IEICE TRANSACTIONS

on Information and Systems

DOI:10.1587/transinf.2023EDP7180

Publicized:2024/04/03

This advance publication article will be replaced by the finalized version after proofreading.

A PUBLICATION OF THE INFORMATION AND SYSTEMS SOCIETY
The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

PAPER

Agent Allocation-Action Learning with Dynamic Heterogeneous Graph in Multi-task Games

Xianglong LI[†], **Yuan LI**[†], **Jieyuan ZHANG**[†], **Xinhai XU**[†*], ***and*** **Donghong LIU**[†*], ***Nonmembers***

SUMMARY In many real-world problems, a complex task is typically composed of a set of subtasks that follow a certain execution order. Traditional multi-agent reinforcement learning methods perform poorly in such multi-task cases, as they consider the whole problem as one task. For such multi-agent multi-task problems, heterogeneous relationships i.e., subtask-subtask, agent-agent, and subtask-agent, are important characters which should be explored to facilitate the learning performance. This paper proposes a dynamic heterogeneous graph based agent allocation-action learning framework. Specifically, a dynamic heterogeneous graph model is firstly designed to characterize the variation of heterogeneous relationships with the time going on. Then a multi-subgraph partition method is invented to extract features of heterogeneous graphs. Leveraging the extracted features, a hierarchical framework is designed to learn the dynamic allocation of agents among subtasks, as well as cooperative behaviors. Experimental results demonstrate that our framework outperforms recent representative methods on two challenging tasks, i.e., SAVETHECITY and Google Research Football full game.

key words: *Dynamic Heterogeneous Graph, Allocation-Action learning, Multi-task Games*

1. Introduction

Great progress has been made in the domain of multi-agent reinforcement learning in recent years [1]–[4]. Most works concentrate on solving a single task by learning the cooperative behaviours of agents. However, many real-world problems often consist of a set of subtasks, which should be completed by the cooperation of agents. For example, firefighting [5] involves subtasks like water supply and fire control. Firefighters need to be distributed to subtasks and then they collaborate to accomplish all subtasks in accordance with the fire-fighting procedure. The multi-agent multi-task game requires coordination among multiple agents, which involves two intertwined problems. One is that agents should be dynamically assigned to subtasks. The other one is that how the policy of each subtask is learned with dynamically assigned agents.

Designing strategies for dynamically allocating agents to complete subtasks poses significant challenges in a multi-agent multi-task game. Firstly, the game state changes dynamically over time, and the relationships between subtasks are complex and changeable. This complexity increases the difficulty of finding an effective agent allocation scheme. Secondly, the decision-making of agents is not only influenced by collaborative relationships within allocated subtasks but also by dynamic changes in the external states of the subtasks. Some works [5]–[8] propose a layered approach to solve the above problems, in which the upper layer is agent allocation and the lower layer is agent execution. For example, the work [9] learns a centralized controller to coordinate decentralized agents into different teams, where each agent in the team executes independently based on local observations. The work [10] leverages task structure decomposability to reduce the learning space by allowing agents to form local groups, enabling targeted attention to specific subtasks and achieving superior performance. However, they only consider the cooperation of agents within subtasks, overlooking the collaboration among agents across different subtasks. Additionally, relationships among subtasks, agents, and subtask-agent interactions are not fully explored. This limitation hinders inter-agent coordination and prevents valuable insights into the agent's learning process.

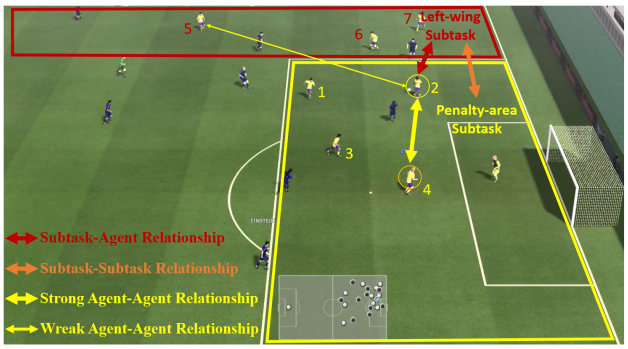


Fig. 1: Different cooperative relationships in a soccer game.

To illustrate such relationships, we provide an example which is shown in Figure 1. There are two subtasks i.e., left-wing subtask and penalty-area subtask, with which players work collaboratively to score a goal. For player 2, to make a good decision, he should first pay attention to players within the same subtask, i.e., player 1, player 3, player 4. Strong relationships between player 2 and the three players should be established. Meanwhile, player 2 should judge the overall situation in the playground, which means information about the two subtasks should be considered. Relationships for the two subtasks and for player 2 with the left-wing subtask should be established. Further, player 2 should also keep

[†]The author is with Academy of Military Sciences, Beijing, 100091, China
[*]E-mail: liu_donghong@sina.cn

an eye on players in the left-wing subtask to decide which direction he should run. Relationships between player 2 and player 5, 6, 7 should be established. Therefore, building a relationship graph among subtasks and agents will be beneficial for the learning of making decisions for each player. How to construct and utilize such graph to facilitate the learning of agent allocation and actions is the main problem considered in our study.

In this paper, we propose an agent allocation-action learning framework based on the construction of the dynamic heterogeneous graph, which enables the effective exploration of different relationships in multi-agent multi-subtask games. Multi-subtasks are typically carried out in accordance with a specific process, and one common method for task decomposition is the utilization of the Behavior Tree (BT), a widely adopted tool. The considered multi-agent multi-subtask problem has two key characters, i.e., subtasks are not active all the time and heterogeneous relationships should be properly modelled. To characterize the variation of relationships with the time going on, we construct a dynamic heterogeneous graph with agent-agent, subtask-agent, subtask-subtask edges. Two types of nodes, i.e., active subtask nodes and agent nodes, are involved in the graph. For extracting features of the heterogeneous graph, we design a multi-subgraph partition method which divide the heterogeneous graph to three different subgraphs. We employ the graph attention network model on each subgraph to calculate weights on different edges and aggregate node features according to the weights. With generated features, we design a hierarchical framework for learning both agent allocation policy and action policy.

Finally, we conduct extensive experiments to evaluate our framework on two challenging benchmarks, i.e., SAVETHECITY [5] and Google Research Football full game. We compare the proposed method with some recent representative baselines. With introduced the heterogeneous graph and feature extraction mechanism, our method performs better than other hierarchical learning methods. Compared to the typical multi-agent method, i.e. QMIX, our method shows great advantages on multi-task problems, gaining much better performance. We also make some ablation studies to show the effectiveness of each component of the proposed framework.

2. Related Work

In this section we delve into some recent hierarchical methods used for learning agent allocation and strategies, show how they handle such problems, and point out differences with our work.

The multi-agent multi-task problem was proposed in [11], which makes a formal analysis of the taxonomy of agent-task allocation in multi-robot systems. Some hierarchical methods have been proposed for this problem. The work [6] introduces a hierarchical method for which the upper layer is the task allocation and the lower layer is the task execution. The task allocation is solved by heuristic algorithms, while the task execution is solved by a coordinated reinforcement learning algorithm. The work [12] introduces a learning-based approach for task allocation, which assumes pre-existing subtask execution policies. However, these works fail to simultaneously learn the agent allocation policy and the task execution policy.

For learning in both layers, the work [5] introduces a general learning method for addressing diverse multi-agent subtasks, where each subtask is presumed to be independent, and each agent only focuses on the assigned subtask. Some works consider to use value decomposition based methods to enhance cooperation of agents in different subtasks. The work [7] learns to group agents into different roles which execute different subtasks. A mix network is used to strengthen cooperative behaviors of all agents. The work [8] involves training a general cooperative decentralized policy to improve teamwork among agents by teaching them different skills. However, the subtask information is not utilized during the learning process. The work [13] proposes a hierarchical multi-agent allocation-action learning framework for multi-subtask games, which including allocation learning and action learning, which could allocate agents to different subtasks as well as compute the action for each agent in real-time. However, these works ignore the impact of the relationship between subtasks, between agents, and between agents and subtasks on agent allocation and action execution.

3. Background

3.1 Heterogeneous Graph Neural Network

Heterogeneous graphs [14] allow for the representation of complex relationships and interactions between entities in various applications, including recommendation systems [15], social networks [16], knowledge graphs [17], and biology [18]. A heterogeneous graph is defined as a tuple $G = (V, E, T_V, T_E)$ with a node set V, an edge set E, a node type set T_V, and an edge type set T_E. Each node v is assigned a node type. Each edge connects two nodes (u, v) where $u, v \in V$, and each edge $e \in E$ is assigned an edge type. Each heterogeneous graph has multiple node and edge types such that $|T_V| + |T_E| > 2$. Different types of nodes and edges have attribute representations of varying dimensions. Heterogeneous graph neural networks (HGNs) [19], [20] are designed to process and extract features from heterogeneous graphs. HGNs acquire node embedding by aggregating and propagating information across diverse node types and their relationships, which enables the extraction of comprehensive semantic information and relational patterns. Subsequently, HGNs utilize the node embedding for tasks like node classification, link prediction, and graph-level prediction. For nodes in a heterogeneous graph, the node embedding can be calculated using the following equation (1):

$$h_v^k = \text{AGGREGATE}_k \left(h_u^{k-1}, \forall u \in N(v) \right) \tag{1}$$

Here, h_v^k represents the embedding of node v at the k-th layer, $N(v)$ denotes the set of neighboring nodes of v, and

h_u^{k-1} represents the embedding of node u at the (k-1)-th layer. AGGREGATE$_k$ is the aggregation function for the k-th layer, used to aggregate the embedding information of neighboring nodes. Common forms of aggregation functions include average pooling, max pooling, or attention mechanisms.

3.2 Multi-agent reinforcement learning

We consider a multi-agent game that can be modeled as a Markov decision process [21]. It be represented the tuple $\langle \mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} \rangle$, where $\mathcal{N} = \{1, 2, \ldots, N\}$ is the set of agents. $\mathcal{S} = \{S^t\}_{t=0}^{T}$ is the environment state where S^t is the state at time t and T is the maximum time steps. $\mathcal{A} = \{A_i\}_{i=0}^{N}$ represents the action space for all agents. $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$ is the reward function. $\mathcal{P} = \{P_{ss'}^{a} \mid s, s' \in \mathcal{S}, a \in \mathcal{A}\}$ is the state transition function, and $P_{ss'}^{a}$ gives the probability from state s to state s' if the action a is taken. A popular solution approach for multi-agent games is the value decomposition based MARL method [7], [21]. Each agent i is equipped with an individual neural network that calculates the agent's specific state-action value Q_i, which represents the expected cumulative future rewards when taking a specific action $a \in A_i$ in a given state s. During the training process, the global state-action value Q_{tot} is computed as a function of the individual values Q_i of all agents to enhance cooperative behaviors of all agents. Currently, various algorithms have been proposed to investigate the relationship between Q_{tot} and Q_i. For example, the summation approach utilized in VDN [22] and the employment of a mixing network in QMIX [23]. The entire neural network is updated using the TD-loss, as represented in (2), where R is the immediate reward obtained after taking action a in the current state. The discount factor γ balances the importance of current and future rewards, while s' denotes the next state achieved after taking action a. The term $\max_{a'} Q_{tot}(s', a')$ represents the target Q-value, indicating the maximum Q-value action a' in the next state s'. On the other hand, $Q_{tot}(s, a)$ represents the current estimate of the global state-action value provided by the model.

$$loss = R + \gamma \max_{a'} Q_{tot}(s', a') - Q_{tot}(s, a) \tag{2}$$

3.3 Behavior Trees

Thanks to the modularity and reactivity of BTs [24]–[26], they serve as organizational structures for multiple subtasks in complex tasks. BTs provide an intuitive way to describe the behavior of an agent and support the decomposition, execution, and decision-making of complex tasks. A BT consists of a series of nodes, with each node representing a specific behavior or control structure. The nodes in a BT can be classified into three main types: *Action*, *Condition*, and *Control* Nodes. *Action* nodes represent concrete actions or operations, such as moving, attacking, or turning on lights. *Condition* nodes determine the execution path based on the satisfaction of certain conditions. *Control* Nodes are used to organize and control the execution order and conditions of other nodes. The common types of control flow nodes are *Sequence*, *Fallback*, and *Parallel*. A BT decomposes tasks into a series of subtasks and representing them as nodes in a behavior tree [27], the execution flow of tasks can be better organized and managed. For example, subtasks that need to be executed in sequence are organized through *Sequence* nodes. *Selection* nodes can organize subtasks with priority. *Parallel* nodes are suitable for situations where multiple subtasks need to be executed simultaneously.

4. Methodology

In this section, we present a dynamic heterogeneous graph based agent allocation-action learning framework (DyHG-3A) for multi-agent multi-task game. In the following, We first introduce the overall design of our framework. Then we detail three important components of our framework, i.e., the dynamic heterogeneous graph construction, the heterogeneous graph attribute embedding and the agent allocation-action Learning. Finally, the training of DyHG-3A is introduced.

4.1 Overall Design of DyHG-3A

The architecture of DyHG-3A contains three modules as shown in Figure 2.

(1) Dynamic Heterogeneous Graph Construction module: The detailed execution process of multiple subtasks is organized through a BT, each agent (red circle) needs be associated with a certain activated subtask (green triangle). A dynamic heterogeneous graph is constructed to characterize relations among subtasks, among agents and subtask-agent.

(2) Heterogeneous Graph Attribute Embedding module: An embedding mechanism is designed to extract features from different node attributes of dynamic heterogeneous graphs. $(G_{AA})^t$, $(G_{SS})^t$, and $(G_{SA})^t$ represent the agent subgraph, the subtask subgraph, and the subtask-agent subgraph at time step t, respectively. Then different GAT networks are used to aggregate distinct node attributes. For the subtask-agent subgraph, two different Multilayer Perceptrons (MLPs) are set up to unify the node feature dimensions.

(3) Agent Allocation-Action Learning module: This module facilitates the allocation of agents to specific subtasks, enabling efficient action execution. It consists of two architectures: the agent allocation learning network and the agent action learning network. The input of the agent allocation learning network includes the observation $o_i^t = (H_A)_i^t \oplus (H_{AA})_i^t$ for each agent and the attribute matrix $X^t = x_1^t, ..., x_j^t$ for all subtasks at the time step t, where $x_j^t = (H_S)_j^t \oplus (H_{SS})_j^t$, and $\oplus$ denotes the concatenation operation.

4.2 Dynamic Heterogeneous Graph Construction

Let us consider a multi-agent multi-task problem in which there are M subtasks and N agents, which is modeled as a heterogeneous graph $G = (V, E, T_V, T_E)$. V represents the set of agents and active subtasks. There are three types of undirected edges in E, i.e., *subtask-subtask* edges signifying collaboration relationships among subtasks, *subtask-agent* edges denoting agents cooperating to complete a certain

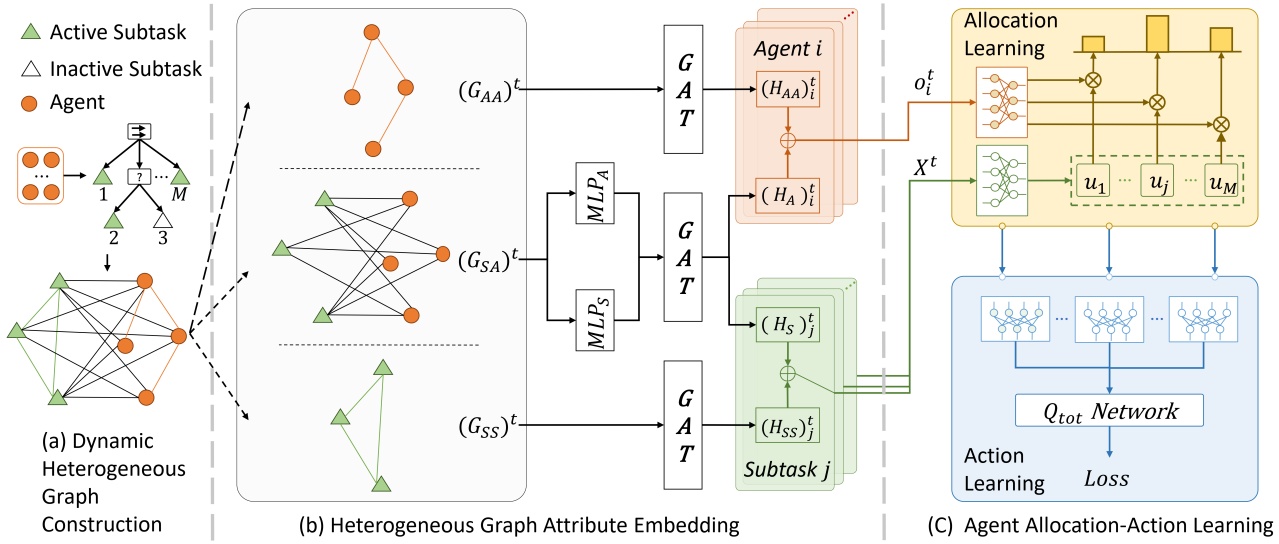


Fig. 2: The overall framework of DyHG-3A.

subtask, and *agent-agent* edges representing collaboration between agents. At time step t, the heterogeneous graph is represented by G^t. $V^t = V_S^t \bigcup V_A$ is the set of heterogeneous nodes in which V_S^t is the set of active subtask nodes and V_A is the set of agent nodes. Note that V_S^t varies with the time as activate status of the subtask always changes. $E^t = \{E_{SS}^t, E_{AA}^t, E_{SA}^t\}$ is the set of heterogeneous edges in G^t, where E_{SS}^t, E_{AA}^t and E_{SA}^t represent the sets of subtask-subtask edges, agent-agent edges, and subtask-agent edges, respectively. The dynamic heterogeneous graph construction process is shown in Algorithm 1, which is named as DyHGC. The input is the current time step t, the subtask set $\mathcal{M}$ and the agent set $\mathcal{N}$. The output is a heterogeneous graph at time step t. Initially, V_S^t and E^t are set to be empty while V_A is set to be $\mathcal{N}$ (step 1). If a subtask m is activated, it is added to the V_S^t (Steps 3-4). Meanwhile, edges between subtask m and all agent nodes are established, which are added to the set E_{SA}^t (Steps 5-6). Additionally, if parallel subtasks are detected between any two subtasks, edges are established and added to E_{SS}^t (Steps 7-9). For any two agent node, we mainly focus on adjacent agents. In a large-scale multi-agent environment, not all agents need to interact with each other because the number of agents is large and the distance between some agents may be extremely large. Therefore, we establish an edge when the Euclidean distance between them is smaller than a predefined threshold d (Steps 10-13). Finally, we obtain a heterogeneous graph with active nodes and generated edges at time step t.

Algorithm 1: DyHGC

Input: Current time step t, All subtask set $\mathcal{M}$, Agent set $\mathcal{N}$
Output: Current heterogeneous graph G^t

1 Set $E^t = \phi$; $V_A = \mathcal{N}$
2 **for** $m \in \mathcal{M}$ **do**
3 **if** *m is activated* **then**
4 Add m to V_S^t;
5 **for** $v \in V_A$ **do**
6 Add edge $e_{m,v}$ to E_{SA}^t;
7 **for** $m' \in \mathcal{M}$ **do**
8 **if** *m and m' are parallel subtasks in current BT and $m \neq m'$* **then**
9 Add edge $e_{m,m'}$ to E_{SS}^t;
10 **for** $v \in V_A$ **do**
11 **for** $v' \in V_A$ **do**
12 **if** $Dis(v, v') < d$, *and* $v \neq v'$ **then**
13 Add edge $e_{v,v'}$ to E_{AA}^t;
14 $E^t = \{E_{SS}^t \cup E_{AA}^t \cup E_{SA}^t\}$; $V^t = \{V_S^t, V_A\}$;
15 return G^t;

4.3 Heterogeneous Graph Attribute Embedding

Utilizing the heterogeneous graph G^t at time step t, the main task of heterogeneous graph attribute embedding is to capture heterogeneous node relationship information. We introduce a multi-subgraph partition method that uses the attention model to capture the heterogeneity attributes. According to the edge type, G^t is decomposed into three types of subgraphs, i.e., agent subgraph $(G_{AA})^t$, subtask subgraph $(G_{SS})^t$, and subtask-agent subgraph $(G_{SA})^t$. For each subgraph, we utilize Graph Attention Network (GAT) [28] model to learn the attention weights of each node's neighborhoods and generate new latent representations by aggregating the attributes of these important neighborhoods. The attention weight coefficient of node pair (z, z') can be calculated as:

$$I_{z,z'}^t = Softmax\left(\frac{\exp\left(\sigma\left(W \cdot \left[a_z^t \| a_{z'}^t\right]\right)\right)}{\sum_{k \in N_z^t} \exp\left(\sigma\left(W \cdot \left[a_z^t \| a_k^t\right]\right)\right)}\right) \tag{3}$$

where a_z^t represents the attribute vector of node z, W is the parameterized weight vector of the attention function, and

N_z^t is the sampled neighbors of node z at time step t. σ represents the LeakyReLU activation function, and $\|$ denotes the operation of concatenation, subsequently normalized by the Softmax function. After that, aggregating the neighbors' latent embedding with calculated attention weight coefficients, we can obtain the final attribute representation of node i for the subgraph type and the time step t as:

$$(H_z)^t = \sigma\left(\sum_{z' \in N_z^t} I_{z,z'}^t \cdot a_z^t\right) \tag{4}$$

where $(H_z)^t$ is the aggregated attribute embedding of node z at the current time step t.

Subtask Subgraph Attribute Embedding The attention attribute embedding for the subtask subgraph aims to learn attention weights for two subtask nodes and generate a new latent attribute representation by aggregating the attributes of the subtask neighborhoods. The attribute of subtask node $j \in V_S^t$ is denoted by $x_j^t \in \mathbb{R}^{d_s}$, where d_s represents the subtask attribute dimension. For $(G_{SS})^t$ at time step t, the final attribute representation of subtask node j calculated by Equations 3 and 4 is denoted by $(H_{SS})_j^t$. To obtain stable and effective attribute representations, we utilize the GAT model with multi-head mechanisms. Specifically, we employ k independent GAT models in parallel and concatenate the learned features to obtain the output embedding. Consequently, the multi-head attention representation of subtask node j can be calculated as $(H_{SS})_j^t = \text{Concat}((H_{SS})_j^1, \ldots, (H_{SS})_j^k)$. After applying the multi-head GAT models, we obtain a set of subtask node attribute representations for $(G_{SS})^t$, denoted as $\{(H_{SS})_1^t, \ldots, (H_{SS})_{|V_S^t|}^t; (H_{SS})_j^t \in \mathbb{R}^{d_s}\}$.

Agent Subgraph Attribute Embedding The agent subgraph attention attribute embedding plays a crucial role in capturing collaborative attributes among different agents. The attribute of agent node $i \in V_A$ is denoted by $o_i^t \in \mathbb{R}^{d_a}$, where d_a represents the agent attribute dimension. For $(G_{AA})^t$ at time step t, the final attribute representation of agent node i calculated by Equations 3 and 4 is denoted by $(H_{AA})_i^t$. After applying the multi-head GAT models, we obtain sets of agent node attribute representations $\{(H_{AA})_1^t, \ldots, (H_{AA})_{|V_A|}^t; (H_{AA})_i^t \in \mathbb{R}^{d_a}\}$.

Subtask-Agent Subgraph Attribute Embedding The subtask-agent subgraph attention attribute embedding enables the capture of heterogeneous attributes between subtasks and agents. Note that subtask nodes and agent nodes have distinct dimensional attributes, making the direct aggregation of heterogeneous neighbor information infeasible. We begin by using two non-linear transformation networks to map the initial attributes of subtask nodes and agent nodes to the same feature space. It can be described by Equation 5, where W_S, W_A, b_S, and b_A are the shared learnable weight matrices and bias vectors.

$$x_j^t = \sigma(W_S \cdot x_j^t + b_S); o_i^t = \sigma(W_A \cdot o_i^t + b_A); \tag{5}$$

Then, in $(G_{SA})^t$, the attention weights coefficient between subtasks and agents are separately calculated using Equation 3. Note that the neighborhood nodes for subtask node j consist of the set of agent nodes, while for agent node i, the neighborhood nodes consist of the set of subtask nodes. Subsequently, node attributes are aggregated through Equation 4, with the aggregated attributes of agent nodes denoted as $(H_A)_j^t$, and the aggregated attributes of subtask nodes denoted as $(H_S)_j^t$. After applying the multi-head GAT models, we obtain sets of attribute representations for subtask nodes and agent nodes, denoted as $\{(H_S)_1^t, \ldots, (H_S)_{|V_S^t|}^t\}$ and $\{(H_A)_1^t, \ldots, (H_A)_{|V_A|}^t\}$, respectively, for $(G_{SA})^t$.

Finally, the attribute embeddings of subtask nodes and agent nodes are denoted as $x_j^t = (H_{SS})_j^t \| (H_S)_j^t$ and $o_i^t = (H_{AA})_i^t \| (H_A)_i^t$, respectively. These embeddings will be utilized in the agent allocation-action learning module.

4.4 Agent Allocation-Action Learning

Based on the node attribute embedding generated in the previous section, we introduce a hierarchical mechanism for learning the dynamic allocation of agents among subtasks, as well as cooperative behaviors. On the upper layer, a subtask allocation module is used to connect an agent network to a subtask network, i.e., assigning each agent to a certain subtask. For the lower layer, an action learning module is designed to compute appropriate actions for each agent with the allocation result. All active subtask attribute embedding $X^t = x_1^t, x_2^t, ..., x_{|V_S^t|}^t$ are encoded by a neural network to vectors, i.e., $U^t = u_1, u_2, ..., u_{|V_S^t|}$. Each agent attribute embedding o_i^t are encoded by another neural network and then multiplied with U^t. The output is the Q-values of all possible choices of subtasks, i.e., $\bar{Q}_1, \bar{Q}_2, ..., \bar{Q}_{|V_S^t|}$. In the execution mode, the index corresponds to the maximum value is the subtask that should be assigned to the agent i. Once a subtask m is assigned to agent i, each agent attribute embedding o_i^t will be passed to the input of the subtask network, which then select a proper action $a \in \mathcal{A}$ for an agent among all possible actions, where $\mathcal{A}$ is action space. The output of the subtask network is the vector of Q-values of all actions for agent $i < |V_A|$, i.e., $Q_1^i, Q_2^i, ..., Q_{|\mathcal{A}|}^i$. To combine Q-values of all agents together, we introduce a neural network f to compute Q_{tot}, i.e., $Q_{tot} = f(Q_{a_1}^1, Q_{a_2}^2, ..., Q_{a_N}^{|V_A|})$. a_i is the action selected by agent i following the $\epsilon-$greedy principle. We train our framework end-to-end by optimizing the following TD-loss function:

$$\begin{aligned} Loss = r^t &+ (\gamma_1 \max_{m'} \bar{Q}(X^{t+1}, o_i^{t+1}, m') - \bar{Q}_(X^t, o_i^t, m)) \\ &+ (\gamma_2 \max_{a'} Q_{tot}(o_i^{t+1}, a') - Q_{tot}(o_i^t, a)) \end{aligned} \tag{6}$$

where γ_1 is the discount factor of allocation learning, γ_2 is the discount factor of action learning.

4.5 The Overall Training

In this section, we describe how heterogeneous graph construct, attribute embedding, and allocation-action learning work together in our framework. The whole learning procedure is illustrated in Algorithm 2. Each episode repre-

Algorithm 2: DyHG-3A Training Procedure

Input: A BT with the set of subtasks $\mathcal{M}$, the set of agents $\mathcal{N}$
Output: Parameters of whole neural networks

1 **for** *episode* $= 1, 2, \ldots, L$ **do**
2 Reset the Environment and BT parameters;
3 Initialize heterogeneous graph $G^t = (V^t, E^t)$, i.e., $V^t = V_S^t \bigcup V_A$, $V_S^t = \phi$, $E^t = \phi$;
4 **while** $t \leq T$ **do**
5 Constructing HetG by Algorithm 1, i.e., $G^t = DyHGC(t, \mathcal{M}, \mathcal{N})$;
6 Collecting agent observations O^t and activating subtask attributes X^t;
7 Calculating $(H_{SS})_j^t$, $(H_{AA})_i^t$, $(H_S)_j^t$ and $(H_A)_i^t$ based on Eq.(4);
8 Getting attribute emmbedding of activated subtasks and all agents is $x_j^t = (H_{SS})_j^t \| (H_S)_j^t$ and $o_i^t = (H_{AA})_i^t \| (H_A)_i^t$;
9 Computing Q-values $\bar{Q}_1, \bar{Q}_2, \ldots, \bar{Q}_{|V_S^t|}$ with the allocation learning;
10 Getting subtask m_i^t assigned to each agent i;
11 Computing Q-values $Q_1^i, Q_2^i, \ldots, Q_{|\mathcal{A}|}^i$ with subtask network m_i^t;
12 Getting the action a_i^t for each agent $i \in V_A$;
13 Running the BT with allocation results and actions of agents, and getting the game reward r^t;
14 Storing the sample $(X^t, o_i^t, m_i^t, a_i^t, r^t)$ to buffer $\mathcal{D}$;
15 **if** *UPDATE* **then**
16 Randomly select samples from $\mathcal{D}$;
17 Computing $Loss$ following (6), and perform gradient descent over the whole network.

sents a game round, and L represents the maximum training episodes. The maximum time steps of the game are T. For each time step t, getting observations of all agents $O^t = o_1^t, o_2^t, \cdot, o_{|V_A|}^t$, and attribute of all activate subtasks X^t. Firstly, with Algorithm 1, a heterogeneous graph is constructed in Step 5. Then, Steps 6-8 to obtain attribute embedding between two different nodes. In steps 9-10 allocate agents to complete the subtask. After completing the agent allocation for each agent, obtain the specific action of each agent through the subtask network (steps 11-12), and related samples are generated. Finally, the keyword "UPDATE" in step 14 is used to control the frequency of updating the neural networks.

5. Experiments

In this section, we evaluate the performance of our proposed framework DyHG-3A on two challenging environments, i.e., SAVETHECITY [5] and Google Research Football (GRF) [29] full game.

5.1 Scenarios

SAVETHECITY is an environment similar to a "Search-and-Rescue task [30]" where agents must coordinate firefighting and rebuilding in a burning city. The complete scenario consists of a 16x16 grid with N agents (circles) and

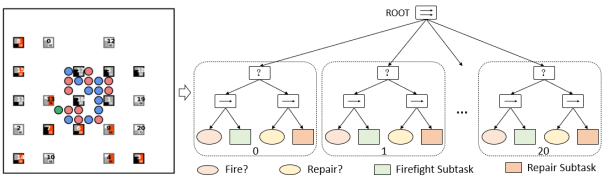
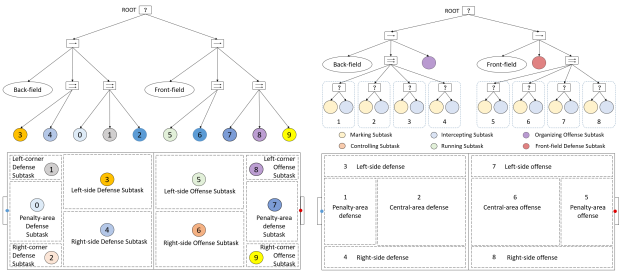


Fig. 3: Left is a snapshot of 20a_20b scenario. Right is the structure of a BT organization subtasks, where the number represents the index of the building.

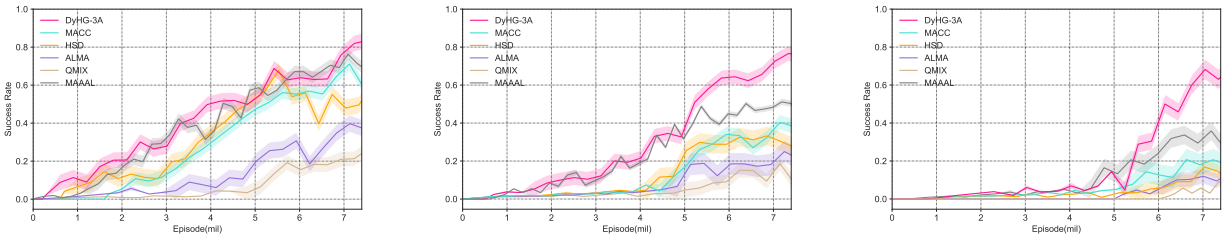
$N + 1$ buildings (squares) as shown in Figure 3(Left). The buildings are randomly generated on the map and have a certain probability of catching fire (red bars) at the start of each episode, which decreases their "health" (black bars). There are three types of agents: firefighter (red), builder (blue), and generalist (green). Firefighters are most effective at extinguishing fires, while builders excel at repairing damaged buildings. The generalists have twice the movement speed of firefighters and builders, and it can prevent further damage to burning buildings while assisting other agents. We first need to assign agents to different burning buildings, and then learn low-level actions to navigate between subtasks and cooperate to complete the firefighting and building repair subtasks.



(a) $10\_subtasks$ (b) $18\_subtasks$

Fig. 4: Two scenarios with different number of subtasks for GRF.

Google Research Football GRF simulates a complete football game with two teams and a ball. Each team is controlled by 11 agents that can receive state information from the field and decide their next actions. The goal of the game is to score as many goals as possible by shooting the ball into the opponent's goal. We divide the field into Back-field and Front-field zones. Under different task decomposition rules, we have constructed two distinct BTs comprising varying quantities of subtasks, $10\_subtasks$ and $18\_subtasks$, as shown in Figure 4. In the $10\_subtasks$ scenario (Figure 4(a)), the Back-field has five defensive subtasks for left-side, right-side, left-corner, right-corner, and penalty. The left-side and right-side subtasks work in parallel to protect the central region of the Back-field. The left-corner, right-corner, and penalty area defensive subtasks operate in parallel, providing protection to the goal area, with their parent nodes executed sequentially. The Front-field has five offensive subtasks for the left-side, right-side, left-corner,



(a) 7a_7b (b) 20a_20b (c) 40a_40b

Fig. 5: Performance on three SAVETHECITY scenarios.

right-corner, and penalty. The left-side and right-side subtasks are parallel, and so are the corner and penalty subtasks, with their parent nodes executed sequentially. For the 18_*subtasks* scenario (Figure 4(b)), the Back-field has one organizing offense subtask and eight defensive subtasks. The defense subtasks execute marking and interception in the left-side, right-side, central-area, and penalty-area areas. The offense subtask activates after gaining ball possession. The Front-field has one defense subtask and eight offense subtasks for ball control and passing in the left-side, right-side, central-area, and penalty-area.

5.2 Setting

We make extensive comparisons for the proposed DyHG-3A framework with the following four baselines. **QMIX** [23]: It is a classical multi-agent value decomposition method, which is used to learn the actions of all agents without allocating the agents. **ALMA** [5]: Agents are allocated to specific subtasks and each agent acts productively towards their assigned subtask alone. **HSD** [8]: It is a hierarchical method designed to improve teamwork among agents by teaching them different skills. **MACC** [10]: It utilizes task decomposability to learn subtask representations, enabling agents to focus on the most relevant subtasks for effective coordination. **MAAAL** [13]: It could automatically learn the allocation of agents and the subtask policy simultaneously. However, it does not model various complex relationships and exploit them. All experiments were conducted on a computer with an i7-11700F CPU, RTX3060 GPU, and 32GB of RAM. The discount factors were set to $\gamma_1 = 0.99$ and $\gamma_2 = 0.99$. Optimization was performed using Adam with a learning rate of 5×10^{-4}. For exploration, we employed the $\epsilon-$greedy strategy with ϵ linearly annealed from 1.0 to 0.05 over 50,000 time steps and then kept constant for the remainder of the training.

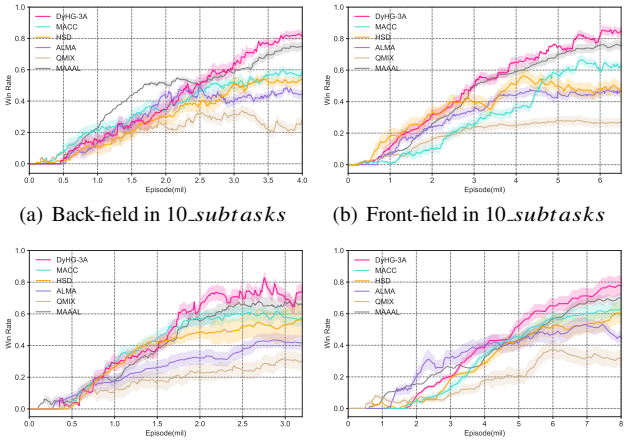
5.3 Performance

SAVETHECITY: We organize the SAVETHECITY task using a BT, as shown in Figure 3(Right). Each building node contains a firefight subtask and a repair subtask, which have different precondition nodes to determine whether the building is on fire and requires repair. Agents receive rewards for successfully extinguishing fires and repairing buildings, while penalties are imposed when buildings are destroyed or their health deteriorates. We considered three scenarios with increasing number of agents and subtasks, i.e., 7a_7b, 20a_20b and 40a_40b.

Figure 5 shows comparing results of the proposed method, DyHG-3A, with other baseline methods. As we can see, DyHG-3A achieves the best performance in all scenarios. In the 7a_7b scenario, even though MAAAL could achieve the satisfactory success rate, DyHG-3A still has a better performance. Compared to MAAAL, the success rate has increased by about 14%. In the 20a_20b and 40a_40b scenarios, it improves the success rate by about 27% and about 28%, respectively, compared with MAAAL. We notice that the performance of all methods decreases as the number of subtasks and agents increases. For the 40a_40b scenario, DyHG-3A drops by about 18%, while MAAAL, MACC, HSD and ALMA drop by about 35%, 48%, 44% and 25%, respectively, which illustrates the applicability of DyHG-3A to larger-scale scenarios. Another point to note is QMIX performs poorly in these two scenarios, the reason is that agents need to deal with different subtasks, but the algorithm based on value decomposition always tends to be similar due to the parameter sharing mechanism.

GRF: In the two GRF scenarios, i.e., 10_*subtasks* and 18_*subtasks*, we apply the proposed method for the Back-field and Front-field, respectively, to optimize player performance. This enables the players to execute the appropriate subtask at different time steps. We compared the training performance of the DyHG-3A framework with the baselines. The reward was +1 for scoring a goal and -1 for conceding a goal. The training curves are shown in Figure 6, where the x-axis represents the training set in millions, and the y-axis represents the win rate. Across all training curves, the DyHG-3A outperforms all baselines on both back-field and front-field for all scenarios.

Figures 6(a) and 6(b) show the performance of all methods in the Back-field and Front-field of 10_*subtasks* scenario, respectively. It can be seen that after sufficient training, DyHG-3A and MAAAL performed the best, However, compared to MAAAL, the win rate in the back-field has increased by about 11%, and the win rate in the front-field has increased by about 14%. It is clear to see that classical value decomposition QMIX method performs poorly due to the lack of assignment of performing agents to subtasks. Compared with ALMA, DyHG-3A achieved an about 40%



(a) Back-field in 10_$subtasks$ (b) Front-field in 10_$subtasks$

(c) Back-field in 18_$subtasks$ (d) Front-field in 18_$subtasks$

Fig. 6: Performance on two GRF scenarios.

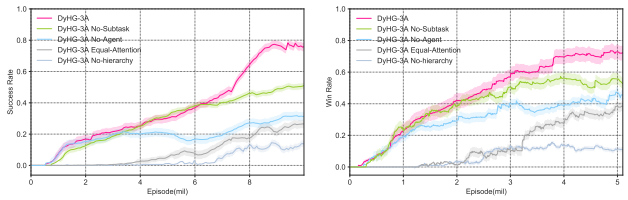
improvement in win rate. This is because ALMA focuses only on the cooperation of agents within specified subtasks, ignoring cooperation among agents across subtasks. Compared with HSD, DyHG-3A exhibits an enhanced win rate of about 30%, attributed to HSD not fully utilizing subtask information during the process of assigning skills to agents. We also observe that MACC performs worse than DyHG-3A, because although MACC considers subtask state information, while it does not consider the more complex cooperative relationship between subtasks. It is interesting to see that the performance in the Front-field is slightly better than in the Back-field, as the Front-field directly increases victories while the Back-field improves the defense win rate.

Furthermore, we conducted the same experiment on the 18_$subtasks$ scenario, with the results shown in Figures 6(c) and 6(d). As we can see, similar conclusions are also held. This further proves that different BT organization methods for subtasks have little impact on the performance of DyHG-3A, which can achieve optimal performance in both scenarios.

5.4 Ablation Studies

In this section, we conducted ablation experiments to evaluate the contribution of each component in the DyHG-3A framework. For the heterogeneous graph attribute embedding module, we separately masked the subtask nodes and agent nodes. The raw attributes of the masked nodes did not go through subgraph attribute embedding and are directly connected to the agent allocation-action module. The framework masking the subtask nodes is named DyHG-3A no-Subtask, and the framework masking the agent nodes is named DyHG-3A no-Agent. Setting equal attention weight coefficients among all nodes is achieved through Equations 3, and is denoted as DyHG-3A Equal-Attention. Additionally, we replace the hierarchical agent allocation-action learning module with a cooperative multi-agent reinforcement learning algorithm like QMIX, named DyHG-3A No-hierarchy.

We conducted ablation studies on the SAVETHECITY 20a_20b scenario and the GRF scenario 10$_{s}ubtasks$, and the results are shown in Figure 7. DyHG-3A No-hierarchy obtains the worst performance, which indicates the hierarchical agent allocation-action learning plays a key role for the performance of DyHG-3A. DyHG-3A outperformed DyHG-3A Equal-Attention, highlighting the significance of exploring diverse attention weights among subtasks, among agents, and agent-subtask. DyHG-3A no-Agent and DyHG-3A no-Subtask perform worse than DyHG-3A, emphasizing the necessity of jointly considering relationships among subtasks, among agents, and agent-subtask for DyHG-3A. Additionally, DyHG-3A no-Subtask outperformed DyHG-3A no-Agent, potentially owing to the dual roles played by agent attribute embeddings in both allocation learning and action learning.



(a) 20a_20b (b) Front-field

Fig. 7: Ablation results for SAVETHECITY and for 10_$subtasks$ in GRF.

6. Conclusion

In this paper, we propose a dynamic heterogeneous graph based agent allocation-action learning (DyHG-3A) framework, which enables the effective exploration of different relationships in multi-agent multi-task games. The DyHG-3A first constructs a dynamic heterogeneous graph to model complex heterogeneous relationships among subtasks, among agents, and between agent-subtask. To effectively extract features from the heterogeneous graph, we propose a multi-subgraph partitioning method. This method employs a graph attention network model to compute edge weights on the heterogeneous graph and aggregates node features based on these weights. Finally, we introduce a hierarchical learning framework that leverages the generated subtask and agent features for allocation learning and action learning. This framework allocates agents to different subtasks and calculates actions for each agent in real time. The experiment results show that the DyHG-3A achieves superior performance compared to the recent representative methods on both SAVETHECITY and Google Research Football full game tasks. We hope that our approach can provide valuable guidance for addressing large-scale multi-agent multi-task problems in real-world settings in the future.

Acknowledgments

This work was financially supported by the National Natural Science Foundation of China Youth Science Foundation under Grants No.61902425, No.62102444, No.62102446.

References

[1] Z.M. Kakish, K. Elamvazhuthi, and S. Berman, "Using reinforcement learning to herd a robotic swarm to a target distribution," International Symposium on Distributed Autonomous Robotic Systems, 2020.

[2] X. Tan, L. Zhou, H. Wang, Y. Sun, H. Zhao, B.C. Seet, J. Wei, and V.C. Leung, "Cooperative multi-agent reinforcement-learning-based distributed dynamic spectrum access in cognitive radio networks," IEEE Internet of Things Journal, vol.9, no.19, pp.19477–19488, 2022.

[3] Y. Liu, Y. Li, X. Xu, Y. Dou, and D. Liu, "Heterogeneous skill learning for multi-agent tasks," Advances in Neural Information Processing Systems, vol.35, pp.37011–37023, 2022.

[4] Y. Wang, Y. Wu, Y. Tang, Q. Li, and H. He, "Cooperative energy management and eco-driving of plug-in hybrid electric vehicle via multi-agent reinforcement learning," Applied Energy, vol.332, p.120563, 2023.

[5] S. Iqbal, R. Costales, and F. Sha, "Alma: Hierarchical learning for composite multi-agent tasks," arXiv preprint arXiv:2205.14205, 2022.

[6] S. Proper and P. Tadepalli, "Solving multiagent assignment markov decision processes," Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1, pp.681–688, 2009.

[7] T. Wang, T. Gupta, A. Mahajan, B. Peng, S. Whiteson, and C. Zhang, "Rode: Learning roles to decompose multi-agent tasks," arXiv preprint arXiv:2010.01523, 2020.

[8] J. Yang, I. Borovikov, and H. Zha, "Hierarchical cooperative multi-agent reinforcement learning with skill discovery," arXiv preprint arXiv:1912.03558, 2019.

[9] B. Liu, Q. Liu, P. Stone, A. Garg, Y. Zhu, and A. Anandkumar, "Coach-player multi-agent reinforcement learning for dynamic team composition," International Conference on Machine Learning, pp.6860–6870, PMLR, 2021.

[10] L. Yuan, C. Wang, J. Wang, F. Zhang, F. Chen, C. Guan, Z. Zhang, C. Zhang, and Y. Yu, "Multi-agent concentrative coordination with decentralized task representation," IJCAI, 2022.

[11] B.P. Gerkey and M.J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," The International Journal of Robotics Research, vol.23, pp.939 – 954, 2004.

[12] N. Carion, N. Usunier, G. Synnaeve, and A. Lazaric, "A structured prediction approach for generalization in cooperative multi-agent reinforcement learning," Advances in neural information processing systems, vol.32, 2019.

[13] X. Li, Y. Li, J. Zhang, X. Xu, and D. Liu, "A hierarchical multi-agent allocation-action learning framework for multi-subtask games," Complex & Intelligent Systems, pp.1–11, 2023.

[14] X. Wang, D. Bo, C. Shi, S. Fan, Y. Ye, and S.Y. Philip, "A survey on heterogeneous graph embedding: methods, techniques, applications and sources," IEEE Transactions on Big Data, 2022.

[15] M. Chen, C. Huang, L. Xia, W. Wei, Y. Xu, and R. Luo, "Heterogeneous graph contrastive learning for recommendation," Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining, pp.544–552, 2023.

[16] L. Gao, H. Wang, Z. Zhang, H. Zhuang, and B. Zhou, "Hetinf: Social influence prediction with heterogeneous graph neural network," Frontiers in Physics, vol.9, p.787185, 2022.

[17] Z. Li, Y. Zhao, Y. Zhang, and Z. Zhang, "Multi-relational graph attention networks for knowledge graph completion," Knowledge-Based Systems, vol.251, p.109262, 2022.

[18] H.C. Yi, Z.H. You, D.S. Huang, and C.K. Kwoh, "Graph representation learning in bioinformatics: trends, methods and applications," Briefings in Bioinformatics, vol.23, no.1, p.bbab340, 2022.

[19] C. Zhang, D. Song, C. Huang, A. Swami, and N.V. Chawla, "Heterogeneous graph neural network," Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pp.793–803, 2019.

[20] X. Yang, M. Yan, S. Pan, X. Ye, and D. Fan, "Simple and efficient heterogeneous graph neural network," Proceedings of the AAAI Conference on Artificial Intelligence, pp.10816–10824, 2023.

[21] 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," International conference on machine learning, pp.5887–5896, PMLR, 2019.

[22] P. Sunehag, G. Lever, A. Gruslys, W.M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J.Z. Leibo, K. Tuyls, *et al.*, "Value-decomposition networks for cooperative multi-agent learning," arXiv preprint arXiv:1706.05296, 2017.

[23] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," International conference on machine learning, pp.4295–4304, PMLR, 2018.

[24] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of behavior trees in robotics and ai," Robotics and Autonomous Systems, vol.154, p.104096, 2022.

[25] M. Kartašev, J. Saler, and P. Ögren, "Improving the performance of backward chained behavior trees using reinforcement learning," arXiv preprint arXiv:2112.13744, 2021.

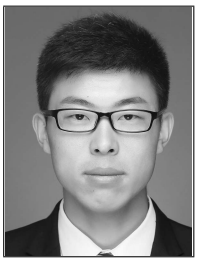
[26] L. Li, L. Wang, Y. Li, and J. Sheng, "Mixed deep reinforcement learning-behavior tree for intelligent agents design.," ICAART (1), pp.113–124, 2021.

[27] F. Rovida, B. Grossmann, and V. Krüger, "Extended behavior trees for quick definition of flexible robotic tasks," 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp.6793–6800, IEEE, 2017.

[28] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, *et al.*, "Graph attention networks," stat, vol.1050, no.20, pp.10–48550, 2017.

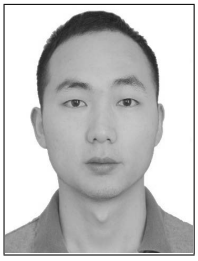
[29] K. Kurach, A. Raichuk, P. Stańczyk, M. Zajac, O. Bachem, L. Espeholt, C. Riquelme, D. Vincent, M. Michalski, O. Bousquet, *et al.*, "Google research football: A novel reinforcement learning environment," Proceedings of the AAAI Conference on Artificial Intelligence, pp.4501–4510, 2020.

[30] B. Liu, Q. Liu, P. Stone, A. Garg, Y. Zhu, and A. Anandkumar, "Coach-player multi-agent reinforcement learning for dynamic team composition," International Conference on Machine Learning, pp.6860–6870, PMLR, 2021.



Xianglong Li received the B.S. degree Computer Science and technology in from the Hubei University of Technology, in 2017. And received the M.S. degree software engineering in from the Tianjin University. He is currently pursuing the Ph.D. degree with the Academy of Military Sciences. His research interests include multi-agent reinforcement learning algorithms and complex system.

Yuan Li received the B.S. degree in computer science from the National University of Defense Technology, China, and the Ph.D. degree in network optimization from Lund University, Sweden, in 2008 and 2015, respectively. His research interests include network modeling, algorithm design, integer programming, and other combinatorial methods with applications in communication networks.



Jieyuan Zhang received the Ph.D degrees in computer science from the National University of Defense Technology, in 2019. His research interests include include reinforcement learning, mechanics, and boundary element method.

Xinhai Xu received the Ph.D degrees in computer science from the National University of Defense Technology, in 2012. His research interests include artificial intelligence algorithm, simulation, and parallel computing.

Donghong Liu received the Ph.D degrees in computer science from the National University of Defense Technology, in 2011. His research interests interests include distributed computing, intelligence algorithm and information systems.