Decentralised Task Allocation in the Fog: Estimators for Effective Ad-hoc Teamwork

Elnaz Shafipour[1], Matheus Aparecido do Carmo Alves[2],
Leandro Soriano Marcolino[1], Plamen Angelov[1]
[1] School of Computing and Communications, Lancaster University
[2] Institute of Mathematics and Computer Science (ICMC), University of São Paulo (USP)
elnaz.shafipour@lancaster.ac.uk, matheus.aparecido.alves@usp.br,
l.marcolino@lancaster.ac.uk, p.angelov@lancaster.ac.uk

ABSTRACT

In many cases (e.g., rescue) agents complete tasks in non-structured partially observable scenarios, without previous knowledge of others. Known as ad-hoc teams, they work even when agents are from different parties, without prior coordination and communication protocols. We employ parametrised type-based reasoning, where a distribution over agents types is estimated, and parameters for each are learnt. These do not need a large number of observations and can be used on-line. To increase the performance of such teams, we introduce OEATA, a novel algorithm for type and parameter estimation. We show that it converges to perfect estimations, and obtain significantly better performance than the state-of-the-art, in terms of type and parameter errors and task completion, in full and partial observability scenarios.

1 INTRODUCTION

Ad-hoc teamwork is a great model for handling challenging real-world domains, since it allows agents to dynamically collaborate to solve tasks without prior coordination rules nor communication protocols. For instance, consider rescue robots from different organisations urgently brought together to aid in a natural disaster – e.g., earthquake. Designing coordination/communication protocols would take time, and resources; avoiding such delays and funding usage could save lives.

Instead of learning on-line any policy, a common approach in the literature is to consider a set of possible agent types and parameters, reducing the problem to estimating those [4, 7]. This approach is more applicable than learning models from scratch, as it does not require such a large number of observations, and can be more easily applied in an on-line manner. Types could be built based on previous experiences [8] or may be derived from the domain [1]. The introduction of parameters for each type allowed more fine-grained models [2].

However, the previous works were not specifically designed for decentralised task allocation, missing an opportunity to obtain better performances in this relevant scenario for multi-agent collaboration. Note that individual agents do not need to share the same representation of the problem, and run algorithms that explicitly "choose" tasks. They could be developed by different parties, and could use different paradigms. All we need are problems that can be modelled as decentralised task allocation for *our* ad-hoc agent. Similarly, a global allocation algorithm is unfeasible in our scenario: agents developed by others would not necessarily follow commands from a central entity, and we are not assuming any communication protocol.

Hence, we present *On-line Estimators for Ad-hoc Task Allocation* (OEATA), a *novel algorithm* for estimating team-mates types and parameters in decentralised task allocation. We show that our algorithm converges to a perfect estimation when the number of tasks to be performed gets larger. Additionally, we run experiments in a collaborative foraging domain, considering (for the first time) both full and partial observability scenarios, where agents collaborate to collect "heavy" boxes together. We show that we can obtain a lower error in parameter and type estimations in comparison with the state-of-the-art, leading to significantly better performance in task execution.

2 RELATED WORK

Ad-hoc teamwork is an important area in multi-agent systems [5]. We consider *type-based* reasoning, where agents are from a known set of potential types [1, 4, 7, 8], and parameters allow fine-grained models. These parameters, however, must also be estimated in an on-line manner. [2] introduced the *AGA* and *ABU* algorithms, which sample sets of parameters for gradient ascent and *Bayesian* estimation. Focusing on decentralised task allocation, we surpass their parameter and type estimations, leading to better performance. We also extend for the first time the paradigm of learning types and parameters to partial observability scenarios.

Another line of work attempts to identify the task being executed by a team [17]; or an agent's strategy for solving a repetitive task [19]. Our work is different since we focus on a set of (known) tasks which must be completed by the team. Recently, in [10] an ad-hoc agent infers which tasks its team-mates are pursuing and uses that to plan its task. However, they consider disjoint tasks and do not learn team-mates models.

I-POMDP based models [11, 12, 14] allow agents to reason over others in planning. However, they are computationally complex, assuming all agents are learning about others recursively, and they consider agents with individual rewards. We propose a lighter MDP model, with a single team reward, allowing us to tackle larger task allocation scenarios.

POMCP is usually employed in problems with partial observability [18]. However, it is originally designed for a discrete state space, making it harder to apply POMCP for parameter estimation.

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 2020, Auckland, New Zealand

https://doi.org/doi

We apply, however, POMCP in combination with OEATA. We also evaluate experimentally the performance of *plain* POMCP for our problem.

[13] proposed a Bayesian MCTS, sampling different MDP models. Our planning approach (inspired from [2, 8]) is similar, as we sample different agent models from our estimations. However, instead of directly working on the complex transition function space, we learn agents types and parameters, which would then translate to a certain transition probability.

OEATA is inspired by Genetic Algorithms [15]: we keep an *estimator* population, and new ones are generated either randomly or using information from previous ones. However, GAs evaluate all individuals simultaneously at each generation, and usually they are immediately eliminated according to a fitness function. Our *estimators*, on the other hand, are evaluated per agent at task completion, survive according to success rate, and are also used for type estimation. They are also generated in different ways than GA mutation/crossover.

3 METHODOLOGY

As usual in ad-hoc teamwork, we consider one agent ϕ, in the same environment as a set of agents $\mathbf{\Omega}$ ($\phi \notin \mathbf{\Omega}$). ϕ must maximise team performance, but it does not know how agents $\omega \in \mathbf{\Omega}$ may behave at each state. As in previous works [2], we consider that agents in $\mathbf{\Omega}$ can be defined by a type $\theta \in \mathbf{\Theta}$, and by a vector of parameters $\mathbf{p}$, each in a fixed range. Estimating θ and $\mathbf{p}$ allows ϕ to estimate ω's behaviour, leading to better decision-making.

Note that ω agents may be using different algorithms than the ones available in our set of types $\mathbf{\Theta}$. Nonetheless, ϕ would still be able to estimate the best type θ and parameter $\mathbf{p}$ to approximate ω's behaviour. For each type $\theta \in \mathbf{\Theta}$, we will estimate the probability $\mathsf{P}(\theta)_\omega$ of ω having type θ.

We consider ad-hoc teamwork in the context of decentralised task allocation: there is a set of tasks to be completed, and agents are able to autonomously decide which one to perform, without being allocated a task through a centralised mechanism [9]. The *decentralised* allocation is quite natural in ad-hoc teamwork, as we cannot assume that other agents would follow a centralised controller.

We consider that a task may require multiple agents to be performed successfully, and may require multiple time steps to be completed. For instance, in foraging a certain heavy item could require two robots to be collected, and the robots would need to move towards the item.

We formalise the problem as a Markov Decision Process (MDP). Although we have multiple agents, we use a *single agent MDP*, as previous works [2, 20]. Each agent ω is modelled as a probability distribution across actions. They are modelled as the *environment*, affecting the next state and the reward obtained. This model allows us to employ single-agent on-line planning techniques (e.g., UCT [16]).

Hence, we consider a set of states $\mathbf{S}$, a set of actions $\mathbf{A}$, a reward function $\mathbf{R} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \rightarrow [0, 1]$, and a transition function $\mathbf{T} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \rightarrow [0, 1]$. ϕ's objective is 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 receives j steps in the future, and $\gamma \in (0, 1]$ is a discount factor. We consider that given a state s, an agent ω has a (unknown) probability distribution (pdf) across a set of actions $\mathbf{A}_\omega$, which is given by ω's internal algorithm. The actions of all agents define the next state and the reward obtained, but we do not directly represent that. Instead, the pdf of each ω defines the transition function. That is, in the *actual* problem the next state depends on the actions of all agents. However, in the point of view of single-agent MDP, we can see it as the next state being probabilistic, given ϕ's action. The uncertainty comes from the randomness of the actions of ω, besides any stochasticity of the environment.

We assume that given a state s, a type θ, and parameters $\mathbf{p}$, we can obtain the pdf of ω in s. E.g., previous works considered an algorithm "template", which outputs the pdf given θ and $\mathbf{p}$ as input [2]. Hence, the problem of estimating the pdfs reduces to estimating types and parameters for each agent. We also consider that the rewards are obtained by solving tasks τ from a set $\mathcal{T}$. That is, ϕ's reward is $\sum r_\tau$, where r_τ is the reward obtained after completing τ, and the sum goes across all tasks that were completed in a given state (not only completed by ϕ). The number of required agents for a task τ depends on each specific task, and the set of agents that are jointly trying to complete it.

We employ UCT for ϕ's decision-making process, a Monte Carlo Tree Search based approach, which performs multiple *roll-out* simulations in a search tree [16]. At each node, the next child node is decided by the UCB1 equation [6], and rewards are backpropagated up the tree to update action statistics at each node.

Given ϕ's action, the next state depends on the actions of all ω in $\mathbf{\Omega}$. Hence, at each node transition, we sample their actions from their estimated pdfs. Agents' true pdfs are unknown (i.e., we do not know the transition function). As in [2], we sample a θ for each ω from our estimated type probabilities each time we re-visit the root during the tree search. We use the current estimated parameters for the corresponding sampled type. Hence, the higher the quality of our estimations, the better the result of the tree search.

3.1 On-line Estimators for Ad-hoc Task Allocation

The main idea of OEATA is to record all tasks that each agent ω accomplishes, to compare them with the predictions of a set of *estimators*. These contain potential parameters $\mathbf{p}$, of a potential type θ, which are used to predict task selections. The ones that are not able to make good predictions are removed after a number of failures, and replaced by *estimators* that are created using successful ones as a basis, or purely at random.

We assume that when ω completes a task τ in state s_τ, it chooses a new task τ'. Hence, we call s_τ a *choose target state*. We keep track of the history of task selections for each ω: $H_\omega = \{(s^0_\omega, \tau^0), \ldots, (s^n_\omega, \tau^n)\}$, where s^i is the *choose target state* and τ^i is the task that ω completed afterwards. We also keep a set of *estimators* $\mathbf{E}^\theta_\omega$ for each type θ, and agent ω. E.g., if there are 2 potential types and 3 agents in the scenario, there would be 6 sets. $\mathbf{E}^\theta_\omega$ have a fixed size N. An *estimator* e is a tuple, $(\mathbf{p}_e, s_e, \tau_e, c_e, f_e)$: $\mathbf{p}_e$ is a parameter vector; s_e is the last *choose target state*; τ_e is the task that ω would try to complete when having parameter $\mathbf{p}_e$ and type θ; c_e holds the number of times that

e was successful in predicting ω's next task; f_e holds the number of *consecutive* failures.

All e are initialised in the first step, and are updated when a task is completed. We simulate ω's task decision process assuming $\mathbf{p}_e$ as ω's parameters and θ as ω's type. The *estimators* are then iteratively updated depending on their success or failure in estimating the target tasks. Our algorithm has four steps: (i) *Initialisation*; (ii) *Evaluation*; (iii) *Generation*; (iv) *Estimation*. Additionally, an *Update* step is executed for all agents in $\mathbf{\Omega}$, any time a task is completed.

Initialisation: Firstly ϕ creates N *estimators* e for each type θ. If there is a lack of prior information, $\mathbf{p}_e$ of each e can be initialised with a random value (e.g., from the uniform distribution). For all e, s_e is set as the initial state of the environment. Since each e has a certain type θ and a certain parameter vector $\mathbf{p}_e$, it allows ϕ to estimate ω's task decision process in the initial state. The estimated chosen task is assigned as τ_e.

Evaluation: Evaluation of *estimators* $\mathbf{E}^{\theta}_{\omega}$ for a certain agent ω starts when ω completes a task τ_ω (Algorithm 1). For every e in $\mathbf{E}^{\theta}_{\omega}$, we check if τ_e is equal to τ_ω. If they are equal, we save all values p_i in $\mathbf{p}_e$ in a respective bag $\mathbf{b}_i$. Note that i refers to a position in the vector of parameters $\mathbf{p}$, and there is a bag $\mathbf{b}_i$ for each i. $\mathbf{b}_i$ is not erased between iterations, and hence they may increase in size at each iteration. There are sets of bags for each $\mathbf{E}^{\theta}_{\omega}$ (one $\mathbf{b}_i$ for each position i), but for a cleaner notation we refer to them as simply $\mathbf{b}_i$. We update c_e as the number of successes across the history of ω and set f_e to 0. *CheckHistory* counts the number of correct task predictions across the history of ω, assuming type θ and parameter $\mathbf{p_e}$.

Algorithm 1 Evaluating Estimator

```
1:  for all $\theta \in \Theta$ do
2:      for each $e \in \mathbf{E}^{\theta}_{\omega}$ do
3:          if $\tau_\omega = \tau_e$ then                      ▷ $\tau_\omega$ is completed task.
4:              for each $p_i \in \mathbf{p}_e$ do
5:                  $\mathbf{b}_i \leftarrow \mathbf{b}_i \cup p_i$          ▷ Union (∪) with repetition.
6:              $c_e \leftarrow CheckHistory(\omega, \theta_i, \mathbf{p}_e) + 1$; $f_e \leftarrow 0$
7:          else
8:              $f_e \leftarrow f_e + 1$; $c_e \leftarrow c_e - 1$
9:              if $f_e > threshold\ \xi$ then
10:                 $remove\ e\ from\ \mathbf{E}^{\theta}_{\omega}$
11:         $s_e \leftarrow state$; $\tau_e \leftarrow new\ estimated\ target$
```

If τ_e is not equal to τ_ω, we increase f_e and decrease c_e. If f_e is greater than a threshold ξ, we remove e. We also update the *choose target state* (s_e) and τ_e of all e. Line 6 of the algorithm is not $c_e \leftarrow c_e + 1$, because of the decrease when a failure happens in line 8: we decrease c_e when there are consecutive failures, but we restore it to the number of successes when a correct prediction is made. This aids in our type estimation.

We also update ω's history H_ω, in order to use it for future evaluations. We add (s_ω, τ_ω) to the history, where τ_ω is the task just completed and s_ω the corresponding *choose target state* (i.e., when τ_ω was chosen as next task). Note, however, that we have no access to the agent's *true choose target state*. Even though when an agent completes a task, the *choose target state* of all e would be the same s_e, these can later change during the execution (as we will see later). Therefore, we use the *estimators* in $\mathbf{E}^{\theta}_{\omega}$ to estimate s_ω: we set s_ω to the one stored in the *estimator* e with highest c_e across all sets $\mathbf{E}^{\theta}_{\omega}$.

Generation: Let $\mathbf{E'}^{\theta}_{\omega}$ be the new set with only the surviving *estimators* (all e that were not removed in Algorithm 1). We generate $N - |\mathbf{E'}^{\theta}_{\omega}|$ new *estimators*, in order to again have sets $\mathbf{E}^{\theta}_{\omega}$ with size N. Let $\mathbf{p'}$ be a new parameter vector associated with a new e'. A proportion m of the new *estimators* is created by randomly sampling from the uniform distribution in the corresponding parameter's range, for all elements p_i of $\mathbf{p'}$.

For the remaining proportion, we create e' using the bags $\mathbf{b}$. Each position i is populated by randomly sampling from $\mathbf{b}_i$. Every time a new e' is created, we check if $\mathbf{p}_{e'}$ would have at least one success across the history. If so, e' is added to $\mathbf{E'}^{\theta}_{\omega}$, otherwise it is discarded. This decreases the likelihood of wasting an *estimator* with a parameter that would not be able to make any correct prediction (estimated from the history). The process repeats until $|\mathbf{E'}^{\theta}_{\omega}| = N$ (Algorithm 2).

Algorithm 2 Generate New Estimators

```
1:  $\eta \leftarrow 0$; $number\ of\ mutations \leftarrow (N - |\mathbf{E'}^{\theta}_{\omega}|) \times m$
2:  while $|\mathbf{E'}^{\theta}_{\omega}| < N$ do
3:      if $\eta < number\ of\ mutations$ then
4:          $\mathbf{p}_e \leftarrow random\ value$                ▷ $p_i$ uniformly sampled.
5:      else
6:          $\mathbf{p}_e \leftarrow random\ from\ \mathbf{b}$            ▷ $p_i$ sampled from bag $\mathbf{b}$.
7:      $hist_{success} \leftarrow CheckHistory(\omega, \theta, \mathbf{p}_e)$
8:      if $hist_{success} > 0$ then
9:          $Calculate\ \tau_e\ in\ CurrentState$
10:         $c_e \leftarrow hist_{success}$; $s_e \leftarrow CurrentState$; $f_e \leftarrow 0$
11:         $Add\ e\ to\ \mathbf{E'}^{\theta}_{\omega}$; $\eta \leftarrow \eta + 1$
```

Estimation: At each iteration, we use the *average* $\mathbf{p}_e$ across all e in $\mathbf{E}^{\theta}_{\omega}$ as the current estimated parameter for ω, when assuming type θ. We also estimate the probabilities $\mathrm{P}(\theta)_\omega$ of each agent ω having type θ. To do so, we use the success rate of all *estimators* of the corresponding type. For each θ, we first calculate: $k^{\theta}_{\omega} = \max(0, \sum_{e \in \mathbf{E}^{\theta}_{\omega}} c_e)$. These are then normalised as: $k'^{\theta}_{\omega} = \frac{k^{\theta}_{\omega}}{\sum_{\theta' \in \Theta} k^{\theta'}_{\omega}}$. Finally, we update the type estimation: $\mathrm{P}'(\theta)_\omega \propto k'^{\theta}_{\omega} \times \mathrm{P}(\theta)_\omega$, where $\mathrm{P}(\theta)_\omega$ is the previous estimation. In the first iteration, we need prior probabilities for $\mathrm{P}(\theta)$. These would normally be initialised with the uniform distribution, in the absence of previous information.

Update: A certain τ may be completed by any subset of agents (including ϕ), which was the target of a certain ω, who was *not* in the subset of agents that have just completed the task. ω would notice that τ is done by other agents and would switch to a different task at that state. Hence, once τ is completed, we check every τ_e in $\mathbf{E}^{\theta}_{\omega}$, for all ω that *have not* just completed τ, to see if there is any e where $\tau_e = \tau$. If there is any, we set s_e as the current state, and update its target task accordingly based on the parameters $\mathbf{p}_e$ and type θ. That is, we simulate ω choosing a new target task at state s_e.

3.2 Analysis

We show that as the number of tasks goes to infinite, we perfectly identify the type and parameters of all ω, given some assumptions: we consider that parameters have a finite number of decimal places. This is a light assumption, as any real number x can be closely approximated by a number x' with finite precision, without much impact in a real application (e.g., any computer has a finite precision). Hence, as parameters have a fixed range, there is a finite number of possible values for every element p_i. To simplify the exposition, we consider n possible values per element (in general they can have different sizes). Let d be the dimension of the parameter space.

We also assume that any parameter estimation in the wrong type, and any parameter $\mathbf{p'} \neq \mathbf{p}^*$ (even in the correct type) will not succeed infinitely often, where $\mathbf{p}^*$ is the correct parameter. That is, as $|\mathcal{T}| \to \infty$ there will be cases where it successfully predicts the task, but the number of cases is limited by a finite constant c. Let θ^* be the correct type, and $\theta^- \neq \theta^*$.

Proposition 3.1. *OEATA estimates the correct parameter and type for all agents as $|\mathcal{T}| \to \infty$.*

Proof. Because of the mutation proportion m, we always have new *estimators* with random $\mathbf{p}_e$ (since wrong parameters eventually reach the failure threshold, so new ones are generated). As we sample from the uniform distribution, $\mathbf{p}^*$ will be sampled with probability $1/n^d > 0$. Hence, eventually it will be generated as $|\mathcal{T}| \to \infty$. In fact, as the generation defines a Bernoulli experiment, from the geometric distribution we have that in expectation we need n^d trials. Therefore, eventually, there will be an *estimator* with the correct parameter vector $\mathbf{p}^*$, and it will eventually receive the highest c_e score across the full history since it has the highest probability of making correct predictions. Furthermore, all $\mathbf{p}_e \neq \mathbf{p}^*$ will eventually reach the failure threshold ξ, and will eventually be discarded. Hence, when $|\mathcal{T}| \to \infty$ the average across $\mathbf{E}_\omega^{\theta^*}$ will be $\mathbf{p}^*$. Concerning type estimation, we have that $c_e \to \infty$ in the set $\mathbf{E}_\omega^{\theta^*}$ of the true type θ^*. Hence, $k_\omega^{\theta^*} \to \infty$, while $c_e < c$ for $\theta^- \neq \theta^*$ (by assumption). Therefore, $k'^{\theta^*}_\omega = \frac{k_\omega^{\theta^*}}{\sum_{\theta' \in \Theta} k_\omega^{\theta'}} \to 1$, while $k'^{\theta^-}_\omega \to 0$, as $|\mathcal{T}| \to \infty$. Hence, the probability of the correct type $\mathrm{P}(\theta^*) \to 1$. □

If a parameter estimation in the wrong type does succeed infinitely often, then the analysis is more intricate, since we may also have that $k_\omega^{\theta^-} \to \infty$. However, we can still show that the correct type will receive a higher probability:

Proposition 3.2. *If a parameter estimation in the wrong type succeeds infinitely often, OEATA still gives a higher probability to the correct type, for a sufficiently large N.*

Proof. We consider that the correct parameter estimation in the correct type succeeds more frequently than parameter estimations in the wrong type. If that is not true than the wrong type would be an even better model to be used for planning, leading to a contradiction (as θ^* would not be the correct type).

Let $k_\omega^\theta(x)$ denote k_ω^θ for x tasks. By the Stolz–Cesàro theorem, the limit of k'^θ_ω as $|\mathcal{T}| \to \infty$ is: $\lim_{|\mathcal{T}| \to \infty} \frac{k_\omega^\theta(|\mathcal{T}|+1) - k_\omega^\theta(|\mathcal{T}|)}{\sum_{\theta' \in \Theta} k_\omega^{\theta'}(|\mathcal{T}+1|) - \sum_{\theta' \in \Theta} k_\omega^{\theta'}(|\mathcal{T}|)}$. We also have that the parameter estimation in the correct type will eventually converge to the true parameter. Hence, as the correct parameter estimation in θ^* succeeds more frequently than parameter estimations in θ^-, there will be N such that $k_\omega^{\theta^*}(x+1) - k_\omega^{\theta^*}(x) > k_\omega^{\theta^-}(x+1) - k_\omega^{\theta^-}(x)$. Therefore: $\lim_{|\mathcal{T}| \to \infty} k'^{\theta^*}_\omega > \lim_{|\mathcal{T}| \to \infty} k'^{\theta^-}_\omega \Rightarrow \mathrm{P}(\theta^*) > \mathrm{P}(\theta^-)$. □

We saw in Proposition 3.1 that a random search from the mutation proportion takes n^d trials in expectation. OEATA, however, finds $\mathbf{p}^*$ much quicker than that, since a proportion of estimators are sampled from $\mathbf{b}$. To show that formally, we make the following assumptions: (i) a correct value p_i^* in any position i may still predict the task wrongly (since other vector positions may be wrong), but it will eventually predict at least one task correctly in at most t trials, where t is a constant; (ii) although each bag could have at most n different values in the worst case, we assume that they have an expected fixed size $\mathfrak{b}$ as $|\mathcal{T}| \to \infty$ (after removing repetitions of p_i^*).

That is, if one of the vector positions i is correct, $\mathbf{p}$ will not fail infinitely, even though other elements may be incorrect. That is valid in many applications, as in some cases only one element is enough to make a prediction. E.g., if a task is nearby, for almost any vision radius it would be predicted as the next one if the vision angle is correct. On the other hand, wrong values do not succeed infinitely, and hence the bags have an expected size $\mathfrak{b} \ll n$. That is also true in many applications: although by the argument above wrong values may make correct predictions, these are a limited number of cases in the real world. E.g., eventually all tasks nearby will be completed, and a correct vision radius estimation becomes more important to make correct predictions.

Proposition 3.3. *OEATA finds $\mathbf{p}^*$ in $O(d \times n \times \mathfrak{b}^d)$.*

Proof. By the same argument as above, sampling the correct value for element p_i would take n trials in expectation. Once a correct value is sampled, it will be added to $\mathbf{b}_i$ if it makes at least one correct task prediction. It may still make incorrect predictions because of wrong values in other elements, and it would be removed if it reaches the failure threshold ξ. However, for a constant number of trials $t \times n$, it would be added to $\mathbf{b}_i$. Similarly, sampling at least one time the correct value for all d dimensions would take $d \times n$ trials in expectation, and in at most $t \times d \times n$ trials all $\mathbf{b}_i$ would have at least one sample of the correct value in position i. The bags store repeated values, but in the worst case there is only one correct example at each $\mathbf{b}_i$, leading to a $1/\mathfrak{b}$ probability to sample the correct value per bag. Hence, given the bag sampling operation, we would find $\mathbf{p}^*$ with $t \times d \times n \times \mathfrak{b}^d$ trials in expectation. □

Our complexity is close to $O(n)$, instead of $O(n^d)$ as the random search (since $\mathfrak{b} \ll n$). A $O(n)$ complexity could still be prohibitive in some real applications if n is large. However, n can be reduced arbitrarily: a designer can trade convergence speed with model precision, reducing the parameter space by aggregation of values (e.g., reducing the decimals).

3.3 Example

Consider Figure 1: we show one ad-hoc agent ϕ, two agents ω_0, ω_1, and three tasks τ^i. They do foraging: tasks are defined as collecting

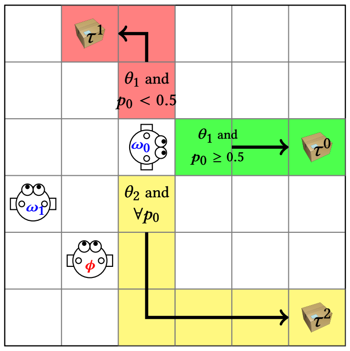

τ^1
θ_1 and $p_0 < 0.5$
ω_0
θ_1 and $p_0 \geq 0.5$
τ^0
ω_f
θ_2 and $\forall p_0$
ϕ
τ^2

Figure 1: ϕ thinking about ω's behaviour in foraging.

a particular item. We assume two possible types: θ_1 and θ_2; two parameters (p_0, p_1), and the following behaviours for ω_0: if type is θ_1, and $p_0 \geq 0.5$, ω_0 goes towards item τ^0. If type is θ_1, and $p_0 < 0.5$, ω_0 goes towards item τ^1. If type is θ_2, $\forall p_0 \in [0, 1]$, ω_0 goes towards item τ^2.

ϕ creates sets $\mathbf{E}^{\theta_1}$ and $\mathbf{E}^{\theta_2}$ for all $\omega \in \mathbf{\Omega}$ (hence 4 sets in total).We start by creating 5 random *estimators* e ($N = 5$). ϕ simulates ω_0's task decision-making for each e in the sets $\mathbf{E}_0^{\theta_1}$ and $\mathbf{E}_0^{\theta_2}$, and obtain its predicted target task τ_e. After some iterations ω_0 gets the item that corresponds to task τ^1, hence completing that task. ϕ then evaluates and updates all e: if τ_e equals τ_1, success counter c_e is increased by 1. Otherwise, counter of consecutive failures (f_e) is increased. Assuming threshold for removing *estimators* $\xi = 1$, some e will survive at $\mathbf{E}_0^{\theta_1}$, while others will be removed. Let's assume that the *estimators* that made a correct prediction had the parameters $(0.4, 0.6)$, and $(0.2, 0.5)$. Hence, the bags for θ_1 will be: $\mathbf{b}_0 = \{0.4, 0.2\}$; $\mathbf{b}_1 = \{0.6, 0.5\}$. Assuming $m = 1/3$, 2 new e will be generated by sampling from these bags, while 1 will be fully random. Therefore, we may create new e with the following $\mathbf{p}_e$: $(0.4, 0.5)$; $(0.2, 0.6)$; $(0.1, 0.7)$, where the last vector is fully random. For $\mathbf{E}_0^{\theta_2}$, since no e predicted τ^1, they will all be removed, and 5 new ones will be generated using the uniform distribution.

Hence, at this iteration, ω_0's p_0 parameter will be estimated as $(2 \times 0.4 + 2 \times 0.2 + 0.1)/5 = 0.26$. ω_0's probability of being θ_1 will be updated as: $k'^{\theta_1} = 2/(2 + 0) \Rightarrow \mathrm{P}(\theta_1) = 1$. Afterwards, given ω_0 new position $(3, 6)$ (next to the box it just collected), ϕ runs again a simulation of ω_0's task decision process for each e in the sets $\mathbf{E}_0^{\theta}$, updating them accordingly.

ω_1's *estimators* are also updated, even though it did not collect any item. Some e in $\mathbf{E}_1^{\theta}$ may have τ^1 as the estimated task τ_e, and that is not a valid task anymore since it was already completed. For each e where $\tau_e = \tau^1$, ϕ simulates ω_1's task decision-making, assuming $\mathbf{p}_e$, and the current state. For these *estimators*, both τ_e and the *choose target state* are updated.

3.4 OEATA in Partial Observability

We define a *single agent* POMDP (extending our previous MDP), allowing us to combine the POMCP algorithm [18] with OEATA: every action a produces an observation $o \in \mathbf{O}$, which represents ϕ's visible environment. POMCP uses a particle filter to approximate the belief state at each node in the UCT tree. Each time we traverse the tree, a state is sampled from the particle filter of the root. Given an action a, a *simulator* samples the next state s' and the observation o. The pair ao defines the next node z in the search tree. State s' is added to z's particle filter, and the process repeats recursively down the tree. We refer to [18] for a detailed explanation. As in UCT, we do not know the true transition and reward functions (since $\mathbf{\Omega}$'s pdfs are unknown). Therefore, we employ the same strategy: each time we go through the search tree, we sample a type for each agent from the estimated type probabilities and use the corresponding estimated parameters. These remain fixed for the whole traversal until we re-visit the root node. These sampled types and parameters are also used in the POMCP *simulator*, when we sample a next state, a reward and an observation after choosing an action in a node. We handle partial observability by sampling a particle from the POMCP root, which corresponds as sampling a state s from the *belief state*. s is then used as the *current state* in OEATA. States that are more likely will be sampled with a higher probability.

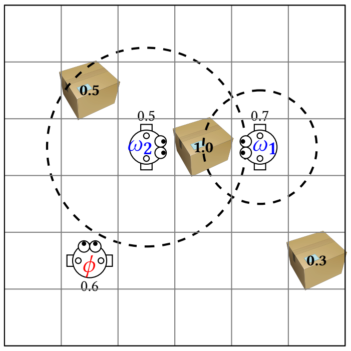
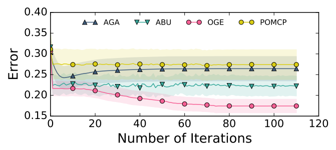


Figure 2: Level-based foraging domain. The number next to the boxes indicate their "weight", and the one next to agents indicate their skill levels.

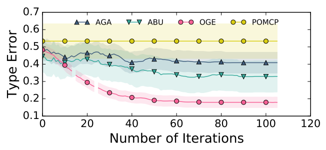
4 EXPERIMENTS

4.1 Level-based Foraging Domain

We evaluate our approach in level-based foraging, a common problem for evaluating ad-hoc teamwork [2, 3, 20]. In this domain, a set of agents must collect items displaced in the environment (which corresponds to the "tasks" in our model). 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 (that are trying to collect an item) surrounding a target is greater than or equal the item's weight, the item is "loaded" by the team (Figure 2). Each agent has 5 possible actions, in a grid-world environment: *North*, *South*, *East*, *West*, and *Load* (which tries to load an item next to the agent, if the agent is facing that item). We consider two possible agent types, taken from previous works in this domain: We use the two "leader" types defined in Albrecht and Stone [2]. Additionally, the visibility region of each ω has an angle and a maximum radius, which are unknown. Therefore, there are 3 parameters to be learned for each ω: *Skill-level*, *Angle* and *Radius*. According to ω's type and parameters, its target item (task) will be selected. We refer the reader to Albrecht and Stone [2] for a detailed description of the types and their respective parameters.



(a) Parameter error



(b) Type error

Figure 3: Parameter and type estimation errors for $|\Omega| = 5$. Lower is better.

4.2 Results

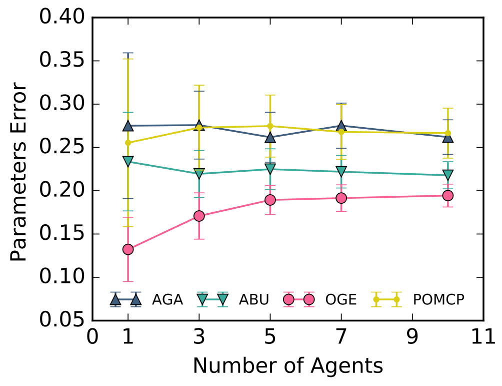
We will compare our algorithm (*OGE*) against two state-of-the-art parameter estimation approaches in ad-hoc teamwork: *AGA* and *ABU* [2]. Both approaches also sample sets of parameters (for a gradient ascent step or a *Bayesian* estimation), and we use the same set size as *estimator* sets (N). Additionally, we compare our approach against using *plain* POMCP for type and parameter estimations. In this case we still consider that the agent is able to see the whole environment; however, agent type and parameters are not observable, and hence are estimated using POMCP's particle filter. We use $N \times |\Omega| \times |\Theta|$ 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 plot the average results and the confidence interval ($\rho = 0.01$). When we say that a result is significant, we mean statistically significant considering $\rho \leq 0.01$.

In *plain* POMCP, we calculate the type probability of a certain agent $P(\theta)$ by counting the frequency that the type θ is assigned to the agent in the root's particle filter. For the parameter estimation, we use the average across the particle filter (for each type), allowing us to calculate the parameter estimation error in our evaluation.

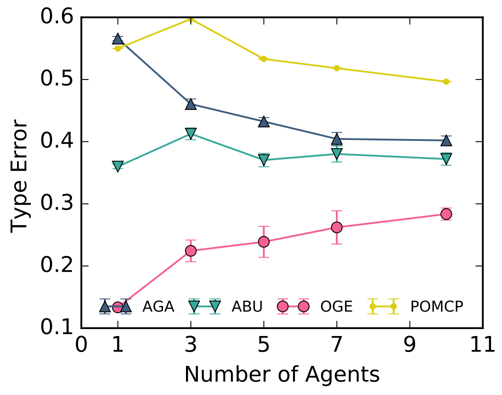
OGE used the following parameters: $N = 100$, $t = 2$, $m = 0.2$. Type and parameters of agents in Ω are chosen uniformly randomly, and the weight of each item is chosen uniformly randomly (between 0 and 1). Each scenario is also randomly generated. Agent ϕ's skill-level is fixed at 1, so every generated instance is solvable. We ran UCT for 100 iterations per time step, and maximum depth 100. We fix the scenario size as 20×20, and ran experiments for a varying number of team-mates ($|\Omega|$). We first show results when ϕ has full observability of the scenario, and later we will show with partial observability. When running experiments in partial observability, we consider a circular visibility region centered on ϕ, with radius 5.

We first show examples of the parameter and type error for $|\Omega| = 5$ (Figure 3). We evaluate the mean absolute error for the parameters, and $1 - P(\theta^*)$ for type; and we show here the average error across all parameters. As we can see, our parameter estimation error is consistently significantly lower than the other algorithms from the second iteration, and it monotonically decreases as the number of iterations increases (as may be expected from Proposition 3.1). AGA, ABU, and POMCP, on the other hand, do not show any sign of converging to a low error as the number of iterations increases. We can also see that our type estimation becomes quickly better than the other algorithms, significantly overcoming them after a few iterations.

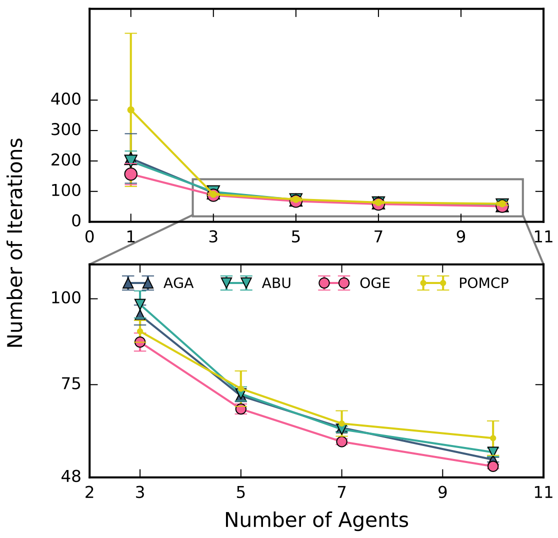
We now study the error for different $|\Omega|$ sizes, and the execution time to complete all tasks (in terms of number of iterations), in



(a) Parameter error

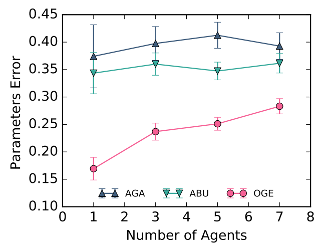


(b) Type error

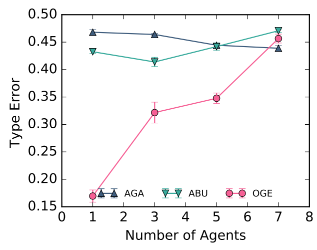


(c) Performance

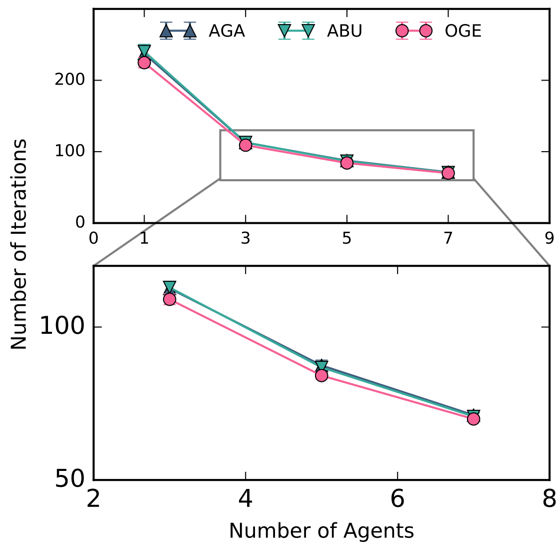
Figure 4: Parameter, type estimation errors, and performance for a varying number of agents in full observability. The graph in the bottom shows results with more than 3 agents in greater detail.



(a) Parameter error



(b) Type error



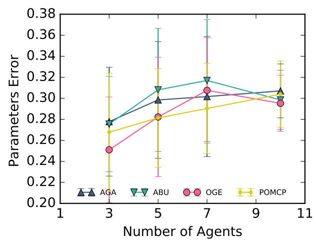
(c) Performance

Figure 5: Parameter, type estimation errors, and performance for a varying number of agents with partial observability. The graph in the bottom shows results with more than 3 agents in greater detail.

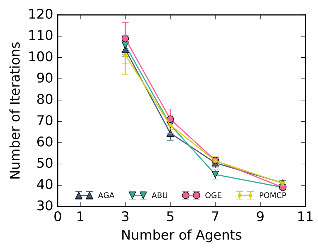
Figure 4. Since we are aggregating several results, we plot in Figure 4 (a) and (b) the average error across all iterations. As we can see, OGE has consistently lower error than the other algorithms, both in terms of parameters and type estimation. In fact, OGE is significantly better than AGA, ABU and POMCP in terms of parameter and type error for all number of agents. Similarly, in Figure 4 (c) we see that OGE is able to complete all tasks faster for all team sizes, and is significantly better in almost all cases (except for POMCP with 3 agents, where $\rho \le 0.2$; and with 1 agent we are significantly better than ABU and POMCP with $\rho \le 0.022$, and AGA with $\rho \le 0.12$). Note that the performance changes drastically when the number of team-mates is greater than 1. Therefore, we also show a second plot in the same figure with a more detailed view with 3 or more agents, for better visualisation.

We show our results for partial observability scenarios in Figure 5. Here all approaches (AGA/ABU/OGE) are now employing the modified POMCP for handling the partial observability, as described in Section 3.4. Again, we obtain significantly lower parameter and type error than previous approaches (Figure 5 (a) and (b)), except only for type error with 7 agents. This happens because the number of items is fixed, hence the higher the number of agents the lower the number of observations that we get for each agent for parameter and type learning. Similarly, in Figure 5 (c) we see that we obtain a better performance than previous approaches, although the impact is smaller than in full observability case: results are significantly better with 1 team-mate; with 3 it holds with $\rho \le 0.06$; while with 5 agents we are significantly better than AGA with $\rho \le 0.013$. With 7 agents are still significantly better than AGA with $\rho \le 0.03$. As before, we show a detailed view with 3 or more agents in the figure on the bottom, for better visualisation, because of the big decrease in the number of iterations with more than 1 agent.

Finally, we study the impact of using these algorithms when ϕ *does not* have full knowledge on the possible types of the other agents. That is, we run experiments where all agents in Ω have a type which is not in Θ. In these experiments, all agents ω follow one of the "follower" agent types (F1) from Albrecht and Stone [2]. Since this type looks at others to decide its target, we run experiments with 3 or more team-mates. We can see our results in Figure 6. As we can see, in this case all algorithms have similar results, and the parameter error is not significantly different across the algorithms. Finally, the overall task performance is not significantly different across all algorithms in most cases (except for AGA with 5 agents, and ABU with 7 agents, which are better than us in these cases). Therefore, we show that we are able to obtain significantly better results than the state-of-the-art when there is a good model of the possible team-mates types. When such models are not available, our performance is similar to previous works in most cases.



(a) Parameter error



(b) Performance

Figure 6: Parameter, type estimation errors, and performance, for a varying number of agents when correct agent type is not available, in full observability scenarios.

5 CONCLUSION

We study ad-hoc teamwork for decentralised task allocation. One ad-hoc agent learns its team-mates, in order to better decisions concerning overall team performance. We propose a novel algorithm that obtains better estimations than previous works in ad-hoc teamwork, leading to better performance. OEATA converges to zero error, and in our experiments the error decreases with the number of iterations. We also show estimations with partial observability for the first time in ad-hoc teamwork, and still outperform previous works in this challenging scenario.

REFERENCES

[1] S. Albrecht, J. Crandall, and S. Ramamoorthy. 2015. An empirical study on the practical impact of prior beliefs over policy types. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*.

[2] S. Albrecht and P. Stone. 2017. Reasoning about Hypothetical Agent Behaviours and their Parameters. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'17)*.

[3] Stefano V. Albrecht and Subramanian Ramamoorthy. 2013. *A Game-Theoretic Model and Best-Response Learning Method for Ad Hoc Coordination in Multiagent Systems*. Technical Report. The University of Edinburgh.

[4] S. V. Albrecht and S. Ramamoorthy. 2016. Exploiting causality for selective belief filtering in dynamic Bayesian networks. *Journal of Artificial Intelligence Research* 55 (2016).

[5] S. V. Albrecht and P. Stone. 2018. Autonomous Agents Modelling Other Agents: A Comprehensive Survey and Open Problems. *Artificial Intelligence (AIJ)* 258 (2018), 66–95.

[6] P. Auer, N. Cesa-Bianchi, and P. Fischer. 2002. Finite time analysis of the multi-armed bandit problem. *Machine Learning* 47, 2–3 (2002), 235–256.

[7] Samuel Barrett, Peter Stone, and Sarit Kraus. 2011. Empirical Evaluation of Ad Hoc Teamwork in the Pursuit Domain. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*.

[8] S. Barrett, P. Stone, S. Kraus, and A. Rosenfeld. 2013. Teamwork with limited knowledge of teammates. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*.

[9] S. Berman, A. Halasz, M. A. Hsieh, and V. Kumar. 2009. Optimized Stochastic Policies for Task Allocation in Swarms of Robots. *IEEE Transactions on Robotics* 25, 4 (Aug 2009).

[10] Shuo Chen, Ewa Andrejczuk, Athirai A. Irissappane, and Jie Zhang. 2019. ATSIS: Achieving the Ad hoc Teamwork by Sub-task Inference and Selection. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 172–179. https://doi.org/10.24963/ijcai.2019/25

[11] P. Doshi, Y. Zeng, and Q. Chen. 2009. Graphical models for interactive POMDPs: Representations and solutions. *JAAMAS* 18, 3 (2009), 376–416.

[12] P. Gmytrasiewicz and P. Doshi. 2005. A framework for sequential planning in multiagent settings. *JAIR* 24 (2005), 49–79.

[13] A. Guez, D. Silver, and P. Dayan. 2013. Scalable and Efficient Bayes-Adaptive Reinforcement Learning Based on Monte-Carlo Tree Search. *Journal of Artificial Intelligence Research (JAIR)* 48 (2013).

[14] T. N. Hoang and K.H. Low. 2013. Interactive POMDP Lite: Towards practical planning to predict and exploit intentions for interacting with self-interested agents. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI)*.

[15] John H. Holland. 1992. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA.

[16] L. Kocsis and C. Szepesvári. 2006. Bandit based Monte-Carlo Planning. In *Proceedings of the 17th European Conference on Machine Learning*.

[17] Francisco S. Melo and Alberto Sardinha. 2016. Ad hoc teamwork by learning teammates' task. *Autonomous Agents and Multi-Agent Systems* 30, 2 (2016).

[18] David Silver and Joel Veness. 2010. Monte-Carlo Planning in Large POMDPs. In *Proceedings of the Twenty-Fourth Annual Conference on Neural Information Processing Systems*.

[19] Maulesh Trivedi and Prashant Doshi. 2018. Inverse Learning of Robot Behavior for Collaborative Planning. In *Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[20] E. S. Yourdshahi, T. Pinder, G. Dhawan, L. S. Marcolino, and P. Angelov. 2018. Towards Large Scale Ad-hoc Teamwork. In *Proceedings of the 3rd International Conference on Agents*.