IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 20, 2021, accepted September 13, 2021, date of publication September 16, 2021,
date of current version September 28, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3113350

QSOD: Hybrid Policy Gradient for Deep Multi-agent Reinforcement Learning

HAFIZ MUHAMMAD RAZA UR REHMAN[1], BYUNG-WON ON[2], DEVARANI DEVI NINGOMBAM[3], SUNGWON YI[3], AND GYU SANG CHOI[1], (Member, IEEE)

[1]Department of Information and Communication Engineering, Yeungnam University, Gyeongsan 38541, South Korea
[2]Department of Software Convergence Engineering, Kunsan National University, Gunsan 54150, South Korea
[3]Planning Division, Electronics and Telecommunications Research Institute, Daejeon 34129, South Korea

Corresponding authors: Gyu Sang Choi (castchoi@ynu.ac.kr), Byung-Won On (bwon@kunsan.ac.kr), and Sungwon Yi (sungyi@etri.re.kr)

This research was supported in part by ETRI grants 19YE1410 and AFOSR grants FA2386-19-1-4020, and by the National Research Foundation of Korea (NRF) Grant by Korean Government through the Ministry of Science and ICT (MSIT) under Grant NRF-2019R1F1A1060752 and Grant NRF-2021R1A6A1A03039493.

ABSTRACT When individuals interact with one another to accomplish specific goals, they learn from others' experiences to achieve the tasks at hand. The same holds for learning in virtual environments, such as video games. Deep multiagent reinforcement learning shows promising results in terms of completing many challenging tasks. To demonstrate its viability, most algorithms use value decomposition for multiple agents. To guide each agent, behavior value decomposition is utilized to decompose the combined Q-value of the agents into individual agent Q-values. A different mixing method can be utilized, using a monotonicity assumption based on value decomposition algorithms such as QMIX and QVMix. However, this method selects individual agent actions through a greedy policy. The agents, which require large numbers of training trials, are not addressed. In this paper, we propose a novel hybrid policy for the action selection of an individual agent known as Q-value Selection using Optimization and DRL (QSOD). A grey wolf optimizer (GWO) is used to determine the choice of individuals' actions. As in GWO, there is proper attention among the agents facilitated through the agents' coordination with one another. We used the StarCraft 2 Learning Environment to compare our proposed algorithm with the state-of-the-art algorithms QMIX and QVMix. Experimental results demonstrate that our algorithm outperforms QMIX and QVMix in all scenarios and requires fewer training trials.

INDEX TERMS Artificial intelligence, multiagent systems, optimization.

I. INTRODUCTION

Recently, reinforcement learning (RL) has proven effective for solving problems related to cooperative multiagent systems (MAS), and the approach has garnered increased attention. Reinforcement learning has shown particular utility for complex tasks such as self-driving vehicles [1], [1], power supply systems, logistics distribution in factories, productivity optimization [2], and cooperative multi-robot exploration systems [3], [4], have commercial prospects in large-scale applications.

Customarily, convergent learning in multiagent reinforcement learning (MARL) is used to tackle the problems of cooperative MAS by considering the MAS as a single agent. Such centralized learning performs remarkably well in many scenarios. However, when using such an approach with an increasing number of agents, the joint action table increases exponentially, and current RL algorithms may not converge on a solution. However, a distributed and decentralized learning approach, where each agent learns individually according to its policy, can handle these problems smoothly. This learning is based on the sum of all the agents' total rewards, which is known as the global reward. Typically, independent Q-learning (IQL) is used in such cases [5]. The main shortcoming of this approach is the occurrence of nonstationary problems, even for only two agents, due to the global reward [6]. To address the non-stationarity problem, it is also possible to train a decentralized policy in a centralized manner.

For the last three to four years in the RL community, this amalgam technique has become very popular [7], [8]. However, even the induction of a hybrid approach cannot

The associate editor coordinating the review of this manuscript and approving it for publication was Yu-Da Lin.

address many of the challenges still faced by MAS. Among these, the most important challenges are the convergence rate and computational power. IQL fails to address these problems because of its non-stationarity nature. Although counterfactual multiagent (COMA) techniques [9] are able to address the convergence problem, they are unable to calculate the combined Q-value from the joint state-action, which is the main criticism leveled against COMA [10]. This shortcoming is attributable to COMA using on-policy learning. Value decomposition networks (VDNs) address this problem, ensuring that learning is performed in a centralized fashion, but the global Q-value function is calculated through a factored approach [11]. In training, a VDN does not utilize the state's additional information because it only presents a shallow class of action values. QMIX [13] and QVMix [36] mitigates this problem by using a neural network to convert the centralized state into the weights of the second neural network, in a manner reminiscent of hyper-networks [12]. However, QVMix required high computational power while QMIX required large training episodes.

StarCraft has been used by many researchers to evaluate deep MARL algorithms, such as in [6], [7], [9], and [13]. Both StarCraft and StarCraft II are the registered trademarks of Blizzard. Almost all of these methods address the convergence issue, but due to their nonstationary environments and greedy policies for action selection, they require either large numbers of training episodes or very high computational power.

In this study, we used the StarCraft II Learning Environment (SC2LE) [13]. We introduce a hybrid policy gradient for deep MARL, known as Q-value Selection using Optimization and DRL (QSOD), to mitigate this problem. It relies on a grey wolf optimizer (GWO) [15], [29]. As in GWO, one agent acts as the head of the group, whereas all other agents act according to the lead agent's instruction. Due to the optimization-based policy agents learn in a faster way with a comparatively small mixer network.

The rest of the paper includes a discussion of prior works in Section 2. In Section 3, we discuss GWO and multiagent systems. In Section 4, we discuss the proposed method. Section 5 presents the experimental results. Finally, Section 6 concludes the discussion and provides an overview of our future directions in this domain.

II. RELATED WORK AND BACKGROUND

The productivity of RL-based techniques over the last several years, particularly in solving cooperative MAS problems, cannot be overlooked. As mentioned previously, MARL uses centralized, decentralized, and hybrid approaches to accomplish its goals. Initially, MARL used centralized approaches [3], [16], which later shifted toward deep learning methods capable of controlling multidimensional states and action slots [6], [7], [17].

Q-learning is a straightforward and powerful algorithm for creating an action sheet for an agent. However, if this action sheet is too long (e.g., in an environment with 10,000 states and 1000 actions per state, wherein the size of the table reaches 10 million cells), then it becomes impossible to handle the vast number of Q-values. This results in two problems: First, the amount of memory required to save and update that table will increase. Second, it will require too much time to explore each state to create the necessary Q-table. To solve these problems, we approximated these Q-values using neural networks. This technique is known as deep Q-learning (DQL).

All the symbols used in Algorithm 1 and the DQL section are listed in Table 1.

In DQL, a deep neural network with weight θ is used to represent the action-value function. In deep Q-networks (DQNs) [19], the Q-value function and replay memory M, which stores the transition tuple, are first initialized. Then, the process repeats until convergence is achieved. In this process, first, we initialize $s_1 = \{x_1$ and $\phi_1 = \phi(s_1)$, where s_1 is the first state, x_1 is the first image, and ϕ_1 is the preprocessor. Subsequently, each step action a_t is selected. The selection depends on the probability value (if $\epsilon >$ probability then a_t select randomly, otherwise a_t is selected through

TABLE 1. Symbols used in Algorithm 1 and equation (1).

Symbols	Description
M	Replay memory used to store a replay of an episode
s_i	Show the state of an agent, *i.* show the number of states
x_t	Current state image
ϕ_1	Preprocessor
$\phi(s_1)$	Preprocessed first image
ϵ	Show the default greedy policy
a_t	The action was taken by the agent at time t
$max_a Q^*(\phi(s_t), a; \theta)$	Basic Q-learning
r_t	A reward after taking action at time t
x_{t+1}	Next state image
b	Batch size
$\mathcal{L}(\theta)$	Loss or squared TD error

Algorithm 1 Deep Q-learning

Require: Initialize replay memory M,
Initialize Q-value function with random weights
while (episode =1 to n) **do**
Initialize $s_1 = \{x_1$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
for (t=1 to T) **do**
if probability $< \epsilon$:
select a random action a_t
else:
select $a_t = max_a Q^*(\phi(s_t), a; \theta)$
end if
execute a_t in emulator and get $r_t and x_{t+1}$
Set: $s_{t+1} = s_t, x_{t+1} = x_t$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
Store: $\{\phi_t, a_t, r_t, \phi_{t+1}\}$ in M
Sample: random minibatch b of transitions $\{\phi_j, a_j, r_j, \phi_{j+1}\}$ from M
Set: $y_j = \begin{cases} r_j \text{for terminal} \phi_{j+1} \\ r_j + \gamma max_a Q^*(\phi_{j+1}, a'; \theta) \text{fornon} - \text{terminal} \phi_{j+1} \end{cases}$
Perform a gradient descent step on $[y_j - Q(\phi_j, a_j; \theta)]$ according to equation(1)
end for
end while

$max_a Q^*(\phi(s_t), a; \theta))$. After selecting a_t, we execute this action in the emulator and receive a reward r_t and the next state image x_{t+1}. We then set $x_{t+1} = x_t and s_{t+1} = s_t$ and store the transition tuple $\{\phi_t, a_t, r_t, \phi_{t+1}\}$ in M. Then, we generate a random sample batch of transitions $\{\phi_j, a_j, r_j, \phi_{j+1}\}$ from M and set the value of y_j according to ϕ_{t+1}. If ϕ_{t+1} is a terminal state, then $y_j = r_j$; otherwise $y_j = r_j + \gamma max_a Q^*(\phi_{j+1}, a'; \theta)$. Through y_j we want to minimize the squared TD error:

$$\mathcal{L}(\theta) = \sum_{j=1}^{b} \left[\left(y_j - Q(\phi_j, a_j; \theta) \right) \right] \tag{1}$$

The distributed and decentralized learning approach tackles problems such as the non-convergence of algorithms and the exponential growth of joint action tables that result from an increasing number of agents in a smooth manner. Instinctively, for scrutinizing policies for an MAS, direct learning of decentralized value functions or policies is preferred. IQL [5] educates self-directed action-value processes for individual agents using Q-learning [18]. Later, solutions to these kinds of task became more diverse [17], using deep neural networks through the induction of DQN [19]. A few other works [7], [20] have focused on the perseverance of learning to some extent; even then, extra state information cannot be considered during the training of learned decentralized value functions.

As expected, the centralized learning of collective actions can handle coordination problems and avoid non-stationarity. However, it is difficult to manage such centralized learning because the collective action space grows exponentially with increasing agent numbers. Classical approaches to ascendable centralized learning make use of coordination graphs [21], which use provisional independencies among agents by decomposing a combined reward function into a sum of agent-local terms. The sparse cooperative Q-learning algorithm [22] uses a tabular approach that learns to synchronize a group of cooperative agents only in certain states where such coordination is mandatory, encrypting these requirements in a coordination graph. These methods require the prior provision of dependencies among agents, although this prior knowledge is not required. Instead, it is assumed that every individual agent contributes towards the global reward, and at every state that agent becomes aware of its contribution.

Recent approaches for centralized learning require even more communication during execution, such as CommNet [23], which uses a centralized network architecture to exchange information between agents. Bic-Net [6] uses bidirectional RNNs to exchange information between agents in an actor–critic setting. This approach requires additional effort to estimate individual agent rewards.

Hybrid approaches exploit centralized learning with fully decentralized execution. COMA [9] uses centralized criteria to train decentralized actors, estimating a counterfactual advantage function for each agent to address the multiagent credit assignment. Similarly, Gupta *et al.* presented a centralized actor–critic algorithm with per-agent critics [24], which scales better than existing techniques for the same number of agents, but mitigates some of the advantages of centralization. In [25], the authors trained a centralized critic for each agent and applied it to competitive games with continuous action spaces. These approaches use on-policy gradient learning, which can suffer from low sample efficiency and are prone to getting stuck in suboptimal local minima.

Sunehag *et al.* [11] proposed value decomposition networks (VDN) as a solution these problems, which allowed centralized value-function learning to be accompanied by decentralized execution. Their algorithm decomposed a central state-action value function into a sum of the individual agent terms. This corresponds to the use of a degenerated, fully disconnected coordination graph. However, VDN does not use additional state information during training and can represent only a limited class of centralized action-value functions.

QMIX [13] is a modern approach which uses a centralized training with decentralized execution (CTDE) [35] method. In this technique, the factorization of the joint state-action value function for all agents is accomplished as a monotonic function by using a mixer network, which is denoted as $Q_{mix}(s_t, u_t)$. The mixer network is used to calculate the joint state-action value of all agents, and by using a monotonic function $\frac{Q_{tot}}{Q_a} \geq 0 \forall a \in \{a_1, \ldots, a_n$, it ensures the individual-global-max condition IMG [35] for each agent. The monotonic condition is achieved through a hyper-network, which predicts a strictly positive weight for the mixer network based on the current state of each agent as an input. Moreover, through this hyper-network, the outputs of the mixer network depend on the current state. The same DQN algorithm as in the optimization procedure is adopted and applied to $Q_{mix}(s_t, u_t)$. Furthermore, the joint action-value function class of QMIX is limited [35].

To address this limitation, QTRAN [37] introduced a novel factorization method to express the complete value function class with the help of IGM consistency. However, this method ensured more general factorization than QMIX but required an inconvenient amount of computational power to implement. Two extra soft regularizations were required for its approximate version, but it still performs below par in complex domains with online data collection [34].

Mahajan *et al.* [34] demonstrated that QMIX has limited exploration ability in certain environments. They proposed a model in which there is a latent space to enhance the performance of all agents. Therefore, for supportive MARL, achieving effective scalability remains an open challenge that is addressed by QPLEX [35]. For both joint and individual action-value functions, QPLEX introduced a dueling structure which then deformalized the IGM principle via advantage-based IGM. This demonstrates the ability of QPLEX to support offline training with high stability. However, although QPLEX performs well, it still requires complex networks to achieve these results. Moreover, it requires

Algorithm 2 Grey Wolf Optimizer (GWO)

Require: Initialize agents $G_i(i = 1, 2, \ldots, n)$, number of iteration K
Initialize a, A, and C
Ensure: calculate the fitness of agents and set alpha, beta, and delta according to fitness
while (iteration = 1 to K) **do**
for (agents = 1 to n) **do**
update position agent using equation (6)
end for
update a, A, and C
compute fitness of agents
update alpha, beta, and delta
end while
return value of alpha

numerous training episodes for a large number of agents, as it uses a greedy policy for the action selection of an individual agent.

In addition, researchers have introduced two new Deep Quality-Value (DQV)-based MARL algorithms known as QVMix and QVMix-Max [36]. These algorithms are established using centralized training with decentralized execution. The results from these algorithms show that overall, QVMix performed better than the other algorithms because it is less susceptible to an overestimation bias of the Q function [36]. However, QVMix also requires high computational power and large amounts of training time because it also uses a greedy policy for the action selection of each individual agent.

Therefore, in this paper, we propose a novel, nature-inspired optimization-based hybrid policy to address these limitations. In this policy, we used GWO along with a greedy policy for the action selection of each individual agent. Optimization algorithms, such as GWO (normally used for finding the prey) or Ant Colony Optimizer (normally used for finding the shortest path), require environmental information, but they perform better than the greedy policy. In GWO, agents trained in a centralized manner, wherein the leader agent helps the other agents [29]. More detail on this topic is provided in Section III (A). Moreover, to gather information about the environment, in the beginning, action-selection is performed through greedy policy. A policy is then selected with the help of maximum reward and the learning rate alpha.

III. GREY WOLF OPTIMIZATION (GWO)

A. INTRODUCTION AND WORKING PARADIGM

The fundamental component of the GWO that makes it more successful than other well-known swarm intelligence algorithms is its hierarchical chain. The governance chain of importance is framed by a specific objective function known as the goal. Thus, the objective function is arranged into cost capacity, estimated cost, and the fitness function, which are utilized to assess the precision of the outcome, as compared with the prearranged structure arrangement [28], [29].

The wolf pack is partitioned into four prevailing positions, as shown in Figure 1. Alpha, beta, and delta wolves compose the main groups. The omega wolves do not reserve any options to settle on choices in a swarm, even though their

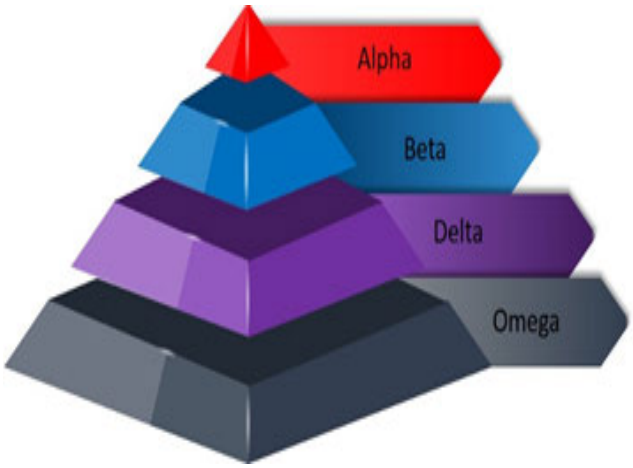


FIGURE 1. Social Hierarchy of GWO.

TABLE 2. Symbols and abbreviations used in Algorithm 2 and GWO.

Symbol	Description
$\vec{D}$	Encircling the Pray
$\vec{C}$	coefficients vectors help to encircling and hunting the prey
$\overrightarrow{X_P}$	The position vector of prey, show the position of prey
$\vec{X}(t)$	The position vector of a wolf shows the current position of a wolf
$\vec{X}(t+1)$	The position vector of a wolf shows the updated position of a wolf
$\vec{A}$	coefficients vectors , through its value wolf attack on prey
X_α , X_β , X_δ	Position vector for an alpha, beta, and delta wolves

presence decides the swarm intelligence. The main purpose of the social chain of command is to lead wolves to the prey's location, and they manage omegas to play out the pursuit. Different operators, such as social hierarchy, encircling the prey, hunting, attacking the prey (exploitation), and pursuing the prey (exploration) mimic the association of cumulative behaviors in a wolf pack.

Table 2 presents the symbols used in the equations. A detailed description of GWO is provided in the subsequent sections.

Encircling Prey

In GWO, grey wolves encircle the prey to examine two points in space and amend the location of one of them to correspond to the other. The following formulas represent the grey wolf encircling methodology:

$$\vec{D} = \left|\vec{C} \cdot \overrightarrow{X_P}(t) - \vec{X}(t)\right|,$$
$$\vec{X}(t+1) = \overrightarrow{X_P}(t) - \vec{A} \cdot \vec{D}, \quad (2)$$

Where t indicates the current iteration, $\vec{X}$ indicates the position of a grey wolf, and $\overrightarrow{X_P}$ is the position vector of the prey. $\vec{A}$ and $\vec{C}$ are coefficient vectors, which can be calculated as

$$\vec{A} = 2\vec{a} \cdot \vec{r_1} - \vec{a},$$

$$\overrightarrow{C} = 2 \cdot \overrightarrow{r_2}, \tag{3}$$

Where the components of $\overrightarrow{a}$ are reduced from 2 to 0 across iterations, and $\overrightarrow{r_1}$ and $\overrightarrow{r_2}$ are randomly created vectors in [0, 1].

Hunting

The hunting scheme of the grey wolves can be mathematically modeled as them approaching the location of the prey with the assistance of alpha, beta, and delta wolf information. Figure 2 shows the estimated updated position of agents in the GWO based on this information. The positions held by the alpha, beta, and delta wolves $X_\alpha, X_\beta, X_\delta$ are calculated as in the following equations: Omegas take action according to Equation (6):

$$\begin{aligned}
\overrightarrow{D_\alpha} &= \left|\overrightarrow{C_1} \cdot \overrightarrow{X_\alpha} - \overrightarrow{X}\right|, \\
\overrightarrow{D_\beta} &= \left|\overrightarrow{C_2} \cdot \overrightarrow{X_\beta} - \overrightarrow{X}\right|, \\
\overrightarrow{D_\delta} &= \left|\overrightarrow{C_3} \cdot \overrightarrow{X_\delta} - \overrightarrow{X}\right|,
\end{aligned} \tag{4}$$

$$\begin{aligned}
\overrightarrow{X_1} &= \overrightarrow{X_\alpha} - \overrightarrow{A_1} \cdot (\overrightarrow{D_\alpha}), \\
\overrightarrow{X_2} &= \overrightarrow{X_\beta} - \overrightarrow{A_2} \cdot (\overrightarrow{D_\beta}), \\
\overrightarrow{X_3} &= \overrightarrow{X_\delta} - \overrightarrow{A_3} \cdot (\overrightarrow{D_\delta}),
\end{aligned} \tag{5}$$

$$\overrightarrow{X}\,(t+1) = \frac{\overrightarrow{X_1}\,(t) + \overrightarrow{X_2}\,(t) + \overrightarrow{X_3}\,(t)}{3} \tag{6}$$

Attacking Prey

Advancing toward the prey requires the wolf to minimize the value of $\overrightarrow{a}$. The variation scope of $\overrightarrow{A}$ is also reduced by $\overrightarrow{a}$. $\overrightarrow{A}$ is a random value in the range of $[-a, a]$, where $\overrightarrow{a}$ is reduced from 2 to 0 across the iterations. If $|A| < 1$, then the wolves are forced to attack the prey; otherwise, they shift toward exploration. The changeover between exploration and exploitation is created by the changing values of $\overrightarrow{a}$ and $\overrightarrow{A}$. Algorithm 2 describes the grey wolf optimization pseudo-code.

GWO first initializes the number of agents/wolves and the values of a, A, and C (where ''a'' is a vector and its value declines from 2 to 0. ''A'' and ''C'' are coefficients, and the agent's exploration or exploitation behavior depends on the value of A. Subsequently, we calculate the fitness of agents, and according to the fitness level, we select alpha, beta, and delta wolves. Then, the position of all agents is updated according to Equation (6). We repeat all steps until the episode ends. Generally, after completing one execution, we are able to attain the value of alpha.

B. OPERATIONAL EXAMPLE

A multiagent system (MAS) is a sub-discipline of distributed artificial intelligence (DAI). It is a combination of comparatively independent parts, known as agents. In an environment, these agents are designed to act as experts, and they have their own actions and behaviors in that specific area. The focus of research in MAS has been to make agents which work without human interaction. According to [33], the most suitable example of MAS is the Internet, wherein millions of computers run independently but can communicate with each other.

In real-life scenarios, humans often work with each other towards a single goal. They achieve their goals more quickly through communication and shared attention. Similarly, agents can achieve a goal with fewer iterations through communication in the MAS. For scenarios involving fewer agents, this is not a significant issue.

To understand the advantages of the proposed algorithm (policy), let us consider a straightforward example of an MAS. In this scenario, there are several robots. These robots are used to explore the entire area of a building or other environment. Investigation of an unknown region starts with no prior information about the obstacles and the design of the territory. In this example, we can compare the performance of GWO with that of a distributed RL. As distributed RL is the backbone of QMIX/QVMix; however, in the case of the proposed policy, GWO plays a vital role. In addition, this example shows how GWO leverages the advantages of centralized learning without the communication constraints and propensity for stuck agents.

In Figure 3, the red color represents the current position of the agents. Yellow represents cells which have not been visited by any agent but for which some agents know the reward values of those cells. White represents the explored cells, and grey represents the unvisited, unknown cells.

Figure 3(a) shows the multiagent exploration using a simple decentralized epsilon ϵ-greedy policy. Agent-1 and agent-3 are near the unexplored area, whereas the other two agents are far from the unexplored area. Agent-2 and agent-4 are stuck because the reward will be the same for all possible next states. After a specific time period, the total explored area will be very low because the maximum exploration is performed using two agents. Although agent-1 and agent-3 are closer to the unexplored area, they cannot help the other agents because of the decentralization constraints. Similarly, in the case of a centralized ϵ-greedy policy, as the number of agents increases, the length of the centralized Q-table increases exponentially; hence, these large table values require very high computational power.

Figure 3(b) and 3(c) show the exploration performed using the GWO. In Figure 3(b), agent-2 has a maximum number of unvisited neighbors; therefore, it becomes the alpha. Similarly, agent-1 has three unvisited neighbors, meaning it acts as a beta. Agent -3 is the delta, and agent-4 is the omega. The next action will be taken with the help of the current combined alpha, beta, and delta information. As the omega (agent-4) has no unvisited neighbor, it has the same reward for all the next possible states. This agent can become stuck, but other agents will help the omega to take the correct next step in the case of GWO. In Figure 3 (c), Agent-1 has a maximum number of unvisited neighbors, and it becomes the alpha at that stage. Similarly, agent-2 and agent-4 become beta, whereas agent-3 becomes delta. Meanwhile, no omega is available at this stage. All the agents take actions according to either the alpha (agent-1) order or the best possible reward.

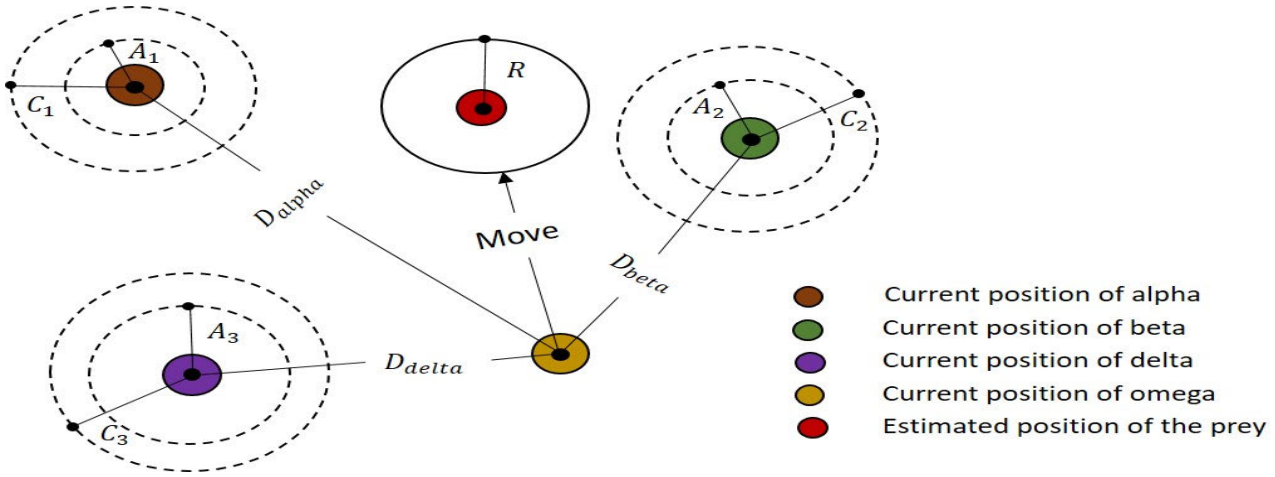


FIGURE 2. Updating the position of agents in GWO.

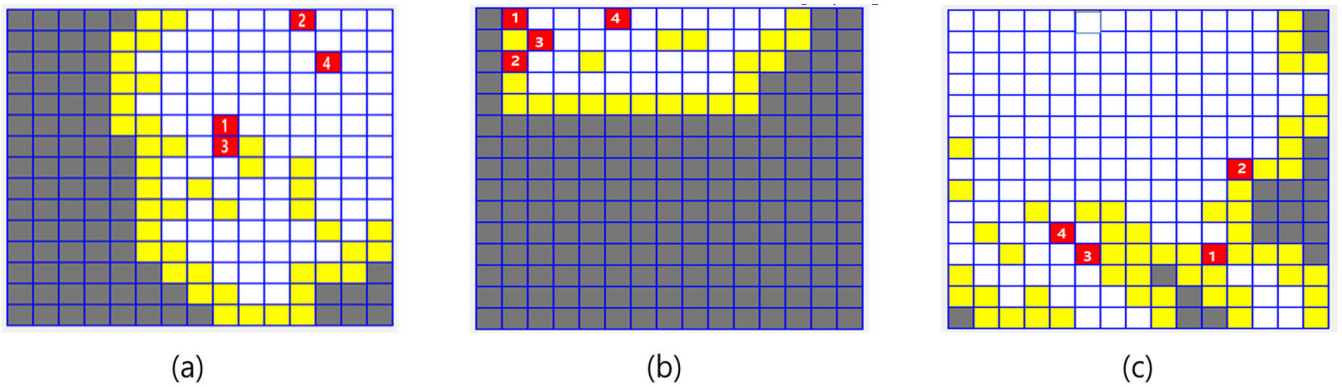


FIGURE 3. (a) multiagent exploration using $\epsilon - greedy$ decentralized policy after n iterations, (b) multiagent exploration using GWO from the beginning, (c) multiagent exploration using GWO after n iterations.

It is evident from Figure 3(a) and 3(c) that if some agents are stuck, then after the same number of episodes, GWO performs significantly better than the ϵ-greedy policy. The reason behind this is that in the case of ϵ-greedy policy, it is difficult to return a stuck agent to an operational state. However, in the case of GWO, a stuck agent returns to the operational state very quickly with the help of other agents, especially the alpha. Moreover, we used GWO along with ϵ-greedy policy to boost up the training process. Although, in some scenarios, GWO performs better than the decentralized ϵ-greedy policy, when the environment changes the performance of GWO decreases gradually. Therefore, we used the GWO policy along with decentralized ϵ-greedy in our proposed algorithm.

IV. Q-VALUE SELECTION USING OPTIMIZATION AND DRL (QSOD)

In most MADRL algorithms, the focus is on upgrading the joint action-value function using different weights [7], [13]. For individual action selection, agents usually use simple Q-learning [30], [31], and attention between agents cannot be developed appropriately. They require a large number of training sessions to deeply learn the environment. Furthermore, in most scenarios, many algorithms fail to address nonstationary problems, such as IQL [5], [16]. Hence, there is no guarantee of system convergence. Similarly, in the case of optimizers, there are many limitations, such as the failure of the algorithm due to environmental change.

We propose a novel technique for the optimization of a deep reinforcement-learning multiagent based on action-value selection using optimization and DRL (QSOD). In the proposed technique, learning takes place both ways either communicatively or non-communicatively, according to the situation using optimization and a greedy policy. Therefore, the benefits of centralized learning can be leveraged without experiencing communication constraints through optimization. Consequently, this technique saves computational power and offers extraordinary performance improvements, even in cases involving numerous agents. This ensures that convergence occurs faster, unlike in traditional DRL algorithms.

A group of recurrent neural networks (RNNs) for calculating the Q action values for the next state is present in QSOD (as shown in Figure 4(c)). Individual agents use individual RNNs [13]. In typical scenarios, the next action selection is performed using the $\epsilon - greedy$ policy [30], [31]. However, in the proposed algorithm, we selected actions using two different policies. The first policy uses the traditional $\epsilon - greedy$ policy for the action selection of agents. In the second policy, action selection takes place according to the GWO. As a result, within the first policy learning takes place in a decentralized or distributed fashion [5], [31], [32], wherein each agent selects action independently. The second policy is based on the last episode reward, wherein we select an agent as leader of the group to be the same as the alpha wolf in GWO [15]. This agent performs the $\epsilon - greedy$ policy for action selection. All other agents then select actions according to the Q-value of the leader agent (like GWO) [15]. The key to this policy is that if some agent moves far away from the other agents based on the leader agent's Q-value, that agent will easily return and join the team again to achieve the goal. If all agents perform in a centralized manner, then the agents' combined power increases. Consequently, the agents achieve a higher reward. In this case, learning is performed in a centralized manner, and attention is properly developed between agents, such as in GWO [5].

Changes in the environment result in optimization failures because for different environments, optimizers need human input. To address this problem, we used both policies and set two conditions to select a policy in each episode. According to the first condition, optimization-based selection started when the agents' accumulative reward in the previous episode was greater than or equal to the threshold for reward ($R_{t-1} \geq R_{th}$) [28]. The value of the threshold reward R_{th} is calculated using Equation (7).

$$R_{th} = \frac{\alpha R_{max}}{2} \tag{7}$$

where $\boldsymbol{R_{th}}$ is the threshold reward, $\boldsymbol{\alpha}$ is the learning rate, and $\boldsymbol{R_{max}}$ is the maximum reward or target reward.

A maximum or targeted reward can only be achieved if all goals are achieved, and at least one agent is still alive because the learning rate α affects the convergence time. For comparatively lower values of α, the training requires a long duration. Therefore, for the proposed algorithm, when a low-value α is used, the optimization policy starts with a lower threshold reward value, and in the case of a high value α, it starts with a high threshold reward value. As a result, in all scenarios, the optimization policy starts after almost the same number of episodes in both cases.

The second condition is that if the current episode's reward is equal to the reward of the previous episode, then the traditional action selection policy ($\epsilon - greedy$) will be used in the next episode. The main reason for this is that the agents explore the entire area while using the optimization policy. After achieving the maximum reward (optimal path), agents always follow the same path, and thus learning stops. However, the likelihood that the environment will change in some subsequent episodes is significant, but the agents will continue follow the same previously optimal path and thereby perform poorly. Furthermore, in some scenarios, the agent can take the selected action because of some unrealistic reward or next state. Thus, in all such cases the optimization policy either becomes stuck or chooses some suboptimal action, which causes the game to stop and execution to be interrupted. This second condition helps to address these challenges. Moreover, this problem is a major limitation for the optimizer.

In addition, using optimization-based action selection for all agents creates a problem in that the learning may be ceased during that episode. During each time step, agents select the action according to the optimization value. The optimizer is used for the maximization function.

They do not use the hit and trial method, as in the case of optimizers. The concept of punishment is not used, which is the basis for reinforcement learning (RL). Therefore, an RL-based optimizer policy for action selection is not possible. We use the $\in -greedy$ policy to select actions for the leader agent. The leader agent's Q-value will help the other agents to select actions [28]. All the agents choose action in the optimizer policy according to the GWO, except the leader agent.

There is a mixer network which functions after calculating the individual Q_a values of all agents. A feed-forward neural network is used as a mixer network. It is used to calculate the combined Q value of agents Q_{tot}. It takes agent network outputs as input, and by mixing that input monotonically, produces values of Q_{tot}. This is the sum of the individual value function Q_a of the agents, as shown in Figure 4(a). The monotonicity in the network is enforced in Equation (8), which shows the relationship between Q_{tot} and Q_a;

$$\frac{Q_{tot}}{Q_a} \geq 0, \quad \forall a \in \{1, 2, \ldots, N\} \tag{8}$$

Separate hyper-networks are used to provide weights for the mixer network, as shown in Figure 4(b). They produce weights by using the current state as an input. These networks are based on single-layer networks, followed by an absolute activation function, ensuring that the weights are positive. A vector-type output is produced by these networks and reshaped later in the matrix. A two-layer hyper-network is used to produce final the bias.

Using the state in hyper-network instead of a mixer network ensures the monotonicity of the system. QSOD is trained in an end-to-end fashion to minimize the overall loss $\mathcal{L}(\theta)$ as shown in Equation (9):

$$\mathcal{L}(\theta) = \sum_{j=1}^{b} \left[\left(y_j^{tot} - Q_{tot}(\tau, u, s : \theta) \right) \right] \tag{9}$$

where b is the batch size, and θ^- and y_j^{tot} are the target network parameters, as in DQN. If the next state is a nonterminal state $y_j^{tot} = r_j + \gamma max_a Q_{tot}\left(\tau', u', s' : \theta^-\right)$.

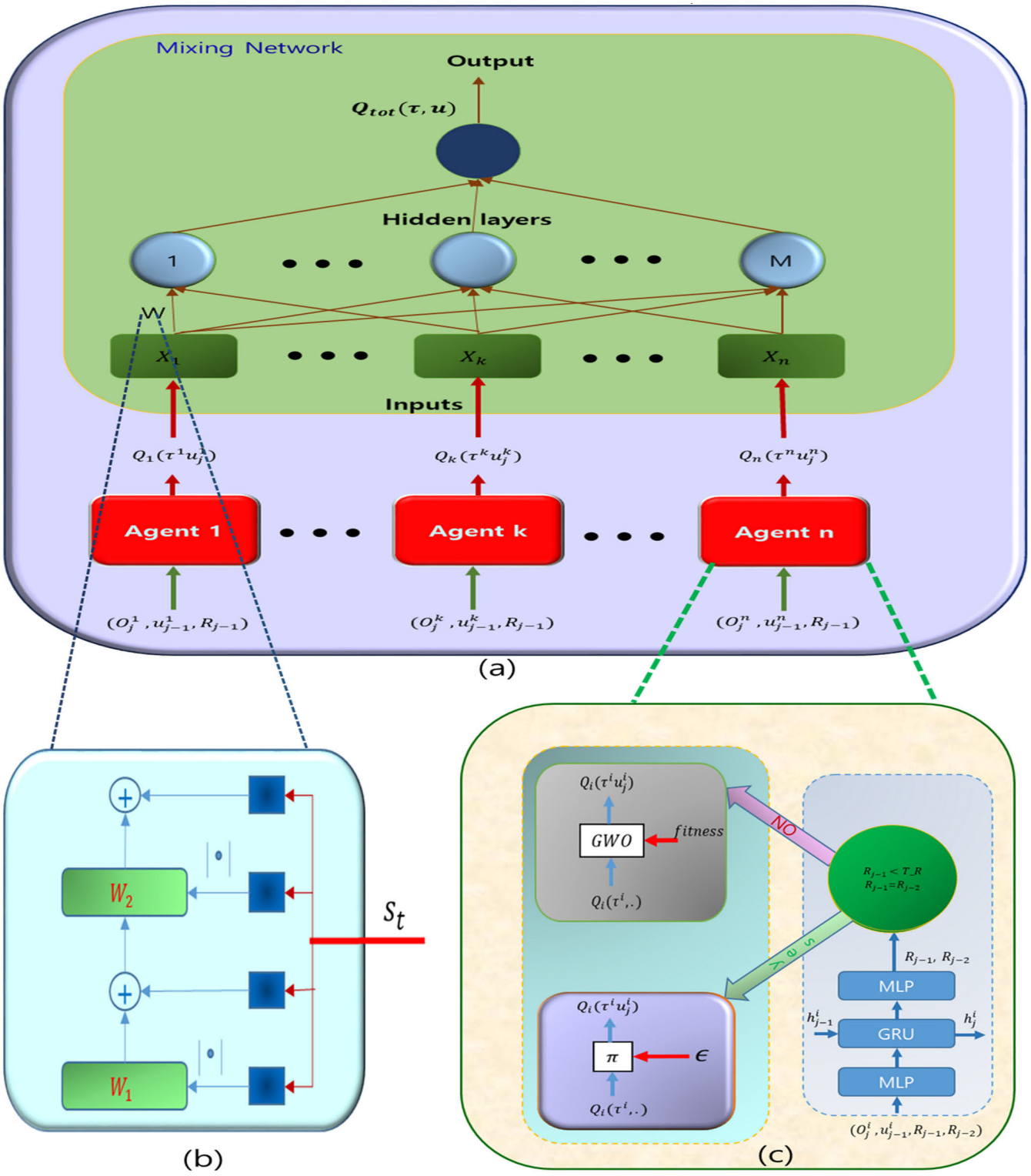


FIGURE 4. (a) Mixer network architecture. (b) Hypernetworks providing weights to the mixer network. (c) Agent network demonstrating how to select the policy according to the environmental conditions. Green represents the optimization policy and purple represents the ϵ −*greedy* policy.

All symbols used in Algorithm 3 and Equation (9) are listed in Table 3.

Algorithm 3 presents the steps of the proposed method. First, in Lines 1 and 2, the required parameters are initialized, including the number of agents, the reward for each action, replay memory to store the current preprocessed sequence, values of alpha and gamma, reward for a win, number of episodes, and number of steps in each episode. After that

TABLE 3. Symbols used in Algorithm 3 and equation (9).

Symbols	Description
Q_{tot}	Combined Q-value function
Q_a	Q-value function for each agent a
M	Replay memory to store batch
α	Learning rate
γ	Discounted rate
R_{max}	Maximum reward that can be achieved in a single episode
s_t	Current state
x_t	Current state image
τ	Current preprocessed sequenced
R_t	The reward for the current episode
R_{t-1}	Last episode reward
ϵ	$\in -greedy$ policy
a_t	The action taken at the current state
τ'	Next preprocessed sequence
$\mathcal{L}(\theta)$	squared TD error
b	batch
y_j^{tot}	parameters of a target network
θ	Weight of target network

in Line 3, the two main conditions are imposed. The first condition is related to the threshold reward, through which the optimization policy is activated. The second condition is related to monotonicity. In Line 4, a repeated process is started, through which we run our algorithm to a required number of times/episodes. In Line 5, before starting each episode, the first state and the first preprocessed sequence are initialized. The first state refers to the initial position of the agents. In Lines 6 and 23, a condition is imposed for activating either policy (GWO or Greedy). For Greedy, this policy is selected if the previous episode's reward is less than the threshold reward or if the previous two consecutive episodes had identical rewards; otherwise the optimization policy is selected. In Lines 7 and 24, a loop is initialized for the number of steps for each episode. In Line 7, a loop is used for the greedy policy, whereas in Line 24 a loop is used for the optimization policy. The Line 8 loop is used to compute the Q-values of all agents. From Lines 9 to 14, simple Q-learning is performed. From Lines 16 to 21, the steps for simple deep Q-learning are performed, except Line 17, in which the accumulative Q-value is computed through the mixer network. On Line 23, if the previous episode's reward was greater than or equal to the threshold reward and the previous two consecutive episodes had different rewards, then the optimization policy is triggered. In Line 25, the fitness level of the agents is calculated, and an alpha or leader agent is selected. From Lines 26 to 29, a loop is used to compute the Q-value for each agent through optimization policy. In the loop from line 8 to 15 an action is selected from the available actions list through IQL. In the loop from Lines 26 to 29, an action is selected from the available actions list through GWO.

After calculating the individual Q_a, the combined Q-value function Q_{tot} is calculated. Then, a random batch b of transitions $\{\tau, a_j, r_j, \tau'\}$ is sampled from M and the value of y_j^{tot} is set according to τ'. If τ' is a terminal state, then $y_j^{tot} = r_j$; otherwise $y_j = r_j + \gamma max_a Q_{tot}\left(\tau', u', s' : \theta^-\right)$.

Algorithm 3 Attention-based hybrid policy for deep MARL (QSOD)

1 Require: Initialize agent, reward R, Replay memory M, Q_{tot}, $\boldsymbol{\alpha}$, $\boldsymbol{\gamma}$,
2 Target reward R_{max}, episode, number of step in each episodes t
3 Ensure: $R_{th} = \frac{\alpha R_{max}}{2}$ **and** $\frac{Q_{tot}}{Q_a} \geq 0$
4 while (episode = 1 to n) **do**
5 Initialize $s_1 = \{x_1$ and preprocessed sequenced $\tau_1 = \tau(s_1)$
6 **if** $(R_{th} < R_{t-1} or R_{t-2} = R_{t-1})$ **then**
7 **for(t=1 to T)do**
8 **for** (agents = 1 to n) **do**
9 **if** probability $< \epsilon$ **then**
10 select a random action a_t
11 **else**
12 select $a_t = max_a Q^*(\tau(s_t), a; \theta)$
13 **end if**
14 **Compute** Q_a according to the a_t
15 **end for**
16 **Set:** $s_{t+1} = s_t, x_{t+1} = x_t$ and preprocess $\tau' = \tau(s_{t+1})$
17 **compute** Q_{tot} using a mixer network
18 **Store:** $\{\tau, a_t, r_t, \tau'\}$ in M
19 **Sample**: random batch b of transitions $\{\tau, a_j, r_j, \tau'\}$ from M
20 **Set:** $y_j^{tot} = \begin{cases} r_j \text{ for terminal } \tau' \\ r_j + \gamma max_a Q_{tot}\left(\tau', u', s' : \theta^-\right) \text{ for non} - \text{terminal } \tau' \end{cases}$
21 Perform a gradient descent step on $\left(y_j^{tot} - Q_{tot}(\tau, u, s : \theta)\right)$ according to Equation(9)
22 **end for**
23 **else**
24 **for(t = 1 to T)do**
25 **Calculate** the fitness of agents and set leader agent according to fitness
26 **for** (agent = 1 to n) **do**
27 **Compute** a_t for all agents using Equation (6) except leader agent
28 **Compute** Q_a according to a_t
29 **end for**
30 **Set:** $s_{t+1} = s_t, x_{t+1} = x_t$ and preprocess $\tau' = \tau(s_{t+1})$
31 **compute** Q_{tot} using a mixer network
32 **Store:** $\{\tau, a_t, r_t, \tau'\}$ in M
33 **Sample**: random batch b of transitions $\{\tau, a_j, r_j, \tau'\}$ from M
34 **Set:** $y_j^{tot} = \begin{cases} r_j \text{ for terminal } \tau' \\ r_j + \gamma max_a Q_{tot}\left(\tau', u', s' : \theta^-\right) \text{ for non} - \text{terminal } \tau' \end{cases}$
35 Perform a gradient descent step on $\left(y_j^{tot} - Q_{tot}(\tau, u, s : \theta)\right)$ according to Equation(9)
36 **end for**
37 **end if**
38 **end while**

Algorithm 3-A Attention-based hybrid policy for deep MARL

1 Require: Initialize agent, reward R, Replay memory M, Q_{tot}, $\boldsymbol{\alpha}$, $\boldsymbol{\gamma}$,
2 Target reward R_{max}, episode, number of step in each episodes t
3 Ensure: $R_{th} = \frac{\alpha R_{max}}{2}$ **and** $\frac{Q_{tot}}{Q_a} \geq 0$
4 while (episode = 1 to n) **do**
5 Initialize $s_1 = \{x_1\}$ and preprocessed sequenced $\tau_1 = \tau(s_1)$
6 **if** $(R_{th} < R_{t-1} or R_{t-2} = R_{t-1})$ **then**
7 for each agent select **action** by using **greedy** policy
8 **else**
9 for each agent select **action** by using **optimization** policy
10 **perform** steps of **MARL**
11 **end if**
12 end while

Through y_j^{tot} we aim to minimize the squared TD error, as shown in Equation (9).

Here, Algorithm 3-A provides a summary of proposed Algorithm 3 (QSOD). It includes only the most important steps of the proposed algorithms because the proposed algorithm (Algorithm 3) has too many steps to be concise.

FIGURE 5. StarCraft II Environment.

Further, it includes a detailed overview of the QSOD, whereas Algorithm 3-A includes a brief overview of the proposed methodology.

V. PERFORMANCE EVALUATION

A. STARCRAFT II

StarCraft II is the sequel to the first StarCraft game. Both StarCraft and StarCraft II are the registered trademark of Blizzard. Both are real-time strategy (RTS) games. During the last six to seven years, RTS games have becomes popular in the DRL field because many researchers have tested their work on RTSs. StarCraft in particular provides a powerful platform to address competitive and collaborative multiagent problems. Many complicated micro-action sets are available in StarCraft. Through these sets, StarCraft allows for the learning of complex collaboration among cooperative agents. [2], [4], [16] performed their experiments using StarCraft. In the present study, we used the StarCraft II Learning Environment (SC2LE) [11], as in QMIX and QVMix. SC2LE is based on the second edition of StarCraft. It provides many different scenarios, and it has better support from developers than the original StarCraft. Figure 5 shows the environment of StarCraft II.

Like QMIX, we chose the decentralized micromanagement problem in StarCraft II. In fighting scenarios, there are two groups of agents available on the map. The first group consists of allied agents. These agents are controlled through the proposed method. The second group consists of enemy agents. These agents are controlled by the built-in AI of the game. The initial position of both groups' agents changes with each episode. All the other setting are similar to that of QMIX [18].

B. EXPERIMENTAL RESULTS

We used the SC2LE and StarCraft Multi-agent Challenge (SMAC) environment to evaluate the performance of our proposed method. The difficulty level of the game was set to medium. We used different scenarios to compare our algorithm with the state-of-the-art algorithms QVMix and QMIX. These scenarios included 3m, 8m, 2s_3z, MMM, and 1c_3s_z. The letters c, m, s, and z are described in Table 4. A vector consisting of the features of the agents is known as the global state. It contains the health, shield, cool-down, and last taken action information. The following different actions were available in each agent's action space: Move (performed in the East, West, South, or North direction), Attack (only performed if the enemy was within range), Stop (performed when the episode ended), and Noop (performed if the next state reward was unrealistic). After every time step, agents received a combined reward. It was calculated through the total damage of the enemies, similar to [18].

TABLE 4. Agent details.

Name	Shooting range (A)	Shield point (C)	Hit Points
Marines (m)	6	N/A	45
Stalkers (s)	6	80	80
Zealots (z)	6	50	100
Colossi (c)	6	150	200

To compare the performance of QVMix, QMIX, and the proposed algorithm, we adopted fewer episodes than [13] and [QVMix]. For each simulation of both methods, we suspended training after every 100 episodes and ran 20 independent episodes, with each agent performing greedy decentralized action selection in the case of QMIX and QVMix or hybrid optimization action selection in the case of the proposed algorithm. We ran a total of 10,000 episodes for training and used a 500 replay buffer size. After every 200 episodes, the target network was updated. For QMIX and our proposed algorithms, we used the normal computational power system for performing the experiments: (GPU = GTX1050, CPU = i7-8750H, RAM = 16GB). For QVMix we used a system with higher computational power (GPU = RTX2080, CPU = i7-8700K, RAM = 32GB).

To highlight the significance of our algorithm, we calculated two different types of results: win rate and average loss. The win rate was calculated as the percentage of episodes in which agents killed all enemies within a given time. The average loss was calculated as the percentage loss across the 100 episodes. All results were calculated for an average of five runs for each algorithm.

Figure 6 shows the rolling average of the win rates of QSOD, QVMix, and QMIX for five different scenarios (8-marines, 3-marines, MMM, 2-stalkers with 3 zealots, and 1-stalker with 2-colossi and 3-zealots). Initially, almost all algorithms performed similarly. However, over time, the proposed QSOD algorithm performed better than QVMix and QMIX, particularly after 4,000 episodes. Moreover, in scenarios with high-power agents, such as MMM, the proposed algorithm showed better performance than QVMix and QMIX. Moreover, in QVMix and QMIX, action selection was conducted through the greedy policy for each agent. Therefore, these algorithms required a higher number of episodes in training for these challenging scenarios.

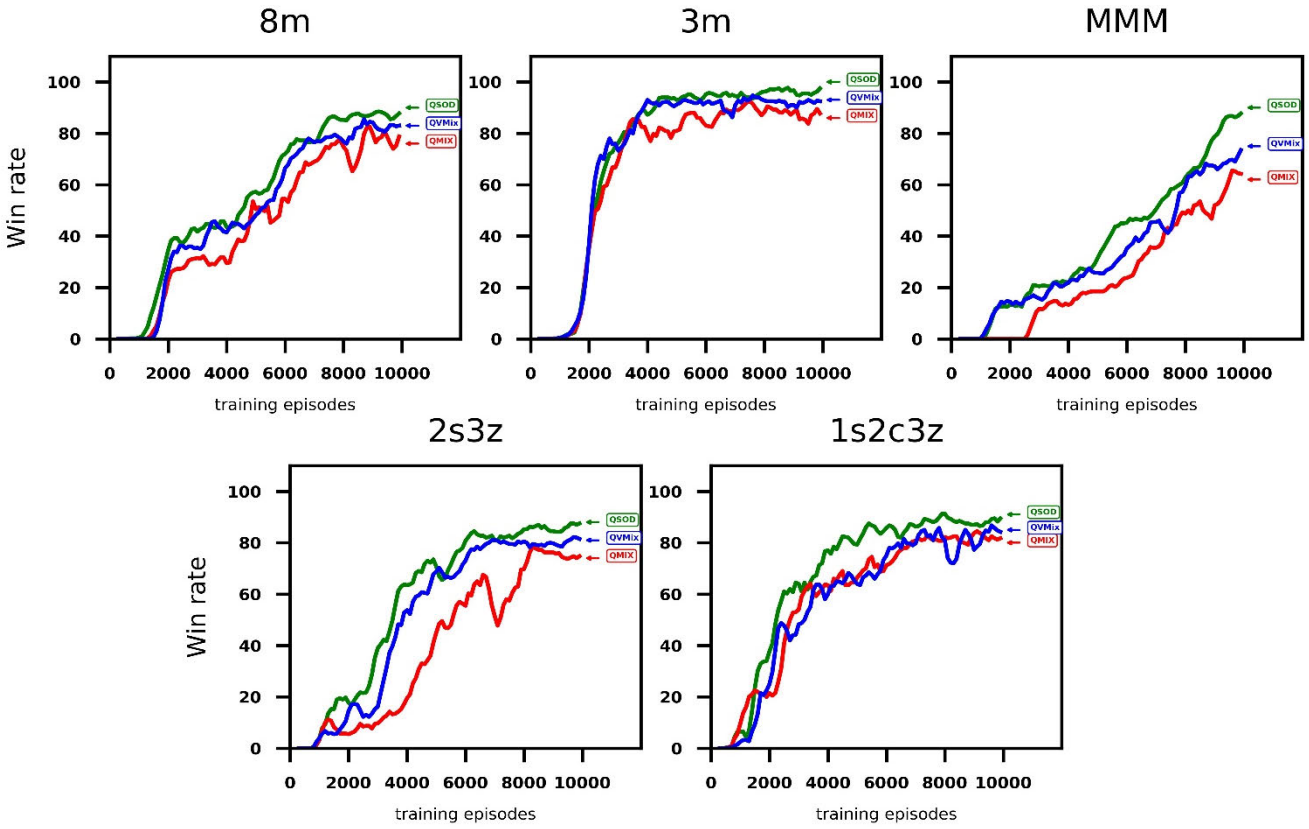


FIGURE 6. Win rates during training for the proposed algorithm, QVMix, and QMIX across five different scenarios.

TABLE 5. Win rate final values.

Scenario	Proposed	QVMix	QMIX
8m	88	83	78
3m	98	93	89
MMM	88	74	64
2s3z	88	82	75
1s2c3z	90	84	82

TABLE 6. Time required by QMIX, QVMix, and the proposed QSOD for different scenarios.

Agents-Map	QMIX (Time(s))	QVMix (Time(s))	Proposed (Time(s))
3m	9190	8920	8690
8m	12334	13834	11734
2s_3z	12402	13962	11142
MMM	17943	17400	16432
1c_3s_5z	29153	31193	26993
3m (50000 episodes)	29491	N/A	27797

However, in the case of the proposed algorithm, a hybrid optimization policy was used, which required much fewer training episodes.

Table 5 shows the maximum win rate value for all three algorithms across five scenarios after 10,000 episodes. Therefore, in all scenarios, our proposed algorithm performed better than the state-of-the-art algorithms QMIX and QVMix. Particularly, in the MMM scenario, our proposed algorithm achieved at least a 12% higher win rate than both of the other techniques. Similarly, in the case of 8m and 3m, our algorithm achieved a 5% greater win rate than QMIX and QVMix.

Furthermore, in the 1-colossi, 3-stalkers, and 5-zealots scenarios, throughout almost all of the training episodes, QSOD performed better than both QVMix and QMIX.

The average training loss is shown in Figure 7. Across all scenarios, the proposed QSOD had the lowest training loss compared to QMIX and QVMix. Notably, in high-level-agent scenarios, QSOD outperformed QMIX and QVMix. This is because the greedy policy wastes time searching the environment repeatedly with multiple agents. However, in the proposed optimization policy, if a leader searched a point in the environment, it shared its experience with the other agents. As a result, fewer episodes were required for training. Moreover, the lowest value of training loss was achieved.

Figure 8 illustrates the most important result, which is the convergence graph. In the case of RL, convergence plays a crucial role in validating the significance of any algorithm. Figure 8 shows the results of the 3-marines scenario

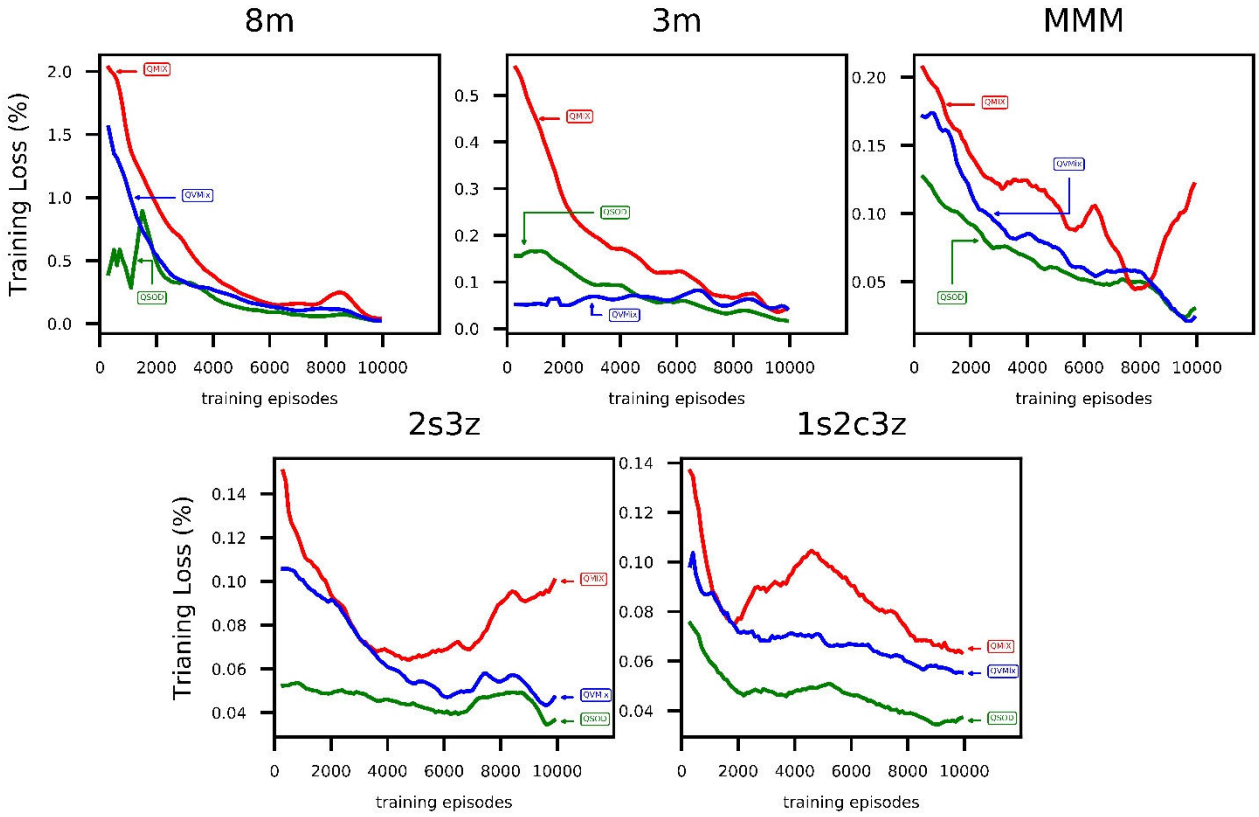


FIGURE 7. Average loss during training for the proposed algorithm (QSOD), QVMix, and QMIX on five different agent maps.

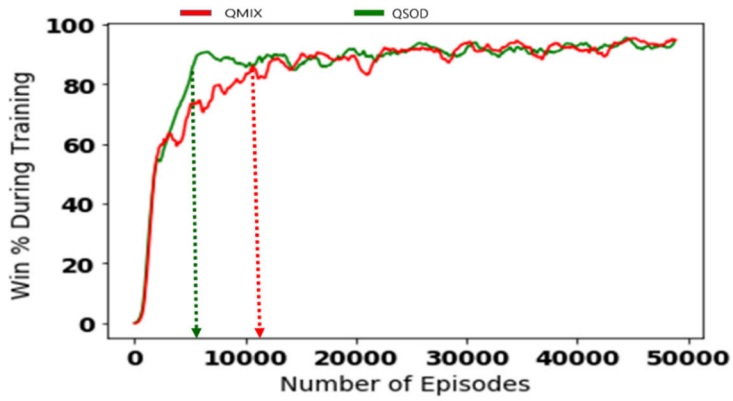


FIGURE 8. Convergence of win rate rolling average in 3m.

for QMIX and the proposed algorithm. According to the figure, convergence occurred after several episodes in both cases. However, the proposed policy's convergence started approximately 2,500 episodes earlier than that for the default QMIX. This was due to the hybrid optimization policy and the conditions for the activation of either policy. Moreover, these conditions played a vital role in controlling the abrupt changes in the results, as shown in Figure 7, and the proposed scheme's curve is smoother than that of the default QMIX. In Figure 8, we present a comparison of our proposed algorithm with only QMIX to demonstrate the importance of our proposed policy over the default greedy policy while using the same network.

Table 6 compares the time efficiency of the proposed scheme with both state-of-the-art algorithms (QMIX and QVMix) for each run. In all cases, the proposed policy required less time than the default QMIX. In particular, in the case of difficult scenarios, the proposed QSOD saved more than 2000 s for 10,000 episodes. Furthermore, in the case of 3m, 50,000 episodes for the proposed algorithm saved

1700 s. Therefore, if either the number of episodes increased or the number of agents increased, the proposed algorithm outperformed the default QMIX policy. After finding the optimal path, the optimization policy required fewer steps to win a game, which saves time in each episode and explains this result.

VI. CONCLUSION

This paper proposed a novel hybrid optimization policy (QSOD) for the Q-value selection of individual agents in MARL. We selected the individual agent's Q-values according to GWO. Through the proposed method, proper attention was paid between the agents, which helped the agents to learn quickly and accurately. Moreover, we used a similar network architecture to QMIX, which requires less computational power than other state-of-the-art algorithms like QVMix and QPLEX. Additionally, the experimental results using SC2LE demonstrate that our proposed algorithm consistently outperformed QMIX and QVMix in all scenarios. Furthermore, our proposed algorithm required less time for each episode than the comparison algorithms.

Our results show the utility of the proposed algorithm. One limitation, however, is that GWO is unable to handle a very large number of agents, particularly when the number of agents reaches 1 million or more. In such cases, the one leader agent cannot control the entire group of agents. We will address this issue in future research.

Moreover, we plan to apply our proposed algorithm to mobile devices such as fire-fighter robots and spy drones to improve their efficiency. For this purpose, we will reduce the size of the agent network and try to use a centralized system as a mixing network.

REFERENCES

[1] Y. Cao, W. Yu, W. Ren, and G. Chen, "An overview of recent progress in the study of distributed multi-agent coordination," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 427–438, Feb. 2013.

[2] W. Ying and S. Dayong, "Multi-agent framework for third party logistics in E-commerce," *Expert Syst. Appl.*, vol. 29, no. 2, pp. 431–436, Aug. 2005.

[3] E. Yang and D. Gu, "Multiagent reinforcement learning for multirobot systems: A survey," Tech. Rep., 2004.

[4] M. Huttenrauch, M. Šošć, and G. Neumann, "Guided deep' reinforcement learning for swarm systems," in *Proc. AAMAS Auton. Robots Multirobot Syst. (ARMS) Workshop*, Sep. 2017, pp. 2–16.

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

[6] P. Peng, Y. Wen, Y. Yang, Q. Yuan, Z. Tang, H. Long, and J. Wang, "Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play StarCraft combat games," 2017, *arXiv:1703.10069*. [Online]. Available: http://arxiv.org/abs/1703.10069

[7] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. S. Torr, P. Kohli, and S. Whiteson, "Stabilising experience replay for deep multi-agent reinforcement learning," in *Proc. The 34th Int. Conf. Mach. Learn.*, 2017, pp. 1146–1155.

[8] E. Jorge, M. Kågebäck, and E. Gustavsson, "Learning to play guess who? and inventing a grounded language as a consequence," in *Proc. NIPS Workshop Deep Reinforcement Learn.*, 2016, pp. 1–10.

[9] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multiagent policy gradients," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 1–9.

[10] C. S. D. Witt, J. Foerster, G. Farquhar, P. Torr, W. Boehmer, and S. Whiteson, "Multi-agent common knowledge reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32. Red Hook, NY, USA: Curran Associates, 2019, pp. 9924–9935.

[11] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *Proc. 17th Int. Conf. Auton. Agents Multiagent Syst.*, 2017, pp. 1–17.

[12] D. Ha, A. Dai, and Q. V. Le, "HyperNetworks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017.

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

[14] O. Vinyals *et al.*, "StarCraft II: A new challenge for reinforcement learning," 2017, *arXiv:1708.04782*. [Online]. Available: http://arxiv.org/abs/1708.04782

[15] N. Mittal, U. Singh, and B. S. Sohi, "Modified grey wolf optimizer for global engineering optimization," *Appl. Comput. Intell. Soft Comput.*, vol. 2016, pp. 1–16, Mar. 2016, doi: 10.1155/2016/7950348.

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

[17] A. Tampuu, T. Matiisen, D. Kodelja, I. K. K. Kuzovkin, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PloS One*, vol. 12, no. 4, 2017, Art. no. e0172395.

[18] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Univ. Cambridge England, Cambridge, U.K., 1989.

[19] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

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

[21] C. Guestrin, D. Koller, and R. Parr, "Multiagent planning with factored MDPs," in *Advances in Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 2002, pp. 1523–1530.

[22] J. R. Kok and N. Vlassis, "Collaborative multiagent reinforcement learning by payoff propagation," *J. Mach. Learn. Res.*, vol. 7, pp. 1789–1828, Sep. 2006.

[23] S. Sukhbaatar and R. Fergus, "Learning multiagent communication with backpropagation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2244–2252.

[24] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Agents and Multiagent Systems*. Springer, 2017, pp. 66–83.

[25] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6382–6393.

[26] N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala, "Episodic exploration for deep deterministic policies: An application to StarCraft micromanagement tasks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 201.

[27] S. Mirjalili, "How effective is the Grey Wolf optimizer in training multi-layer perceptrons," *Appl. Intell.*, vol. 43, no. 1, pp. 150–161, 2015.

[28] E. Emary, H. M. Zawbaa, and C. Grosan, "Experienced gray wolf optimization through reinforcement learning and neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 3, pp. 681–694, Mar. 2018.

[29] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Adv. Eng. Softw.*, vol. 69, pp. 46–61, Mar. 2014.

[30] M. Irodova and R. H. Sloan, "Reinforcement learning and function approximation," in *Proc. FLAIRS Conf.*, 2005, pp. 455–460.

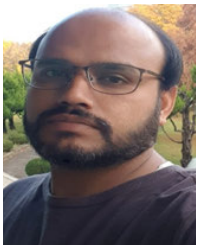
[31] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE Access*, vol. 7, pp. 133653–133667, 2019, doi: 10.1109/ACCESS.2019.2941229.

[32] J. Z. Leibo, V. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel, "Multi-agent reinforcement learning in sequential social dilemmas," in *Proc. 16th Conf. Auto. Agents Multiagent Syst.*, 2017, pp. 464–473.

[33] S. Sawyer, "Conceptual errors and social externalism," *Phil. Quart.*, vol. 53, no. 211, pp. 265–273, Apr. 2003.

[34] A. Mahajan, T. Rashid, M. Samvelyan, and S. Whiteson, "MAVEN: MultiAgent variational exploration," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates, 2019, pp. 7611–7622.

[35] J. Wang, Z. Ren, T. Liu, Y. Yu, and C. Zhang, "QPLEX: Duplex dueling multi-agent Q-learning," 2020, *arXiv:2008.01062*. [Online]. Available: http://arxiv.org/abs/2008.01062
[36] P. Leroy, D. Ernst, P. Geurts, G. Louppe, J. Pisane, and M. Sabatelli, "QVMix and QVMix-max: Extending the deep quality-value family of algorithms to cooperative multi-agent reinforcement learning," 2020, *arXiv:2012.12062*. [Online]. Available: http://arxiv.org/abs/2012.12062
[37] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi, "Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5887–5896.



HAFIZ MUHAMMAD RAZA UR REHMAN received the bachelor's degree in electronics engineering from The Islamia University of Bahawalpur, Pakistan, in 2012. He is currently pursuing the Ph.D. degree with Yeungnam University, South Korea. His research interests include RL, gaming, data science, and networks optimizers.

BYUNG-WON ON received the M.S. degree from the Department of Computer Science and Engineering, Korea University, Seoul, South Korea, in 2000, and the Ph.D. degree from the Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, in 2007. He is currently an Associate Professor with the Department of Software Convergence Engineering, Kunsan National University, Gunsan-si, Jeollabuk-do, South Korea, where he also leads the Data Intelligence Laboratory. His current research interests include around data mining, in particular probability theory and applications, machine learning, and artificial intelligence, mainly working on abstractive summarization, creative computing, and multiagent reinforcement learning. He is an Editor of journal of *Korean Institute of Information Scientists and Engineers* (KIISE), *Electronics and Telecommunications Research Institute Journal* (ETRI), and *Quality and Quantity*. Currently, he is serving as a Committee Member for ISO/IEC JTC 1/SC 32, Korean Association of Data Science, and SIG on Human Language Technology with the Korean Institute of Information Scientists and Engineers (KIISE). He is a Committee Member of the Informatization Committee and Jeonbuk Large Leap Policy Consulation Body in Jeollabuk-do Provincial Government.

DEVARANI DEVI NINGOMBAM received the M.Tech. degree in communication engineering from the National Institute of Technology Karnataka (NITK), Surathkal, India, in 2015, and the Ph.D. degree in computer engineering from Chosun University, Gwangju, South Korea, in 2019. Then, she joined the Planning Division, Electronics and Telecommunications Research Institute (ETRI), South Korea, in 2019, where she is currently a Postdoctoral Researcher. Her current research interests include multiagent reinforcement learning (MARL), the Internet of Things (IoT), device-to-device (D2D) communications, and wireless networks.

SUNGWON YI received the M.S. and Ph.D. degrees in computer science and engineering from Pennsylvania State University, in 2004 and 2005, respectively. Before joining Pennsylvania State University, he worked at LG-CNS, as a System Engineer. Since 2005, he has been with ETRI, South Korea, where he is currently a Researcher with the Future Technology Research Laboratory and the Planning Division. His research interests include the areas of network security, storage systems, mobile computing, and machine learning. He has been served on the Technical Program Committee for the IEEE GLOBECOM and ICC, since 2005.

GYU SANG CHOI (Member, IEEE) received the Ph.D. degree in computer science and engineering from Pennsylvania State University. He was a Research Staff Member with Samsung Advanced Institute of Technology (SAIT), Samsung Electronics, from 2006 to 2009. Since 2009, he has been with Yeungnam University, where he is currently an Assistant Professor. His research interests include data mining, deep learning, computer vision, storage systems, parallel and distributed computing, supercomputing, cluster-based web servers, and data centers. His current focus is on text mining and reinforcement learning and deep learning with computer vision, whereas his prior research mainly focused on improving the performance of clusters. He is a member of the ACM.