Dynamic Deep Factor Graph for Multi-Agent Reinforcement Learning

Yuchen Shi, Shihong Duan, Cheng Xu, *Member, IEEE*, Ran Wang, *Graduate Student Member, IEEE*, Fangwen Ye, and Chau Yuen, *Fellow, IEEE*

***Abstract*—Multi-agent reinforcement learning (MARL) requires effective coordination among multiple decision-making agents to achieve joint goals. Approaches based on a global value function face the curse of dimensionality, while fully decomposed *centralized training with decentralized execution* (CTDE) methods often suffer from *relative overgeneralization*. *Coordination graphs* mitigate this issue but typically fail to capture *dynamic* collaboration patterns that evolve over time and across tasks. We propose *Dynamic Deep Factor Graphs* (DDFG), a value decomposition algorithm that represents the global value via factor graphs and *learns* graph structures on the fly through a graph-generation policy, adapting to evolving inter-agent relations. We provide a theoretical upper bound on the approximation error of high-order decompositions and reveal how the maximum order D trades off accuracy against computation, offering guidance for balancing performance and cost. Using max-sum for inference, DDFG efficiently derives joint policies. Experiments on higher-order predator–prey and SMAC show consistent gains over strong value-decomposition baselines, demonstrating improved sample efficiency and robustness in complex settings.**

***Index Terms*—Dynamic graph, factor graph, multi-agent reinforcement learning (MARL), relative overgeneralization, dynamic collaboration.**

I. Introduction

COLLABORATIVE multi-agent systems have become increasingly relevant across a broad spectrum of real-world scenarios, encompassing areas such as autonomous vehicular navigation [1], collective robotic decision-making [2], and distributed sensor networks. In this context, Reinforcement Learning (RL) has demonstrated remarkable efficacy in tackling a diverse range of collaborative multi-agent challenges. Notably, intricate tasks such as the coordination of robotic swarms and the automation of vehicular control are often conceptualized within the framework of Collaborative Multi-Agent Reinforcement Learning (MARL) [3].

Addressing these complex tasks frequently necessitates the disaggregation of either the agents' policy mechanisms or their value functions. Within the domain of policy-based methodologies, Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [4] has been recognized for its ability to learn distributed policies tailored to continuous action spaces. Multi-Agent Proximal Policy Optimization (MAPPO) [5] achieves better performance than MADDPG by extending Proximal Policy Optimization (PPO) [6] to multi-agent scenarios. Meanwhile, Decomposed Off-Policy Policy Gradient (DOP) [7] introduces the idea of value function decomposition into the Actor-Critic framework, addressing the credit assignment problem in both discrete and continuous action spaces. On the spectrum of value function-based approaches, Value Decomposition Networks (VDN) [8] pioneer the decomposition of the collective action-value function into an aggregation of individual action-value functions. Furthering this paradigm, QMIX [9] innovates by conceptualizing the collective action-value function through a monotonic function, thereby enhancing the representational capacity of the system.

However, in the realm of collaborative multi-agent systems, methodologies often grapple with a game-theoretic complication known as *relative overgeneralization*, a phenomenon where the punitive consequences for non-collaboration amongst agents overshadow the rewards for cooperative engagement, culminating in less-than-optimal performance outcomes [10]. To mitigate this issue, QTRAN [11] ameliorates the constraints on value function decomposition inherent in QMIX by introducing the Individual-Global-Max (IGM) condition and devising two soft regularization methods aimed at fine-tuning the action selection process to balance between joint and individual value functions. Subsequently, QPLEX [12] advances this concept by proposing an enhanced IGM that parallels the original in importance and incorporates the Dueling architecture to further refine the decomposition of the joint value function. Meanwhile, Weighted QMIX (WQMIX) [13] transitions from "monotonic" to "non-monotonic" value functions through the introduction of a weighted operator, addressing the limitations of monotonicity constraints. Despite these advancements, the stated algorithms occasionally falter in identifying the

Received 19 August 2025; revised 6 November 2025; accepted 16 November 2025. Date of publication 19 November 2025; date of current version 4 February 2026. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62101029, in part by Guangdong Basic and Applied Basic Research Foundation under Grant 2023A1515140071, and in part by the China Scholarship Council Award under Grant 202006465043 and Grant 202306460078. Recommended for acceptance by Y. Yu. *(Corresponding author: Cheng Xu.)*

Yuchen Shi, Shihong Duan, Cheng Xu, Ran Wang, and Fangwen Ye are with the School of Computer and Communication Engineering, Shunde Innovation School, University of Science and Technology Beijing, Beijing 100083, China (e-mail: shiyuchen199@sina.com; duansh@ustb.edu.cn; xucheng@ustb.edu.cn; wangran423@foxmail.com; yfwen2000@outlook.com).

Chau Yuen is with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798 (e-mail: chau.yuen@ntu.edu.sg).

Code is available at https://github.com/SICC-Group/DDFG.

This article has supplementary downloadable material available at https://doi.org/10.1109/TPAMI.2025.3634378, provided by the authors.

Digital Object Identifier 10.1109/TPAMI.2025.3634378

globally optimal solution due to the inherent constraints of value decomposition.

The Coordination Graph (CG) framework [14] presents a viable alternative that preserves the benefits of value function decomposition while simultaneously addressing the issue of relative overgeneralization. Within this framework, agents and the synergistic gains from their interactions are represented as vertices and edges, respectively, in a graph structured according to the joint action-observation space. Deep Coordination Graphs (DCG) [15] approximate these gain functions using deep neural networks and disseminate the derived value functions across the graph [16] via a message-passing algorithm. The fixed structure of coordination graphs in DCG limits their applicability to diverse real-world scenarios. NL-CG [17] attempts to extend CG to nonlinear scenarios, while SOPCG [18] and CASEC [19] enhance the representational capacity by constructing sparse coordination graph structures. The Deep Implicit Coordination Graph (DICG) [20] differs from the aforementioned CG-based methods in that it represents message passing through Graph Convolutional Networks (GCN) [21] instead of using the maximum-sum algorithm on the CG. DICG integrates an attention mechanism within the GCN, thereby introducing an implicit framework for the coordination graph. Whether using the coordination graph framework explicitly or implicitly, these methods are still constrained by the inherent limitations of the coordination graph structure, especially when dealing with complex coordination relationships.

To surmount these limitations, this paper introduces a novel value decomposition algorithm termed the *Dynamic Deep Factor Graph* (DDFG), which leverages factor graphs [22] for the decomposition of the global value function into a sum of local value functions of arbitrary order, thereby offering a more robust characterization capability than coordination graphs. Distinctly, DDFG eschews predefined graph structures in favor of dynamically generated graphs that accurately represent the collaborative relationships among agents, based on real-time observations processed through neural networks. This paper defines the graph generation policy as a quasi-multinomial distribution and provides relevant theoretical proofs. Following the idea of probability modeling from the PPO [6], we constructed a networked graph policy over a quasi-multinomial distribution to generate dynamic factor graph structures in various scenarios. The adaptability of the graph structure is further enhanced by the application of the message-passing algorithm [23], optimizing the learning of agents' policies and the representational efficiency of value function decomposition.

The core contributions of our study are delineated as follows:

1) *Canonical Polyadic (CP) Decomposition-based Factor Graph Value Function Network.* We leverage factor graphs to decompose the global value functions into a sum of higher-order local value functions, augmented by tensor-based CP decomposition [24], [25]. This approach, an extension of the matrix low-rank approximation, significantly reduces parameter count and increases update frequency by decomposing higher-order value functions into the outer products of rank-one tensors. Concurrently, we theoretically prove the error upper bound of the global value function under the aforementioned decomposition. We uncover the relationship between the maximum order D and the error upper bound, which highlights the trade-off between model accuracy and computational cost.
2) *Graph Structure Generation Policy.* We propose a new graph structure generation policy, defined as a quasi-multinomial distribution. It is analogous to agent policies, for generating real-time, variable factor graph structures through probabilistic modeling, akin to the Proximal Policy Optimization (PPO) algorithm. This policy facilitates the depiction of agent collaboration, employing a neural network that processes agents' observations to generate dynamic graph structures. We have presented the Policy improvement lower bound for the graph policy and derived the loss for the graph policy, thereby providing a theoretical foundation for its optimization and convergence.
3) *Dynamic Deep Factor Graph-Based MARL Algorithm.* The DDFG algorithm, leveraging factor graphs over coordination graphs, addresses the pitfall of relative overgeneralization by providing a more versatile decomposition approach and generating dynamic factor graph structures to enhance expressive power. Through the application of factor graph message passing algorithms to various graph structures, DDFG facilitates the learning of agents' policies. Comparative analysis with contemporary MARL methodologies within the experimental section showcases DDFG's efficacy in complex scenarios such as high-order predator-prey and SMAC tasks.

The rest of this paper is organized as follows. Section II introduces the definition of factor graphs and Markov decision processes based on factor graphs. Section III presents the proposed Dynamic Deep Factor Graph algorithm. Section IV presents the composition of algorithm related loss functions. The simulation results and corresponding analysis are given in Section V. In Section VI, we conclude this paper.

II. Background

A. FG-POMDP

A factor graph G is defined by a set of variable nodes V, a set of factor nodes F, and a set of undirected edges $\mathcal{E}$. We correspond intelligent agents with variable nodes; If the local value function corresponds to a factor node, then a factor graph can represent a decomposition form of the global value function. Define the adjacency matrix of the factor graph at each time t as A_t and add it to POMDP, proposing a factor graph based POMDP, namely FG-POMDP. FG-POMDP consists of a tuple $< n, S, A, \mathcal{U}, R, P, \{O^i\}_{i=1}^n, \gamma >$ [26]. Here, n represents the number of agents, S describes the true state of the environment, and $\mathcal{U}^i \in \mathcal{U}$ denotes the set of discrete actions available to agent i. At discrete time t, $A_t \in A$ is the dynamic factor graph structure in real-time. We denote it by the adjacency matrix $A_t \in \{0,1\}^{n\times m}$, where m denotes the number of factor nodes. The next state $s_{t+1} \in S$ is obtained from the transfer probability $s_{t+1} \sim P(\cdot \mid s_t, \boldsymbol{u}_t)$ with $s_{t+1} \in S$ and $\boldsymbol{u}_t \in \mathcal{U} := \mathcal{U}^1 \times \cdots \times \mathcal{U}^n$ as conditions. Each agent shares the same reward $r_t := r(s_t, \boldsymbol{u}_t)$ at moment t and

$\gamma \in [0, 1)$ denotes the discount factor. Due to partial observability, at moment t, the individual observations $o_t^i \in O^i$ of each agent, the history of observations o_t^i and actions u_t^i of agent i are represented as $\tau_t^i := (o_0^i, u_0^i, o_1^i, \ldots, o_{t-1}^i, u_{t-1}^i, o_t^i) \in (O^i \times \mathcal{U}^i)^t \times O^i$. Without loss of generality, the trajectory of the task in this paper is denoted as $\mathcal{T} = (s_0, \{o_0^i\}_{i=1}^n, A_0, \boldsymbol{u}_0, r_0, \ldots, s_T, \{o_T^i\}_{i=1}^n)$, where T is the task length.

FG-POMDP *modifies* the decision process by introducing a **time-varying structural variable** A_t (the factor-graph adjacency matrix) into the model's tuple. Concretely, A_t conditions both (i) the **value decomposition** of the joint action-value into local factors Q_j (and their inputs) and (ii) the **max-sum** message passing used for joint action selection. Thus, the learned structure directly shapes the agent-wise messages and, in turn, the selected joint action $\boldsymbol{u}_t$; the policy is a function of (τ_t, A_t), not merely τ_t.

Collaborative MARL aims to find the optimal policy $\pi : S \times \mathcal{U} \to [0, 1]$ that selects joint actions $\boldsymbol{u}_t \in \mathcal{U}$ maximizing the expected discounted sum of future rewards, achieved by estimating the optimal Q-value function. The optimal policy π greedily selects actions $\boldsymbol{u} \in \mathcal{U}$ that maximize the corresponding optimal Q-value function. However, when facing large joint action spaces, deep neural networks with θ parameters, such as DQN [27] and Double-DQN [28], may struggle to approximate the optimal Q-value function Q_θ. To address this issue, various value decomposition algorithms have been proposed to perform Q-learning in MARL efficiently. We describe these value decomposition algorithms(VDN [8], QMIX [9], WQMIX [13], QTRAN [11], QPLEX [12], DCG [15], SOPCG [18], CASEC [19], MAPPO [5]) in Appendix Section A, available online.

The learned policy cannot depend on the state s_t in a partially observable environment. Therefore, the Q-value function Q_θ is conditioned on the agent's observation-action history $\boldsymbol{\tau}_t := \{\tau_t^i\}_{i=1}^n$, which can be approximated as $Q_\theta := Q_\theta(\boldsymbol{u} \mid \boldsymbol{\tau}_t)$ [29]. To achieve this, the agent's observation $\boldsymbol{o}_t := (o_t^1, \ldots, o_t^n)$ and previous action $\boldsymbol{u}_{t-1}$ are fed into an RNN network, such as a GRU, to obtain the hidden state $\boldsymbol{h}_t := h_\psi(\cdot \mid \boldsymbol{h}_{t-1}, \boldsymbol{o}_t, \boldsymbol{u}_{t-1})$, where $\boldsymbol{h}_0 = \boldsymbol{0}$. The Q-value function Q_θ is then conditioned on the hidden state $\boldsymbol{h}_t$, i.e., $Q_\theta := Q_\theta(\boldsymbol{u} \mid \boldsymbol{h}_t)$.

B. Factor Graph

A factor graph $G = \langle V, F, \mathcal{E} \rangle$ [22] is defined by the set of variable nodes V, the set of factor nodes F and the set of undirected edges $\mathcal{E}$. Factor graphs are bipartite graphs represented as factorizations of global functions. Each variable node $v_i \in V$ in the factor graph corresponds to a variable. Similarly, each factor node $f_j \in F$ corresponds to a local function after the global function decomposition and is connected by an edge $\mathcal{E}$ to the variable node v_i and the factor node f_j when and only when v_i is an argument of f_j. A factor graph with n variable nodes and m function nodes has a binary adjacency matrix defined as $A \in \{0, 1\}^{n \times m}$.

In multi-agent reinforcement learning (MARL), agents are regarded as variable nodes, and the value functions of the agents serve as factor nodes. The factor graph G decomposes the global value function $Q(\boldsymbol{\tau}, \boldsymbol{u})$ into a sum of several local value functions [30], and the decomposition relations are given by the adjacency matrix A. Each variable node $v_i \in V$ represents an agent i. Each factor node $f_j \in F$ is equivalent to a local value function Q_j, where $n(Q_j)$ denotes the set of all variable nodes connected to Q_j, i.e., $i, j \in \mathcal{E}$. In the remainder of the paper, all factor nodes previously denoted by f_j will be replaced with Q_j for clarity. Q_j denotes the joint value function of all agents $i \in n(Q_j)$. The factor graph G (as shown in Fig. 1) represents the joint value function.

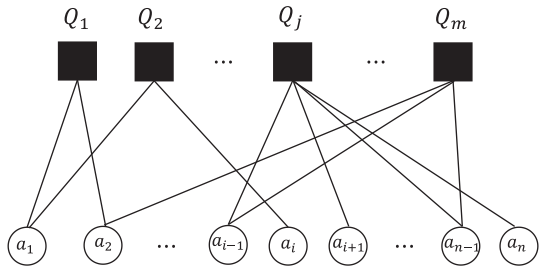

Fig. 1. Visualization of the factor graph $Q(\boldsymbol{\tau}, \boldsymbol{u}) = \sum_{j \in \mathcal{J}} Q_j(\boldsymbol{u}^j \mid \boldsymbol{\tau})$.

$$Q(\boldsymbol{\tau}, \boldsymbol{u}) = \sum_{j \in \mathcal{J}} Q_j \left(\{v_i\}_{\{i,j\} \in \mathcal{E}} \right) = \sum_{j \in \mathcal{J}} Q_j \left(\boldsymbol{u}^j \mid \boldsymbol{\tau} \right) \tag{1}$$

where $\mathcal{J}$ is the set of factor nodes and $\boldsymbol{u}^j \in \prod_{i \in n(Q^j)} \mathcal{U}^i$ denotes the joint action of all agents.

III. Method

As depicted in Fig. 2, the network structure of DDFG comprises three main components: *the graph policy*, responsible for generating the graph structure, *the Q-value function network*, and *the max-sum algorithm*. First, we obtain the agent's hidden state h_t^i through a shared RNN network. Both the graph policy and the Q-value function network share h_t^i, but the RNN network only updates jointly with the Q-value function network. The graph policy takes h_t^i as input and outputs a real-time factor graph structure A_t based on $\boldsymbol{o}_t = CONCAT[o_t^1, o_t^2, \ldots, o_t^N]$. Meanwhile, the Q-value function network receives h_t^i and A_t as inputs and generates the local value function Q_j. Finally, the max-sum algorithm is utilized to compute the local value function Q_j and obtain the action and global value functions of the agent.

A. Q-Value Function Network

Given a known policy for generating graph structures ρ (explained in Section III-C), at time t, we use $A_t \sim \rho$ to represent the higher-order decomposition of the global value function $Q(\boldsymbol{\tau}_t, \boldsymbol{u}_t)$. Simultaneously, using the adjacency matrix A_t, we represent the local value function $Q_j(\boldsymbol{\tau}_t, \boldsymbol{u}_t^j)$ as a network parameterized by θ_j. The joint value function, represented by

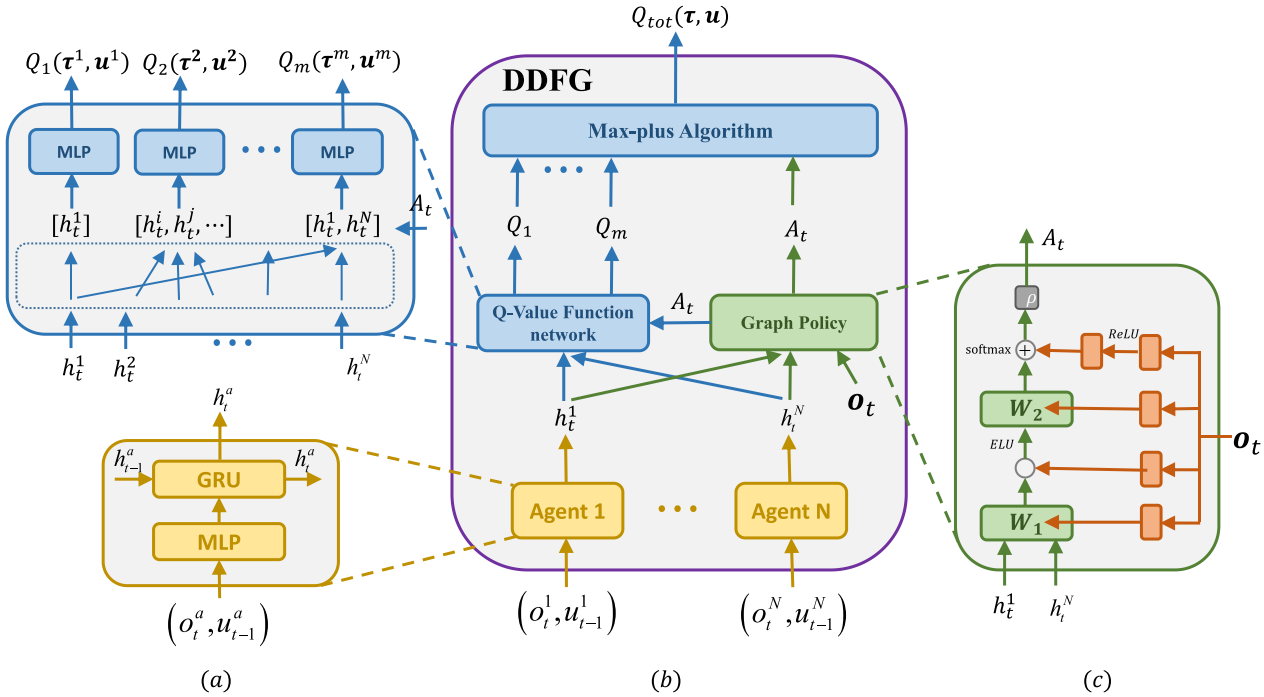

Fig. 2. The algorithmic framework of DDFG. (a) The network structure of Q-value function (Section III-A). (b) The overall architecture of DDFG (Section III). (c) The network structure of graph structure generation policy (Section III-C).

the factor graph G, is given by

$$Q(\boldsymbol{\tau}_t, \boldsymbol{u}_t, A_t) = \sum_{j \in \mathcal{J}} Q_j \left(\boldsymbol{u}_t^j \mid \boldsymbol{\tau}_t; \theta_j \right) \tag{2}$$

where $\mathcal{J}$ is the set of factor nodes, and $\boldsymbol{u}_t^j$ denotes the joint action of all agents at time t.

DCG uses a deep network to learn the coordination graph's utility and payoff functions. Similarly, DDFG extends DCG to factor graphs by learning the functions corresponding to the factor nodes in the factor graph through a deep network. We define the "order" of the value function Q_j as $D(j)$, which represents the number of agent j's connections. The order of these value functions can exceed the maximum 2in the coordination graph, making the value function decomposition network more representable. We present the design principles for the Q-value function network of the DDFG as follows:

a) The local value function Q_j accepts only the local information of the agent $i \in n(Q_j)$.
b) Adopt a common recurrent neural network sharing parameters across all agents (yellow part in Fig. 2).
c) Employing a common fully connected neural network sharing parameters on a local value function Q_j of the same order $D(j)$ (blue part in Fig. 2).
d) Use tensor-based Canonical Polyadic (CP) decomposition for the local value function Q_j of $D(j) \geq 2$.
e) Allowing generalization to different factor graph structures.

According to principle a), the local value function $Q_j(\boldsymbol{u}_t^j \mid \boldsymbol{\tau}_t; \theta_j) = Q_j(\boldsymbol{u}_t^j \mid \boldsymbol{\tau}_t^j; \theta_j)$, where $\boldsymbol{\tau}_t^j$ denotes the joint observation-action history of all agents $i \in n(Q_j)$. Based on principle b), all agents share parameters using a recurrent neural network (RNN) with generic architecture, denoted as $h_t^i = h_\psi(\cdot \mid h_{t-1}^i, o_t^i, u_{t-1}^i)$. This RNN is initialized with $h_0^i = h_\psi(\cdot \mid \mathbf{0}, o_0^i, \mathbf{0})$. According to principle c), the local value function $Q_j(\boldsymbol{u}_t^j \mid \boldsymbol{\tau}_t^j; \theta_j)$ can be approximated as $Q_j(\boldsymbol{u}_t^j \mid \boldsymbol{h}_t^j; \theta_{D(j)})$, where all local value functions of order $D(j)$ share the parameter $\theta_{D(j)}$, improving operational efficiency. The local value functions are spliced together using the matrix $\boldsymbol{h}_t^j = CONCAT_{i \in n(Q_j)} h_t^i$.

The size $\prod_{i=1}^{D(j)} |\mathcal{U}^i|$ of the joint action space involved in the local value function Q_j grows exponentially with the order $D(j)$. Since only executed action pairs are updated during Q-learning, the parameters of many outputs remain constant for a long time while the underlying RNN network is continuously updated. This can slow down the training speed and affect the message delivery. To reduce the number of parameters and increase the frequency of their updates, we extend the low-rank approximation of matrices in DCG [15] and propose to use the Canonical Polyadic (CP) decomposition of the tensor [24], [25] to approximate the value function Q_j. The CP decomposition of Q_j of rank K is defined by:

$$Q_j \left(\cdot | \boldsymbol{h}_t^j; \theta_{D(j)} \right) := \sum_{k=1}^{K} \left(f_k \left(\cdot | \boldsymbol{h}_t^j; \theta_{D(j)}^1 \right) \right.$$
$$\left. \otimes f_k \left(\cdot | \boldsymbol{h}_t^j; \theta_{D(j)}^2 \right) \otimes \cdots \otimes f_k \left(\cdot | \boldsymbol{h}_t^j; \theta_{D(j)}^{D(j)} \right) \right) \tag{3}$$

where $\otimes$ is the outer product.

To approximate the value function in our reinforcement learning model, we use a network of local value functions with

parameters $\{\theta^d_{D(j)}\}^{D(j)}_{d=1}$ and output $D(j)K|\mathcal{U}^i|, \forall i \in n(Q_j)$ (as described in (3). The tensor rank is determined by balancing the approximation's accuracy against the parameter learning speed.

However, as the adjacency matrix A_t changes in real time, the value of the global value function becomes unstable. To address this, we fix all local value functions by setting $D(j) = 1$, which yields the global value function $Q(\boldsymbol{\tau}_t, \boldsymbol{u}_t)$ plus the VDN decomposition expressed as Q_{vdn}. This results in a new adjacency matrix $A_t{}' = CONCAT[A_t, I_n]$. I_n is defined as an $n \times n$ identity matrix. The final global value function is then given by:

$$Q_{tot}(\boldsymbol{\tau}_t, \boldsymbol{u}_t, A_t; \theta, \psi) = Q(\boldsymbol{\tau}_t, \boldsymbol{u}_t) + Q_{vdn}$$
$$= \sum_{j \in \mathcal{J}} \frac{1}{|\mathcal{J}_{D(j)}|} Q_j(\boldsymbol{u}^j_t \mid \boldsymbol{h}^j_t; \theta_{D(j)}) + \frac{1}{n} \sum_{i=1}^{n} Q_i\left(u^i_t \mid h^i_t; \theta_{D(1)}\right) \tag{4}$$

B. Theoretical Analysis of Value Function Decomposition and Approximation

In this section, we approach the decomposition of the global value function from a different perspective. Theoretically, we prove the error upper bound of the global value function decomposition and reveal the potential relationship between the error upper bound and the maximum decomposition order D of the value function.

In Section III-A, we define the decomposition of the global value function Q_{tot} using factor graphs, and the final decomposed form is given by (4). When we examine the decomposition form of (4) from a different perspective and arrange the local value function Q_j in ascending order of their orders, (4) transforms into:

$$Q_{tot}(\boldsymbol{\tau}_t, \boldsymbol{u}_t) = f_0 + \frac{1}{n} \sum_{i=1}^{n} Q_i(u^i_t | h^i_t) + \frac{1}{|J_2|} \sum_{j \in \mathcal{J}_2} Q_j(\boldsymbol{u}^j_t | \boldsymbol{h}^j_t) + \cdots + Q_n(\boldsymbol{u}_t | \boldsymbol{h}_t) \tag{5}$$

where f_0 is a constant, n is the number of agents, $\mathcal{J}_d = \{j | j \in \mathcal{J} \; and \; D(j) = d\}$ denotes the set of local value functions Q_j with order d.

In Section III-A, we use the CP decomposition of tensors to approximate the local value function Q_j. Substituting (3) into (5), we obtain:

$$Q_{tot}(\boldsymbol{\tau}_t, \boldsymbol{u}_t) = f_0 + \frac{1}{n} \sum_{i=1}^{n} Q_i(u^i_t | h^i_t) + \frac{1}{|J_2|} \sum_{j \in J_2} \sum_{k=1}^{K_2} \prod_{i=1}^{2} \bar{Q}_i(u^i_t | h^i_t) + \cdots + \sum_{k=1}^{K_n} \prod_{i=1}^{n} \bar{Q}_i(u^i_t | h^i_t) \tag{6}$$

where K_d denotes the rank of the d-order local value function. $\bar{Q}_i$ corresponds to the value of each $f_k(\cdot|\theta^i_{D(j)})$ in (3) when the action u^i_t is taken, which can be regarded as the value of the marginal density function of the local value function corresponding to the agent i.

Subsequently, we can represent the global value function Q_{tot} as a Scalable Polynomial Additive Models (SPAM) [31]. SPAM is an extension of Generalized Additive Models (GAM) [32], which leverages the tensor rank decomposition of polynomials to learn powerful and inherently interpretable models.

$$Q_{tot}(\boldsymbol{\tau}_t, \boldsymbol{u}_t) = f_0 + \frac{1}{n} \sum_{i=1}^{n} \left\langle w_1(u^i_t), h^i_t \right\rangle + \frac{1}{|J_2|} \sum_{j \in J_2} \sum_{k=1}^{K_2} \lambda_{2k} \prod_{i=1}^{2} \left\langle w_{2k}(u^i_t), h^i_t \right\rangle + \cdots + \sum_{k=1}^{K_n} \lambda_{nk} \prod_{i=1}^{n} \left\langle w_{nk}(u^i_t), h^i_t \right\rangle \tag{7}$$

where λ_{dk} denotes the eigenvalues corresponding to different ranks after tensor decomposition of the local value function of order d. w_{dk} denotes the trainable parameters of each network $f_k(\cdot|\theta^i_d)$.

We present an assumption regarding the decay of eigenvalues and a lemma. Through Proposition 2 in SPAM [31] and the above content, we derive the error upper bound for the decomposition of the global value function.

Assumption 1. (γ-Exponential Decay): For the eigenvalue λ_{dk} corresponding to the local value function Q_j in (7), it is assumed to follow γ-exponential decay, that is, there exists an absolute constant $C_1 < 1$ and $C_2 = O(1)$, such that λ_{dk} decays exponentially with the increase of the rank k: $|\lambda_{dk}| \leq C_1 \cdot \exp(-C_2 \cdot k^\gamma)$

Lemma 1. (The Lipschitz condition for MSE loss): A loss function $\mathcal{L}(f(x), y)$ is the MSE loss, and $f(x) \in [-C/2, C/2]$, then $\mathcal{L}(f(x), y)$ be $2C$-Lipschitz.

Detailed proof of lemma 1 can be found in Appendix Section C, available online. Through the above assumptions and lemmas, we can derive the error upper bound for the decomposition of the global value function Q_{tot} under the L_2 regularized models(see Appendix Section C), available online.

Theorem 1: Let $\mathcal{L}$ be $2C_Q - Lipschitz$, $\delta \in (0, 1]$, and Assumption 1 hold with constants $\{C_1, C_2, \gamma\}$. The generalization error for the optimal global value function $Q^\star_{tot}$ be $\mathcal{E}(Q^\star_{tot})$ and the generalization error for the empirical risk minimizer $\widehat{Q^{D,K}_{tot}}$ be $\mathcal{E}(\widehat{Q^{D,K}_{tot}})$ with $K = \{K_d\}^D_{d=1}$. Then, we have for L_2-regularized ERM, where $\|w_{dk}(u^i_t)\|_2 \leq B_{w,2}$, $1 \leq d \leq D$, $B_{h,2} = \sup_h \|\bar{Q}_i\|_2$, and $\|\boldsymbol{\lambda}\|_2 \leq B_{\lambda,2}$ where $\boldsymbol{\lambda} = \{\{\lambda_{dk}\}^{K_d}_{k=1}\}^D_{d=1}$, with probability at least $1 - \delta$,

$$\mathcal{E}(\widehat{Q^{D,K}_{tot}}) - \mathcal{E}(Q^\star_{tot}) \leq 4C_Q B_{\lambda,2} B_{h,2} \sum_{d=1}^{D} \left((B_{w,2})^d \sqrt{\frac{K_d}{n_s}} \right) + \sum_{d=D+1}^{n} \frac{C_1}{C_2} \exp\left(-(K_d + 1)^\gamma\right) + \left(2(C_Q)^2 + C_Q\right) \sqrt{\frac{2\log(4/\delta)}{n_s}} \tag{8}$$

where n_s denotes the number of samples, K_d denotes the tensor rank for each order d.

In Section III-A, we decompose the local value functions of the same order d into the outer products of rank-one tensors. In this paper, however, our focus is not on the rank size for the same order, but rather on the selection of the maximum order D in the decomposition of the global value function. Therefore, we set K_d (the tensor rank) for all orders d to a constant C_K. Under these circumstances, the error upper bound becomes:

$$\mathcal{E}(\widehat{Q_{tot}^{D}}) - \mathcal{E}(Q_{tot}^{\star}) \leq C_{final1} \cdot \frac{(B_{w,2})^{D} - 1}{B_{w,2} - 1} - C_{final2} \cdot D + C_{final3} \quad (9)$$

where $C_{final1} = 4C_Q B_{\lambda,2} B_{h,2} B_{w,2} \sqrt{\frac{C_K}{n_s}}$, $C_{final2} = \frac{C_1}{C_2} \exp(-(C_K+1)^{\gamma})$, $C_{final3} = (2(C_Q)^2 + C_Q)\sqrt{\frac{2\log(4/\delta)}{n_s}} + \frac{C_1}{C_2} \exp(-(C_K+1)^{\gamma})(n-1)$.

Eq. (9) reveals the relationship between the error upper bound and the maximum order D of the value function decomposition, which highlights the trade-off between computational cost and model accuracy. As D increases, the complexity of the model (the first term in (9)) increases, which means higher computational cost, while the approximation error (the second term in (9)) decreases. This indicates that selecting a moderate D can effectively reduce the error and improve the algorithm performance. The related experiments will be presented in Appendix Section D, available online.

C. Graph Structure Generation Policy

At each time step, we represent the graph policy as $\rho(A_t \mid \boldsymbol{\tau}_t)$. With the graph policy ρ, we can obtain the adjacency matrix $A_t \sim \rho$. As described in Section III-A, the decomposition of a global value function is associated with A_t. However, the decomposition of the global value function should not be static, and the collaborative relationship among the agents should be dynamic when facing different environmental states. While the factor graph can decompose the global value function, its decomposition corresponds to a fixed adjacency matrix. Similarly, DCG decomposes the global value function into a fixed, fully connected structure. Therefore, this section proposes a graph policy ρ that can dynamically generate real-time graph structures that represent the changing collaborative relationships among agents.

We propose a graph policy ρ that can dynamically generate real-time graph structures to address this issue. We represent the graph policy $\rho(A_t \mid \boldsymbol{\tau}_t) = \rho(A_t \mid \boldsymbol{\tau}_t; \varphi)$ using a network parameterized by φ. The graph policy network shares the same RNN network as the Q-value function network but does not participate in the parameter update. We use $h_t^i = h_\psi(\cdot \mid h_{t-1}^i, o_t^i, u_{t-1}^i)$ as the input of the graph policy network. To better represent the global relationships of all agents, we use the global states s_t as the input of the hypernetwork to obtain the parameters of the graph policy network. We refer to the network structure design of QMIX and train the weights W of the hypernetwork without absolute value restrictions to obtain more information in $\boldsymbol{o}_t$.

We employ a two-layer hypernetwork and incorporate a fully connected layer as an action layer to initialize the probability of each edge equally. Subsequently, we apply a softmax activation function to process the output of the graph structured network, constraining the output between 0 and 1, in order to derive the probability of edge connections corresponding to the adjacency matrix A_t. Specifically, the graph policy $\rho(A_t \mid \boldsymbol{\tau}_t)$ outputs a matrix $P(A_t) = \{a_{ij}\}^{N \times M}$, representing the probabilities of edge connections corresponding to the adjacency matrix A_t, where N is the number of intelligent agents (variable nodes), M is the number of local value functions (factor nodes), and a_{ij} represents the probability of connection between the ith intelligent agent and the jth local value function ($\sum_{i=0}^{N} a_{ij} = 1$). For each local value function Q_j, the probability distribution of its connections with all N agents is defined as a multinomial distribution.

Definition 1: In the graph policy $\rho(A_t \mid \tau_t)$, for each local value function Q_j, randomly Variable $X = (X_1, X_2, \ldots, X_N)$ represents the number of connections between Q_j and N agents, where X_i represents the number of times the ith agent is connected to Q_j. X satisfies:

1) $X_i \geq 0 (1 \leq i \leq N)$, and $X_1 + X_2 + \cdots + X_N = m$, m is the total number of connections between Q_j and N agents;
2) Let $m_1, m_2, \ldots, m_N$ be any non-negative integer, and $m_1 + m_2 + \cdots + m_N = m$, then the probability of event $\{X_1 = m_1, X_2 = m_2, \ldots, X_N = m_N\}$ occurring is:

$$P\{X_1 = m_1, X_2 = m_2, \ldots, X_N = m_N\} = \frac{m!}{m_1! m_2! \cdots m_N!} {p_1}^{m_1} {p_2}^{m_2} \cdots {p_N}^{m_N} \quad (10)$$

where $p_i \geq 0 (1 \leq i \leq N), p_1 + p_2 + \cdots + p_N = 1$.

Then X conforms to the multinomial distribution, that is, $X \sim P_m(m : p_1, p_2, \ldots, p_N)$.

Proposition 1: In the graph-generation policy $\rho(A_t \mid \boldsymbol{\tau}_t)$, for each local value function Q_j, let $X \sim P_D(D : p_1, p_2, \ldots, p_N)$ be the random variable denoting the number of all agents connected to Q_j. Then the algorithm's maximum factor order is D.

Proposition 2: In the graph policy $\rho(A_t \mid \boldsymbol{\tau}_t)$, for each local value function Q_j, when The number of times it is connected to N agents corresponds to a random variable $X \sim P_D(D : p_1, p_2, \ldots, p_N)$, then each Q_j corresponds to "sub-policy" conforms to a quasi-multinomial distribution, that is, $\rho(Q_j) \sim \tilde{P}_D(D : p_1, p_2, \ldots, p_N)$.

Detailed proof of Propositions 1 and 2 can be found in Appendix Section C, available online. Through the above definitions and inferences, we represent the graph structure generation policy as a a quasi-multinomial distribution, and train it with a neural network. Meanwhile, it can be concluded that by setting the maximum order D of the graph policy network, the factor graph can represent all collaborative relationships of less than or equal to D agents.

Next, we present the Policy improvement lower bound [33], [34] for the graph policy ρ, which provides the theoretical

foundation for learning ρ. As shown later, this bound implies that by maximizing it, we can guarantee policy improvement at each step of the learning process.

Theorem 2. (Graph Policy Improvement Lower Bound): Consider the behavior (trajectory-collecting) graph policy ρ_{old}. For the current graph policy ρ_{new} that we need to improve, we have

$$J(\rho_{new}) - J(\rho_{old}) \geq \frac{1}{1-\gamma} \underset{\substack{s\sim D^{\Pi_{old}} \\ A\sim\rho_{new} \\ u\sim\pi_{old}}}{E} [\mathcal{A}_{\rho_{old},\pi_{old}}(s,A,\boldsymbol{u})] - \frac{C^{\Pi}\gamma}{(1-\gamma)^2} \underset{(s,A)\sim D^{\Pi_{old}}}{E} \left[\left|\frac{\rho_{new}(A|s)}{\rho_{old}(A|s)} - 1\right|\right] \tag{11}$$

where J denotes the objective function optimized in reinforcement learning, $C^{\Pi} = \max_{s\in S} \left| \underset{\substack{A\sim\rho_{new} \\ u\sim\pi_{old}}}{E} [\mathcal{A}_{\rho_{old},\pi_{old}}(s,A,\boldsymbol{u})] \right|$, $\Pi = \rho(A|s)\pi(\boldsymbol{u}|s,A)$ denotes the joint policy of all agent policies.

Detailed proof of Theorem 2 can be found in Appendix Section C, available online. We refer to the first term in the lower bound of Theorem 2 as the surrogate objective (SO), and to the second term as the penalty term (PT). It should be noted that, during policy improvement, as long as the lower bound of each update is positive, the new graph policy is guaranteed to outperform the old one.

To ensure that the lower bound of each policy improvement step remains positive, we need to guarantee that $SO > PT$. In this case, the SO should be maximized while constraining the size of the PT. Maximizing SO can be reduced to maximizing the objective J (18) in Section IV-B. At the same time, (23) in Section IV-B is used to bound PT so that the graph policy can be effectively improved. Taken together, this proof and the loss derivation for the graph policy in Section IV-B provide the theoretical basis for the optimization and convergence of the graph policy ρ.

D. Max-Sum Algorithm

In the previous section, we obtained the local value functions using the Q-value function network. In this section, we will demonstrate how to use the max-sum algorithm [30], [35] on factor graphs to obtain the final action for each agent.

First, we obtain the adjacency matrix A_t from the graph structure generation policy ρ and concatenate it with I_n to form ${A_t}'$. To facilitate the derivation, we consider all factor nodes represented by ${A_t}'$ as a single entity. Then, the action selection process aims to solve the following problem(See appendix C for detailed derivation), available online:

$$\begin{aligned} \boldsymbol{u}_t^* &= \arg\max_{\boldsymbol{u}_t} Q_{tot}(\boldsymbol{\tau}_t, \boldsymbol{u}_t, A_t; \theta, \psi) \\ &= \arg\max_{(\boldsymbol{u}_1,\ldots,\boldsymbol{u}_n)} \left(\sum_{j\in J'} Q_j \left(\{v_i\}_{\{i,j\}\in\mathcal{E}'} \right) \right) \end{aligned} \tag{12}$$

where $\mathcal{J}' = \mathcal{J} \bigcup \{m+i\}_{i=1}^{n}, \mathcal{E}' = \mathcal{E} \bigcup \{\{i,j\} \mid 1 \leq i \leq n, j = j+1 (j = 1 \sim n)\}$.

In illustrating the max-sum algorithm for factor graphs, we will omit moment t's representation. The max-sum algorithm for factor graphs consists of two types of messages. Let $N(x)$ denote the set of neighbors of node x. The message sent from the variable node v_i to the factor node Q_j is:

$$\mu_{v_i\to Q_j}(v_i) = \sum_{Q_k\in N(v_i)\backslash\{Q_j\}} \mu_{Q_k\to v_i}(v_i) + c_{v_i\to Q_j} \tag{13}$$

where $N(v_i)\backslash\{Q_j\}$ denotes the set of nodes in $N(v_i)$ except Q_j, and $c_{v_i\to Q_j}$ is the normalization term.

The message sent from the factor node Q_p to the variable node v_l is:

$$\mu_{Q_p\to v_l}(v_l) = \max_{\boldsymbol{v}_p\backslash v_l} \left(Q_p(\boldsymbol{v}_p) + \sum_{v_k\in N(Q_p)\backslash\{v_l\}} \mu_{v_k\to Q_p}(v_k) \right) + c_{Q_p\to v_l} \tag{14}$$

where $\boldsymbol{v}_p\backslash v_l$ denotes the parameters in the local function Q_p except v_l, and $c_{Q_p\to v_l}$ is the normalization term. The agent continuously sends, accepts, and recomputes messages until the values of the messages converge. After convergence, the behavior of the agent is given by the following equation:

$$\boldsymbol{u}^* = \left[\arg\max_{u_i} \sum_{Q_k\in N(v_i)} \mu_{Q_k\to v_i}(v_i) \right]_{i=1}^{n} \tag{15}$$

However, this convergence exists only in acyclic factor graphs, meaning that the exact solution can only be achieved in such graphs through the max-sum algorithm. When the factor graph contains loops, as depicted in Fig. 1, the general max-sum algorithm cannot guarantee an exact solution due to the uncertain manner nodes receive messages containing messages sent from the node, leading to message explosion. Asynchronous messaging [36] and messaging with damping [37] can mitigate this issue.

IV. Loss Function

The value function and graph policy networks use different loss functions and alternate updates. This section will explain the form of loss construction for each network.

A. Loss of the Q-Value Function Network

We update the Q-value function network using the loss form that is referred to in DQN. We maintain a target network [28] and a replay buffer [38]. We also deposit the adjacency matrix A, obtained from the graph policy, into the replay buffer. This way, we obtain the decomposed form of the global value function Q_{tot} through the adjacency matrix A. In contrast, the decomposed local value function Q_j is obtained through the θ, ψ parameterized Q-value function network. Finally, we use the globally optimal joint action $\boldsymbol{u}^*$ obtained by the max-sum algorithm. To achieve

this, we learn θ, ψ by minimizing the TD error, as follows:

$$\mathcal{L}_Q(\boldsymbol{\tau}, A, \boldsymbol{u}, r; \theta, \psi) := E\left[\frac{1}{T}\sum_{t=0}^{T}\left(Q_{tot}(\boldsymbol{\tau}_t, \boldsymbol{u}_t, A_t) - y^{dqn}\left(r_t, \tau_{t+1}, A_{t+1}; \bar{\theta}, \bar{\psi}\right)\right)^2\right] \tag{16}$$

where $\boldsymbol{\tau} = \{\boldsymbol{\tau}_t\}_{t=1}^{T}$ (note: $\boldsymbol{\tau}$ in this paper only denotes the set of observation-action histories $\boldsymbol{\tau}_t$, not the trajectory of the whole task), and r_t is the reward for performing action $\boldsymbol{u}_t$ transitions to $\boldsymbol{\tau}_{t+1}$ in the observation history $\boldsymbol{\tau}_t$. $y^{dqn}(r_t, \boldsymbol{\tau}_{t+1}, A_{t+1}; \bar{\theta}, \bar{\psi}) = r_t + \gamma Q_{tot}(\boldsymbol{\tau}_{t+1}, u^*_{t+1}, A_{t+1}; \bar{\theta}, \bar{\psi})$, $\boldsymbol{u}^*_{t+1} = [\arg\max_{u^i_{t+1}} \sum_{Q_k \in N(v_i)} \mu_{Q_k \to v_i}(v_i)]_{i=1}^{n}, \bar{\theta}, \bar{\psi}$ is the parameter copied periodically from θ, ψ.

B. Loss of Graph Policy Network

MARL aims to maximize the expected discounted sum of future rewards. Therefore, the natural objective of the graph policy network is to find the optimal graph policy $\rho(A_t \mid \boldsymbol{\tau}_t; \varphi)$ that achieves this goal. However, if we use a DQN-style loss function, the gradient cannot be returned to the graph policy network, making training impossible. To address this issue, we adopt the policy gradient approach [6] and treat the graph policy $\rho(A_t \mid \boldsymbol{\tau}_t; \varphi)$ as the action policy $\pi(\boldsymbol{u}_t \mid \boldsymbol{\tau}_t)$, and use the policy gradient approach to design the graph policy network's loss function.

The loss function for the graph policy network is:

$$\mathcal{L}_G(\boldsymbol{\tau}, A, \boldsymbol{u}, r; \varphi) = \mathcal{L}_{PG} - \lambda_{\mathcal{H}}\mathcal{L}_{entropy} \tag{17}$$

where $\lambda_{\mathcal{H}}$ is the weight constant of $\mathcal{L}_{entropy}$. Our objective is to maximize the expected discounted return, for which we define the maximization objective function as follows:

$$\underset{\varphi}{maximize}\ \ J(\theta, \psi, \varphi) = \underset{\varphi}{maximize}\ \ v_{\Pi_{\theta,\psi,\varphi}}(s_0) \tag{18}$$

where $\Pi_{\theta,\psi,\varphi} = \Pi_{\theta,\psi,\varphi}(\boldsymbol{u}, A \mid s) = \rho_\varphi(A \mid s)\pi_{\theta,\psi}(\boldsymbol{u} \mid s, A)$ denotes the joint policy of all agent policies and graph policies, $v_{\Pi_{\theta,\psi,\varphi}}(s_0)$ is the policy $\Pi_{\theta,\psi,\varphi}$ under s_0 as a function of the state value of s_0. The derivation of the objective function $J(\cdot)$ yields (see Appendix Section C for the corresponding derivation), available online:

$$\nabla_\varphi J(\theta, \psi, \varphi) = \underset{(\boldsymbol{\tau}_t, \boldsymbol{u}_t, A_t) \sim \mathcal{T}}{E}\left[Q_{tot}(\boldsymbol{\tau}_t, \boldsymbol{u}_t, A_t)\nabla \ln \rho_\varphi(A_t | \boldsymbol{\tau}_t)\right] \tag{19}$$

We refer to the PPO [6], which uses importance sampling for graph policies to improve training efficiency:

$$\nabla_\varphi J = \underset{(\boldsymbol{\tau}_t, \boldsymbol{u}_t, A_t) \sim \mathcal{T}_{old}}{E}\left[\frac{\rho_\varphi(A_t \mid \boldsymbol{\tau}_t)}{\rho_{\varphi_{old}}(A_t \mid \boldsymbol{\tau}_t)} Q_{tot}(\boldsymbol{\tau}_t, \boldsymbol{u}_t, A_t) \nabla \ln \rho_\varphi(A_t \mid \boldsymbol{\tau}_t)\right] \tag{20}$$

In order to get a more accurate graph policy ρ, we use the local value function Q_j instead of the global value function Q_{tot} for calculation. The graph policy ρ can be decomposed into "sub-policy" in which all local value functions are connected to the agent. Comparing the value function used in the actor-critic algorithm to evaluate the quality of the policy, in DDFG the global value function Q_{tot} can evaluate the graph policy ρ Then it is natural to use the local value function to evaluate the quality of the "sub-policy". Essentially, this is a credit allocation idea. The global value function Q_{tot} is the sum of the local value functions Q_j, The larger Q_j, the greater its contribution to Q_{tot} as Q_j whole, which means that the corresponding local The greater the contribution of the value function Q_j to the "sub-policy" connected to the agent. Therefore, the "sub-policy" should be accurately evaluated by Q_j, then $\nabla_\varphi J$ becomes:

$$\nabla_\varphi J = \underset{(\boldsymbol{\tau}_t, \boldsymbol{u}_t, A_t) \sim \mathcal{T}_{old}}{E}\left[\sum_{j \in \mathcal{J}}\left(\frac{P_{\varphi,j}(m_j)}{P_{\varphi_{old},j}(m_j)} Q_j\left(\boldsymbol{\tau}^j_t, \boldsymbol{u}^j_t, A_t\right)\right) \nabla \ln \rho_\varphi(A_t \mid \boldsymbol{\tau}_t)\right] \tag{21}$$

where $P_{\varphi,j}(m_j) = \sum_{\{m_{j,1},\ldots,m_{j,N}\} \in H_m} p_{j,1}^{m_{j,1}} \cdots p_{j,i}^{m_{j,i}} \cdots p_{j,N}^{m_{j,N}}$, $p_{j,i}$ represents the probability that the local value function Q_j is connected to agent i, $m_{j,i}$ represents the number of times the local value function Q_j is connected to agent i, $m_j = \{m_{j,i}\}_{i=1-N}$.

Proposition 3: In the graph policy $\rho(A_t \mid \boldsymbol{\tau}_t)$, for each local value function Q_j, when Its corresponding "sub-policy" $\rho(Q_j) \sim \tilde{P}_D(D : p_1, p_2, \ldots, p_N)$, then (21) can be derived from (20).

Detailed proof of Proposition 3 can be found in Appendix Section C, available online.

Generalized advantage estimation (GAE) [39] is adopted to estimate the advantage function $\mathcal{A}$ and use the advantage function $\mathcal{A}_j$ instead of the local value function Q_j, thus reducing the variance:

$$\hat{\mathcal{A}}^{GAE}_j\left(\boldsymbol{\tau}^j_t, \boldsymbol{u}^j_t, A_t\right) := \sum_{t=0}^{\infty}(\gamma\lambda_{GAE})^l \delta^{Q_j}_{t+l} \tag{22}$$

where λ_{GAE} is the discount factor in GAE, $\delta^{Q_j}_t = Q_j(\boldsymbol{\tau}^j_t, \boldsymbol{u}^j_t, A) - V_j(\boldsymbol{\tau}^j_t)$. We construct the relationship between V_{tot} and V_j using the same A_t and train V_{tot} with TD-error (see Appendix Section B for detailed descriptions), available online.

We use the clip function to constrain Importance sampling, and derive $\nabla_\varphi \mathcal{L}_{PG}$:

$$\nabla_\varphi \mathcal{L}_{PG} = -E_{(\boldsymbol{\tau}_t, \boldsymbol{u}_t, A_t) \sim \mathcal{D}}\left[\sum_j \min\left(r^j_t(\varphi)\hat{\mathcal{A}}_j, f_{clip}(r^j_t(\varphi), \epsilon)\hat{\mathcal{A}}_j\right) \nabla \ln \rho_\varphi(A_t | \boldsymbol{\tau}_t)\right] \tag{23}$$

where $r^j_t(\varphi) = P_{\varphi,j}(m_j)/P_{\varphi_{old},j}(m_j)$, $f_{clip}(r^j_t(\varphi), \epsilon) = clip(r^j_t(\varphi), 1-\epsilon, 1+\epsilon)$, $\mathcal{D}$ is the replay buffer of the graph

policy network and is independent of the replay buffer of the Q-value function network.

$\mathcal{L}_{entropy}$ is the loss function of the policy entropy. We add the policy entropy to the loss to improve the exploration ability of the graph policy network:

$$\begin{aligned}\mathcal{L}_{entropy} &= -E_{\boldsymbol{\tau}_t\sim\mathcal{D}}[\mathcal{H}\left(\rho(\cdot \mid \boldsymbol{\tau}_t;\varphi)\right)] \\ &= -E_{\boldsymbol{\tau}_t\sim\mathcal{D}}\left[\sum_i\sum_j \rho_{ij}(\cdot \mid \boldsymbol{\tau}_t;\varphi)\log\left(\rho_{ij}(\cdot \mid \boldsymbol{\tau}_t;\varphi)\right)\right]\end{aligned} \tag{24}$$

where $\rho_{ij}(\cdot \mid \tau_t;\varphi)$ denotes the probability of connecting agent i with the local function Q_j.

V. Experiment

This section delineates a comparative analysis of the Dynamic Deep Factor Graph (DDFG) algorithm against a suite of state-of-the-art algorithms including Value Decomposition Networks (VDN) [8], QMIX [9], Combined-Weighted QMIX (CW-QMIX)/Optimistically Weighted QMIX (OW-QMIX) [13], Multi-Agent Proximal Policy Optimization (MAPPO) [5], QTRAN [11], QPLEX [12], Deep Coordination Graphs (DCG) [15], Sparse Optimistic Policy Coordination Graph (SOPCG) [18], and Coordination Among Sparsely Communicating Entities (CASEC) [19]. The implementation of DDFG, constructed using PyTorch, is publicly available alongside baseline experiment codes in the designated repository.

To elucidate the efficacy of DDFG, we evaluate it across two distinct scenarios, detailed further in Appendix Section D, available online, with regards to hyperparameter configuration and additional experiments: (1) an advanced Higher-Order Predator-Prey model [15], and (2) the StarCraft II Multi-agent Challenge (SMAC) [40].

A. Higher-Order Predator-Prey

The Predator-Prey environment, as utilized in DCG, was augmented to engender a Higher-Order Predator-Prey (HO-Predator-Prey) scenario, exhibiting increased complexity.

Original Predator-Prey Scenario: This model positions the prey on a grid, where agents may execute capture actions. Successive captures by two or more agents on the prey, positioned in adjacent squares (top, bottom, left, or right), result in a collective reward of r. Conversely, an unsuccessful solo capture action inflicts a sub-reward penalty of p.

Higher-order Predator-Prey (HO-Predator-Prey): The HO-Predator-Prey scenario (as shown in Fig. 3) extends the adjacency to include eight squares around the prey: Upper, Lower, Left, Right, Upper Left, Upper Right, Lower Left, and Lower Right, necessitating X or more simultaneous capture actions for success, with X set to 3. Furthermore, the decomposition of the DDFG value function is constrained to a highest order of 3, linking each local value function with no more than three agents.

Our experimental framework evaluated the influence of varying penalty values p (0, -0.5, -1, -1.5), with the outcomes depicted in Fig. 4. In scenarios devoid of penalties (Fig. 4(a)),

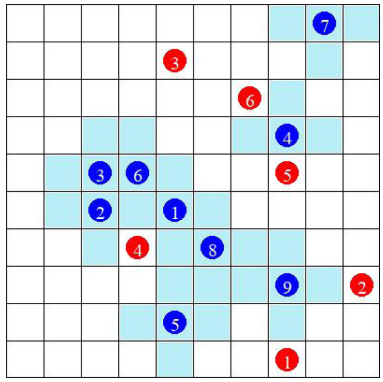

Fig. 3. HO-Predator-Prey environment. Predators are marked in blue, prey is marked in red, and the blue grid represents the range of movement of the predator.

a broad array of algorithms including Value Decomposition Networks (VDN), QMIX, Combined-Weighted QMIX (CW-QMIX), Optimistically Weighted QMIX (OW-QMIX), QPLEX, Deep Coordination Graphs (DCG), Sparse Optimistic Policy Coordination Graph (SOPCG), Coordination Among Sparsely Communicating Entities (CASEC), and Dynamic Deep Factor Graph (DDFG) converged to the optimal solution. In contrast, QTRAN and MAPPO only managed to achieve partial success. As a policy gradient method, MAPPO's performance in the HO-Predator-Prey environment is not as good as that of other value-based methods. Notably, DDFG exhibited a reduced rate of convergence, attributable to its ongoing optimization of graphical policies concurrent with value function fitting, which inherently diminishes convergence velocity. Upon the introduction of penalties (Fig. 4(b)–(d)), all baseline algorithms except for CW-QMIX, OW-QMIX, DCG, SOPCG, CASEC, and DDFG succumbed to failure, unable to circumvent the pitfall of relative overgeneralization. As penalties intensified, CW-QMIX, OW-QMIX, DCG, SOPCG, and CASEC progressively struggled to attain the optimal solution, with complete failures recorded in certain trials.

When penalty p was set to -0.5, CW-QMIX exhibited partial success; however, as the penalty increased to -1 or beyond, it failed outright. OW-QMIX, despite partial successes across varying penalty levels, encountered difficulties in accurate action learning as penalties escalated. This is in stark contrast to the outright failure of QMIX, where CW-QMIX/OW-QMIX's partial successes can be attributed to the Weighted QMIX operator's non-monotonic function mapping, thus boosting algorithm performance. Despite DCG's ability to learn successfully across all scenarios, its average reward and convergence rate fell short of DDFG's performance. This discrepancy is linked to DCG's static coordination graph structure, which hinders learning of collaborative policies involving more than two agents due to redundant information in message passing and an inability to adapt to complex collaborative decisions among three or more agents. Conversely, SOPCG and CASEC, despite their advancement in dynamic sparse graph structures, were hampered by the intrinsic limitations of the coordination graph's expressive capacity, thus failing to achieve comprehensive success. DDFG,

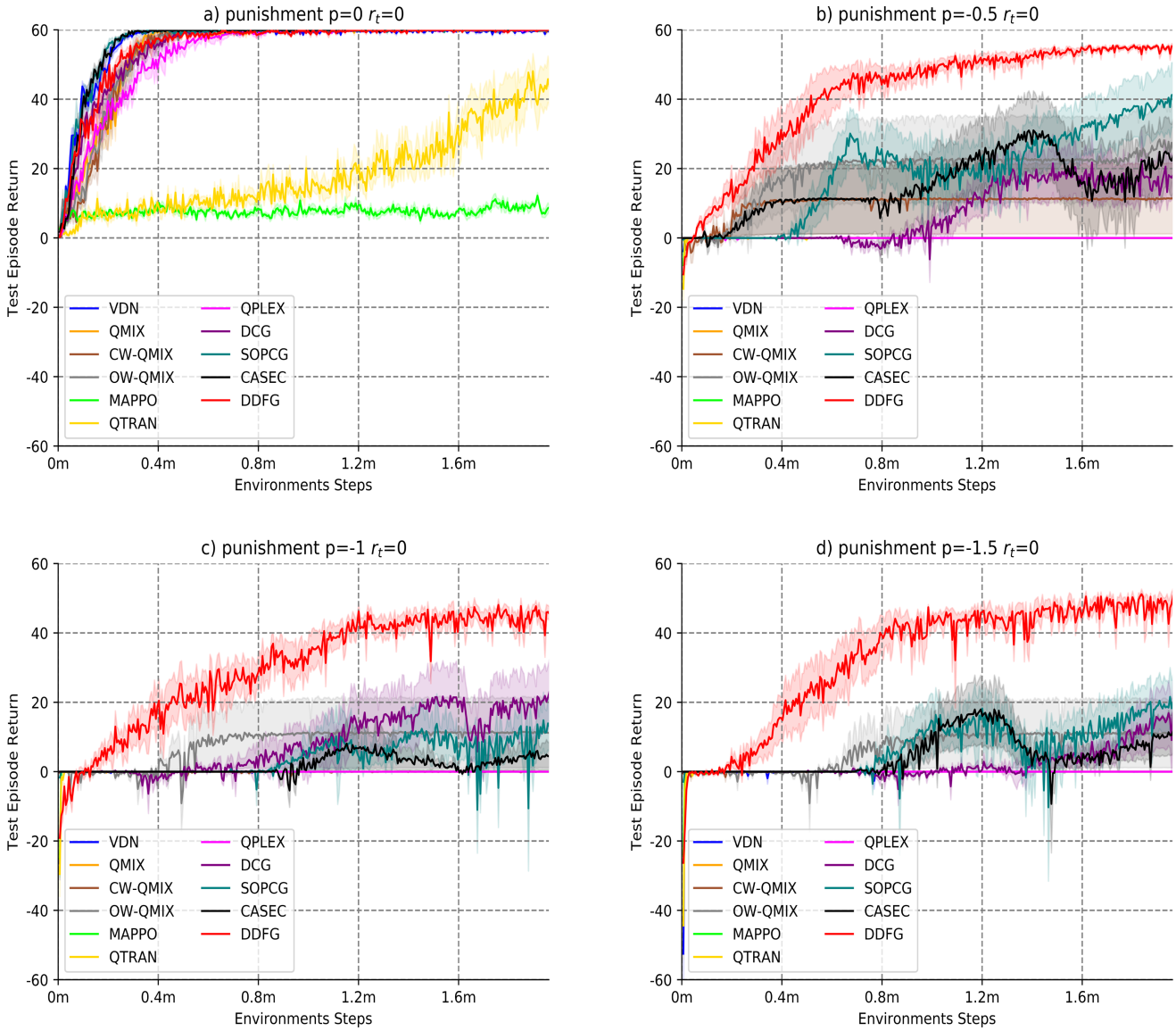

Fig. 4. Median test return for the Higher-order Predator-Prey task with different penalties p(0,-0.5,-1,-1.5), comparing DDFG and baselines.

however, demonstrated proficiency in learning optimal policies across all scenarios by dynamically modeling cooperation among predators through graph policy networks and employing factor graphs for collaborative decision-making among multiple agents, facilitated by the max-sum algorithm.

Further experimentation, incorporating a time-step penalty $r_t = -0.1$ while maintaining constant penalty values p (-0.5, -1), is presented in Fig. 5. The imposition of a temporal penalty r_t, equating to a negative reward at each time step when agents remain stationary, ostensibly complicates convergence by augmenting negative rewards. Nonetheless, this approach simultaneously fosters exploration. In scenarios with $p = -0.5$, the inclusion of r_t enabled algorithms such as VDN, QMIX, CW-QMIX/OW-QMIX, SOPCG, and CASEC to enhance exploration, thereby overcoming relative overgeneralization to achieve optimal solution convergence in certain instances. Yet, in a majority of cases, these algorithms still faced outright failure. With $p = -1$, the augmentation of r_t failed to facilitate success for these algorithms. Particularly for SOPCG and CASEC, the substantial increase in r_t markedly impeded algorithmic success probability. The dynamic sparse coordination graph structure exhibited no discernible advantage in the HO-Predator-Prey context, with CASEC's performance even trailing behind DCG. For DCG, the negative temporal reward r_t adversely affected convergence, rendering outcomes in scenarios with $p = -1$ less favorable compared to those without r_t. Conversely, DDFG consistently converged to the optimal policy irrespective of r_t's presence, though the introduction of r_t inevitably slowed DDFG's convergence by accruing more negative rewards.

Finally, this paper visualizes the dynamic structure of factor graphs in HO-Predator-Prey tasks, as shown in Fig. 6. We tested the DDFG after two million steps of training in an experimental scenario with a penalty of $p = -1.5$ and $r = 0$. In an episode shown in Figs. 6 and 9 agents successfully captured 6 prey

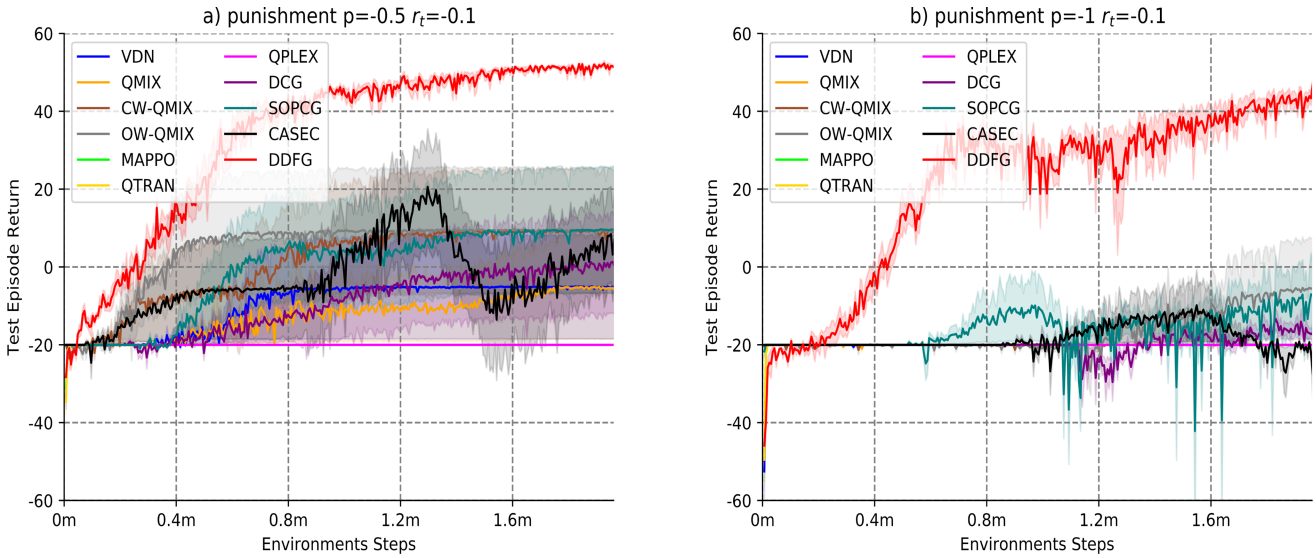

Fig. 5. Median test return for the Higher-order Predator-Prey task with r_t, comparing DDFG and baselines.

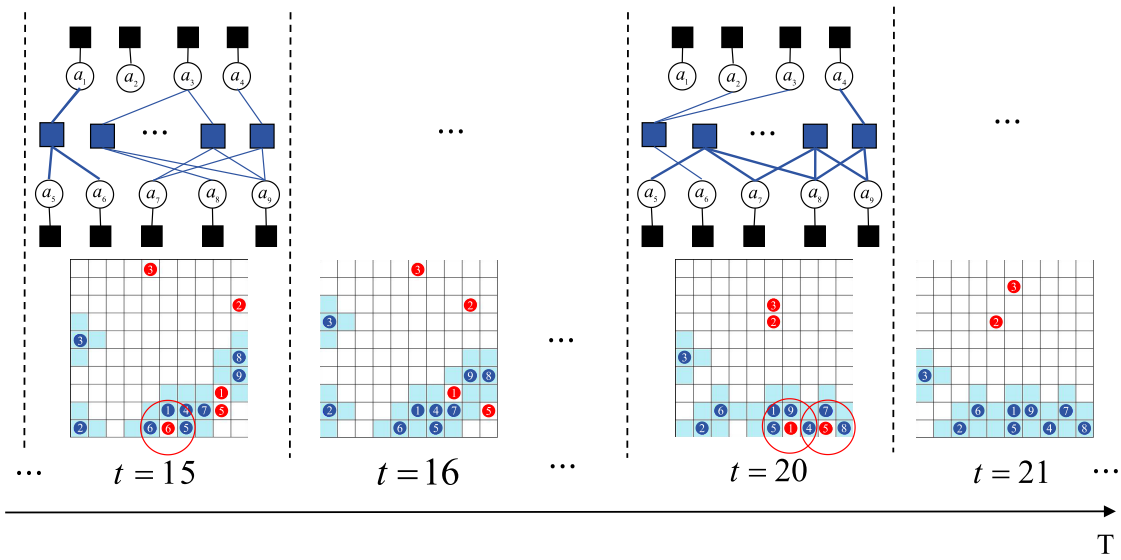

Fig. 6. The dynamic changes in the graph structure of DDFG in the Higher-order Predator-Prey task (performing the "capture" action).

within 50 time steps (ending 150 time steps earlier). This paper selects scenarios where agents cooperate to capture prey at two moments, $t = 15$ and $t = 20$, and visualizes the factor graph structure at these two moments.

At $t = 15$, agents 1, 4, 5, and 6 collectively encompass prey 6 and successfully captured it. The actions executed at $t = 15$ reveal that agents 1, 5, and 6 initiated the 'capture' action leading to the successful capture of prey 6. Insight from the factor graph structure at $t = 15$ demonstrates that, the dynamic graph policy generated joint value function nodes for agents 1, 5, and 6, managing to orchestrate their capture actions effectively.

At $t = 20$, agents 1, 4, 5, and 9 encircle prey 1, while agents 4, 7, and 8 concurrently surround prey 5, leading to the capture of both prey. In reference to the list of actions executed by the agents, agents 4, 5, 7, 8, and 9 are all noted for executing the 'capture' action. This means that agents 4, 5, and 9 are responsible for capturing prey 1, whereas agents 4, 7, 8 are accountable for the capture of prey 5. Notably, agent 1 is not involved in the capture, while agent 4 contributes to both capture actions. This pattern is reflected in the factor graph structure at time $t = 20$. This reflects that the graph structure generated by the dynamic graph policy will directly guide the coordinated actions between agents. Moreover, even though the dynamic graph policy does not successfully generate joint nodes for agents 4, 5, 9 or 4, 7, 8, the capture action is still successfully completed. This highlights the significance of the message passing algorithm. Starting from the factor nodes, (14) and (13) are alternately called to propagate and aggregate the value function information between the factor nodes and variable nodes, which ultimately leads to the successful capture of two preys by five agents.

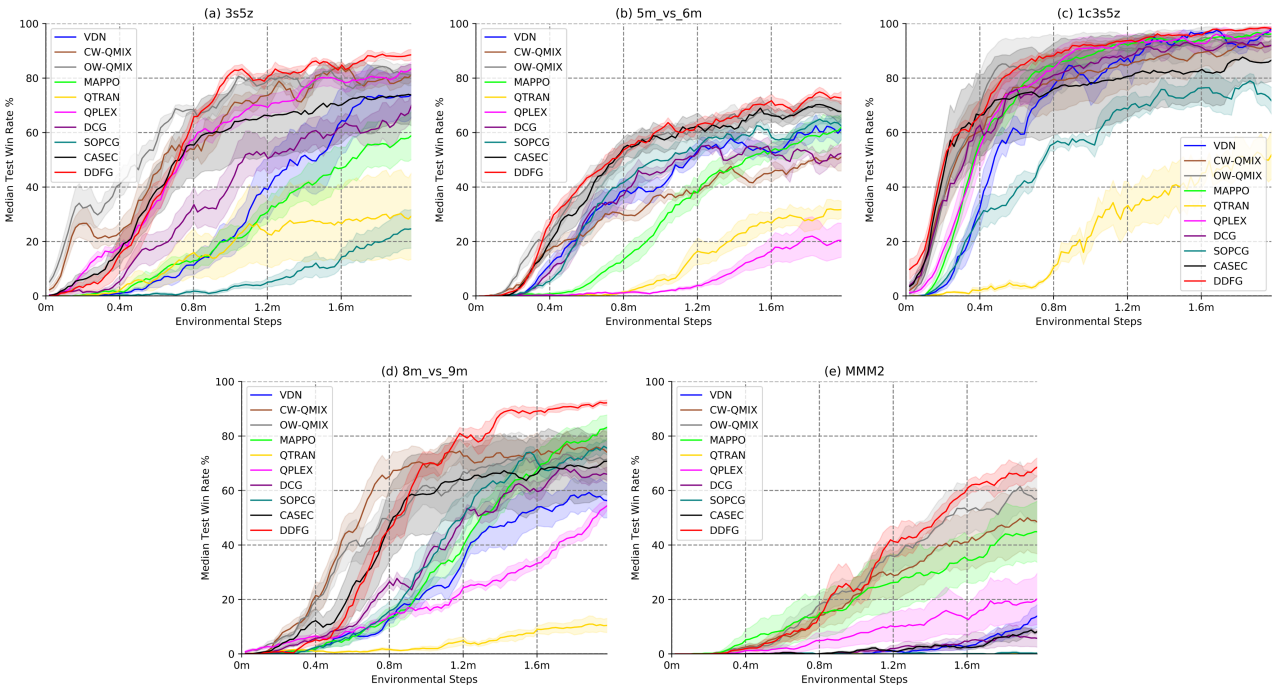

Fig. 7. Median test win rate % for the SMAC task, comparing DDFG and baselines.

In summary, the disparity in the factor graph structures between $t = 15$ and $t = 20$ as indicated in Fig. 6 underscores the ability of the dynamic graph policy introduced in this paper to generate factor graph structures that adapt to the real-time changes. Meanwhile, the above discussion confirms that the dynamic graph policy has the ability to generate real-time factor graph structures, thereby guiding multi-agent collaboration. Additionally, the scenario at $t = 20$ highlights that even without the generation of an 'absolutely correct' joint value function node, the message passing algorithm can ensure successful coordinated actions among agents, hence proving the resilience of the dynamic graph policy.

B. SMAC

Transitioning to the SMAC domain, we confront a series of more daunting experiments within the StarCraft Multi-Intelligent Challenge Benchmark, evaluating empirical performance across various environments through exploration and FIFO buffers. For our purposes, we have selected a total of five maps:: (a) 3s5z (Easy), (b) $5m\_vs\_6m$ (Hard), (c) 1c3s5z (hard), (d) $8m\_vs\_9m$(hard), (e) MMM2 (superhard), with the opponent AI set to hard in both instances.

The experimental results, showcased in Fig. 7, indicate that in the $5m\_vs\_6m$ map, DDFG matches the performance of OW-QMIX and CASEC, outperforming the remainder of the baselines. In contrast, on other maps(3s5z, 1c3s5z, $8m\_vs\_9m$ and MMM2), DDFG demonstrates superior performance relative to all baseline models. Initial learning velocity for DDFG is slower due to two primary factors: the SMAC scenarios' immunity to relative overgeneralization, preventing baseline algorithms from stagnating at local optima, and the temporal requirements for DDFG's graph policy to identify the optimal structural decomposition. This temporal lag results in DDFG exhibiting a slower initial learning curve when compared to some baselines during the experiment's initial phase.

The experiments validate DDFG's scalability and efficacy in complex tasks like SMAC, even in the absence of relative overgeneralization. Notably, DDFG demonstrates superior performance to DCG in all scenarios by generating dynamic graph structures that facilitate adaptive agent collaboration against adversaries. DDFG's ability to control varied agents for joint decision-making against enemies secures a higher victory rate. SOPCG and CASEC exhibit diminished success rates on the MMM2 map, while maintaining average performance on the $8m\_vs\_9m$ scenario. This discrepancy underscores the variability and stability concerns associated with dynamically sparse graph structures in fluctuating contexts, as evidenced by the divergent success rates across different maps. DDFG's employment of high-order value function networks and the max-sum algorithm for global policy optimization culminates in elevated success rates across varied SMAC maps.

Fig. 8 presents the visualization of the factor-graph structure in the StarCraft II experiment. This paper uses the 3s5z scenario and tests the DDFG after two million steps of training. In an episode displayed in the figure, our team controls 3 Stalkers and 5 Zealots against the enemy's identical setup of 3 Stalkers and 5 Zealots and eventually secures the victory.

At $t = 24$, agents 4, 5, and 7 collaboratively attack a Zealot. The graph-structured policy successfully learns the cooperation between 4 and 7, and manipulates 5 through separate factor nodes to accomplish the collective cooperation of the three agents. At the same time, the graph-structured policy constructs cooperation between agents 1, 3 and 2, 3. The message passing is

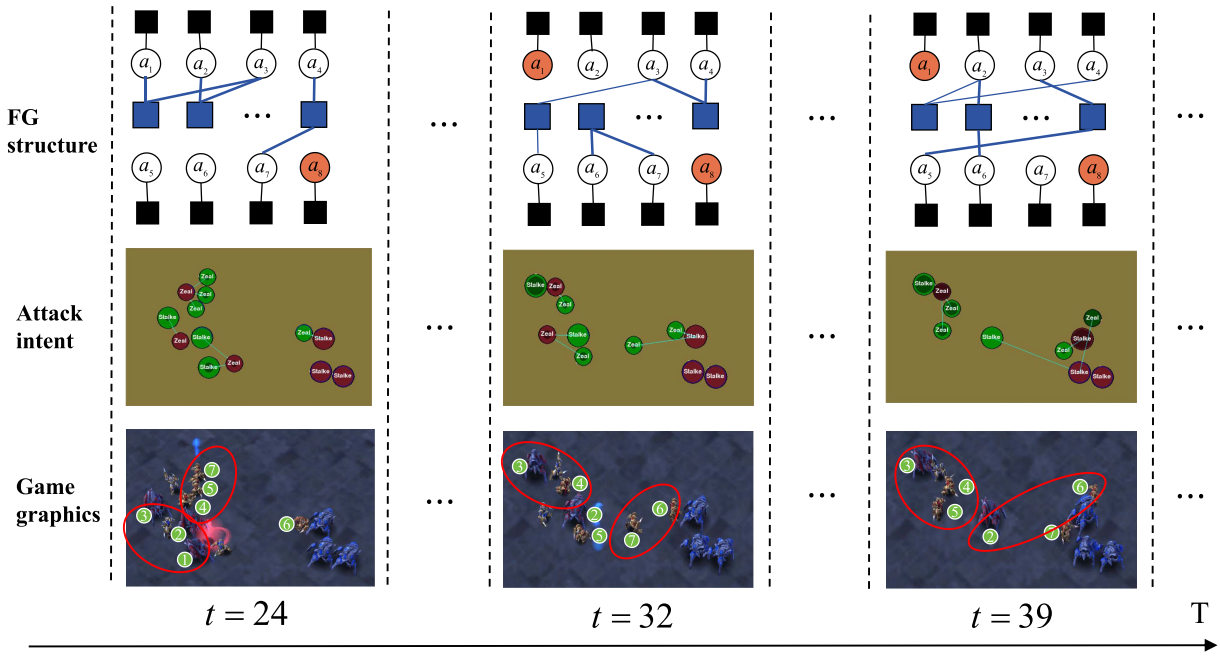

Fig. 8. In the StarCraft II task (3s5z map), the dynamic changes in the graph structure of DDFG. The first row shows the dynamic FG structure at different time moments t, where the red agent nodes represent dead agents. The second row indicates the intent to attack from our agents, with the green nodes being our agents and the red ones being the enemies. The third row shows the corresponding game screen, where the red circle marks collaboration between agents, and the green numbers are agent IDs (corresponding to the FG structure in the first row).

conducted through the max-sum algorithm on the factor graph, ultimately achieving the collaborative attack on the enemy by agents 1 and 3.

At $t = 32$, agents 3, 4 collaboratively attack a Stalker, while agents 6 and 7 jointly attack a Zealot. The graph-structured policy successfully learns both cooperations simultaneously. At $t = 39$, agents 2 and 6 attack a Stalker, while agents 3, 4, and 5 collaborate to attack a Zealot. The graph-structured policy learns the cooperation of 2 and 6. As for the collaboration of agents 3, 4, and 5, the policy only learns the cooperation between 3 and 5, but also manipulates 4 through a separate factor node to ultimately complete the attack.

In summary, Fig. 8 displays different factor graph structures at three different moments, validating the effectiveness of the graph-structured policy. In most situations, the graph-structured policy can directly create corresponding factor nodes to accomplish collaborative attacks. In other cases, the policy can pass messages among different agents by executing the max-sum algorithm on the factor graph.

C. Ablation Study

The Dynamic Deep Factor Graph (DDFG) algorithm is distinguished by its integration of a value function module and a graph generation policy. As delineated in Appendix Section B, available online, a notable inference is that DDFG, when constrained such that every pair of agents is connected to a factor node, effectively reduces to the Deep Coordination Graph (DCG) framework. This scenario can be interpreted as DDFG operating under a static graph structure. DCG employs a fully connected graph structure, and as evidenced in Figs. 4, 5, and 7, it achieves a measurable degree of success/win rate across various scenarios. This observation underscores the intrinsic value of the function module within the algorithm. This section aims to elucidate the contributions of graph generation policies through the deployment of ablation studies.

First, we initiate a comparison between dynamic and fixed graph policies. Observations from Figs. 4, 5, and 7 reveal that DDFG consistently outperforms DCG across all tested environments. This disparity suggests that static graph strategies are inadequate in encapsulating the dynamic collaborative relationships among agents within the environment. Conversely, the graph generation policy introduced in this work adeptly captures these collaborative dynamics, thereby enhancing overall algorithmic performance. Moreover, DDFG consistently demonstrates a superior learning velocity compared to DCG, implying that the iterative alternation between updating the dynamic graph policy and the value function network contributes to expedited learning processes.

Second, the study contrasts the dynamic graph policy with a random graph policy, with results depicted in Fig. 9. The random graph policy is characterized by a uniform probability distribution governing the connectivity between the value function and agents, effectively equating the local value function to a uniform connection probability among agents. Selecting specific scenarios/maps within the HO-Predator-Prey and SMAC frameworks for experimentation, Fig. 9 delineates a pronounced divergence in performance between the dynamic and random graph policies. This divergence validates the dynamic graph policy's capability to adaptively learn the evolution of collaborative relationships among agents across different temporal junctures, a feat unachievable by the random graph policy. Furthermore, in the context of HO-Predator-Prey experiments, the application of a random graph policy within the DDFG framework engendered

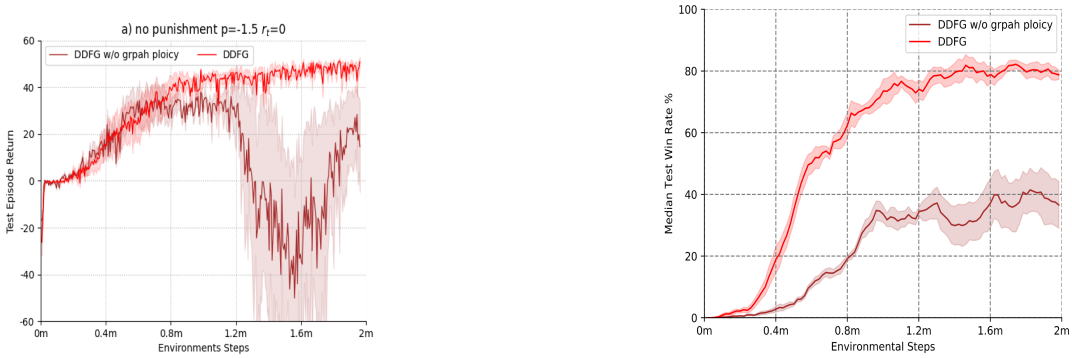

Fig. 9. The ablation experiment of graph generation policy in HO-Predator-Prey and SMAC environments.

a negative feedback loop between the learning mechanisms of the random graph and the value function. This phenomenon highlights the detrimental effects of a random graph policy on the value function network within challenging environments, thereby emphasizing the criticality of accurately learning collaborative relationships.

For other ablation experiments, please refer to the Appendix Section D, available online.

VI. Conclusion & Future Work

This paper has introduced the Dynamic Deep Factor Graph (DDFG) algorithm, leveraging factor graph decomposition for global value functions, significantly enhancing the algorithm's capability to counteract relative overgeneralization. Employing tensor-based CP decomposition, DDFG adeptly navigates large action spaces with heightened efficiency. Furthermore, through the development of a graph structure generation policy utilizing a hypernetwork, DDFG dynamically generates adjacency matrices, capturing the fluid collaborative dynamics among agents. Employing the max-sum algorithm, the optimal policy for agents is ascertained. This paper theoretically prove the error upper bound of the global value function under high-order decomposition. This paper discovers the relationship between the maximum decomposition order D and the error upper bound highlights the trade-off between model accuracy and computational cost. This provides a theoretical foundation for balancing the computational overhead of approximating the global value function and the performance degradation from decomposition. DDFG's effectiveness is empirically validated in complex higher-order predator-prey tasks and numerous challenging scenarios within the SMAC II framework. Compared to DCG, DDFG explicitly learns dynamic agent collaboration and offering a more adaptable decomposition policy. Future endeavors will explore the elimination of the highest order limitation within DDFG to enhance the decomposition method's flexibility while preserving algorithmic learning efficiency.

References

[1] Z. Cao, K. Jiang, W. Zhou, S. Xu, H. Peng, and D. Yang, "Continuous improvement of self-driving cars using dynamic confidence-aware reinforcement learning," *Nature Mach. Intell.*, vol. 5, no. 2, pp. 145–158, 2023.

[2] H. Wang and J. Wang, "Enhancing multi-UAV air combat decision making via hierarchical reinforcement learning," *Sci. Rep.*, vol. 14, no. 1, 2024, Art. no. 4458.

[3] A. Oroojlooy and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," *Appl. Intell.*, vol. 53, pp. 13677–13722, 2023.

[4] R. Lowe et al., "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6382–6393.

[5] C. Yu et al., "The surprising effectiveness of PPO in cooperative multi-agent games," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 24611–24624.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:: 1707.06347*.

[7] Y. Wang, B. Han, T. Wang, H. Dong, and C. Zhang, "Off-policy multi-agent decomposed policy gradients," 2020, *arXiv:: 2007.12322*.

[8] P. Sunehag et al., "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *Proc. 17th Int. Conf. Auton. Agents MultiAgent Syst.*, 2018, pp. 2085–2087.

[9] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Monotonic value function factorisation for deep multi-agent reinforcement learning," *J. Mach. Learn. Res.*, vol. 21, no. 1, pp. 7234–7284, 2020.

[10] L. Panait, S. Luke, and R. P. Wiegand, "Biasing coevolutionary search for optimal multiagent behaviors," *IEEE Trans. Evol. Comput.*, vol. 10, no. 6, pp. 629–645, Dec. 2006.

[11] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi, "QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2019, pp. 5887–5896.

[12] J. Wang, Z. Ren, T. Liu, Y. Yu, and C. Zhang, "QPLEX: Duplex dueling multi-agent Q-learning," in *Proc. Int. Conf. Learn. Representations*, 2021, pp 11609–11635.

[13] T. Rashid, G. Farquhar, B. Peng, and S. Whiteson, "Weighted QMIX: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 10199–10210.

[14] C. Guestrin, M. Lagoudakis, and R. Parr, "Coordinated reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2002, pp. 227–234.

[15] W. Böhmer, V. Kurin, and S. Whiteson, "Deep coordination graphs," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2020, pp. 980–991.

[16] J. Sheng et al., "Learning structured communication for multi-agent reinforcement learning," *Auton. Agents Multi-Agent Syst.*, vol. 36, no. 2, 2022, Art. no. 50.

[17] Y. Kang, T. Wang, Q. Yang, X. Wu, and C. Zhang, "Non-linear coordination graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 25655–25666.

[18] Q. Yang, W. Dong, Z. Ren, J. Wang, T. Wang, and C. Zhang, "Self-organized polynomial-time coordination graphs," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2022, pp. 24963–24979.

[19] T. Wang, L. Zeng, W. Dong, Q. Yang, Y. Yu, and C. Zhang, "Context-aware sparse deep coordination graphs," in *Proc. Int. Conf. Learn. Representations*, 2022, pp. 8126–8148.

[20] S. Li, J. K. Gupta, P. Morales, R. Allen, and M. J. Kochenderfer, "Deep implicit coordination graphs for multi-agent reinforcement learning," in *Proc. 20th Int. Conf. Auton. Agents MultiAgent Syst.*, 2021, pp. 764–772.

[21] J. Jiang, C. Dun, T. Huang, and Z. Lu, "Graph convolutional reinforcement learning," 2018, *arXiv:: 1810.09202*.
[22] H.-A. Loeliger, "An introduction to factor graphs," *IEEE Signal Process. Mag.*, vol. 21, no. 1, pp. 28–41, Jan. 2004.
[23] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.
[24] A.-H. Phan, P. Tichavsk, K. Sobolev, K. Sozykin, D. Ermilov, and A. Cichocki, "Canonical polyadic tensor decomposition with low-rank factor matrices," in *Proc. 2021 IEEE Int. Conf. Acoust. Speech Signal Process.*, 2021, pp. 4690–4694.
[25] M. Zhou et al., "Factorized Q-learning for large-scale multi-agent systems," in *Proc. 1st Int. Conf. Distrib. Artif. Intell.*, 2019, pp. 1–7.
[26] F. A. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs*. Berlin, Germany: Springer, 2016.
[27] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[28] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.
[29] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," in *Proc. AAAI Fall Symp. Ser.*, 2015, pp. 29–37.
[30] Z. Zhang and D. Zhao, "Clique-based cooperative multiagent reinforcement learning using factor graphs," *IEEE/CAA J. Automatica Sinica*, vol. 1, no. 3, pp. 248–256, Jul. 2014.
[31] A. Dubey, F. Radenovic, and D. Mahajan, "Scalable interpretability via polynomials," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 36748–36761.
[32] T. J. Hastie and R. J. Tibshirani, "Generalized additive models," in *Statistical Models in S*, J. M. Chambers and T. J. Hastie, Eds., New York, NY, USA: Routledge, 2017, pp. 249–307.
[33] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2017, pp. 22–31.
[34] J. Queeney, Y. Paschalidis, and C. G. Cassandras, "Generalized proximal policy optimization with sample reuse," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 11909–11919.
[35] J. R. Kok and N. Vlassis, "Using the max-plus algorithm for multiagent decision making in coordination graphs," in *RoboCup 2005: Robot Soccer World Cup IX 9*, Berlin, Germany: Springer, 2006, pp. 1–12.
[36] X. Lan, S. Roth, D. Huttenlocher, and M. J. Black, "Efficient belief propagation with learned higher-order Markov random fields," in *Proc. 9th Eur. Conf. Comput. Vis.*, Graz, Austria, Springer, 2006, pp. 269–282.
[37] P. Som and A. Chockalingam, "Damped belief propagation based near-optimal equalization of severely delay-spread UWB MIMO-ISI channels," in *Proc. IEEE Int. Conf. Commun.*, 2010, pp. 1–5.
[38] J. Foerster et al., "Stabilising experience replay for deep multi-agent reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2017, pp. 1146–1155.
[39] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015, *arXiv::1506.02438*.
[40] M. Samvelyan et al., "The starcraft multi-agent challenge," in *Proc. 18th Int. Conf. Auton. Agents MultiAgent Syst.*, 2019, pp. 2186–2188.

Yuchen Shi received the BE and MS degrees from the University of Science and Technology Beijing (USTB), China, in 2021 and 2024, respectively. He is currently working towards the doctoral degree with USTB. His research interests include multi-agent reinforcement learning, patter recognition and Internet of Things.

Shihong Duan received the PhD degree in computer science from the University of Science and Technology Beijing (USTB). She is an associate professor with the School of Computer and Communication Engineering, USTB. Her research interests include wireless indoor positioning, multi-robots network and Internet of Things.

Cheng Xu (Member, IEEE) received the BE, MS, and PhD degrees from the University of Science and Technology Beijing (USTB), China, in 2012, 2015, and 2019, respectively. He was a visiting scholar with Université libre de Bruxelles (ULB) and Nayang Technological University (NTU), in 2023–2025. He is currently working as an associate professor with the Data and Cyber-Physical System Lab (DCPS), University of Science and Technology Beijing. He is supported by the Post-doctoral Innovative Talent Support Program from Chinese government, in 2019. He is an associate editor of *International Journal of Wireless Information Networks*. His research interests now include swarm intelligence, multi-robots network, wireless localization and Internet of Things.

Ran Wang (Graduate Student Member, IEEE) received the BE degree from the Beijing Information Science and Technology University, China, in 2013, and the MS and PhD degrees from the University of Science and Technology Beijing (USTB), China, in 2016 and 2025, respectively. She is currently working working as a lecture with Beijing Union University. Her research interests include multi-robots network, quantum optimization, distributed security and Internet of Things.

Fangwen Ye received the BE and MS degrees from the University of Science and Technology Beijing (USTB), China, in 2022 and 2025, respectively. He is currently working toward the doctoral degree with the University of Science and Technology Beijing. His research interests include swarm intelligence, multi-robots network, and blockchain system.

Chau Yuen (Fellow, IEEE) He was a post-doctoral fellow with Lucent Technologies Bell Labs, Murray Hill, New York, in 2005, and a visiting assistant professor with The Hong Kong Polytechnic University, in 2008. From 2006 to 2010, he was with the Institute for Infocomm Research, Singapore. From 2010 to 2023, he was with the Engineering Product Development Pillar, Singapore University of Technology and Design. Since 2023, he has been with the School of Electrical and Electronic Engineering, Nanyang Technological University. He has three U.S. patents and published more than 500 research papers at international journals or conferences. He was a recipient of the Lee Kuan Yew Gold Medal, the Institution of Electrical Engineers Book Prize, the Institute of Engineering of Singapore Gold Medal, the Merck Sharp and Dohme Gold Medal, and twice a recipient of the Hewlett Packard Prize. He received the IEEE ICC Best Paper Award, in 2023, the IEEE Communications Society Fred W. Ellersick Prize, in 2023, the IEEE Marconi Prize Paper Award in Wireless Communications, in 2021, and EURASIP Best Paper Award for Journal on Wireless Communications and Networking, in 2021. He received the IEEE Asia Pacific Outstanding Young Researcher Award, in 2012 and the IEEE VTS Singapore Chapter Outstanding Service Award, in 2019. He is a distinguished lecturer of the IEEE Vehicular Technology Society, a Top 2% Scientists by Stanford University, and also a Highly Cited Researcher by Clarivate Web of Science.

Dynamic Deep Factor Graph for Multi-Agent Reinforcement Learning — Supplementary Material —

Yuchen Shi, Shihong Duan, Cheng Xu, *Member, IEEE,* Ran Wang, Fangwen Ye, Chau Yuen, *Fellow, IEEE*

In this supplementary material, we give additional related work discussion, detailed derivations and proof of the equations and propositions mentioned in the main text. Supplementary experiment settings and the code reproduction steps are also presented.

APPENDIX A
RELATED WORK

A.1 VDN

The main assumption made and utilized by the VDN is that the joint action-value function of the system can be decomposed into a sum of the value functions of the single agent:

$$Q((h^1, h^2, \ldots, h^N), (u^1, u^2, \ldots, u^N)) \approx \sum_{i=1}^{N} \tilde{Q}_i(h^i, u^i) \tag{1}$$

where $\tilde{Q}_i$ depends only on the local observation of each agent. We learn $\tilde{Q}_i$ by back-propagating the rule in Q-learning, which learns the joint reward by summation. The $\tilde{Q}_i$ is learned implicitly, not from any reward specific to agent i. The algorithm does not constrain $\tilde{Q}_i$ to be an action-value function for any particular reward. The agents in the VDN algorithm can be deployed independently because the greedy policy of each agent $u^i = \arg\max_{u^i} \tilde{Q}_i(h^i, u^i)$ for the local value function $\tilde{Q}_i$ is equivalent to choosing the joint action to maximize $\sum_{i=1}^{N} \tilde{Q}_i$.

A.2 QMIX

QMIX implements two improvements over VDN: 1) the inclusion of global information to assist in the training process and 2) the use of a hybrid network to merge local value functions of single agents.

QMIX, as a representative value decomposition method, follows the centralized training, distributed execution (CTDE) paradigm [1]. The core part of QMIX is the hybrid network, which is responsible for credit assignments. In QMIX, each agent a has a separate Q-network $Q_i(\tau^i, u^i)$. The output of the single Q-network is passed through the hybrid network, which implicitly assigns credits to each agent and generates an approximation of the global Q-value $Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}, s; \theta)$. The Q_{tot} is computed as follows:

$$Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}, s; \theta) = W_2 ELU\left(W_1 \boldsymbol{Q} + b_1\right) + b_2 \tag{2}$$

where $\boldsymbol{Q} = [Q_1(\tau_1, u_1), Q_2(\tau_2, u_2), \ldots, Q_n(\tau_n, u_n)]^T \in \mathbb{R}^{n \times 1}$ is the output of the independent Q-network. $W_1 \in \mathbb{R}_+^{m \times n}, W_2 \in \mathbb{R}_+^{1 \times m}, b_1 \in \mathbb{R}^{m \times 1}, b_2 \in \mathbb{R}$ is the weight generated by the HyperNetworks. Since the elements of W_1 and W_2 are non-negative, QMIX satisfies the following condition:

$$\frac{\partial Q_{tot}}{\partial Q_i} \geq 0, \forall i \in \{1, 2, \cdots, n\} \tag{3}$$

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62101029, in part by Guangdong Basic and Applied Basic Research Foundation under Grant 2023A1515140071, and in part by the China Scholarship Council Award under Grant 202006465043 and 202306460078. (Corresponding author: Cheng Xu)

- *Yuchen Shi, Shihong Duan, Cheng Xu, Ran Wang and Fangwen Ye are with the School of Computer and Communication Engineering and the Shunde Innovation School, University of Science and Technology Beijing, Beijing 100083, China (email: shiyuchen199@sina.com; duansh@ustb.edu.cn; xucheng@ustb.edu.cn; wangran423@foxmail.com; yfwen2000@outlook.com).*
- *Chau Yuen is with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798 (email: chau.yuen@ntu.edu.sg).*

This property guarantees the individual-global maximum (IGM) principle [2], i.e., that the optimal action of each agent is jointly constituted as an optimal joint action of Q_{tot}:

$$\arg\max_{\boldsymbol{u}} Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}, s) = \begin{pmatrix} \arg\max_{u^1} Q_1(\tau^1, u^1) \\ \vdots \\ \arg\max_{u^n} Q_n(\tau^n, u^n) \end{pmatrix} \tag{4}$$

In this way, the agent can choose the best local action based only on its local observation-action history, while the joint action is the best action for the whole system.

The system is updated by minimizing the square of the TD loss on Q_{tot}, according to the following equation:

$$\mathcal{L}(\theta) = \left(y^{dqn} - Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}, s; \theta)\right)^2 \tag{5}$$

where $y^{dqn} = r + \gamma \max_{\boldsymbol{u}'} Q_{tot}(\boldsymbol{\tau}', \boldsymbol{u}', s'; \theta^-)$ is the TD objective.

A.3 WQMIX

Although QMIX improves the form of value function decomposition of VDN, the monotonicity constraint of QMIX makes its function expressiveness somewhat limited.WQMIX improves the value function expressiveness of QMIX by defining the QMIX operator and weighting the QMIX operator according to the importance of each action in the joint action space, which in turn maps it to the "non-monotonic" functions.

First, WQMIX defines the function space where the global value function Q_{tot} resides:

$$Q^{mix} = \{Q_{tot} \mid Q_{tot}(s, \boldsymbol{u}) = f_s(Q_1(s, u^1), \ldots, Q_n(s, u^n)), \frac{\partial f_s}{\partial Q_i} \geq 0, Q_i(s, u^i) \in \mathbb{R}\} \tag{6}$$

Constraining Q^{mix} in Q_{tot} can be seen as solving the following optimization problem:

$$\arg\min_{q \in Q^{mix}} \sum\nolimits_{\boldsymbol{u} \in U} \left(\mathcal{T}^* Q_{tot}(s, \boldsymbol{u}) - q(s, \boldsymbol{u})\right)^2, \forall s \in S \tag{7}$$

where $\mathcal{T}^*$ is the Bellman optimality operator.

Meanwhile, define the corresponding projection operator Π_{Qmix}:

$$\Pi_{Qmix} Q := \arg\min_{q \in Q^{mix}} \sum\nolimits_{\boldsymbol{u} \in U} \left(Q(s, \boldsymbol{u}) - q(s, \boldsymbol{u})\right)^2 \tag{8}$$

Define the QMIX operator as a composite of the Bellman optimality operator and the projection operator: $\mathcal{T}^*_{Qmix} = \Pi_{Qmix} \mathcal{T}^*$.

The QMIX operator $\mathcal{T}^*_{Qmix}$ itself has many problems, such as $\mathcal{T}^*_{Qmix}$ is not a compression mapping, and the joint actions obtained by maximizing Q_{tot} in QMIX are erroneous in some scenarios (underestimating the value of some joint actions).Based on these drawbacks, WQMIX proposes the Weighted QMIX operator.

First, define the new projection operator Π_w:

$$\Pi_w Q := \arg\min_{q \in Q^{mix}} \sum\nolimits_{\boldsymbol{u} \in U} w(s, \boldsymbol{u}) \left(Q(s, \boldsymbol{u}) - q(s, \boldsymbol{u})\right)^2 \tag{9}$$

where the weight function $w : S \times U \to (0, 1]$, when $w(s, \boldsymbol{u}) \equiv 1$, $\Pi_w = \Pi_{Qmix}$.

And the weight calculation needs to use Q^*, so it is necessary to additionally learn $\hat{Q}^*$ to fit Q^*, and Q^* is updated by the following operator:

$$\mathcal{T}^*_w \hat{Q}^*(s, u) := E[r + \gamma \hat{Q}^*(s', \arg\max_{u'} Q_{tot}(s', u'))] \tag{10}$$

Then the weighted QMIX operator is defined as: $\mathcal{T}^*_{WQMIX} \hat{Q}^* := \Pi_w \mathcal{T}^*_w \hat{Q}^*$.

Eventually WQMIX trains both Q_{tot} and $\hat{Q}^*$, updating the network by minimizing the square of the TD loss of the two Q functions:

$$\mathcal{L}(\theta) = w(s, \boldsymbol{u}) \left(y^{dqn} - Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}, s; \theta_1)\right)^2 + \left(y^{dqn} - \hat{Q}^*(\boldsymbol{\tau}, \boldsymbol{u}, s; \theta_2)\right)^2 \tag{11}$$

where $y^{dqn} = r + \gamma \hat{Q}^*(\boldsymbol{\tau}', \arg\max_{\boldsymbol{u}'} Q_{tot}(\boldsymbol{\tau}', \boldsymbol{u}', s'; \theta_1), s'; \theta_2)$ is the TD target.

And WQMIX obtains two WQMIX algorithms, Centrally-Weighted QMIX(CW-QMIX) and Optimistically-Weighted QMIX(OW-QMIX), by designing different weight functions $w(s, \boldsymbol{u})$.

Weighting functions in CW-QMIX:

$$w(s, u) = \begin{cases} 1 & y^{dqn} > \hat{Q}^*(\boldsymbol{\tau}, \hat{\boldsymbol{u}}^*, s) \; or \; \boldsymbol{u} = \hat{\boldsymbol{u}}^* \\ \alpha & otherwise \end{cases} \tag{12}$$

where $\hat{\boldsymbol{u}}^* = \arg\max_{\boldsymbol{u}} Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}, s)$.

Weighting functions in OW-QMIX:

$$w(s, \boldsymbol{u}) = \begin{cases} 1 & Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}, s) < y^{dqn} \\ \alpha & otherwise \end{cases} \tag{13}$$

A.4 QTRAN

QTRAN improves the decomposition form of the value function in VDN, QMIX. QTRAN proposes the IGM condition and proves that the linear decomposition in VDN and the monotonic decomposition in QMIX are sufficient but not necessary for the IGM. This leads to the decomposition of value functions in VDN and QMIX in a way that is too constrained, making it impossible to solve non-monotonicity tasks. QTRAN proposes to satisfy the sufficient and necessary conditions for IGM and proves its correctness.

Theorem proposed by QTRAN: A global value function $Q_{tot}\left(\boldsymbol{\tau}, \boldsymbol{u}\right)$ can be decomposed into $\left[Q_i\left(\tau^i, u^i\right)\right]_{i=1}^N$, if it satisfies:

$$\sum_{i=1}^{N} Q_i\left(\tau^i, u^i\right) - Q_{tot}\left(\boldsymbol{\tau}, \boldsymbol{u}\right) + V_{tot}\left(\boldsymbol{\tau}\right) = \begin{cases} 0 & \boldsymbol{u} = \bar{\boldsymbol{u}} \\ \geq 0 & \boldsymbol{u} \neq \bar{\boldsymbol{u}} \end{cases} \tag{14}$$

where $V_{tot}\left(\boldsymbol{\tau}\right) = \max_{\boldsymbol{u}} Q_{tot}\left(\boldsymbol{\tau}, \boldsymbol{u}\right) - \sum_{i=1}^{N} Q_i\left(\tau^i, \bar{u}^i\right)$.

QTRAN learns Q_i, Q_{tot}, V_{tot} through the network separately and designs the loss function based on Q_{tot}, V_{tot}:

$$\mathcal{L}\left(\boldsymbol{\tau}, \boldsymbol{u}, r, \boldsymbol{\tau}'; \theta\right) = \mathcal{L}_{td} + \lambda_{opt}\mathcal{L}_{opt} + \lambda_{nopt}\mathcal{L}_{nopt} \tag{15}$$

$\mathcal{L}_{td}$ is used to fit Q_{tot}, $\mathcal{L}_{opt}$ and $\mathcal{L}_{nopt}$ is used to fit V_{tot}. λ_{opt} and λ_{nopt} are the weight constants for two losses. $\mathcal{L}_{td}, \mathcal{L}_{opt}, \mathcal{L}_{nopt}$ are defined as follows:

$$\begin{aligned} \mathcal{L}_{td}\left(;\theta\right) &= \left(y^{dqn} - Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u})\right)^2, y^{dqn} = r + \gamma Q_{tot}(\boldsymbol{\tau}', \bar{\boldsymbol{u}}'; \theta^{-}) \\ \mathcal{L}_{opt}\left(;\theta\right) &= \left({Q'}_{tot}(\boldsymbol{\tau}, \bar{\boldsymbol{u}}) - \hat{Q}_{tot}(\boldsymbol{\tau}, \bar{\boldsymbol{u}}) + V_{tot}\left(\boldsymbol{\tau}\right)\right)^2 \\ \mathcal{L}_{nopt}\left(;\theta\right) &= \left(\min\left[{Q'}_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) - \hat{Q}_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) + V_{tot}\left(\boldsymbol{\tau}\right), 0\right]\right)^2 \end{aligned} \tag{16}$$

where θ^{-} is the target network parameter. $\mathcal{L}_{opt}$ and $\mathcal{L}_{nopt}$ are used to ensure that the above theorem holds.

A.5 QPLEX

QPLEX proposes an improvement to the IGM. By drawing on the f dueling decomposition structure $Q = V + A$ proposed by Dueling DQN [3], QPLEX formalizes the IGM as an advantage-based IGM.

Advantage-based IGM: For the joint action-value function $Q_{tot} : \mathcal{T} \times U \rightarrow \mathbb{R}$ and the individual action-value function $[Q_i : \mathcal{T} \times U \rightarrow \mathbb{R}]_{i=1}^n$, where $\forall \tau \in \mathcal{T}, \forall u \in U, \forall i \in N$,

Joint Dueling $\quad Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) = V_{tot}(\boldsymbol{\tau}) + \mathcal{A}_{tot}(\boldsymbol{\tau}, \boldsymbol{u})\ and\ V_{tot}(\boldsymbol{\tau}) = \max_{u'} Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}')$

Individual Dueling $\quad Q_i(\tau^i, u^i) = V_i(\tau^i) + \mathcal{A}_i(\tau^i, u^i)\ and\ V_{tot}(\tau^i) = \max_{u^{i\prime}} Q_i(\tau^i, u^{i\prime})$

such that the following holds:

$$\arg\max_{\boldsymbol{u} \in U} \mathcal{A}_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) \ = \left(\arg\max_{u^1 \in U} \mathcal{A}_i(\tau^1, u^1), \ldots, \arg\max_{u^n \in U} \mathcal{A}_n(\tau^n, u^n)\right) \tag{17}$$

Then, $[Q_i]_{i=1}^n$ satisfies the Advantage-based IGM of Q_{tot} (which is equivalent to the IGM).

The structure of QPLEX consists of two parts: (i) an Individual Action-Value Function for each agent, and (ii) a Duplex Dueling component.

An Individual Action-Value Function consists of an RNN Q-network for each agent, where the network inputs are the hidden state h_{t-1}^i and action u_{t-1}^i of the previous moment, and the local observation o_t^i of the current moment. The output is a localized value function $Q_i(\tau^i, u^i)$.

The Duplex Dueling component connects local and joint action-valued functions through two modules: (i) a Transformation network module (ii) a Dueling Mixing network module.

The Transformation network module uses centralized information to transform the local duel structure $[V_i(\tau^i), \mathcal{A}_i(\tau^i, u^i)]_{i=1}^n$ to $[V_i(\boldsymbol{\tau}), \mathcal{A}_i(\boldsymbol{\tau}, u^i)]_{i=1}^n$ for any agent i, i.e., $Q_i(\boldsymbol{\tau}, u^i) = w_i(\boldsymbol{\tau})Q_i(\tau^i, u^i) + b_i(\boldsymbol{\tau})$, thus:

$$V_i(\boldsymbol{\tau}) = w_i(\boldsymbol{\tau})V_i(\tau^i) + b_i(\boldsymbol{\tau}) \quad and \quad \mathcal{A}_i(\boldsymbol{\tau}, u^i) = w_i(\boldsymbol{\tau})\mathcal{A}_i(\tau^i, u^i) \tag{18}$$

where $w_i(\boldsymbol{\tau}) > 0$, this positive linear transformation maintains the consistency of greedy action selection.

The Dueling Mixing network module takes as input the output of the Transformation network module $[V_i, \mathcal{A}_i]_{i=1}^n$ and outputs the global value function Q_{tot}.

Advantage-based IGM conditions do not impose constraints on V_{tot}, so QPLEX uses a simple sum structure for V_{tot}: $V_{tot}(\tau) = \sum_{i=1}^{n} V_i(\tau)$.

To satisfy the Advantage-based IGM condition, QPLEX calculates the joint advantage function as follows:

$$\mathcal{A}_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) = \sum_{i=1}^{n} \lambda_i(\boldsymbol{\tau}, \boldsymbol{u})\mathcal{A}_i(\boldsymbol{\tau}, u^i),\ where\ \lambda_i(\boldsymbol{\tau}, \boldsymbol{u}) > 0 \tag{19}$$

The positive linear transformation guided by λ_i will continue to maintain consistency in greedy action selection. To be able to efficiently learn λ_i with joint histories and actions, QPLEX uses an extensible multi-head attention module:

$$\lambda_i(\boldsymbol{\tau}, \boldsymbol{u}) = \sum_{k=1}^{K} \lambda_{i,k}(\boldsymbol{\tau}, \boldsymbol{u}) \phi_{i,k}(\boldsymbol{\tau}) v_k(\boldsymbol{\tau}) \tag{20}$$

where K is the number of attention heads, $\lambda_{i,k}(\boldsymbol{\tau}, \boldsymbol{u})$ and $\phi_{i,k}(\boldsymbol{\tau})$ are the attention weights activated by the sigmoid function, and $v_k(\boldsymbol{\tau}) > 0$ is the positive key for each head.

Eventually, the joint action-value function Q_{tot} is expressed as follows:

$$Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) = V_{tot}(\boldsymbol{\tau}) + \mathcal{A}_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) = \sum_{i=1}^{n} Q_i(\boldsymbol{\tau}, u^i) + \sum_{i=1}^{n} (\lambda_i(\boldsymbol{\tau}, \boldsymbol{u}) - 1) \mathcal{A}_i(\boldsymbol{\tau}, u^i) \tag{21}$$

A.6 DCG

DCG is a MARL algorithm that combines coordination graphs and deep neural networks. In DCG, a coordination graph CG decomposes the joint value function into the sum of the utility function f^i and the cost function f^{ij} as follows:

$$Q^{CG}(s_t, \boldsymbol{u}) = Q^{CG}(\boldsymbol{\tau}_t, \boldsymbol{u}) = \frac{1}{|V|} \sum_{v^i \in V} f^i(u^i \mid \boldsymbol{h}_t) + \frac{1}{|\varepsilon|} \sum_{i,j \in \varepsilon} f^{ij}(u^i, u^j \mid \boldsymbol{h}_t) \tag{22}$$

DCG uses the max-sum algorithm of the coordination graph to deliver messages. At each time step t, every node i sends a message $\mu_t^{ij}(u^j) \in \mathbb{R}$ to all neighboring nodes $i, j \in \varepsilon$. The message can be decentralized and computed using the following equation:

$$\mu_t^{ij}(u^j) \leftarrow \max_{a^i} \left\{ \frac{1}{|V|} f^i(u^i \mid \boldsymbol{h}_t) + \frac{1}{|\varepsilon|} f^{ij}(u^i, u^j \mid \boldsymbol{h}_t) + \sum_{k,i \in \varepsilon} \mu_t^{ki}(u^i) - \mu_t^{ji}(u^i) \right\} \tag{23}$$

This message computation and passing must be repeated several times until convergence so that each agent i can find the optimal action u^i to obtain the maximized estimated joint action value $Q^{CG}(\boldsymbol{\tau}_t, \boldsymbol{u}_*)$. The optimal action u_*^i can be computed using the following equation:

$$u^i := \arg\max_{u^i} \left\{ \frac{1}{|V|} f^i(u^i \mid \boldsymbol{h}_t) + \sum_{k,i \in \varepsilon} \mu_t^{ki}(u^i) \right\} \tag{24}$$

At each time step t, DCG approximates the utility function f^i and the cost function f^{ij} using a network parameterized by θ, ϕ, and ψ. The joint action value is denoted as follows:

$$Q_{\theta\phi\psi}^{DCG}(\boldsymbol{\tau}_t, \boldsymbol{u}) := \frac{1}{|V|} f_\theta^v(u^i \mid h_t^i) + \frac{1}{2|\varepsilon|} \sum_{i,j \in \varepsilon} (f_\phi^e(u^i, u^j \mid h_t^i, h_t^j) + f_\phi^e(u^j, u^i \mid h_t^j, h_t^i)) \tag{25}$$

Finally, the DCG updates the parameters of the DCG network end-to-end according to the DQN-style loss function:

$$\mathcal{L}_{DQN} := E\left[\frac{1}{T} \sum_{t=0}^{T-1} \left(Q_{\theta\phi\psi}^{DCG}(\boldsymbol{u}_t \mid \boldsymbol{\tau}_t) - \left(r_t + \gamma \max Q_{\bar{\theta}\bar{\phi}\bar{\psi}}^{DCG}(\cdot \mid \boldsymbol{\tau}_{t+1}) \right) \right)^2 \right] \tag{26}$$

where $\bar{\theta}, \bar{\phi}, \bar{\psi}$ is the parameter copied periodically from θ, ϕ, ψ.

A.7 SOPCG

SOPCG uses a class of polynomial-time coordination graphs to construct a dynamic and state-dependent topology. SOPCG first models coordination relations upon the graph-based value factorization specified by DCG, where the joint value function of the multi-agent system is factorized into the summation of individual utility functions f_i and pairwise payoff functions f_{ij}.

$$Q(\tau_t, u; G) = \sum_{i \in [n]} f^i(u^i \mid \tau_t^i) + \sum_{(i,j) \in G} f^{ij}(u^i, u^j \mid \tau_t^i, \tau_t^j) \tag{27}$$

where the coordination graph G is represented by a set of undirected edges. With this second-order value decomposition, the hardness of greedy action selection is highly related to the graph topology.

Polynomial-Time Coordination Graph Class: A graph class G is a Polynomial-Time Coordination Graph Class if there exists an algorithm that can solve any induced DCOP of any coordination graph $G \in \mathcal{G}$ in $Poly(n, A)$ running time.

The set of undirected acyclic graphs, denoted as $Guac$, forms a polynomial-time tractable graph class. However, within an environment containing n agents, an undirected acyclic graph can have at most $n-1$ edges, limiting the expressive capacity of functions. To mitigate this issue, SPOCG permits the dynamic evolution of the graph's topology through transitions in the environmental state. Across different environmental states, the joint value can be decomposed by selecting coordinating graphs from a predefined graph

class $G \subseteq Guac$. This graph class choice augments the limited representational capacity while preserving the accuracy of greedy action selection.

SOPCG introduce an imaginary coordinator agent whose action space refers to the selection of graph topologies, aiming to select a proper graph for minimizing the suboptimality of performance within restricted coordination. The graph topology G can be regarded as an input of joint value function $Q(\tau_t, u; G)$. The objective of the coordinator agent is to maximize the joint value function over the specific graph class. SOPCG handle the imaginary coordinator as a usual agent in the multi-agent Q-learning framework and rewrite the joint action as $\boldsymbol{u}^{cg} = (u^1, \cdots, u^n, G)$

Formally, at time step t, greedy action selection indicates the following joint action:

$$\boldsymbol{u}_t^{cg} \leftarrow \underset{(u^1,\cdots,u^n,G)}{\arg\max}\ Q(\boldsymbol{\tau}_t, u^1, \cdots u^n; G) \tag{28}$$

Hence the action of coordinator agent G_t is:

$$G_t \leftarrow \underset{G\in\mathcal{G}}{\arg\max}\ \left(\max_u Q(\boldsymbol{\tau}_t, \boldsymbol{u}; G)\right) \tag{29}$$

After determining the graph topology G_t, the agents can choose their individual actions to jointly maximize the value function $Q(\tau_t, u; G)$ upon the selected topology.

With the imaginary coordinator, SOPCG can reformulate the Bellman optimality equation and maximize the future value over the coordinator agent's action:

$$Q^*(\tau, u; G) = \underset{\tau'}{E}\left[r + \max_{G'}\max_{u'} Q^*(\tau', u'; G')\right] \tag{30}$$

Since the choice of the graph is a part of the agents' actions, $Q(\tau, u; G)$ can be updated using temporal difference learning:

$$\mathcal{L}_{cg}(\theta) = \underset{(\tau,u,G,r,\tau')\sim D}{E}\left[\left(y_{cg} - Q(\tau, u; G; \theta)\right)^2\right] \tag{31}$$

A.8 CASEC

CASEC utilizes the variance of paired payoff functions as the criterion for selecting edges. A sparse graph is employed when choosing greedy joint actions for execution and updating the Q-function. Simultaneously, to mitigate the impact of estimation errors on sparse topological learning, CASEC further incorporates a network structure based on action representation for utility and payoff learning.

If the actions of agent j significantly influence the expected utility for agent i, then agent i needs to coordinate its action selection with agent j. For a fixed action u^i, $Var_{u^j}[q_{ij}(u^i, u^j \mid \tau_t^i, \tau_t^j)]$ can quantify the impact of agent j on the expected payoff. This motivates the use of variance of payoff functions as the criterion for edge selection in CASEC:

$$\zeta_{ij}^{q_{\text{var}}} = \max_{u^i} Var_{u^j}[q_{ij}(u^i, u^j \mid \tau_t^i, \tau_t^j)] \tag{32}$$

The maximization operator guarantees that the most affected action is considered. When $\zeta_{ij}^{q_{\text{var}}}$ is large, the expected utility of agent i fluctuates dramatically with the action of agent j, and they need to coordinate their actions. Therefore, to construct sparse coordination graphs, we can set a sparseness controlling constant $\lambda \in (0, 1)$ and select $\lambda|\mathcal{V}|(|\mathcal{V}| - 1)$ edges with the largest $\zeta_{ij}^{q_{\text{var}}}$ values.

CASEC consists of two main components–learning value functions and selecting greedy actions.

In CASEC, agents learn a shared utility function $q_\theta(\cdot \mid \tau_t^i)$, parameterized by θ, and a shared pairwise payoff function $q_\varphi(\cdot \mid \tau_t^i, \tau_t^j)$, parameterized by φ. The global Q value function is estimated as:

$$Q_{tot}(\boldsymbol{\tau}_t, \boldsymbol{u}) = \frac{1}{|\mathcal{V}|}\sum_i q_\theta(u^i \mid \tau_t^i) + \frac{1}{|\mathcal{V}|(|\mathcal{V}|-1)}\sum_{i\neq j} q_\varphi(u^i, u^j \mid \tau_t^i, \tau_t^j) \tag{33}$$

which is updated by the TD loss:

$$\mathcal{L}_{TD}(\theta, \varphi) = E_D\left[\left(r + \gamma\widehat{Q}_{tot}(\boldsymbol{\tau}', Max - Sum(q_\theta, q_\varphi)) - Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u})\right)^2\right] \tag{34}$$

$Max - Sum(\cdot, \cdot)$ is the greedy joint action selected by Max-Sum, $\widehat{Q}_{tot}$ is a target network with parameters $\hat{\theta}, \hat{\varphi}$ periodically copied from Q_{tot}. Meanwhile, we also minimize a sparseness loss:

$$\mathcal{L}_{sparse}^{q_{\text{var}}}(\varphi) = \frac{1}{|\mathcal{V}|(|\mathcal{V}|-1)|A|}\sum_{i\neq j}\sum_{u^i} Var_{u^j}[q_{ij}(u^i, u^j \mid \tau_t^i, \tau_t^j)] \tag{35}$$

which is a regularization on $\zeta_{ij}^{q_{\text{var}}}$. Introducing a scaling factor $\lambda_{sparse} \in (0, 1]$ and minimizing $\mathcal{L}_{TD}(\theta, \varphi) + \lambda_{sparse}\mathcal{L}_{sparse}^{q_{\text{var}}}(\varphi)$ builds in inductive biases which favor minimized coordination graphs that would not sacrifice global returns.

In CASEC, the issue of estimation error becomes particularly pronounced as the construction of the coordination graph relies on estimates of q_{ij}. As the construction of the coordination graph also impacts the learning of q_{ij}, a negative feedback loop is generated. This loop leads to instability in learning. To address this problem and stabilize training, CASEC proposes the following strategies: 1) Periodically constructing the graph using a target payoff function to maintain a consistent graph construction approach; 2) Accelerating the training of the payoff function between updates of the target network, by learning action representations to reduce estimation error.

For 2), CASEC proposes a conditional approach on action representations to enhance sample efficiency. We train an action encoder $f_\phi(u)$, where the input is a one-hot encoding of an action u, and the output is its representation z_a. The action representation z_a, along with the current local observation, is used to predict rewards and observations in the next time step. The prediction loss is minimized to update the action encoder $f_\phi(u)$. At the beginning of learning, the action encoder is trained with a small number of samples [4] and remains fixed for the rest of the training process.

Using action representations, the utility and payoff functions can now be estimated as:

$$\begin{aligned} &q_\theta(u^i \mid \tau_t^i) = h_\theta(\tau_t^i)^T z_{u^i}; \\ &q_\varphi(u^i, u^j \mid \tau_t^i, \tau_t^j) = h_\theta(\tau_t^i, \tau_t^j)^T [z_{u^i}, z_{u^j}] \end{aligned} \tag{36}$$

A.9 MAPPO

The MAPPO algorithm extends the PPO algorithm to the multi-agent setting and outperforms PPO and IPPO algorithms in experimental validation. PPO is a widely used on-policy reinforcement learning algorithm, but its usage rate in multi-agent environments is far lower than that of off-policy learning algorithms. This is primarily because the sample efficiency of PPO in multi-agent systems is much lower than that of off-policy methods.

In multi-agent environments, the low adoption rate of the PPO algorithm can be attributed to two factors: firstly, PPO exhibits lower sample efficiency compared to off-policy algorithms; secondly, the hyperparameter tuning methods that work effectively for PPO in single-agent scenarios often fail to deliver robust performance when transferred to multi-agent settings. Whereas the MAPPO algorithm, with minimal modifications (primarily to the critic network), demonstrates final performance comparable to other off-policy algorithms in multi-agent scenarios.

The MAPPO algorithm is primarily divided into two components: policy update and value function estimation. That is to say, there are two networks involved: the policy network π_θ and the value function network V_ϕ. The value function V_ϕ needs to learn a mapping: $S \to \mathbb{R}$. The policy function π_θ learns a mapping from the observation o_t^i to a probability distribution over the action space, or to the mean and variance of a Gaussian function for subsequent action sampling.

The optimization objective of the Actor network is:

$$\nabla_\theta J = \underset{(\tau_t, u_t)\sim\mathcal{T}_{old}}{E} \left[\sum_j \min\left(r_t^i(\varphi)\hat{\mathcal{A}}_i, clip\left(r_t^i(\varphi), 1-\epsilon, 1+\epsilon\right)\hat{\mathcal{A}}_i \right) \nabla \ln \pi_\theta\left(u_t^i|\tau_t^i\right) + \sigma\mathcal{H}\left(\pi_\theta\left(\cdot|\tau_t^i;\theta\right)\right) \right] \tag{37}$$

where $r_t^j(\theta) = \frac{\pi_\theta\left(u_t^i|\tau_t^i\right)}{\pi_{\theta_{old}}\left(u_t^i|\tau_t^i\right)}$. The advantage function $\hat{\mathcal{A}}_i$ is computed using the Generalized Advantage Estimation (GAE) method, where $\mathcal{H}$ denotes the entropy of the policy, and σ is a hyperparameter controlling the entropy coefficient.

The optimization objective of the Critic network is:

$$\mathcal{L}_{critic}(\phi) = -E_{(s_t)\sim\mathcal{D}}\left[\max\left[(V_\phi(s_t) - r)^2, (clip\left(V_\phi(s_t), V_{\phi_{old}}(s_t) - \varepsilon, V_{\phi_{old}}(s_t) + \varepsilon\right) - r)^2\right]\right] \tag{38}$$

A.10 Factor graph and message passing algorithm

Modern computer science and engineering are replete with complex algorithms, and factor graphs serve as a powerful tool to enhance their interpretability. A critical challenge in algorithms handling multivariate global functions lies in their computational inefficiency. This issue is mitigated by decomposing the intricate global function into a product of simpler local functions, each dependent on a subset of the global function's variables. This factorization can be represented by a factor graph: a bipartite graph that explicitly encodes the relationships between variables and their associated local functions.

In optimization problems aimed at "inferring the most probable state of a system," other graphical models, such as Bayesian networks [5] and Markov random fields [6], can be seamlessly transformed into factor graphs [7]. Furthermore, factor graphs can be employed to address numerous problems in artificial intelligence, signal processing, and digital communications. Many algorithms in these domains can be interpreted as special cases of message-passing algorithms [8] operating on factor graphs.

A factor graph $G = \langle X, F, \mathcal{E}\rangle$ [9] is defined by the set of variable nodes X, the set of factor nodes F and the set of undirected edges $\mathcal{E}$. Factor graphs are bipartite graphs represented as factorizations of global functions. Suppose that the global function $g(x_1, \ldots, x_n)$ into a product of several local functions, each having some subset of as arguments; i.e., suppose that

$$g(x_1, \ldots, x_n) = \prod_{j\in J} f_j(X_j) \tag{39}$$

where J is a discrete index set, X_j is a subset of $\{x_1, \ldots, x_n\} \triangleq X$, and $f_j(X_j)$ is a function having the elements of X_j as arguments.

Each variable node $x_i \in X$ in the factor graph corresponds to a variable. Similarly, each factor node $f_j \in F$ corresponds to a local function after the global function decomposition and is connected by an edge $\mathcal{E}$ to the variable node x_i and the factor node f_j when and only when x_i is an argument of f_j. A factor graph with n variable nodes and m function nodes has a binary adjacency matrix defined as $A \in \{0,1\}^{n\times m}$.

Message passing algorithms on factor graphs can efficiently perform exact inference by leveraging the structure of the factor graph. Various message passing algorithms exist on factor graphs, such as the sum-product algorithm, max-sum algorithm, max-product

algorithm, and min-sum algorithm. Max-product algorithm is equivalent to min-sum algorithm [10]; the only (completely superficial) difference is that messages and beliefs are represented as probabilities rather than costs.

In signal processing, Hidden-Markov Models and Kalman filtering have been applied in various applications, and both techniques are considered instances of the sum-product algorithm. Similarly, the Fast Fourier Transform (FFT) algorithm can also be viewed as an instance of the sum-product algorithm. In this paper, we consider value decomposition algorithms in the field of MARL as an instance of the max-sum algorithm.

Next, we will introduce the rules of the sum-product algorithm. The Sum-Product Update Rule:

The message sent from a node x_i on an edge ε is the product of the local function at x_i (or the unit function if x_i is a variable node) with all messages received at on edges other than ε, summarized for the variable associated with ε.

Let $\mu_{x_i \to f_j}(x_i)$ denote the message sent from node x_i to node f_j, let $\mu_{f_j \to x_i}(x_i)$ denote the message sent from node f_j to node x_i. Also, let $n(x)$ denote the set of neighbors of a given node x in a factor graph.

Variable to local function:

$$\mu_{x_i \to f_j}(x_i) = \prod_{h \in n(x_i) \backslash \{f_j\}} \mu_{h \to x_i}(x_i) \tag{40}$$

Local function to variable:

$$\mu_{f_j \to x_i}(x_i) = \sum_{\sim \{x_i\}} f_j(X_j) \prod_{y \in n(f_j) \backslash \{x_i\}} \mu_{y \to f_j}(y) \tag{41}$$

The introduction of the generalized distributive law (GDL) [11] indicates that message passing algorithms such as sum-product, max-product (min-sum), and max-sum can all be unified under a common framework based on commutative semirings. A commutative semiring is a set K, together with two binary operations called "$+$" and "$\cdot$", which satisfy the following three axioms:

- The operation "$+$" is associative and commutative, and there is an additive identity element called "0" such that $k + 0 = k$ for all $k \in K$. (This axiom makes $(K, +)$ a commutative monoid.)
- The operation "$\cdot$" is also associative and commutative, and there is a multiplicative identity element called "1" such that $k \cdot 1 = k$ for all $k \in K$. (Thus $(K, \cdot)$ is also a commutative monoid.)
- The distributive law holds, i.e.,

  $$(a \cdot b) + (a \cdot c) = a \cdot (b + c)$$

  for all triples (a, b, c) from K.

The sum-product algorithm corresponds to the operators of a commutative semiring as follows: "$+$" and "$\cdot$". The max-sum algorithm corresponds to the operators of a commutative semiring as follows: "max" and "$+$". The operators corresponding to the max-product and min-sum algorithms are as follows: "max" and "$\cdot$", "min" and "$+$".

Based on the above explanation, we will now provide the rules for the max-sum algorithm, while the rules for the other message passing algorithms will not be repeated.

Variable to local function:

$$\mu_{x_i \to f_j}(x_i) = \sum_{h \in n(x_i) \backslash \{f_j\}} \mu_{h \to x_i}(x_i) \tag{42}$$

Local function to variable:

$$\mu_{f_j \to x_i}(x_i) = \max_{\sim \{x_i\}} \left(f_j(X_j) + \sum_{y \in n(f_j) \backslash \{x_i\}} \mu_{y \to f_j}(y) \right) \tag{43}$$

APPENDIX B
ADDITIONAL NOTES FROM DDFG

As depicted in Figure 1, when the set of edges $\mathcal{E}$ is empty (i.e., ${A_t}' = I_n$), the algorithm becomes VDN. On the other hand, when every two agents are connected to a factor node (i.e., $\mathcal{E} := \{\{i, j\}, \{i+k, j\} \mid 1 \leq i \leq n, 1 \leq k \leq n-i, j = j+1 (j = 1 \sim n(n-1)/2)\}$), the algorithm degenerates to DCG.

In Section IV-B of the main text, we propose to construct the advantage function $\mathcal{A}_j$ by GAE instead of the value function Q_j. This is precisely because the Q_j function itself has too much variance, which is not conducive to the learning of graph policies. And the advantage function $\mathcal{A}_j$ can better evaluate the goodness of graph policies. The design of the advantage function $\mathcal{A}_j$ is as follows:

$$\hat{\mathcal{A}}_j^{GAE}\left(\boldsymbol{\tau}_t^j, \boldsymbol{u}_t^j, A_t\right) := \sum_{t=0}^{\infty} \left(\gamma \lambda_{GAE}\right)^l \left(Q_j\left(\boldsymbol{\tau}_t^j, \boldsymbol{u}_t^j, A_t\right) - V_j\left(\boldsymbol{\tau}_t^j\right)\right) \tag{44}$$

where λ_{GAE} is the discount factor in GAE.

We construct the network of $V_j\left(\boldsymbol{\tau}_t^j\right)$ by a network structure similar to that of Section III-A. Meanwhile, we construct a network for the global value function V_{tot}. $V_{tot}(s_t; \psi)$ denotes a network parameterized by ψ.

For Network $V_{tot}(s_t; \psi)$,we update the network parameters directly with the TD-error:

$$\mathcal{L}_V\left(s, r; \psi\right) := E\left[\frac{1}{T} \sum_{t=0}^{T} \left(r_t + \gamma V_{tot}(s_{t+1}; \bar{\psi}) - V_{tot}(s_t)\right)^2\right] \tag{45}$$

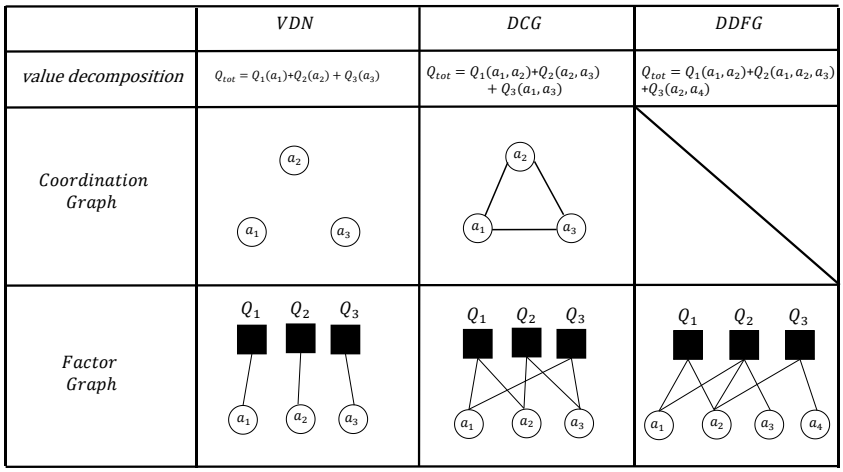

Fig. 1: The graph structure representation of VDN, DCG and DDFG, using both coordination graph and factor graph.

where r_t is the reward for performing action $\boldsymbol{u}_t$ transitions to $\boldsymbol{\tau}_{t+1}$ in the observation history $\boldsymbol{\tau}_t$, $\bar{\psi}$ is the parameter copied periodically from ψ.

For Network $V_j(\boldsymbol{\tau}_t^j)$, we use the adjacency matrix A_t, we represent $V_j(\boldsymbol{\tau}_t)$ as a network parameterized by ϕ_j. The joint value function $\widetilde{V}_{tot}$, represented by the factor graph G:

$$\widetilde{V}_{tot}(\boldsymbol{\tau}_t, A_t) = \sum_{j \in J} V_j(\boldsymbol{\tau}_t; \phi_j) \tag{46}$$

where $\mathcal{J}$ is the set of factor node numbers.

The network of V_j also share h_t^i with Q-value function networks, using h_t^i as an input to the network:

$$\widetilde{V}_{tot}(\boldsymbol{\tau}_t, A_t) = \sum_{j \in J} V_j(\boldsymbol{u}_t^j \mid \boldsymbol{h}_t^j; \phi_{D(j)}) \tag{47}$$

Unlike the Q-value function network, the network of V_j does not need to go through the calculation of the max-sum algorithm. We minimize the distance between the joint value function $\widetilde{V}_{tot}$ obtained by summing over the factor graph G and the global value function $V_{tot}(s_t; \psi)$, using a mean squared error (MSE) loss:

$$\mathcal{L}_V(\boldsymbol{\tau}, A; \phi) := E\left[\frac{1}{T}\sum_{t=0}^{T}\left(\widetilde{V}_{tot}(\boldsymbol{\tau}_t, A_t; \phi) - V_{tot}(s_t)\right)^2\right] \tag{48}$$

where $\boldsymbol{\tau} = \{\boldsymbol{\tau}_t\}_{t=1}^T$.

APPENDIX C
SUPPLEMENTARY PROOF

C.1 Derivation of Equation (8)

Replace K_d with the constant C_k, and substitute it into Theorem 1:

$$\begin{aligned}
&\mathcal{E}(\widehat{Q}_{tot}(D)) - \mathcal{E}(Q_{tot}^{\star}) \\
&\leq 4C_Q B_{\lambda,2} B_{h,2} \sqrt{\frac{C_K}{n_s}} \sum_{d=1}^{D}\left((B_{w,2})^d\right) + \frac{C_1}{C_2}\exp\left(-(C_K+1)^{\gamma}\right)(n-D-1) + \left(2(C_Q)^2 + C_Q\right)\sqrt{\frac{2\log(4/\delta)}{n_s}} \\
&= 4C_Q B_{\lambda,2} B_{h,2} B_{w,2} \sqrt{\frac{C_K}{n_s}}\left(\frac{(B_{w,2})^D - 1}{B_{w,2} - 1}\right) + \frac{C_1}{C_2}\exp\left(-(C_K+1)^{\gamma}\right)(n-1-D) + \left(2(C_Q)^2 + C_Q\right)\sqrt{\frac{2\log(4/\delta)}{n_s}}
\end{aligned}$$

Let $C_{final1} = 4C_Q B_{\lambda,2} B_{h,2} B_{w,2}\sqrt{\frac{C_K}{n_s}}$, $C_{final2} = \frac{C_1}{C_2}\exp\left(-(C_K+1)^{\gamma}\right)$, $C_{final3} = \left(2(C_Q)^2 + C_Q\right)\sqrt{\frac{2\log(4/\delta)}{n_s}} + \frac{C_1}{C_2}\exp\left(-(C_K+1)^{\gamma}\right)(n-1)$, substitute it into the above equation, and we obtain:

$$\mathcal{E}(\widehat{Q}_{tot}(D)) - \mathcal{E}(Q_{tot}^{\star}) \leq C_{final1} \cdot \frac{(B_{w,2})^D - 1}{B_{w,2} - 1} - C_{final2} \cdot D + C_{final3} \tag{49}$$

C.2 Derivation of Equation (10)

To facilitate the derivation, we consider all factor nodes represented by $A_t{}'$ as a single entity. Then, the action selection process aims to solve the following problem:

$$\begin{aligned}
\boldsymbol{u}_t^* &= \arg\max_{\boldsymbol{u}_t} Q_{tot}(\boldsymbol{\tau}_t, \boldsymbol{u}_t, A_t; \theta, \psi) \\
&= \arg\max_{(u_t^1,\ldots,u_t^n)} \left(\sum_{j\in\mathcal{J}} \frac{1}{|\mathcal{J}_{D(j)}|} Q_j(\boldsymbol{u}_t^j \mid \boldsymbol{h}_t^j; \theta_{D(j)}) + \frac{1}{n}\sum_{i=1}^{n} Q_i(u_t^i \mid h_t^i; \theta_{D(1)}) \right) \\
&= \arg\max_{(u_t^1,\ldots,u_t^n)} \left(\sum_{j\in J'} Q_j(\boldsymbol{u}_t^j \mid \boldsymbol{h}_t^j; \theta_{D(j)}) \right) \\
&= \arg\max_{(u_t^1,\ldots,u_t^n)} \left(\sum_{j\in J'} Q_j(\{v_i\}_{\{i,j\}\in\mathcal{E}'}) \right)
\end{aligned} \tag{50}$$

where $\mathcal{J}' = \mathcal{J} \bigcup \{m+i\}_{i=1}^{n}$,$\mathcal{E}' = \mathcal{E} \bigcup \{\{i,j\} \mid 1 \le i \le n, j = j+1(j = 1 \sim n)\}$. For ease of writing, the normalized parameters are omitted in the third line.

C.3 Derivation of Equation (17)

The derivation of Eq. (13) in Section IV-B of the main text is as follows:

Since the graph policy network parameters are φ, we first derive the derivatives of $J(\theta, \psi, \varphi)$ with respect to φ. Note that the subsequent gradients are all found for φ, and the graph policies are determined by φ. For the sake of brevity of the derivation, we will $\nabla_\varphi, \pi_{\theta,\psi}, v_{\Pi_{\theta,\psi,\varphi}}, Q_{\Pi_{\theta,\psi,\varphi}}$ are abbreviated as $\nabla, \pi, v_\Pi, Q_\Pi$ respectively.

$$\begin{aligned}
&\nabla_\varphi J(\theta,\psi,\varphi) = \nabla_\varphi v_{\Pi_{\theta,\psi,\varphi}}(s_0) = \nabla_\varphi \left[\sum_{\boldsymbol{u}_0, A_0} \Pi_{\theta,\psi,\varphi}(\boldsymbol{u}_0, A_0 \mid s_0) Q_\Pi(s_0, \boldsymbol{u}_0, A_0) \right] \\
&= \nabla_\varphi \left[\sum_{\boldsymbol{u}_0} \pi_{\theta,\psi}(\boldsymbol{u}_0 \mid s_0, A_0) \sum_{A_0} \rho_\varphi(A_0 \mid s_0) Q_\Pi(s_0, \boldsymbol{u}_0, A_0) \right] \\
&= \sum_{\boldsymbol{u}_0} \left[\begin{array}{l} \pi(\boldsymbol{u}_0 \mid s_0, A_0) \nabla \sum_{A_0} \rho_\varphi(A_0 \mid s_0) Q_\Pi(s_0, \boldsymbol{u}_0, A_0) \\ + \pi(\boldsymbol{u}_0 \mid s_0, A_0) \sum_{A_0} \rho_\varphi(A_0 \mid s_0) \nabla Q_\Pi(s_0, \boldsymbol{u}_0, A_0) \end{array} \right] \\
&= \sum_{\boldsymbol{u}_0} \left[\begin{array}{l} \pi(\boldsymbol{u}_0 \mid s_0, A_0) \sum_{A_0} \nabla \rho_\varphi(A_0 \mid s_0) Q_\Pi(s_0, \boldsymbol{u}_0, A_0) \\ + \pi(\boldsymbol{u}_0 \mid s_0, A_0) \sum_{A_0} \rho_\varphi(A_0 \mid s_0) \nabla \sum_{s_1, r_1} p(s_1, r_1 \mid s_0, \boldsymbol{u}_0, A_0)(r_1 + \gamma v_\Pi(s_1)) \end{array} \right] \\
&= \sum_{\boldsymbol{u}_0} \pi(\boldsymbol{u}_0 \mid s_0, A_0) \sum_{A_0} \nabla \rho_\varphi(A_0 \mid s_0) Q_\Pi(s_0, \boldsymbol{u}_0, A_0) \\
&\quad + \sum_{\boldsymbol{u}_0} \pi(\boldsymbol{u}_0 \mid s_0, A_0) \sum_{A_0} \rho_\varphi(A_0 \mid s_0) \sum_{s_1} p(s_1 \mid s_0, \boldsymbol{u}_0, A_0) \cdot \gamma \nabla v_\Pi(s_1) \\
&= \sum_{\boldsymbol{u}_0} \pi(\boldsymbol{u}_0 \mid s_0, A_0) \sum_{A_0} \nabla \rho_\varphi(A_0 \mid s_0) Q_\Pi(s_0, \boldsymbol{u}_0, A_0) \\
&\quad + \sum_{\boldsymbol{u}_0} \pi(\boldsymbol{u}_0 \mid s_0, A_0) \sum_{A_0} \rho_\varphi(A_0 \mid s_0) \sum_{s_1} p(s_1 \mid s_0, \boldsymbol{u}_0) \cdot \gamma \sum_{\boldsymbol{u}_1} \pi(\boldsymbol{u}_1 \mid s_1, A_1) \\
&\quad \sum_{A_1} \nabla \rho_\varphi(A_1 \mid s_1) Q_\Pi(s_1, \boldsymbol{u}_1, A_1) \\
&\quad + \sum_{\boldsymbol{u}_0} \pi(\boldsymbol{u}_0 \mid s_0, A_0) \sum_{A_0} \rho_\varphi(A_0 \mid s_0) \sum_{s_1} p(s_1 \mid s_0, \boldsymbol{u}_0) \cdot \gamma \sum_{\boldsymbol{u}_1} \pi(\boldsymbol{u}_1 \mid s_1, A_1) \\
&\quad \sum_{A_1} \nabla \rho_\varphi(A_1 \mid s_1) \sum_{s_2} p(s_2 \mid s_1, \boldsymbol{u}_1, A_1) \cdot \gamma \nabla v_\Pi(s_1)
\end{aligned}$$

$$
\begin{aligned}
\nabla_{\varphi} J(\theta,\psi,\varphi) &= \sum_{\boldsymbol{u}_0} \pi\left(\boldsymbol{u}_0 \mid s_0, A_0\right) \sum_{A_0} \nabla \rho_{\varphi}\left(A_0 \mid s_0\right) Q_{\Pi}\left(s_0, \boldsymbol{u}_0, A_0\right) \\
&+ \sum_{\boldsymbol{u}_0} \pi\left(\boldsymbol{u}_0 \mid s_0, A_0\right) \sum_{A_0} \rho_{\varphi}\left(A_0 \mid s_0\right) \sum_{s_1} p(s_1 \mid s_0, \boldsymbol{u}_0) \cdot \gamma \sum_{\boldsymbol{u}_1} \pi\left(\boldsymbol{u}_1 \mid s_1, A_1\right) \\
&\sum_{A_1} \nabla \rho_{\varphi}\left(A_1 \mid s_1\right) Q_{\Pi}\left(s_1, \boldsymbol{u}_1, A_1\right) + \cdots \\
&= \sum_{s_0} \Pr\left(s_0 \rightarrow s_0, 0, \Pi\right) \sum_{\boldsymbol{u}_0} \pi\left(\boldsymbol{u}_0 \mid s_0, A_0\right) \sum_{A_0} \nabla \rho_{\varphi}\left(A_0 \mid s_0\right) \gamma^0 Q_{\Pi}\left(s_0, \boldsymbol{u}_0, A_0\right) \\
&+ \sum_{s_1} \Pr\left(s_0 \rightarrow s_1, 1, \Pi\right) \sum_{\boldsymbol{u}_1} \pi\left(\boldsymbol{u}_1 \mid s_1, A_1\right) \sum_{A_1} \nabla \rho_{\varphi}\left(A_1 \mid s_1\right) \gamma^1 Q_{\Pi}\left(s_1, \boldsymbol{u}_1, A_1\right) + \cdots \\
&= \sum_{s_0} \Pr\left(s_0 \rightarrow s_0, 0, \Pi\right) \sum_{\boldsymbol{u}_0} \pi\left(\boldsymbol{u}_0 \mid s_0, A_0\right) \sum_{A_0} \rho_{\varphi}\left(A_0 \mid s_0\right) \left[\gamma^0 Q_{\Pi}\left(s_0, \boldsymbol{u}_0, A_0\right) \nabla \ln \rho_{\varphi}\left(A_0 \mid s_0\right)\right] \\
&+ \sum_{s_1} \Pr\left(s_0 \rightarrow s_1, 1, \Pi\right) \sum_{\boldsymbol{u}_1} \pi\left(\boldsymbol{u}_1 \mid s_1, A_1\right) \sum_{A_1} \rho_{\varphi}\left(A_1 \mid s_1\right) \left[\gamma^1 Q_{\Pi}\left(s_1, \boldsymbol{u}_1, A_1\right) \nabla \ln \rho_{\varphi}\left(A_1 \mid s_1\right)\right] \\
&+ \cdots \\
&= \sum_{t=0}^{\infty} \sum_{s_t} \Pr\left(s_0 \rightarrow s_t, t, \Pi\right) \sum_{\boldsymbol{u}_t} \pi\left(\boldsymbol{u}_t \mid s_t, A_t\right) \sum_{A_t} \rho_{\varphi}\left(A_t \mid s_t\right) \left[\gamma^t Q_{\Pi}\left(s_t, \boldsymbol{u}_t, A_t\right) \nabla \ln \rho_{\varphi}\left(A_t \mid s_t\right)\right]
\end{aligned}
$$

where γ is the discount factor, $\Pr\left(s_0 \rightarrow s_0, 0, \Pi\right) = 1$, $\Pr\left(s_0 \rightarrow s_1, 1, \Pi\right) = \sum_{\boldsymbol{u}_0} \Pi\left(\boldsymbol{u}_0 \mid s_0\right) p\left(s_1 \mid s_0, \boldsymbol{u}_0, A_0\right) = \sum_{a_0} \pi\left(\boldsymbol{u}_0 \mid s_0\right) \sum_{A_0} \rho_{\varphi}\left(A_0 \mid s_0\right) p\left(s_1 \mid s_0, \boldsymbol{u}_0, A_0\right)$.

Then, we sum over t moments in ∇J:

$$
\begin{aligned}
&\nabla_{\varphi} J(\theta,\psi,\varphi) \\
&= \sum_{t=0}^{\infty} \sum_{s_t} \Pr\left(s_0 \rightarrow s_t, t, \Pi\right) \sum_{\boldsymbol{u}_t} \pi\left(\boldsymbol{u}_t \mid s_t, A_t\right) \\
&\sum_{A_t} \rho_{\varphi}\left(A_t \mid s_t\right) \left[\gamma^t Q_{\Pi}\left(s_t, \boldsymbol{u}_t, A_t\right) \nabla \ln \rho_{\varphi}\left(A_t \mid s_t\right)\right] \\
&= \sum_{t=0}^{\infty} \sum_{s_t} \gamma^t \Pr\left(s_0 \rightarrow s_t, t, \Pi\right) \sum_{\boldsymbol{u}_t} \pi\left(\boldsymbol{u}_t \mid s_t, A_t\right) \\
&\sum_{A_t} \rho_{\varphi}\left(A_t \mid s_t\right) \left[Q_{\Pi}\left(s_t, \boldsymbol{u}_t, A_t\right) \nabla \ln \rho_{\varphi}\left(A_t \mid s_t\right)\right] \\
&= \sum_{x \in S} \sum_{t=0}^{\infty} \gamma^t \Pr\left(s_0 \rightarrow x, t, \Pi\right) \sum_{\boldsymbol{u}} \pi\left(\boldsymbol{u} \mid x, A\right) \\
&\sum_{A} \rho_{\varphi}\left(A \mid x\right) \left[Q_{\Pi}\left(x, \boldsymbol{u}, A\right) \nabla \ln \rho_{\varphi}\left(A \mid x\right)\right] \\
&= \sum_{x \in S} d^{\Pi}\left(x\right) \sum_{\boldsymbol{u}} \pi\left(\boldsymbol{u} \mid x, A\right) \sum_{A} \rho_{\varphi}\left(A \mid x\right) \left[Q_{\Pi}\left(x, \boldsymbol{u}, A\right) \nabla \ln \rho_{\varphi}\left(A \mid x\right)\right]
\end{aligned}
\tag{51}
$$

where $d^{\Pi}\left(x\right) = \sum\limits_{t=0}^{\infty} \gamma^t \Pr\left(s_0 \rightarrow x, t, \Pi\right)$ is the discounted state distribution.

Finally, write ∇J in the expected form:

$$
\begin{aligned}
&\nabla_{\varphi} J(\theta,\psi,\varphi) \\
&= \sum_{x \in S} d^{\Pi}\left(x\right) \sum_{\boldsymbol{u}} \pi\left(\boldsymbol{u} \mid x, A\right) \sum_{A} \rho_{\varphi}\left(A \mid x\right) \left[Q_{\Pi}\left(x, \boldsymbol{u}, A\right) \nabla \ln \rho_{\varphi}\left(A \mid x\right)\right] \\
&= \frac{1}{1-\gamma} \sum_{x \in S} \left(1-\gamma\right) d^{\Pi}\left(x\right) \sum_{\boldsymbol{u}} \pi\left(\boldsymbol{u} \mid x, A\right) \sum_{A} \rho_{\varphi}\left(A \mid x\right) \left[Q_{\Pi}\left(x, \boldsymbol{u}, A\right) \nabla \ln \rho_{\varphi}\left(A \mid x\right)\right] \\
&= \frac{1}{1-\gamma} \sum_{x \in S} D^{\Pi}\left(x\right) \sum_{\boldsymbol{u}} \pi\left(\boldsymbol{u} \mid x, A\right) \sum_{A} \rho_{\varphi}\left(A \mid x\right) \left[Q_{\Pi}\left(x, \boldsymbol{u}, A\right) \nabla \ln \rho_{\varphi}\left(A \mid x\right)\right] \\
&= \frac{1}{1-\gamma} \mathop{E}_{\substack{s \sim D^{\Pi} \\ A \sim \rho_{\varphi} \\ \boldsymbol{u} \sim \pi}} \left[Q_{\Pi}\left(s, \boldsymbol{u}, A\right) \nabla \ln \rho_{\varphi}\left(A \mid s\right)\right] \\
&\propto \mathop{E}_{\substack{s \sim D^{\Pi} \\ A \sim \rho_{\varphi} \\ \boldsymbol{u} \sim \pi}} \left[Q_{\Pi}\left(s, \boldsymbol{u}, A\right) \nabla \ln \rho_{\varphi}\left(A \mid s\right)\right]
\end{aligned}
$$

where $D^{\Pi}(x)$ is the standard distribution and $\sum_{x \in S} D^{\Pi}(x) = 1$.

$$\begin{aligned}
&\sum_{x \in S} D^{\Pi}(x) = (1-\gamma) \sum_{x \in S} d^{\Pi}(x) \\
&= (1-\gamma) \sum_{x \in S} \sum_{t=0}^{\infty} \gamma^t \Pr(s_0 \to x, t, \Pi) \\
&= (1-\gamma) \sum_{t=0}^{\infty} \gamma^t \sum_{x \in S} \Pr(s_0 \to x, t, \Pi) \\
&= (1-\gamma) \sum_{t=0}^{\infty} \gamma^t \\
&= 1
\end{aligned}$$

We usually ignore the scale factor so that the final form of ∇J is:

$$\nabla_{\varphi} J(\theta, \psi, \varphi) = \underset{(s, \boldsymbol{u}, A) \sim \mathcal{T}}{E} \left[Q_{\Pi}(s, \boldsymbol{u}, A) \nabla \ln \rho_{\varphi}(A \mid s) \right] \tag{52}$$

where $\mathcal{T} = \left(s_0, \{o_0^i\}_{i=1}^n, A_0, u_o^i, r_0, \ldots, s_T, \{o_T^i\}_{i=1}^n\right)$.

C.4 Proof of Lemma 1

Lemma 1. (The Lipschitz condition for MSE loss) A loss function $\mathcal{L}(f(x), y)$ is the MSE loss, and $f(x) \in [-C/2, C/2]$, then $\mathcal{L}(f(x), y)$ be $2C$-Lipschitz.

Proof. It is known that $\mathcal{L}(f(x), y)$ is the MSE loss, that is, $\mathcal{L}(f(x), y) = (f(x) - y)^2$. Then, $\frac{\partial \mathcal{L}}{\partial f} = 2(f - y)$, and we already know that $f(x) \in [-C/2, C/2]$. We can derive that $\max |2(f - y)| = 2C$.

According to the Derivative Lipschitz Theorem, if a function is differentiable and its derivative is bounded, then the function is Lipschitz continuous, and the Lipschitz constant L is equal to the maximum value of the derivative. Thus, the MSE loss be $2C$-Lipschitz within $[-C/2, C/2]$.

C.5 Proof of Theorem 1

Theorem 1. Let $\mathcal{L}$ be $2C_Q - Lipschitz$, $\delta \in (0, 1]$, and Assumption 1 hold with constants $\{C_1, C_2, \gamma\}$. The generalization error for the optimal global value function $Q_{tot}^{\star}$ be $\mathcal{E}(Q_{tot}^{\star})$ and the generalization error for the empirical risk minimizer $\widehat{Q_{tot}^{D,K}}$ be $\mathcal{E}(\widehat{Q_{tot}^{D,K}})$ with $K = \{K_d\}_{d=1}^D$. Then, we have for L_2-regularized ERM, where $\left\|w_{dk}(u_t^i)\right\|_2 \leq B_{w,2}$, $1 \leq d \leq D$, $B_{h,2} = \sup_h \left\|\bar{Q}_i\right\|_2$, and $\|\boldsymbol{\lambda}\|_2 \leq B_{\lambda,2}$ where $\boldsymbol{\lambda} = \{\{\lambda_{dk}\}_{k=1}^{K_d}\}_{d=1}^D$, with probability at least $1 - \delta$,

$$\begin{aligned}
\mathcal{E}(\widehat{Q_{tot}^{D,K}}) - \mathcal{E}(Q_{tot}^{\star}) \leq 4C_Q B_{\lambda,2} B_{h,2} \sum_{d=1}^{D} \left((B_{w,2})^d \sqrt{\frac{K_d}{n_s}} \right) \\
+ \sum_{d=D+1}^{n} \frac{C_1}{C_2} \exp\left(-(K_d + 1)^{\gamma}\right) + \left(2(C_Q)^2 + C_Q\right) \sqrt{\frac{2\log(4/\delta)}{n_s}}
\end{aligned} \tag{53}$$

where n_s denotes the number of samples, K_d denotes the tensor rank for each order d.

Proof. For any function $f : X \to Y$ and any bounded $2C$-Lipschitz loss function, the training error over n_s samples as,

$$\hat{\mathcal{E}}_{n_s}(f) = \frac{1}{n_s} \sum_{i=1}^{n_s} \mathcal{L}(f(x_i), y)$$

Similarly, we define the expected risk over the sample distribution $\mathcal{B}$,

$$\mathcal{E}(f) = E_{(x,y) \sim \mathcal{B}}[\mathcal{L}(x, y)]$$

Our objective is to bound the expected risk of the ERM low-rank decomposed global value function $\mathcal{E}(\widehat{Q_{tot}^{D,K}})$ with the maximum order D and its corresponding rank $K = \{K_d\}_{d=1}^D$, i.e., $\mathcal{E}(Q_{tot}^{\star})$. Note that there are no restrictions on the order and rank of $\mathcal{E}(Q_{tot}^{\star})$. Then, we define the global optimal Q-value:

$$Q_{tot}^{\star} = \arg\min_{\theta} \left(\mathcal{E}(Q_{tot}(\cdot; \theta))\right)$$

Similarly, we have that the empirical risk minimizer low-rank decomposed global value function $\mathcal{E}(\widehat{Q_{tot}^{D,K}})$ satisfies,

$$\widehat{Q_{tot}^{D,K}} = \arg\min_{\theta} \left(\hat{\mathcal{E}}_{n_s}(Q_{tot}(\cdot;\theta))\right) \tag{54}$$

We can now begin the proof. For the optimal global value function $Q_{tot}^{\star}$, we denote the corresponding singular values as $\{\{\lambda_{dk}^{\star}\}_{k=1}^{K_d^{\max}}\}_{d=1}^{n}$($n$ denotes the number of agents, and the maximum order of the corresponding global value function is n), the network parameter as $\{\{\underbrace{w_{dk}^{\star}}\}_{k=1}^{K_d^{\max}}\}_{d=1}^{n}$, the bias as $f_0^{\star}$. We will bound the excess risk $\mathcal{E}(\widehat{Q_{tot}^{D,K}}) - \mathcal{E}(Q_{tot}^{\star})$ by introducing a third global value function $\widetilde{Q_{tot}^{D,K}}$ which is defined as a truncated version of $Q_{tot}^{\star}$ with the maximum order D and its corresponding rank $K = \{K_d\}_{d=1}^{D}$. Specifically, we set the singular values as $\{\{\widetilde{\lambda}_{dk}\}_{k=1}^{K_d^{\max}}\}_{d=1}^{n}$, the network parameter as $\{\{\widetilde{w}_{dk}\}_{k=1}^{K_d^{\max}}\}_{d=1}^{n}$, the bias as $\widetilde{f}_0$ as,

$$For\ 1 \le k \le K_d,\ 1 \le d \le D,\ \widetilde{\lambda}_{dk} = \lambda_{dk}^{\star}, \widetilde{w}_{dk} = w_{dk}^{\star}, \widetilde{f}_0 = f_0^{\star}$$

Thus, we can write the error upper bound as:

$$\mathcal{E}(\widehat{Q_{tot}^{D,K}}) - \mathcal{E}(Q_{tot}^{\star}) = \underbrace{\mathcal{E}(\widehat{Q_{tot}^{D,K}}) - \hat{\mathcal{E}}_{n_s}(\widehat{Q_{tot}^{D,K}})}_{(A)} + \underbrace{\hat{\mathcal{E}}_{n_s}(\widehat{Q_{tot}^{D,K}}) - \hat{\mathcal{E}}_{n_s}(\widetilde{Q_{tot}^{D,K}})}_{\le 0} + \underbrace{\hat{\mathcal{E}}_{n_s}(\widetilde{Q_{tot}^{D,K}}) - \mathcal{E}(Q_{tot}^{\star})}_{(B)} \tag{55}$$

The term $\hat{\mathcal{E}}_{n_s}(\widehat{Q_{tot}^{D,K}}) - \hat{\mathcal{E}}_{n}(\widetilde{Q_{tot}^{D,K}}) \le 0$ since $\widehat{Q_{tot}^{D,K}}$ minimizes the empirical risk(Eq. 54). Therefore, we need to restrict (A) and (B) in sequence and calculate their upper bounds respectively.

(A) According to Lemma 1, Theorem 8 of Bartlett and Mendelson [12], and McDiarmid's inequality, We have that with probability at least $1 - \delta/2$ for any $\delta \in (0, 1]$,

$$\mathcal{E}(\widehat{Q_{tot}^{D,K}}) - \hat{\mathcal{E}}_{n_s}(\widehat{Q_{tot}^{D,K}}) \le \mathcal{R}_n(\mathcal{L} \circ Q(\Theta_{D,K})) + 2(C_Q)^2\sqrt{\frac{2\log(4/\delta)}{n_s}} \tag{56}$$

where $\mathcal{R}_{n_s}(\mathcal{F}) = E_{\epsilon}[\sup_{f\in\mathcal{F}} \frac{1}{n_s}\sum_{i=1}^{n_s} \epsilon_i f(x_i)]$ is the empirical Rademacher complexity of the function family $\mathcal{F}$.

According to Lemma 1, $\mathcal{R}_{n_s}(\mathcal{L} \circ Q(\Theta_{K,d})) \le 2 \cdot 2C_Q \cdot \mathcal{R}_{n_s}(Q) = 4C_Q \sum_{d=1}^{D} \frac{1}{|J_d|} \sum_{j\in J_d} \mathcal{R}_j(Q_d)$.

We have $Q_{j,d} = \sum_{k=1}^{K_d} \lambda_{dk} \prod_{i=1}^{d} \langle w_{dk}(u_t^i), h_t^i \rangle$,with $\|\lambda_k\|_2 \le \|\lambda\|_2 \le B_{\lambda,2}$, $\lambda = \{\lambda_{dk}\}_{k=1}^{K_d}$, substituting it into a, we obtain:

$$\mathcal{R}_j(Q_d) = E_{\epsilon}[\sup_{\substack{\|\lambda_{dk}\|_2 \le B_{\lambda,2} \\ \left\|w_{dk}(u_t^i)\right\|_2 \le B_{w,2}}} \frac{1}{n_s}\sum_{s=1}^{n_s} \epsilon_s \sum_{k=1}^{K_d} \lambda_{dk} \prod_{i=1}^{d} \langle w_{dk}(u_t^i), h_t^i \rangle]$$

According to the Cauchy-Schwarz inequality and the independent and identically distributed (i.i.d.) assumption:

$$\begin{aligned}
\mathcal{R}_j(Q_d) &= E_{\epsilon}[\sup_{\substack{\|\lambda_{dk}\|_2 \le B_{\lambda,2} \\ \left\|w_{dk}(u_t^i)\right\|_2 \le B_{w,2}}} \frac{1}{n_s}\sum_{k=1}^{K_d} \lambda_{dk} \sum_{s=1}^{n_s} \epsilon_s \prod_{i=1}^{d} \langle w_{dk}(u_t^i), h_t^i \rangle] \\
&\le \frac{1}{n_s} \sup_{\|\lambda_{dk}\|_2 \le B_{\lambda,2}} \left(\sum_{k=1}^{K_d} \lambda_{dk}^2\right)^{1/2} \cdot \sqrt{\sum_{k=1}^{K_d} E_{\epsilon}[\sup_{\left\|w_{dk}(u_t^i)\right\|_2 \le B_{w,2}} \sum_{s=1}^{n_s} \epsilon_s \left(\prod_{i=1}^{d} \langle w_{dk}(u_t^i), h_t^i \rangle\right)^2]} \\
&\le \frac{1}{n_s} B_{\lambda,2} \cdot \sqrt{\sum_{k=1}^{K_d} E_{\epsilon}[\sup_{\left\|w_{dk}(u_t^i)\right\|_2 \le B_{w,2}} \sum_{s=1}^{n_s} \epsilon_s^2 \cdot \prod_{i=1}^{d} \langle w_{dk}(u_t^i), h_t^i \rangle^2]}
\end{aligned}$$

According to the Khintchine inequality:

$$\begin{aligned}
&\sqrt{\sum_{k=1}^{K_d} E_{\epsilon}[\sup_{\left\|w_{dk}(u_t^i)\right\|_2 \le B_{w,2}} \sum_{s=1}^{n_s} \epsilon_s^2 \cdot \prod_{i=1}^{d} \langle w_{dk}(u_t^i), h_t^i \rangle^2]} \le \sqrt{\sum_{k=1}^{K_d} \sum_{s=1}^{n_s} \prod_{i=1}^{d} \langle w_{dk}(u_t^i), h_t^i \rangle^2} \\
&\le (B_{w,2})^d \sqrt{\sum_{k=1}^{K_d} n_s (B_{h,2})^d}
\end{aligned}$$

There is an inherent problem in learning high-order local Q-values: as the order of interaction terms increases, the product of two features differs from the original values by an order of magnitude,and consequently, higher-order products are progressively

disproportionate. To mitigate this, we rescale features such that (a) the scale is preserved across terms, and (b) the variance in interactions is captured. We scale the Q-values, $\bar{Q}_i = sign(\bar{Q}_i) \cdot \bar{Q}_i^{1/d}$ and we define $B_{h,2} = \sup_h \left\|\bar{Q}_i\right\|_2$, then we get:

$$\begin{aligned} \mathcal{R}_j(Q_d) &\leq \frac{1}{n_s} B_{\lambda,2} \cdot (B_{w,2})^d B_{h,2} \sqrt{\sum_{k=1}^{K_d} n_s} \\ &= B_{\lambda,2} B_{h,2} \cdot (B_{w,2})^d \sqrt{\frac{K_d}{n_s}} \end{aligned} \tag{57}$$

Substituting Eq. 57 into $\mathcal{R}_{n_s}(\mathcal{L} \circ Q(\Theta_{K,d}))$:

$$\begin{aligned} &\mathcal{R}_{n_s}(\mathcal{L} \circ Q(\Theta_{D,K})) \leq 4C_Q \sum_{d=1}^{D} \frac{1}{|J_d|} \sum_{j \in J_d} B_{\lambda,2} B_{h,2} \cdot (B_{w,2})^d \sqrt{\frac{K_d}{n_s}} \\ &= 4C_Q B_{\lambda,2} B_{h,2} \sum_{d=1}^{D} \left((B_{w,2})^d \sqrt{\frac{K_d}{n_s}} \right) \end{aligned}$$

Finally, the term (A) becomes:

$$\mathcal{E}(\widetilde{Q_{tot}^{D,K}}) - \hat{\mathcal{E}}_{n_s}(\widetilde{Q_{tot}^{D,K}}) \leq 4C_Q B_{\lambda,2} B_{h,2} \sum_{d=1}^{D} \left((B_{w,2})^d \sqrt{\frac{K_d}{n_s}} \right) + 2(C_Q)^2 \sqrt{\frac{2\log(4/\delta)}{n_s}} \tag{58}$$

(B) The derivation of term (B):

$$\begin{aligned} &\hat{\mathcal{E}}_{n_s}(\widetilde{Q_{tot}^{D,K}}) - \mathcal{E}(Q_{tot}^{\star}) = \hat{\mathcal{E}}_{n_s}(\widetilde{Q_{tot}^{D,K}}) - \mathcal{E}(\widetilde{Q_{tot}^{D,K}}) + \mathcal{E}(\widetilde{Q_{tot}^{D,K}}) - \mathcal{E}(Q_{tot}^{\star}) \\ &\leq \underbrace{\left| \hat{\mathcal{E}}_{n_s}(\widetilde{Q_{tot}^{D,K}}) - \mathcal{E}(\widetilde{Q_{tot}^{D,K}}) \right|}_{(B.1)} + \underbrace{\left| \mathcal{E}(\widetilde{Q_{tot}^{D,K}}) - \mathcal{E}(Q_{tot}^{\star}) \right|}_{(B.2)} \end{aligned} \tag{59}$$

(B.1) We can apply the Azuma-Hoeffding inequality [13]:

$$P\left(\left| \hat{\mathcal{E}}_{n_s}(\widetilde{Q_{tot}^{D,K}}) - \mathcal{E}(\widetilde{Q_{tot}^{D,K}}) \right| \geq t \right) \leq 2\exp\left(-\frac{2n^2t^2}{\sum_{i=1}^{n_s} c_i^2} \right) \leq 2\exp\left(-\frac{2n^2t^2}{\sum_{i=1}^{n_s} (2C_Q)^2} \right)$$

Then, we obtain with probability at least $1 - \delta/2$, $\delta \in (0, 1]$:

$$\left| \hat{\mathcal{E}}_{n_s}(\widetilde{Q_{tot}^{D,K}}) - \mathcal{E}(\widetilde{Q_{tot}^{D,K}}) \right| \leq C_Q \sqrt{\frac{2\log(4/\delta)}{n_s}} \tag{60}$$

(B.2) According to Lemma 1:

$$\begin{aligned} &\left| \mathcal{E}(\widetilde{Q_{tot}^{\mathrm{D},K}}) - \mathcal{E}(Q_{tot}^{\star}) \right| \leq \left| E_{(h_t,y)\sim\mathcal{B}} \left[\mathcal{L}(\widetilde{Q_{tot}^{K,d}}(h_t), y) - \mathcal{L}(Q_{tot}^{\star}(h_t), y) \right] \right| \\ &\leq E_{(h_t,y)\sim\mathcal{B}} \left[\left| \mathcal{L}(\widetilde{Q_{tot}^{K,d}}(h_t), y) - \mathcal{L}(Q_{tot}^{\star}(h_t), y) \right| \right] \\ &\leq E_{(h_t,y)\sim\mathcal{B}} \left[\left| 2C_Q \cdot \left| \widetilde{Q_{tot}^{K,d}}(h_t) - y \right| - 2C_Q \cdot |Q_{tot}^{\star}(h_t) - y| \right| \right] \\ &\leq 2C_Q \cdot E_{(h_t,y)\sim\mathcal{B}} \left[\left| \widetilde{Q_{tot}^{K,d}}(h_t) - Q_{tot}^{\star}(h_t) \right| \right] \\ &\leq 2C_Q \cdot \sup_{h_t} \left| \widetilde{Q_{tot}^{K,d}}(h_t) - Q_{tot}^{\star}(h_t) \right| \end{aligned}$$

In the main text, we present the decomposition form of the global value function:

$$Q_{tot}(\boldsymbol{\tau}_t, \boldsymbol{u}_t) = f_0 + \frac{1}{n} \sum_{i=1}^{n} \langle w_1(u_t^i), h_t^i \rangle + \frac{1}{|J_2|} \sum_{j \in J_2} \sum_{k=1}^{K_2} \lambda_{2k} \prod_{i=1}^{2} \langle w_{2k}(u_t^i), h_t^i \rangle + \cdots + \sum_{k=1}^{K_n} \lambda_{nk} \prod_{i=1}^{n} \langle w_{nk}(u_t^i), h_t^i \rangle \tag{61}$$

According to the preceding text, we define the truncated version of $Q_{tot}^{\star}$ and substitute Eq. (61) into it to obtain:

$$\left|\widetilde{Q_{tot}^{D,K}}(h_t) - Q_{tot}^{\star}(h_t)\right| = \left|\sum_{d=D+1}^{n} \frac{1}{|J_d|} \sum_{j\in J_d} \sum_{k=K_d+1}^{K_d^{\max}} \lambda_{dk}^{\star} \prod_{i=1}^{d} \langle w_{dk}(u_t^i), h_t^i\rangle\right|$$
$$\leq \sum_{d=D+1}^{n} \frac{1}{|J_d|} \sum_{j\in J_d} \sum_{k=K_d+1}^{K_d^{\max}} \left|\lambda_{dk}^{\star} \prod_{i=1}^{d} \langle w_{dk}(u_t^i), h_t^i\rangle\right|$$
$$\leq B_{h,2} \sum_{d=D+1}^{n} (B_{w,2})^d \frac{1}{|J_d|} \sum_{j\in J_d} \sum_{k=K_d+1}^{K_d^{\max}} |\lambda_{dk}^{\star}|$$

According to Lemma 2:

$$\sum_{k=K_d+1}^{K_d^{\max}} |\lambda_{dk}^{\star}| \leq \sum_{k=K_d+1}^{K_d^{\max}} C_1 \cdot \exp\left(-C_2 \cdot k^{\gamma}\right) \leq \int_{k=K_d+1}^{\infty} C_1 \cdot \exp\left(-C_2 \cdot k^{\gamma}\right)$$

According to Equation E.16 from Yang et al. [14], and given $\gamma > 1$:

$$\left|\widetilde{Q_{tot}^{D,K}}(\boldsymbol{h}_t) - Q_{tot}^{\star}(\boldsymbol{h}_t)\right| \leq \sum_{d=D+1}^{n} \frac{1}{|J_d|} \sum_{j\in J_d} \frac{C_1}{C_2} \exp\left(-(K_d+1)^{\gamma}\right)$$
$$= \sum_{d=D+1}^{n} \frac{C_1}{C_2} \exp\left(-(K_d+1)^{\gamma}\right) \frac{1}{|J_d|} \sum_{j\in J_d} 1$$
$$= \sum_{d=D+1}^{n} \frac{C_1}{C_2} \exp\left(-(K_d+1)^{\gamma}\right)$$

We combine $(B.1)$ and $(B.2)$, we obtain with probability at least $1-\delta/2$, $\delta \in (0,1]$:

$$\hat{\mathcal{E}}_{n_s}(\widetilde{Q_{tot}^{\mathrm{D},K}}) - \mathcal{E}(Q_{tot}^{\star}) \leq \sum_{d=D+1}^{n} \frac{C_1}{C_2} \exp\left(-(K_d+1)^{\gamma}\right) + C_Q \sqrt{\frac{2\log(4/\delta)}{n_s}} \tag{62}$$

Finally, we combine (A) and (B), we obtain with probability at least $1-\delta$, $\delta \in (0,1]$:

$$\begin{aligned}
&\mathcal{E}(\widetilde{Q_{tot}^{D,K}}) - \mathcal{E}(Q_{tot}^{\star}) \\
&\leq 4C_Q B_{\lambda,2} B_{h,2} \sum_{d=1}^{D} \left((B_{w,2})^d \sqrt{\frac{K_d}{n_s}}\right) + 2(C_Q)^2 \sqrt{\frac{2\log(4/\delta)}{n_s}} + \sum_{d=D+1}^{n} \frac{C_1}{C_2} \exp\left(-(K_d+1)^{\gamma}\right) + C_Q \sqrt{\frac{2\log(4/\delta)}{n_s}} \\
&= 4C_Q B_{\lambda,2} B_{h,2} \sum_{d=1}^{D} \left((B_{w,2})^d \sqrt{\frac{K_d}{n_s}}\right) + \sum_{d=D+1}^{n} \frac{C_1}{C_2} \exp\left(-(K_d+1)^{\gamma}\right) + \left(2(C_Q)^2 + C_Q\right) \sqrt{\frac{2\log(4/\delta)}{n_s}}
\end{aligned} \tag{63}$$

C.6 Proof of Proposition 1

Because $X \sim P_{D_{\max}}(D_{\max} : p_1, p_2, \ldots, p_N)$, there are $m_1 + m_2 + \cdots + m_N = D_{\max}$. When $\max m_i = 1$, Q_j connects exactly $D_{\max}$ agents, and its order is $D_{\max}$; when $\max m_i > 1$, there must be at least one agent i and Q_j is connected more than once, then the number of connected agents Q_j is less than $D_{\max}$, and its order is less than $D_{\max}$. So for each Q_j, its maximum order is $D_{\max}$, that is, $D_{\max}$ is The maximum order in the algorithm.

C.7 Proof of Proposition 2

Through Proposition 1, it can be seen that the "sub-policy" corresponding to Q_j needs to be able to generate a collaborative relationship of $D \leq D_{\max}$ agents. When the order Q_j generated by "sub-policy" is D_j, Q_j will be the same as D_j agents are connected, and the set of D_j agents is defined as $G \in H_{D_j}$, $H_{D_j} = \{\{i_1, \ldots, i_d, \ldots, i_{D_j}\} \mid i_d \in \{1, 2, \ldots, N\}\}$. Then the probability of the "sub-policy" corresponding to Q_j is the probability of Q_j being connected to all agents in G:

$$\tilde{P}\{G\} = \sum_{\{m_1,\ldots,m_N\}\in H_m} P\{X_1 = m_1, X_2 = m_2, \ldots, X_N = m_N\} \tag{64}$$

where $H_m = \{\{m_1, \ldots, m_N\} \mid \sum_{i_d \in G} m_{i_d} = D_{\max}, \forall i_d \in G, m_{i_d} \geq 1\}$.

Then, it is necessary to prove that $\tilde{P}\{G\}$ is the probability mass function of the class multinomial distribution $\tilde{P}_{D_{\max}}(D_{\max} : p_1, p_2, \ldots, p_N)$.

(1) $\tilde{P}\{G\} \geq 0$

Because $P\{X_1 = m_1, X_2 = m_2, \ldots, X_N = m_N\}$ is the multinomial distribution $P_{D_{\max}}(D_{\max} : p_1, p_2, \ldots, p_N)$ probability mass function, and $P\{X_1 = m_1, X_2 = m_2, \ldots, X_N = m_N\} \geq 0$, then according to Eq.(77), $\tilde{P}\{G\} \geq 0$.

(2) $\sum_{D_j=1}^{D_{\max}} \sum_{G \in H_{D_j}} \tilde{P}\{G\} = 1$

Because $P\{X_1 = m_1, X_2 = m_2, \ldots, X_N = m_N\}$ is the probability mass function of multinomial distribution $P_{D_{\max}}(D_{\max} : p_1, p_2, \ldots, p_N)$, and we have:

$$\sum_{m_1+m_2+\cdots+m_N=D_{\max}} P\{X_1 = m_1, X_2 = m_2, \ldots, X_N = m_N\} = 1 \tag{65}$$

where there are a total of $C_{D_{\max}+N-1}^{N-1}$ items in Eq.(65).

$$\begin{aligned} &\sum_{D_j=1}^{D_{\max}} \sum_{G \in H_{D_j}} \tilde{P}\{G\} \\ &= \sum_{D_j=1}^{D_{\max}} \sum_{G \in H_{D_j}} \sum_{\{m_1,\ldots,m_N\} \in H_m} P\{X_1 = m_1, X_2 = m_2, \ldots, X_N = m_N\} \end{aligned} \tag{66}$$

Next, we will calculate how many terms there are in equation Eq.(66):

$$\begin{aligned} &\sum_{D_j=1}^{D_{\max}} \sum_{G \in H_{D_j}} \sum_{\{m_1,\ldots,m_N\} \in H_m} 1 = \sum_{D_j=1}^{D_{\max}} C_N^{D_j} C_{D_{\max}-1}^{D_j-1} = \sum_{D_j=1}^{D_{\max}} C_N^{D_j} C_{D_{\max}-1}^{D_{\max}-D_j} \\ &= \sum_{D_j=0}^{D_{\max}} C_N^{D_j} C_{D_{\max}-1}^{D_{\max}-D_j} = C_{N+D_{\max}-1}^{D_{\max}} = C_{N+D_{\max}-1}^{N-1} \end{aligned} \tag{67}$$

According to the definition of H_{D_j}, for different D_j, H_{D_j} are disjoint with each other; and the composition of H_m depends on G and corresponds to G one-to-one, while $G \in H_{D_j}$, so there is no intersection between H_m. Then it means that there are no duplicates in the summation of $P\{X_1 = m_1, \ldots, X_N = m_N\}$. Through Eq. (67), we know that there are $C_{N+D_{\max}-1}^{N-1}$ terms in Eq. (66), and each term is The probability mass function in the multinomial distribution $P_{D_{\max}}(D_{\max} : p_1, p_2, \ldots, p_N)$, and does not repeat, then you can get $\sum_{D_j=1}^{D_{\max}} \sum_{G \in H_{D_j}} \tilde{P}\{G\} = 1$.

C.8 Proof of Theorem 2

Theorem 2. (Graph Policy Improvement Lower Bound). Consider the behavior (trajectory-collecting) graph policy ρ_{old}. For the current graph policy ρ_{new} that we need to improve, we have

$$\begin{aligned} J(\rho_{new}) - J(\rho_{old}) \geq & \frac{1}{1-\gamma} \underset{\substack{s \sim D^{\Pi_{old}} \\ A \sim \rho_{new} \\ u \sim \pi_{old}}}{E} \left[\mathcal{A}_{\rho_{old}, \pi_{old}}(s, A, \boldsymbol{u})\right] \\ & - \frac{C^{\Pi}\gamma}{(1-\gamma)^2} \underset{(s,A) \sim D^{\Pi_{old}}}{E} \left[\left| \frac{\rho_{new}(A|s)}{\rho_{old}(A|s)} - 1 \right|\right] \end{aligned} \tag{68}$$

where J denotes the objective function optimized in reinforcement learning, $C^{\Pi} = \max_{s \in S} \left| \underset{\substack{A \sim \rho_{new} \\ u \sim \pi_{old}}}{E} \left[\mathcal{A}_{\rho_{old}, \pi_{old}}(s, A, \boldsymbol{u})\right] \right|$, $\Pi = \rho\left(A|s\right) \pi\left(\boldsymbol{u}|s, A\right)$ denotes the joint policy of all agent policies.

Proof. The proof process of Theorem 2 mainly refers to CPO [15] and GePPO [16]. We define, during learning, the discrepancy between the new graph policy and the old graph policy after each update as follows:

$$J(\rho_{new}) - J(\rho_{old}) = E_{s_0, A_0, \boldsymbol{u}_0, s_1, A_1, \boldsymbol{u}_1, \ldots} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{A}_{\rho_{old}}(s_t, A_t, \boldsymbol{u}_t)\right]$$

Rearranging the above equation, we obtain:

$$\begin{aligned} &J(\rho_{new}) = J(\rho_{old}) + E_{s_0, A_0, \boldsymbol{u}_0, s_1, A_1, \boldsymbol{u}_1, \ldots} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{A}_{\rho_{old}}(s_t, A_t, \boldsymbol{u}_t)\right] \\ &= J(\rho_{old}) + \sum_{t=0}^{\infty} \gamma^t \sum_{s_t} p(s_t|s_{t-1}, \boldsymbol{u}_{t-1}, A_{t-1}) \sum_{A_t} \rho_{new}\left(A_t|s_t\right) \sum_{\boldsymbol{u}_t} \pi_{old}\left(\boldsymbol{u}_t|s_t, A_t\right) \mathcal{A}_{\rho_{old}, \pi_{old}}(s_t, A_t, \boldsymbol{u}_t) \\ &= J(\rho_{old}) + \sum_{s} \sum_{t=0}^{\infty} \gamma^t p_{\rho_{new}}(s) \sum_{A} \rho_{new}\left(A|s\right) \sum_{\boldsymbol{u}} \pi_{old}\left(\boldsymbol{u}|s, A\right) \mathcal{A}_{\rho_{old}, \pi_{old}}(s, A, \boldsymbol{u}) \end{aligned}$$

Starting from the above equation, we obtain:

$$
\begin{aligned}
J(\rho_{new}) - J(\rho_{old}) &= \sum_{s} \sum_{t=0}^{\infty} \gamma^t p_{\rho_{new}}(s) \sum_{A} \rho_{new}\left(A|s\right) \sum_{\boldsymbol{u}} \pi_{old}\left(\boldsymbol{u}|s, A\right) \mathcal{A}_{\rho_{old},\pi_{old}}(s, A, \boldsymbol{u}) \\
&= \frac{1}{1-\gamma} \sum_{s} (1-\gamma) \sum_{t=0}^{\infty} \gamma^t p_{\rho_{new}}(s) \sum_{A} \rho_{new}\left(A|s\right) \sum_{\boldsymbol{u}} \pi_{old}\left(\boldsymbol{u}|s, A\right) \mathcal{A}_{\rho_{old},\pi_{old}}(s, A, \boldsymbol{u}) \\
&= \frac{1}{1-\gamma} \underset{\substack{s\sim D^{\Pi_{new}} \\ A\sim\rho_{new} \\ \boldsymbol{u}\sim\pi_{old}}}{E} \left[\mathcal{A}_{\rho_{old},\pi_{old}}(s, A, \boldsymbol{u})\right]
\end{aligned}
$$

where $D^{\Pi_{new}} = (1-\gamma)\sum_{t=0}^{\infty} p(s_t = s|\rho_{new}, \pi_{old})$ is the discounted future state distribution.

By doing so, we have

$$
\begin{aligned}
&J(\rho_{new}) - J(\rho_{old}) = \frac{1}{1-\gamma} \underset{\substack{s\sim D^{\Pi_{new}} \\ A\sim\rho_{new} \\ \boldsymbol{u}\sim\pi_{old}}}{E} \left[\mathcal{A}_{\rho_{old},\pi_{old}}(s, A, \boldsymbol{u})\right] \\
&= \frac{1}{1-\gamma} \underset{\substack{s\sim D^{\Pi_{old}} \\ A\sim\rho_{new} \\ \boldsymbol{u}\sim\pi_{old}}}{E} \left[\mathcal{A}_{\rho_{old},\pi_{old}}(s, A, \boldsymbol{u})\right] + \frac{1}{1-\gamma} \left(\underset{\substack{s\sim D^{\Pi_{new}} \\ A\sim\rho_{new} \\ \boldsymbol{u}\sim\pi_{old}}}{E} \left[\mathcal{A}_{\rho_{old},\pi_{old}}(s, A, \boldsymbol{u})\right] - \underset{\substack{s\sim D^{\Pi_{old}} \\ A\sim\rho_{new} \\ \boldsymbol{u}\sim\pi_{old}}}{E} \left[\mathcal{A}_{\rho_{old},\pi_{old}}(s, A, \boldsymbol{u})\right] \right) \\
&\geq \frac{1}{1-\gamma} \underset{\substack{s\sim D^{\Pi_{old}} \\ A\sim\rho_{new} \\ \boldsymbol{u}\sim\pi_{old}}}{E} \left[\mathcal{A}_{\rho_{old},\pi_{old}}(s, A, \boldsymbol{u})\right] - \frac{1}{1-\gamma} \left| \underset{\substack{s\sim D^{\Pi_{new}} \\ A\sim\rho_{new} \\ \boldsymbol{u}\sim\pi_{old}}}{E} \left[\mathcal{A}_{\rho_{old},\pi_{old}}(s, A, \boldsymbol{u})\right] - \underset{\substack{s\sim D^{\Pi_{old}} \\ A\sim\rho_{new} \\ \boldsymbol{u}\sim\pi_{old}}}{E} \left[\mathcal{A}_{\rho_{old},\pi_{old}}(s, A, \boldsymbol{u})\right] \right|
\end{aligned} \tag{69}
$$

We bound the second term in Eq. (69) using Hölder's inequality:

$$
\begin{aligned}
&J(\rho_{new}) - J(\rho_{old}) \\
&\geq \frac{1}{1-\gamma} \underset{\substack{s\sim D^{\Pi_{old}} \\ A\sim\rho_{new} \\ \boldsymbol{u}\sim\pi_{old}}}{E} \left[\mathcal{A}_{\rho_{old},\pi_{old}}(s, A, \boldsymbol{u})\right] - \frac{1}{1-\gamma} \left\| D^{\Pi_{new}} - D^{\Pi_{old}} \right\|_1 \left\| \underset{\substack{A\sim\rho_{new} \\ \boldsymbol{u}\sim\pi_{old}}}{E} \left[\mathcal{A}_{\rho_{old},\pi_{old}}(s, A, \boldsymbol{u})\right] \right\|_{\infty}
\end{aligned} \tag{70}
$$

Define:

$$
\left\| \underset{\substack{A\sim\rho_{new} \\ \boldsymbol{u}\sim\pi_{old}}}{E} \left[\mathcal{A}_{\rho_{old},\pi_{old}}(s, A, \boldsymbol{u})\right] \right\|_{\infty} = \max_{s\in S} \left| \underset{\substack{A\sim\rho_{new} \\ \boldsymbol{u}\sim\pi_{old}}}{E} \left[\mathcal{A}_{\rho_{old},\pi_{old}}(s, A, \boldsymbol{u})\right] \right| = C^{\Pi} \tag{71}
$$

Substituting Lemma 2 and Eq. (71) into Eq. (70), we obtain:

$$
\begin{aligned}
&J(\rho_{new}) - J(\rho_{old}) \\
&\geq \frac{1}{1-\gamma} \underset{\substack{s\sim D^{\Pi_{old}} \\ A\sim\rho_{new} \\ \boldsymbol{u}\sim\pi_{old}}}{E} \left[\mathcal{A}_{\rho_{old},\pi_{old}}(s, A, \boldsymbol{u})\right] - \frac{1}{1-\gamma} \frac{2\gamma}{1-\gamma} \underset{s\sim D^{\Pi_{old}}}{E} \left[D_{TV}(\rho_{new}||\rho_{old})[s]\right] C^{\Pi} \\
&= \frac{1}{1-\gamma} \underset{\substack{s\sim D^{\Pi_{old}} \\ A\sim\rho_{new} \\ \boldsymbol{u}\sim\pi_{old}}}{E} \left[\mathcal{A}_{\rho_{old},\pi_{old}}(s, A, \boldsymbol{u})\right] - \frac{2C^{\Pi}\gamma}{(1-\gamma)^2} \underset{s\sim D^{\Pi_{old}}}{E} \left[D_{TV}(\rho_{new}||\rho_{old})[s]\right]
\end{aligned}
$$

According to the definition of the total variation distance:

$$
\begin{aligned}
&J(\rho_{new}) - J(\rho_{old}) \\
&\geq \frac{1}{1-\gamma} \mathop{E}_{\substack{s\sim D^{\Pi_{old}} \\ A\sim\rho_{new} \\ \boldsymbol{u}\sim\pi_{old}}} [\mathcal{A}_{\rho_{old},\pi_{old}}(s,A,\boldsymbol{u})] - \frac{2C^{\Pi}\gamma}{(1-\gamma)^2} \mathop{E}_{s\sim D^{\Pi_{old}}} \left[\frac{1}{2}\int_A |\rho_{new}(A|s) - \rho_{old}(A|s)|dA\right] \\
&= \frac{1}{1-\gamma} \mathop{E}_{\substack{s\sim D^{\Pi_{old}} \\ A\sim\rho_{new} \\ \boldsymbol{u}\sim\pi_{old}}} [\mathcal{A}_{\rho_{old},\pi_{old}}(s,A,\boldsymbol{u})] - \frac{C^{\Pi}\gamma}{(1-\gamma)^2} \mathop{E}_{s\sim D^{\Pi_{old}}} \left[\int_A \rho_{old}(A|s)\left|\frac{\rho_{new}(A|s)}{\rho_{old}(A|s)} - 1\right|dA\right] \\
&= \frac{1}{1-\gamma} \mathop{E}_{\substack{s\sim D^{\Pi_{old}} \\ A\sim\rho_{new} \\ \boldsymbol{u}\sim\pi_{old}}} [\mathcal{A}_{\rho_{old},\pi_{old}}(s,A,\boldsymbol{u})] - \frac{C^{\Pi}\gamma}{(1-\gamma)^2} \mathop{E}_{(s,A)\sim D^{\Pi_{old}}} \left[\left|\frac{\rho_{new}(A|s)}{\rho_{old}(A|s)} - 1\right|\right]
\end{aligned}
$$

Lemma 2. $\left\|D^{\Pi_{new}} - D^{\Pi_{old}}\right\|_1$ is defined by the average discrepancy between the joint policies Π_{new} and Π_{old}:

$$
\left\|D^{\Pi_{new}} - D^{\Pi_{old}}\right\|_1 \leq \frac{2\gamma}{1-\gamma} \mathop{E}_{s\sim D^{\Pi_{old}}} [D_{TV}(\rho_{new}||\rho_{old})[s]] \tag{72}
$$

where $\mathop{E}_{s\sim D^{\Pi_{old}}} [D_{TV}(\rho_{new}||\rho_{old})[s]] = (1/2)\sum_A |\rho_{new}(A|s) - \rho_{old}(A|s)|$.

According to Eq. (18) in CPO [15], we have:

$$
D^{\Pi} = (1-\gamma)(I - \lambda P_{\Pi})^{-1}\mu
$$

Define the following identities, $G_{new} \doteq (I - \gamma P_{\Pi_{new}})^{-1}, G_{old} \doteq (I - \gamma P_{\Pi_{old}})^{-1}$ and $\Delta = P_{\Pi_{new}} - P_{\Pi_{old}}$. According to Eq. (21) in CPO [15], we have:

$$
D^{\Pi_{new}} - D^{\Pi_{old}} = \gamma G_{old}\Delta D^{\Pi_{old}}
$$

Substituting the above into the equation, we obtain:

$$
\begin{aligned}
\left\|D^{\Pi_{new}} - D^{\Pi_{old}}\right\|_1 &= \gamma\left\|G_{old}\Delta D^{\Pi_{old}}\right\|_1 \\
&\leq \gamma\|G_{old}\|_1\left\|\Delta D^{\Pi_{old}}\right\|_1
\end{aligned} \tag{73}
$$

$\|G_{old}\|_1$ is bounded by:

$$
\|G_{old}\|_1 = \left\|(I - \gamma P_{\Pi_{old}})^{-1}\right\|_1 \leq \sum_{t=0}^{\infty}\gamma^t \left\|P_{\Pi_{old}}\right\|_1^t = (1-\gamma)^{-1} \tag{74}
$$

$\left\|\Delta D^{\Pi_{old}}\right\|_1$ is bounded by:

$$
\begin{aligned}
\left\|\Delta D^{\Pi_{old}}\right\|_1 &= \sum_{s_{new}}\left|\sum_{s_{old}}\Delta(s'|s)D^{\Pi_{old}}(s)\right| \\
&\leq \sum_{s,s'}|\Delta(s'|s)|D^{\Pi_{old}}(s) \\
&= \sum_{s,s'}\left|\sum_{u,A}p(s'|s,u,A)\pi_{old}(u|A,s)(\rho_{new}(A|s) - \rho_{old}(A|s))\right|D^{\Pi_{old}}(s) \\
&\leq \sum_{s,u,A,s'}p(s'|s,u,A)\pi_{old}(u|A,s)\,|\rho_{new}(A|s) - \rho_{old}(A|s)|D^{\Pi_{old}}(s) \\
&= \sum_{s,u,A}\pi_{old}(u|A,s)\,|\rho_{new}(A|s) - \rho_{old}(A|s)|D^{\Pi_{old}}(s) \\
&= \sum_{s,A}|\rho_{new}(A|s) - \rho_{old}(A|s)|D^{\Pi_{old}}(s) \\
&= 2\mathop{E}_{s\sim D^{\Pi_{old}}}[D_{TV}(\rho_{new}||\rho_{old})[s]]
\end{aligned} \tag{75}
$$

Substituting Eqs. (74) and (75) into Eq. (73), we obtain:

$$
\begin{aligned}
&\left\|D^{\Pi_{new}}-D^{\Pi_{old}}\right\|_1 \\
&\leq \gamma(1-\gamma)^{-1}\times 2\underset{s\sim D^{\Pi_{old}}}{E}\left[D_{TV}(\rho_{new}||\rho_{old})[s]\right] \\
&\leq \frac{2\gamma}{1-\gamma}\underset{s\sim D^{\Pi_{old}}}{E}\left[D_{TV}(\rho_{new}||\rho_{old})[s]\right]
\end{aligned}
$$

C.9 Proof of Proposition 3

The global function Q_{tot} can be decomposed into the sum of local value functions Q_j:

$$
\begin{aligned}
&\frac{\rho_{\varphi}\left(A_t \mid \boldsymbol{\tau}_t\right)}{\rho_{\varphi_{old}}\left(A_t \mid \boldsymbol{\tau}_t\right)}Q_{tot}\left(\boldsymbol{\tau}_t,\boldsymbol{u}_t,A_t\right) \\
&=\frac{\rho_{\varphi}\left(A_t \mid \boldsymbol{\tau}_t\right)}{\rho_{\varphi_{old}}\left(A_t \mid \boldsymbol{\tau}_t\right)}\sum_{j\in\mathcal{J}}Q_j\left(\boldsymbol{\tau}_t^j,\boldsymbol{u}_t^j,A_t\right) \\
&=\sum_{j\in\mathcal{J}}\frac{\rho_{\varphi}\left(A_t \mid \tau_t\right)}{\rho_{\varphi_{old}}\left(A_t \mid \tau_t\right)}Q_j\left(\boldsymbol{\tau}_t^j,\boldsymbol{u}_t^j,A_t\right)
\end{aligned}
$$

Because the value of each local value function Q_j is only related to the agent it is connected to, and has nothing to do with the connection relationship of other local value functions, so in the graph policy evaluated by Q_j, items unrelated to j can be ignored, that is, Q_j accurately evaluates the "sub-policy" of Q_j connected to the agent. . At the same time, the random variable corresponding to the "sub-policy" is $\rho(Q_j)\sim\tilde{P}_{D_{\max}}(D_{\max}:p_1,p_2,\ldots,p_N)$, then by Definition 1 available:

$$
\begin{aligned}
&\sum_{j\in\mathcal{J}}\frac{\rho_{\varphi}\left(A_t \mid \boldsymbol{\tau}_t\right)}{\rho_{\varphi_{old}}\left(A_t \mid \boldsymbol{\tau}_t\right)}Q_j\left(\boldsymbol{\tau}_t^j,\boldsymbol{u}_t^j,A_t\right) \\
&=\sum_{j\in\mathcal{J}}\frac{\underset{\{m_{j,1},\ldots,m_{j,N}\}\in H_m}{\sum}P_{\varphi}\{X_{j,1}=m_{j,1},\ldots,X_{j,N}=m_{j,N}\}}{\underset{\{m_{j,1},\ldots,m_{j,N}\}\in H_m}{\sum}P_{\varphi_{old}}\{X_{j,1}=m_{j,1},\ldots,X_{j,N}=m_{j,N}\}}Q_j\left(\boldsymbol{\tau}_t^j,\boldsymbol{u}_t^j,A_t\right) \\
&=\sum_{j\in\mathcal{J}}\frac{\left(\underset{\{m_{j,1},\ldots,m_{j,N}\}\in H_m}{\sum}\frac{C!}{m_{j,1}!\cdots m_{j,N}!}p_{j,1}^{m_{j,1}}\cdots p_{j,N}^{m_{j,N}}\right)_{\varphi}}{\left(\underset{\{m_{j,1},\ldots,m_{j,N}\}\in H_m}{\sum}\frac{C!}{m_{j,1}!\cdots m_{j,N}!}p_{j,1}^{m_{j,1}}\cdots p_{j,N}^{m_{j,N}}\right)_{\varphi_{old}}}Q_j\left(\boldsymbol{\tau}_t^j,\boldsymbol{u}_t^j,A_t\right)
\end{aligned}
$$

Since the graph policy uses importance sampling, there is $A_t\sim\mathcal{T}_{old}$, which means that sample the same graph structure under φ and φ_{old}, that is, $\left(m_{j,1}!\cdots m_{j,N}!\right)_{\varphi}=\left(m_{j,1}!\cdots m_{j,N}!\right)_{\varphi_{old}}$, then we can get:

$$
\begin{aligned}
&\sum_{j\in\mathcal{J}}\frac{\underset{\{m_{j,1},\ldots,m_{j,N}\}\in H_m}{\sum}P_{\varphi}\{X_{j,1}=m_{j,1},\ldots,X_{j,N}=m_{j,N}\}}{\underset{\{m_{j,1},\ldots,m_{j,N}\}\in H_m}{\sum}P_{\varphi_{old}}\{X_{j,1}=m_{j,1},\ldots,X_{j,N}=m_{j,N}\}}Q_j\left(\boldsymbol{\tau}_t^j,\boldsymbol{u}_t^j,A_t\right) \\
&=\sum_{j\in\mathcal{J}}\sum_{\{m_{j,1},\ldots,m_{j,N}\}\in H_m}\frac{\left(p_{j,1}^{m_{j,1}}\cdots p_{j,N}^{m_{j,N}}\right)_{\varphi}}{\left(p_{j,1}^{m_{j,1}}\cdots p_{j,N}^{m_{j,N}}\right)_{\varphi_{old}}}Q_j\left(\boldsymbol{\tau}_t^j,\boldsymbol{u}_t^j,A_t\right) \\
&=\sum_{j\in\mathcal{J}}\frac{P_{\varphi,j}\left(m_j\right)}{P_{\varphi_{old},j}\left(m_j\right)}Q_j\left(\boldsymbol{\tau}_t^j,\boldsymbol{u}_t^j,A_t\right)
\end{aligned}
$$

Finally got:

$$
\frac{\rho_{\varphi}\left(A_t \mid \boldsymbol{\tau}_t\right)}{\rho_{\varphi_{old}}\left(A_t \mid \boldsymbol{\tau}_t\right)}Q_{tot}\left(\boldsymbol{\tau}_t,\boldsymbol{u}_t,A_t\right)=\sum_{j\in\mathcal{J}}\frac{P_{\varphi,j}\left(m_j\right)}{P_{\varphi_{old},j}\left(m_j\right)}Q_j\left(\boldsymbol{\tau}_t^j,\boldsymbol{u}_t^j,A_t\right) \tag{76}
$$

APPENDIX D
SUPPLEMENTARY EXPERIMENTS

D.1 HO-Predator-Prey

The experimental environment is formulated as a complex 10×10 grid, wherein nine agents are tasked with the capture of six prey entities. Agents are endowed with the capacity to navigate in one of the four cardinal directions, remain stationary, or engage in a capture action. Actions deemed unfeasible, such as moving to an already occupied space or attempting a capture action outside the eight adjacent squares surrounding a prey, are systematically restricted. Prey exhibit random movement within the four cardinal directions and will

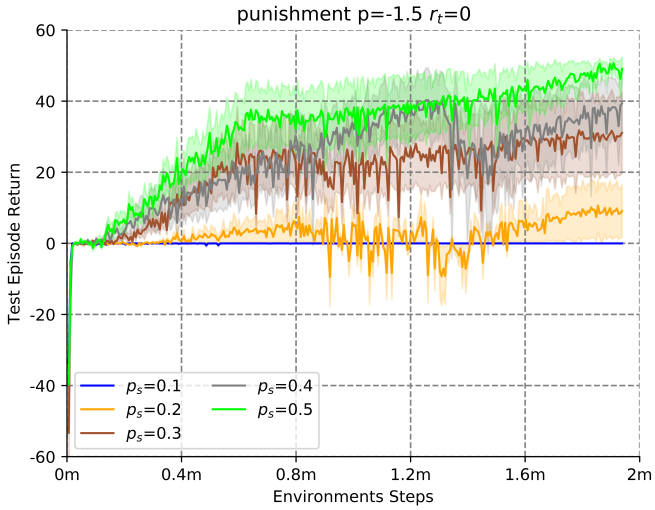

Fig. 2: Median test return for the Higher-order Predator-Prey task with different factor graph sparity $p_{sparity}$.

remain stationary should all adjacent positions be occupied. Upon successful capture, prey are removed from the grid, permitting agents to pursue subsequent targets. Each agent's perceptual field encompasses a 5×5 subgrid centered upon its location. Agents and prey that have been removed from play are rendered invisible within the simulation. An episode reaches its conclusion either when all prey have been captured or after the elapse of 200 time steps.

Hyper-parameters: The implementation of all algorithms adheres to the MARLBenchmark framework, with the source code accessible via https://github.com/SICC-Group/DDFG.

For all tasks, a discount factor $\gamma = 0.98$ is employed alongside ϵ-greedy exploration strategy. This strategy initiates with $\epsilon = 1$ and linearly decays to $\epsilon = 0.05$ over the first 50,000 time steps. Evaluation occurs every 2,000 steps through the execution of ten greedy test trajectories at $\epsilon = 0$. Each algorithm undergoes three independent trials, with the mean and standard error computed for each trial's outcomes.

A replay buffer, with a capacity for 5,000 entries, is maintained across all algorithms, alongside normalization of rewards contained within. A recurrent neural network (RNN) h_ψ processes agent observations across all methodologies. The Adam optimization algorithm is uniformly applied. For VDN, QMIX, MADDPG, QTRAN, and QPLEX, a learning rate of 7×10^{-4} is set, while CW-QMIX, OW-QMIX, DCG, SOPCG, and CASEC adopt a learning rate of 1×10^{-3}. Within DDFG, the Q-value and V-value function networks utilize a learning rate of 1×10^{-3}, whereas the graph policy network employs a learning rate of 1×10^{-5}. DCG incorporates a fully connected graph structure $\mathcal{E} := \{\{i, j\} \mid 1 \leq i < n, i < j \leq n\}$. The DDFG constrains the local value function's highest order to 3 and incorporates a distinct replay buffer, with a capacity of 8 entries, for the graph policy (referencing PPO), setting the constant $\lambda_{\mathcal{H}}$ at 0.01.

- **the selection of the number of factor nodes** m: In the experiment, the selection of m is related to the pre-set upper limit D of the factor graph order.

$$C_n^D * p_{sparity} \tag{77}$$

where C_n^{dD} is the combination number, n represents the number of agents, and $p_{sparity} \in (0, 1]$ represents the sparsity of the factor graph.
C_n^D represents the maximum number of combinations of selecting D agents from a factor graph with n agents and a maximum order of D.

To verify the impact of $p_{sparity}$ on the algorithm's performance, we conducted ablation experiments in the high-order predator-prey scenario (with $p = -1.5$, $r_t = 0$), and the results are shown in Figure 2. We can observe that as the $p_{sparity}$ increases, the algorithm's performance continuously improves. It is evident that the number of factor nodes still has a significant impact on the algorithm. As the sparsity increases, the frequency of communication between agents also increases, and at the same time, the probability of the factor nodes representing "optimal agent cooperation" in the graph policy at time t also increases. However, in scenarios with a larger number of agents, too high a sparsity can lead to a significant increase in computational overhead. Therefore, to balance computational cost and algorithm performance, we choose $p_{sparity} = 0.3$ in the Higher-order Predator-Prey and SMAC experiment.

D.2 SMAC

In our study, we delve into the intricacies of unit micromanagement tasks within the StarCraft II environment, a domain characterized by its strategic complexity and real-time decision-making demands. In these scenarios, enemy units are orchestrated by the game's built-in artificial intelligence, whereas allied units are governed by a reinforcement learning algorithm. The composition of both groups

TABLE 1: The StarCraft multi-Agent challenge benchmark

Map Name	Ally Units	Enemy Units
3s5z	3 Stalkers & 5 Zealots	3 Stalkers & 5 Zealots
$5m\_vs\_6m$	5 Marines	6 Marines
1c3s5z	9 Marines	9 Marines
$8m\_vs\_9m$	8 Marines	9 Marines
MMM2	1 Medivac, 2 Marauders & 7 Marines	1 Medivac, 2 Marauders & 8 Marines

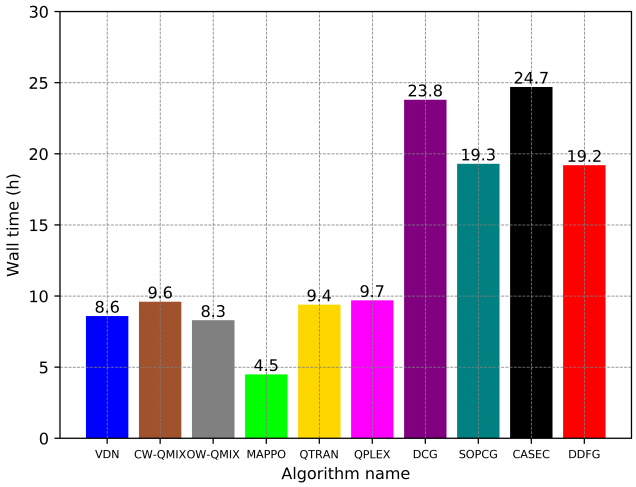

Fig. 3: In 3s5z map, the wall-time duration of DDFG and baselines.

may encompass a variety of soldier types, albeit restricted to a single race per faction. Agents are afforded a discrete set of actionable choices at each timestep, including inaction, directional movement, targeted attacks, and cessation of activity. These actions facilitate agent navigation and engagement on continuous maps. The reinforcement learning framework assigns a global reward to Multi-Agent Reinforcement Learning (MARL) agents, proportional to the cumulative damage inflicted on adversary units. Furthermore, the elimination of enemy units and overall combat victory are incentivized with additional reward bonuses of 10 and 200 points, respectively. A concise overview of the StarCraft Multi-Agent Challenge (SMAC) scenarios investigated in this study is provided in Table 1, incorporating the inclusion of particularly challenging maps: 1c3s5z (hard), $8m\_vs\_9m$(hard), and MMM2 (superhard).

Hyper-parameters: For all conducted SMAC experiments, version 2.4.10 of StarCraft II was utilized. Experimental setups consistently applied a discount factor (γ) of 0.97 and an ϵ-greedy exploration strategy, which linearly decreased from $\epsilon = 1$ to $\epsilon = 0.05$ across the initial 50,000 timesteps. Evaluations, comprising ten greedy test trajectories, were systematically conducted every 10,000 steps at $\epsilon = 0$. Each algorithm was subjected to three distinct initialization seeds, facilitating the computation of mean performance and standard error metrics.

All implemented algorithms employed a replay buffer capacity of 5,000 entries, with rewards within this buffer normalized for consistency. Observational data from agents were processed through a recurrent neural network (RNN) denoted as h_ψ, with the Adam optimization algorithm uniformly applied across all models. The learning rates for VDN, QMIX, CW-QMIX, OW-QMIX, MADDPG, QTRAN, DCG, SOPCG, and CASEC were set at 1×10^{-3}, while QPLEX adopted a slightly reduced learning rate of 1×10^{-4}. Within the Dynamic Deep Factor Graph (DDFG) framework, both the Q-value and V-value function networks (as outlined in section III-A of the main text and Appendix B, respectively) utilized a learning rate of 1×10^{-3}, with the graph policy network assigned a learning rate of 1×10^{-6}. The DDFG model imposes a limitation on the highest order of local value function to 2 and incorporates a dedicated replay buffer of eight entries for the graph policy (referencing Proximal Policy Optimization, PPO). Additionally, a constant $\lambda_{\mathcal{H}}$ value of 0.01 was established.

- **the wall-time duration of DDFG and baselines**: We believe that comparing the wall-time duration between DDFG and baseline algorithms can more comprehensively evaluate the practical performance of the methods.

The experimental results are shown in Figure 3. We conducted experiments using the 3s5z map in SMAC. The experimental results show that MAPPO takes the least time, while the wall-time of graph-based value decomposition algorithms is greater than that of other value decomposition algorithms. Graph-based value decomposition algorithms require message passing processes on the graph, and some of them even need additional training to obtain a good graph structure. Therefore, their time consumption is greater than that of other IGM-based value decomposition algorithms. However, DDFG does not exhibit additional time overhead while still achieving good performance. As for MAPPO, due to its on-policy algorithm structure, it can perform parallel training in the RNN version, which greatly reduces its time consumption. Hence, its wall-time is much smaller than that of all other off-policy algorithms.

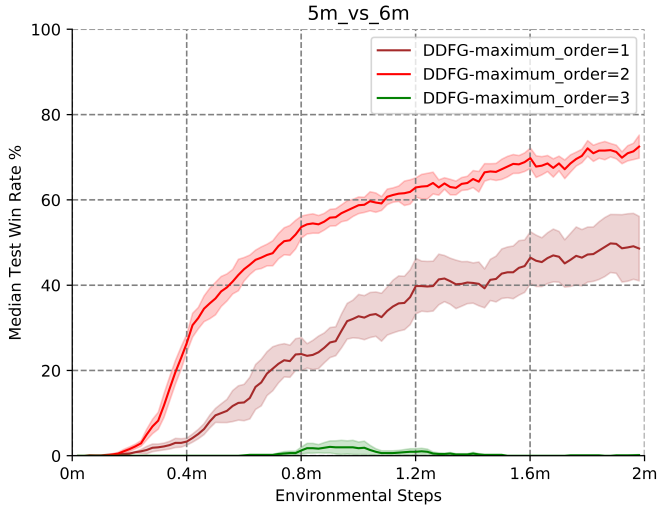

Fig. 4: In $5m\_vs\_6m$ map, the Median test win rate $\%$ of DDFG under different D.

- **The selection of the maximum order** D: In Section 3, Eq. (78) reveals the relationship between the error upper bound and the maximum order D of the value function decomposition, which highlights the trade-off between computational cost and model accuracy. As D increases, the complexity of the model increases (with higher computational cost), while the approximation error decreases. Therefore, selecting an appropriate D can effectively improve the performance of the algorithm.

$$\begin{aligned} &\mathcal{E}(\widehat{Q_{tot}^{D}}) - \mathcal{E}(Q_{tot}^{\star}) \\ &\leq C_{final1} \cdot \frac{(B_{w,2})^{D} - 1}{B_{w,2} - 1} - C_{final2} \cdot D + C_{final3} \end{aligned} \tag{78}$$

where $C_{final1} = 4C_Q B_{\lambda,2} B_{h,2} B_{w,2} \sqrt{\frac{C_K}{n_s}}$, $C_{final2} = \frac{C_1}{C_2} \exp\left(-(C_K+1)^{\gamma}\right)$, $C_{final3} = \left(2(C_Q)^2 + C_Q\right) \sqrt{\frac{2\log(4/\delta)}{n_s}} + \frac{C_1}{C_2} \exp\left(-(C_K+1)^{\gamma}\right)(n-1)$.

We select the $5m\_vs\_6m$ map in SMAC as an example. We set D to $1-3$ and conducted experiments respectively, with the results shown in Figure 4. The results show that DDFG achieves the best performance when $D=2$; its performance partially degrades when $D=1$, and it has almost no winning rate when $D=3$. It can be seen from the results that an appropriate D has a significant impact on the algorithm's performance. When $D=1$, the algorithm is equivalent to VDN, and there is no collaboration among agents, thus degrading the algorithm's performance. When $D=3$, the 3rd-order factor graph introduces excessive model complexity, causing the algorithm to fail to converge. Ultimately,$D=2$ is the optimal choice in SMAC.

- **The impact of communication interruption on the algorithm's performance**: In DDFG, the graph policy network generates dynamic factor graph structures in real-time, which serves as the input to the max-sum algorithm. This means that agents need to communicate at each time step, and when facing limited communication bandwidth or communication-constrained environments, the performance of the algorithm will be challenged.

To assess the impact of communication constraints, we use the `SMAC` map $5m\_vs\_6m$ and conduct two ablation studies; results are shown in Fig. 5. We define the *communication interruption rate (edge)* p_{edge} as the probability that communication between any two agents is interrupted at time t (i.e., the corresponding edge is absent in the factor graph). In contrast, the *communication interruption rate (graph)* p_{graph} is the probability that, at time t, all inter-agent communication is cut off (i.e., the adjacency matrix collapses to the identity).

From Fig. 5, when either interruption rate lies in the range 0–30$\%$, the algorithm's performance is largely unaffected. This robustness arises because `DDFG`'s dynamic graph makes fluctuations in p_{edge} difficult to accumulate, and occasional full-graph outages with $p_{\text{graph}} \in [0, 30]\%$ are handled by the fully decomposed fallback, maintaining a high win rate. However, as the interruption rate increases to 40–90$\%$, performance degrades progressively: agents execute more independently, coordination via communication diminishes, and `DDFG` gradually approaches the behavior of `VDN`.

- **The generalization capability of the graph policy**. To better demonstrate the advantages of the Graph Generation Policy, we explore its generalization capability in SMAC.

Under the original training protocol and budgets, we adopt a "train on Map A $\rightarrow$ **freeze graph policy** $\rightarrow$ train Q on Map B" setup to isolate the effect of the graph policy. In the first transfer, the graph policy is trained on *5m_vs_6m* and then frozen; the frozen graph policy is reused while training the Q-function on the *2s3z*. In the second transfer, the graph policy is trained on *8m_vs_9m*, frozen, and reused on *3s5z*. Throughout, absolute map cues are removed and inputs are expressed as relative, normalized features to avoid map-specific leakage and to strengthen internal validity.

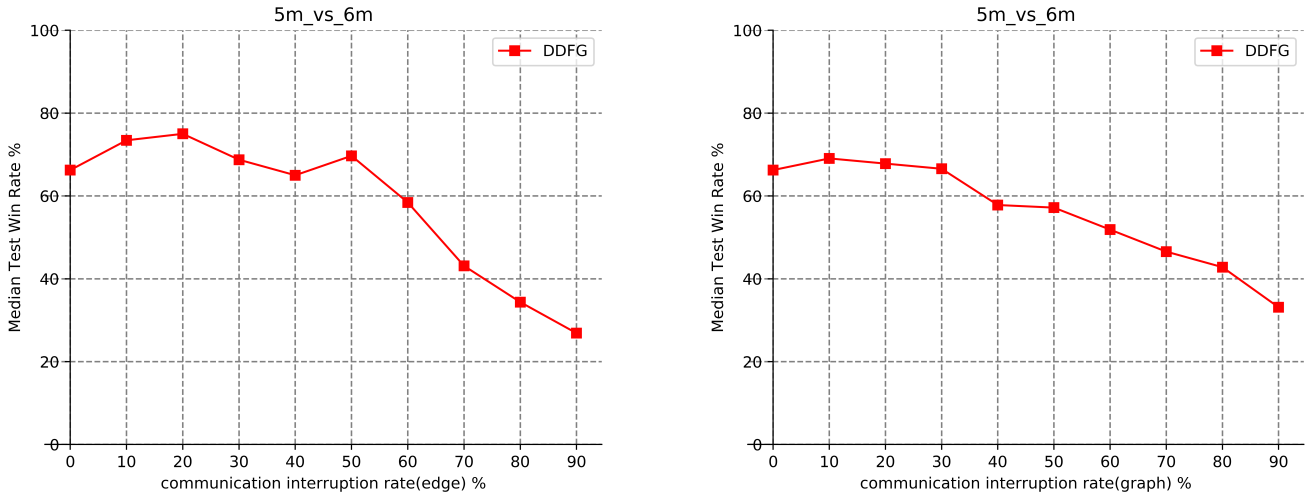

Fig. 5: Under the $5m\_vs\_6m$ map, the impact of the communication interruption on the algorithm's performance.

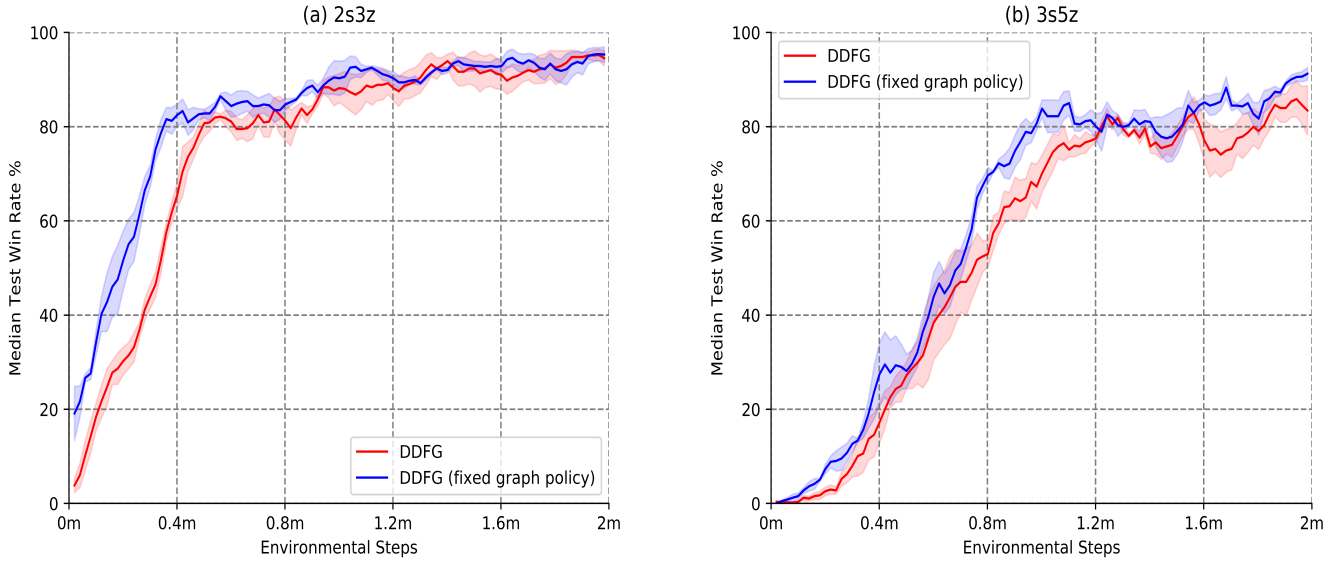

Fig. 6: The ablation experiment of fixed graph policy in SMAC environments.

Across both transfers, the **frozen** graph policy yields **consistent gains in sample efficiency and final win rate** over strong baselines without a learned graph structure, and it converges faster under identical training budgets. The corresponding learning curves and final win rates are shown in Fig. 6. Taken together, these SMAC results provide an internally valid assessment that the learned coordination structures *transfer across tasks*, supporting the effectiveness of the proposed graph generation mechanism.

APPENDIX E
CODE DEPLOYMENT AND UTILIZATION

E.1 Setup

Here we briefly describe the setup of DDFG. More information is given in the README file of the repository.

1) Install the dependencies.
   Here we give an example installation on CUDA == 10.1. For non-GPU and other CUDA version installation, please refer to the PyTorch website.
   $ conda create -n marl python==3.6.1
   $ conda activate marl
   $ pip install torch==1.5.1+cu101 torchvision==0.6.1+cu101
   -f https://download.pytorch.org/whl/torch_stable.html
   $ pip install -r requirements.txt
2) Obtain the source code and install.

```
$   git clone https://github.com/SICC-Group/DDFG.git
$   cd off-policy
$   pip install -e .
```

3) Install StarCraftII 4.10

```
$   pip install git+https://github.com/oxwhirl/smac.git
$   echo "export SC2PATH= /StarCraftII/" >  /.bashrc
# Download SMAC Maps, and move it to  /StarCraftII/Maps/
```

E.2 Reproduction

1) Train of DDFG

```
$   cd offpolicy/scripts
$   chmod +x ./train_<experiment>_<algorithm>.sh
$   .train_<experiment>_<algorithm>.sh
# <experiment> can choose between prey or smac, <algorithm> can choose between rmdfg or other baselines.
# The trained model and data files can be found under ./results/<experiment_name>/<algorithm_name>/debug/run1/models/.
# <experiment_name> can choose between Predator_prey or StarCraft2, <algorithm_name> can choose between rddfg_cent_rw or other baselines.
```

2) Test of DDFG

```
# Select the trained model to load and test
$   .train_<experiment>_<algorithm>.sh
   –model_dir ./results/<experiment_name>/<algorithm_name>/debug/run1/models/
```

3) Replot the experimental results of this manuscript

```
$   cd off-policy/experiment_eval_data
# Execute file plot_experiment_pp.py or plot_experiment_smac.py to obtain results.
```

REFERENCES

[1] L. Kraemer, B. Banerjee, Multi-agent reinforcement learning as a rehearsal for decentralized planning, Neurocomputing 190 (2016) 82–94.
[2] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, Y. Yi, Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning, in: International conference on machine learning, PMLR, 2019, pp. 5887–5896.
[3] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, N. Freitas, Dueling network architectures for deep reinforcement learning, in: International conference on machine learning, PMLR, 2016, pp. 1995–2003.
[4] T. Wang, T. Gupta, A. Mahajan, B. Peng, S. Whiteson, C. Zhang, Rode: Learning roles to decompose multi-agent tasks, in: International Conference on Learning Representations, 2020.
[5] J. Pearl, Probabilistic reasoning in intelligent systems: networks of plausible inference, Elsevier, 2014.
[6] G. GemanS, gibbsdistri—butions, andthe bayesian restoration ofimages, IEEE TransPatternAnal& MachIntell 6 (1984) 721–741.
[7] J. S. Yedidia, W. T. Freeman, Y. Weiss, et al., Understanding belief propagation and its generalizations, Exploring artificial intelligence in the new millennium 8 (236–239) (2003) 0018–9448.
[8] F. R. Kschischang, B. J. Frey, H.-A. Loeliger, Factor graphs and the sum-product algorithm, IEEE Transactions on information theory 47 (2) (2001) 498–519.
[9] H.-A. Loeliger, An introduction to factor graphs, IEEE Signal Processing Magazine 21 (1) (2004) 28–41.
[10] J. S. Yedidia, Message-passing algorithms for inference and optimization: “belief propagation” and “divide and concur”, Journal of Statistical Physics 145 (2011) 860–890.
[11] S. M. Aji, R. J. McEliece, The generalized distributive law, IEEE transactions on Information Theory 46 (2) (2000) 325–343.
[12] P. L. Bartlett, S. Mendelson, Rademacher and gaussian complexities: Risk bounds and structural results, Journal of Machine Learning Research 3 (Nov) (2002) 463–482.
[13] B. Bercu, B. Delyon, E. Rio, Concentration inequalities for sums and martingales, Springer, 2015.
[14] Z. Yang, C. Jin, Z. Wang, M. Wang, M. I. Jordan, On function approximation in reinforcement learning: Optimism in the face of large state spaces, arXiv preprint arXiv:2011.04622.
[15] J. Achiam, D. Held, A. Tamar, P. Abbeel, Constrained policy optimization, in: International conference on machine learning, PMLR, 2017, pp. 22–31.
[16] J. Queeney, Y. Paschalidis, C. G. Cassandras, Generalized proximal policy optimization with sample reuse, Advances in Neural Information Processing Systems 34 (2021) 11909–11919.