A Survey of Multi-Agent Deep Reinforcement Learning with Graph Neural Network-Based Communication*

Valentin Cuzin-Rambaud[1], Laetitia Matignon[1], Maxime Morge[1]

[1]Université Lyon 1, INSA Lyon, CNRS, LIRIS, UMR 5205, Lyon, France

{valentin.cuzin-rambaud, laetitia.matignon, maxime.morge}@univ-lyon1.fr

Résumé

En apprentissage par renforcement multi-agents (MARL), l'intégration de mécanismes de communication permet aux agents d'apprendre à coordonner leurs actions et à converger vers leurs objectifs en partageant des informations. Sur la base d'un graphe d'interaction, une sous-classe de méthodes utilise des réseaux de neurones de graphes (GNN) pour apprendre à communiquer, ce qui permet aux agents d'améliorer leur représentation interne enrichie par les informations échangées. Avec l'essor récent des travaux dans ce domaine, nous constatons un manque de visibilité dans la distinction et la classification des approches MARL avec communication basée sur les GNNs. Ainsi, cet article passe en revue les travaux récents dans ce domaine. Nous proposons ici un processus généralisé de communication basé sur les GNNs dans le but de rendre plus évidents et accessibles les concepts sous-jacents à ces approches.

Mots-clés

Apprentissage par renforcement multi-agents, Communication, Réseau de neurones de graphes.

Abstract

In multi-agent reinforcement learning (MARL), the integration of a communication mechanism, allowing agents to better learn to coordinate their actions and converge on their objectives by sharing information. Based on an interaction graph, a subclass of methods employs graph neural networks (GNNs) to learn the communication, enabling agents to improve their internal representations by enriching them with information exchanged. With growing research, we note a lack of explicit structure and framework to distinguish and classify MARL approaches with communication based on GNNs. Thus, this paper surveys recent works in this field. We propose a generalized GNN-based communication process with the goal of making the underlying concepts behind the methods more obvious and accessible.

Keywords

Multi-Agent Reinforcement Learning, Communication, Graph Neural Networks.

*We gratefully acknowledge Université Lyon 1 and the AAP AEC for their support of this research

1 Introduction

Multi-Agent Systems (MAS) address a wide range of applications in robotics, video games, and cybersecurity. The main difficulty in designing MAS lies in developing the internal mechanisms that govern agents' behaviors and interactions. Reinforcement Learning (RL) provides a framework through which an agent can learn to behave effectively through experience. In the Multi-Agent Reinforcement Learning (MARL) setting, agents learn simultaneously while attempting to coordinate with one another by executing local policies. However, each agent typically operates under partial observability of the global state of the system and must therefore act based on incomplete information. Moreover, because all agents update their policies concurrently, the environment becomes non-stationary from the point of view of each agent: the transition dynamics depend on the evolving behaviors of the other agents. This non-stationarity significantly complicates learning in MAS and may prevent convergence to optimal policies.

To tackle partial observability and non-stationarity in MARL, we can consider communication between agents. Indeed, agents can communicate some information, e.g. their local observation or an internal representation of their mental state, to obtain a broader view of the environment and make a well-informed decision. Yet communication raises additional challenges. It requires specifying how messages are created, when and with whom communication should occur, how incoming messages are interpreted and aggregated, and how communication is integrated into the learning and execution phases.

Various approaches have been proposed in the literature to integrate communication into MARL and address its challenges. A particularly active research direction in recent years has focused on the use of Graph Neural Networks (GNNs) [20] for communication in MARL. GNNs have emerged as a popular solution to learn and extract knowledge from a graph. When data contains relationships between entities, modeling a graph permits extracting useful information. In many cases, graphs are used to represent data such as infrastructure, biological, social, and collaboration networks. In MARL with a communication context, we can thus represent the state of an environment as a graph where agents are positioned as nodes, and communication

between them is represented by edges. Zhu et al. cover MARL with communication [32] more broadly than our work, which focuses on a deeper analysis of twelve major recent GNN-based communication methods, reflecting the growing interest in GNNs for communication.
We note a lack of explicit structure and framework to distinguish and classify MARL approaches with communication based on GNNs, as there are many methods, with great diversity. Hence, this survey analyzes different methods of communication in MARL based on GNNs. We motivate the interest in using GNNs, and dive into state-of-the-art methods, by comparing them, extracting tendencies, and pointing out limitations of such approaches. Our study leads us to derive a generic algorithm which generalizes GNN-based communication process in MARL.
We first take a step back in the background Section 2 to present GNNs, RL, MARL, and MARL with communication. Section 3 presents the generic GNN-based communication process we propose and a survey of GNN-based communication MARL methods. This is followed by a specific focus on how realistic communication constraints are taken into account in state-of-the-art approaches. Finally, we conclude with future research directions in Section 4.

2 Background

In this section, we first outline key aspects of GNNs, as they are widely used for communication. Secondly, we briefly establish the foundation of RL, then explore MARL, focusing progressively on communication in MARL.

2.1 Graph Neural Networks

Several tasks on graphs, such as node classification, link prediction, and graph classification leverage machine learning methods [31]. A major approach for solving these tasks is the use of deep learning methods, in particular Graph Neural Networks (GNNs) [20].

Definition 2.1. A **graph** at time t, denoted $G^t(V, E)$, is defined by a set of n nodes V, and a set of edges E, where an edge represents the influence of the source over the target, possibly reciprocal. $\mathcal{N}(i)$ denotes the set of neighbors of node i, as $\forall j \in \mathcal{N}(i), \exists (j,i) \in E$ and $N_i = deg(i) = |\mathcal{N}(i)|$. The graph can potentially admit self-loops such as $\forall i \in V, (i,i) \in E$ and thus i is included in its neighborhood $i \in \mathcal{N}(i)$. We denote $A \in \mathbb{R}^{n \times n}$ the adjacency matrix, with $A_{ji} = 1 \iff (j,i) \in E$, and $X \in \mathbb{R}^{n \times d}$ the node feature matrix, with $X[i] \in \mathbb{R}^d$ being the feature vector of the node i and d the feature dimension. If edges also have features, we denote E_{attr} the edge feature matrix, with $E_{attr}[(j,i)]$ abbreviating $e_{j,i}$ referring to the feature vector of the edge (j,i). The graph is directed and dynamic as it evolves in a finite number of steps $t \in [0, T]$, and can be potentially weighted, with each edge having $ew_{j,i}$ the weight of the edge $(j,i) \in E$.

A GNN is defined as a parameterized function $f(X, A, E_{attr}; \theta)$. In a layered GNN with L layers, learnable parameters θ are a set of weight matrices $\{W^{(l)}\}_{l=0}^{L-1}$ shared among all nodes of the graph. The GNN learns to compute node representations (or embeddings), noted $h_i^{(l)}$ for the representation of the node i at layer l. It does so by applying iteratively (L times) the layer process. One layer process consists of two main steps for each node: 1) the **aggregation** step to gather representations from its neighbors, defined by the adjacency matrix; 2) the **transformation** step to transform the aggregated information using a weight matrix to obtain updated node representations. Thus, the edges of the graph determine which neighboring representations are aggregated, while the weight matrix controls how this information is transformed.
Inspired by the Convolutional Neural Network (CNN), one of the first GNN architectures is the Graph Convolutional Networks (GCN) [11], which is applied on non-euclidean data. Applying a convolution on an image uses a kernel with a fixed size, which is impossible in a graph, as the number of neighbors N_i of each node i varies. Thus, the aggregation step of the GCN layer process consists of pooling neighbors' features, and the transformation step consists of computing the weighted sum of neighbors' values, with learnable weights. The updated representation of a node i at layer l is:

$$h_i^{(l)} = \sigma\Big(\sum_{j \in \mathcal{N}(i)} \frac{ew_{j,i}}{\sqrt{deg(i)deg(j)}} h_j^{(l-1)} W^{(l-1)}\Big) \quad (1)$$

The average is weighted by the degree of nodes preventing high-degree nodes from dominating the aggregation and keeping the feature magnitudes stable.
More recently, Graph Attention Networks (GAT) [26] proposes a new model based on attention heads:

$$h_i^{(l)} = \sum_{j \in \mathcal{N}(i)} \alpha_{ji} h_j^{(l-1)} W^{(l-1)} \quad (2)$$

with α_{ji} the attention weights from j to i. The use of a learnable attention vector allows choosing how much attention to give to each neighbor.
GCN and GAT are two popular architectures among the many variants of GNNs. Most of them can be instantiated to a single common framework: Message Passing Neural Networks (MPNNs) [7]. The layer process with MPNN is:

$$h_i^{(l)} = \psi^{(l)}(h_i^{(l-1)}, \bigoplus_{j \in \mathcal{N}(i)} \phi^{(l)}(h_i^{(l-1)}, h_j^{(l-1)}, e_{j,i})) \quad (3)$$

with ψ and ϕ differentiable functions, e.g. Multi Layer Perceptrons (MLP), and $\bigoplus$ the aggregator function, (e.g. mean, add, max). $\phi^{(l)}$ represents the message construction, and for GCN and GAT cases, it would contain weights $W^{(l)}$.
The main advantage of using MPNN is transmitting information in the graphs to l-hops by stacking layers, with nodes communicating only with immediate neighbors. In Figure 1, the green node aggregates information from its neighbors (nodes 2,3 and 4) in 1-hop. Repeating the process permits accessing information from 2-hop nodes (node 5) as their features were previously aggregated by 1-hop

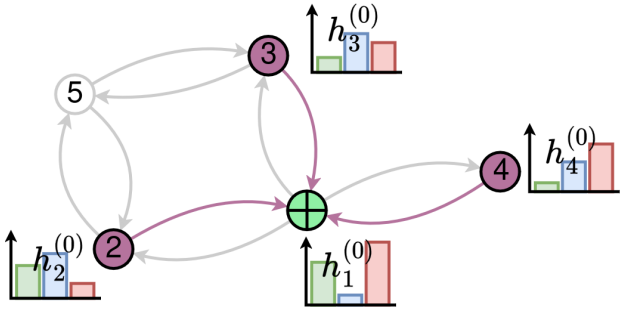

Figure 1: An example of MPNN: the green node, aggregates features from its 1-hop neighbors $\in \mathcal{N}(1)$

nodes. Other advantages of GNNs include generalizability to unseen nodes (inductive learning), invariance of permuting node order during aggregation, handling of non-Euclidean structure, and capturing local and global information to learn complex patterns. All these advantages have led communication methods to integrate GNNs in their process. In particular, in distributed communication using GNNs with $l > 1$, agents (considered as nodes) indirectly aggregate information about unreachable agents, effectively extending information propagation despite limited communication range.

2.2 Reinforcement Learning

In a sequential decision-making setting, reinforcement learning (RL) algorithms enable an agent to learn solutions through repeated experiments with an environment.

The standard model to define the sequential decision process is the Markov Decision Process (MDP) defined with S the set of environment states and μ its initial state distribution such that $s_0 \sim \mu$, A the set of actions for the agent, $\mathcal{R}$ the reward function and $\mathcal{T}$ the state transition probability function. In a more realistic case, the agent only partially observes the state of the environment. Formally, we define the Partially Observable Markov Decision Process (POMDP) as an MDP extended with O, the set of observations. $\mathcal{O}$ defines a probability distribution over possible observations. The agent interacts with the environment during a set of episodes (i.e. sequence of states and actions until a terminal state).

The discounted return from time step t is defined as the sum of discounted rewards over time, i.e. $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ where the discount factor $\gamma \in [0, 1[$ ensures finite return in non-terminating MDPs, and r_t is the reward received at time step t. A policy $\pi(a|s)$ defines a mapping from states to a probability distribution over actions. The goal of an agent is to find an optimal policy π^* which maximizes the *expected discounted return* from every state $s \in S$, i.e. $\pi^* = \arg\max_\pi \mathbb{E}_\pi[R_t|s_t = s]$. We can define two useful functions: the value function $V^\pi(s) = \mathbb{E}_\pi[R_t|s_t = s]$, and the action-value function $Q^\pi(s, a) = \mathbb{E}_\pi[R_t|s_t = s, a_t = a]$ which enable to evaluate the expected return based on a state s or a couple (s, a). The use of V or Q is essential, as an action can lead to a high immediate reward, but in the long term, a poor return.

In RL, a common assumption is that the agent has no a priori knowledge about the transition and reward functions, leading to the need to collect experiences to learn a policy π [27].

The use of deep neural networks to parameterize and approximate value functions and policies over large state and action spaces has become standard practice, giving rise to Deep Reinforcement Learning (DRL). We can classify DRL algorithms into two categories: value-based algorithms, which optimize a value function like DQN [17]; and policy-based algorithms, which optimize the policy, like REINFORCE [28]. A particular subclass of policy-based algorithm is the actor-critic algorithm, which combines the use of a value function as a critic to guide policy learning such as A2C [16] or PPO [21].

2.3 Multi-Agent Reinforcement Learning

In a multi-agent setting, a POMDP extends to a Partially Observable Stochastic Game (POSG).

Definition 2.2. A **POSG** [1] is defined by a tuple $\langle I, S, \mu, \{O_i\}, \{A_i\}, \{\mathcal{R}_i\}, \mathcal{T}, \{\mathcal{O}_i\}\rangle$, with I the set of agents indexed as $\{1, \ldots, n\}$[1], S the set of states and μ the initial state distribution such as $s^0 \sim \mu$, O_i the set of observations for agent i, A_i the set of actions. We denote the joint action space $\mathbf{A} = \times_{i \in I} A_i$. Thus, $\mathcal{R}_i : S \times \mathbf{A} \times S \to \mathbb{R}$ is the reward function for agent i, $\mathcal{T} : S \times \mathbf{A} \times S \to [0, 1]$ is the state transition probability function, and $\mathcal{O}_i : S \times \mathbf{A} \times O_i \to [0, 1]$ is the observation function of agent i.

At each time step t, each agent i receives a local observation of the current state $o_i^t \subset s^t$ given by its observation function $\mathcal{O}_i(o_i^t|s^t, a^{t-1})$. All the individual observations give the joint observation $o^t = \langle o_1^t, \ldots, o_n^t\rangle$, which approximates the current state $o_i^t \subset o^t \subseteq s^t$. In addition, each agent can memorize its observation-action history $\tau_i^t = [(o_i^0, a_i^0), \ldots, (o_i^t, a_i^t)]$ often encoded by a Recurrent Neural Network (RNN). Thus, by following its policy $\pi_i(a_i^t|\tau_i^t)$, each agent i chooses action a_i^t, forming the joint action $a^t = \langle a_1^t, \ldots, a_n^t\rangle$. The game transitions to the next state $s^{t+1} \in S$ with $\mathcal{T}(s^{t+1}|s^t, a^t)$ probability, and each agent i receives its reward $r_i^t = \mathcal{R}_i(s^t, a^t, s^{t+1})$. We note $R_i = \sum_{t=0}^{\infty} \gamma^t r_i^t$ the discounted return for the agent i.

Depending on the reward structure, POSGs can be classified as competitive, cooperative, or mixed games. In competitive games, agents are opponents (e.g., $\sum_{i \in I} \mathcal{R}_i(a) = 0, \forall a \in \mathbf{A}$). In cooperative games, agents aim to maximize the global expected return of all agents $\sum_{i \in I} \mathbb{E}_{\pi_i}[R_i]$ (e.g., common-reward: $\mathcal{R}_1 = \mathcal{R}_2 = \cdots = \mathcal{R}_n$). In mixed games, agents may share partially aligned interests while still pursuing individual objectives.

Example: Predator-Prey. The Predator-Prey environment is well-known and massively used to evaluate MARL algorithms. Let n agents (predators) evolve in a square grid by

[1] In the multi-agent setting, time steps are denoted using superscripts rather than subscripts to avoid confusion with agent indices.

choosing a movement action {up, down, left, right, stay}. Their objective is to catch a stationary prey while constrained by a limited vision range. Once a predator reaches the prey, it stays there and always gets a positive reward until the end of the episode (reaching the time step limit or all predators have reached the prey). The environment is cooperative: agents obtain a better reward when more agents reach the prey.

Multi-Agent DRL (MADRL) algorithms fall into two paradigms.

Decentralized Training and Execution (DTDE). DTDE addresses the problem independently for each agent. Each agent uses its own local information to choose its own behavior. This learning framework permits both training and execution in a physically distributed setting, such as robotic scenarios. All RL algorithms can be adapted to DTDE, e.g. IDQN [25] and IPPO [3]. However, the main problem resides in the non-stationarity of the environment, as many agents learn simultaneously. From the point of view of one agent, all other agents are part of the environment, so the environment always changes even without its action.

Centralized Training and Decentralized Execution (CTDE). In the CTDE setting, additional global information can be centralized and exploited during training to stabilize and improve learning. However, at execution time, each agent follows a decentralized policy using only its own local observations, ensuring fully decentralized execution. The use of centralized information during training helps to reduce the non-stationarity environment problem. In most environments, centralized information is accessible only during training, as the training process is typically simulated. This is the reason why many popular algorithms follow the CTDE paradigm. Some use value decomposition to centrally combine individual utility functions during training, like VDN [24] or QMIX [19], while others use a centralized critic and decentralized policies like MADDPG [15] and MAPPO [30].

Whatever the paradigm is, agents can either share the same set of network parameters or not. If agents share their parameters, only one set of parameters is learned and then duplicated into each agent, for the decentralized execution. In common-reward cooperative games, sharing weights of the model during learning permits converging faster, but at the cost of having less chance to develop diversity in behaviors between policies [1, 2].

2.4 MARL with Communication

Classical CTDE settings assume no communication between agents during execution. However, the centralization of information during training can be seen as an implicit communication process: all agents send their local observation to a centralized node that aggregates information to form the joint observation. Indeed, the messages exchanged during the training phase are not formally specified. In contrast, the explicit communication process that we refer here is learnable, where information is explicitly communicated between agents. This grant a broader representation of the global state for each agent, while remaining in a decentralized execution setting.

Most methods use communication during execution to balance the partial observability while enabling better coordination strategies. The communication process can also be used only during training, for instance to learn a more effective information combination than predefined CTDE schemes. The positive impact of communication on coordination explains why cooperative scenarios are the most relevant setting for its use.

Incorporating explicit communication in MARL requires extending the joint action space to include communication actions. Zhu et al. extend POSG with $\mathcal{M}$, the shared message space, leading to POSG-Comm [32].

Learning to communicate brings some **new challenges**:

1. Generating the content of the message to be sent.
2. Choosing when and with whom to communicate.
3. Interpreting/combining received messages.
4. Leveraging the acquired knowledge.

Many methods have been proposed to solve these new challenges [32]. Each of these issues can be resolved either through learning or by relying on a fixed, manually designed solution. Most state-of-the-art approaches incorporate learning to handle at least one of these challenges. The communication is often learned in an end-to-end fashion (with backpropagation flow for all differentiable functions), so agents communicate directly to improve the global return.

A key distinction between MARL methods with communication is whether the communication requires a central control node or not. This means that some methods need a centralized data structure to control the communication policy. This is endorsed by the proxy, an agent that cannot directly interact with the environment but is responsible for the communication policy, e.g. learning to aggregate observations for a centralized critic. The use of a proxy during the training phase is a soft constraint, as it is often implemented within a simulator and centralized. Yet during execution it is a harder constraint, as it requires perfect communication between all the agents and the proxy. While this constraint may be realistic in some settings, such as warehouse environments, it restricts the applicability to other contexts, e.g. fleet of drones. In methods without a proxy, each agent communicates in a distributed manner, within a communication-range, which makes the approach more flexible and applicable to a wider set of scenarios.

Algorithm 1 summarizes the process for the training and the execution phases. Parametric functions are any common learnable function $f(\cdot;\theta)$ with associated weight θ. For example, it can be used to compute action probabilities for the policy or to serve a critic in actor-critic methods. The communication process permits obtaining relevant information by exchanging messages between agents, which influences the policy of agents. Agents are either fully connected, or constrained by the communication range. Various mechanisms using GNNs to address communication challenges will be explored in the following section.

Algorithm 1: General Pipeline for MARL with communication

Input: The environment env

Initialize all parametric functions $f_i(\cdot;\theta_i)$ with their associated weights θ_i ;
for *every episode* **do**
  Observe at $o_1^0, \ldots, o_n^0$ from env at $t = 0$;
  for *time step* $t = 1, 2, \ldots, T-1$ **do**
    ▷ `Create representations with Algo. 2`
    $h_1^t, \ldots, h_n^t = Communicate(o_1^t, \ldots, o_n^t)$;
    Sample actions $a_1^t, \ldots, a_n^t \sim \pi_1^t(\cdot|h_1^t;\theta_1), \ldots, \pi_n^t(\cdot|h_n^t;\theta_n)$;
    Perform actions in env ;
    Collect rewards $r_1^t, \ldots, r_n^t$ from env ;
    Observe $o_1^{t+1}, \ldots, o_n^{t+1}$ from env ;
    if *in Training phase* **then**
      Compute losses for parametric functions, based on agents' interactions with env ;
      Update θ_i by backpropagating gradients ;

3 MADRL with GNN-Based Communication

GNNs excel at propagating information between nodes and at modeling structured and dynamic interactions among agents [10, 12, 13]. In this section, we first abstract and generalize the communication process used in existing state-of-the-art approaches to design a generic algorithm for communication in MADRL. Secondly, we survey GNN-based communication methods, their advances, and limitations, including the widespread neglect of communication constraints.

3.1 Communication Model with GNNs

Introducing GNNs in the communication process can help to solve challenges. Creating messages through GNNs permits communication in multiple rounds. Building a graph at each timestep and designing an MPNN determines when[2] and with whom to communicate, and how to create messages and interpret received messages. The last aggregation and transformation made by the MPNN gives a final representation (i.e., communication-aware representation) that is integrated in the learning process, and optionally in the execution.

To aggregate many GNN-based methods within a high-level framework, we propose a generic communication process using GNNs in MADRL, presented in Algorithm 2. We distinguish between two cases: the use of a proxy or distributed communication without a proxy. The associated proxy part is detailed in Algorithm 3.

The algorithm can be unfolded into four main parts:

[2] At each timestep, the agent decides whom to communicate with, so agents implicitly learn when to communicate.

Algorithm 2: Generic communication process in MADRL with GNNs at time t for agent i

Input: o_i^t: the local observation

▷ `Encode the message to send`
1 Encode current representation: $x_i^t \leftarrow E(o_i^t)$;

▷ `Decide with whom to communicate`
2 **if** *exists a proxy* P **then**
3 Send x_i^t to P ;
  ▷ `Execute Algo. 3`
4 Receive $h_{P,i}^{t,L}$ from P;
5 **if** *exists a distributed comm* **then**
6 Send x_i^t to $\{j \in \mathcal{N}_r^t(i)\}$;
7 $X_i^t \leftarrow \{x_j^t | i \in \mathcal{N}_r^t(j), \forall j \in I\}$;

  ▷ `Combine received msg`
8 $G_i^t(V_i^t, E_i^t) \leftarrow G_{build}(X_i^t)$: the local graph ;
9 $H_i^{t,0} = X_i^t$;
10 **for** $l = 1 \ldots L$ **do**
    ▷ `Aggregate msg`
11    $h_i^{t,l} \leftarrow GNN_i^l(H_i^{t,l-1}, E_i^t)[i]$;
12   **if** *multi-round comm. and* $l > 1$ **then**
      ▷ `Exchange updated representation`
13   Send $h_i^{t,l}$ to $\{j \in \mathcal{N}_r^t(i)\}$;
14      $H_i^{t,l} \leftarrow \{h_j^{t,l} | i \in \mathcal{N}_r^t(j), \forall j \in I\}$;
15   **if** *evolution of relation* **then**
16      $G_i^t(V_i^t, E_i^t) \leftarrow G_{build}^l(H_i^{t,l})$;

Output: The final representation $h_i^{t,L}$, Possibly final representation from the proxy $h_{P,i}^{t,L}$

▷ `Inner integration to policy/value-level`

1. Encode the message to send. To generate the first message to send at time t, we use $E(\cdot)$, the message encoder: it maps local observation of the agent o_i^t to encoded message x_i^t (line 1). It can be implemented as any parametric function (e.g., MLP, RNN, CNN), or as a predefined mapping, such as the identity function, which preserves the raw input. Using an RNN introduces a hidden state to capture history.

2. Decide with whom to communicate. In proxy communication settings, all agents send their local representations to the proxy (line 3). In distributed settings, each agent i sends its representation to agents $j \in \mathcal{N}_r^t(i)$ with $\mathcal{N}_r^t(i)$ containing the set of reachable neighbors from i, at time t (line 6). Each agent i, received messages from every agent $j \in I$, that considers $i \in \mathcal{N}_r^t(j)$ (line 7). $\mathcal{N}_r^t(k)$ is determined by potential communication range or learnable choices to decide with whom it is possible or beneficial to communicate.

3a. Combine received messages with distributed communication. Following the previous message-sending phase at time t, each agent i receives a set of messages X_i^t

from the others (line 7). A local graph for agent i at time t, noted G_i^t, is constructed by $G_{build}(\cdot)$ function, with all previous received messages X_i^t (line 8). This local graph G_i^t represents all reachable agents from i (all $j \in \mathcal{N}_r^t(i)$) as nodes, their communication as edges, and message contents as node feature $X_i^t[j] : \forall j \in \mathcal{N}_r^t(i)$ (see Def. 2.1). G_i^t is then used to aggregate messages with GNNs. Combining messages until the last layer L of GNNs leads the agent to obtain a final representation for its corresponding node in the graph, $h_i^{t,L}$, which is intended to be a broader representation of the current global state of the world. If $l = 1$, information is aggregated only with the 1-hop neighbor, but if $l > 1$, two possible cases occur. In the *multi-round communication* case, at each layer of GNNs, each agent communicates with its neighbors, and its intern representation $h_i^{t,l}$ is updated at layer l (line 13). The *multi-round communication* leverages the diffusion/propagation mechanism of MPNN (see Fig. 1). Otherwise, each agent i computes representations l times, for all reachable nodes, which implies that the i's representation of agent j may differ from j's self-representation (i.e., they do not have the same local graph). The multi-round communication ensures construction of more globally consistent representations, but demands more messages. Whereas without multi-round communication, agents communicate only once, but representations have limited global consistency and may be biased. Furthermore, with the *evolution of relation* enabled, G_i^t can be updated at each layer via $G_{build}^l(\cdot)$ to learn a different communication structure within the same timestep (line 16). The final representation $h_i^{t,L}$ is integrated in the MADRL pipeline in the next phase. The final representation replaces the usage of raw observation o_i^t.

Algorithm 3: Proxy' communication process at time t

1 Receive $X_P^t \leftarrow \{x_i^t | \forall i \in I\}$;
2 $G_P^t(V_P^t, E_P^t) \leftarrow G_{build}(X_P^t)$: the graph of Proxy ;
3 $H_P^{t,0} = X_P^t$;
4 **for** $l = 1 \dots L$ **do**
5 $H_P^{t,l} \leftarrow GNN_P^l(H_P^{t,l-1}, E_P^t)$;
6 **if** *evolution of relation* **then**
7 $G_P^t(V_P^t, E_P^t) \leftarrow {G_{build}}^l(H_P^{t,l})$;
8 $\forall i \in I$, send $H_P^{t,L}[i]$;

3b. Combine received messages with the proxy (Algo. 3). In proxy communication, the construction of the graph is centralized. The proxy receives messages from all agents X_P^t (line 1). This global view allows to build a more globally consistent graph than the distributed communication, where all nodes are represented, and edges are not restricted by a communication range (line 2). Communication is established once, with the strong requirement that all agents are connected to the proxy. Information of nodes are aggregated through successive layers via the GNN. The final joint representation of all nodes $H_P^{t,L}$ can be used directly as centralized information (replacing the joint observation), and the proxy sends back to each agent its self-representation $h_{P,i}^{t,L} = H_P^{t,L}[i]$.

4. Inner integration to policy/value-level. The obtained final representation $h_i^{t,L}$ is leveraged in training, execution, or both, depending on the MADRL method (value-based, value-decomposition, or actor-critic). If a proxy exists, each agent retrieves its final representation made by the proxy $h_{P,i}^{t,L}$ and can potentially use its. We note that the full representation $H^{t,L}$ can serve for centralized training. The final representation can be combined with other knowledge of the agent before any usage, e.g. the concatenation with raw observation $h_i^{t,L} \leftarrow [o_i^t \oplus h_i^{t,L}]$. The communication is often learned in an end-to-end manner, but in specific methods, an additional objective function can update communication-dependent weights during supervised, self-supervised, or reinforcement learning.

Example: Predator-Prey with distributed communication. We assume that three agents are in range at time t. If agents communicate their local observations, they can cover a broad search space. By sharing information about previously explored areas where the prey was not observed, agents can take more informed actions. If agent i finds the prey, its message will inform its two neighbors. With *multi-round communication*, the information can propagate beyond the communication range limit of the agent, similarly to Figure 1. Any agent connected to the real communication graph, and at $L - hop$ from agent i, can obtain the prey's position.

3.2 GNN for communication

Building upon the generic algorithm introduced earlier, we provide in this section a survey of GNN-based communication MADRL methods by instantiating each component of our generic algorithm (cf. Table 1). The proposed generic algorithm facilitates comparative analysis, reveals common tendencies and structural patterns, and enables the classification of existing methods.

Proxy-Communication Methods. Many methods are based on a proxy to handle all communications. One of the first methods to use a proxy is Deep Implicit Coordination Graphs (DICG) [12], which extends MAPPO [30] and uses communication for the centralized critic only. This permits fully decentralized execution since the critic is used only during training. DICG uses the proxy to compute a complete weighted graph called Implicit Coordination Graph. Each edge obtains a weight computed by a self-attention mechanism. The proxy then applies a GCN (Eq. 1) on the entire graph to obtain the final joint representation matrix, which feeds the centralized critic. Basically, DICG aggregates observations instead of using a concatenation of local observations to create the joint observation.

Unlike DICG, Game Abstraction Communication (GA-Comm) [13] uses a proxy for both training and execution. The graph is built in two phases: hard attention (dropping edge) and soft attention (weighting all remain edges via self-attention). This process is called the game abstraction of agents' interactions. The GNN used is a

Table 1: Taxonomy of GNN-based communication methods, organized in two categories: proxy-based (upper part) and distributed-based (lower part) according to the generic Algorithm 2. $E(\cdot)$ encodes the local representation, $\mathcal{N}_r^t(\cdot)$ defines reachable agents, $G_{build}(\cdot)$ builds a graph, GNN is the architecture used, and Inner integration shows how to leverage the final representation.

Methods	$E(\cdot)$	$\mathcal{N}_r^t(\cdot)$	$G_{build}(\cdot)$	GNN	Inner integration
DICG [12]	$\{LSTM, MLP\}(o_i^t)$	All agents	G_P^t : complete, weighted	$L=2, GCN$	$V(H_P^{t,L})$
GA-Comm [13]	$LSTM(MLP(o_i^t))$	All agents	G_P^t : sparse, weighted	$L=1, MPNN$ with attention	$\pi_i(\cdot\|h_{P,i}^{t,L})$
GAAC [13]	$LSTM(MLP(o_i^t))$	All agents	G_P^t : sparse, weighted	$L=1, MPNN$ with attention	$Q_i(h_{P,i}^{t,L})$
MAGIC [18]	$MLP(LSTM(MLP(o_i^t)))$	All agents	$G_P^{t,l}$: sparse, dynamic	$L=2/3, GAT$	$\pi_i(\cdot\|h_{P,i}^{t,L}), V_i(h_{P,i}^{t,L})$
GACG [4]	$MLP(o_i^t)$	All agents	G_P^t : sparse, weighted, group	$L=2, GCN$	$Q_i(h_{P,i}^{t,L})$
LTS-CG [5]	$MLP(o_i^t)$	All agents	G_P^t : sparse, weighted, temporal learned	$L=2, GCN$	$Q_i(h_{P,i}^{t,L})$
DGN [10]	$MLP(o_i^t)$	near agents	G_i^t : sparse	$L=2, MPNN$ with attention	$Q_i(h_i^{t,L})$
LSC [14]	$MLP(o_i^t)$	near hierarchic agents	G_i^t : sparse, hierarchic, heterogeneous	$L=3, MPNN$	$Q_i(h_i^{t,L})$
MAGE-X [29]	$MLP(o_i^t)$	near agents	G_i^t : sparse	$L=2, GCN$	$\pi_i(\cdot\|h_i^{t,L})$
MAGEC [8]	$Identity(o_i^t)$	near agents	G_i^t : sparse, heterogeneous	$L=10, GraphSAGE$	$\pi_i(\cdot\|h_i^{t,L}), V_i(h_i^{t,L})$
(Het)GPPO [2]	$Identity(o_i^t)$	near agents	G_i^t : sparse, undirected	$L=1, MPNN$	$\pi_i(\cdot\|h_i^{t,L}), V_i(h_i^{t,L})$
HetNet [22]	$MLP_i(LSTM(MLP(o_i^t)))$	near agents	G_i^t : sparse, undirected	$L=3, HetGAT$	$\pi_i(\cdot\|h_i^{t,L}), V(H_{P,i}^{t,L})$

custom one (Eq. 3), resulting from aggregation weighted by both hard and soft attention computed weights: $h_i = \sum_{j \neq i} W_h^{i,j} W_s^{i,j} h_j$. The final representation is then used for the policy trained with an independent multi-agent version of the REINFORCE algorithm [28]. A second implementation called Game Abstraction Actor-Critic (GAAC) is used to compute the local Q_i critic, thus, execution remains totally decentralized, without communication [13].

Multi-Agent Graph-attention Communication (MAGIC) [18] can be seen as an upgrade of GA-Comm, as it keeps the principle of hard and soft attention. The hard attention is handled here by a "sub-scheduler" process, while soft attention is directly learn through GAT (Eq. 2). The first sub-scheduler uses a GAT in the complete graph to compute the first node feature matrix X_P^t, then it processes through a sample of effective edges (probabilities to binary). Any additional sub-scheduler reuses X_P^t and samples a new adjacency matrix, updating the graph. The key contribution is the possibility of stacking several sub-schedulers followed by GAT aggregation. Thus, the graph is rebuilt between each aggregation of messages, (cf. line 6 of Algo. 3). Several layers of GNNs with evolving structure permit learning different kinds of relationships at each timestep. All created representations of an agent are concatenated and used for critic during training, and for policy during both training and execution.

Group Aware Coordination Graph (GACG) [4] builds a graph by enforcing the group structure to help agents cohesion. First, groups are formed with a classifier looking at a temporal window in the joint-observation history: if two agents have similar representations, they are in the same group. Then, the proxy computes the agent-pair matrix, which encode weight of relation between all agents, and besides it computes the edge-group matrix, which encodes if edges belong or not to the same group. A weighted adjacency matrix is then sample from a Gaussian distribution based on the agent-pair matrix as mean and the edge-group matrix as covariance. The method, which is built on top of QMIX [19], uses a GCN to aggregate messages, and the final representation is integrated into individual Q_i values.

Latent Temporal Sparse Coordination Graph (LTS-CG) [5] is another proxy method, focusing on building graphs by leveraging temporal information. First, the method creates an agent-pair matrix as GACG, from correlations between previous trajectories of agents. Then, a graph is sampled on a Bernoulli distribution with probabilities from the agent-group matrix. The key contribution comes from two pretext tasks used during the learning: predict-future observation and infer-present states. These tasks guide graph construction toward a latent temporal sparse graph that integrates temporal information in the structure of the graph. Furthermore, the graph is weighted and uses GCN (Eq. 1) to aggregate messages, integrated in the individual Q_i value for the QMIX framework [19].

Using a proxy during the execution requires all agents to have perfect communication with the proxy. Even under decentralized execution, the proxy centralizes the communication policy. So the proxy is a very strong constraint, and proxy-methods can not adapt to fully decentralized execution. It is worth noting that among proxy-based methods, only DICG and GAAC assume a fully decentralized execution, as they use the proxy only during training and do not communicate during execution.

Distributed Communication Methods. As discussed previously, using a proxy is a strong constraint on the environment, and establishing communication in a distributed manner among agents enables a wider range of applications. Moreover, communications between nearby agents are often more relevant than those between distant agents, because distant agents' observations are often not useful to others and can introduce irrelevant context. For this purpose, Graph Convolutional Reinforcement Learning (DGN) extends IDQN with GNNs for communication [10]. Each agent exchange its encoded message to reachable neighbors (cf. lines 6-7 of Algo. 2). A custom MPNN (Eq. 3) aggregates messages using self-attention weights during averaging. DGN enable *multi-round communication* (cf. lines 12-14 of Algo. 2), so each agent re-sends its updated representation $h_i^{t,1}$ to its neighbors. As explained in Example 2.3, this leverages the MPNN propagation mechanism (as

in Fig. 1). The inner integration plays a role in training and execution as Q_i is conditioned on the concatenation of $h_i^{t,1}$ and $h_i^{t,2}$ for helping the model to understand different kinds of relationships.

Scaling to larger number of agents is challenging in MARL. Using a hierarchical structure leverage more sparse communication topology. Learning Structured Communication (LSC) exploit a two-level hierarchical structure, with two agent types: low and high [23]. All low agents can communicate only with high-level neighbors. High agents are elected depending on a computed weight attribute based on local observations. If in the agent i range there is no high agent, and the i's weight is bigger than neighbors' weight, i becomes the new high agent. Agents do not aggregate information in the same ways, and thus, the behavior is not the same for all agents. Integrating this approach into our generic algorithm implies that high-level agent receive information, build a local graph G_i, and aggregate information at $l = 1$ with a GNN. Then, they exchange with high-level agents only, updating topology of G_i (cf. lines 15-16 of Algo. 2), and perform a second aggregation at $l = 2$. Finally, they send back final representations to each reachable low-level agent. A low-level agent, just sends information to high-level agents and receives an answer from them, which will be aggregated only once with $L_{low} = 1$. Their custom GNN sums received features and updates via a MLP (like a GCN without degree-normalization). The inner integration is on computing local Q_i values, during both training and execution, as LSC extends IDQN [25].

Although Multi-Agent Graph-Enhanced Commander-Executor (MAGE-X) [29] is presented as a distributed communication method, it still requires centralization at execution time. A supervising agent first assigns a goal to each agent in navigation tasks. Then, each agent constructs a local complete subgraph that includes all reachable neighbors. A first GCN is applied to this subgraph to compute node attributes, after which an adjacency sampling process is performed to obtain the graph G_t^i. A second GCN is then applied to G_t^i to produce the final representation. This representation is combined with the encoded goal assignment and integrated into both the policy and value functions within an IPPO-based framework [3].

Multi-Agent Graph Embedding-based Coordination (MAGEC) [8] addresses the patrolling problem, where agents navigate a graph to minimize the node idleness. This method builds a heterogeneous graph that includes both game's nodes and neighboring agents. An agent builds first a sub-graph with its limited observation: each node contains the type of node (agent/game node), the idleness time, and the degree of the node. Edges include a relative-position attribute. Agents communicate their position, goal-reached notifications, and attribution notifications (i.e. if the agent dies) to others. All received communication extends the graph with new agent node. The GNN used is GraphSAGE [9], with ten layers and a single-round communication, so many aggregations happen to enrich the final representation. GraphSAGE was designed for inductive learning and generalizes well to unseen nodes. The final representation is passed to the policy, which chooses the next node to visit. The main advantage is that the method shows resilience against noisy communication and agent attributions, compared to other patrolling algorithms.

Graph Proximal Policy Optimization (GPPO) and Heterogeneous GPPO (HetGPPO) [2] are two models of communication extending IPPO [3] with GNNs. GPPO uses parameter sharing while HetGPPO uses independent parameters. In detail, the local sub-graph is built in a predefined manner: bidirectional edges appear if nodes are in range of communication. Then, any MPNN (Eq. 3) process only once to aggregate received messages. Finally, $[o_i^t \oplus h_i^{t,1}]$ are fed into an MLP, and the output serves both policy and critic. The authors argue that HetGPPO leads to better heterogeneous behaviors, while being more resilient in noisy environments.

Heterogeneous Policy Networks (HetNet) studies communication between physically heterogeneous agents [22]. The method posits that agents needs to learn different type of communication, depending on the class of agents. The graph is built in a predefined manner, like GPPO. Then, a custom MPNN called HetGAT learns different sets of weights for each class received messages. Concretely, if agent i receives a message x_j^t from an agent of a different class, it uses a dedicated attention weight α^{j2i}. If it receives a message x_k^t from an agent of the same class, it instead uses the attention weight α^{i2i}. Moreover, HetGAT accounts for bandwidth by learning to sample fixed bit size messages. HetNet stacks several HetGAT layers, with *multi-round communication* at each layer (cf. lines 12-14 of Algo. 2). Extending MAPPO, the final representation serves the policy during training and execution. For the centralized critic, the method uses a proxy as DICG to learn with a HetGAT on a complete graph. This method uses both a proxy and distributed communication (cf. lines 2 and 5 of Algo. 2), but the proxy is used only during the centralized training.

3.3 Communication constraints

To deploy MADRL with communication in real-world applications, several other constraints come into play, in particular concerning the communication. Indeed, in realistic scenarios, communication is subject to constraints and associated costs. Table 2 provides a summary of the communication constraints evaluated during execution and/or accounted for in state-of-the-art methods. The first constraint relates to connectivity, characterized by a limited communication range (CR) among agents. While this constraint is not compatible with proxy-based architectures, it is naturally incorporated in distributed communication methods, especially, multi-round communication provides a solution to overcome this limitation. Another constraint is the scalability of the method in terms of number of agents. Scaling to larger number of agents is easier in distributed communication, while a proxy communication suffers from greater complexity due to the size of the joint observation space. Another important aspect of realistic communica-

tion is the limited bandwidth (LB), which requires to optimally encode information when communication capacity is limited. Few works begin to focus on this, as does HetNet [22] or Bandwidth-constrained Variational Message Encoding (BVME) [6] very recently. Moreover, noisy messages (NM) and communication loss (CL) are yet to be fully taken into account. Few methods are resilient to noisy messages [2, 8].

Table 2: Tested communication constraints during execution

Methods	CR	$\max(n)$	LB	NM	CL
GA-Comm [13]		20			
MAGIC [18]		20			
GACG [4]		10			
LTS-CG [5]		27			
DGN [10]	×	60			
LSC [14]	×	60			
MAGE-X [29]	×	50			
MAGEC [8]	×	6		×	×
GPPO [2]	×	2			
HetGPPO [2]	×	2		×	
HetNet [22]	×	10	×		

To deploy communicating learning agents in real-world environment, integrating directly these constraints into the learning process of communication will enable the handling of more complex and realistic scenarios.

4 Conclusion

Communication has emerged as a major solution to overcome partial observability and non-stationarity problems of Multi-Agent Deep Reinforcement Learning (MADRL) algorithms. Nevertheless, communication methods must address several key design challenges to be effective: determining the content of transmitted messages, selecting appropriate recipients, interpreting and combining received information, and leveraging communication-aware representations for decision-making. Our survey highlights the use of Graph Neural Networks (GNNs) in the Multi-Agent Deep Reinforcement Learning (MADRL) literature as a principled approach to partially address these four challenges.

We propose a generic algorithm that generalizes the GNN-based communication processes, which presents how GNNs handle communication challenges. Our algorithm captures the diversity in methods using GNNs and permits a comparative study of existing approaches, emphasizing common and different structural elements and limitations. We have classified existing methods from two complementary perspectives: proxy communication and distributed communication. The proxy assumption enables the construction of a global graph, facilitating stronger coordination among agents. However, it relies on the assumption of perfect communication. In contrast, distributed communication supports local coordination within communication range, reflecting common practical constraints. In addition, we show the limitations of GNN-based communication methods in communication-constrained environments. Lastly, we observe that taking into account the temporal information and the structural information is a key point for better communication between agents. Encoding messages often uses an RNN to aggregate observations from the past, and its help to converge in a partially observable environment. However, the correlation of temporal and structural information, as LTS-CG leverages [5], can be a potential research direction.

The validation of the proposed generalization is intended to rely on the design of a unified framework enabling systematic comparison, reproducibility and empirical verification across existing and future methods.

Finally, future research should more explicitly address realistic constraints, as summarized in Table 2, to bridge the gap between theoretical communication models and practical applications of multi-agent systems.

References

[1] Stefano V. Albrecht, Filippos Christianos, and Lukas Schäfer. *"Multi-Agent Reinforcement Learning: Foundations and Modern Approaches"*. MIT Press, 2024.

[2] Matteo Bettini, Ajay Shankar, and Amanda Prorok. "Heterogeneous Multi-Robot Reinforcement Learning". In *AAMAS*, pages 1485–1494, 2023.

[3] Christian Schroeder De Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. "Is Independent Learning All You Need in the Starcraft Multi-Agent Challenge?". *arXiv preprint arXiv:2011.09533*, 2020.

[4] Wei Duan, Jie Lu, and Junyu Xuan. "Group-Aware Coordination Graph for Multi-Agent Reinforcement Learning". In *IJCAI*, 2024.

[5] Wei Duan, Jie Lu, and Junyu Xuan. "Inferring Latent Temporal Sparse Coordination Graph for Multiagent Reinforcement Learning". *IEEE Transactions on Neural Networks and Learning Systems*, 36(8):14358–14370, 2025-08.

[6] Wei Duan, Jie Lu, En Yu, and Junyu Xuan. "Bandwidth-Constrained Variational Message Encoding for Cooperative Multi-Agent Reinforcement Learning". In *AAMAS*, page 13, 2026.

[7] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. "Neural Message Passing for Quantum Chemistry". In *ICML*, pages 1263–1272, 2017.

[8] Anthony Goeckner, Yueyuan Sui, Nicolas Martinet, Xinliang Li, and Qi Zhu. "Graph Neural Network-Based Multi-Agent Reinforcement Learning for Resilient Distributed Coordination of Multi-Robot Systems". In *IROS*, pages 5732–5739, 2024.

[9] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In *NeurIPS*, volume 30, pages 1025–1035, 2017.

[10] Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. "Graph Convolutional Reinforcement Learning". In *ICLR*, 2020.

[11] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In *ICLR*, 2017.

[12] Sheng Li, Jayesh K. Gupta, Peter Morales, Ross Allen, and Mykel J. Kochenderfer. "Deep Implicit Coordination Graphs for Multi-Agent Reinforcement Learning". In *AAMAS*, pages 764–772, 2021.

[13] Yong Liu, Weixun Wang, Yujing Hu, Jianye Hao, Xingguo Chen, and Yang Gao. "Multi-Agent Game Abstraction via Graph Attention Neural Network". In *AAAI*, volume 34, pages 7211–7218, 2020.

[14] Zeyang Liu, Lipeng Wan, Xue Sui, Zhuoran Chen, Kewu Sun, and Xuguang Lan. "Deep Hierarchical Communication Graph in Multi-Agent Reinforcement Learning.". In *IJCAI*, pages 208–216, 2023.

[15] 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 *NeurIPS*, volume 30, pages 6382–6393, 2017.

[16] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous Methods for Deep Reinforcement Learning". In *ICML*, pages 1928–1937, 2016.

[17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing Atari With Deep Reinforcement Learning". *arXiv preprint arXiv:1312.5602*, 2013.

[18] Yaru Niu, Rohan Paleja, and Matthew Gombolay. "Multi-Agent Graph-Attention Communication and Teaming.". In *AAMAS*, pages 964–973, 2021.

[19] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. "Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning". *JMLR*, 21(178):1–51, 2020.

[20] 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.

[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms". *arXiv preprint arXiv:1707.06347*, 2017.

[22] Esmaeil Seraj, Zheyuan Wang, Rohan Paleja, Daniel Martin, Matthew Sklar, Anirudh Patel, and Matthew Gombolay. "Learning Efficient Diverse Communication for Cooperative Heterogeneous Teaming". In *AAMAS*, pages 1173–1182, 2022.

[23] Junjie Sheng, Xiangfeng Wang, Bo Jin, Junchi Yan, Wenhao Li, Tsung-Hui Chang, Jun Wang, and Hongyuan Zha. "Learning Structured Communication for Multi-Agent Reinforcement Learning". *JAAMAS*, 36(2):50, 2022.

[24] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. "Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward". In *AAMAS*, pages 2085–2087, 2018.

[25] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. "Multiagent Cooperation and Competition With Deep Reinforcement Learning". *PloS one*, 12(4):e0172395, 2017.

[26] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. "Graph Attention Networks". In *ICLR*, 2018.

[27] Christopher JCH Watkins and Peter Dayan. "Q-learning". *Machine learning*, 8(3):279–292, 1992.

[28] Ronald J Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". *Machine learning*, 8(3):229–256, 1992.

[29] Xinyi Yang, Shiyu Huang, Yiwen Sun, Yuxiang Yang, Chao Yu, Wei-Wei Tu, Huazhong Yang, and Yu Wang. "Learning Graph-Enhanced Commander-Executor for Multi-Agent Navigation". In *AAMAS*, pages 1652–1660, 2023.

[30] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. "The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games". *NeurIPS*, 35:24611–24624, 2022.

[31] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. "Graph Neural Networks: A Review of Methods and Applications". *AI Open*, 1:57–81, 2020.

[32] Changxi Zhu, Mehdi Dastani, and Shihan Wang. "A Survey of Multi-Agent Deep Reinforcement Learning With Communication". *JAAMAS*, 38(1), 2024.