Towards Open Ad Hoc Teamwork Using Graph-based Policy Learning

Arrasy Rahman
University of Edinburgh
arrasy.rahman@ed.ac.uk

Niklas Hopner
University of Amsterdam
n.r.hopner@uva.nl

Filippos Christianos
University of Edinburgh
f.christianos@ed.ac.uk

Stefano V. Albrecht
University of Edinburgh
s.albrecht@ed.ac.uk

ABSTRACT

Ad hoc teamwork is the challenging problem of designing an autonomous agent that can adapt quickly to collaborate with teammates without prior coordination mechanisms, including joint training. Prior work in this area has focused on closed teams in which the number of agents is fixed. In this work, we consider *open* teams by allowing agents with different fixed policies to enter and leave the environment without prior notification. Our solution builds on graph neural networks to learn agent and joint-action value models under varying team compositions. We contribute a novel action-value computation that integrates the agent and joint-action value model to produce action-value estimates. We empirically demonstrate that our approach successfully models the effects other agents have on the learner, which leads to policies that robustly adapt to dynamic team compositions and significantly outperform several alternative methods.

KEYWORDS

Ad Hoc Teamwork, Reinforcement Learning, Graph Neural Networks

1 INTRODUCTION

Many real-world problems require autonomous agents to perform tasks in the presence of other agents. Recent multi-agent reinforcement learning (MARL) approaches [e.g. 12, 14, 24, 30] solve such problems by jointly training a set of agents with shared learning procedures assuming knowledge of each agent's reward function. However, as agents become capable of long-term autonomy and are used for a growing number of tasks, it is possible that agents may have to interact with previously unknown other agents, without the opportunity for prior joint training via MARL. Thus, research in *ad hoc teamwork* [32] aims to design a single autonomous agent, which we refer to as the *learner*, that can interact effectively with other agents without pre-coordination such as joint training.

Prior ad hoc teamwork approaches [2, 6, 7, 11, 31] achieved this aim by combining single-agent RL and agent modeling techniques to learn from direct interaction with other agents. These approaches were designed for *closed teams* in which the number of agents is fixed. In practice, however, many tasks also require the learner to adapt to a changing number of agents in the environment. For example, consider an autonomous car that needs to drive differently depending on the number of nearby vehicles, which may be driven by humans or produced by different manufacturers, and their respective driving styles.

We make a step towards the full ad hoc teamwork challenge by considering *open teams* in which agents of varying fixed policies may enter and leave the team at any time and without prior notification. *Open ad hoc teamwork* involves three main challenges that must be addressed without pre-coordination. First, the learner must quickly adapt its policy to the unknown policies of other agents. Second, handling openness requires the learner to adapt to changing team sizes in addition to other agents' types, which may affect the policy and role a learner must adopt within the team [36]. Third, the changing number of agents results in a state vector of variable length, which causes standard RL approaches that require fixed-length state vectors to perform poorly, as we show in our experiments.

We propose a novel algorithm designed for open ad hoc teamwork, called *Graph-based Policy Learning* (GPL)[1], which addresses the aforementioned challenges. GPL adapts to dynamic teams by training a *joint action value model* which allows the learner to disentangle the effect each agent's action has on the learner's returns. To select optimal actions from the joint action value model, we contribute a novel action-value computation method which integrates joint-action value estimates with action predictions learned using an agent model. To handle dynamic team sizes, the joint action value model and agent model are both based on graph neural network (GNN) architectures [10, 35] which have proven useful for dealing with changing input sizes [18, 22]. Our computed action values can be used within different value-based single-agent RL algorithms; in our experiments we test two versions of GPL, one based on Q-learning [26] and one based on soft policy iteration [17].

Our experiments evaluate GPL and various baselines in three multi-agent environments (Level-based foraging [3], Wolfpack [23], FortAttack [13]) for which we use different processes to determine when agents enter or leave the environment and their type assignments. We compare GPL against ablations of GPL that integrate agent models using input concatenation, a common approach used by prior works [15, 35]; as well as two MARL approaches (MADDPG [24] and DGN [22]). Our results show that both tested GPL variants achieve significantly higher returns than all other baselines in most learning tasks, and that GPL generalizes more effectively to previously unseen team sizes/compositions. We also provide a detailed analysis of learned concepts within GPL's joint action value models.

[1]The implementation of GPL is provided in https://github.com/uoe-agents/GPL

2 RELATED WORK

Ad hoc teamwork: Ad hoc teamwork is at its core a single-agent learning problem in which the agent must learn a policy that is robust to different teammate types [32]. Early approaches focused on matrix games in which the teammate behavior was known [1, 33]. A predominant approach in ad hoc teamwork is to compute Bayesian posteriors over defined teammate types and utilizing the posteriors in reinforcement learning methods (such as Monte Carlo Tree Search) to obtain optimal responses [2, 6]. Recent methods applied deep learning-based techniques to handle switching agent types [31] and to pretrain and select policies for different teammate types [11]. All of these methods were designed for closed teams. In contrast, GPL is the first algorithm designed for open ad hoc teamwork in which agents of different types can dynamically enter and leave the team.

Agent modeling: An agent model takes a history of observations (e.g actions, states) as input and produces a prediction about the modeled agent, such as its goals or future actions [5]. Recent agent modeling frameworks explored in deep reinforcement learning [19, 28, 29] are designed for closed environments. In contrast, we consider open multi-agent environments in which the number of active agents and their policies can vary in time. Tacchetti et al. [35] proposed to use graph neural networks for modeling agent interactions in closed environments. Unlike GPL, their method uses predicted probabilities of future actions to augment the input into a policy network, which did not lead to higher final returns in their empirical evaluation and performs worse when generalizing to teams with different sizes, as we will show in our experiments.

Multi-agent reinforcement learning (MARL): MARL algorithms use RL techniques to co-train a set of agents in a multi-agent system [27]. In contrast, ad hoc teamwork focuses on training a single agent to interact with a set of agents of unknown types that are in control over their own actions. One approach in MARL is to learn factored action values to simplify the computation of optimal joint actions for agents, using agent-wise action values [30, 34] and coordination graphs (CG) [10, 40]. Unlike these methods which use CGs to model joint action values for fully cooperative setups, we use CGs in ad hoc teamwork to model the impact of other agents' actions towards the learning agent's returns. Jiang et al. [22] consider MARL in open systems by utilizing GNN-based architectures as value networks. In our experiments we use a baseline following their method and show that it performs significantly worse than GPL.

3 PROBLEM FORMULATION

The goal in open ad hoc teamwork is to train a learner agent to interact with other agents that have unknown behavioural models, called *types*, and which may enter or leave the environment at any timestep. We formalize the problem of open ad hoc teamwork by extending the Stochastic Bayesian Game model [2] to allow for openness.

3.1 Open Stochastic Bayesian Games (OSBG)

An OSBG is a tuple (N, S, A, Θ, R, P), where N,S,A,Θ represent the set of agents, the state space, the action space, and the type space, respectively. For simplicity we assume a common action space for all agents, but this can be generalised to individual action spaces for each agent. To define joint actions under variable number of agents, we define a joint agent-action space $\boldsymbol{A_N} = \{a | a \in \mathcal{P}(N \times A), \forall (i, a^i), (j, a^j) \in a : i = j \Rightarrow a^i = a^j\}$ and refer to the elements $a \in \boldsymbol{A_N}$ as *joint agent-actions*. Similarly, we define a joint agent-type space $\boldsymbol{\Theta_N} = \{\theta | \theta \in \mathcal{P}(N \times \Theta), \forall (i, \theta^i), (j, \theta^j) \in \theta : i = j \Rightarrow \theta^i = \theta^j\}$, refer to $\theta \in \boldsymbol{\Theta_N}$ as the *joint agent-type* and use θ^i to denote the type of agent i in θ. The conditions in the definition of $\boldsymbol{\Theta_N}$ and $\boldsymbol{A_N}$ constrain each agent to only select one action while also being assigned to one type. Each agent follows its policy π_{θ_i}.

We assume the learner can observe the current state of the environment and the past actions of other agents but not their types. The learner's reward is determined by $R : S \times \boldsymbol{A_N} \mapsto \mathbb{R}$. Assuming Δ to be the set of all possible probability distributions over a random variable, the transition function $P : S \times \boldsymbol{A_N} \mapsto \Delta(S \times \boldsymbol{\Theta_N})$ determines the probability of the next state and joint agent-types given the current state and agent types. Although the transition function allows agents to have changing types, our current work assumes that an agent's type is fixed between entering and leaving the environment.

Under an OSBG, the game starts by sampling an initial state, s_0, and an initial set of agents, N_0, with associated types, θ_0^i with $i \in N_0$, from a starting distribution $P_0 \in \Delta(S \times \boldsymbol{\Theta_N})$. At state s_t, agents $N_t \subseteq N$ with types $\theta_t^i, i \in N_t$ exist in the environment and choose their actions a_t^i by sampling from π_{θ_i}. As a consequence of the selected joint agent-actions, the learner receives a reward computed through R. Finally, the next state s_{t+1} and the next set of existing agents N_{t+1} and types are sampled from P given s_t and joint agent-actions.

3.2 Optimal Policy for OSBG

Assuming that i denotes the learning agent, the learning objective in an OSBG is to estimate the optimal policy defined below:

Definition 1. Let the joint actions and the joint policy of agents other than i at time t be denoted by a_t^{-i} and $\boldsymbol{\pi}_t^{-i}$, respectively. Given a discount factor, $0 \leq \gamma \leq 1$, we define the action-value of policy π^i , $\bar{Q}_{\pi^i}(s, a^i)$, as :

$$\mathbb{E}_{a_t^i \sim \pi^i, a_t^{-i} \sim \boldsymbol{\pi}_t^{-i}, P}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,\middle|\, s_0 = s, a_0^i = a^i\right], \tag{1}$$

which denotes the expected discounted return after the learner executes a^i at s. A policy, $\pi^{i,*}$, is optimal if :

$$\forall \pi^i, s, a^i, \bar{Q}_{\pi^{i,*}}(s, a^i) \geq \bar{Q}_{\pi^i}(s, a^i). \tag{2}$$

Given $\bar{Q}_{\pi^{i,*}}(s, a)$, an OSBG is solved by always choosing a^i with the highest action-value at s.

4 GRAPH-BASED POLICY LEARNING

We introduce the general components of GPL and their respective role for estimating an OSBG's optimal policy. We also describe the neural network architectures and learning procedures used for implementing each component. A general overview of GPL's architecture is provided in Figure 1 while the complete learning pseudocode is given in Appendix[2] F.

[2]Appendix can be downloaded from http://bit.ly/37F2BJz

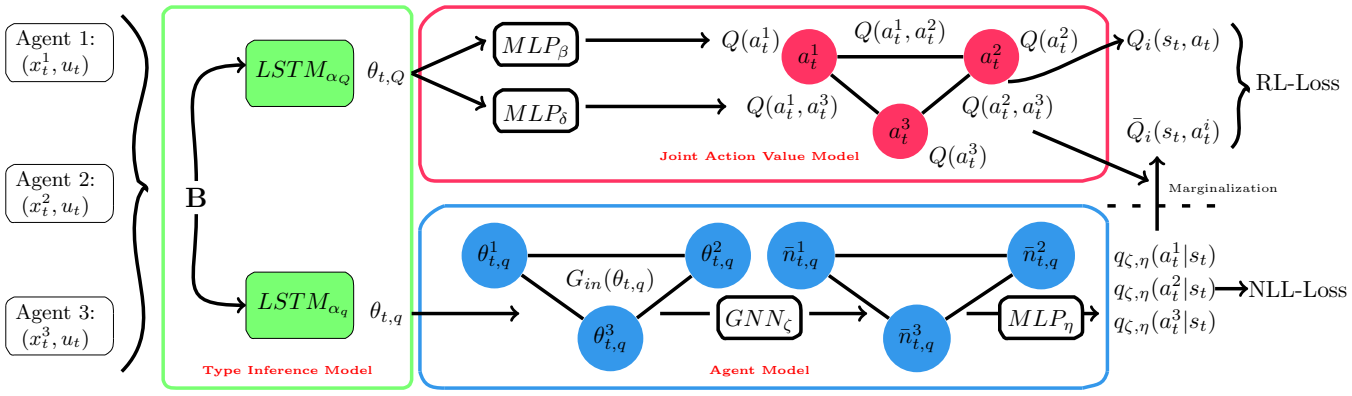


Figure 1: Overview of GPL. The joint action value model (red box) and agent model (blue box) receive type vectors produced by their own type embedding networks (green box), which are parameterized by α_Q and α_q respectively. These type vectors are processed by the joint-action value and agent model that are parameterized by (β, δ) and (ζ, η) respectively. The outputs from the joint action value model and agent model are combined via Equation 7 to compute the action value. Gradients are not backpropagated through the dashed line.

4.1 Method Overview and Motivation

In an OSBG, agents' joint actions inherently affect the learner's return through the rewards and next states it experiences. A learner using common value-based RL methods such as Q-Learning [39] will always update the action-values of the learner's previous action, even if that action had minimal impact towards the observed reward from an OSBG. In MARL, a similar credit assignment problem is commonly addressed by using a centralized critic [14, 24] that disentangles the effects of other agents' actions to a learner's return by estimating a joint-action value function. Inspired by the importance of joint-action value modeling for credit assignment, GPL includes a component for *joint-action value* estimation. The joint-action value of a policy, $Q_{\pi^i}(s, a)$, is defined as:

$$\mathbb{E}_{a_t^i \sim \pi^i, a_t^{-i} \sim \boldsymbol{\pi}^{-i}, P}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,\middle|\, s_0 = s, a_0 = a\right]. \tag{3}$$

In contrast to Equation (1), this joint-action value denotes the learner's expected return after the joint agent-action a at s. Modeling joint-action values prevents the learner from only crediting its own action if it has minimal contribution towards rewards that it experienced.

Using a joint-action value model for ad hoc teamwork introduces problems for training the model and action selection. Due to the inherent assumption of not knowing other agents' policies, MARL methods' TD error computation for training the joint-action value model, which assumes other jointly trained agents will follow their known policies at the next state, cannot be applied in ad hoc teamwork. Furthermore, being optimistic by choosing a^i from the joint agent-action with maximum value yields suboptimal policies for ad hoc teamwork since other agents may choose actions that do not maximize its returns.

To enable joint-action value training and action selection for ad hoc teamwork, we prove the following Equation:

$$\bar{Q}_{\pi^i}(s_t, a_t^i) = \mathbb{E}_{a_t^{-i} \sim \boldsymbol{\pi}_t^{-i}(.|s_t, a_t^i, \theta^{-i})}[Q_{\pi^i}(s_t, a)|a^i = a_t^i], \tag{4}$$

in Appendix A to show that GPL's joint-action value estimates can be used to compute $\bar{Q}^{\pi^i}(s_t, a_t^i)$, which facilitates greedy action selection and enables TD error computation for training the joint-action value model. Since $\pi_t^{-i}(a_t^{-i}|s_t, a_t^i, \theta^{-i})$ in Equation (4) is unknown to the learner, it is estimated by GPL's *agent modelling* component described in Section 4.2.

GPL's last component is the *type inference* component. In an OSBG, types affect $Q(s, a)$ and $\pi^{-i}(a^{-i}|s, a^i, \theta^{-i})$ by determining other agent's immediate and future actions. Estimation of $Q(s, a)$ and $\pi^{-i}(a^{-i}|s, a^i, \theta^{-i})$ must therefore take agent types as input. However, agent types are unknown and must be inferred from agents' observed behavior, which we detail in Section 4.2.

The aforementioned GPL components are finally implemented using neural networks that facilitate an efficient computation of $\bar{Q}^{\pi^{*,i}}(s, a^i)$ by imposing a simple factorization of the joint action value based on coordination graphs [16]. Furthermore, environment openness is handled by implementing the joint-action value and agent modeling components with GNNs, which we describe in the next section. This produces a flexible way of computing $\bar{Q}^{\pi^{*,i}}(s, a^i)$ for any team size.

4.2 The GPL Architecture

In this section, we outline the neural network architectures used for implementing the three components of GPL (cf. Figure 1). This is followed by a description of the loss functions for training each component's neural network.

Type inference: In the absence of knowledge over the inherent type space of an OSBG, GPL assumes that types can be represented as fixed-length vectors. Since type inference requires reasoning over agent's behavior over an extended period of time, LSTMs [20] are used for the implementation. The LSTM takes the observation (u_t) and agent-specific information (x_t^i), which are both derived from s_t, to produce a hidden-state vector as an agent's type embedding. Full details on the computation of type vectors can be

found in Appendix D. GPL uses the type embeddings as input for the joint action value and agent modeling network. Although the joint action value and agent modeling feature may use the same type inference network, separate networks are used to prevent both model's gradients from interfering against each other during training.

Joint action value estimation: The learner's joint action value is represented as a fully connected Coordination Graph [16] (CGs). A fully connected CG factorizes the joint action value as:

$$Q_{\beta,\delta}(s_t, a_t) = \sum_{j \in N_t} Q^j_\beta(a^j_t|s_t) + \sum_{\substack{j,k \in N_t \\ j \neq k}} Q^{j,k}_\delta(a^j_t, a^k_t|s_t), \quad (5)$$

with $Q^j(a^j|s)$ and $Q^{j,k}(a^j, a^k|s)$ being the singular and pairwise utility terms respectively. The singular and pairwise utility terms are computed using fully connected networks parameterized by β and δ respectively. When computing a utility term, both networks receive type embeddings of the learner and agents associated to the terms as input, which enables each term to model the associated agent's contribution towards the learner's returns. Details on the computation of the utility terms are provided in Appendix E.

We represent our joint action value as CGs for three reasons. First, CGs impose a joint action value factorization that facilitates an efficient action-value computation. We further elaborate the computational complexity of computing action-values with CGs when discussing about GPL's action-value computation. Second, CGs are implementable as GNNs [10], which makes it a good fit for handling openness. Third, CG's value factorization enables GPL to model the contribution of other agents' actions towards the returns. Specifically, $Q^j(a^j|s)$ estimates the contribution of agent j while $Q^{j,k}(a^j, a^k|s)$ estimates the contribution of agent j and k as a pair.

Agent modeling: GPL uses the Relational Forward Model (RFM) architecture [9] for agent modeling. RFM are a class of recurrent graph neural networks that have demonstrated good performance for agent modeling in open teams. Assuming ζ denotes its parameters, the RFM network receives agent type embeddings, θ_q, as its node input to compute a fixed-length embedding, $\bar{n}$, for each agent. Assuming a^j is the action taken by j in the joint agent-action a^{-i}, we use each agent's updated embedding to approximate $\pi^{-i}(a^{-i}|s, a^i)$ as:

$$q_{\eta,\zeta}(a^{-i}|s, a^i) \approx \prod_{j \in -i} \text{Softmax}(MLP_\eta(\bar{n}_j))(a^j), \quad (6)$$

with η being the parameter of an MLP that transforms the updated agent embeddings. Note that we approximate $q_{\eta,\zeta}(a^{-i}|s, a^i)$ with a distribution where a^i is independent of a^{-i} given s because in ad hoc teamwork the learner cannot pre-coordinate its action with teammates and all agents execute their actions at s simultaneously.

Action value computation: Evaluating Equation (4) can be inefficient in larger teams. For instance, a team of k agents which may choose from n possible actions requires the evaluation of n^k joint-action terms, which number grows exponentially with the increase in team size. By contrast, a more efficient action-value computation arises from factorizing the joint action value network and using RFM-based agent modeling networks. We prove in Appendix B that substituting the joint-action value and agent models from Equation (5) and (6) into Equation (4) for action-value computation enables us to express GPL's action-value using the following expression :

$$\begin{aligned} \bar{Q}(s_t, a^i) = {} & Q^i_\beta(a^i|s_t) \\ & + \sum_{a^j \in A_j, j \neq i} \left(Q^j_\beta(a^j|s_t) + Q^{i,j}_\delta(a^i, a^j|s_t)\right) q_{\zeta,\eta}(a^j|s_t) \\ & + \sum_{a^j \in A_j, a^k \in A_k, j,k \neq i} Q^{j,k}_\delta(a^j, a^k|s_t) q_{\zeta,\eta}(a^j|s_t) q_{\zeta,\eta}(a^k|s_t). \end{aligned} \quad (7)$$

Unlike Equation (4), Equation (7) is defined in terms of singular and pairwise action terms. In this case, the number of terms that need to be computed only increases quadratically as the team size increases. The computation of the required terms can be efficiently done in parallel with existing GNN libraries [38].

Model optimization: As the learner interacts with teammates during learning, it stores a dataset of states, agents' actions, and rewards that it observed. Given a dataset of other agents' actions it collected at different states, $\{(s_t, a_t)\}^D_{t=1}$, the agent modeling network is trained to estimate $\pi(a^{-i}_t|s_t, a^i_t)$ through supervised learning by minimizing the negative log likelihood loss defined below:

$$L_{\zeta,\eta} = -\log(q_{\zeta,\eta}(a^{-i}_t|s_t, a^i_t)). \quad (8)$$

On the other hand, the collected dataset is also used to update GPL's joint-action value network using value-based reinforcement learning. Unlike standard value-based approaches [26], we use the joint action value as the predicted value. The loss function for the joint action value network is then defined as:

$$L_{\beta,\delta} = \frac{1}{2}\left(Q_{\beta,\delta}(s_t, a_t) - y(r_t, s_{t+1})\right)^2, \quad (9)$$

with $y(r_t, s_{t+1})$ being a target value which computation depends on the algorithm being used. We subsequently train GPL with Q-Learning (GPL-Q) [39] and Soft-Policy Iteration (GPL-SPI) [17], which produces a greedy and stochastic policy respectively. The target value computations of both methods are defined as the following:

$$\begin{aligned} y_{\text{QL}}(r_t, s_{t+1}) &= r_t + \gamma \max_{a^i} \bar{Q}\left(s_{t+1}, a^i\right), \\ y_{\text{SPI}}(r_t, s_{t+1}) &= r_t + \gamma \sum_{a^i} p_{\text{SPI}}(a^i|s_{t+1}) \bar{Q}\left(s_{t+1}, a^i\right), \end{aligned}$$

where GPL-SPI's policy uses the Boltzmann distribution,

$$p_{\text{SPI}}(a^i_t|s_t) \propto \exp\left(\frac{\bar{Q}(s_t, a^i)}{\tau}\right), \quad (10)$$

with τ being the temperature parameter.

5 EXPERIMENTAL EVALUATION

In this section, we describe our open ad hoc teamwork experiments and demonstrate GPL's performance in them, followed by a detailed analysis of concepts learned by GPL's joint action value model.

5.1 Multi-Agent Environments

We conduct experiments in three fully observable multi-agent environments with different game complexity:

Models	GNN	Agent Model	Joint Action-Value
QL			
QL-AM		✓	
GNN	✓		
GNN-AM	✓	✓	
GPL-Q	✓	✓	✓
GPL-SPI	✓	✓	✓

Table 1: Types of ablations based on their value network architecture and their use of agent modelling.

Level-based foraging (LBF): In LBF [3], agents and objects with levels $l \in \{1, 2, 3\}$ are spread in a 8×8 grid world. The agents' goal is to collect all objects. Agent actions include actions to move along the four cardinal directions, stay still, or to collect objects in adjacent grid locations. An object is collected if the sum of the levels of all agents involved in collecting at the same time is equal to or higher than the level of the object. Upon collecting an object, every agent that collects an object is given a reward equal to the level of the object. An episode finishes if all available objects are collected or after 50 timesteps.

Wolfpack: In Wolfpack [23], a team of hunter agents must capture moving prey in a 10×10 grid world. Episodes consist of 200 timesteps and prey are trained to avoid capture using DQN [26]. While the agents have full observability of the environment, prey only observe a limited patch of grid cells ahead of them. Agents in this environment can move along the four cardinal directions or stay still at their current location. To capture a prey, at least two hunters must form a pack by where every pack members is located next to a prey's grid location. Every hunter in a pack that captured a prey is given a reward of two times the size of the capturing pack. However, we penalize agents by -0.5 for positioning themselves next to a prey without teammates positioned in other adjacent grids from the prey. Prey are respawned after they are captured.

FortAttack: FortAttack [13] is situated on a two-dimensional plane where a team of attackers aim to reach a region, which we refer to as the fort, defended by defenders whose aim is to prevent any attackers from reaching the fort. Our learning agent assumes the role of a defender. Agents are equipped with actions to move along the four cardinal directions, rotate, and shoot any opposing team members located in a triangular shooting range defined by the agent's angular orientation and location. An episode ends when either an attacker reaches the fort, the learner is shot by attackers, or 200 timesteps have elapsed. The learner receives a reward of -3 for getting destroyed and 3 for destroying an attacker. On the other hand, a reward of -10 is given when an attacker reaches the fort and a reward of 10 when guards manage to defend the fort for 200 timesteps. A cost of -0.1 is also given for shooting.

5.2 Baselines

We design different learners, which can be categorized into single-agent value-based RL and MARL-based learners, to compare against GPL. Note that baselines that do not use GNNs require fixed-length inputs. To enable these approaches to handle changing number of agents in their observations, we impose an upper limit on the number of agents in the environment and preprocess the observation to ensure a fixed-length input by adding placeholder values. Details of this preprocessing method is provided in Appendix G.2.

Single-agent RL baselines: In line with GPL, all baselines are trained with synchronous Q-learning [25] and take as input the type vectors from the type inference network, but differ in their action-value computation and their use of an agent model. **QL** takes the concatenation of type vectors as input into a feedforward network to estimate action values. **GNN** applies multi-head attention [22] to the type vectors and predicts action values based on the learner's node embedding. The agent model used by **QL-AM** and **GNN-AM** is identical in architecture and training procedure to GPL's agent model. However, the predicted action probabilities are concatenated to the individual agent representations x_t as explored by prior methods [35]. An overview of baselines and their components can be found in Table 1. In alignment to the work of Huang et al. [21], comparing methods with and without GNNs will enable us to investigate the advantage GNNs provide in training and generalization performance. On the other hand, the different ways GPL and baselines use action probabilities will also provide insight on how GPL's method of combining action probabilities to compute action values compares to prior methods.

MARL baselines: While in principle our ad hoc teamwork setting precludes joint training of agents via MARL (we only control a single learner agent and may also not know rewards of other agents), we use MARL approaches by assuming that (during training) we control all teammates and all teammates are using the same reward function as the learner. We compare with two MARL algorithms: MADDPG [24] and DGN [22]. MADDPG is a MARL algorithm for closed environments, while DGN is a GNN-based MARL approach designed for joint training in open environments. For evaluation in open ad hoc teamwork, after MARL training completes, we select one of the jointly trained agents and measure its performance when interacting with the teammate types used in our ad hoc teamwork settings (see Sec. 5.3).

5.3 Experimental Setup

We construct environments for open ad hoc teamwork by creating a diverse set of teammate types for each environment. Agent types are designed such that a learner must adapt its policy to achieve optimal return when interacting with the different types. In LBF and Wolfpack, each type's policy is implemented either via different heuristics or reinforcement learning-based policies. We vary the teammate policies in terms of their efficiency in executing a task and their roles in a team. Further details of the teammate policies and diversity analysis for Wolfpack and LBF are provided in Appendix C.4. We use pretrained policies provided by Deka and Sycara [13] for FortAttack.

In our experiments, openness is simulated by creating an open process that determines how agents enter and leave during episodes, for both training and testing. To demonstrate GPL's ability to solve open ad hoc teamwork across different open processes, we utilize these two open processes in our experiments :

Open process with changing team sizes. In LBF and Wolfpack, this type of open process determines the number of timesteps an agent can exist in the environment by uniformly sampling from a certain range of integers. After staying for the predetermined

number of timesteps, agents are removed from the environment by the open process. For FortAttack, agents are removed once they are shot by opponents. After being removed, agents can reenter the environment after a specific period of waiting time. This open process also determines the type of an agent entering an environment by uniformly sampling from all available types. Further details of the open process for each environments are provided in Appendix G.1.

To evaluate generalization capability in terms of number of agents, this open process impose different limits to the maximum team size for training and testing. For testing, we increase the upper limit on team size to expose the learner against team configurations it has never encountered before. Specifically, in all environments we set the maximum team size to 3 agents during training while increasing it to 5 agents during testing. Note that for FortAttack where there are two teams competing against each other, this team size restriction applies for both the attacking and defending team.

Open process with fixed team sizes. We also demonstrate GPL's performance in an open process where the team always consists of two agents. In LBF and Wolfpack, this open process decides the type of the learner's teammate by uniformly sampling from all available types at the start of the episode. Once the episode starts, the teammate is never removed while no new teammates are introduced to the environment. Similar to LBF and Wolfpack, the type of the learner's teammate and the two opposing attackers in FortAttack are decided by uniformly sampling from all available types at the start of the episode. Once an agent is shot by the opposition, it will immediately be reintroduced to the environment in the next time step. Hence, under this open process the type and number of agents will never change in the middle of an episode for FortAttack. Like the evaluation for the previous open process, evaluation is done by increasing the team size at the start of the episode. However, the type and number of teammates will remain fixed even during evaluation.

5.4 Open Ad Hoc Teamwork Results

Figure 2 shows the training performance of GPL-Q, GPL-SPI, and the baselines. This figure shows that MARL-based approaches produce similar or worse performance than our worst performing single-agent RL baseline during training. While MARL policies performs better alongside other jointly trained agents, it generalizes poorly against the ad hoc teamwork teammates that cannot be jointly trained with MARL. For completeness, we show MARL learner's improved performance when interacting with other jointly trained agents in Appendix H.

Figure 2 also shows that GPL-based approaches significantly outperform other baselines that use agent models, such as QL-AM and GNN-AM, in terms of training performance. Despite both being based on GNNs, GPL outperforming GNN-AM highlights GPL's action-value computation method over GNN-AM. As further indicated by the similarity in performance between QL/QL-AM or GNN/GNN-AM, concatenating action probabilities towards observations also does not improve training performance in most cases, which aligns with previous results from Grover et al. [15] and Tacchetti et al. [35]. The reason GPL significantly outperforms others in training is because the joint-action model learns to disentangle the effects of other agents' actions, which we will further elaborate in Section 5.5.

The generalization performance of GPL and baselines are provided by Table 2. The way GPL-Q, GPL-SPI, GNN-QL and GNN-QL-AM outperform other baselines in generalization for LBF despite having similar training performances shows that GNNs are important components for generalizing between different open processes. Furthermore, GPL-Q and GPL-SPI outperforming GNN-QL-AM's generalization capability shows that using agent models for action-value computation using Equation (7) also plays a role in improving generalization capability between open processes. In QL-AM and GNN-AM, the value networks must learn a model that integrates the predicted action probabilities to compute good action-value estimates, which may not generalize well to teams with previously unseen sizes. By contrast, GPL uses predicted action probabilities as weights in its action-value computation following Equation (7), which is proven in Appendix B to be correct for any team size.

To further demonstrate GPL's superior performance for open ad hoc teamwork, we finally compare GPL-Q's performance to the single-agent RL baselines in the open process with fixed team sizes mentioned in Section 5.3. GPL-SPI is omitted from this comparison since its performance is similar to GPL-Q in all environments under open processes where the team size changes. On the other hand, MARL baselines are excluded from this analysis since the single-agent RL baselines perform better than them under the previous open process. From Figure 3, we observed similar findings to what we observed in Figure 2 and Table 2. GPL still outperforms the single-agent RL baselines in terms of training performance. Furthermore, GPL-Q, GNN-AM, and GNN also outperforms baselines that are not equipped with GNNs in terms of generalization.

5.5 Joint Action Value Analysis

We investigate how the joint action value model enables GPL-Q to significantly outperform the single-agent RL baselines during training in FortAttack, which is our most complex environment. For completeness, a similar analysis for Wolfpack is provided in Appendix J.

When comparing the resulting behavior from learning with GPL-Q and baselines, Figure 4a shows that GPL-Q's shooting accuracy improves at a faster rate than baselines and eventually converges at a higher value. Investigating the way GPL components encourage faster and better shooting performance may therefore highlight the reason why GPL-based approaches outperform the baselines. We specifically investigate the joint-action value model by defining several shooting-related metrics derived from GPL's CG, average their values over 480000 sample states gathered at different training checkpoints, and measure their correlation coefficient with GPL's average return. Among all metrics, the highest Pearson correlation coefficient of 0.85 is attained by $\bar{Q}_{j,k}$ when j is a defender and k is an attacker in j's shooting range. $\bar{Q}_{j,k}$ is specifically defined as :

$$\bar{Q}_{j,k} = \frac{\sum_{a^k} Q_\delta^{j,k}(a^j = \text{shoot}, a^k|s)}{|A^k|}. \tag{11}$$

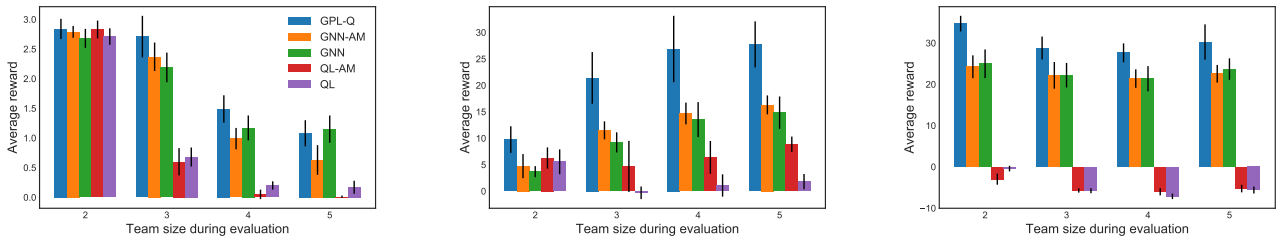
This metric is derived from GPL's pairwise utility terms and can be viewed as GPL's estimate of agent j's average contribution towards the learner when j decides to shoot k, averaged over all possible

(a) Results in Level-based foraging **(b) Results in Wolfpack** **(c) Results in FortAttack**

Figure 2: Open ad hoc teamwork results (training): Average and 95% confidence bounds of GPL & baseline returns during training (up to 3 agents in a team for LBF, Wolfpack, and attacker & defender teams in FortAttack). For each algorithm, training is done using eight different seeds and the resulting models are saved and evaluated every 160000 global steps.

Environment \ Algorithm	GPL-Q	GPL-SPI	QL	QL-AM	GNN	GNN-AM	MADDPG	DGN
LBF	**2.32±0.22**	**2.40±0.16***	1.41±0.14	1.22±0.29	2.07±0.13	1.80±0.11	0.64 ± 0.90	0.91 ± 0.10
Wolfpack	**36.36±1.71***	**37.61±1.69***	20.57±1.95	14.24±2.65	8.88±1.57	30.87±0.95	2.18 ± 0.66	19.20 ± 2.22
FortAttack	**14.20±2.42***	**16.82±1.92***	-3.51±0.60	-3.51±1.51	7.01±1.63	8.12±0.74	-5.98 ± 0.82	-4.83 ± 1.24

Table 2: Open ad hoc teamwork results (testing): Average and 95% confidence bounds of GPL and baselines during testing (up to 5 agents in a team for LBF, Wolfpack, and attacker & defender teams in FortAttack). For each algorithm, data was gathered by running the greedy policy resulting from the eight value networks stored at the checkpoint which achieved the highest average performance during training. The asterisk indicates significant difference in returns compared to the single-agent RL baselines.



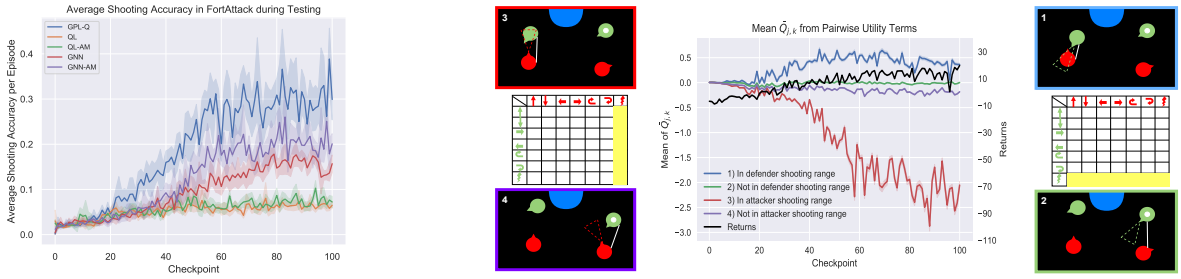
(a) Results in level-based foraging **(b) Results in Wolfpack** **(c) Results in FortAttack**

Figure 3: Team size generalization results: Average rewards and 95% confidence bounds of GPL and baselines from team size generalization experiments collected over eight seeds. Agent was trained to interact in a team of two agents for LBF, Wolfpack, and FortAttack. With FortAttack, we used a setup where the number of attackers was always equal to the number of teammate defenders. Value networks are stored every 160000 steps and the performance of greedy policies from value networks stored at the checkpoint with the highest average performance during training were used to compute the average returns and their 95% confidence interval. This result shows that GPL-Q still outperforms other single-agent RL baselines, except for LBF where its performance is not significantly different from GNN and GNN-AM.

a^k. Therefore, GPL-Q's return strongly correlates with the pairwise utility terms computed by MLP_δ of the joint-action value model when a defender chooses to shoot an attacker inside its shooting range.

Unlike GPL that is equipped with a pairwise utility model (MLP_δ), learning to shoot with the single-agent RL baselines requires increasing the value network's estimate on shooting when an attacker ventures inside the learner's shooting range. However, this can be difficult to do when the learner is exploring since it requires the learner to position itself at the right distance and orientation from an attacker. Even if the learner manages to get to the right distance from an attacker, a trained attacker can shoot a suboptimal learner instead if it does not orient itself properly or if it does not shoot the attacker first.

We now highlight how MLP_δ is the primary reason behind GPL-based learner's ability to learn shooting faster. Consider the case



(a) Shooting accuracy in FortAttack

(b) Evolution of shooting metrics derived from GPL's pairwise utilites.

Figure 4: Shooting-related metrics for FortAttack: (a) For GPL-Q and the baselines, we measure the percentage of times the learner successfully shot an attacker at each checkpoint during FortAttack training (cf. Section 5.3). (b) We measure $\bar{Q}_{j,k}$, which is a metric derived from GPL-Q's pairwise utility terms that represents GPL-Q's estimate of the contribution towards the returns resulting from agent j shooting an opponent agent, k. Each line in the plot corresponds to a different scenario in which the pairwise utility term is measured. Lines 1 and 2 represent $\bar{Q}_{j,k}$ when j is a defender and k is an attacker inside (1) or outside (2) j's shooting range. Lines 3 and 4 contrast the value of $\bar{Q}_{j,k}$ when j is an attacker and k is a defender inside (3) or outside (4) j's shooting range. To provide an example pairwise interaction where $\bar{Q}_{j,k}$ is computed from for each line, we visualize four sample pairwise interactions in FortAttack (white line in black boxes). Each black box is numbered after the line plot it corresponds to. The fort is represented by the blue half circle, attackers by red circles, defenders by green circles, the learner is marked with a white dot, and shooting ranges are indicated with dashed view cones. The matrices represent the joint action space for an attacker and defender, where the yellow marked fields refer to the actions that are averaged over to compute $\bar{Q}_{j,k}$ shown in the middle plot. This figure shows that the learner becomes increasingly aware of the benefits of shooting attackers inside a defender's shooting range and the negative consequences of a defender approaching an attacker's shooting range as training progresses, which causes GPL-Q's strong performance.

of increasing the pairwise utility terms associated to shooting attackers inside a defender's shooting range. For a learner equipped with MLP_δ, this can be done using value-based learning once the learner observes a defender successfully shooting an attacker. The learner will increase the joint-action value associated to shooting through bootstrapping once it sees many examples where successfully shooting attackers results in advantageous states where the attacker is removed from the environment. When the teammate defenders are well-trained, instances of successful shots from teammates occurs more frequently than instances with successful shots from a learner under exploration. Therefore, the joint-action value associated to shooting is more often increased than the baselines' action-value of shooting.

Increasing the aforementioned shooting-related pairwise utility terms indirectly encourages the learner to shoot attackers. Once MLP_δ increases its pairwise utility terms associated with shooting attackers inside a defender's shooting range, MLP_δ will also increase the shooting-related pairwise utility terms when an attacker is inside the learner's shooting range. This happens due to applying the same pairwise utility model to any two agents and the fact that the learner itself is a defender. The learner is eventually encouraged to get attackers inside its shooting range and shoot more, which yields the strong performances we see from GPL-Q and GPL-SPI.

Aside from learning the value of shooting attackers inside a defender's shooting range, Figure 4b also shows that MLP_δ learns to associate negative values when defenders enter an attacker's shooting range. Using the aforementioned mechanism, this enables the learner to learn to avoid the shooting range of attackers. Therefore, GPL's MLP_δ indirectly enables the learner to acquire useful skills demonstrated by other agents in FortAttack. We show in Appendix K that baselines that are not equipped with a joint action value model are not be able to learn how to shoot attackers. This leads to their significantly worse performances compared to GPL-based methods.

6 CONCLUSION

This work addresses the challenging problem of open ad hoc teamwork, in which the goal is to design an autonomous agent capable of robust teamwork under dynamically changing team composition without pre-coordination mechanisms such as joint training. Our proposed algorithm GPL uses coordination graphs to learn joint action-value functions that model the effects of other agents' actions towards the learning agent's returns, along with a GNN-based model trained to predict actions of other teammates. We empirically tested our approach in three multi-agent environments showing that our learned policies can robustly adapt to dynamically changing teams. We empirically show that GPL's success can be attributed to its ability to learn meaningful concepts to explain the effects of other agents' actions on the learning agent's returns. This enables GPL to produce action-values that lead to significantly better training and generalization performances than various baselines. An interesting direction for future work is to automatically learn the appropriate value factorization from observations to handle situations in which the inherent return structure does not follow CG's factorization method. Furthermore, extensions to environments with partial observability and/or continuous actions are another direction.

REFERENCES

[1] Noa Agmon and Peter Stone. 2012. Leading ad hoc agents in joint action settings with multiple teammates.. In *AAMAS*. 341–348.

[2] Stefano V Albrecht, Jacob W Crandall, and Subramanian Ramamoorthy. 2016. Belief and truth in hypothesised behaviours. *Artificial Intelligence* 235 (2016), 63–94.

[3] Stefano V. Albrecht and Subramanian Ramamoorthy. 2013. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*. 1155–1156.

[4] Stefano V. Albrecht and Peter Stone. 2017. Reasoning about Hypothetical Agent Behaviours and their Parameters. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems*. 547–555.

[5] Stefano V Albrecht and Peter Stone. 2018. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence* 258 (2018), 66–95.

[6] Samuel Barrett, Avi Rosenfeld, Sarit Kraus, and Peter Stone. 2017. Making friends on the fly: Cooperating with new teammates. *Artificial Intelligence* 242 (2017), 132–171.

[7] Samuel Barrett and Peter Stone. 2015. Cooperating with Unknown Teammates in Complex Domains: A Robot Soccer Case Study of Ad Hoc Teamwork. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence* (Austin, Texas, USA).

[8] Samuel Barrett, Peter Stone, and Sarit Kraus. 2011. Empirical Evaluation of Ad Hoc Teamwork in the Pursuit Domain. In *Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)* (Taipei, Taiwan).

[9] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. 2018. Relational inductive biases, deep learning, and graph networks. *CoRR* (2018).

[10] Wendelin Böhmer, Vitaly Kurin, and Shimon Whiteson. 2019. Deep Coordination Graphs. *CoRR* (2019).

[11] Shuo Chen, Ewa Andrejczuk, Zhiguang Cao, and Jie Zhang. 2020. AATEAM: Achieving the Ad Hoc Teamwork by Employing the Attention Mechanism. In *AAAI*. 7095–7102.

[12] Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. 2020. Shared Experience Actor-Critic for Multi-Agent Reinforcement Learning. In *34th Conference on Neural Information Processing Systems*.

[13] Ankur Deka and Katia Sycara. 2020. Natural Emergence of Heterogeneous Strategies in Artificially Intelligent Competitive Teams. *arXiv preprint arXiv:2007.03102* (2020).

[14] JN Foerster, G Farquhar, T Afouras, N Nardelli, and SA Whiteson. 2018. Counterfactual Multi-agent Policy Gradient. 2974–2982.

[15] Aditya Grover, Maruan Al-Shedivat, Jayesh Gupta, Yuri Burda, and Harrison Edwards. 2018. Learning Policy Representations in Multiagent Systems. In *International Conference on Machine Learning*. 1802–1811.

[16] Carlos Guestrin, Michail G Lagoudakis, and Ronald Parr. 2002. Coordinated reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*. 227–234.

[17] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290* (2018).

[18] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*. 1024–1034.

[19] He He, Jordan Boyd-Graber, Kevin Kwok, and Hal Daumé III. 2016. Opponent modeling in deep reinforcement learning. In *International Conference on Machine Learning*. 1804–1813.

[20] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[21] Wenlong Huang, Igor Mordatch, and Deepak Pathak. 2020. One policy to control them all: Shared modular policies for agent-agnostic control. In *International Conference on Machine Learning*. PMLR, 4455–4464.

[22] Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. 2019. Graph Convolutional Reinforcement Learning. In *International Conference on Learning Representations*.

[23] Joel Z. Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. 2017. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. 464–473.

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

[25] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[27] Georgios Papoudakis, Filippos Christianos, Arrasy Rahman, and Stefano V Albrecht. 2019. Dealing with Non-Stationarity in Multi-Agent Deep Reinforcement Learning. *arXiv preprint arXiv:1906.04737* (2019).

[28] Neil C. Rabinowitz, Frank Perbet, H. Francis Song, Chiyuan Zhang, S. M. Ali Eslami, and Matthew Botvinick. 2018. Machine theory of mind. In *Proceedings of the 35th International Conference on Machine Learning*. 4215–4224.

[29] Roberta Raileanu, Emily Denton, Arthur Szlam, and Rob Fergus. 2018. Modeling others using oneself in multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*. 4254–4263.

[30] Tabish Rashid, Mikayel Samvelyan, Christian Schroder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. 2018. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*. 4292–4301.

[31] Manish Ravula, Shani Alkobi, and Peter Stone. 2019. Ad hoc Teamwork with Behavior Switching Agents. In *International Joint Conference on Artificial Intelligence (IJCAI)* (Macau, China).

[32] Peter Stone, Gal A Kaminka, Sarit Kraus, Jeffrey S Rosenschein, et al. 2010. Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination.. In *AAAI*. 6.

[33] Peter Stone, Gal A Kaminka, and Jeffrey S Rosenschein. 2009. Leading a best-response teammate in an ad hoc team. In *Agent-mediated electronic commerce. Designing trading strategies and mechanisms for electronic markets*. Springer, 132–146.

[34] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinícius Flores Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. 2018. Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. 2085–2087.

[35] Andrea Tacchetti, H. Francis Song, Pedro A. M. Mediano, Vinícius Flores Zambaldi, János Kramár, Neil C. Rabinowitz, Thore Graepel, Matthew Botvinick, and Peter W. Battaglia. 2019. Relational forward models for multi-agent learning. In *7th International Conference on Learning Representations*.

[36] Milind Tambe. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research* 7 (1997), 83–124.

[37] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. 2017. Multiagent cooperation and competition with deep reinforcement learning. *PloS one* 12, 4 (2017).

[38] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander J Smola, and Zheng Zhang. 2019. Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs. *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019).

[39] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.

[40] Ming Zhou, Yong Chen, Ying Wen, Yaodong Yang, Yufeng Su, Weinan Zhang, Dell Zhang, and Jun Wang. 2019. Factorized Q-learning for large-scale multi-agent systems. In *Proceedings of the First International Conference on Distributed Artificial Intelligence*. 1–7.