Towards Open Ad Hoc Teamwork Using Graph-based Policy Learning

Arrasy Rahman [1] **Niklas Höpner** [2] **Filippos Christianos** [1] **Stefano V. Albrecht** [1]

Abstract

Ad hoc teamwork is the challenging problem of designing an autonomous agent which 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 models and joint-action value models under varying team compositions. We contribute a novel action-value computation that integrates the agent model 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, leading to policies that robustly adapt to dynamic team compositions and significantly outperform several alternative methods.

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. Christianos et al., 2020; Foerster et al., 2018; Rashid et al., 2018; Lowe et al., 2017) solve such problems by jointly training a set of agents with shared learning procedures. 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. Thus, research in *ad hoc teamwork* (Stone et al., 2010) 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.

[1]School of Informatics, University of Edinburgh, Edinburgh, United Kingdom [2]University of Amsterdam, Amsterdam, Netherlands. Correspondence to: Arrasy Rahman <arrasy.rahman@ed.ac.uk>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

Prior ad hoc teamwork approaches (Barrett & Stone, 2015; Albrecht et al., 2016; Barrett et al., 2017; Ravula et al., 2019; Chen et al., 2020) 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 (Albrecht et al., 2021).

We make a step towards the full ad hoc teamwork challenge by considering *open teams* in which agents with various 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 (Tambe, 1997). 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) [3], 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 (Tacchetti et al., 2019; Böhmer et al., 2020) which have proven useful for dealing with changing input sizes (Hamilton et al., 2017;

[3]Implementation code can be found at https://github.com/uoe-agents/GPL

Jiang et al., 2019). 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 (Mnih et al., 2015) and one based on soft policy iteration (Haarnoja et al., 2018).

Our experiments evaluate GPL and various baselines in three multi-agent environments (Level-based foraging (Albrecht & Ramamoorthy, 2013), Wolfpack (Leibo et al., 2017), FortAttack (Deka & Sycara, 2020)) for which we use different processes to specify 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 (Grover et al., 2018; Tacchetti et al., 2019); as well as two MARL approaches (MADDPG (Lowe et al., 2017), DGN (Jiang et al., 2019)). 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.

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 (Stone et al., 2010). Early approaches focused on matrix games in which the teammate behavior was known (Agmon & Stone, 2012; Stone et al., 2009). 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 (Barrett et al., 2017; Albrecht et al., 2016). Recent methods applied deep learning-based techniques to handle switching agent types (Ravula et al., 2019) and to pretrain and select policies for different teammate types (Chen et al., 2020). 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 (Albrecht & Stone, 2018). Recent agent modeling frameworks explored in deep reinforcement learning (Raileanu et al., 2018; Rabinowitz et al., 2018; He et al., 2016) 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. (2019) 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. Our experiments also show that their method leads to worse generalization to teams with different sizes.

Multi-agent reinforcement learning (MARL): MARL algorithms use RL techniques to co-train a set of agents in a multi-agent system (Papoudakis et al., 2019). 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 (Rashid et al., 2018; Sunehag et al., 2018) and coordination graphs (CG) (Böhmer et al., 2020; Zhou et al., 2019). 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. (2019) 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 (Albrecht et al., 2016) to allow for openness. The current formulation focuses on fully observable environments and extending the framework to the partially observable case is left as future work.

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. Let $\mathcal{P}$ denote the power set. 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.

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 $R : S \times \boldsymbol{A_N} \mapsto \mathbb{R}$. 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 joint agent-actions, where $\Delta(X)$ is the set of all probability distributions over X. 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. 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 D.

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 (Watkins & Dayan, 1992) 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 (Lowe et al., 2017; Foerster et al., 2018) 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.

Unlike centralized training for MARL (Lowe et al., 2017) where the TD-error for training a joint-action value model can be approximated by using other agents' known policies, in ad hoc teamwork using joint-action value models introduces problems in computing optimal actions and temporal difference errors. This results from the assumption in ad hoc teamwork of not knowing other agents' policies during training and execution. Therefore, the uncertainty in other agents' actions must be accounted for in the action value computation in ad hoc teamwork.

To perform joint-action value training and action selection for ad hoc teamwork, we approximate the learner's action value function $\bar{Q}_{\pi^i}$ by simultaneously learning a factorized joint action value model and an agent model that that approximates $\pi_t^{-i}(a_t^{-i}|s_t, \theta^{-i})$ (see Section 4.2). We can then approximate the expectation in Equation 1 by weighting the action value components with the likelihood of the corresponding agents' actions following :

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

Other strategies to deal with the uncertainty in other agents actions are possible. Being optimistic one can choose a^i from the joint agent-action with maximum value, which can yield in suboptimal policies for ad hoc teamwork since other agents may choose actions that do not maximize the learners returns. Given an agent model that predicts agents next actions one could also sample other agents actions and compute an average joint action value for each of the learners actions. During learning of the action value model, one can even use information of joint actions taken at previous state to compute the TD-error following SARSA (Rummery & Niranjan, 1994). However, these sampling-based

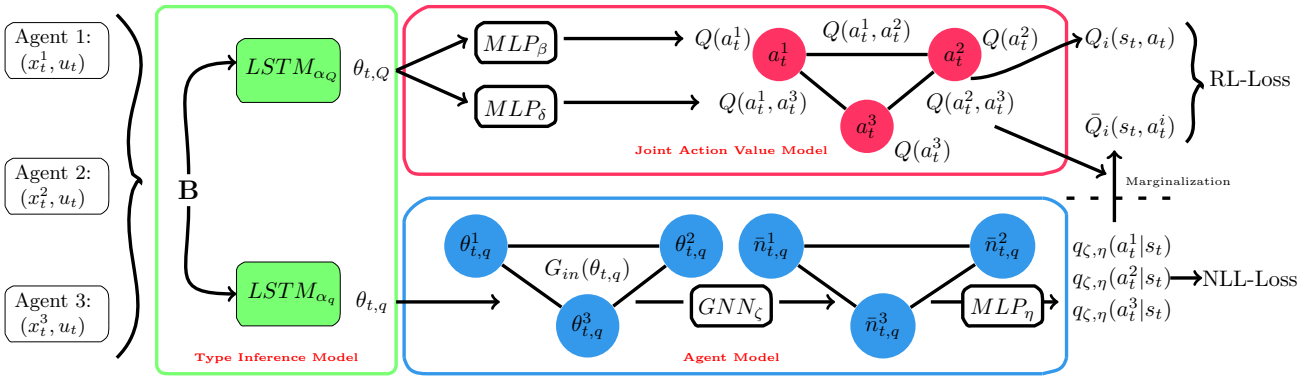


Figure 1. **Overview of GPL.** The joint action value model (red box) and agent model (blue box) receive type vectors produced by their own type embedding networks (green box), which are parameterized by α_Q and α_q respectively. These type vectors are processed by the joint-action value and agent model that are parameterized by (β, δ) and (μ, ν) respectively. Output from joint action value model and agent model is finally combined using Equation 9 to compute the action-value function, using which the learner chooses its action.

approaches may increase the variance of the action value estimates, which may decrease learner's performance.

GPL's last component is the *type inference* component. In an OSBG, types affect $Q(s, a)$ and $\pi^{-i}(a^{-i}|s, \theta^{-i})$ by determining other agent's immediate and future actions. Estimation of $Q(s, a)$ and $\pi^{-i}(a^{-i}|s, \theta^{-i})$ 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 (Guestrin et al., 2002). 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 sizes.

4.2. The GPL Architecture

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

Type inference: Without knowledge of the type space of an OSBG, GPL assumes types can be represented as fixed-length vectors. Since type inference requires reasoning over agent's behavior over an extended period of time, we use LSTMs (Hochreiter & Schmidhuber, 1997) for type inference. Our usage of LSTMs for type inference aligns with previous approaches for creating fixed-length embeddings of agents (Grover et al., 2018; Rabinowitz et al., 2018).

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. GPL then 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. Further details on the preprocessing method to derive u_t and x_t^i from s_t, along with the computation of type vectors are provided in Appendix C.

Joint action value estimation: We implement GPL's joint action value model as fully connected Coordination Graphs (Guestrin et al., 2002) (CGs) for three reasons. First, CGs factorize the joint action value in a way that facilitates efficient action-value computation, which we will elaborate further when discussing GPL's action-value computation. Second, CGs can be implemented as GNNs (Böhmer et al., 2020), which we demonstrate in Section 5.5 to be important for handling environment openness. Third, the aforementioned action value factorization also enables CGs to model the contribution of other agents' actions towards the learner's returns, which we show in Section 5.6 to be the main reason behind GPL's superior performance compared to baselines.

Given a state, a fully connected CG factorizes the learner's joint action value as a sum of individual utility terms, $Q_\beta^j(a_t^j|s_t)$, and pairwise utility terms, $Q_\delta^{j,k}(a_t^j, a_t^k|s_t)$, under the following Equation:

$$Q_{\beta,\delta}(s_t, a_t) = \sum_{j \in N_t} Q_\beta^j(a_t^j|s_t) + \sum_{\substack{j,k \in N_t \\ j \neq k}} Q_\delta^{j,k}(a_t^j, a_t^k|s_t). \tag{5}$$

$Q_\beta^j(a^j|s)$ intuitively represents j's contribution towards the

learner's returns by executing a^j, while $Q_\delta^{j,k}(a^j, a^k|s)$ denotes j and k's contribution towards the learner's returns by jointly choosing a^j and a^k.

We implement $Q_\beta^j(a_t^j|s_t)$ and $Q_\delta^{j,k}(a_t^j, a_t^k|s_t)$ as multilayer perceptrons (MLPs) parameterized by β and δ to enable generalization across states. We then enable MLP_β and MLP_δ to model the contribution of agents' individual and pairwise joint actions towards the learner's returns by providing type vectors of agents associated to each utility term, along with the the learner's type vector as input to MLP_β and MLP_δ. Given type vectors, θ_t^i and θ_t^j, MLP_β outputs a vector with a length of $|A|$ that estimates $Q_\beta^j(a^j|s_t)$ for each possible actions of j following:

$$Q_\beta^j(a^j|s_t) = \text{MLP}_\beta(\theta_t^j, \theta_t^i)(a^j). \tag{6}$$

On the other hand, instead of outputting the pairwise utility for the $|A| \times |A|$ possible pairwise actions of agent j and k, MLP_δ outputs an $K \times |A|$ matrix ($K \ll |A|$) given its type vector inputs. Assuming a low-rank factorization of the pairwise utility terms, the output of MLP_δ is subsequently used to compute $Q_\delta^{j,k}(a_t^j, a_t^k|s_t)$ following:

$$Q_\delta^{j,k}(a_t^j, a_t^k|s_t) = (\text{MLP}_\delta(\theta_t^j, \theta_t^i)^\intercal \text{MLP}_\delta(\theta_t^k, \theta_t^i))(a_t^j, a_t^k). \tag{7}$$

Previous work from Zhou et al. (2019) demonstrated that low-rank factorization enables scalable pairwise utility computation even under thousands of possible pairwise actions. Finally note that MLP_β and MLP_δ are shared between agents to encourage knowledge reuse for utility term computation, which importance to GPL's superior performance in open ad hoc teamwork is demonstrated in Section 5.6.

Agent modeling: GPL's agent model assumes that other agents choose their actions independently, but models the effect other agents have on an agent's actions by using the Relational Forward Model (RFM) architecture (Tacchetti et al., 2019). RFMs are a class of recurrent graph neural networks that have demonstrated high accuracy in predicting agents' next actions (Tacchetti et al., 2019). The RFM receives agent type embeddings, θ_q, as its node input to compute a fixed-length embedding, $\bar{n}$, for each agent and is parameterized by ζ. Let a^j be the action taken by agent j in the joint other agent-action a^{-i}, we use each agent's updated embedding to approximate $\pi^{-i}(a^{-i}|s, \theta_{-i})$ as:

$$\begin{aligned} q_{\zeta,\eta}(a^{-i}|s) &= \prod_{j \in -i} q_{\zeta,\eta,}(a^j|s), \\ q_{\zeta,\eta,}(a^j|s) &= \text{Softmax}(\text{MLP}_\eta(\bar{n}_j))(a^j), \end{aligned} \tag{8}$$

with η being the parameter of an MLP that transforms the updated agent embeddings.

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. By substituting the joint-action value and agent models from Equation (5) and (8) into Equation (4), we obtain Equation (9) as our action-value estimate, which we prove in Appendix A:

$$\begin{aligned} \bar{Q}(s_t, a^i) = {} & Q_\beta^i(a^i|s_t) \\ & + \sum_{a^j \in A_j, j \neq i} \left(Q_\beta^j(a^j|s_t) + Q_\delta^{i,j}(a^i, a^j|s_t)\right) q_{\zeta,\eta}(a^j|s_t) \\ & + \sum_{a^j \in A_j, a^k \in A_k, j,k \neq i} Q_\delta^{j,k}(a^j, a^k|s_t) q_{\zeta,\eta}(a^j|s_t) q_{\zeta,\eta}(a^k|s_t). \end{aligned} \tag{9}$$

Unlike Equation (4), Equation (9) 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 (Wang et al., 2019).

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, $\{(s_t, a_t)\}_{t=1}^D$, the agent modeling network is trained to estimate $\pi(a_t^{-i}|s_t, a_t^i)$ through supervised learning by minimizing the negative log likelihood loss defined below:

$$L_{\zeta,\eta} = -\log(q_{\zeta,\eta}(a_t^{-i}|s_t)). \tag{10}$$

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 (Mnih et al., 2015), we use the joint action value as the predicted value. The loss function for the joint action value network is then defined as:

$$L_{\beta,\delta} = \frac{1}{2}\left(Q_{\beta,\delta}\left(s_t, a_t\right) - y\left(r_t, s_{t+1}\right)\right)^2, \tag{11}$$

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) (Watkins & Dayan, 1992) and Soft-Policy Iteration (GPL-SPI) (Haarnoja et al., 2018), which produces a greedy and stochastic policy respectively. The target value computations of both methods are defined as the following:

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

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

$$p_{\text{SPI}}(a_t^i|s_t) \propto \exp\left(\frac{\bar{Q}(s_t, a^i)}{\tau}\right), \tag{12}$$

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. This is 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:

Level-based foraging (LBF): In LBF (Albrecht & Ramamoorthy, 2013), 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 (Leibo et al., 2017), 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 (Mnih et al., 2015). 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 (Deka & Sycara, 2020) 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 E.2.

Single-agent RL baselines: In line with GPL, all baselines are trained with synchronous Q-learning (Mnih et al., 2016) 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 (Jiang et al., 2019) 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 (Tacchetti et al., 2019). An overview of baselines and their components can be found in Table 1. These baselines will allow us to investigate what advantage GNNs provide in training and generalization performance, when action probabilities help learning, which method of integrating action probabilities is most useful, and how GPL's approach to computing 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 (Lowe et al., 2017) and DGN (Jiang et al., 2019). 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. 5.4).

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

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

5.3. Data Collection for Training

While type inference requires reasoning over consecutive observations, we want to avoid increasing the storage and computational resources required for training GPL and baselines. Therefore, we collect experiences across multiple environments in parallel instead of using a single environment and storing experiences in an experience replay. This resembles the data collection process of A3C (Mnih et al., 2016). However, we collect experiences synchronously as opposed to A3C's asynchronous data collection process.

5.4. 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 type. 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 B.4. We use pretrained policies provided by Deka & Sycara (2020) for FortAttack.

We simulate openness by creating an open process that determines how agents enter and leave during episodes, for both training and testing. In LBF and Wolfpack, the number of timesteps an agent can exist in the environment is determined by uniformly sampling from a certain range of integers. After staying for the predetermined number of timesteps, agents are removed from the environment. 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. The type of an agent entering an environment is uniformly sampled from all available types. Further details of the open process for each environments are provided in Appendix E.1.

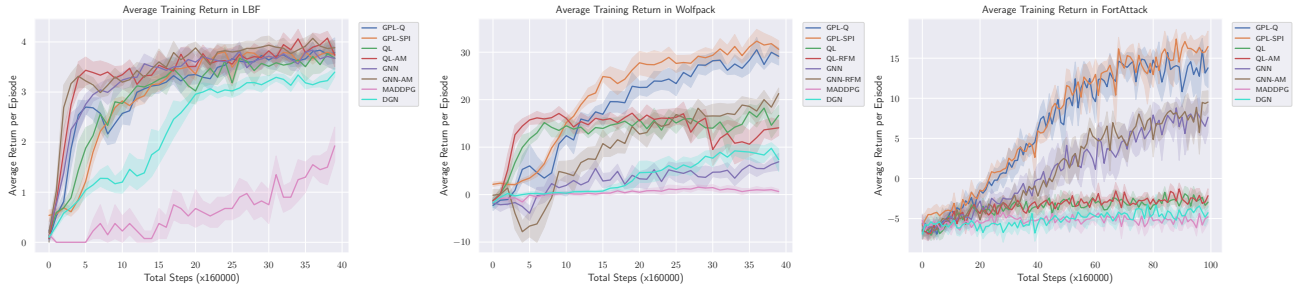
After every 160000 global training timesteps, GPL and baselines' are stored and evaluated based on their achieved return under the open process. To evaluate generalization capability in terms of number of agents, we impose different limits to the maximum team size resulting from the open process for 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. In all three environments, we specifically limit the team size to three agents during training time and increase this limit to five agents for testing. Since FortAttack has two opposing teams in its environment, these team size restriction is imposed on both teams.

5.5. Open Ad Hoc Teamwork Results

Figure 2 shows the training performance of GPL-based approaches and the baselines. It 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 F.

Since results between GPL-Q and GPL-SPI are similar, we use GPL to refer to both in comparison with baselines. Figure 2 also shows that GPL significantly outperforms 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. (2018) and Tacchetti et al. (2019). The reason GPL significantly outperforms others in training is because the joint-action model learns to disentangle the effects of other agents' actions. We further elaborate on GPL's superiority over other methods in terms of training performance through an analysis over its resulting joint action value estimates which we provide in Section 5.6.

The generalization performance of GPL and the baselines is provided in Table 2. The way GPL, GNN and GNN-AM outperform single-agent RL baselines in generalization for LBF despite having similar training performances shows that GNNs are important components for generalizing between different open processes. Furthermore, GPL outperforming GNN-AM's generalization capability shows that using agent models for action-value computation using Equation (9) 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



(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.

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.

Env.	GPL-Q	GPL-SPI	QL	QL-AM	GNN	GNN-AM	DGN	MADDPG
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 ± 0.9	0.91 ± 0.10
Wolf.	**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
Fort.	**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

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 (9), which is proven in Appendix A to be correct for any team size. Finally, the low generalization performance of MARL-based baselines naturally follows from their low performance during training.

5.6. Joint Action Value Analysis

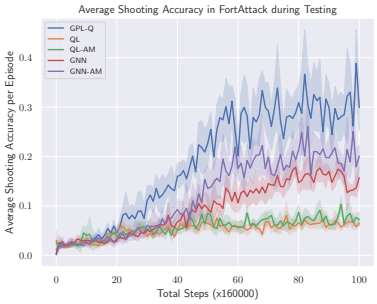
We investigate how the joint action value model enables GPL-Q to significantly outperform the single-agent RL baselines during training in FortAttack, which is our most complex environment. For completeness, a similar analysis for Wolfpack is provided in Appendix H.

When comparing the resulting behavior from learning with GPL-Q and baselines, Figure 3a 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 why GPL-based approaches outperform the baselines. We specifically investigate several shooting-related metrics derived from GPL's component, 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:

$$\bar{Q}_{j,k} = \frac{\sum_{a^k} Q_\delta^{j,k}(a^j = \text{shoot}, a^k|s)}{|A^k|}. \tag{13}$$

$\bar{Q}_{j,k}$ is derived from GPL's pairwise utility terms, $Q_\delta^{j,k}(a^j, a^k|s)$, 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^k. Therefore, this shows that GPL-Q's return strongly correlates with the pairwise utility terms assigned by the joint-action value model when a defender chooses to shoot an attacker inside its shooting range.

Rather than merely being strongly correlated to GPL's returns, we now elaborate why the pairwise utility terms produced by the joint-action value model is the main reason behind GPL-based learner's higher final returns. Consider when MLP_δ increases its pairwise utility terms associated with shooting attackers inside a defender's shooting range. Since the learner itself is a defender, MLP_δ will also increase values of shooting-related pairwise utility terms when an attacker is inside the learner's shooting range. This encourages the learner to get attackers inside its shooting range and shoot more. Eventually, the learner achieves a higher return and $\bar{Q}_{j,k}$ becomes strongly correlated with the learner's return when j is a defender and k is an attacker inside j's shooting range. Furthermore, Figure 3b also shows that MLP_δ learns to associate negative values when defenders enter an attacker's shooting range, which enables the learner to learn to avoid the shooting range of attackers.



(a) Shooting accuracy in FortAttack

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

Figure 3. **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 5.4). **(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.

Unlike GPL, we show in Appendix I that baselines that are not equipped with MLP_δ will not be able to learn these important concepts, which lead to their significantly worse performances. Specifically, 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. In turn, agents can only increase the action value estimate of shooting after experiencing the positive rewards from successfully shooting an opposing team's agent. However, this can be especially difficult during exploration 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.

6. Conclusion and Future Work

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 related to extensions towards environments with partial observability and/or continuous action spaces. On the other hand, GPL's restrictive assumption that the learner's joint action value function must factorize following a fully connected CG can degrade the learner's returns for environments with certain reward functions (Castellini et al., 2019). Thus, automatically learning the most appropriate joint action value factorization through CG graph structure learning can potentially improve GPL's computational efficiency and return estimates.

Acknowledgement

This research was financially supported in part by: the University of Edinburgh Enlightenment Scholarship (A.R.); the Hybrid Intelligence Center, funded by the Netherlands Organisation for Scientific Research under grant number 024.004.022 (N.H.); the UK EPSRC Centre for Doctoral Training in Robotics and Autonomous Systems (F.C.); and the US Office of Naval Research (ONR) grant N00014-20-1-2390 (S.A).

References

Agmon, N. and Stone, P. Leading ad hoc agents in joint action settings with multiple teammates. In *Proc. of 12th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, June 2012.

Albrecht, S. V. and Ramamoorthy, S. 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*, pp. 1155–1156, 2013.

Albrecht, S. V. and Stone, P. Reasoning about hypothetical agent behaviours and their parameters. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems*, pp. 547–555, 2017.

Albrecht, S. V. and Stone, P. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, 2018.

Albrecht, S. V., Crandall, J. W., and Ramamoorthy, S. Belief and truth in hypothesised behaviours. *Artificial Intelligence*, 235:63–94, 2016.

Albrecht, S. V., Brewitt, C., Wilhelm, J., Gyevnar, B., Eiras, F., Dobre, M., and Ramamoorthy, S. Interpretable goal-based prediction and planning for autonomous driving. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

Barrett, S. and Stone, P. 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*, January 2015.

Barrett, S., Stone, P., and Kraus, S. Empirical evaluation of ad hoc teamwork in the pursuit domain. In *Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2011.

Barrett, S., Rosenfeld, A., Kraus, S., and Stone, P. Making friends on the fly: Cooperating with new teammates. *Artificial Intelligence*, 242:132–171, 2017.

Böhmer, W., Kurin, V., and Whiteson, S. Deep coordination graphs. In *International Conference on Machine Learning*, pp. 980–991. PMLR, 2020.

Castellini, J., Oliehoek, F. A., Savani, R., and Whiteson, S. The representational capacity of action-value networks for multi-agent reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1862–1864, 2019.

Chen, S., Andrejczuk, E., Cao, Z., and Zhang, J. Aateam: Achieving the ad hoc teamwork by employing the attention mechanism. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7095–7102, 2020.

Christianos, F., Schäfer, L., and Albrecht, S. V. Shared experience actor-critic for multi-agent reinforcement learning. In *34th Conference on Neural Information Processing Systems*, 2020.

Deka, A. and Sycara, K. Natural emergence of heterogeneous strategies in artificially intelligent competitive teams. *arXiv preprint arXiv:2007.03102*, 2020.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Grover, A., Al-Shedivat, M., Gupta, J., Burda, Y., and Edwards, H. Learning policy representations in multiagent systems. In *International Conference on Machine Learning*, pp. 1802–1811, 2018.

Guestrin, C., Lagoudakis, M. G., and Parr, R. Coordinated reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 227–234, 2002.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pp. 1861–1870. PMLR, 2018.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pp. 1024–1034, 2017.

He, H., Boyd-Graber, J., Kwok, K., and Daumé III, H. Opponent modeling in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1804–1813, 2016.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jiang, J., Dun, C., Huang, T., and Lu, Z. Graph convolutional reinforcement learning. In *International Conference on Learning Representations*, 2019.

Leibo, J. Z., Zambaldi, V., Lanctot, M., Marecki, J., and Graepel, T. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pp. 464–473, 2017.

Lowe, R., Wu, Y. I., Tamar, A., Harb, J., Abbeel, O. P., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in*

neural information processing systems, pp. 6379–6390, 2017.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Papoudakis, G., Christianos, F., Rahman, A., and Albrecht, S. V. Dealing with non-stationarity in multi-agent deep reinforcement learning. *arXiv preprint arXiv:1906.04737*, 2019.

Rabinowitz, N. C., Perbet, F., Song, H. F., Zhang, C., Eslami, S. M. A., and Botvinick, M. Machine theory of mind. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 4215–4224, 2018.

Raileanu, R., Denton, E., Szlam, A., and Fergus, R. Modeling others using oneself in multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 4254–4263, 2018.

Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J. N., and Whiteson, S. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 4292–4301, 2018.

Ravula, M., Alkobi, S., and Stone, P. Ad hoc teamwork with behavior switching agents. In *International Joint Conference on Artificial Intelligence (IJCAI)*, August 2019.

Rummery, G. A. and Niranjan, M. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

Stone, P., Kaminka, G. A., and Rosenschein, J. S. Leading a best-response teammate in an ad hoc team. In *Agent-mediated electronic commerce. Designing trading strategies and mechanisms for electronic markets*, pp. 132–146. Springer, 2009.

Stone, P., Kaminka, G., Kraus, S., and Rosenschein, J. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, 2010.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V. F., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., and Graepel, T. 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*, pp. 2085–2087, 2018.

Tacchetti, A., Song, H. F., Mediano, P. A. M., Zambaldi, V. F., Kramár, J., Rabinowitz, N. C., Graepel, T., Botvinick, M., and Battaglia, P. W. Relational forward models for multi-agent learning. In *7th International Conference on Learning Representations*, 2019.

Tambe, M. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., and Vicente, R. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4), 2017.

Wang, M., Yu, L., Zheng, D., Gan, Q., Gai, Y., Ye, Z., Li, M., Zhou, J., Huang, Q., Ma, C., Huang, Z., Guo, Q., Zhang, H., Lin, H., Zhao, J., Li, J., Smola, A. J., and Zhang, Z. Deep graph library: Towards efficient and scalable deep learning on graphs. *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Watkins, C. J. and Dayan, P. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Zhou, M., Chen, Y., Wen, Y., Yang, Y., Su, Y., Zhang, W., Zhang, D., and Wang, J. Factorized q-learning for large-scale multi-agent systems. In *Proceedings of the First International Conference on Distributed Artificial Intelligence*, pp. 1–7, 2019.