Applied Intelligence
https://doi.org/10.1007/s10489-020-02065-9

Multi-agent deep reinforcement learning with type-based hierarchical group communication

Hao Jiang[1] · Dianxi Shi[2,3] · Chao Xue[2,3] · Yajie Wang[1] · Gongju Wang[2] · Yongjun Zhang[2]

Accepted: 4 November 2020

Abstract
Real-world multi-agent tasks often involve varying types and quantities of agents. These agents connected by complex interaction relationships causes great difficulty for policy learning because they need to learn various interaction types to complete a given task. Therefore, simplifying the learning process is an important issue. In multi-agent systems, agents with a similar type often interact more with each other and exhibit behaviors more similar. That means there are stronger collaborations between these agents. Most existing multi-agent reinforcement learning (MARL) algorithms expect to learn the collaborative strategies of all agents directly in order to maximize the common rewards. This causes the difficulty of policy learning to increase exponentially as the number and types of agents increase. To address this problem, we propose a type-based hierarchical group communication (THGC) model. This model uses prior domain knowledge or predefine rule to group agents, and maintains the group's cognitive consistency through knowledge sharing. Subsequently, we introduce a group communication and value decomposition method to ensure cooperation between the various groups. Experiments demonstrate that our model outperforms state-of-the-art MARL methods on the widely adopted StarCraft II benchmarks across different scenarios, and also possesses potential value for large-scale real-world applications.

Keywords Multi-agent reinforcement learning · Group cognitive consistency · Group communication · Value decomposition

1 Introduction

Many real-world multi-agent tasks contain scenarios in which an agent is required to deal with varying numbers and types of cooperative agents, antagonist enemies, or other entities. For example, consider the game of soccer. Individual players (i.e., agents) have varying capabilities and specializations, meaning that players have different target tasks or attributes. Players can also be categorized into types, such as defenders, forwards, goalkeeper, etc.; moreover, any game of soccer may require these different types of players to cooperate, yielding a rich array of possible strategies. However, the types of interaction relationships between different types of players also differ, while players of the same type may exhibit similar behaviors. For example, if a "breakaway" situation occurs (in which the two forwards with the ball have passed the defense), the interaction between the two forwards takes the form of the players cooperating to beat the goalkeeper in order to score (see Fig. 1a). Moreover, if the situation of a "long pass" occurs, in which the goalkeeper wants to pass the ball to the center forwards, the interaction between the goalkeeper and the center forwards takes the form of the goalkeeper trying to move the ball as close as possible to the center forwards, while the center forwards aim to avoid the opposing players and catch the ball accurately (see Fig. 1b). Therefore, random combinations of these player types will result in particularly complex interaction types; furthermore, the number of agent interaction types will increase exponentially as the number of agent types increases. This will pose a significant challenge to the policy learning process, particularly in a large-scale multi-agent system containing varying amounts and types of agents.

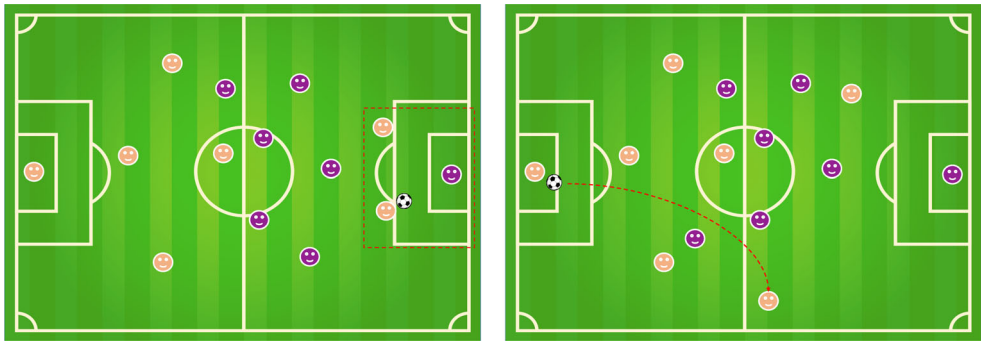
✉ Dianxi Shi
dxshi@nudt.edu.cn

[1] College of Computer, National University of Defense Technology, Changsha, China

[2] Artificial Intelligence Research Center, National Innovation Institute of Defense Technology, Beijing, China

[3] Tianjin Artificial Intelligence Innovation Center, Tianjin, China

Published online: 16 January 2021



(a) Breakaway scenario in soccer (b) Long pass scenario in soccer

Fig. 1 **a** Agents (two forwards) in the red square ignore the other agents outside of this region and interact with each other. **b** Agents (goalkeeper and center forwards) cooperate to complete the long pass (red line)

Inspired by these observations, the present work focuses on simplifying the policy learning process and reducing the complexity of interactions between agents. Simplifying the learning process is a crucial research area [6, 23]. Existing works on large-scale multi-agent reinforcement learning [5, 47] have tended to focus on loosely coupled multi-agent systems, as well as adopting techniques such as game abstraction and knowledge transfer in order to help accelerate multi-agent reinforcement learning [22, 27, 48]. However, in a large-scale multi-agent environment, agents are usually related to other agents rather than being independent, meaning that the previously learned single-agent knowledge has limited use. Some works have focused on achieving game abstraction through predefined rules (e.g., the distance or communication between agents) [19, 26, 47]. However, different types of interactions occur between agents with the same types and agents with different types. It is not easy to describe and simplify the complex interactions between agents in the system in terms of their distance or communication. In this work, we group agents according to prior domain knowledge (e.g. agent type, agent task) or predefine rule (e.g. location, health) in order to reduce the complexity of interactions between agents, and further propose a layered approach to facilitate the learning of cooperation strategies.

The key to the learning of cooperative strategy in large-scale multi-agent systems involves the learning of interactions between agents [23]. Recent works have used the "learning for consensus approach" [17, 49] and "learning to communicate approach" [10, 19, 35] to learn the importance distribution of all agents i, along with other agents, in anticipation of directly learning the cooperation strategy of agent i with all other agents. In these two types of methods, however, the strategy learned by agent i is based on information about all other agents with the same weight. In other words, these methods cannot reduce the complexity of the interaction between agents, and also cannot ignore unrelated agents in order to simplify the process of learning cooperative strategies.

In this work, we propose a model, called **T**ype-based **H**ierarchical **G**roup **C**ommunication (THGC), which is trained end-to-end by backpropagation. This model enables agents to learn different types of agent interactions in a partially observable distributed environment. It further simplifies the learning process of large-scale multi-agent cooperation strategies and reduces the difficulty associated with strategy learning. The major contributions of this work can be summarized as follows:

1. We use prior domain knowledge or predefine rule to group agents. Based on this, we propose a novel knowledge-sharing method to promote group cognitive consistency.
2. To coordinate the cooperation of various groups, we introduce a group communication method. The group communication can share part of the knowledge required for cooperation through information transmission when the cognitive of each group is inconsistent.
3. We introduce the method of value decomposition to further strengthen the cooperation between various groups and establish the relationship between the all agents' joint action-value functions (Q_{tot}) and the group agents' joint action-value functions (Q_{te}).

Experiments are conducted in a challenging multi-agent collaborative environment with a high-dimensional state space: namely, a variety of units in the StarCraft II battle game forming a team to defeat their enemies. Results show that THGC outperforms state-of-the-art methods and demonstrates its potential value for large-scale real-world applications.

2 Related work

Many natural systems have the phenomenon of group cooperation, such as bees [18], ants [13], and humans [3]. In these systems, the attribute of members is closely related to the division of labor and is crucial to the improvement

of labor efficiency. Many multi-agent systems are inspired by these natural systems. They decompose the task, make agents with the same attribute specialize in certain sub-tasks, and thus reduce the design complexity [8, 9, 46]. These methodologies are designed for tasks with a clear structure, such as software engineering [2]. Unlike them, in this work, we focus on how to use predefined attributes and related goals to group agents so that there are more interactions between agents with the same attributes or goals, thereby reducing the complexity of the interaction.

Multi-agent deep reinforcement learning has witnessed vigorous progress in recent years. MADDPG [25] and COMA [12] both adopt the framework of centralized training with decentralized execution, which enables the policies to use extra information at training time. This is a simple extension of actor-critic policy gradient methods [38, 41], in which the critic is augmented with additional information about other agents, while the actor only has access to local information. MAAC [17] employs an attention network and graph neural network to model a centralized critic, from which decentralized actors are derived using a soft actor-critic mechanism. This mechanism is able to dynamically select which agents should be attended to at each time step. DGN [19] applies a graph convolutional network to model a centralized Q-function for each agent using a deep Q-network [28].

Another line of research focuses on communication among agents. BiCNet [15] uses a recurrent neural network (RNN) as the communication channel to connect each individual agent's policy and value networks. BiCNet is able to handle real-time strategy games, such as StarCraft micro-management tasks. IC3Net [36] actively selects and masks messages from other agents during communication by applying a gating function in the message aggregation step. ATOC [20] and TarMAC [10] enable agents to learn when to communicate and who to send messages to, respectively, through the use of an attention mechanism. These communication approaches prove that communication is helpful in facilitating cooperation, which can in turn help each agent to use more extensive information in its individual decision making. However, these approaches require both a separate communication channel and a well-established communication environment for message exchange. Furthermore, all of these approaches construct a communication channel that directly connects all agents at the same level, which is based on the principle that the interaction type of agent i will be the same as all other agents. This means that agent i is required to deal with the cooperation information sent by all other agents. Accordingly, this type of communication is unable to achieve good collaboration in large-scale MARL [33].

Moreover, some other works have addressed the topic of generalization and transfer across multi-agent scenarios. Carion et al. [4] devise an approach for assigning agents to tasks, assuming the existence of low-level controllers to carry out the tasks, and show that it can scale to much larger scenarios than those seen in training. While improving generalization to larger tasks, this method assumes the existence of low-level action policies that can execute pre-defined sub-tasks; by contrast, we aim to learn low-level action policies from experience with no pre-defined sub-tasks. Our work considers environments with only low-level actions, where we must learn policies using these actions. Long et al. [24] introduce an attention-based model for MARL as well as an evolution-based method for automated curriculum learning to enable scaling up to larger scenarios. This approach is also tested on particle environments. Finally, Wang et al. [43] propose using graph neural network models to enable the sharing of experience across scenarios. They test several methods of curriculum learning in order to improve performance on large-scale StarCraft II environments with homogeneous agents.

While several of these works introduce attention or graph neural network-based models for MARL in dynamic scenarios, they typically focus on curriculum learning across a limited number of similar scenarios, often with homogenous agents. These curriculum learning techniques generally learning smaller scenarios in order to initialize training in larger scenarios and do not require policies to simultaneously perform well on several diverse scenarios. By contrast, our work focuses on generalization across a variety of scenarios with heterogeneous agent types, and we test on strategically complex environments.

3 Background

3.1 Decentralized partially observable Markov game

Partially-observable Markov decision process (POMDP) have been widely used to model a single agent's dynamic decision making under stochastic environment. In this work, we consider decentralized partially observable Markov decision process (Dec-POMDP) [30], an extension of POMDP to multi-agent system, to model the interaction of multiple agents with the environment. Dec-POMD is formally defined as a tuple $G =< S, U, P, r, Z, O, n, \gamma >$. $s \in S$ describes the true state of the environment. At each time step, each agent $i \in A \equiv (1, \ldots, n)$ chooses an action $u^i \in U$ which forms the joint action $\mathbf{u} \in \mathbf{U} \equiv U^n$ This causes a transition on the environment according to the state transition function $P(s'|s, \mathbf{u}) : S \times \mathbf{U} \times S \mapsto [0, 1]$. All agents share the same reward function $r(s, \mathbf{u}) : S \times U \mapsto R$ and $\gamma \in [0, 1)$ is the discount factor. We consider partially observable settings, where each agent does not have access to the full state and instead samples observations $z \in Z$

according to observation function $O(s,i) : S \times A \mapsto Z$. The action-observation history for an agent i is $\tau^i \in T \equiv (Z \times U)^*$ on which it can condition its policy $\pi^i(\mu^i|\tau^i) : T \times U \mapsto [0,1]$. The joint policy π is based on action-value function: $Q^\pi(s_t, \mathbf{u}_t) = \mathbb{E}_{s_{t+1:\infty}, \mathbf{u}_{t+1:\infty}}[R_t|s_t, \mathbf{u}_t]$, where $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the discounted return. The goal of the problem is to find the optimal action value function Q^*. During centralised training, the learning algorithm has access to the action-observation histories of all agents and the full state. However, each agent can only condition on its own local action-observation history τ^i during decentralised execution. For such methods factoring action values over agents, we represent the individual agent utilities by Q_i, $i \in A$. An important concept for such methods is decentralisability also called Individual-Global-Max (IGM) [37], which asserts that $\exists Q_i$, such that $\forall s, \mathbf{u}$:

$$\arg\max_{\mathbf{u}} Q^*(s,\mathbf{u}) = \begin{pmatrix} \arg\max_{u^1} Q_1(\tau^1, u^1) \\ \vdots \\ \arg\max_{u^n} Q_n(\tau^n, u^n) \end{pmatrix} \tag{1}$$

3.2 Deep Q-learning

Q-Learning and DQN [28] are popular methods in reinforcement learning and have been previously applied to multi-agent settings. Q-Learning makes use of an action-value function for policy π as $Q^\pi(s,a) = \mathbb{E}[R|s_t = t, a_t = a]$. This Q function can be recursively rewritten as $Q^\pi(s,a) = \mathbb{E}_{s'}[r(s,a) + \gamma \mathbb{E}_{a' \sim \pi}[Q^\pi(s',a')]]$. DQN learns the action-value function Q^* corresponding to the optimal policy by sampling batches of b transitions from the replay memory and minimizing the squared *TD error*:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[\left(Q^*(s,u;\theta) - y_i^{DQN} \right)^2 \right],$$
$$where \quad y^{DQN} = r + \gamma \max_{u'} \widetilde{Q}^*(s',u',\theta'). \tag{2}$$

where $\widetilde{Q}$ is a target Q function. θ' are the parameters of a target network that are periodically copied from θ and kept constant for a number of iterations. A crucial component of stabilizing DQN is the use of an experience replay buffer D containing tuples (s,u,r,s').

Q-Learning can be directly applied to multi-agent settings by having each agent i learn an independently optimal function Q_i. However, because agents are independently updating their policies as learning progresses, the environment appears non-stationary from the view of anyone agent, violating Markov assumptions required for convergence of Q-learning.

3.3 QMIX

QMIX is a value-based method that learns a monotonic approximation Q_{tot} [45] for the joint action-value function. QMIX factors the joint action-value Q_{tot} into a monotonic nonlinear combination of individual Q-value Q_i of each agent which learns via a*utility network*. A *mixing network* with nonnegative weights is responsible for combining the agent's utilities for their chosen actions u^i into $Q_{tot}(\mathbf{s}, \mathbf{u})$. This nonnegativity ensures that:

$$\frac{\alpha Q_{tot}(s,\mathbf{u})}{\alpha Q_i(s,u^i)} \geq 0 \tag{3}$$

Equation (3) in turn guarantees (1). This decomposition allows for an efficient, tractable maximization as it can be performed in $O(n|U|)$ time as opposed to $O(|U|^n)$. Additionally, it allows for easy decentralization as each agent can independently perform an $argmax$. During learning, the QMIX agents use ϵ-greedy exploration over their individual utilities to ensure sufficient exploration For VDN [40] the factorization is further restrained to be just the sum of utilities:

$$Q_{vdn}(s,\mathbf{u}) = \sum_i Q_i(s,u^i) \tag{4}$$

3.4 Graph attention network

The graph attention getwork (GAT) [42] is an effective model to process structured data that is represented as a graph. The GAT has proposed a way to compute the node-embedding vector of graph nodes by aggregating node embeddings h_j from neighboring nodes $j \in \mathcal{N}_i$ that are connected to the target node i as $h_i' = \sigma(\sum_{j \in \mathcal{N}_i} \alpha_{i,j} \mathbf{W} h_j)$. The attention weight $\alpha_{i,j} = softmax_j(e_{ij})$, where $e_{i,j} = a(\mathbf{W}h_i, \mathbf{W}h_j)$, quantifies the importance of node j to node i in computing node-embedding value h_i'.

4 THGC method

In this section, we introduce our proposed novel model THGC and further provide a comprehensive description. THGC is based on a deep reinforcement learning model. It can achieve good cooperation in large-scale multi-agent systems characterized by a large number of agents with different tasks and complex between-agent interactions. The complete THGC learning framework is presented in Fig. 3. First, we use prior domain knowledge or predefine rule to group agents in a multi-agent environment. Second, knowledge-sharing is proposed to determine the agent's cognition regarding the group environment. This approach shares the low-level knowledge to achieve group cognition

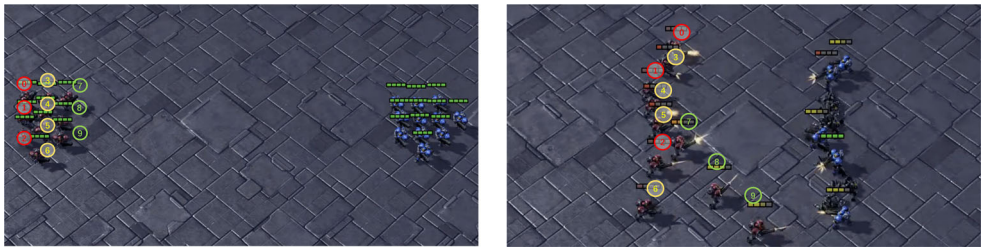
Algorithm 1 THGC.

Initialise θ, the parameters of the mixing network, agent networks and hypernetwork
Set the learning rate α and replay buffer $\mathcal{D}$
for episode = 1 to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration, and receive state information s
  for $t = 1$ to max-episode-length **do**
    for each agent i in agent group k
    $\tau_t^i = \tau_{t-1}^i \cup (o_t, u_{t-1})$
    $u_i = \mu(o_i; \theta_k) + \mathcal{N}_t$
    Execute actions $\mathbf{u}$ and receive reward r_t and new observation s_{t+1}
    Store $(s_t, \mathbf{u}_t, r_t, s_{t+1})$ in replay buffer $\mathcal{D}$
    for each time step t in each episode in batch b **do**
      random batch of epochs b from $\mathcal{D}$
      for $k = 1$ to K for agent group C^k **do**
        for agent $i = 1$ to $|C^k|$ in group C^k **do**
          $H_i^k = f(LSTM_i(h_i^k), z_i^k) \quad where \quad h_i^k = f(o_i^k, u_i^k); z_i^k = \sum_{j \neq i} \alpha_{i,j} h_j^k$
        end for
        $V^k = \sum_{i=1}^{|C^k|} H_i^k$
      end for
      $Q_{te}^k(\boldsymbol{\tau}^k, \mathbf{u}^k) = f(V^k, \widetilde{z}^k) \quad where \quad \widetilde{z}^k = sum_{k \neq l} \alpha_{k,l} V^l$
      $Q_{tot} = Mixing{-}network(Q_{te}^1, \ldots, Q_{te}^k); Hypernetwork(s_t; \theta))$
      Calculate target Q_{tot} using $Mixing{-}network$ with $Hypernetwork(s_t; \theta'))$
    end for
    $\triangle Q_{tot} = y^{tot} - Q_{tot}$ // Eq. 11
    $\triangle \theta = \nabla_\theta (\triangle Q_{tot})^2$
    $\theta = \theta - \alpha \triangle \theta$
  end for
end for

consistency, thereby achieving collaboration within the group. Third, we use a group communication and value decomposition method in order to achieve collaboration between the groups. Finally, we synthesize all of these above-mentioned components into a synergistic learning framework.

4.1 Agent grouping

In many natural systems, such as bees [18] and ants [13], the type of members is closely related to the division of labor and is crucial to the improvement of labor efficiency. According to the types or tasks of the members, bees can be divided into three groups: queen bees, worker bees, and drones. Inspired by these natural systems, in our multi-agent system, we cluster all of the agents into distinct groups using prior domain knowledge (e.g. agent type, agent task). There are N agents with k kind of agent types or tasks in our multi-agent system. The N agents can be clustered into distinct groups C^k, such that $N = \sum_{k=1}^{K} |C^k|$. When a multi-agent system has only a single type or one task of agents ($k = 1$), all agents can be categorized into a single group. Moreover, if agents in the multi-agent system have multiple types or tasks ($k = K$). Here, we can cluster the agents into K groups. Consider the example presented in Fig. 1, where there are seven players (i.e., agents) with varying capabilities and specializations, specifically three defenders, three forwards, and one goalkeeper. We can cluster these agents into three groups according to their capabilities. The three defenders form one group, the three forwards form another group, and the one goalkeeper forms a third. Also, we can divide the agent into k groups according to location or health. Consider the example presented in Fig. 2, where there are 10 allied Marines facing 11 enemy Marines (10m_vs_11m). At the beginning of training, all agents have the same health. At this time, we mainly group the agents according to their positions (see Fig. 2a). As the training progresses, the health of the agent becomes inconsistent. At this time, we mainly group the agents according to their health (see Fig. 2b). In this work, we assume that the agents can be easily clustered into K groups using prior knowledge about the agents, which implies that THGC utilizes enhanced relative inductive biases regarding the group relationships.



(a) Group by location (b) Group by health

Fig. 2 **a** The same color indicates the same group. **b** The blue agent has the maximum health, while the red ones are minimum

4.2 Knowledge sharing

In large-scale multi-agent systems, agents with the same type or task will interact more and exhibit more similar behavior. This implies the necessity of stronger collaboration between agents with the same type or task. Agents that maintain consistent cognitions about their environments are vital if effective cooperation is to be achieved [1, 26, 31]. In this context, *cognition* refers to the agent's understanding of the local environment; this includes not only the observations of all agents in the same group, but also the high-level knowledge extracted from these observations (e.g., the knowledge that has been learned through deep neural networks). Mao et al. [26] first proposed the method of neighborhood cognitive consistency (NCC). To implement NCC, these authors decompose the high-level knowledge into both an agent-specific cognitive representation and a neighborhood-specific cognitive representation. They then assume that each neighborhood contains a true hidden cognitive variable, after which they make all neighboring agents learn to align their neighborhood-specific cognitive representations to this true hidden cognitive variable using variational inference. MADDPG [25] uses a centralized critic capable of obtaining all agent actions and global observations in order to ensure that all agents receive the same environmental information. When all agents observe the same environmental information, it can be concluded that these agents have the same cognitive consistency. Adopting a different approach, in this work, we use the method of knowledge sharing to achieve group cognitive consistency.

As shown in Fig. 3, we divide the N agents into K groups. The agents i and j in same group $k = 1, \ldots, K$ are represented as C_i^k and C_j^k. Agent i has the local observation o_i. Under partially observable conditions, agent i computes

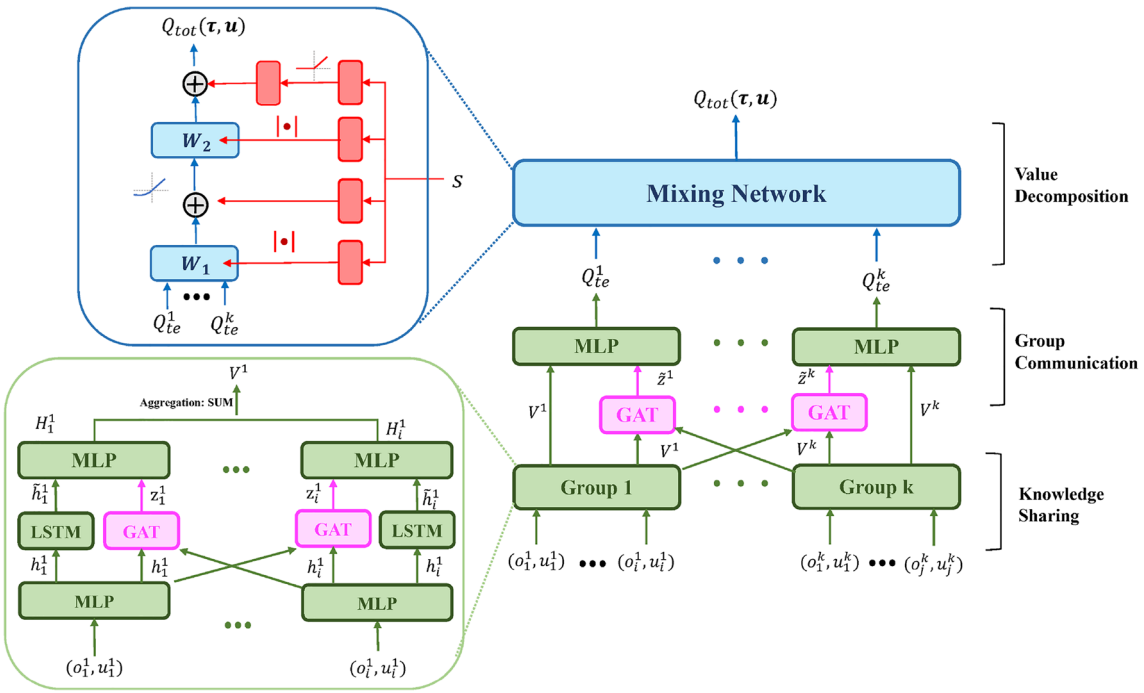


Fig. 3 Overview of THGC

the different node-embedding vectors h_i^k. The h_i^k that is encoded by multilayer perceptron (MLP) [32] is a low-level cognition vector and contains only agent i's local observation, which can also be seen as an agent-specific cognitive representation. MLP is a class of feedforward artificial neural network. Its multi-layer structure and nonlinear activation function enable it to distinguish data that cannot be linearly separated, and our environmental information is a kind of nonlinear data. C_i^k and C_j^k have different agent-specific cognitive representations. In order to ensure that agents in the same group have consistent cognitions of the environment, we aggregate all h_i^k within the group k as input, then use GAT to further extract the high-level cognition vectors H_i^k as follows:

$$H_i^k = f(LSTM_i(h_i^k), z_i^k) \quad where \quad h_i^k = f(o_i^k, u_i^k) \tag{5}$$

In (5), $f(\cdot)$ is a multilayer perceptron for embedding. We define the attention embedding z_i^k by a weighted sum of the each agent's low-level cognition vector h_j^k for $j \neq i$:

$$z_i^k = \sum_{j \neq i} \alpha_{i,j} h_j^k \tag{6}$$

The coefficient $\alpha_{i,j}$ is computed by:

$$\begin{aligned} \alpha_{i,j} &= softmax_j(att(h_i^k, h_j^k)) \\ &= \frac{exp(\sigma(W^T \cdot [h_i^k || h_j^k]))}{\sum_{j \neq i} exp(\sigma(W^T \cdot [h_i^k || h_i^k])} \end{aligned} \tag{7}$$

where σ denotes the activation function, $||$ denotes the concatenate operation and W is parameter to learn.

In order to calculate the high-level cognitive vector H_i^k for agent i, first, we use the agent i's low-level cognition vectors h_i^k at the current time and the previous time as the input of long short-term memory networks (LSTM) [16], and output the agent i's cognition vectors $\widetilde{h}_i^k$. Second, we aggregate all the low-level cognition vectors $h_1^k, \ldots, h_i^k$ as the input of GAT network, and output the attention embedding z_i^k. Finally, we use the cognition vectors $\widetilde{h}_i^k$ and attention embedding z_i^k as the input of MLP, and output of the high-level cognitive vector H_i^k. Since LSTM can learn long-term dependencies, it can more accurately calculate the cognitive vector of the agent i based on the historical trajectory of the agent i. Since we represent the low-level cognition vectors of other agents with a weighted mean z_i^k from (6), we can model the interactions between agent i and an arbitrary number of other agents within the same group, which allows us to easily deal with changes in the number of agents in our group. H_i^k includes the low-level cognitions of all agents in group k. In other words, the low-level cognition vectors h_i^k are shared within the same group of agents, such that agent i can obtain the other agents' agent-specific cognitive representation in group k. Accordingly, through knowledge sharing, all agents in the group k have consistent cognition of the environment.

4.3 Group communication

Group cognitive consistency can achieve coordination within the group, cooperation between groups is also necessary. To facilitate collaboration between the various groups, we allow between-group communication using the communication channel. In this work, we introduce the attention mechanism to act as the communication channel. Unlike CommNet [39] and BiCNet [15], where all agents communicate with each other all the time, our attention unit enables dynamic communication among agents only when necessary. This is much more practical, because in real-world applications communication is restricted by bandwidth and/or range and incurs additional cost, and thus it may not be possible or cost too much to maintain full connectivity among all the agents. On the other hand, dynamic communication can keep the agent from receiving useless information compared to full connectivity.

As shown in Fig. 3, we aggregate the high-level cognition vector $H_1^k, \ldots, H_i^k$ of group k as $V^k = \sum_{i=1}^{|C_i^k|} H_i^k$. $V^1 \ldots, V^k$ is used as the input of GAT network, and output $\widetilde{z}^k = sum_{k \neq l} \alpha_{k,l} V^l$, where $l \in K$ and coefficient $\alpha_{k,l}$ has a similar calculation method as (7). Next, we use the V^k and $\widetilde{z}^k$ as the input of MLP to calculate the joint value Q_{te}^k, as follows:

$$Q_{te}^k(V^k) = f(V^k, \widetilde{z}^k) \tag{8}$$

In (8), Q_{te}^k is the joint value of group k, while $Q_{te}^k(\boldsymbol{\tau}^k, \mathbf{u}^k) = Q_{te}^k(V^k)$, Here $\boldsymbol{\tau}^k$ and $\mathbf{u}^k$ represent the joint trajectory and the joint action respectively of all agents in group k. The GAT can use the backpropagation through time (BPTT) [44] to compute the backward gradients.

4.4 Value decomposition

In order to further strengthen the cooperation between various groups, we introduce the method of value decomposition. Here, Q_{tot} is used to represent the joint value of all agents. Moreover, inspired by QMIX [45], we need to ensure that a global arg max performed on Q_{tot} yields the same result as a set of each group's arg max operations performed on each Q_{te}^k. Equation (1) can thus be rewritten as:

$$\arg\max_{\mathbf{u}} Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \begin{pmatrix} \arg\max_{u^1} Q_{te}^1(V^1) \\ \vdots \\ \arg\max_{u^k} Q_{te}^k(V^k) \end{pmatrix} \tag{9}$$

where $\boldsymbol{\tau}$ and $\mathbf{u}$ represent the joint trajectory and joint action of all agents, respectively. In addition, Q_{tot} and Q_{te}^{k} observe the monotonicity assumption [45], which can be generalised from (10). Recalling (3), the relationship between Q_{tot} and Q_{te}^{k} can be described as follows:

$$\frac{\alpha Q_{tot}}{\alpha_{te}^{k}} \geq 0 \tag{10}$$

To enforce (10), THGC represents Q_{tot} using a mixing network and a set of hyper-networks [14]. As illustrated in Fig. 3, the mixing network is a feed-forward neural network that takes the BiLSTM network outputs as input and mixes them monotonically, thereby producing the values of Q_{tot}. The weights (but not the biases) of the mixing network are restricted to ensure they are non-negative in order to enforce the monotonicity constraint in (10). This allows the mixing network to approximate any monotonic function with an arbitrary degree of closeness [11]. Figure 3 illustrates the mixing network and the hyper-networks. The weights of the mixing network are produced by separate hyper-networks; each hyper-network takes the state s as input, then generates the weights of one layer of the mixing network. Unlike QMIX [45], each of our hyper-networks consist of two fully connected layers with ELU [7] nonlinearity rather than ReLU [29]. It should be noted here that a ReLU would not be a good choice, since a negative input to the mixing network is likely to remain negative (depending on the biases) and would thus be zeroed out by the ReLU, leading to no gradients for all agent networks. Accordingly, we opt to use an ELU in our model.

THGC is trained end-to-end to minimize the following loss:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[\left(Q_{tot}\left(\boldsymbol{\tau}, \mathbf{u}, s, \theta\right) - y^{tot} \right)^2 \right], \tag{11}$$

where b is the batch size of transitions sampled from the replay buffer, $y^{tot} = r + \gamma \max_{\mathbf{u}'} Q_{tot}(\boldsymbol{\tau}', \mathbf{u}', s', \theta')$ and θ' are the parameters of a target network as in DQN. Moreover, (11) is analogous to the standard DQN loss of (2). Since (9) holds, moreover, we can perform the maximization of Q_{tot} in a time period that increases linearly with the number of agents (as opposed to scaling exponentially in the worst case).

In Algorithm 1, we outline the pseudocode for the specific implementation of THGC we use for all our experiments. In each episode, we initialize the parameters θ, and each agent i uses ϵ-greedy exploration or its policy to select and execute the actions u_i. The agents then receive the new observation s_{t+1} and reward r_t. We store the quad information of all agents $(s_t, \mathbf{u}_t, r_t, s_{t+1})$ in the replay buffer $\mathcal{D}$. When enough experience is stored in this buffer, we sample a random minibatch of samples for training. First, we use the knowledge sharing method to aggregate the information of all agents in each group into a high-level cognitive vector H_i^k. Subsequently, we use the GAT to calculate a joint value Q_{te}^{k} for each group. Finally, we calculate the joint value Q_{tot} for all agents using the mixing network. Furthermore, we can use the joint value Q_{tot} to calculate the gradient of the agent i's expected return through backpropagation and use Adam optimizer to update its strategy. At the end of each episode, we update the target network parameters for each agent i. The initialization and training process is subsequently repeated until victory is achieved.

5 Experiments

5.1 Settings

In this section, we comprehensively evaluate the proposed THGC method in the game of StarCraft II based on the learning environment mini-game settings in SMAC. We compare the THGC method with a number of state-of-the-art MARL methods. StarCraft II is a popular real-time strategy game and it has been studied under MARL settings. In the experiments, we consider symmetric battle games in StarCraft II, where both single type agents and mixed type agents are considered.

Specifically, the considered scenarios contain 1C2S3Z, 2C3S5Z, 3C5S7Z, MMM-3, MMM-4, and MMM-5. We briefly introduce these map scenarios in Table 1 and illustrate three map scenarios of MMM-3, MMM-4, and MMM-5 in Fig. 4. MMM-5 (see Fig. 4c), includes 10 Terran units, specifically 2 Marines, 2 Medivacs, 2 Marauders, 2 Vikings, and 2 Ghosts. The attributes of each type of unit are described in Table 2. Each type of units in the map scenario have their own unique skills and plays different functions. The agents (i.e. allied units) can achieve victory only when each unit performs to the best of its ability and cooperates with the actions of other units.

Each agent is described by several attributes including the health point (HP), weapon cooling down (CD), shield (for 1C2S3Z, 2C3S5Z, and 3C5S7Z), unit type, last action and the relative distance of the observed units. The enemy unit is described in the same way except that CD is excluded. The partial observation of an agent is composed by the attributes of the units, including both the agents and the enemy units, shown up within its view range that is a circle with a certain radius. The action space contains 4 move directions, attack actions where is the fixed maximum number of the enemy units in a map, stop and none-operation. The input dimension and the output action dimension are fixed with a certain ordering over the agents and enemy units. Dead enemy units will be masked out from the action space to ensure the executed action is valid. At

Table 1 Map scenarios

Name	Ally Units	Enemy Units	Type
1C2S3Z	1 Colossi & 2 Stalkers & 3 Zealots	1 Colossi & 2 Stalkers & 3 Zealots	Symmetric Heterogeneous
2C3S5Z	2 Colossi & 3 Stalkers & 5 Zealots	2 Colossi & 3 Stalkers & 5 Zealots	Symmetric Heterogeneous
3C5S7Z	3 Colossi & 5 Stalkers & 7 Zealots	3 Colossi & 5 Stalkers & 7 Zealots	Symmetric Heterogeneous
MMM-3	2 Marines & 2 Medivacs & 2 Marauders	2 Marines & 2 Medivacs & 2 Marauders	Symmetric Heterogeneous
MMM-4	2 Marines & 2 Medivacs & 2 Marauders & 2Ghosts	2 Marines & 2 Medivacs & 2 Marauders & 2 Ghosts	Symmetric Heterogeneous
MMM-5	2 Marines & 2 Medivacs & 2 Marauders 2Ghosts& 2 Vikings	2 Marines & 2 Medivacs & 2 Marauders & 2 Ghosts 2 Vikings	Symmetric Heterogeneous

each time step, the agents receive a joint team reward which is defined by the total damage of the agents and the total damage from the enemy side. More environmental details are shown in Appendix A.

In all the scenarios, following the configurations in [12, 45], we use THGC and baseline algorithms to train the agent against the build-in AI enemy units. We use the winning rate and rewards as the evaluation metrics for the performance of the algorithm. The higher the win rate and rewards, the better the algorithm performance. Baseline algorithms include COMA [12], QMIX [45] and BiCNet [15]. More training configuration details are shown in Appendix B. The key winning strategy for MMM-3 is that the Medivacs approach the enemies first, absorb fire and then retreat to heal the appropriate ally (i.e. the one with the least health). In MMM-5, moreover, we expect that the Vikings will be able to change their modes in a reasonable manner according to the situation on the battlefield. When the enemy's air power is strong, a Viking can enter a fighter mode that enables it to attack the enemy's aerial forces, conversely, when the enemy's ground strength is stronger than ours, the Vikings can switch to assault mode.

5.2 Expansion experiment of the number of agents

To evaluate the performance and verify that our algorithm can easily expand the number of agents, we freeze the training every 100 epochs and test the model over 20 epochs to compute an average test winning rate. These experiments are carried out with 5 different random seeds, and results are shown with a 95% confidence interval. The results are reported in Fig. 5, where the averaged winning rates vs. the training epoch on the three battle scenarios (1C2S3Z, 2C3S5Z, 3C5S7Z) are given. In 1C2S3Z (see Fig. 5a), the win rate of the THGC, QMIX, COMA, and BiCNet algorithms can reach about 95% 89%, 81%, and 64% respectively, indicating that these algorithms can learn a good policy for the agents on scenario 1C2S3Z. On 3C5S7Z (see Fig. 5b), moreover, the win rate of the ThGC, QMIX, COMA, and BiCNet algorithms are 91%, 78%, 69%, and 58%, respectively; this indicates that baseline algorithms may fail to learn a good policy for the agents on scenario 3C5S7Z (see Fig. 5c). These results demonstrate that THGC outperforms QMIX and other popular methods.

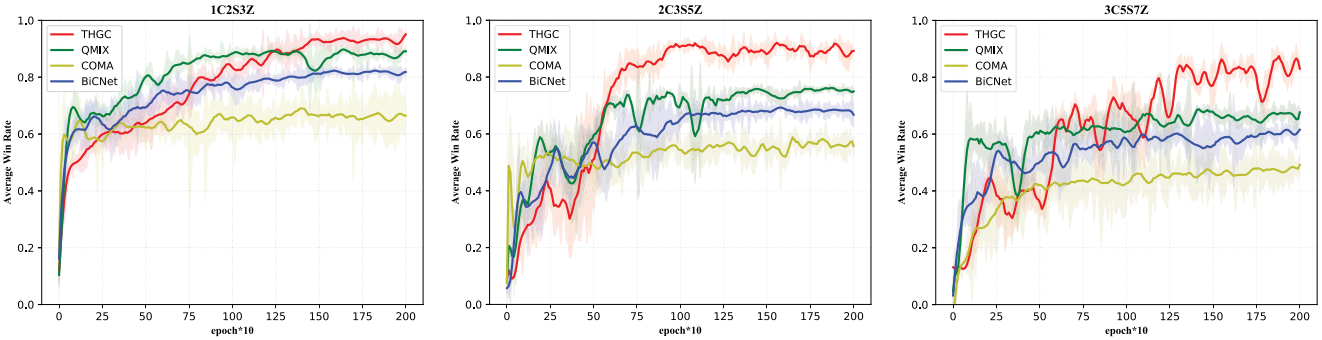
The difficulty experienced by agents in learning strategy increases with the number of agents. As shown in Fig. 6a, as the number of units increases, the baseline algorithms of QMIX, COMA, and BiCNet achieve worse performance, while their winning rate also decreases. However, the THGC algorithm can still maintain a high win rate and learn the correct strategies even if the number of agents increases. In order to further explore the trend of algorithm performance decreasing along with the number of agents,

(a) MMM-3 (b) MMM-4 (c) MMM-5

Fig. 4 Decentralised unit micromanagement in StarCraft II, where each learning agent controls an individual unit. The goal is to coordinate behaviour across agents to defeat all enemy units

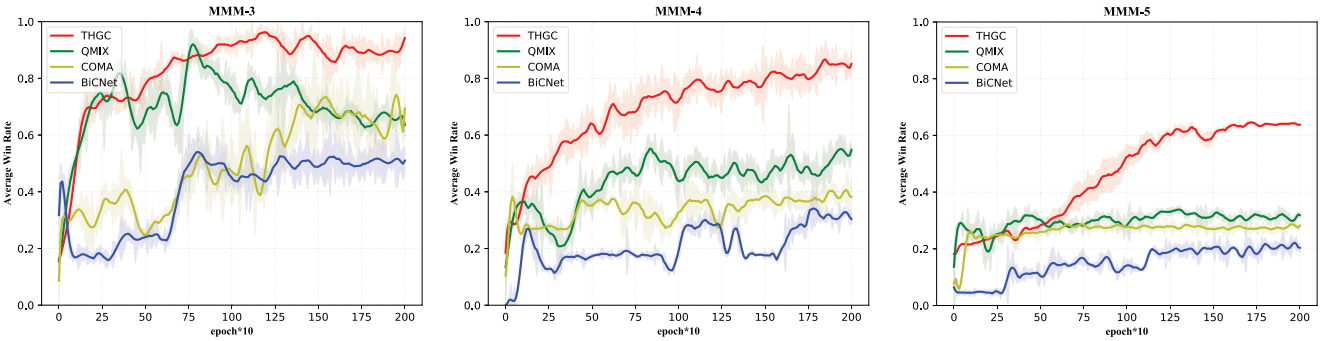
Table 2 The attributes of each type of units

Name	Function	Type
Medivac	Healing ability (can provide treatment to the injured units); Transportation ability (can withdraw units from the battlefield).	Terran
Marauder	Slowing ability (can reduce the speed of enemy units and pursue the enemy); High defense (can withstand more attacks from the enemy).	Terran
Marines	Its low cost is suitable for mass production.	Terran
Ghosts	Stealth ability (can disappear from the enemy's field of vision); Sniper ability (can attack the enemy units at long range).	Terran
Viking	Assault Mode, it can only attack ground units; Fighter Mode, it an only attack air units.	Terran



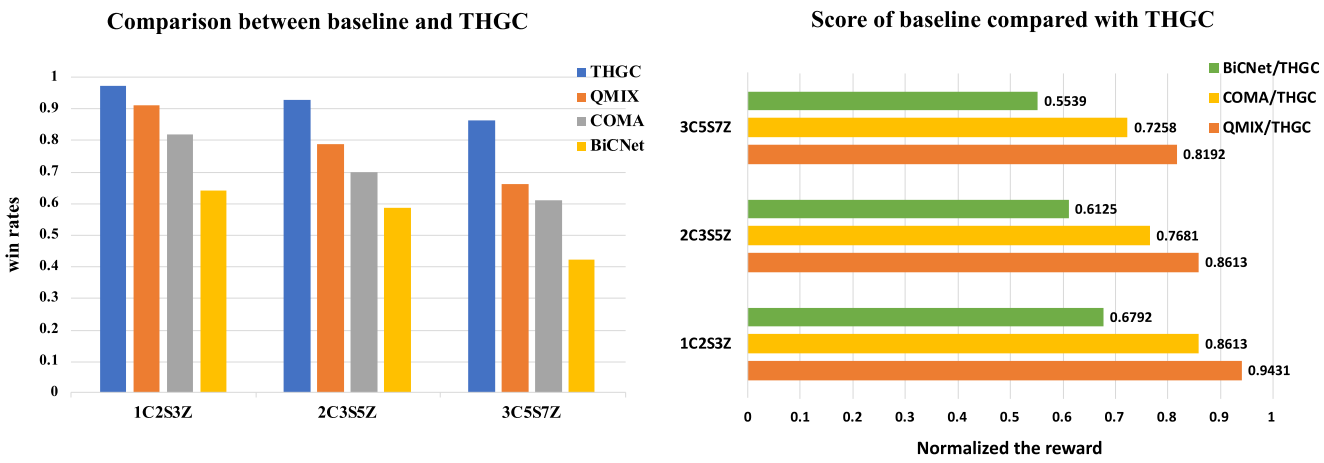
(a) 1C2S3Z (b) 2C3S5Z (c) 3C5S7Z

Fig. 5 The average win rates for COMA, BiCNet, QMIX and THGC on three combat maps with different numbers of agents



(a) MMM-3 (b) MMM-4 (c) MMM-5

Fig. 6 Comparison between baseline algorithms and the THGC algorithm on three combat maps (1C2S3Z, 2C3S5Z, 3C5S7Z). **b**: each bar cluster shows the 0-1 normalized score for rewards of learned policies with different numbers of agents. As the number of cooperating agents increases, our THGC algorithm achieves increasingly better performance compared with baseline algorithms



(a) The average win rate

(b) Normalized the reward

Fig. 7 The average win rates for COMA, BiCNet, QMIX and THGC on three combat maps (MMM-3, MMM-4, MMM-5) with different types of agents

we normalize the reward to obtain the score of QMIX, COMA, and BiCNet compared with THGC in for same scene. Experimental results (see Fig. 6b) show that the scores of QMIX, COMA, and BiCNet are lower than that of THGC with the same increase in agent quantity. This demonstrates that our THGC algorithm has the potential to be applied to a large number of agents.

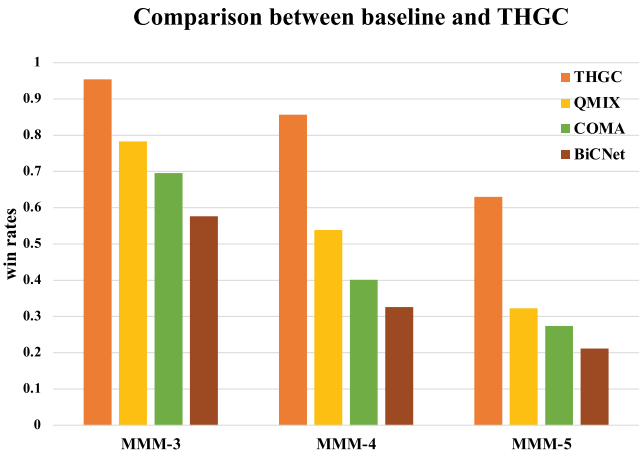
5.3 Agent type expansion experiments

To evaluate performance and verify that our algorithm has good type scalability, we conduct further experiments on the three scenarios (MMM-3, MMM-4, MMM-5) shown in Fig. 4. In these three scenarios, as the unit type complexity gradually increases, the agent strategy learning becomes more and more difficult, while the interactions between various types of agents also become increasingly different and complex. Taking Medivac and Marauder as an example, the type of interaction that takes place between the two Medivacs is different from that between the Medivac and the Marauder from a unit function perspective. The interaction of two Medivacs takes the form of cooperating on a treatment task to achieve the maximum amount of treatment on the battlefield. By contrast, the interaction between Medivacs and Marauders takes the form of Marauders protecting the Medivacs from being killed, while Medivacs protect the Marauders from damage and provide them with treatment.

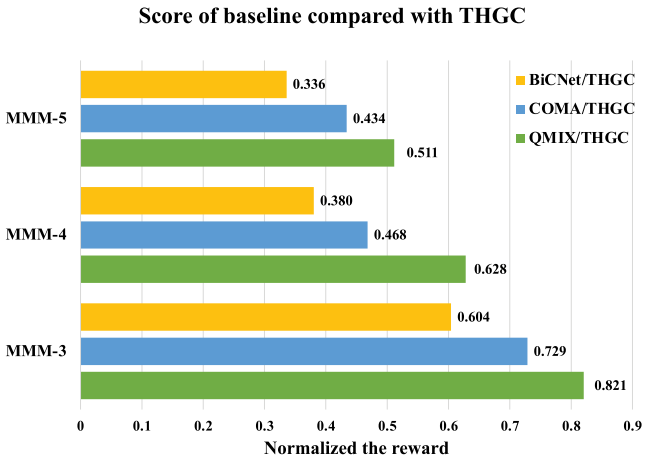
These experiments are carried out with 5 different random seeds, and results are shown with a 95% confidence interval. The results are reported in Fig. 7, where the averaged winning rates vs. the training epoch on the three battle scenarios are given. These results demonstrate that THGC outperforms QMIX and other popular methods. Even if there are many units with complex types and different capabilities on the battlefield, our THGC algorithm is still able to achieve good cooperation between all units and win the battle. As shown in Fig. 8a, as the scenario complexity increases, the performance of the THGC, QMIX, COMA, and BiCNet algorithms deteriorates. As shown in Fig. 8b, the performance of the QMIX, COMA, and BiCNet algorithms also deteriorates more rapidly than that of the THGC algorithm. In the MMM-3 scenario (see Fig. 7a), the THGC algorithm can achieve a win rate of about 94%. In the hard MMM-5 scenario (see Fig. 7c), moreover, it can achieve a win rate of about 63%.

Furthermore, the win rates of the QMIX, COMA, and BiCNet algorithms are lower than that of THGC in all three scenarios, especially in the MMM-5 scenario; in short, for all these scenarios, THGC consistently achieves the best performance. This demonstrates that considering the different interactions between various types of agents can ultimately induce better-trained policies.

It should be noted here that because our network structure has an additional set of communication parts (that is, BiLSTM) relative to the baseline algorithm QMIX [45], this requires our method to spend more time on calculation. In the same experimental environment and parameter settings, we recorded the time required for THGC and QMIX to pass 1000 epochs of training in three scenarios (MMM-3, MMM-4, and MMM-5). To facilitate comparison, we normalize the time based on THGC. The experimental results are shown in Fig. 9. As can be seen from the figure, as the complexity of the environment increases, the THGC algorithm takes more time than QMIX. However, even in the most complex scenario (MMM-5), THGC takes only around 10% more time than QMIX. Compared to the performance improvement, this increased consumption appears tolerable.



(a) The average win rate



(b) Normalized the reward

Fig. 8 Comparison between baseline algorithms and the THGC algorithm on three combat maps (MMM-3, MMM-4, MMM-5). **b** each bar cluster shows the 0-1 normalized score for rewards of learned policies with different types of agents. As the number of agent types increases, our THGC algorithm achieves increasingly better performance compared with baseline algorithms

5.4 Ablations

To investigate the effect of the three components (knowledge-sharing, group communication, and value decomposition) on policy performance, we evaluate three variants of THGC in an ablation study. These three variants are THGC-K, THGC-C, and THGC-V. THGC-K is THGC without knowledge-sharing, meaning that the low-level cognitions are directly used as input to the communication channel. THGC-C is THGC without the communication channel, meaning that collaboration between groups can rely only on value decomposition. THGC-V is THGC without value decomposition, such that we input the group joint value Q_{te} into the fully connected layer and output the Q_{tot}. We conduct experiments on the map scenarios of 2S3Z and MMM-3.

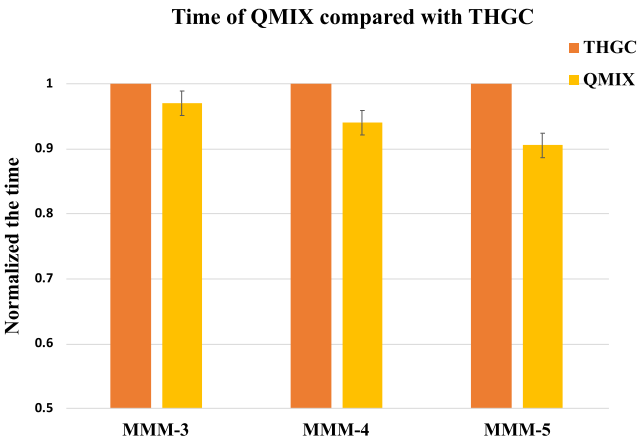


Fig. 9 Each bar cluster shows the 0-1 normalized time

As shown in Fig. 10, by comparing THGC and THGC-K, we can see that the removal of knowledge-sharing causes a drop in performance. In this experiment, it can be observed that THGC agents of the same group (3 Zealots) always gather in the same area and exhibit more consistent and synchronized behavior. When the agents of other groups (2 Stalkers) need help, the agents of the main group (3 Zealots) will move to the agent and provide support. However, THGC-K agents are more likely to be distracted by nearby enemies or the plight of friends and experience a dilemma regarding whether to attack the enemy or rescue the friend. This results in fewer defeats of the enemy, as this outcome is likely to require consistent cooperation among agents in order to maintain the maximum effective attack output. Moreover, when comparing THGC and THGC-C (THGC-V), we can see that the removal of the communication channel (value decomposition) results in a slight performance decline. In this experiment, we can see that the various groups of THGC-C (THGC-V) agents cooperate less frequently and are more inclined to act alone as separate groups. These experimental results show that group cognitive consistency can indeed strengthen cooperation between the agents in the same group, that group communication can coordinate cooperation between various groups, and that value decomposition can further enhance the collaboration between various groups.

6 Conclusions and future work

In this work, we focus on simplifying policy learning in large-scale multi-agent systems. THGC can reduce

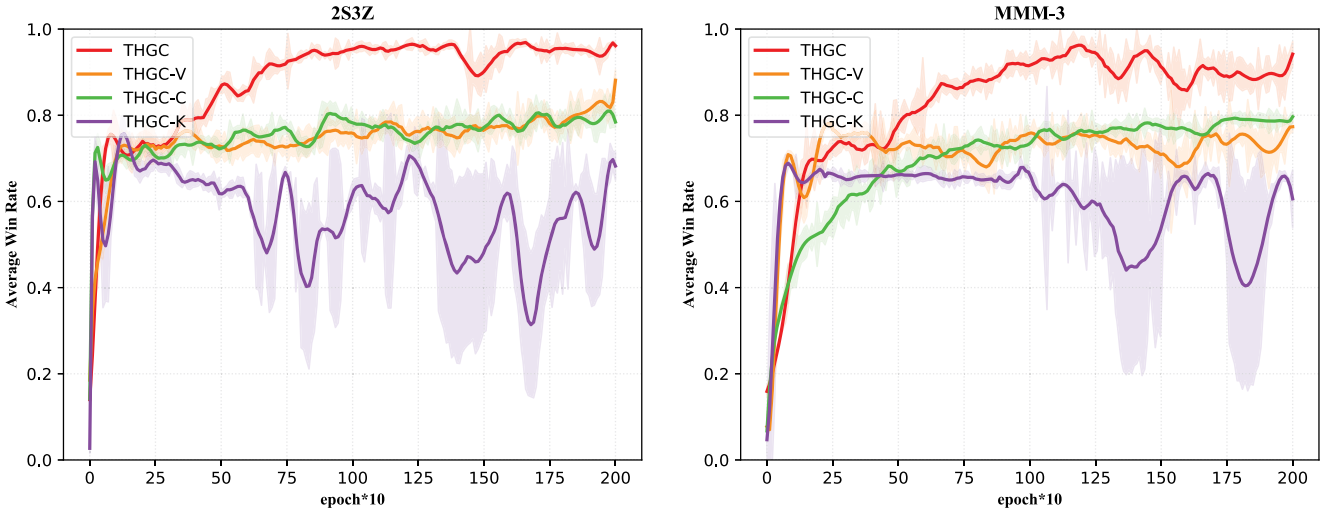


Fig. 10 Win rates for THGC and ablations on 2S3Z and MMM-3 maps

the complexity of interaction between various agents and facilitate the learning of between-agent relationships. Empirically, we demonstrated that the learned model outperformed other MARL models in a variety of multi-agent environments. Moreover, the THGC algorithm has been proven to expand the number and types of agents easily, and may therefore be valuable for large-scale real-world applications.

It is important for simplifying policy learning to learn the interaction relationships between agents. However, at different timesteps in an episode, the interaction relationships between agents are constantly changing. In future work, we plan to focus on designing a method that can dynamically learn the interaction relationships between agents.

Acknowledgments This work was supported in part by the National Key Research and Development Program of China under Grant No.2017YFB1001901, in part by the Key Program of Tianjin Science and Technology Development Plan under Grant No.18ZXZNGX00120 and in part by the China Postdoctoral Science Foundation under Grant No.2018M643900.

Appendix A: Environment details

We follow the settings of SMAC [34], which could be referred in the SMAC paper. For clarity and completeness, we state these environment details again.

A.1 States and observations

At each time step, agents receive local observations within their field of view. This encompasses information about the map within a circular area around each unit with a radius equal to the sight range, which is set to 9. The sight range makes the environment partially observable for agents. An agent can only observe others if they are both alive and located within its sight range. Hence, there is no way for agents to distinguish whether their teammates are far away or dead. If one unit (both for allies and enemies) is dead or unseen from another agent's observation, then its unit feature vector is reset to all zeros. The feature vector observed by each agent contains the following attributes for both allied and enemy units within the sight range: distance, relative x, relative y, health, shield, and unit type. If agents are homogeneous, the unit type feature will be omitted. All Protos units have shields, which serve as a source of protection to offset the damage and can regenerate if no new damage is received. Lastly, agents can observe the terrain features surrounding them, in particular, the values of eight points at a fixed radius indicating height and walkability.

The global state is composed of the joint unit features of both ally and enemy soldiers. Specifically, the state vector includes the coordinates of all agents relative to the center of the map, together with unit features present in the observations. Additionally, the state stores the energy/cooldown of the allied units based on the unit property, which represents the minimum delay between attacks/healing. All features, both in the global state and in individual observations of agents, are normalized by their maximum values

A.2 Action space

The discrete set of actions which agents are allowed to take consists of move[direction], attack[enemy id], stop and no-op. Dead agents can only take no-op action while live agents cannot. Agents can only move with a fixed movement

amount 2 in four directions: north, south, east, or west. To ensure decentralization of the task, agents are restricted to use the attack[enemy id] action only towards enemies in their shooting range. This additionally constrains the ability of the units to use the built-in attack-move micro-actions on the enemies that are far away. The shooting range is set to be 6 for all agents. Having a larger sight range than a shooting range allows agents to make use of the move commands before starting to fire. The unit behavior of automatically responding to enemy fire without being explicitly ordered is also disabled. As healer units, Medivacs use heal[agent id] actions instead of attack[enemy id].

A.3 Rewards

At each time step, the agents receive a joint reward equal to the total damage dealt on the enemy units. In addition, agents receive a bonus of 10 points after killing each opponent, and 200 points after killing all opponents for winning the battle. The rewards are scaled so that the maximum cumulative reward achievable in each scenario is around 20.

Appendix B: Training configurations

The training time is about 14 hours to 24 hours on these maps (Intel (R) Core (TM) i7-8700 CPU @ 3.20GHz, 32 GB RAM, Nvidia GTX 1050 GPU), which is ranging based on the agent numbers and map features of each map. The number of the total training steps is about 2 million and every 10 thousand steps we train and test the model. When training, a batch of 32 epochs are retrieved from the replay buffer which contains the most recent 1000 epochs. We use ϵ-greedy policy for exploration. The starting exploration rate is set to 1 and the end exploration rate is 0.05. Exploration rate decays linearly at the first 50 thousand steps. We keep the default configurations of environment parameters. Hyperparameters were based on the PyMARL [34] implementation of QMIX and are listed in Table 3. All hyperparameters are the same in StarCraft II.

Table 3 Hyperparameter settings across all runs and algorithms/baselines

Name	Description	Value
lr	learning rate	0.0005
optimizer	type of optimizer	Adam [21]
optim α	Adam param	0.99
optim ϵ	Adam param	$1e-5$
target update interval	copy live params to target params every _ epochs	200
bs	batch size (# of epochs per batch)	32
grad clip	reduce global norm of gradients beyond this value	10
$\|\mathcal{D}\|$	maximum size of replay buffer (in epochs)	5000
γ	discount factor	0.99
starting ϵ	starting value for exploraton rate annealing	1.0
ending ϵ	ending value for exploraton rate annealing	0.05
anneal time	number of steps to anneal exploration rate over	500000

References

1. Bear A, Kagan A, Rand DG (2017) Co-evolution of cooperation and cognition: the impact of imperfect deliberation and context-sensitive intuition. Proc Royal Soc B Biol Sci 284(1851):20162326
2. Bresciani PG, Giunchiglia P, Mylopoulos F, Perini J, TROPOS A (2004) An agent oriented software development methodology. Journal of autonomous agents and multiagent systems, Kluwer Academic Publishers
3. Butler E (2012) The condensed wealth of nations. Centre for Independent Studies
4. Carion N, Usunier N, Synnaeve G, Lazaric A (2019) A structured prediction approach for generalization in cooperative multi-agent reinforcement learning. In: Advances in neural information processing systems, pp 8130–8140
5. Chen Y, Zhou M, Wen Y, Yang Y, Su Y, Zhang W, Zhang D, Wang J, Liu H (2018) Factorized q-learning for large-scale multi-agent systems. arXiv:1809.03738
6. Chuang L, Chao X, Jie H, Wenzhuo L et al (2017) Hierarchical architecture design of computer system. Chinese J Comput 40(09):1996–2017
7. Clevert DA, Unterthiner T, Hochreiter S (2015) Fast and accurate deep network learning by exponential linear units (elus). arXiv:1511.07289
8. Cossentino M, Gaglio S, Sabatucci L, Seidita V (2005) The passi and agile passi mas meta-models compared with a unifying proposal. In: International central and eastern european conference on multi-agent systems, pp 183–192. Springer
9. Cossentino M, Hilaire V, Molesini A, Seidita V (2014) Handbook on agent-oriented design processes. Springer, Berlin
10. Das A, Gervet T, Romoff J, Batra D, Parikh D, Rabbat M, Pineau J (2018) Tarmac: Targeted multi-agent communication. arXiv:1810.11187
11. Dugas C, Bengio Y, Bélisle F., Nadeau C, Garcia R (2009) Incorporating functional knowledge in neural networks. J Mach Learn Res 10(Jun):1239–1262
12. Foerster JN, Farquhar G, Afouras T, Nardelli N, Whiteson S (2018) Counterfactual multi-agent policy gradients. In: Thirty-second AAAI conference on artificial intelligence
13. Gordon DM (1996) The organization of work in social insect colonies. Nature 380(6570):121–124
14. Ha D, Dai A, Le QV (2016) Hypernetworks. arXiv:1609.09106
15. Henriques R, Madeira SC (2016) Bicnet: Flexible module discovery in large-scale biological networks using biclustering. Algorithms Mol Biol 11(1):14
16. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Computat 9(8):1735–1780
17. Iqbal S, Sha F (2018) Actor-attention-critic for multi-agent reinforcement learning. arXiv:1810.02912

18. Jeanson R, Kukuk PF, Fewell JH (2005) Emergence of division of labour in halictine bees: contributions of social interactions and behavioural variance. Anim Behav 70(5):1183–1193
19. Jiang J, Dun C, Lu Z (2018) Graph convolutional reinforcement learning for multi-agent cooperation. arXiv:1810.09202,2(3)
20. Jiang J, Lu Z (2018) Learning attentional communication for multi-agent cooperation. In: Advances in neural information processing systems, pp 7254–7264
21. Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. arXiv:1412.6980
22. Liu Y, Hu Y, Gao Y, Chen Y, Fan C (2019) Value function transfer for deep multi-agent reinforcement learning based on n-step returns. In: Proceedings of the twenty-eighth international joint conference on artificial intelligence, pp 457–463
23. Liu Y, Wang W, Hu Y, Hao J, Chen X, Gao Y (2019) Multi-agent game abstraction via graph attention neural network. arXiv:1911.10715
24. Long Q, Zhou Z, Gupta A, Fang F, Wu Y, Wang X (2020) Evolutionary population curriculum for scaling multi-agent reinforcement learning. arXiv:2003.10423
25. Lowe R, Wu YI, Tamar A, Harb J, Abbeel OP, Mordatch I (2017) Multi-agent actor-critic for mixed cooperative-competitive environments. In: Advances in neural information processing systems, pp 6379–6390
26. Mao H, Liu W, Hao J, Luo J, Li D, Zhang Z, Wang J, Xiao Z (2019) Neighborhood cognition consistent multi-agent reinforcement learning. arXiv:1912.01160
27. Melo FS, Veloso M (2011) Decentralized mdps with sparse interactions. Artif Intell 175(11):1757–1789
28. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533
29. Nair V, Hinton GE (2010) Rectified linear units improve restricted boltzmann machines. In: ICML
30. Oliehoek FA, Amato C et al (2016) A concise introduction to decentralized POMDPs, vol 1. Springer, Berlin
31. OroojlooyJadid A, Hajinezhad D (2019) A review of cooperative multi-agent deep reinforcement learning. arXiv:1908.03963
32. Pal SK, Mitra S (1992) Multilayer perceptron, fuzzy sets classifiaction
33. Ryu H, Shin H, Park J (2020) Multi-agent actor-critic with hierarchical graph attention network. In: AAAI, pp 7236–7243
34. Samvelyan M, Rashid T, de Witt CS, Farquhar G, Nardelli N, Rudner TG, Hung CM, Torr PH, Foerster J, Whiteson S (2019) The starcraft multi-agent challenge. arXiv:1902.04043
35. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv:1707.06347
36. Singh A, Jain T, Sukhbaatar S (2018) Learning when to communicate at scale in multiagent cooperative and competitive tasks. arXiv:1812.09755
37. Son K, Kim D, Kang WJ, Hostallero DE, Yi Y (2019) Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. arXiv:1905.05408
38. Stone P, Veloso M (2000) Multiagent systems: a survey from a machine learning perspective. Auton Robot 8(3):345–383
39. Sukhbaatar S, Fergus R et al (2016) Learning multiagent communication with backpropagation. In: Advances in neural information processing systems, pp 2244–2252
40. Sunehag P, Lever G, Gruslys A, Czarnecki WM, Zambaldi V, Jaderberg M, Lanctot M, Sonnerat N, Leibo JZ, Tuyls K et al (2017) Value-decomposition networks for cooperative multi-agent learning. arXiv:1706.05296
41. Sutton RS, McAllester DA, Singh SP, Mansour Y (2000) Policy gradient methods for reinforcement learning with function approximation. In: Advances in neural information processing systems, pp 1057–1063
42. Veličković P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y (2017) Graph attention networks. arXiv:1710.10903
43. Wang W, Yang T, Liu Y, Hao J, Hao X, Hu Y, Chen Y, Fan C, Gao Y (2020) From few to more: large-scale dynamic multiagent curriculum learning. In: AAAI, pp 7293–7300
44. Werbos PJ (1990) Backpropagation through time: what it does and how to do it. Proc IEEE 78(10):1550–1560
45. Whiteson S (2018) Qmix: Monotonic value function factorisation for deep multi- agent reinforcement learning
46. Wooldridge M, Jennings NR, Kinny D (2000) The gaia methodology for agent-oriented analysis and design. Auton Agents Multi-Agent Syst 3(3):285–312
47. Yang Y, Luo R, Li M, Zhou M, Zhang W, Wang J (2018) Mean field multi-agent reinforcement learning. arXiv:1802.05438
48. Yu C, Zhang M, Ren F, Tan G (2015) Multiagent learning of coordination in loosely coupled multiagent systems. IEEE Trans Cybern 45(12):2853–2867
49. Zhang Z, Yang J, Zha H (2019) Integrating independent and centralized multi-agent reinforcement learning for traffic signal network optimization. arXiv:1909.10651

Hao Jiang received the B.S. degree in computer science from Sichuan University, Chengdu, Sichuan, China, in 2018. He is currently pursuing the masters degree with the National University of Defense Technology. His current research interests include multi-agent reinforcement learning algorithms and graph attention network.

Dianxi Shi is a Researcher and the Deputy Director of the Artificial Intelligence Research Center of National Innovation Institute of Defense Technology. He received the B.S., M.S. and Ph.D.degrees in computer science at National University of Defense Technology, Changsha, China, in 1989, 1996 and 2000. His research interests include distributed object middleware technology, software component technology, adaptive software technology and intelligent unmanned cluster system software architecture.

Chao Xue received the B.S. and Ph.D. degrees in department of computer science and technology from Tsinghua University in 2011 and 2017. He is currently a postdoctoral researcher at postdoctoral research center, Academy of Military Science. His research interest includes modeling and evaluation of networked computing systems and applications, with a particular interest in cloud computing, reinforcement learning, parallel learning, and cloud native machine learning system.

Yajie Wang received the B.S. degree in computer science from Beihang University, Beijing, China, in 2018. She is currently pursuing the masters degree with the National University of Defense Technology. Her current research interests include multi-agent reinforcement learning algorithms and attention mechanism.

Gongju Wang received the B.E. degree in computer science from Beijing Jiaotong University, Beijing, China, in 2017. He is pursuing the master's degree with the Artificial Intelligence Research Center, National Innovation Institute of Defense Technology, Beijing 100166, China. His current interests include multi-agent reinforcement learning and hierarchical reinforcement learning.

Yongjun Zhang received the Ph.D. degree in computer science at National University of Defense Technology in 2000. He has participated in the National High Technology Research and Development Program of China and the National Natural Science Foundation of China, and has published more than 20 papers.