Graph Convolutional Value Decomposition in Multi-Agent Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

We propose a novel framework for value function factorization in multi-agent deep reinforcement learning using graph neural networks (GNNs). In particular, we consider the team of agents as the set of nodes of a complete directed graph, whose edge weights are governed by an attention mechanism. Building upon this underlying graph, we introduce a mixing GNN module, which is responsible for two tasks: i) factorizing the team state-action value function into individual per-agent observation-action value functions, and ii) explicit credit assignment to each agent in terms of fractions of the global team reward. Our approach, which we call GraphMIX, follows the centralized training and decentralized execution paradigm, enabling the agents to make their decisions independently once training is completed. Experimental results on the StarCraft II multi-agent challenge (SMAC) environment demonstrate the superiority of our proposed approach as compared to the state-of-the-art.

1 Introduction

Multi-agent systems are ubiquitous, appearing in many application areas such as autonomous driving (Zhao et al., 2019; Chu et al., 2020), drone swarms (Zanol et al., 2019), communication systems (Naderializadeh et al., 2020), multi-robot search and rescue (Malaschuk & Dyumin, 2020), and smart grid (Xie et al., 2019). In many of the aforementioned domains, instructive feedback is not available, as there are no ground-truth solutions or decisions available. These phenomena have given rise to a plethora of literature on multi-agent reinforcement learning (MARL) algorithms, with a special focus on deep-learning-driven methods over the past few years.

More recently, algorithms with centralized training and decentralized execution have gained interest, due to their applicability in practical real-world scenarios. In (Lowe et al., 2017; Foerster et al., 2018), policy gradient algorithms are considered, where the actors, which are responsible for taking the actions for each agent, are decentralized, while the critic is assumed to be centralized, trained in conjunction with the actors in an end-to-end manner over the course of training. The authors in (Sunehag et al., 2018; Rashid et al., 2018) take a different value-based approach to train the agents. Specifically, they have a *value factorization* module (linear in the case of VDN (Sunehag et al., 2018) and a state-based non-linear multi-layer peceptron (MLP) in the case of QMIX (Rashid et al., 2018)), which is responsible for implicit *credit assignment*; i.e., how to decompose the global state-action value function to individual observation-action value functions for different agents.

One of the main drawbacks of the aforementioned algorithms is that they do not explicitly capture the underlying structure of the team of agents in the environment, which can be modeled using a graph topology. There have been some attempts to connect MARL with graph representation learning methods in the literature. As an example, Jiang et al. (2020) propose a MARL algorithm based on the graph convolutional network (GCN) architecture (Kipf & Welling, 2016). However, it needs both centralized training and centralized execution (or at the very least, the agents need to communicate with each other multiple times during the inference phase), and therefore, it does not allow for decentralized decision making by the agents.

In this paper, we propose an algorithm for centralized training of MARL agents based on graph neural networks (GNNs) that enables them to be deployed in a distributed manner during execution. In particular, we consider a directed weighted graph, where each node represents an agent, and there is a directed edge between any pair of nodes. We use an attention mechanism to dynamically adjust the weights of the edges in this graph based on the agents' observations. Leveraging such a graph structure, we propose to use a mixing GNN that produces a global state-action value function at its output given the individual agents' observation-action value functions at the input. A monotonicity constraint is enforced on the GNN, ensuring that the individual agent decisions are consistent with the case where a central entity would be responsible for making the decisions for all the agents.

We also use the mixing GNN as a backbone to derive an effective fraction of global team reward for each of the agents. We use these reward fractions to minimize per-agent local losses, alongside the global loss using the global state-action value function. The mixing GNN, the attention mechanism, and the agent parameters are all trained centrally in an end-to-end fashion, and after training is completed, each agent can make its decisions in a decentralized manner. We evaluate our proposed algorithm, which we refer to as GraphMIX, on the StarCraft II multi-agent challenge (SMAC) environment (Samvelyan et al., 2019) and show that it is able to outperform the state-of-the-art QMIX algorithm (Rashid et al., 2018) across different maps.

2 Related Work

A growing focus in the recent deep reinforcement learning literature is on multi-agent cooperation (Lowe et al., 2017; Foerster et al., 2018; Sunehag et al., 2018; Rashid et al., 2018; Papoudakis et al., 2020). Methods exist on a spectrum from a single unified or centralized agent to independent and decentralized agents. At one extreme end of this spectrum, a multi-agent reinforcement learning (MARL) problem might be reduced to a standard deep reinforcement learning problem, with a single centralized network returning a joint action vector for all agents. At this extreme, issues that arise are general motivations for the development of MARL. Joint multivariate action and state spaces exponentially increase in size with the number of agents. This creates difficulties in generalization to different numbers of agents, parameter memory scalability, and training sample efficiency. At the other extreme, independent and decentralized agents face difficulty as coordination becomes more complex. A recent trend that aims to find an effective middle ground is centralized training and decentralized execution. These methods aim to produce decentralized controllers, and enforce implicit coordination with a centralized measure of value used only in training. A related trend which skews more toward a centralized agent studies the impact of communication between agents during execution (Foerster et al., 2016; Sukhbaatar et al., 2016).

Graph neural networks (GNNs), on the other hand, have gained popularity as a prominent method to manage structured input data and incorporate neighborhood information (Scarselli et al., 2008; Kipf & Welling, 2016; Zhou et al., 2018). Similarly to convolutional and recurrent neural networks, GNNs formalize structured treatment of data that would otherwise be concatenated and treated purely as vectors in a high dimensional space. Due to the flexibility in modelling structured data, GNNs have seen widespread application, in areas such as knowledge representation (Park et al., 2019), natural language processing (Ji et al., 2019), social network analysis (Fan et al., 2019), wireless communications (Eisen & Ribeiro, 2020), chemistry (Hu* et al., 2020), and physics (Ju et al., 2020).

In MARL, GNN-based architectures have recently been used to improve sample efficiency by adding invariance to permutation of inputs from multiple agents in multi-agent critics (Liu et al., 2020) and emphasize observations of neighborhoods in individual agent controllers (Jiang et al., 2020). Graph structure has also been tied into neural attention modules, especially when using attention mechanisms to compute graph edge weights (Veličković et al., 2017; Thekumparampil et al., 2018). These mechanisms gained popularity in sentence translation tasks for handling associations between structured data components (Vaswani et al., 2017; Devlin et al., 2018), and they have seen use in general reinforcement learning as well (Zambaldi et al., 2018; Baker et al., 2019; Iqbal & Sha, 2019).

Most related to our work is a branch of value-based MARL methods, which decompose a joint state-action value function to allow individual agents to be trained from a single global reward. VDN (Sunehag et al., 2018) initially approximated a joint state-action value function over all agents' actions as the sum of individual observation-action value functions from each agent. QMIX (Rashid et al., 2018) observes that the joint state-action value function can more generally be represented by a monotonic function of individual observation-action value functions. Additionally, QMIX allows the joint state-action value function to be informed by global information, which is potentially unavailable to the individual observation-action value functions. The specifics of this factorization continue to be analyzed in works such as Son et al. (2019). In this work, we show how to re-visit the analysis of the joint state-action value function to reflect graphical structure from the multi-agent setting.

3 System Model

We consider a multi-agent environment, where a team of M agents collaborate with each other to solve a cooperative task. In particular, we consider a partially-observable Markov decision process (POMDP), represented by a tuple $< M, \mathcal{S}, O, \mathcal{Z}, \mathcal{A}, T, R_g, \gamma >$. At each time step t, the environment is in global state $s(t) \in \mathcal{S}$, with $\mathcal{S}$ denoting the global state space, and each agent $m \in \{1, \ldots, M\}$ receives as observation $o_m(t) = O(s_t, m) \in \mathcal{Z}$, where $\mathcal{Z}$ denotes the per-agent observation space, and $O : \mathcal{S} \times \{1, ..., M\} \rightarrow \mathcal{Z}$ denotes the per-agent observation function.

Upon receiving its observation, each agent $m \in \{1, \ldots, M\}$ takes an action $a_m(t) \in \mathcal{A}$, with $\mathcal{A}$ denoting the per-agent action space. These actions will cause the environment to transition to the next state $s(t+1) \sim T\left(s' | s(t), \{a_m(t)\}_{m=1}^M\right)$, with $T : \mathcal{S} \times \mathcal{S} \times \mathcal{A}^M \rightarrow [0, 1]$ denoting the state transition function. This transition is accompanied with a global reward $r_g(t) = R_g\left(s(t), \{a_m(t)\}_{m=1}^M\right)$, where $R_g : \mathcal{S} \times \mathcal{A}^M \rightarrow \mathbb{R}$ denotes the global reward function.

In this setting, the goal of the agents at each time step t is to take a joint set of actions $\{a_m(t)\}_{m=1}^M$ so as to maximize the *discounted cumulative global reward*, defined as $\sum_{t'=t}^{\infty} \gamma^{t-t'} r_g(t')$. In order to make such decisions, each agent $m \in \{1, \ldots, M\}$ is equipped with a policy $\pi_m : \mathcal{A} \times (\mathcal{Z} \times \mathcal{A})^* \rightarrow [0, 1]$ that determines its action given its observation-action history, where $(\mathcal{Z} \times \mathcal{A})^*$ denotes the set of all possible observation-action histories. In particular, the action of agent m at time step t is distributed as $a_m(t) \sim \pi_m\left(a | \tau_m(t)\right)$, where $\tau_m(t) = \left(\{o_m(t')\}_{t'=1}^t, \{a_m(t')\}_{t'=1}^{t-1}\right)$ denotes the set of current and past local observations, as well as past actions of agent m at time step t. Letting π denote the set of policies of all M agents, its induced joint state-action value function is defined as

$$Q^\pi\left(s(t), \{a_m(t)\}_{m=1}^M\right) = \mathbb{E}\left[\sum_{t'=t}^{\infty} \gamma^{t-t'} r_g(t')\right], \tag{1}$$

where the expectation is taken with respect to the set of future states and actions.

4 GraphMIX: Graph-Based Value Function Factorization

We assume that each agent $m \in \{1, \ldots, M\}$ is equipped with a deep recurrent Q-network (DRQN), coupled with a policy π_m. At each time step t, the agent chooses its action in a decentralized manner based on its current and past local observations, alongside its past actions. In particular, it will choose an action $a_m(t)$ according to the distribution $\pi_m\left(a | \tau_m(t)\right)$. This will lead to its local observation-action value function $Q^{\pi_m}(\tau_m(t), a_m(t))$.

Next, we model the team of agents as a (directed) graph, denoted by $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the set of M graph nodes, each of which corresponds to an agent. Due to this one-to-one correspondence, hereafter in this paper, we use "agent" and "node" interchangeably. Moreover, $\mathcal{E} = \mathcal{V} \times \mathcal{V}$ denotes the set of M^2 graph edges, implying that the graph is a complete graph. For two agents $u, v \in \mathcal{V}$, we let w_{uv} denote the weight of the edge from node v to node u. These edge weights can possibly be varying over time, and we will later describe how they are determined.

To train the agents, we borrow the notion of monotonic value function factorization from Rashid et al. (2018), where the idea is to decompose the global state-action value function $Q^\pi\left(s(t), \{a_m(t)\}_{m=1}^M\right)$ into a set of local observation-action value functions $\{Q^{\pi_m}(\tau_m(t), a_m(t))\}_{m=1}^M$ such that an increase in the local observation-action value function of each agent leads to a corresponding increase in the global state-action value function. To be precise, the decomposition can be written as

$$Q^\pi\left(s(t), \{a_m(t)\}_{m=1}^M\right) = \Psi_{\mathsf{mix}}\Big(Q^{\pi_1}(\tau_1(t), a_1(t)), \ldots, Q^{\pi_M}(\tau_M(t), a_M(t))\Big), \tag{2}$$

where the *mixing function* Ψ_{mix} satisfies

$$\frac{\partial \Psi_{\mathsf{mix}}(x_1, ..., x_M)}{\partial x_m} \geq 0, \forall m \in \{1, \ldots, M\}. \tag{3}$$

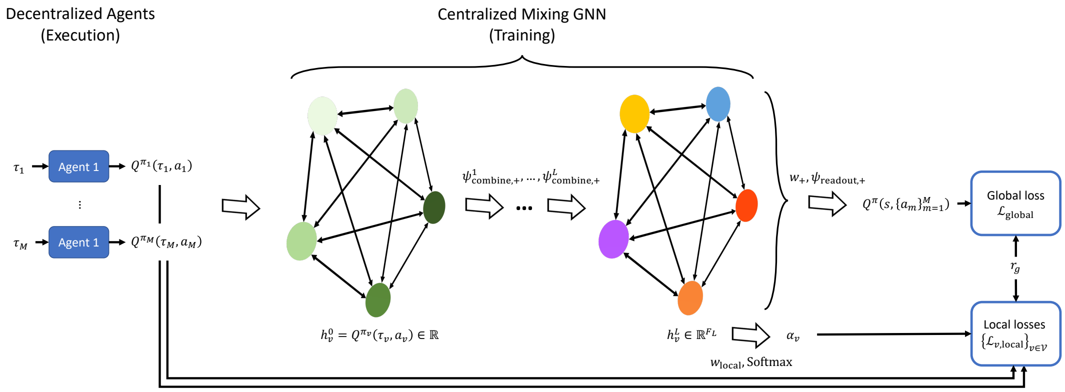


Figure 1: The GraphMIX architecture, comprising individual agents that make their decisions in a decentralized manner, alongside a mixing GNN that is used for centralized training of the agents.

The monotonicity condition in equation 3 ensures that if each agent takes the action that maximizes its local observation-action value function, it would also be the best action for the entire team.

In this work, we propose to use a graph-based approach for combining the local per-agent observation-action value functions into the global state-action value function. In particular, we leverage the aforementioned graph of the agents G to define a *mixing GNN* architecture, as shown in Figure 1. In this GNN, each node $v \in \mathcal{V}$ starts with a scalar feature, which is the corresponding agent's local observation-action value function, i.e.,

$$h_v^0 = Q^{\pi_v}(\tau_v, a_v), \tag{4}$$

where we have dropped the dependence on time for brevity. The features are then passed through one or multiple hidden layer(s), whose number is denoted by L. At the l^{th} layer, $l \in \{1, \ldots, L\}$, the features of each node $v \in \mathcal{V}$ are updated as

$$h_v^l = \psi^l_{\text{combine},+}\left(h_v^{l-1}, \{h_u^{l-1}\}_{u\in\mathcal{V}\setminus\{v\}}, \{w_{uv}\}_{u\in\mathcal{V}\setminus\{v\}}\right), \tag{5}$$

where $\psi^l_{\text{combine},+}(\cdot)$ denotes a monotonically-increasing (and potentially non-linear) combining function. This implies that each node uses its own features and the other agents' features, alongside its outgoing edge weights to map its input feature (vector) of dimension F_{l-1} to an output (vector) of dimension F_l, with $F_0 = 1$. Note that the function $\psi^l_{\text{combine},+}(\cdot)$ is in general a parametrized and differentiable function, whose parameters are trained in an end-to-end fashion.

At the output of the L^{th} layer, each node $v \in \mathcal{V}$ will end up with a feature vector $h_v^L \in \mathbb{R}^{F_L}$. We then define the global state-action value function as

$$Q^\pi\left(s, \{a_m\}_{m=1}^M\right) = w_+^T\, \psi_{\text{readout}}(\{h_v^L\}_{v\in\mathcal{V}}), \tag{6}$$

where $\psi_{\text{readout}} : \overbrace{\mathbb{R}^{F_L} \times \cdots \times \mathbb{R}^{F_L}}^{M} \to \mathbb{R}^{F_L}$ is a graph readout operation (such as average/max pooling), and $w_+ \in \mathbb{R}_+^{F_L}$ is a non-negative parameter vector that maps the graph embedding $\psi_{\text{readout}}(\{h_v^L\}_{v\in\mathcal{V}})$ into the global state-action value function. Note that the monotonicity of $\psi^l_{\text{combine},+}(\cdot)$ in equation 5 and the non-negativity of w_+ in equation 6 guarantee that the mixing monotonicity condition in equation 3 is satisfied.

As an additional component, we introduce another weight vector $w_{\text{local}} \in \mathbb{R}^{F_L}$ that maps the output feature vector of each node to an *effective reward fraction* for the corresponding agent, defined as

$$\alpha_v = \text{Softmax}_{\mathcal{V}}\left(w_{\text{local}}^T\, h_v^L\right) = \frac{\exp\left(w_{\text{local}}^T\, h_v^L\right)}{\sum_{u\in\mathcal{V}} \exp\left(w_{\text{local}}^T\, h_u^L\right)}. \tag{7}$$

We interpret these values as the effective fraction of the global reward that each agent receives at each time step. The significance of w_{local} lies in the fact that its parameters do not need to be non-negative, which can improve the expressive power of the mixing GNN beyond monotonic functions.

Our proposed architecture, which we refer to as GraphMIX, is trained in an end-to-end fashion by minimizing the aggregate loss

$$\mathcal{L} = \mathcal{L}_{\text{global}} + \sum_{v \in \mathcal{V}} \mathcal{L}_{v,\text{local}}, \tag{8}$$

with the global loss defined as

$$\mathcal{L}_{\text{global}} = \sum_{i \in \mathcal{B}} \left[\left(\left(r_g + \gamma \max_{a'_1,\dots,a'_M} Q^{\pi}\left(s', \{a'_m\}_{m=1}^{M}\right) \right) - Q^{\pi}\left(s, \{a_m\}_{m=1}^{M}\right) \right)^2 \right]_i, \tag{9}$$

where $\mathcal{B}$ denotes a batch of transitions that are sampled from the experience buffer at each round of training, and s' and $\{a'_m\}_{m=1}^{M}$ respectively correspond to the environment state and agents' actions in the following time step. Moreover, for each node $v \in \mathcal{V}$, the local loss is defined as

$$\mathcal{L}_{v,\text{local}} = \sum_{i \in \mathcal{B}} \left[\left(\left(\alpha_v r_g + \gamma \max_{a'_v} Q^{\pi_v}(\tau'_v, a'_v) \right) - Q^{\pi_v}(\tau_v, a_v) \right)^2 \right]_i, \tag{10}$$

where τ'_v denotes the observation-action history of the agent corresponding to node v at the following time step. Note how minimizing the local losses in equation 10 creates a shortcut for backpropagating gradients to the individual agent networks, compared to the alternative path through the mixing GNN by minimizing the global loss in equation 9. Moreover, such local updates of the agent networks maintain the consistency of the global policy and the decentralized agent policies.

4.1 Attention-Based Edge Weights

To define the edge weights of the graph G, i.e., $\{w_{uv}\}_{(u,v) \in \mathcal{E}}$, we use an attention mechanism similar to the one proposed by Li et al. (2020). In particular, the agent observations at each time step are first encoded using a shared encoder mechanism $\phi : \mathcal{Z} \to \mathbb{R}^{F'}$ to an F'-dimensional embedding. Then, for each pair of nodes $u, v \in \mathcal{V}$, the weight of the edge from node v to node u is defined as

$$w_{uv} = \mathsf{Softmax}\Big(\phi(o_u)^T W \phi(o_v)\Big) = \frac{\exp\left(\phi(o_u)^T W \phi(o_v)\right)}{\sum_{u' \in \mathcal{V}} \exp\left(\phi(o_{u'})^T W \phi(o_v)\right)}, \tag{11}$$

where $W \in \mathbb{R}^{F' \times F'}$ denotes the attention weight matrix, whose parameters are trained in an end-to-end fashion alongside the agent and mixing GNN parameters. The softmax operation in equation 11 ensures that the weights of the *outgoing* edges from each node to the other nodes sum up to unity.

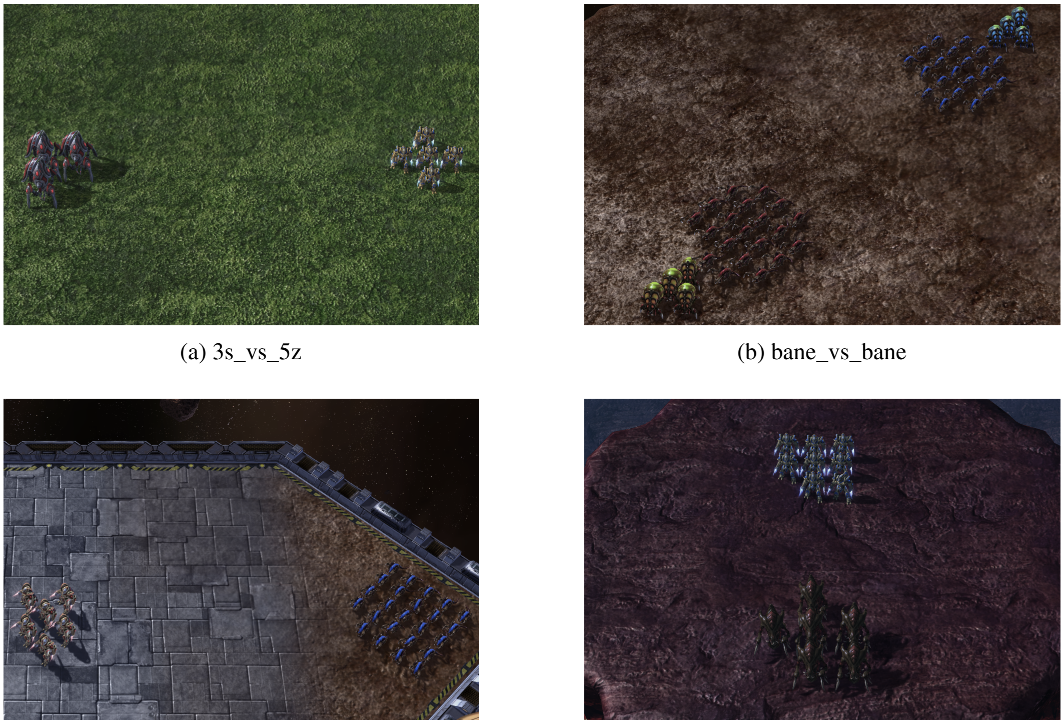
4.2 Isolating Dead Agents

Over the course of an episode, the agents in the team might get killed, for example by carelessly approaching the enemies in the opponent team. Because of that, we isolate the dead agents from the other nodes in graph G, and we remove them from calculations of the attention weights and the GNN operations entirely. Specifically, the readout operation in equation 6, the effective reward fraction calculation in equation 7, and the softmax operation in equation 11 for the edge weights are constrained to the agents that are still alive in the corresponding time steps.

5 Experimental Results

We evaluate GraphMIX on the StarCraft II multi-agent challenge (SMAC) environment (Samvelyan et al., 2019), which provides a set of different micromanagement challenges for benchmarking distributed multi-agent reinforcement learning methods. We specifically consider a set of four maps, namely 3s_vs_5z, bane_vs_bane, corridor and 6h_vs_8z. These maps have been classified as either hard or super-hard by Rashid et al. (2018). In each map, the *allied* team of agents are controlled by the MARL policy, while the enemy units are controlled by the game's built-in AI. Table 1 provides an overview of these maps in terms of team sizes, unit types, and the micromanagement skills to be learned by the allied agents. Moreover, Figure 2 shows a screenshot of each of the maps.

We use a gated recurrent unit (GRU) for each of the decentralized agents with 64 hidden units. Each agent uses an ϵ-greedy policy, where the probability of random actions decays from 100% to



(a) 3s_vs_5z (b) bane_vs_bane

(c) corridor (d) 6h_vs_8z

Figure 2: Screenshots of the four considered SMAC maps at the beginning of each episode.

Table 1: List of the considered SMAC maps and their corresponding features and required skills that the allied agents need to learn in order to succeed against the enemy units.

Map name	Allied agents	Enemy units	Micro-trick
3s_vs_5z	3 Stalkers	5 Zealots	Kiting
bane_vs_bane	20 Zerglings and 4 Banelings	20 Zerglings and 4 Banelings	Positioning
corridor	6 Zealots	24 Zerglings	Wall off
6h_vs_8z	6 Hydralisks	8 Zealots	Focus fire

5% over 50,000 time steps. The agents are trained over consecutive episodes, where at the end of each episode, a batch of 32 episodes is randomly sampled from an experience buffer of size 5000 episodes for a round of training. The learning rate is fixed at 5×10^{-4}. To stabilize training, double Q-learning is used (Hasselt, 2010; Hasselt et al., 2016), where the target agent Q-network and mixing GNN parameters are replaced with those of their main counterparts every 200 episodes. Training is conducted for 2×10^6 time steps. Moreover, every 20,000 time steps, training is paused and the agents are evaluated on a set of 32 test episodes.

The observation encoder for the attention mechanism in equation 11 is implemented using a single-layer mapping followed by a non-linearity; i.e., $\phi(o_v) = \sigma(Bo_v), \forall v \in \mathcal{V}$, where $\sigma(\cdot)$ denotes a non-linearity and $B \in \mathbb{R}^{F' \times \mathsf{dim}(\mathcal{Z})}$ denotes the encoding matrix, with $\mathsf{dim}(\mathcal{Z})$ representing the dimensionality of the observation space. We use the exponential linear unit (ELU) as the non-linearity and set $F' = 128$. Moreover, for the mixing GNN, we use a graph convolutional network (GCN) (Kipf & Welling, 2016) as also used by Li et al. (2020). Since the attention mechanism described in Section 4.1 leads to an effective outgoing degree of one for each graph node, the combining operation in equation 5 can be simplified as

$$h_v^l = \sigma\left(A_+^l \sum_{u \in \mathcal{V}} w_{uv} h_u^{l-1}\right), \forall v \in \mathcal{V}, \forall l \in \{1, \dots, L\}, \tag{12}$$

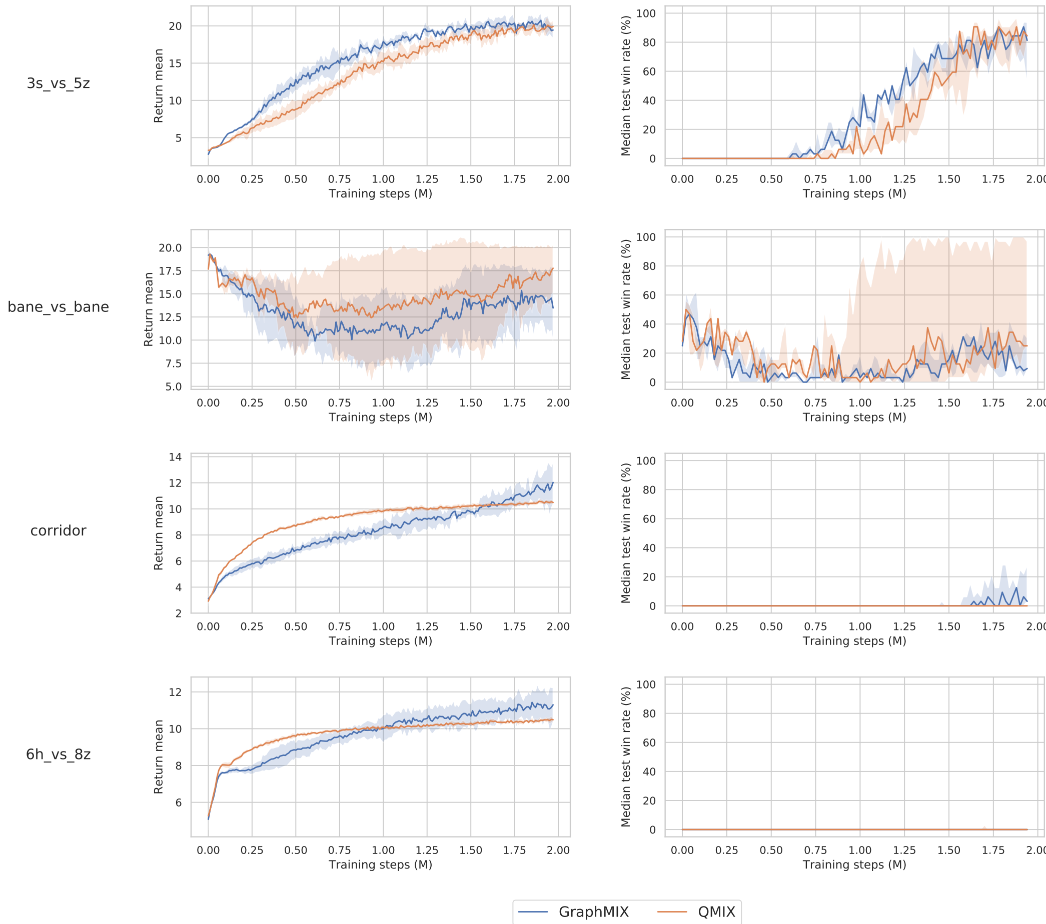


Figure 3: Comparison of the mean training return and median test win rate achieved by GraphMIX and QMIX (Rashid et al., 2018) on the four SMAC maps under study.

where $\sigma(\cdot)$ denotes a non-linearity, and $A^l_+ \in \mathbb{R}_+^{F_l \times F_{l-1}}$ is a trainable non-negative weight matrix that is trained in an end-to-end manner. We also use ELU as the non-linearity in equation 12, and use a single hidden layer for the GCN with 32 nodes per hidden layer and average pooling readout at the output. To evaluate the performance of GraphMIX against QMIX (Rashid et al., 2018), we use a similar MLP-based architecture for the mixing network in QMIX; i.e., an MLP with a single hidden layer, 32 neurons per hidden layer, and ELU non-linearity.

Furthermore, similar to QMIX, we use a hypernetwork architecture to determine the parameters of the mixing GNN via the global environment state during training. In particular, each of the mixing GNN parameters (A^1_+, w_+, and w_{local}) is the reshaped output of a neural network with a single hidden layer of size 64 and ELU non-linearity, which takes the global state as the input. The non-negativity constraint for the parameters of the mixing GNN in GraphMIX and the mixing MLP in QMIX is satisfied by taking the absolute value of the parameters after each gradient descent iteration.

Figure 3 shows the average training returns and median test win rates achieved by GraphMIX and compares it with those of QMIX over the four aforementioned SMAC maps. The solid curves on the left (resp., right) column show the mean (resp., median) across five training runs with different random seeds, with the shaded areas representing the standard deviation (resp., 25-75 percentiles). As the figure demonstrates, GraphMIX is able to considerably outperform QMIX in terms of the sample complexity (in 3s_vs_5z), reduced variance (in bane_vs_bane), and the final average training return (corridor and 6h_vs_8z) and test win rate (corridor).

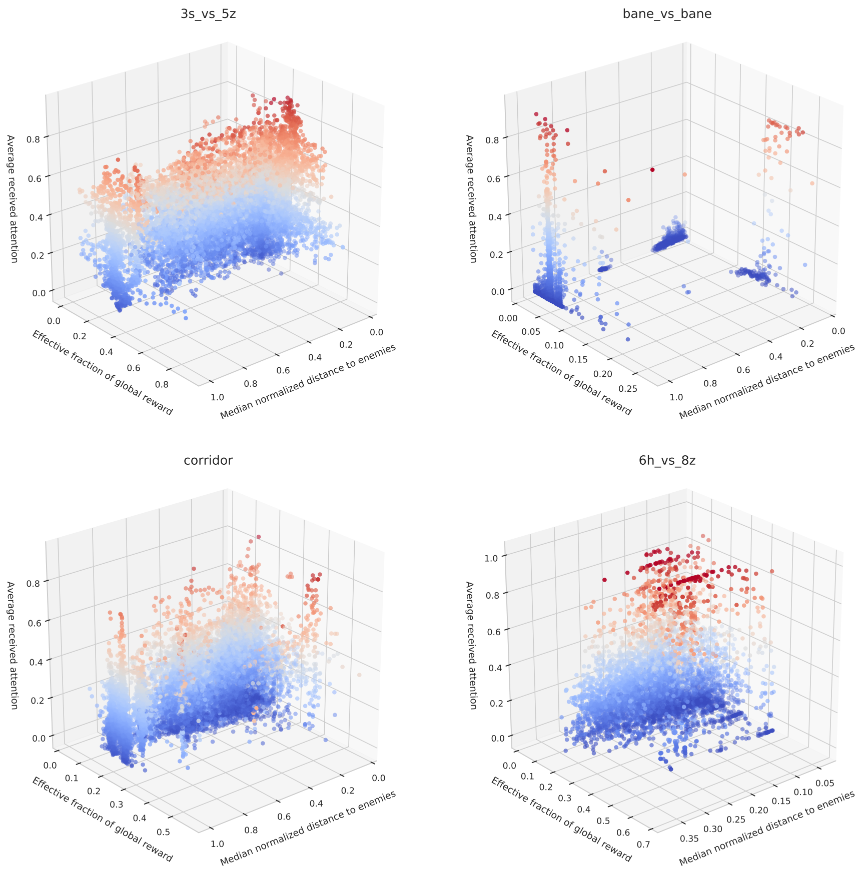


Figure 4: Trade-off between the average attention received by an agent from its (alive) team-mates, the effective fraction of global reward allocated to that agent, and the agent's median distance to the enemy units. Each point represents an alive agent in a single time step within a batch of transitions sampled from the experience buffer while training GraphMIX.

Moreover, Figure 4 illustrates the trade-off between the average received attention of each agent from its team-mates, its effective fraction of the global team reward, and its median distance to the enemy units. As the figure demonstrates, GraphMIX learns an adaptive underlying attention and credit assignment mechanism that dynamically adjusts itself to different environment conditions. Of particular interest is the trend happening in the bane_vs_bane scenario. It appears that, as opposed to other maps, the vast majority of agents in this map receive little attention from their team-mates. This implies that in this map, little coordination might be required between the agents to defeat the enemy units, which is consistent with the observation by Rashid et al. (2018) that independent Q-learning (Tan, 1993) easily succeeds in this map. This sheds light on how our proposed approach can automatically adapt the coordination between agents in different settings.

6 Conclusion

We introduced GraphMIX, a novel approach to decompose joint state-action value functions in multi-agent deep reinforcement learning using a graph neural network formulation under the centralized training and decentralized execution paradigm. Our proposed method allows for a more explicit representation of agent-to-agent relationships by leveraging an attention-based graph topology that models the dynamics between the agents as the episodes progress. To build upon the factorized state-action value function's implicit assignment of global reward, we define additional per-agent loss terms derived from the output node embeddings of the graph neural network, which explicitly divide the global reward to individual agents. Experiments in the StarCraft Multi-Agent Challenge (SMAC) environment show improved performance over the state of the art in multiple settings.

REFERENCES

Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

Tianshu Chu, Sandeep Chinchali, and Sachin Katti. Multi-agent reinforcement learning for networked system control. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Syx7A3NFvH.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Mark Eisen and Alejandro Ribeiro. Optimal wireless resource allocation with random edge graph neural networks. *IEEE Transactions on Signal Processing*, 68:2977–2991, 2020.

Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, pp. 417–426, 2019.

Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in neural information processing systems*, pp. 2137–2145, 2016.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients, 2018. URL https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17193.

Hado V. Hasselt. Double Q-learning. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta (eds.), *Advances in Neural Information Processing Systems 23*, pp. 2613–2621. Curran Associates, Inc., 2010. URL http://papers.nips.cc/paper/3964-double-q-learning.pdf.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 2094–2100. AAAI Press, 2016.

Weihua Hu*, Bowen Liu*, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJlWWJSFDH.

Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 2961–2970. PMLR, 2019.

Tao Ji, Yuanbin Wu, and Man Lan. Graph-based dependency parsing with graph neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2475–2485, 2019.

Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. Graph convolutional reinforcement learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HkxdQkSYDB.

Xiangyang Ju, Steven Farrell, Paolo Calafiura, Daniel Murnane, Lindsey Gray, Thomas Klijnsma, Kevin Pedro, Giuseppe Cerati, Jim Kowalkowski, Gabriel Perdue, et al. Graph neural networks for particle reconstruction in high energy physics detectors. *arXiv preprint arXiv:2003.11603*, 2020.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Sheng Li, Jayesh K Gupta, Peter Morales, Ross Allen, and Mykel J Kochenderfer. Deep implicit coordination graphs for multi-agent reinforcement learning. *arXiv preprint arXiv:2006.11438*, 2020.

Iou-Jen Liu, Raymond A Yeh, and Alexander G Schwing. PIC: permutation invariant critic for multi-agent deep reinforcement learning. In *Conference on Robot Learning*, pp. 590–602, 2020.

Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems*, pp. 6379–6390, 2017.

O Malaschuk and A Dyumin. Intelligent multi-agent system for rescue missions. In *Advanced Technologies in Robotics and Intelligent Systems*, pp. 89–97. Springer, 2020.

Navid Naderializadeh, Jaroslaw Sydir, Meryem Simsek, and Hosein Nikopour. Resource management in wireless networks via multi-agent deep reinforcement learning. In *2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp. 1–5, 2020.

Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Comparative evaluation of multi-agent deep reinforcement learning algorithms. *arXiv preprint arXiv:2006.07869*, 2020.

Namyong Park, Andrey Kan, Xin Luna Dong, Tong Zhao, and Christos Faloutsos. Estimating node importance in knowledge graphs using graph neural networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 596–606, 2019.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 4295–4304, 2018.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft multi-agent challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2186–2188, 2019.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:1905.05408*, 2019.

Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in neural information processing systems*, pp. 2244–2252, 2016.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinícius Flores Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *AAMAS*, pp. 2085–2087, 2018.

Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pp. 330–337, 1993.

Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Shangyu Xie, Yuan Hong, and Peng-Jun Wan. A privacy preserving multiagent system for load balancing in the smart grid. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2273–2275, 2019.

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

Riccardo Zanol, Federico Chiariotti, and Andrea Zanella. Drone mapping through multi-agent reinforcement learning. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–7. IEEE, 2019.

Tianyang Zhao, Yifei Xu, Mathew Monfort, Wongun Choi, Chris Baker, Yibiao Zhao, Yizhou Wang, and Ying Nian Wu. Multi-agent tensor fusion for contextual trajectory prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12126–12134, 2019.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.