Task-based ad-hoc teamwork with adversary

Elnaz Shafipour and Saber Fallah

University of Surrey
{e.shafipour,s.fallah}@surrey.ac.uk

Abstract. Many real-world applications require agents to cooperate and collaborate to accomplish shared missions; though, there are many instances where the agents should work together without communication or prior coordination. In the meantime, agents often coordinate in a decentralised manner to complete tasks that are displaced in an environment (e.g., foraging, demining, rescue or firefighting). Each agent in the team is responsible for selecting their own task and completing it autonomously. However, there is a possibility of an adversary in the team, who tries to prevent other agents from achieving their goals. In this study, we assume there is an agent who estimates the model of other agents in the team to boost the team's performance regardless of the enemy's attacks. Hence, we present *On-line Estimators for Ad-hoc Task Allocation with Adversary* (OEATA-A), a novel algorithm to have better estimations of the teammates' future behaviour, which includes identifying enemies among friends.

Keywords: Autonomous Systems, Adversary Agent, Learning Agent, Multi-agent system, Decentralised Task Allocation.

1 Introduction

The world is moving towards "smart systems", which rely on some form of intelligent agent technology, that can autonomously collect information from their surrounding environment and act upon it. An example is multiple rovers in space, which attempt to accomplish their missions cooperatively. These agents may work collaboratively toward the completion of common tasks that they cannot handle individually. However, there might be differences between the agents in terms of origin, access to information, and perceptual and actuation capabilities. Therefore, such teamwork might take place without any prior coordination protocol or even, in some cases, any form of explicit communication. These kinds of teams are known as ad-hoc teams. Moreover, many domains require agents to work together to accomplish tasks that are distributed across the system. In these systems, several tasks need to be accomplished in an uncertain environment with no centralised mechanism to allocate tasks. Accordingly, the agents in the team are not managed to perform their tasks, and they autonomously decide which one to complete, without being directly assigned [4]. The *decentralised* allocation is quite natural in ad-hoc teamwork, as we cannot assume that other agents would be programmed to follow a centralised controller. For example, imagine a natural disaster and hazardous situation where autonomous robots (agents) have been dispatched from different countries or different organisations to handle the emergency conditions.

Rather than waiting for communication and coordination protocols to develop, these robots need to act immediately to avoid putting lives at risk. In other words, each robot chooses its own strategy for saving as many lives as possible and behaves accordingly.

Nevertheless, there is a possibility of existing potential enemies in the system which are unknown to the rest of the team. The agents of this type display destructive behaviours that prevent their teammates from reaching their targets. This work focused on the tasks-based teams where the team involves multiple agents with a range of cooperative and disruptive behaviours in a decentralised distributed system. As such, we refer to this task-based ad hoc team working with an adversary as *Task-based Ad-hoc Teamwork with Adversary*. Hence, learning and reasoning about the team members are mandatory to improve the team's performance. In our system, there are some learning agents, who are aware of pre-existing standards for coordination and communication, so they can try to learn about their teammates with limited information [3]. Through such intelligent coordination in this ad-hoc team, the shared goals will be achieved more efficiently. However, the sole aim of our team study is not to improve collaboration, and we need to reduce the hostile behaviour of some team members by identifying and examining the enemies correctly. Our solution to this problem is *On-line Estimators for Ad-hoc Task Allocation with Adversary* (OEATA-A), a *novel algorithm* for estimating teammates future behaviours. We show that our algorithm converges to a perfect estimation when the number of tasks to be performed gets larger.

2 Related Works

In the literature, there are many works considering the presence of opponents in the team. In the majority of these studies, the team members know who the adversary agent is. Celli [5] focuses on ex-ante coordination, where team members have an opportunity to discuss and agree on tactics before the game starts, but will be unable to communicate during the game.

Mirchevska [8] presents a domain-independent Multi-Agent Strategy Discovering Algorithm (MASDA), which discovers strategic behaviour patterns of a group of agents under the described conditions. The algorithm represents the observed multi-agent activity as a graph, where graph connections correspond to performed actions and graph nodes correspond to environment states at action starts. Based on such data representation, the algorithm applies hierarchical clustering and rule induction to extract and describe strategic behaviour.

There is another work [9], which is focused on resilience in cooperative MAS and propose an Antagonist-Ratio Training Scheme (ARTS) by reformulating the original target MAS as a mixed cooperative-competitive game between a group of protagonists which represent agents of the target MAS and a group of antagonists which represent failures in the MAS. However, Lin [7] introduces a novel attack where the attacker first trains a policy network with reinforcement learning to find a wrong action it should encourage the victim agent to take. Then, the adversary uses targeted adversarial examples to force the victim to take this action. Uesato [11] addresses the problem of evaluating learning systems in safety-critical domains such as autonomous driving, where failures can have catastrophic consequences. In our work, we assume that we are not aware

of which teammate is the adversary agent. However, by observing their behaviour, we show that our method could obtain a better estimation which leads better performance for the team.

3 Methodology

3.1 Ad-hoc Teamwork with Adversaries

Our ad-hoc team consists of several agents, which do not have enough knowledge about each other. The team's goal is to work together and cooperate to accomplish shared goals. There is, however, a possibility of there being an adversary agent among team members. This agent is attempting to minimise the team's performance in a way the other agents are not aware of.

Three main groups of agents are working together as part of this ad hoc team. The first group is the naive agents ($\omega \in \mathbf{\Omega}$), which attempt to improve the team's achievement. Agents of this type use static algorithms to accomplish their tasks, and they cannot learn from what is happening in their environment. Second are the adversaries, who attempt to defeat the goals of other agents. In our team, we assume there is only one adversary agent, Λ. The last group is the learning agents, and again we consider only one learning agent, ϕ, in our team. The objective of the learning agent is to find the best actions that maximise the performance of the team. The ϕ agent is the only agent in the team which can learn teammates' future actions as it estimates and discovers their models over time.

In this system, there is a set of tasks ($\mathcal{T}$) that team members make an effort to accomplish autonomously, except the adversary group. A task $\tau \in \mathcal{T}$ may require multiple agents, as well as several time steps to finish successfully. For instance, in a foraging problem, a heavy item may require two or more robots to be collected. Furthermore, the robots would need to move towards the task location, taking multiple time steps to move from their initial position.

Model of Naive Agents All naive agents try to perform their tasks autonomously within the environment. However, choosing and completing each task τ by each ω is dependent on its internal algorithm and capabilities. The algorithm for each ω can be varied in different domains. We assume that all these algorithms have a set of inputs, which we denote as *parameters* of these algorithms. For example, in the foraging domain [10] (explained in detail in Section 4.1), there might be multiple boxes in the robot's visible area. Hence, the algorithms in this domain would be the way the robot chooses an item to collect. The algorithm might be selecting the closest box or the lightest box among the visible ones. In addition, the size of the robot's visible cone, as well as its ability to collect the box, are considered its parameters. Like previous works [1,12], we consider the algorithm of choosing targets as the type of naive agents. Furthermore, we suppose the learning agent knows the set of possible types $\mathbf{\Theta}$ in the system. However, the type of each ω agent is unknown to it. Thus, naive agents' behaviour and actions mirror the type and parameter of the agents, and we define each $\omega \in \Omega$ as a tuple (θ, $\mathbf{p}$). $\theta \in \mathbf{\Theta}$ in this tuple is ω agent's type and $\mathbf{p}$ represents its

parameters, which is a vector $\mathbf{p} =< p_1, p_2, ..., p_n >$. Each element p_i in the vector $\mathbf{p}$ is defined in a fixed range [p_i^{min}, p_i^{max}] [1]. Choosing a new task (considered as the agent's "target") happens in the very first state, and whenever ω agent finishes a task. We call these states as *Choose Target State* ($\mathfrak{s}$).

Model of Adversary Agent The adversary agent has the full observation of the environment, and we define a *Markov Decision Problem* model for it. Although there are multiple agents in the team, we set the model *under the point of view of the agent* Λ. Therefore, we consider a set of states $\mathcal{S}_\Lambda$, a set of actions $\mathcal{A}_\Lambda$, a reward function $\mathcal{R} : \mathcal{S}_\Lambda \times \mathcal{A}_\Lambda \times \mathcal{S}_\Lambda \rightarrow [0, 1]$, and a transition function $\mathcal{T} : \mathcal{S}_\Lambda \times \mathcal{A}_\Lambda \times \mathcal{S}_\Lambda \rightarrow [0, 1]$ for the Λ agent. The actions in the model are only the Λ agent's actions and not any of others. Additionally, the goal of the Λ agent is minimising the reward function. In Λ agent's MDP model, all naive agents and the learning agent are considered as a part of the environment, and they are not directly represented in the MDP model. The Λ agent can only decide its own actions and has no control over the actions of any other agents in the team. However, the Λ agent has not the ability to learn the other teammates' types and parameters. Therefore, it will not be able to estimate the future behaviour of the teammates, and it considers them as obstacles in the environment. The Λ agent employ *UCT-H* [12] for its on-line planning.

Model of the Learning Agent Like the adversary agent, the learning agent has full observability and its model is defined as a single agent MDP, *under the point of view of the agent* ϕ, as in previous works [1,12]. Like the adversary agent, for the ϕ agent, we consider a set of states $\mathcal{S}_\phi$, a set of actions $\mathcal{A}_\phi$, a reward function $\mathcal{R} : \mathcal{S}_\phi \times \mathcal{A}_\phi \times \mathcal{S}_\phi \rightarrow [0, 1]$, and a transition function $\mathcal{T} : \mathcal{S}_\phi \times \mathcal{A}_\phi \times \mathcal{S}_\phi \rightarrow [0, 1]$, where the actions in the model are only the ϕ agent's actions and not any of others. Similar to the adversary agent, we apply *UCT-H* to solve the MDP model of the learning agent. It is clear that in the *actual* problem, the next state depends on the actions of all agents as they are dynamic in the environment. Whereas, the ϕ agent is unsure about the teammates' next actions. By taking naive agents' into account, given a state s, an agent $\omega \in \mathbf{\Omega}$ has an unknown probability distribution (pdf) across a set of actions $\mathcal{A}_\omega$, which is given by ω's internal algorithm (θ, $\mathbf{p}$). Additionally, as we mentioned earlier, the learning agent has the ability to estimate teammates' future actions. Note that the agents' types and parameters are actually not observable, but in this MDP model that is not directly considered. The estimated types and parameters are used during online planning, affecting the current transition function.

As mentioned earlier, in this task-based ad-hoc team, ϕ agent attempts to help the team to get the highest possible achievement. For this reason, the learning agent needs to find the optimal value function, which maximises the expected sum of discounted rewards $E[\sum_{j=0}^{\infty} \gamma^j r_{t+j}]$, where t is the current time, r_{t+j} is the reward ϕ agent receives at j steps in the future, $\gamma \in (0, 1]$ is a discount factor. Also, we consider that we obtain the rewards by solving the tasks $\tau \in \mathcal{T}$ of the team. That is, we define ϕ agent's reward as $\sum r_\tau$, where r_τ is the reward obtained after the task τ completion. Note that the sum of rewards is not only across the tasks completed by ϕ agent but all tasks are completed by any set of agents in a given state. Furthermore, there might be some tasks

in the system that cannot be achieved without cooperation between the agents. Hence, the number of required agents for finishing a task τ depends on each specific task and the set of agents that are jointly trying to complete it.

3.2 On-line Estimators for Ad-hoc Task Allocation with Adversary

In this paper, we introduce *On-line Estimators for Ad-hoc Task Allocation with Adversary (OEATA-A)*, which is based on the work done by Shafipour [10], called OEATA. In this method, we want to check if all the team members collaborate to finish common tasks. In other words, our goal is to check if there is any adversary agent in the team that has non-collaborative behaviours and wishes to avoid other team members to reach their goals which are called In OEATA-A, when the learning process starts, we assume there is no adversary agent in the team. Additionally, we suppose all non-learning agents will accomplish shared tasks. For this purpose, we record all tasks that each agent accomplishes (except for the learning agent ϕ). The reason for keeping the completed task by each agent is to compare them with the predictions of a set of *estimators*.

All *estimators* are initialised at the beginning of the process and evaluated whenever a task is done. The ones that are not able to make good predictions are removed after several incorrect estimations, and replaced by new *estimators* that can either be created using successful ones or entirely random. Moreover, if the agent is an adversary, then there will not be a recorded task for it.

In OEATA-A as well as OEATA, we have a *set of estimators* to keep the potential parameters $\mathbf{p}$ for a possible type θ, which are applied to predict task selections. Additionally, we have *history of tasks* to keep track of all tasks completed by each non-learning agent. Additionally, in OEATA-A, we have *bags of successful parameters*, which is borrowed from OEATA. However, in OEATA-A, we introduce *suspicious agent* to hold any uncooperative behaviour of the agent. The details of all these fundamentals are described below.

3.3 OEATA-A Fundamentals

3.4 Sets of Estimators

In OEATA-A, there are sets of *estimators* $\mathbf{E}_\delta^\theta$ for each type θ and each non-learning agents ($\delta \in \Delta$), whereas each set $\mathbf{E}_\delta^\theta$ has a fixed number of N *estimators*. Therefore, the total number of sets of *estimators* for all agents are $|\mathbf{\Delta}| \times |\mathbf{\Theta}|$. An *estimator* e of $\mathbf{E}_\delta^\theta$ is a tuple: $\{\mathbf{p}_e, \mathfrak{s}_e, \tau_e, c_e, f_e\}$, $\mathbf{p}_e$ is the vector of estimated parameters, and each element of the parameter vector is defined in the corresponding element range; $\mathfrak{s}_e$ is the initial state or the last *Choose Target State* , where the non-learning agent δ completed a task and wants to find a new task; τ_e is the task that δ agent would try to complete, assuming type θ and parameters $\mathbf{p}_e$. By having estimated parameters $\mathbf{p}_e$ and type θ, we assume it is easy to predict non-learning agent's target task at $\mathfrak{s}_e$; c_e holds the number of times that e was successful in predicting δ agent's next task; f_e holds the number of failures in predicting correct task.

History of Tasks As well as OEATA, in OEATA-A, we keep the history of the completed task for all non-learning agents. Therefore, along with the sets of *estimators*, ϕ agent keeps track of the tasks completed by each non-learning agent, as *History of Tasks*. Hence, the *History of Tasks* is defined as $\mathbf{H}_\delta = \{(\mathfrak{s}^0, \tau^0), \ldots, (\mathfrak{s}^n, \tau^n)\}$, where $\mathfrak{s}^i$ is the ith *Choose Target State*, where δ agent plans to find a new target, and τ^i is the actual task that the same agent completes afterwards. As mentioned before, *Choose Target State* is the initial state or the state where δ agent accomplishes a task and wants to choose a new one.

Bags of successful parameters As we mentioned earlier, we assume all non-learning agents as naive agents. Therefore, the same as OEATA, we keep a bag of successful parameters for each δ agent. Hence, if any *estimator* e succeeds in task prediction, for the vector of parameters $\mathbf{p}_e =< p_1, p_2, ..., p_n >$, we keep each element of the parameter vector $\mathbf{p}_e$ in their respective bags of successful parameters.

Suspicious Agent In OEATA-A, we have a new variable called *Suspicious Agent* ζ_δ. this value increases when the learning agent notices an unusual behaviour from a specific agent.

3.5 Process of Estimation

After presenting the fundamental elements of *OEATA-A*, we will explain how we define the process of estimating the parameters and type for each non-learning agent. The algorithm has five steps: (i) *Initialisation*; (ii) *Evaluation*; (iii) *Generation* and (iv) *Estimation*. Additionally, an (v) *Revision* step is executed for all agents in $\mathbf{\Delta}$, any time a task is completed by any agent of the team, including agent ϕ. Notice that the other difference between OEATA and OEATA-A is here in the processing stage. Unlike OEATA, OEATA-A does not have *update* step, and instead, we have *revision* step to find out the adversary agents. These steps are described below:

Initialisation At the very first step, all *estimators* should be initialised. Therefore, agent ϕ needs to generate N *estimators* for each type $\theta \in \mathbf{\Theta}$ and each $\delta \in \mathbf{\Delta}$. For every estimator, first, we create a random value per element of the parameter vectors $\mathbf{p}_e$ from the uniform distribution. Generated elements of the parameter vector should be in their defined range. For all *estimators*, in the initialisation phase, the initial state of the environment is set as the *Choose Target State* $\mathfrak{s}_e$. By having the type θ and the parameter vector $\mathbf{p}_e$ of the δ agent, the agent ϕ will be able to estimate its future task τ_e. Lastly, both c_e and f_e are initialised to zero.

Evaluation The evaluation of all sets of *estimators* $\mathbf{E}_\delta^\theta$ for a certain agent δ starts when it completes a task τ_δ. In this step we check if the τ_e (estimated task by assuming $\mathbf{p}_e$ to be δ's parameters with type θ in state $\mathfrak{s}_e$) is equal to τ_δ. If they are equal, we consider them as successful parameters and save each p_i in the $\mathbf{p}_e$ vector in a respective bag $\mathbf{B}_\delta^{\theta,i}$. If the estimated task τ_e is equal to the real task τ_δ, we set f_e to zero and increase

c_e. This penalisation of *estimators* for successive failures aids us in the type estimation. If τ_e is not equal to τ_δ, then we increase f_e and decrease c_e. We do not remove an *estimator* e after a failure since it may still have correct parameters. Hence, we define a *threshold* ξ for it, and if f_e is greater than ξ, we remove e from its belonging set. In this step, after finding successful and failing *estimators*, we update $\mathfrak{s}_e$ and τ_e of all survived *estimators* of the sets $\mathbf{E}_\delta^\theta$. We replace every $\mathfrak{s}_e$ with the current state s_c, and the τ_e with the new predicted task, by considering the current state s_c as the *Choose Target State* and assuming $\mathbf{p}_e$ as δ agent's parameter vector, and θ as its type. Additionally, at the end of this step, as a task has just been completed, we update δ agent's history $\mathbf{H}_\delta$, in order to use it for future evaluations.

Generation Lets suppose that $\mathbf{E'}_\delta^\theta$ is the new set with only the surviving *estimators* for agent δ and type θ that were not removed in the *Evaluation* step. In this step, the aim is to generate new *estimators*, in order to have the size of the sets $\mathbf{E}_\delta^\theta$ equal to N again. Therefore, $N - |\mathbf{E'}_\delta^\theta|$ new *estimators* should be generated. Unlike the *Initialisation* step, we do not only create random parameters for new *estimators*, but generate a proportion of them using previously successful parameters from the *bags* $\mathbf{B}_\delta^{\theta,i}$. Therefore, we will be able to use a new combination of parameters that had at least one victory in previous steps. Moreover, as the number of copies of the parameter p_i in the bag $\mathbf{B}_\delta^{\theta,i}$ is equivalent to the number of successes of the same parameter in previous steps, the chance of choosing very successful parameters will increase. The main part of producing new *estimators* is creating a new parameter vector $\mathbf{p'}$, and then updating the other elements of the *estimator* accordingly. Parameters for a portion $(N - |\mathbf{E'}_\delta^\theta|) \times \frac{1}{m}$ (where $m > 1$) of the new *estimators* will be randomly sampled from a distribution (e.g., uniform within the parameters range, if there is no domain knowledge). The other portion $(N - |\mathbf{E'}_\delta^\theta|) \times (1 - \frac{1}{m})$ will be generated as a new combination from the corresponding bags, which are holding previously victorious parameters. That is, each position p'_i of the parameter vector $\mathbf{p'}$ of the new *estimator* is populated by randomly sampling from the corresponding bag $\mathbf{B}_\delta^{\theta,i}$. If the corresponding bag $\mathbf{B}_\delta^{\theta,i}$ is empty, then that position of the parameter vector will be randomly generated. If all bags are empty, then all parameters will be random. Before creating a new *estimator* e', we check if the newly generated parameter $\mathbf{p'}$ would have at least one success across the history $\mathbf{H}_\delta$ so far. This improves our algorithm since it decreases the likelihood of wasting an *estimator* with a parameter $\mathbf{p'}$ that would not be able to make any correct prediction in the previous steps. As a result, if the output of the function is zero then $\mathbf{p'}$ will be discarded, otherwise, it will be considered as the parameter vector $\mathbf{p}_{e'}$ of the new *estimator* e'.

Estimation At each iteration after doing *evaluation* and *generation*, it is required to estimate a parameter and type for each $\delta \in \mathbf{\Delta}$ for decision-making. First, based on the current sets of *estimators*, we calculate the probability distribution over the possible types. First of all, we calculate the probability of the agent being adversary. For that, we consider ζ_δ value. If it is bigger than zero we will assume that the agent δ is the adversary. Otherwise, we calculate the probability of agent δ having type θ, $\mathsf{P}(\theta)_\delta$, we use the success rate c_e of all *estimators* of the corresponding type θ. That is, for each $\delta \in \mathbf{\Delta}$, we add up the non-negative success rates c_e of all *estimators* in $\mathbf{E}_\delta^\theta$ of each type

θ:$k_\delta^\theta = \sum_{e \in \mathbf{E}_\delta^\theta} \max(0, c_e)$. It means that we want to find out which set of *estimators* is the most successful in estimating correctly the tasks that the corresponding non-learning agent completed. In the next step we normalise the calculated k_δ^θ, to convert it to a probability estimation: $\mathsf{P}(\theta)_\delta = \frac{k_\delta^\theta}{\sum_{\theta' \in \mathbf{\Theta}} k_\delta^{\theta'}}$. After calculating the probability distribution over types for each $\delta \in \mathbf{\Delta}$, we use aggregation rules like median, mode, or mean across all parameter vectors $\mathbf{p}_e$ of each set of *estimators* $\mathbf{E}_\delta^\theta$. As a result, we will have one estimated parameter vector $\mathbf{p}$ per $\theta \in \mathit{\Theta}$ for each $\delta \in \mathbf{\Delta}$.

Revision The *Revision* step triggers when a task τ is completed by any agent in the team. As mentioned earlier, there is a possible issue that might arise in our estimation process when a certain task τ is accomplished by any of the team members (including agent ϕ), and some other non-learning agent was targeting to achieve it. This step has two sub-steps: *Updating Tasks* and *Checking Suspicious Agents*.

- ***Updating Tasks:*** Consequently, agent δ, would notice in the state s that the task is completed by other agents, and it will try to find a different task at this state. Hence, s would be a new *Choose Target State* for the agent δ. This problem would affect all *estimators* as well. Therefore, once a task τ is completed by any agent in the team, we check every τ_e in all sets $\mathbf{E}_\delta^\theta$, for all non-learning agents ($\delta \in \mathbf{\Delta}$) that *have not* just completed τ, to see if there is any *estimator* e that predicts the same task as τ. If there is any e with the same task, we will consider s as the *Choose Target State* $\mathfrak{s}_e$ of e, and will update its target task τ_e accordingly based on the current parameters of the *estimator* $\mathbf{p}_e$ and the type θ of the set.

- ***Checking Suspicious Agents:***In this sub-step, we check the sum of all success rates for each agent $\sum_{e \in \mathbf{E}_\delta^\theta} c_e$. If the result is zero then we increase the value of ζ_δ by 1.

4 Experiments

4.1 Level-based Foraging Domain

We evaluate our approach in level-based foraging, a common problem for evaluating ad-hoc teamwork [2,1,10]. In this domain, a set of collaborative agents must collect items (tasks) displaced in the environment and non-collaborative (adversary) agents surround items and prevent other collaborative agents from reaching items. Each item has a certain weight, and each agent has a certain (unknown) *skill-level*. If the sum of the skill levels of the agents (try to collect an item) that surround a target is greater than or equal to the item's weight, it is "loaded" by the team (Figure 1). Each collaborative agent has 5 possible actions, in a grid-world environment: *North*, *South*, *East*, *West*, and *Load*.

For the *Naive Agents*, the two "leader" types defined in [1]. Additionally, the visibility region of each δ has an angle and a maximum radius, which are unknown. Therefore, there are 3 parameters to be learned for each δ: *Skill-level*, *Angle* and *Radius*. Based on the agent's type and parameters, the target item (task) will be selected. These two types

are $L1$ and $L2$. For $L1$, the target is the furthest visible item that has a lower weight than the agent's skill level. If the agent has the type $L2$, its target will be the visible item with the highest weight below own skill-level, or the item with the highest weight if none are below own level; In both types, the target will be if the agent could not find any item that meets the criteria. After choosing the target, the *naive agents* will move towards the target using the A^* algorithm [6].

Each non-collaborative agent has 5 possible actions, in a grid-world environment: *North*, *South*, *East*, *West*, and *Stay*. In our experiments both the adversary agent and the learning agent have full observation of the whole environment.

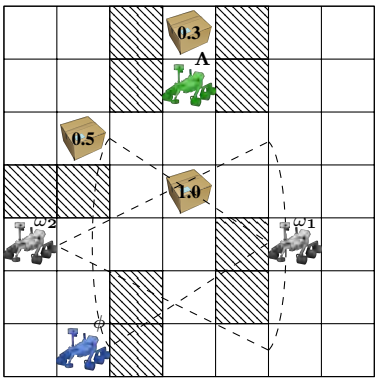


Fig. 1: Level-based foraging domain. There are four agents in three different types in the grid. Boxes are the items that should be collected. Dashed cells in the grid are obstacles.

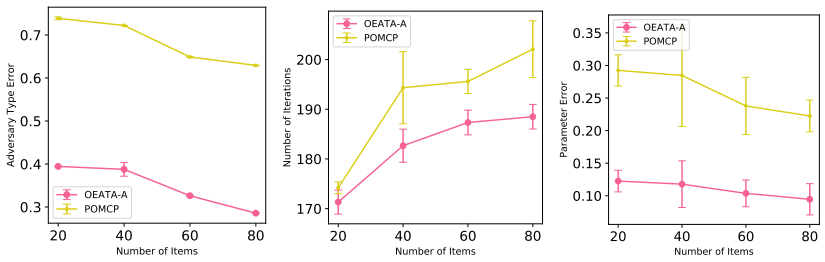
4.2 Results

For evaluating our novel method, we compare our algorithm *OEATA-A* against using POMCP-based Estimation [10] for finding the existence of an adversary in the team. When using POMCP-based Estimation to find the enemy, we still consider that the agent can see the whole environment. However, agent type and parameters are not observable and hence are estimated using POMCP's particle filter. We use $N \times |\mathbf{\Delta}| \times |\mathbf{\Theta}|$ particles, matching the total number of *estimators* in our approach (since we have N per agent, for each type). We executed 100 runs for each experiment and plotted the average results and the confidence interval ($\rho = 0.01$). When we say that a result is significant, we mean statistically significant considering $\rho \leq 0.01$.

OEATA-A used the following parameters: $N = 100$, $t = 2$, $m = 0.2$. Type and parameters of agents in $\mathbf{\Delta}$ are chosen uniformly randomly, and the weight of each item is chosen uniformly randomly (between 0 and 1). Each scenario is also randomly generated. Agent ϕ and agent Λ's skill-level are fixed at 1, so every generated instance is solvable. We ran UCT-H, which introduced [12] for 100 iterations per time step, and a maximum depth of 100. We fixed the scenario size as 20×20, and ran experiments for a varying number of items ($|\mathcal{T}|$). We first show how the learning agent ϕ is recognising the adversary agent among six teammates where the number of non-learning agents is 5 ($|\mathbf{\Omega}| = 5$). The Figure 2 illustrates, the mean absolute error for the type, and $1 - \mathsf{P}(\theta^*)$ we show here the average error across all types.

As it is shown, the type estimation error of OEATA-A is consistently significantly lower than the other algorithm from the second iteration, and it monotonically decreases as the number of iterations increases. POMCP-estimation, on the other hand, does not show any sign of converging to a low error as the number of iterations increases. We can also see that type estimation of OEATA-A becomes quickly better than POMCP, significantly overcoming them after a few iterations. In Figure 3 (a), we showed how finding the adversary among agents works among six agents with a varying number of

items in the grid. In these scenarios, the size of the grid is 20×20. As it is clear, we are significantly better than POMCP-estimation and as the number of items increases the error of finding the enemy decreases.



(a) Type estimation errors (b) Performance (c) Parameter

Fig. 3: Type estimation errors for a varying number of items in full observability.Error estimating agent parameters when there are 7 agents in the team with $|\mathbf{\Omega}| = 5$.

Figure 3 (b) illustrate the performance of the team as the number of items increases in the grid size 20×20, and with the same seven agents in the team where one of them is the learning agent ϕ, one in adversary agent Λ and the other five agents are the naive ones. As we see, with 20 items we are better with a p-value less than 0.05, but as the number of items increases, we can say that we are significantly better. In addition to estimating adversary agents, we need to estimate the parameters of the ω agents as well and the Figure 3 (c), we proved that results for OEATA-A have an error between 0 and 0.15. Additionally, for all number of items we are better than the other method.

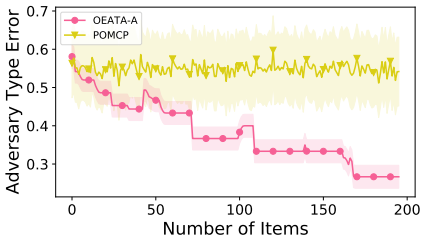


Fig. 2: Error of finding adversary agent when there are 7 agents in the team with $|\mathbf{\Omega}| = 5$.

5 Conclusion

We studied ad-hoc teamwork with an adversary for decentralised task allocation. One ad-hoc agent learns its teammates and could distinguish the opponent agent in the team and despite its existence, makes better decisions concerning overall team performance. We proposed a novel algorithm *On-line Estimator for Ad-hoc Task Allocation with adversary*, that obtained better estimations than previous works in ad-hoc teamwork, leading to better performance. OEATA-A converged to zero error, and in our experiments, the error decreased with the number of iterations. We also showed estimations with partial observability for the first time in ad-hoc teamwork, and still outperform previous

works. In our future works, we are planning to increase the number of the adversary and learning agents to find out how the results would change.

References

1. Albrecht, S., Stone, P.: Reasoning about hypothetical agent behaviours and their parameters. In: Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems. AAMAS'17 (May 2017)
2. Albrecht, S.V., Ramamoorthy, S.: A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. Tech. rep., The University of Edinburgh (February 2013)
3. Barrett, S., Rosenfeld, A., Kraus, S., Stone, P.: Making friends on the fly: Cooperating with new teammates. Artificial Intelligence **242**, 132–171 (2017)
4. Berman, S., Halasz, A., Hsieh, M.A., Kumar, V.: Optimized stochastic policies for task allocation in swarms of robots. IEEE Transactions on Robotics **25**(4) (Aug 2009)
5. Celli, A., Ciccone, M., Bongo, R., Gatti, N.: Coordination in adversarial sequential team games via multi-agent deep reinforcement learning. arXiv preprint arXiv:1912.07712 (2019)
6. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics **4**(2), 100–107 (1968)
7. Lin, J., Dzeparoska, K., Zhang, S.Q., Leon-Garcia, A., Papernot, N.: On the robustness of cooperative multi-agent reinforcement learning. arXiv preprint arXiv:2003.03722 (2020)
8. Mirchevska, V., Luštrek, M., Bežek, A., Gams, M.: Discovering strategic behaviour of multiagent systems in adversary settings. Computing and Informatics **33**(1), 79–108 (2014)
9. Phan, T., Gabor, T., Sedlmeier, A., Ritz, F., Kempter, B., Klein, C., Sauer, H., Schmid, R., Wieghardt, J., Zeller, M., et al.: Learning and testing resilience in cooperative multi-agent systems. In: Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems. pp. 1055–1063 (2020)
10. Shafipour Yourdshahi, E., Do Carmo Alves, M., Soriano Marcolino, L., Angelov, P.: Decentralised task allocation in the fog: Estimators for effective ad-hoc teamwork. In: 11th International Workshop on Optimization and Learning in Multiagent Systems (2020)
11. Uesato, J., Kumar, A., Szepesvari, C., Erez, T., Ruderman, A., Anderson, K., Heess, N., Kohli, P., et al.: Rigorous agent evaluation: An adversarial approach to uncover catastrophic failures. arXiv preprint arXiv:1812.01647 (2018)
12. Yourdshahi, E.S., Pinder, T., Dhawan, G., Marcolino, L.S., Angelov, P.: Towards large scale ad-hoc teamwork. In: 2018 IEEE International Conference on Agents (ICA). pp. 44–49. IEEE (2018)