ and $a_i^{t-1}$ to Eval_Agent_Net;
        Save $(S_t, S_{t+1}, o_i^t, a_i^t, Q_i(\tau_i))$ as $D$;
        if $episode \% T = 0$ then
            Parameters: Eval_Agent_Net $\rightarrow$
             Target_Agent_Net
        else if $D > d$ then
            Empirical sampling
        else
            Send $S_t$ and $Q_i(\tau_i)$ to Stochastic_Net;
            Random noise perturbs $S_t$;
            $S_t$ and $Q_i(\tau_i)$ for weighted summation;
            Get $Q_i(\tau)$ based on $\tau$;
            Send $S_t$ and $Q_i(\tau)$ to Competitive_Net;
            Decompose $Q_i(\tau)$ into $A_i(\tau)$ and $S_i(\tau)$;
            $A_i(\tau) \leftarrow \delta \cdot A_i(\tau)$;
            $A_{tot}(\tau) \leftarrow \sum_{i=1}^{n} A_i(\tau)$;
            $S_{tot}(\tau) \leftarrow \sum_{i=1}^{n} S_i(\tau)$;
            $Q_{tot} \leftarrow A_{tot}(\tau) + S_{tot}(\tau)$;
    end
end
```

A. Agent_Net

The Agent_Net network is devised to evaluate the fitness of the agent's action. Its input and output layers are implemented as a multilayer perceptron (MLP), and the hidden layers consist of Gate Recurrent Unit (GRU). Because the multi-agent environment is considered partially observable, we believe this problem can be mitigated by adding recurrent neural network layers. The GRU is not computationally intensive and can be easy to train. All traces during execution are saved to the experience pool, including the states, observations, actions, and rewards, for the training of GRU.

Figure 3(a) describes the internal structure of Agent_Net. The input to Agent_Net is the observation o_i^t of agent i at moment t and its action a_i^{t-1} performed at the previous moment. The output is value function $Q_i(\tau_i)$ of agent i based on local information τ_i. The state information of hidden layer at moment $t-1$ and moment t are denoted as h_i^{t-1} and h_i^t. ReLU is employed as the activation function. We implement the Agent_Net using a similar idea as the Deep Q-networks described in [27], where two neural networks with the identical structure are implemented, namely the Eval_Agent_Net and Target_Agent_Net. The Eval_Agent_Net is updated using gradient descent, while the Target_Agent_Net is randomly initialized and updated by a direct copy of the val_Agent_Net's parameters after a fixed batch of training. This implementation can significantly improve the stability of our proposed QDN algorithm. The Eval_Agent_Net receives input information (o_i^t, a_i^{t-1}) of agent i and it send the parameters of its network to the corresponding Target_Agent_Net after a fixed time step to minimize the correlation between data. The Target_Agent_Net updates its loss function after receiving new parameters and it saves the tuple $(S_t, S_{t+1}, o_i^t, a_i^t, Q_i(\tau_i))$ in its experience buffer. If the experience buffer is full, not only are Target_Agent_Net parameters updated but also the experiences with the same observation in the buffer are grouped.

B. Stochastic_Net

The Stochastic_Net is a neural network that consists of a random noise module and a weighted summation mod-

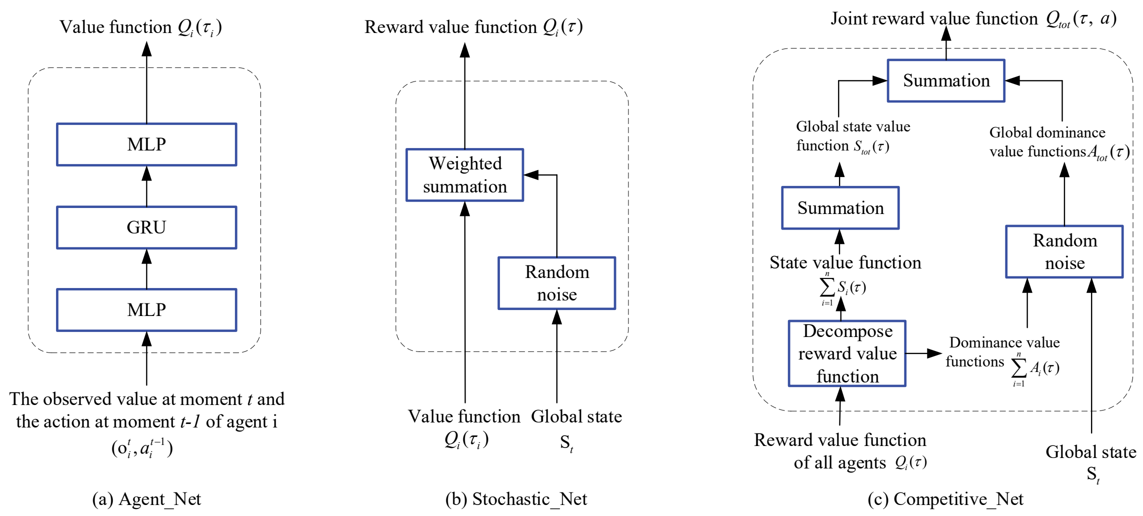
Value function $Q_i(\tau_i)$
MLP
GRU
MLP
The observed value at moment *t* and the action at moment *t-1* of agent i
(o_i^t, a_i^{t-1})
(a) Agent_Net
Reward value function $Q_i(\tau)$
Weighted summation
Random noise
Value function $Q_i(\tau_i)$
Global state S_t
(b) Stochastic_Net
Joint reward value function $Q_{tot}(\tau, a)$
Summation
Global state value function $S_{tot}(\tau)$
Global dominance value functions $A_{tot}(\tau)$
Summation
State value function $\sum_{i=1}^{n} S_i(\tau)$
Random noise
Decompose reward value function
Dominance value functions $\sum_{i=1}^{n} A_i(\tau)$
Reward value function of all agents $Q_i(\tau)$
Global state S_t
(c) Competitive_Net

Fig. 3. The structures of the proposed Agent_Net, Stochastic_Net and Competitive_Net

ule, as shown in Fig.3(b). A random function perturbs the Stochastic_Net's weight and bias parameters to facilitate the agent's information exploration. The parameters of the random noise module introduce uncertainty to the network parameters, which can be adaptively adjusted during the training process.

Its input information is the agent's reward value function $Q_i(\tau_i)$ based on local information τ_i and current global state S_t at moment t. The random noise module adds parameterized noise to S_t and randomizes the network's weight and bias parameters. The weighted summation module then combines $Q_i(\tau_i)$ and S_t and outputs the agent's reward value function $Q_i(\tau)$ based on the global information. The effect of noise perturbation on global state S_t is mainly to weaken the influence of partial observability in a cooperative multi-agent environment. It also combines the agent's global and local state information to speed up the exploration efficiency in the early stage of the agent's training. The noise parameter θ in the random noise module is defined as give by

$$\theta = \mu + \sum \odot \varepsilon, \tag{3}$$

where μ and $\sum$ are vectors of learnable noise parameters, ε is a vector of zero-mean noise with a fixed statistic, and $\odot$ denotes element-by-element multiplication. The output of the noise layer can be described by

$$y = (\mu^w + \sigma^w \odot \varepsilon^w) \cdot x + \mu^b + \sigma^b \odot \varepsilon^b, \tag{4}$$

where x is global state S_t sent to random noise module, and y is state $S_t{}'$ after noise randomization perturbation. The weight parameter term $\mu^w + \sigma^w \odot \varepsilon^w$ should be a positive number for the agent to receive beneficial feedback. Its parameters and loss function are updated after Stochastic_Net outputs the reward value function $Q_i(\tau)$ for each agent based on the global information τ.

C. *Competitive_Net*

Competitive_Net decomposes joint reward value function into state value function and dominance value function. It divides the difference of each agent by dominance function, which enhances the agent's perception of a complex multi-agent environment and motivates the agent to complete the cooperative task better and faster. Competitive_Net also redefines the Individual-Global-Max (IGM) principle [28] as dominance function-based IGM. The Competitive_Net converts IGM consistency into a constraint on the range of values of the dominance function. It also eliminates the restriction on the relationship between joint reward value function and local reward value function; this makes QDN algorithm applicable to more complex and variable cooperative multi-agent scenarios and improves the generality of the QDN algorithm. The structure of Competitive_Net is shown in Fig.3(c).

Competitive_Net takes reward value function $Q_i(\tau)$ of all agents based on global information τ and global state S_t outputted by Stochastic_Net as input. After summing the reward value function $Q_i(\tau)$, the estimated state value function $S_i(\tau)$ is obtained according to the current state of each agent. The reward value function $Q_i(\tau)$ minus state value function $S_i(\tau)$ gives the advantage value function $A_i(\tau)$ of each agent based on global information τ, as given by

$$A_i(\tau) = Q_i(\tau) - S_i(\tau). \tag{5}$$

The dominance value function $A_i(\tau)$ can be used to measure the fitness of actions performed by agents. After decomposing $Q_i(\tau)$ into $A_i(\tau)$ and $S_i(\tau)$, a random noise module is added. Each dominance value function $A_i(\tau)$ is based on global information τ and $A_i(\tau)$ is multiplied by weight δ that is trained by the random noise module to avoid the inability to discriminate the good or bad actions of agents due to identical agents' dominance value functions $A_i(\tau)$. Then, the dominance value function $\tau \cdot A_i(\tau)$ and the state value function $S_i(\tau)$ are summed to obtain $A_{tot}(\tau)$ and $S_{tot}(\tau)$ for each agent after noise perturbation. Finally, $A_{tot}(\tau)$ and $S_{tot}(\tau)$ are summed to obtain the joint reward value function $Q_{tot}(\tau, a)$ of the multi-agent system based on global information τ. The joint reward value function $Q_{tot}(\tau, a)$ can be described as follows:

$$\begin{aligned} Q_{tot}(\tau, a) &= S_{tot}(\tau) + A_{tot}(\tau) \\ &= \sum_{i=1}^{n} Q_i(\tau) + \delta \cdot \sum_{i=1}^{n} A_i(\tau) \end{aligned} \tag{6}$$

Equation 6 shows that the magnitude of the joint reward value function $Q_{tot}(\tau, a)$ is constrained by the range of values of the dominant value function $A_i(\tau)$. This constraint does not need to consider the relationship between the local reward value function $Q_i(\tau)$ and $Q_{tot}(\tau, a)$ as in the traditional value decomposition algorithm. Thus, the QDN algorithm fully expresses IGM and is suitable for more general and complex multi-agent scenarios. After the value decomposition of each agent, Competitive_Net updates its parameters and loss function and distributes the updated information to each agent, where an action strategy is developed as the joint reward value function $Q_{tot}(\tau, a)$ is obtained. The observation value and action value are hence fed into Agent_Net, and the previous value decomposition process is repeated until the training is terminated.

V. Experiment

A. Experimental Setup

We choose StarCraft 2 (SMAC) [29] as the test environment for the proposed QDN scheme. All the agents receive a positive reward after defeating an opposing combat unit. The total reward for the entire multi-agent system equals to the overall received rewards for all the damages inflicted on the enemy combat units. Twelve SMAC maps are used, including 1c3s5z, 2s3z, 2s_vs_1sc, 3s5z, 3s_vs_5z, 5m_vs_6m, 5s10z, 8m, 10m_vs_11m, 25m, MMM, so_many_baneling. The difficulty of the maps varies from the easy to difficult level. Our proposed QDN scheme is compared with the performance of the RELIF [25], QTRAN [19], QMIX [18] and VDN [17] algorithms.

In addition to the SMAC multi-agent challenge, we elect four additional multi-agent tasks for the experiments to demonstrate the robustness under various test environments [30], including *Pass*, *Secret-room*, *Push-box*, and *Island*.

In the training experiments, each agent develops its strategy based on the local observation and performs the actions according to its own strategy. The discount factor γ for the cumulative reward of all agents is 0.98, the learning rate α is 0.005, the experience buffer contains a maximum of 5000 experience tuples, and 1000 experience tuples are adopted at each time. The Eval_Agent_Net sends its parameter list to the Target_Agent_Net every 500 batches. The intermediate dimension of the network layer is 32 for the Agent_Net and 64 for both the Stochastic_Net and the Competitive_Net. The maximum number of time steps for all experiments is five million.

B. Experimental Results

*1) **The convergence performance using SMAC:*** The comparison of our proposed QDN scheme and four state-of-the-art algorithms are presented in Fig. 4. The results are the average of each algorithm running five times with different random seeds.

We observe that the average success rate of the proposed QDN algorithm fluctuates and converges to about 95% after one million timesteps in the easy SMAC challenges. The average success rates of both the REFIL and QTRAN algorithms can reach 90% at 1.5 million timesteps. Although the average success rate of REFIL and QTRAN is comparable with that of QDN, the convergence speed is much slower than our proposed QDN scheme. Furthermore, the VDN and QMIX algorithms are the worst among all algorithms. The average success rate of the VDN and QMIX algorithms stabilizes at about 75%$\sim$80%. However, both algorithms exhibit a slow convergence rate, requiring approximately 2.5 million timesteps to stabilize. Our proposed QDN scheme excels in the convergence speed due to its extracting logical topological relations between agents using random noise and the competitive network.

For the maps with normal difficulties, the experimental results show that, in the normal SMAC challenges, the average success rate of the QDN algorithm stabilizes at approximately 95% after 1.5 million timesteps. Meanwhile, the REFIL, QTRAN, QMIX and VDN algorithm do not behave as effectively as our proposed QDN in terms of the convergence rate. The average success rate of RELIF and QTRAN algorithms can reach 90% after 2 million timesteps. Nevertheless, the average success rate of QMIX and VDN algorithms can barely reach 80% even after 4 million timesteps.

For the difficult SMAC maps, we can make a similar conclusion about our proposed QDN scheme, i.e., the average success rate of the QDN algorithm stabilizes above approximately 95% after 1.5 million timesteps. In contrast, the QTRAN algorithm takes an average of 2.5 million timesteps to reach a 95% success rate. In contrast, the convergence speeds of RELIF, QMIX, and VDN deteriorate significantly in the difficult SMAC maps. The average success rate of RELIF algorithm stabilizes above approximately 95% after 4 million timesteps. However, the average success rate of both QMIX and VDN algorithms can barely reach 65% even after 5 million timesteps.

In summary, we conclude that our proposed QDN scheme excels in all the SMAC maps regarding the average success rate. It can converge to 95% after about 1.5 million timesteps in all maps. In contrast, the VDN and QMIX algorithms cannot perform well in all challenges. The QTRAN and REFIL algorithms perform slightly better in terms of the convergence rate. However, neither algorithm can achieve a 95% success rate within four million timesteps when the SMAC challenge is difficult. Our proposed QDN algorithm is an excellent candidate to boost the convergence performance, which is capable of reaching a 95% success rate in a timely manner

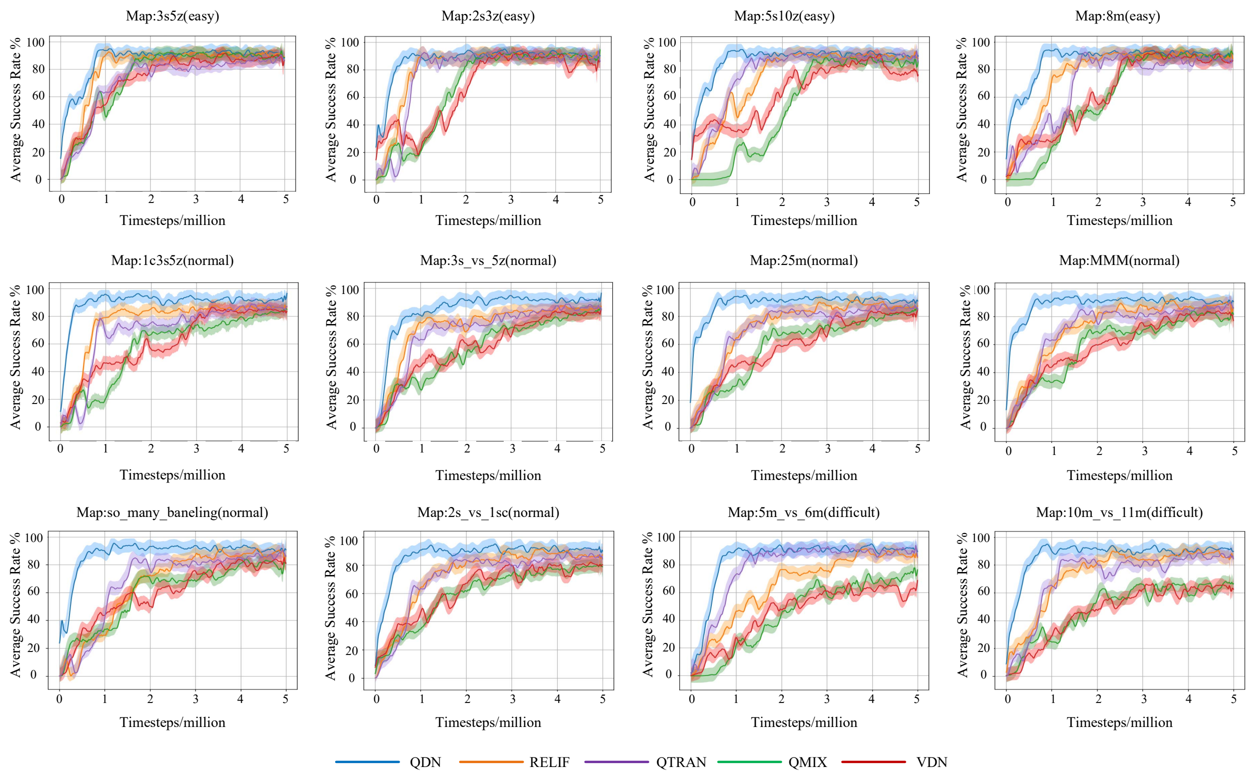
Fig. 4. The performance comparison of our QDN algorithm and four other algorithms in 12 SMAC challenges with varying map difficulties

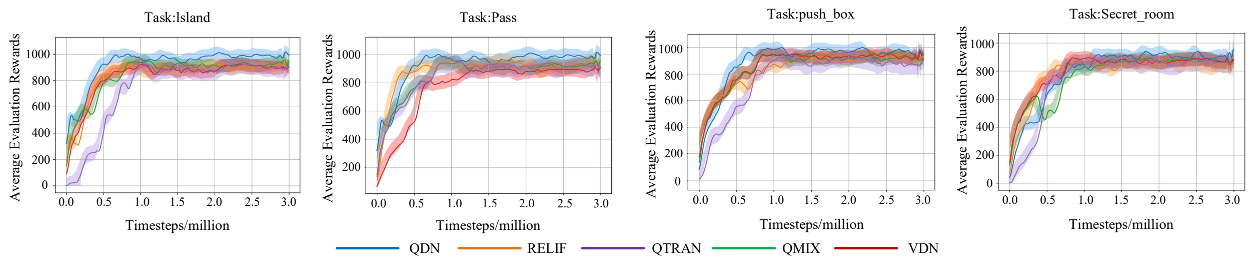
Fig. 5. The average evaluation rewards obtained by the agents versus the timesteps for four MAS tasks

for the SMAC challenges.

2) ***Evaluation using other MAS tasks:*** We extend our experiments by using four additional MAS tasks, including Pass, Secret-room, Push-box, and Island [30]. The purposes of these experiments are to evaluate the QDN algorithm's robustness in various multi-agent challenges. Our proposed QDN scheme is compared with RELIF, QTRAN, QMIX, and VDN algorithms. Each algorithm runs five times using varying random seeds, and the results are averaged for demonstration. Figure 5 presents the average rewards obtained by the agent versus the timesteps. A maximum of three million timesteps is used in these experiments.

We observe that our proposed QDN method can give the highest average rewards compared to four state-of-the-art algorithms, including RELIF, QTRAN, QMIX, and VDN. Furthermore, the convergence performance of our proposed QDN algorithm is also slightly better than that of the four counterparts, i.e., it requires 0.6 million timesteps to get stabilized at the maximum rewards. In comparison, the QTRAN algorithm performs the worst in Island and Push_box tasks; the QMIX performs the worst in the Secret_room task, and the VDN performs the worst in the Pass task. The RELIF algorithm can achieve a similar convergence performance as our QDN scheme, but its stabilized rewards are slightly lower than the QDN algorithm in all four MAS challenges. Therefore, we conclude that our proposed QDN scheme excels in all four different MAS challenges regarding the average rewards and the convergence performance.

VI. Conclusion

This work proposes an improved multi-agent reinforcement learning method, QDN, to implement an efficient cooperative multi-agent reinforcement learning framework. The proposed QDN scheme utilizes a competitive network to enable the agents to learn the value of the environmental state without the influence of actions. Hence, the error between the estimated joint reward value function and the actual joint reward value function can be reduced, preventing the emergence of sub-optimal actions. Meanwhile, the proposed QDN method employs the parametric noise on the network weights to randomize the network's weights so that the agents can explore the environments effectively, thereby improving the convergence performance of the QDN algorithm.

We evaluate the proposed QDN scheme using the SMAC challenges with various map difficulties. The experimental results show that the QDN algorithm excels regarding the convergence performance and the success rate in all challenges compared to some state-of-the-art methods. Furthermore, we extend our experiments using four additional multi-agent tasks, including Pass, Secret-room, Push-box, and Island. The experimental results demonstrate that the QDN algorithm is robust in various MAS tasks and can significantly improve the training convergence performance compared with the state-of-the-art methods.

References

[1] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: a survey," *Artificial Intelligence Review*, vol. 55, no. 2, pp. 895–943, 2022.

[2] J. Hook, S. El-Sedky *et al.*, "Learning data-driven decision-making policies in multi-agent environments for autonomous systems," *Cognitive Systems Research*, vol. 65, pp. 40–49, 2021.

[3] T. Chu, J. Wang, L. Codecà, and Z. Li, "Multi-agent deep reinforcement learning for large-scale traffic signal control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 1086–1095, 2019.

[4] S. Wang, J. Wan *et al.*, "Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination," *Computer networks*, vol. 101, pp. 158–168, 2016.

[5] A. OroojlooyJadid and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," *arXiv preprint arXiv:1908.03963*, 2019.

[6] Z. Tang, W. Jia, X. Zhou, W. Yang, and Y. You, "Representation and reinforcement learning for task scheduling in edge computing," *IEEE Transactions on Big Data*, vol. 8, no. 3, pp. 795–808, 2022.

[7] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-Learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, 2016.

[8] M. Xia, K. Wang, W. Song *et al.*, "Non-intrusive load disaggregation based on composite deep long short-term memory network," *Expert Systems with Applications*, vol. 160, p. 113669, 2020.

[9] A. S. Vezhnevets, S. Osindero, T. Schaul *et al.*, "FeUdal networks for hierarchical reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70. PMLR, 06–11 Aug. 2017, pp. 3540–3549.

[10] S. Devlin and D. Kudenko, "Theoretical considerations of potential-based reward shaping for multi-agent systems," in *the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '11)*, 2011, p. 225–232.

[11] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian, "Deep decentralized multi-task multi-agent reinforcement learning under partial observability," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70. PMLR, 06–11 Aug 2017, pp. 2681–2690.

[12] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," in *2015 AAAI fall symposium series*, 2015.

[13] K. Lin, R. Zhao, Z. Xu, and J. Zhou, "Efficient large-scale fleet management via multi-agent deep reinforcement learning," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1774–1783.

[14] Z.-W. Hong, T.-Y. Shann, S.-Y. Su, Y.-H. Chang, T.-J. Fu, and C.-Y. Lee, "Diversity-driven exploration strategy for deep reinforcement learning," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS '18)*, 2018, p. 10510–10521.

[15] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[16] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[17] P. Sunehag, G. Lever *et al.*, "Value-decomposition networks for cooperative multi-agent learning," *arXiv preprint arXiv:1706.05296*, 2017.

[18] T. Rashid, M. Samvelyan *et al.*, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *International Conference on Machine Learning*, vol. 80. PMLR, 2018, pp. 4295–4304.

[19] K. Son, D. Kim *et al.*, "QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning," in *International Conference on Machine Learning*, vol. 97. PMLR, 2019, pp. 5887–5896.

[20] Y. Yang, J. Hao, B. Liao *et al.*, "Qatten: A general framework for cooperative multiagent reinforcement learning," *arXiv preprint arXiv:2002.03939*, 2020.

[21] T. Rashid, G. Farquhar, B. Peng, and S. Whiteson, "Weighted QMIX: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning," in *Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS '20)*, 2020.

[22] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*, 2017, p. 6382–6393.

[23] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," in *Proceedings of the 36th International Conference on Machine Learning*, vol. 97. PMLR, 9–15 June 2019, pp. 2961–2970.

[24] S. Z. Gou and Y. Liu, "DQN with model-based exploration: efficient learning on environments with sparse rewards," *arXiv preprint arXiv:1903.09295*, 2019.

[25] S. Iqbal, C. A. S. De Witt *et al.*, "Randomized entity-wise factorization for multi-agent reinforcement learning," in *the 38th International Conference on Machine Learning*, vol. 139. PMLR, 18-24 July 2021, pp. 4596–4606.

[26] W. Du and S. Ding, "A survey on multi-agent deep reinforcement learning: from the perspective of challenges and applications," *Artificial Intelligence Review*, vol. 54, no. 5, pp. 3215–3238, 2021.

[27] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of the 33th International Conference on Machine Learning*, vol. 48. PMLR, 19–24 June 2016, pp. 1995–2003.

[28] J. Wang, Z. Ren, T. Liu, Y. Yu, and C. Zhang, "QPLEX: Duplex dueling multi-agent q-learning," *arXiv preprint arXiv:2008.01062*, 2020.

[29] M. Samvelyan, T. Rashid, C. S. De Witt *et al.*, "The starcraft multi-agent challenge," in *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS'19)*, 2019.

[30] T. Wang, J. Wang, Y. Wu, and C. Zhang, "Influence-based multi-agent exploration," in *International Conference on Learning Representations (ICLR)*, 2020.