Ad hoc teamwork with unknown task model

Gonçalo Rodrigues Alberto Sardinha Francisco S. Melo

ABSTRACT

We address the problem of ad hoc teamwork, where an agent must coordinate with other agents in an common task but without a pre-defined coordination mechanism. We focus on settings where the task and world model are unknown, and the agent must learn this task, learn its teammate models and adapt to them. We propose a new approach that combines model-based reinforcement learning with Monte Carlo tree search to enable an ad hoc agent to learn the underlying task and coordinate with its teammates. Our results show that our approach is competitive against state-of-the-art approaches that rely on a model of the task.

1. INTRODUCTION

The challenge of developing autonomous agents that are capable of cooperatively engaging with other unknown agents in a joint common task is known as the *ad hoc teamwork problem* [21]. Ad hoc teamwork was first introduced in the work of Stone et al. [21] and can be regarded as both an extension and an instantiation of the multi-agent reinforcement learning problem [18, 19].

In ad hoc teamwork, an agent (the "ad hoc agent") is paired with a group of unknown teammates (the "legacy" agents) in a common joint task. The ad hoc agent has no prior knowledge regarding the teammates; in particular, it does not know how the teammates act (i.e., their policy) and no communication or coordination protocol is in place that can be used by the ad hoc agent to determine such policy beforehand. As such, the ad hoc agent must identify, in runtime, the teammate's policy and adjust its own behavior in accordance.

In this paper, we follow Melo and Sardinha [17], and assume that the ad hoc teamwork problem, in its most general formulation, comprises three distinct sub-problems: (i) identify the target task; (ii) identify the teammate's policy; (iii) plan and act accordingly. In light of this subdivision, the ad hoc teamwork literature can roughly be organized according to which of the three problems are addressed. For example, earlier works on ad hoc teamwork focused on (iii), assuming that the task and behavior of the teammates was known [7, 22]. More recent work addressed (ii) and (iii) or even (i)-(iii), instead considering different assumptions regarding what type of information is available to the ad hoc agent and how such information impacts its learning process [9, 17].

Our approach considers the ad hoc teamwork problem in full, addressing (i)-(iii) above. Our setup is similar to that considered by Barrett and Stone [4, 6], in that the agent has available a reinforcement signal from the environment that guides its identification of the target task. In this sense, the ad hoc teamwork problem can be seen as an instantiation of a multi-agent reinforcement learning problem. We then use model-based reinforcement learning to learn the dynamics of the environment and, simultaneously, construct a model of the teammates. We then use Monte Carlo tree search (MCTS) with the learned model to plan the agent's actions.

1.1 Related Work

Research relevant to ad hoc teamwork can be surveyed in light of the three subproblems outlined above—task identification, teammate identification and planning.

Earlier work in ad hoc teamwork assumes that the task and teammate behavior are known, and focuses on the problem of planning a proper course of action for the ad hoc agent that optimizes the performance of the team as a whole. For example, Stone et al. [20] address the problem of an ad hoc agent that must lead a greedy teammate (that always selects the action with largest empirical average reward) to the optimal joint action in a finite-horizon decision problem. The agents select their actions in turns. Barrett and Stone [2] extended this approach to discounted infinite horizon problems, while Agmon and Stone [1] considers the situation where the ad hoc agent is paired against N teammates.

Along this same line of work, Brafman and Tennenholtz [11] consider a variation of the same problem, where the agents must select their actions simultaneously. In this variant of the ad hoc teamwork problem, the ad hoc agent is assumed to know how the actions of the teammate depend on the history of interaction. Stone et al. [22] consider the situation where the ad hoc agent is paired with a bounded-memory teammate following an ε-greedy policy. Unfortunately, the complexity of their proposed planning algorithm grows exponentially with the size of the memory of the teammate. Barrett et al. [8] explores the use of communication in the same setting, and show that using decision theoretic models such as MDPs to describe the planning problem of the ad hoc agent can lead to more efficient solutions.

In all aforementioned work, the ad hoc agent has full knowledge both of the target task and the decision-making process of the teammates, and ad hoc teamwork essentially reduces to a problem of *planning*. Towards a more general ad hoc teamwork setting, Chakraborty and Stone [12] consider the situation where the teammates are assumed to follow a Markov policy—generally unknown. In other words, at each moment t, the action of the teammates depends only on the value of a number of features at time $t-1$. By combining a number of alternative MDP models, the proposed approach

is able to identify the behavior of the teammates and use it to plan the ad hoc agent's actions. The more general setup considered by [12] goes one step beyond those previously discussed in that it addresses both *teammate identification* and planning. In this broader perspective, ad hoc teamwork is closely related to the problem of *learning in games*, for which an extensive literature exists [13].

Along the same line, Barrett et al. proposed PLASTIC-Model [6], that relies on a library of teammate models and Monte Carlo tree search (MCTS) planning [15] to address the ad hoc teamwork problem in the well known pursuit domain. In the pursuit domain, four predators must coordinate to capture a prey moving in a grid toroidal world [10]. It provides an excellent benchmark domain to test ad hoc teamwork, since the predators must work as a team and carefully coordinate in order to successfully capture the prey. We adopt this same domain in this paper. PLASTIC-Model uses the library of teammates in conjunction with a decision tree learner to identify the teammates' (the other predators) behavior. An MDP model is then built that describes the joint behavior of the teammates and the prey, and MCTS is then used to determine the ad hoc predator's actions.

More recently, Barrett and Stone [4] proposed PLASTIC-Policy, in which the problem of ad hoc teamwork is addressed in its full generality: an ad hoc agent is paired with a group of unknown teammates, with which it must coordinate in an unknown joint task. The approach relies on a preliminary set of "exploratory games" during which the ad hoc agent plays in different teams, and uses reinforcement learning to learn the best policy when playing in each of those teams. During such exploratory games, the agent also builds a simple predictive model that can be used to identify which team it is playing with. Then, at playtime, the agent adopts the learned policy for the team that best matches the predictive model. The work of Barrett and Stone [4] is among the first to address the ad hoc teamwork in all its components: task identification, teammate identification and planning, for which it uses reward information provided by the environment.

Melo and Sardinha [17] also addresses the full ad hoc teamwork problem, but relying on a different set of assumptions. In this work, teammates are assumed bounded-memory best responders. The target task belongs to a set of possible tasks but is otherwise unknown, and the ad hoc agent must simultaneously determine the behavior of the teammates and use it to infer the target task. Once this is done, it then proceeds to compute the best action. Unlike Barrett and Stone [4], Melo and Sardinha [17] the agent does not have available any reward signal from the environment and instead rely on the assumption that the teammates are bounded-memory best responders to address the full ad hoc teamwork problem.

We assume that our ad hoc agent has available a reward signal from the environment that can be used for task identification.[1] However, unlike Barrett and Stone [4], we do not require the ad hoc agent to play "exploratory games" and instead propose the use of a model-based reinforcement learning approach to learn separate models for the environment and the teammates. The ad hoc agent then uses the learned model to select its actions using MCTS planning. Our use of separate model for the environment and for the teammates is in contrast with the PLASTIC-Policy approach, in which the agent implicit learns a joint environment+teammate model. Our approach has two advantages: (i) the use of separate models takes advantage of the natural factorization of the problem, thus leading to smaller models; (ii) the use of separate environment and teammate models allows reusing the environment model when the agent is faced with a new team, potentially rendering the learning process more efficient.

Our approach also follows well-established architectures from multiagent reinforcement learning. For example, Wang and Sandholm [23] propose OAL, a model-based multiagent reinforcement learning approach in which the teammates are also modeled independently of the environment. OAL uses a variant of fictitious play to learn the behavior of the teammates and combines the resulting model with a tabular model-based reinforcement learning algorithm. A posterior work generalized OAL to accommodate the use of function approximation [16]. Our teammate modeling approach can itself be understood as an instance of fictitious play.

1.2 Contributions

We contribute a novel ad hoc agent architecture addressing the ad hoc teamwork problem in full, including task identification, teammate identification and planning. Our approach comprises three main modules, each addressing one of the aforementioned challenges:

- We use of model-based reinforcement learning to learn a model of the environment and task. The model uses a deep neural network that predicts, given the current state and the actions of all agents in the environment (ad hoc and teammates), the resulting state and reward.
- We use a fictitious-play-like approach to model the teammates, constructing a predictive model that, given an input state, returns a distribution over the teammates' actions.
- Finally, using the environment, task and teammate models, we use of Monte Carlo tree search to select the action of the ad hoc agent.

We illustrate our approach in the well-established pursuit domain, providing a comparative analysis between our own performance and that of state-of-the-art methods from the literature.

2. PROBLEM

Ad hoc teamwork is a research problem introduced by [21] in which an agent must cooperate with a team of agents, with unknown behavior, without any pre-coordination. This differs from the normal multi-agent setting because the ad hoc agent may not share the same algorithm as its teammates and cannot pre-coordinate to match the strategy that the team follows. Therefore, the agent must be able to understand the teammate's behavior and then plan optimal actions according to such behavior. This leads us to two challenges: teammate behavior modeling and planning. In this work, we further assume that the environment is not known, that is, the agent does not have information about the environment dynamics, which leads to a third challenge: environment modeling.

[1] In this setting, the ad hoc teamwork problem can be interpreted as an instantiation of a multiagent reinforcement learning problem.

In the remainder of this section, we discuss these three challenges and how we address them, and also introduce the formalization of the problem described.

2.1 Problem formalization

We model the problem as an Multi-Agent Markov Decision Process (MMDP) $\mathcal{M} = (N+1, \mathcal{X}, \mathcal{A}, \{\mathbf{P}_{\boldsymbol{a}}\}, r, \gamma)$, where

- $N+1$ is the number of agents considered. We assume throughout that the ad hoc agent is agent 0 and there are N teammates.
- $\mathcal{X}$ is the (countable) set of possible states of the environment. The state at time step t is denoted as the random variable $\mathrm{x}(t)$ (we use upright letters to denote random variables) taking values in $\mathcal{X}$.
- $\mathcal{A}$ is the *joint action space*. In other words, $\mathcal{A}$ is the cartesian product of $N+1$ individual action sets, $\mathcal{A}_0, \ldots, \mathcal{A}_N$. Each $\mathcal{A}_n$ is the individual action set of agent n.

  We write a_n to refer to an element of $\mathcal{A}_n$, and $\boldsymbol{a}$ to refer to an element of $\mathcal{A}$, understood as a vector

  $$\boldsymbol{a} = (a_0, a_1, \ldots, a_N).$$

  We write $\boldsymbol{a}_{-0}$ to denote a *reduced joint action*

  $$\boldsymbol{a}_{-0} = (a_1, \ldots, a_N),$$

  obtained from a joint action $\boldsymbol{a}$ by removing the component corresponding to the action of agent 0. The joint action at time step t is denoted as a random vector $\mathbf{a}(t) \in \mathcal{A}$, and we write $\mathrm{a}_n(t)$ to denote the nth component of $\mathbf{a}(t)$, corresponding to the individual action of agent n at time step t.
- $\mathbf{P}_{\boldsymbol{a}}$ denotes the transition probabilities associated with joint action $\boldsymbol{a}$. We write, interchangeably, $\mathbf{P}_{\boldsymbol{a}}(y \mid x)$ and $\mathbf{P}(y \mid x, \boldsymbol{a})$ to denote the transition probability

  $$\mathbf{P}_{\boldsymbol{a}}(y \mid x) = \mathbf{P}(y \mid x, \boldsymbol{a}) = \mathbb{P}\left[\mathrm{x}(t+1) = y \mid \mathrm{x}(t) = x, \mathbf{a}(t) = \boldsymbol{a}\right].$$
- $r(x, \boldsymbol{a})$ is the expected reward function representing the target task.
- γ is a constant in $[0, 1)$, denoting the discount (see below).

The difference between solving the MMDP in the ad hoc teamwork setting is that we only control one of the agents and therefore only one action is provided at each timestep. Also, the transition function $\mathbf{P}_a$ is not known to the ad hoc agent, which constitutes another obstacle to directly solving the MMDP.

Since we cannot control other agents' actions, we can also see this problem as a Markov Decision Process (MDP) where the transition function is a combination of the teammates' behaviors and the environment dynamics. Denoting a_{legacy} as the joint action of the legacy agents, and a_{adhoc} as the ad hoc agent's action and defining $\pi(x, a_{\text{legacy}}) \to [0, 1]$ as the teammate behavior function, then this MDP transition function would simply be:

$$\mathbf{P}^{\text{MDP}}_{a_{\text{adhoc}}}(x(t+1)|x(t)) = \sum_{a_{legacy} \in \mathcal{A}_{\text{legacy}}} \pi(x(t), a_{\text{legacy}}) \mathbf{P}^{\text{MMDP}}_{a_{\text{legacy}}+a_{\text{adhoc}}}(x(t+1)|x(t))$$

This representation of the problem is much more interesting and allows us to focus on a single action - the ad hoc agent's. So, the new MDP can be described as the tuple $(\mathcal{X}, \mathcal{A}_0, \mathbf{P}^{\text{MDP}}_a, r^{\text{MDP}}, \gamma)$, where:

- $\mathcal{X}$ is the same state space as the MMDP described previously
- $\mathcal{A}_0$ is the action space of the ad hoc agent
- $\mathbf{P}^{\text{MDP}}_a$ is the transition function as computed in Equation 1, which is simply the expected transition function for a given teammate behavior
- $r^{\text{MDP}}(x, a)$ is, similarly to the transition function, the expected reward for a given teammate behavior
- γ is the same discount factor as in the MMDP

It is clear that if we knew the legacy teammates' behaviors and the dynamics of the environment, we could solve the MDP with the classical value iteration solution. It is also tempting to simply use Q-learning to learn a policy without knowing the transition function $\mathbf{P}_{\text{MDP}}$. We show in the Results section that this approach takes a lot of iterations to converge to an acceptable performance, partly due to the huge state-action space of the domains used.

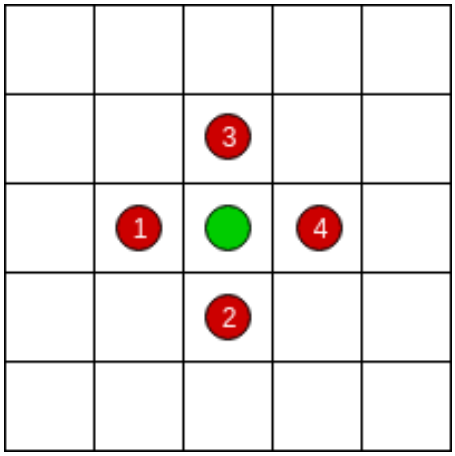

Figure 1: An example of a capture position. Prey is represented as a green circle and predators are represented as red circles. In this case, the prey is unable to move in any direction, ending the game and all predators earn a reward.

2.2 Pursuit domain

Throughout this work, we use the pursuit domain as a test scenario, similarly to Barrett et al. [5], which is used widely in multi agent reinforcement learning. In this section we detail the MMDP model introduced in the previous section for this specific domain.

In the pursuit domain, there is a $N \times N$ toroidal grid world where four predators chase a prey. At each timestep,

the prey chooses a random action, either Up, Down, Right or Left and tries to execute it. If the cell is blocked, the move fails and the prey remains in the same cell. If the prey is surrounded by all four sides, it is said to be captured and the episode ends – an example can be seen in Figure 1. Similarly, predators choose one of the four actions at each timestep with the goal of surrounding the prey. If two agents try move to the same cell, the one who succeeds is chosen at random and the other remains in its cell.

This domain is well suited for the ad-hoc teamwork problem, as the goal can only be achieved if all agents cooperate and therefore the ad-hoc agent should learn how its teammates behave in order to maximize its performance.

Formally, we define this domain as the MMDP $\mathcal{M}_L = (4, \mathcal{X}, \mathcal{A}, \{\mathbf{P}_{\boldsymbol{a}}\}, r, \gamma)$, where:

- L is the height and width of the grid
- $\mathcal{X} = \{(x_0, y_0, \cdots, x_4, y_4) : \forall_{1 \leq i \leq 4}, 0 \leq x_i, y_i \leq L\}$ Each (x, y) represents the position of one agent
- $\mathcal{A} = \{(a_0, a_1, a_2, a_3) : \forall_{1 \leq i \leq 4}, a_i \in \{U, D, L, R\})\}$ Where (U, D, L, R) represent the actions of moving Up, Down, Left or Right.
- $r = \begin{cases} 0 & \text{if prey is blocked} \\ -1 & \text{otherwise} \end{cases}$

The transition function $\mathbf{P}_{\boldsymbol{a}}$ is defined by the movements of the agents. If an agent moves towards a direction, the probability of success is always 1 if the destination cell is empty and if no other agent tried to move towards it. If that's not true, then a collision occurs and one of the agents (selected at random) is able to move and all the other ones are unable to move. The prey always chooses each direction with probability 0.25.

3. 3-LAYERED ARCHITECTURE FOR AD-HOC TEAMWORK

Our solution can be segmented in three parts that address the three distinct challenges described in the previous section. These three parts are: model-based reinforcement learning using a deep neural network to model the task and environment, fictitious-play like model to learn teammate's policies and MCTS planning to select the optimal action given the previous models. In this section we'll go into detail about these segments and how they interact to provide a solution to the ad hoc teamwork problem.

3.1 Task identification with model-based reinforcement learning

Task identification can be split into two other tasks: reward and enviroment dynamics prediction. Environment dynamics are described by the distribution $\mathbf{P}_{\boldsymbol{a}}(y|x)$ for every $\boldsymbol{a}, x, y$, where $\boldsymbol{a}$ denotes the joint action, x the current state and y the next state.

To learn the dynamics of the environment, a feedforward neural network is used, as seen in Figure 2, similarly to the teammate behavior. In this case, however, we do not have an output node for each possible state, as it would require a huge number of states. Instead, we assume the state can be fully described by a reasonable number of features for which a metric distance is defined. We denote these features $\mathcal{F}(x)$. This function could be the same as the one used in the teammate behavior model. For example, in a grid world we could have a set of nodes for positions for each object.

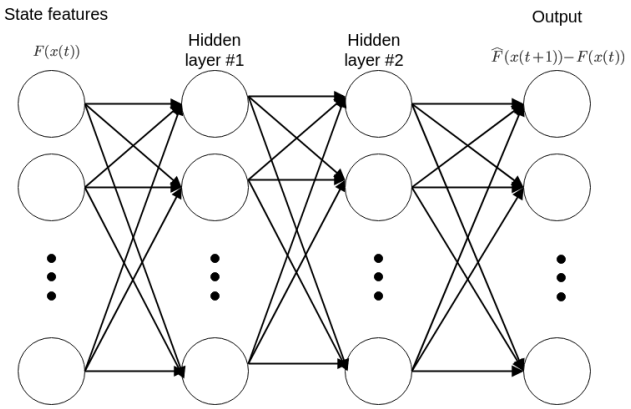

Figure 2: Architecture of environment transition neural network

The output nodes of the neural network are the difference between the features of the current state and the next state, that is $\mathcal{F}(x(t+1)) - \mathcal{F}(x(t))$. By using this variable change trick, the neural network only needs to output a smaller range of values, making the training fairly easier as validated by our experiments.

The input nodes are the features that describe the current state, $\mathcal{F}(x(t))$, along with the actions of all agents, $\boldsymbol{a}$. Actions are described as one hot vectors.

This model predicts the most likely next state given the current state and the actions taken. By using an environment model, we can reuse this information across different teams. That is, even though the teammate behavior changes through episodes, the environment dynamics remain the same, allowing the agent to improve its policy with changing teams.

The architecture described in Figure 2 of this neural network is composed of two hidden layers, an input layer containing the current state features and outputs the predicted difference between the current state and next state. The activation function used is *relu* for every layer except the last, where we do not use any activation. We optimize, using ADAM [14], the loss function $||\mathcal{F}(x(t+1)) - \hat{\mathcal{F}}(x(t+1))||^2$, which, denoting $\boldsymbol{o}$ as the output of the neural network, corresponds to $||\mathcal{F}(x(t+1)) - \mathcal{F}(x(t)) - \boldsymbol{o}||^2$.

3.1.1 Task identification in the pursuit domain

In the pursuit domain, the reward given is -1 for every state except for the capture state, which is 0. Environment dynamics regarding the teammates positions are deterministic as long as there are no collisions, but the prey acts randomly, which makes the problem a bit harder. In case of collision, ties are selected randomly as each cell can only be occupied by a single agent. To represent the state, we select the following features:

- Position in the x-axis of the prey
- Position in the y-axis of the prey
- Position in the x-axis of teammate 1 (ad-hoc agent)
- Position in the y-axis of teammate 1 (ad-hoc agent)

- Position in the x-axis of teammate 2
- Position in the y-axis of teammate 2
- Position in the x-axis of teammate 3
- Position in the y-axis of teammate 3
- Position in the x-axis of teammate 4
- Position in the y-axis of teammate 4

This feature set contains all information needed to reconstruct the state and also the metric distance between states is very well defined, which is useful to compute similarity between states.

3.2 Learning teammate's policies with a deep neural network

One of the challenges of the ad hoc teamwork setting is to predict the behavior of the legacy teammates, that is the function $\pi(x, a_{\text{legacy}})$. We define $\phi_i(x)$ as a set of features for a given state x as seen by agent i. In this work, we make the following assumptions:

- Teammates have a static policy – meaning the action distribution depends only on the current state. This implies they do not learn over time.
- Actions chosen are visible to the ad hoc agent
- Behaviors are homogeneous among the teammates – meaning they all follow the same policy assuming states are relative to each agent. This means $\phi_i(x_1) = \phi_j(x_2) \Rightarrow \pi(x_1, a_i) = \pi(x_2, a_j)$

These assumptions allow us to model the teammate policy using the fictitious play algorithm. Using fictitious play, the predicted policy is given by the following equation:

$$\hat{\pi}(x, \boldsymbol{a}_{-0}) = \frac{N(x, \boldsymbol{a}_{-0})}{\sum_{\boldsymbol{a'}_{-0} \in \mathcal{A}_{-0}} N(x, \boldsymbol{a'}_{-0})} \quad (1)$$

Where $N(x, \boldsymbol{a})$ represents how many times the team played actions $\boldsymbol{a}$ in the state x in the past. A problem with this approach is that it requires lots of state-action visits if the domain has a sizable state-action space. One way to improve this is to leverage the similarity between different state-action pairs.

We propose looking at this problem as a supervised learning problem, where the input is the state (or state features as seen from one teammate $\phi_i(x)$ and the output is the action distribution of a single teammate. Samples to train this model are collected as the ad hoc agent navigates through the environment and observes other agent's actions.

We chose a neural network as the teammate behavior model, with the input being state features and an output for every action estimating the probability that a given legacy teammate chooses that action in the input state. Since every teammate has the same policy, we can use the same neural network to predict each of the agent's actions, and train it with all actions observed.

Using a neural network to model the teammate policy allows us to leverage the behavior of the team in a given state to similar states, using state features to measure this similarity. This works in huge domains where mapping every state-action to a probability in a non-parametric way, for example when using fictitious play, would be intractable as available memory is limited.

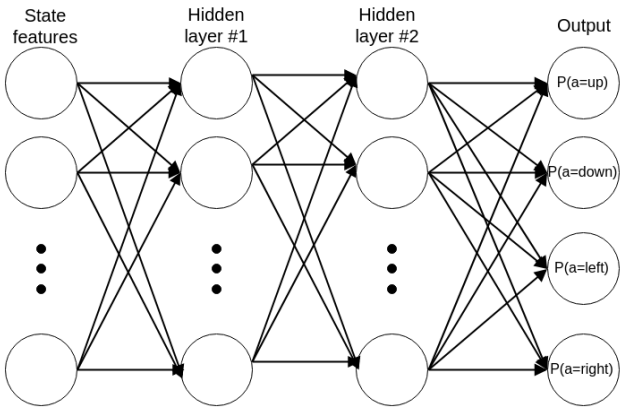

Figure 3: Architecture of teammate behavior neural network

The architecture used can be seen in Figure 3. It is composed of two hidden layers, each with 128 nodes. Its input are the state features, relative to teammate i and it outputs the predicted distribution of teammate i actions. A *relu* activation function is used between all layers except the last one, where *softmax* is used instead. During training, the neural network is optimized using the ADAM [14] optimizer.

3.2.1 *Learning teammates' policies in the pursuit domain*

In the pursuit domain, teammates can select one of four actions at every timestep: *up*, *down*, *left* or *right*. Our goal is to find out the distribution of each teammate's action, $\pi(x, a_i)$. Using the neural network described previously requires us to build features of the state relative to each teammate. We selected the following feature set:

- Difference of positions in the x-axis between the teammate and the prey
- Difference of positions in the y-axis between the teammate and the prey
- Difference of positions in the x-axis between the teammate and its closest teammate
- Difference of positions in the y-axis between the teammate and its closest teammate
- Difference of positions in the x-axis between the teammate and its 2nd closest teammate
- Difference of positions in the y-axis between the teammate and its 2nd closest teammate
- Difference of positions in the x-axis between the teammate and its 3rd closest teammate
- Difference of positions in the y-axis between the teammate and its 3rd closest teammate

This set allows to describe the state relative to a teammate. If, in a given pursuit world, we switch the position of 2 agents, their feature set is identically switched. This only happens because this feature set is completely independent

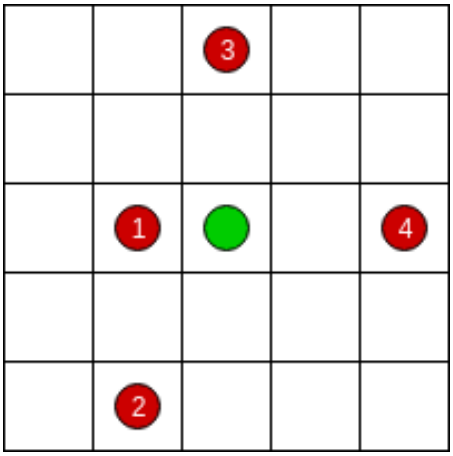

Figure 4: Example state in the pursuit domain

of teammate's id's. Since teammate behaviors are homogeneous and we have features relative to each teammate, we can use the same neural network to predict actions of all teammates.

In figure 4, we can see an example of a valid state in the pursuit domain. In that state, the feature set for teammate 1 is $[1, 0, 0, 2, -2, 0, 1, -2]$. Firstly, we have the distance to the prey $(1, 0)$, then the closest teammate is teammate 2, so we have distance $(0, 2)$, then teammate 4 for which the distance is $(-2, 0)$ since the world is toroidal and then finally teammate 3, which is $(1, -2)$ blocks apart from teammate 1. If we switch the numbers 2, 3 and 4 in the figure, the feature set for teammate 1 remains the same, as it is independent of the teammate identifiers.

3.3 MCTS planning

If we knew both the teammate behavior and environment models, we could compute $\mathbf{P}^{\text{MDP}}$ as described in Equation 1. Since we know every component of the MDP, we could solve it using Value Iteration. However, we are interested in problems which have large state spaces and Value Iteration would take a long time in those cases.

We decide to use Monte Carlo Tree Search (MCTS) to plan the optimal actions for the ad hoc agent as it provides a good approximation of the optimal policy. Given the teammate behavior model and the environment model for the MMDP, it is possible to compute the environment dynamics for the resulting MDP. In this work, we use Upper Confidence bound for Trees (UCT). Of course that the planning won't be optimal since both the teammate behavior model and the environment model are just approximations and we do not have infinite time.

An example of the expansion step of the MCTS is shown in Figure 5. In this step, we use the model described in Section 3.2 to predict what actions the ad-hoc agent's teammates will take in a given state. After we have the predicted actions, we concatenate them with the ad-hoc's action to build a joint action. This joint action and the given state is fed into the environment model described in Section 3.1 which outputs the most likely next state, concluding the expansion step of MCTS.

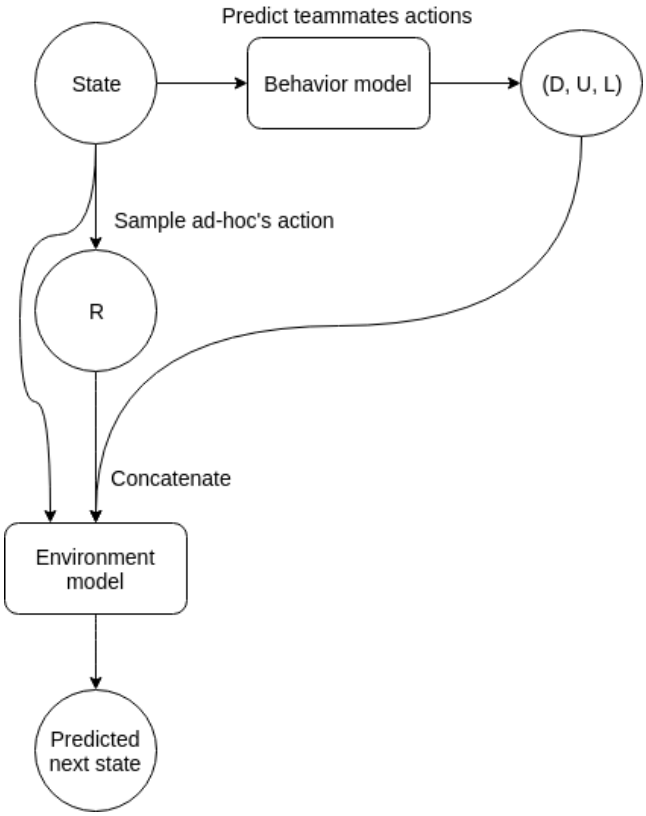

Figure 5: An example of the expansion step of MCTS planning using previously learned models of both behavior and environment

4. RESULTS

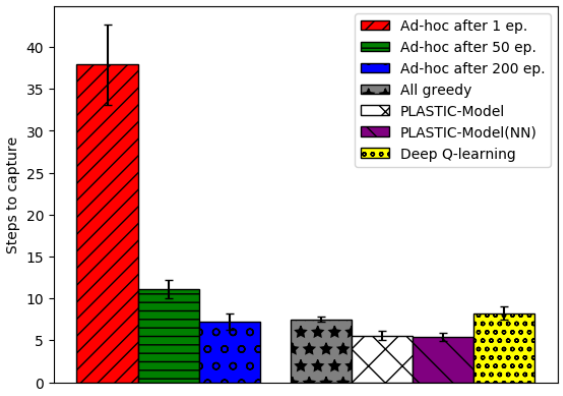

Figure 6: Results for our ad-hoc agent after 1, 50 and 200 episodes in a 5x5 world

4.1 Greedy teammates

In this section we evaluate our work using greedy teammates, as defined in [9]. Greedy teammates behave according to a fixed set of rules:

- If already adjacent to the prey, move towards it
- Pick the closest free cell adjacent to the prey
- If distance in x-axis is bigger than the y-axis, move either left or right, whichever gets it closer to the prey, else move either up or down, whichever gets it closer to the prey.
- If both directions blocked, move randomly

This is a very simple teammate, almost deterministic and doesn't take into account its teammates unless they are already neighboring the prey. As a result, it works well in small (e.g. 5×5) worlds, but gets worse as we increase the world size. Our ad hoc agent will play with these teammates for some episodes, and learn as it plays.

In figure 6 we show its score after playing 1, 50 and 200 episodes in a 5×5 world. We can see that it improves with more episodes of training and even surpasses a team of all greedy agents.

Compared to PLASTIC-Model and PLASTIC-Model with our teammate behavior model, it performs somewhat worse. This is expected as those two solutions have a perfect model of the environment.

Deep Q Network (DQN) has the worst result of all solutions, presumably because it requires more data to converge to a good performance.

In figures 7 and 8 we show its score for bigger worlds. It's clear that its performance can keep up with bigger worlds and in all three world sizes it has similar results to using all greedy teammates.

Even though it scores worse than PLASTIC-Model, this can be easily explained as PLASTIC-Model has the correct model of the greedy agents and knows the task exactly, while our approach learns it with time. When using our teammates' behavior model in PLASTIC-Model(NN), it gets closer to our results.

Looking at the results in figure 8, we conclude that the DQN approach gets even worse with bigger worlds, due to

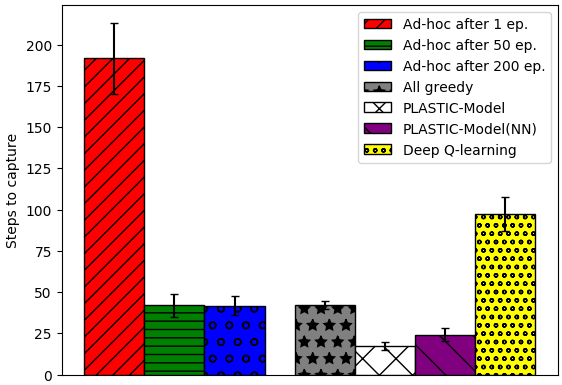

Figure 7: Results for our ad-hoc agent after 1, 50 and 200 episodes in a 10x10 world

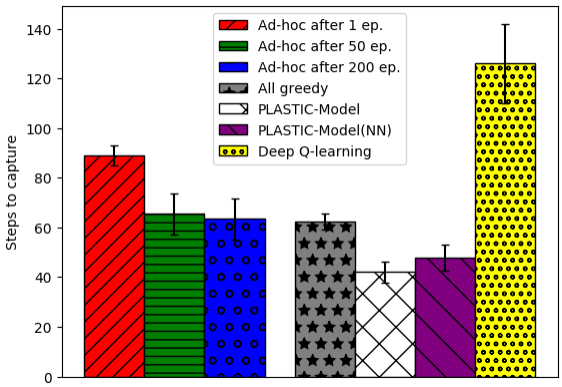

Figure 8: Results for our ad-hoc agent after 1, 50 and 200 episodes in a 20x20 world

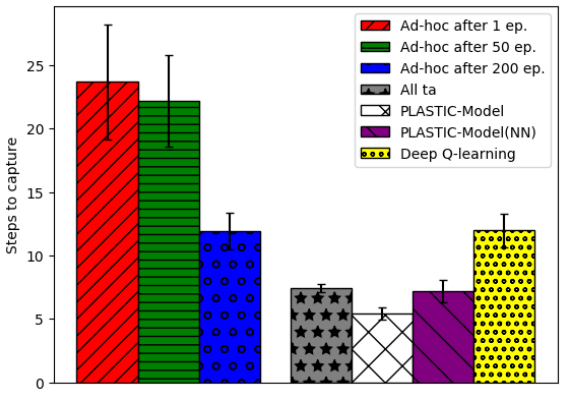

Figure 9: Results for our ad-hoc agent, playing with team aware agents, after 1, 50 and 200 episodes in a 5×5 world

having only a small number of episodes for training. Our solution can cope with this increasing state-space size while also having only 200 episodes to train.

4.2 Teammate aware teammates

In this section we evaluate our work using teammate aware agents, again defined in [9]. Teammate aware agents try to prioritize farther from the prey agents so they can use the fastest path to chase the prey. They also use A* to plan for the optimal path assuming teammates are static obstacles. Formally, they follow the following set of rules:

- Sort the predators based on the largest distance to the prey
- For each predator, pick the free cell that is neighbor to the prey, that is closest to them and has not been picked yet
- If the predator is already at the picked cell, move towards the prey
- Else, Use A* to pick an action, assuming teammates are static obstacles

This behavior works much better in bigger worlds, as can be seen in Figure 11 when compared to Figure 8. It is also a behavior that isn't as easy to learn due to the avoidance of obstacles done by A* and the pre-coordination used in the destination picking.

Again, in figure 9 we show its score after playing 1, 50 and 200 episodes in a 5×5 world. When compared to greedy agents in Section 4.1, we can see that our solution doesn't work as well. This is explained with 2 reasons: team aware agents have a more complex behavior, which is hard to learn and team aware agents rely a lot on their team (to avoid collisions and prioritize agents farther from the prey), making mistakes more heavily penalized.

In figures 7 and 11 we show our score in 10×10 and 20×20 worlds. The results are similar to the previous section regarding greedy agents: the ad-hoc agent still manages to get a good score in big worlds when following our solution, but it degrades heavily when using DQN. However, all results are overall worse than when the ad-hoc agent played with greedy teammates, and again this is explained with the increased complexity of playing with team aware agents.

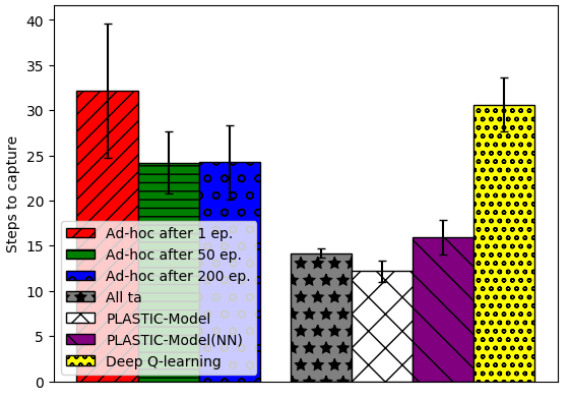

Figure 10: Results for our ad-hoc agent, playing with team aware agents, after 1, 50 and 200 episodes in a 10×10 world

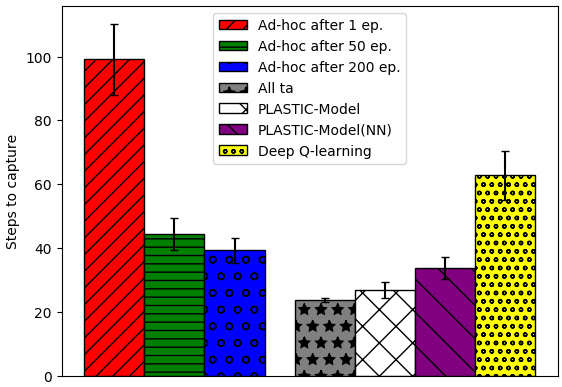

Figure 11: Results for our ad-hoc agent, playing with team aware agents, after 1, 50 and 200 episodes in a 20×20 world

5. CONCLUSION

Ad hoc teamwork is an emerging research field which intends to create an agent that can cooperate with a team of other agents without pre-coordinating with them. As more agents enter the real world, it is expected that they need to cooperate with others that they have never worked with and with unknown behaviors, making ad hoc agents necessary and useful.

Most of existing work in this area focuses on theoretical results with very strong restrictions regarding number of teammates, complexity of behavior and available information. Some work focuses on empirical results such as PLASTIC-Model and PLASTIC-Policy, where our inspiration comes from.

We propose a novel architecture combining deep neural networks to model the teammates and the task and MCTS for planning. This differs from both PLASTIC-Model, where the task is known and teammates behaviors are either known or learned beforehand with decision trees, and PLASTIC-Policy, where the policy is directly learned without learning teammate's behaviors and environment dynamics.

By comparing our solution with state-of-the-art PLASTIC-Model, we show that we can get comparable performance while using less information regarding the team and the task. Looking at PLASTIC-Model results with our learned teammate behavior, we can see that the difference in performance comes from the approximation error in learning teammates' policies.

We also compare it with DQN, a solution widely used in reinforcement learning, with similar assumptions to our work. DQN is quite similar to PLASTIC-Policy for our intents and purposes, as we are more interested in online learning of teams that the ad hoc agent has never seen before. We show that DQN, when compared to our work, takes a long time to learn, as usual for model-free approaches. Additionally, explicitly modeling the task as we do, allows the agent to reuse this information when teams change, unlike DQN and PLASTIC-Policy.

6. FUTURE WORK

In the future, it would be interesting to adapt the algorithm to work in continuous state and action-spaces such as the Half Field Offense domain, so it would be possible to compare it against the PLASTIC-Policy [3] algorithm.

Regarding planning, MCTS turned out to be quite slow. A faster planning algorithm would make this work more useful and interesting. One way would be to use a well-studied optimization of MCTS or to simply use an alternative to MCTS.

It would also be interesting to try other supervised learning techniques instead of neural networks as they might be more data efficient.

References

Noa Agmon and Peter Stone. Leading ad hoc agents in joint action settings with multiple teammates. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, (June):341–348, 2012.

Samuel Barrett and Peter Stone. Ad Hoc Teamwork Modeled with Multi-armed Bandits : An Extension to Discounted Infinite Rewards. *Teacher*, (May), 2011.

Samuel Barrett and Peter Stone. Cooperating with Unknown Teammates in Robot Soccer. *AAAI Workshop on Multiagent Interaction without Prior Coordination (MIPC 2014)*, (July):2–7, 2014.

Samuel Barrett and Peter Stone. Cooperating with Unknown Teammates in Complex Domains : A Robot Soccer Case Study of Ad Hoc Teamwork. *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, (January):2010–2016, 2015.

Samuel Barrett, Peter Stone, and Sarit Kraus. Empirical evaluation of ad hoc teamwork in the pursuit domain. *Autonomous Agents and Multiagent Systems (AAMAS)*, (May):567–574, 2011. URL `http://dl.acm.org/citation.cfm?id=2031678.2031698`.

Samuel Barrett, P Stone, Sarit Kraus, and A Rosenfeld. Learning teammate models for ad hoc teamwork. *AAMAS Adaptive Learning Agents (ALA) Workshop*, (June):57–63, 2012. URL `http://u.cs.biu.ac.il/ sarit/data/articles/ala2012.pdf`.

Samuel Barrett, Peter Stone, S Kraus, and A Rosenfeld. Teamwork with limited knowledge of teammates. *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, (July):102–108, 2013. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/viewFi`

Samuel Barrett, Noa Agmon, Noam Hazon, Sarit Kraus, and Peter Stone. Communicating with unknown teammates. *Frontiers in Artificial Intelligence and Applications*, 263(May):45–50, 2014. ISSN 09226389. doi: 10.3233/978-1-61499-419-0-45.

Samuel Rubin Barrett, Peter Stone, and Raymond Mooney. Making Friends on the Fly : Advances in Ad Hoc Teamwork. 2014. doi: 10.1007/978-3-319-18069-4.

M Benda, V Jagannathan, and R Dodhiawala. On Optimal Cooperation of Knowledge Sources - An Empirical Investigation. Technical Report BCS–G2010–28, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, WA, USA, 1986. URL `http://www.cs.utexas.edu/ shivaram`.

Ronen I Brafman and Moshe Tennenholtz. On Partially Controlled Multi-Agent Systems. 4:477–507, 1996.

Doran Chakraborty and Peter Stone. Cooperating with a markovian ad hoc teammate. *International Conference on Autonomous Agents and Multiagent Systems,*, pages 1085–1092, 2013. URL `http://dl.acm.org/citation.cfm?id=2485091`.

Drew Fudenberg and David K. Levine. *The Theory of Learning in Games*. 2000. ISBN 0262061945. URL `http://www.amazon.com/dp/0262061945`.

Diederik P Kingma and Jimmy Lei Ba. Adam: A Method of Stochastic Optimization. In *International Conference on Learning Representations 2015*, pages 1–15, 2015.

L Kocsis and C Szepesvari. Bandit-based Monte-Carlo Planning. In *ECML'06*, pages 282–293, 2006. URL `http://ggp.stanford.edu/readings/uct.pdf`.

Francisco S. Melo and M. Isabel Ribeiro. Coordinated learning in multiagent MDPs with infinite state-space. *Autonomous Agents and Multi-Agent Systems*, 21(3):321–367, 2010. ISSN 13872532. doi: 10.1007/s10458-009-9104-y.

Francisco S. Melo and Alberto Sardinha. Ad hoc teamwork by learning teammates's task (Extended Abstract). *Autonomous Agents and Multi-Agent Systems*, pages 577–578, 2016. ISSN 15737454. doi: 10.1007/s10458-015-9280-x.

Yoav Shoham, Rob Powers, and Trond Grenager. On the Agenda(s) of Research on Multi-Agent Learning. 2001.

Yoav Shoham, Rob Powers, and Trond Grenager. If multi-agent learning is the answer, what is the question? pages 1–21, 2006.

Peter Stone, Ramat Gan, and Sarit Kraus. To teach or not to teach? Decision making under uncertainty in ad hoc teams. *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, (May):117–124, 2010. ISSN 15582914.

Peter Stone, Gal A. Kaminka, Sarit Kraus, and Jeffrey S Rosenschein. Ad Hoc Autonomous Agent Teams : Collaboration without Pre-Coordination. *Twenty-Fourth AAAI Conference on Artificial Intelligence*, (July):1504–1509, 2010.

Peter Stone, Gal A. Kaminka, and Jeffrey S. Rosenschein. Leading a best-response teammate in an ad hoc team. *Lecture Notes in Business Information Processing*, 59 LNBIP (May):132–146, 2010. ISSN 18651348. doi: 10.1007/978-3-642-15117-0_10.

Xiaofeng Wang and Tuomas Sandholm. Reinforcement learning to play an optimal Nash equilibrium in team Markov games. *Advances in Neural Information Processing Systems*, 15:1571–1578, 2002. ISSN 1049-5258. doi: 10.1.1.19.2669.