Towards Large Scale Ad-hoc Teamwork

Elnaz Shafipour Yourdshahi
School of Computing and Communications
Lancaster University
Lancaster, UK
elnaz.shafipour@lancaster.ac.uk

Thomas Pinder
School of Computing and Communications
Lancaster University
Lancaster, UK
t.pinder2@lancaster.ac.uk

Gauri Dhawan
Computer Science Department
Vellore Institute of Technology
Chennai, India
dhawan.gauri@gmail.com

Leandro Soriano Marcolino
School of Computing and Communications
Lancaster University
Lancaster, UK
l.marcolino@lancaster.ac.uk

Plamen Angelov
School of Computing and Communications
Lancaster University
Lancaster, UK
p.angelov@lancaster.ac.uk

***Abstract*—In complex environments, agents must be able to cooperate with previously unknown team-mates, and hence dynamically learn about other agents in the environment while searching for optimal actions. Previous works employ Monte Carlo Tree Search approaches. However, the search tree increases exponentially with the number of agents, and only scenarios with very small team sizes have been explored. Hence, in this paper we propose a history-based version of UCT Monte Carlo Tree Search, using a more compact representation than the original algorithm. We perform several experiments with a varying number of agents in the level-based foraging domain, an important testbed for ad-hoc teamwork. We achieve better overall performance than the state-of-the-art and better scalability with team size. Additionally, we contribute an open-source version of our system, making it easier for the research community to use the level-based foraging domain as a benchmark problem for ad-hoc teamwork.**

***Index Terms*—Collaborative Intelligence, Learning (Artificial Intelligence), Algorithms**

I. Introduction

Agents are able to accomplish great tasks with joint collaborative work. However, smart autonomous agents must be able to effectively collaborate with any team, instead of assuming pre-programmed coordination rules; leading to the exciting research challenge known as “ad-hoc teamwork” [1].

Recent research in ad-hoc teamwork considered potential team-mates *types*, which are continuously estimated in an *online* manner [2]–[4]. Type-based reasoning allows an agent to re-use previous knowledge (when interacting with different teams) and is much faster than learning models from scratch. Meanwhile, a Monte Carlo Tree Search (MCTS) approach is usually employed to estimate best actions, given the current type estimations [5], [6]. However, the uncertainty over team-mates actions leads to a combinatorial explosion on the number of possible next states, leading to an exponential number of possible children for any given node in the search tree.

Another approach for ad-hoc teamwork represents team-mates as probabilistic deterministic finite state controllers [7], and employ a variation of the POMCP algorithm for estimating best actions [8]. However, they still struggle with scalability, and present results limited to only two agents.

Therefore, in order to enable large scale ad-hoc teamwork, we first formalize the problem as a Markov Decision Process (MDP), and then propose UCT-H, a new version of UCT Monte Carlo Tree Search, using a history-based compact representation. We evaluate our approach in the level-based foraging domain, which has commonly been used to evaluate ad-hoc teamwork research [6], [9], but with larger team sizes than what has been explored before. We evaluate overall task performance, computational time, and memory usage. We find that our compact representation achieves better results than the previous MCTS approaches for any team size, and scales better with the number of agents. In fact, we show that the difference in performance between UCT-H and UCT tends to increase as the number of agents grows, reaching 65% better performance with 10 agents; and the memory usage of UCT-H is roughly constant, while UCT’s memory usage increases exponentially.

As an additional contribution, our implementation of level-based foraging (including all MCTS algorithms) is freely available to the community as open source, encouraging further research using this domain as an important ad-hoc teamwork benchmark.

II. Related Work

Ad-hoc teamwork is a very active topic of research, as shown in a recent survey on autonomous agents dynamically modelling other agents [10]. Our research fits within the *type-based reasoning* approach, where it is assumed that agents are from a known set of potential types, and we must estimate each team-mate’s type [2]–[6]. The type-based reasoning is important for ad-hoc teamwork, since it is much faster than learning models from scratch, and the types could be built based on previous interactions with different teams.

In particular, Barrett et al. (2011) [5] introduced the idea of sampling types for each agent, based on the current beliefs, at each roll-out iteration of the UCT Monte Carlo Tree Search method [11]. Albrecht and Stone (2017) [6] employ a similar search technique, but they consider that parameters may affect the behavior of each type, and they introduced techniques for dynamically estimating these parameters. However, even though type-based reasoning should be more scalable than learning models from scratch, the previous type-based works still did not go beyond four agents (while the "traditional" pre-coordinated teams can easily scale to hundreds of agents (e.g., [12]–[14])). Therefore, it is essential to design more scalable ad-hoc teamwork techniques.

With regards to search approaches, POMCP is a very famous extension of the traditional UCT Monte Carlo Tree Search, when considering *partially observable* environments [8]. Recently, Panella and Gmytrasiewicz (2017) [7] proposed an extension of POMCP for ad-hoc teamwork, when agents can be represented by probabilistic deterministic finite state controllers. However, their approach still does not scale easily to a large number of agents. In fact, their results are limited to only two agents (the main planning agent, and a single unknown agent).

Another possible approach for scalability in ad-hoc teamwork is to learn a single model for a *team* of agents, instead of individual models for each agent. For instance, in the RoboCup soccer domain, Riley and Veloso (2002) [15] propose a method to identify the type of an adversarial team, which defines probabilities for agents locations in the field. Similarly, Barrett and Stone (2015) [16] assume a series of previous games with potential teams, which are used to train team policies. Then, at execution time, the most likely current team is estimated, and its corresponding policy executed. Obviously, however, a single team model is less flexible than learning models for each individual agent.

In this paper, we learn a model for each team-mate, and propose a modification for the UCT Monte Carlo Tree Search algorithm, inspired by the representation strategy used in POMCP [8]. However, our compact representation is aimed at scalability in ad-hoc teamwork, instead of handling partial observability. Additionally, in POMCP it is assumed full knowledge of the transition function (embedded in a "black-box simulator"), while in our work the states are sampled from an estimated transition function, according to the current estimations of types and parameters for each agent. Our approach is used for finding optimal actions while dynamically learning types and parameters, as in Albrecht and Stone (2017) [6], but leads to a significantly better performance, and scales better in terms of memory usage with team size.

III. Methodology

A. Ad-hoc Teamwork Model

The ad-hoc teamwork problem can be modelled a *Stochastic Game*, given the presence of many agents acting in the same environment, with fully distributed controllers (which we assume may or may not be cooperative).

That is, let m be the number of agents, $\mathbf{S}$ a set of states, $\mathbf{A}_i$ the set of actions available to agent i, and $\mathbf{A}$ the joint action space $\mathbf{A}_1 \times \mathbf{A}_2 \times \ldots \times \mathbf{A}_m$. We consider a transition function $\mathbf{T}$: $\mathbf{S} \times \mathbf{A} \times \mathbf{S} \to [0,1]$, and a reward function $\mathbf{R}_i$ for each agent: $\mathbf{S} \times \mathbf{A} \to \mathbb{R}$. Hence, given a state s and the actions of all agents, there is a probability distribution across next states s', and an individual reward r_i for each agent.

In order to employ Monte Carlo Tree Search on-line planning techniques, we will model the problem, *under the point of view of an agent* ϕ, as a Markov Decision Process (MDP). That is, we consider a set of agents $\mathbf{\Omega}$, which act in the same world as ϕ, affecting the reward obtained at each state. However, ϕ can only decide its own actions and has no control over the actions of agents in the set $\mathbf{\Omega}$. Additionally, each agent $\omega \in \mathbf{\Omega}$ has a probability distribution (pdf) over $\mathbf{A}_\omega$, given the current state s. Under the MDP point of view, we consider that a state s may include not only the "external" environment state, but also internal states of agents in $\mathbf{\Omega}$. The actions of all agents define the probabilities of the next potential states s', and the obtained ϕ's reward r_ϕ. Hence, *under the point of view of* ϕ, the problem can be modelled as a single-agent MDP, where the uncertainty over the actions of the other agents is embedded in a transition function $\mathbf{T}$ and in a stochastic reward function $\mathbf{R}$. Hence, we consider a set of states $\mathbf{S}$, a set of actions $\mathbf{A}_\phi$, a reward function $\mathbf{R} : \mathbf{S} \times \mathbf{A}_\phi \times \mathbb{R} \to [0,1]$, and a state transition function $\mathbf{T} : \mathbf{S} \times \mathbf{A}_\phi \times \mathbf{S} \to [0,1]$. Note that we consider here stochastic rewards, and $\mathbf{R}$ gives the probability of a reward r_ϕ, given a state s and ϕ's action a.

As usual, we consider that ϕ's objective is to find the optimal value function, which maximizes the expected sum of discounted rewards $E[\sum_{j=0}^{\infty} \gamma^j r_{t+j}]$, where t is the current time and r_{t+j} is the reward ϕ receives j steps in the future. The discount factor $\gamma \in [0,1]$ defines how much ϕ considers future rewards.

Note that given the ad-hoc teamwork context, ϕ's reward function $\mathbf{R}$ *is not necessarily the one of a "selfish" agent*. That is, even though ϕ maximizes its own value function, the rewards obtained can be defined according to team performance metrics, instead of being focused only on ϕ's performance (as we will see in our experiments in Section IV).

The transition and reward functions ($\mathbf{T}$, $\mathbf{R}$) are not known in advance since ϕ does not know the pdfs of the agents in $\mathbf{\Omega}$. As in Albrecht and Stone (2017) [6], we assume that each $\omega \in \mathbf{\Omega}$ may be from a set of possible types $\mathbf{\Theta}$. Additionally, each type has a vector of parameters $\mathbf{p}$. Given the true type θ_ω and the true parameters $\mathbf{p}_\omega$ of an agent ω, the agent ϕ would be able to determine ω's pdf at each state s (since it could also keep track of ω's internal state). However, since θ and $\mathbf{p}$ are unknown for all agents in $\mathbf{\Omega}$, ϕ must continuously estimate a probability distribution across all types and parameters for each $\omega \in \mathbf{\Omega}$, and must also continuously estimate the internal state of all $\omega \in \mathbf{\Omega}$. ϕ's estimations lead to estimated transition and reward functions: $\tilde{\mathbf{T}}$, $\tilde{\mathbf{R}}$. Additionally, since the internal state of agents in $\mathbf{\Omega}$ is not directly observable, ϕ maintains an estimated current state $\tilde{s}$.

As our main contribution is a new search approach for large scale ad-hoc teamwork, we refer the reader to Albrecht and Stone (2017) [6] for the methods employed for estimating types, parameters, and internal states for all agents.

B. Monte-Carlo Tree Search

Although Albrecht and Stone (2017) did not explicitly formalize the ad-hoc teamwork problem as a MDP [6], they employed a traditional UCT Monte Carlo Tree Search [11] (which are used for solving MDPs in an online fashion). That is, x simulations are performed in a search-tree, where each node n represents a state s. The root node represents the current state, and each simulation expands a next state s' at each node, until a certain limit horizon l. When the limit horizon is reached, the rewards are back-propagated, and a new simulation is started again from the root node.

Each node holds a *Q-table*, where the average value of each action across all simulations is stored, and the UCB1 algorithm [17] is employed when deciding which action to simulate at each node. After all x simulations are performed, the agent is able to estimate the best action to take, by considering the *Q-table* at the root note. Once a new state is reached, the whole algorithm is repeated, in order to decide the next action in that new state. A pseudo-code of the original UCT algorithm is shown in Algorithm 1.

```
1  Function UCT(state):
2      repeat
3          Search (state, 0)
4      until Timeout;
5      return bestAction(state, 0)
6  Function Search(state, depth):
7      if Terminal(state) then return 0;
8      if Leaf(state, depth) then return Evaluate(state);
9      action := selectAction(state, depth)
10     (nextState, reward) := simulateAction(state, action)
11     q := reward + $\gamma$ Search (nextState, depth + 1)
12     UpdateValue(state, action, q, depth)
13     return q
```

Algorithm 1: Original UCT algorithm.

Note that in line 10, the next state and reward is simulated, which leads to the next node to be explored in the search tree. Because of the estimated transition function $\tilde{\mathbf{T}}$ (i.e., the possible actions of all agents $\mathbf{\Omega}$), given the current state s and a ϕ's action a_ϕ to be simulated, there is a set of possible next states s' (and a correspondingly set of next nodes n'). If every $\omega \in \mathbf{\Omega}$ assigns a non-zero probability to every action in $\mathbf{A}_\omega$, there are precisely $\prod_{\omega\in\mathbf{\Omega}} |\mathbf{A}_\omega|$ possible next states s' for each s. Hence, the search tree grows exponentially with the number of agents $|\mathbf{\Omega}|$ (if every agent ω has the same number of available actions, we find $|\mathbf{A}_\omega|^{|\mathbf{\Omega}|}$ possible next states for each s).

Therefore, in this paper, we propose UCT-H, a modification over the original UCT algorithm for large scale ad-hoc teamwork. Our main idea is to represent a *history* at every node n in the search tree, instead of a single state s. That is, instead of a node n representing a specific state s, it will represent a sequence of actions $a_0, a_1, \dots, a_{d-1}$, taken from the root up to the current depth d. Hence, all the possible states reachable from the root by the sequence of actions $a_0, a_1, \dots, a_{d-1}$ will be represented by exactly the same node n[1].

Note that the root node still represents a unique state s_0. Each time we simulate taking an action a from the root towards a child node n', we will sample the next state s' by simulating taking action a in state s_0. Similarly, each time we go down from a node n to a child node n', by taking action a, we will sample the next state s' by simulating taking action a in state s (which will be fully determined by the current sequence of action simulations up to n). We re-start the process each time we go back to the root node for a new simulation. Therefore, at each simulation, the same node may represent different states. Consequently, instead of each node storing a *Q-Table* with action-value pairs $Q(s, a)$ for a certain state s, we will store action-values $Q(h, a)$ for each *history* h.

In Algorithm 2, we present UCT-H. We also illustrate the difference between UCT and UCT-H in Figure 1, assuming a problem with two possible actions, and two possible next states per action. We only show the root and the nodes immediately below the root in the figure. Note that in the extract of the tree shown, the original UCT has 5 *Q-Tables* (one for each state), while UCT-H has only 3 *Q-Tables* (one for each history).

```
1  Function UCT-H(state):
2      root = new Node
3      repeat
4          Search (state, root, 0)
5      until Timeout;
6      return bestAction(root, 0)
7  Function Search(state, node, depth):
8      if Terminal(state) then return 0;
9      if Leaf(state, depth) then return Evaluate(state);
10     action := selectAction(node, depth)
11     (nextState, reward) := simulateAction(state, action)
12     nextNode := child(node, action)
13     q := reward + $\gamma$ Search (nextState, nextNode, depth
        + 1)
14     UpdateValue(node, action, q, depth)
15     return q
```

Algorithm 2: Our proposed history-based UCT (UCT-H).

Note, however, that in our case we do not have the true MDP model, as mentioned in the previous section, and hence the simulator used in the search tree (line 10 and line 11 for UCT and UCT-H, respectively) does not match the true problem, for both UCT and UCT-H. This happens because the transition and reward functions ($\mathbf{T}$, $\mathbf{R}$) depend on the pdfs over actions given by the agents in $\mathbf{\Omega}$. These pdfs, however, are a function of the type, parameter and internal state of each $\omega \in \mathbf{\Omega}$, which are unknown.

As in Albrecht and Stone (2017) [6], each time we re-start a simulation from the root node, we sample a type for each agent from our estimated type probabilities, which remains

[1]We still consider that these are ϕ's actions, but we drop the ϕ subscript for notation convenience.

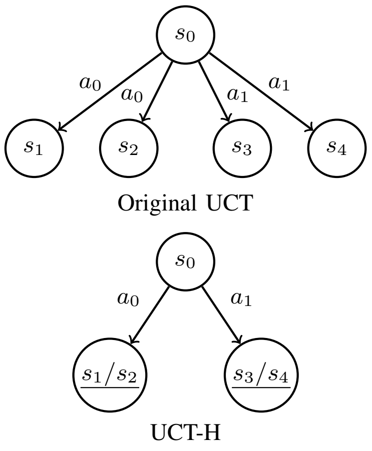


Fig. 1: Illustration of original UCT, and our modified version (UCT-H). The same action may lead to different states, given the uncertainty in the environment. Underlined states will be sampled at each simulation, according to the estimated transition function. UCT-H uses a single *Q-Table* for each node, instead of a *Q-Table* for each state.

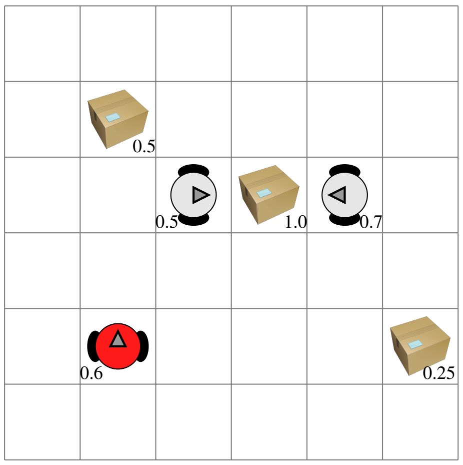


Fig. 2: Level-based foraging domain. The number next to the boxes indicate their "weight", and the one next to agents indicate their skill levels. The red (dark) agent runs MCTS to plan actions, and must estimate the type and parameters of the other agents.

fixed for each agent for that simulation (i.e., until we reach the limited horizon l), and is re-sampled next time a simulation is re-started from the root node. Given a type, we use the current estimated parameters when sampling the agents' pdfs in order to simulate the reward r_ϕ and the next state s'.

IV. Experiments

A. Level-based Foraging Domain

We ran experiments in the level-based foraging domain, which is a common problem for evaluating ad-hoc teamwork [6], [9]. Based on the descriptions in Albrecht and Stone (2017) [6], we have implemented our own open source version[2].

The main idea of the domain is that a set of agents must collaboratively collect items from an environment. Each item has a certain weight, and each agent has a certain (unknown) *skill-level*. We consider a grid-world environment, and hence each agent has 5 possible actions: *North*, *South*, *East*, *West*, and *Load*. The *Load* action attempts to load an item next to the agent if the agent is currently facing that item. When the sum of the skill-levels of the agents (that issued the load action) surrounding a target is greater than or equal the item's weight, the item is "loaded" by the team. Therefore, this domain requires close collaboration between agents, being well suited for ad-hoc teamwork evaluations (Figure 2 shows an illustration).

As in Albrecht and Stone (2017) [6] we consider four possible types for the agents in $\boldsymbol{\Omega}$: two "leader" types, which choose items in the environment to move towards, and two "follower" types, which attempt to go towards the same items as other agents, in order to help them load items. Additionally, each ω's field of vision has an angle and a maximum radius (which are unknown). Based on ω's type and parameter values, ω's target item or target agent (if "follower" type) will be selected, and ω's internal state will be set to the position of that target. Afterwards, ω will move towards the target using the A^* algorithm [18]. Due to space constraints, we refer the reader to [6] for a precise description of the four types and their respective parameters.

As mentioned, we consider an agent ϕ using MCTS to decide upon actions, given its current estimations of its team-mates types, parameters and internal states. Note that the reward r_ϕ for agent ϕ is the number of boxes collected by *all agents at each state transition*, not only the number of boxes collected by ϕ. Hence, ϕ's objective is to maximize the overall team performance. We will compare the original UCT algorithm (UCT) against our history-based UCT version (UCT-H).

B. Results

In this section, we evaluate the overall performance, computational time and memory usage of UCT and UCT-H. We evaluate each execution of the algorithms in randomly generated scenarios. We run 15 executions per experiment and plot the average results. Error bars show the 90% confidence interval. Additionally, when we say that one result is "significantly better" than another, we mean better with statistical significance, considering $p < 0.1$.

We evaluated the performance across several numbers of agents ($|\boldsymbol{\Omega}|$), with the scenario size fixed to 20×20. We consider "performance" as the number of time steps required to collect all items in the scenario (hence, the lower the better). For both UCT and UCT-H, we performed 100 simulations for each state, and considered a limit horizon 100. Additionally, we used discount factor 0.95, and UCB1 exploration constant $\frac{0.5}{\sqrt{2}}$.

We show results for UCT and UCT-H using two different parameter estimation approaches: Approximate Gradient Ascent

[2]Available at https://github.com/ElnazShy/MultiAgents

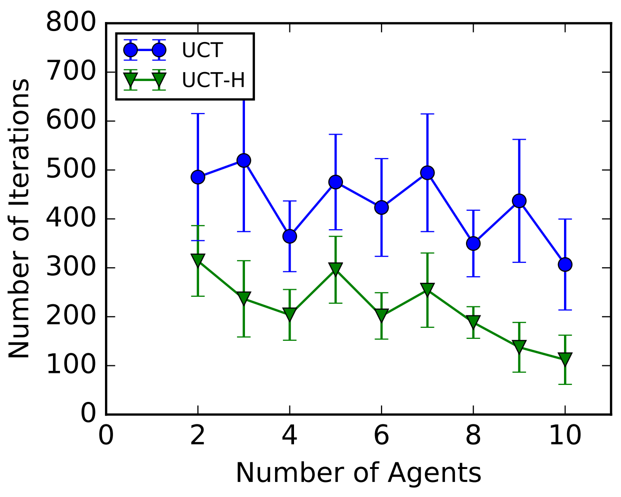


Fig. 3: Performance of different MCTS algorithms as the number of agents increases (the lower the better).

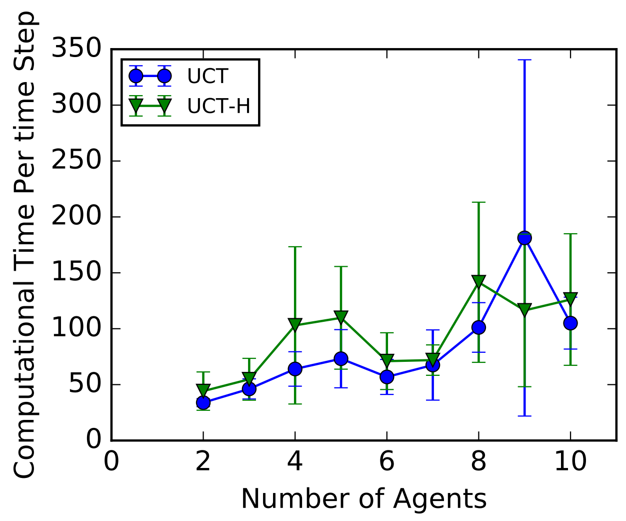


Fig. 4: Computational time of MCTS algorithms as the number of agents increases.

(AGA) and Approximate Bayesian Updating (ABU), from [6]. We do not consider the Exact Global Optimization approach since it is significantly more computationally expensive than the other two.

In Figure 3 we show the results for an increasing number of agents ($|\boldsymbol{\Omega}|$). As we can see, UCT-H has always a significantly better performance than UCT. Additionally, the difference between UCT and UCT-H seems to increase with $|\boldsymbol{\Omega}|$: we can see that UCT-H is around 35% better than UCT with 2 agents, but around 65% better with 10 agents.

In Figure 4 we evaluate the computational time per time step for each algorithm (as we limit the time of the MCTS by the number of simulations). As we can see, the difference in computational time is not significant between both algorithms. Hence, UCT-H uses about the same computational time as UCT but achieves a better performance.

We also evaluate the memory usage of both algorithms, in Figure 5. As we can see, both UCT and UCT-H tended to use a similar amount of memory up to 8 agents, although UCT tended to use more memory than UCT-H (up to 8 agents, the difference is only significant with 3 agents). For more than 8 agents, however, *UCT uses a significantly higher amount of memory*. In fact, we can notice that UCT-H memory usage tends to remain constant at $|\boldsymbol{\Omega}|$, while UCT tends to increase exponentially as the number of agents increases. Therefore, not only UCT-H achieves a better overall performance than UCT, but it is also more scalable in terms of memory usage as the number of agents in the system grows.

Additionally, we can also notice that UCT had a much larger variance than UCT-H in terms of memory usage, especially for a larger number of agents. Therefore, when using UCT-H one can have a better expectation of the amount of memory necessary to run the system.

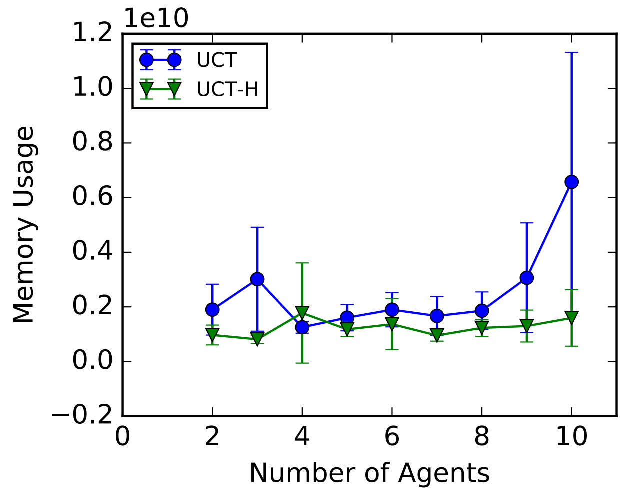
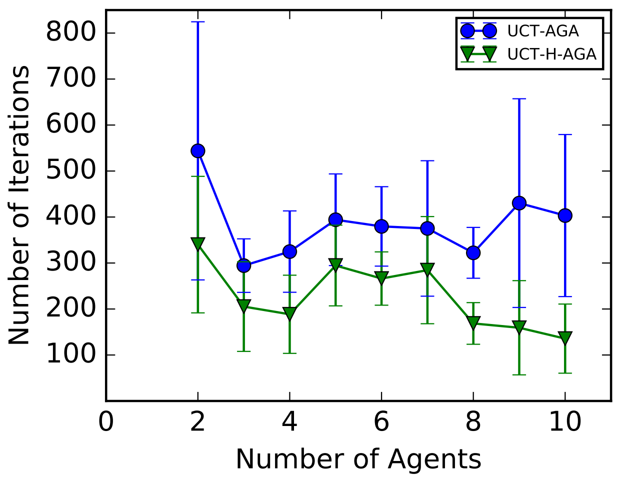


Fig. 5: Memory usage of MCTS algorithms as the number of agents increases.

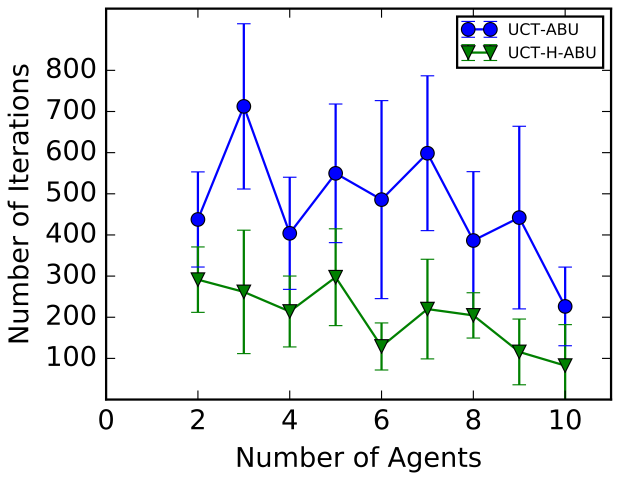
C. Results for Different Estimation Methods

For the interested reader, we also show the performance results for UCT and UCT-H, depending on the specific parameter estimation method used (Approximate Gradient Ascent or Approximate Bayesian Updating), as the results were combined in the previous section.

In Figure 6 we show the results. We can still see that UCT-H obtains a significantly better performance than UCT for both estimation methods. For AGA, we find statistical significance for all number of agents except 2, 3, 5 and 7 (due to the large variance in the UCT results, even though UCT-H *clearly has a lower average for all team sizes*). Concerning ABU, our performance is again significantly better for all number of agents.



(a) Approximate Gradient Ascent



(b) Approximate Bayesian Updating

Fig. 6: Performance of different MCTS algorithms as the number of agents increases (the lower the better), for different estimation methods.

V. Conclusion

In this paper, we presented a novel UCT Monte Carlo Tree Search algorithm for ad-hoc teamwork. Our approach introduces a more compact representation, by representing each node as a *history* instead of a *state*.

We perform several experiments in the level-based foraging domain, a problem that requires close cooperation between agents, and thus very well suited for ad-hoc teamwork evaluation. We show that our approach achieves a better performance than the current state-of-the-art, using roughly the same amount of computational time, and the difference tends to increase as the number of agents grows.

Additionally, we evaluate the memory usage of our approach and the state-of-the-art. We found that our approach tends to use a roughly constant amount of memory, while the memory usage of the state-of-the-art grows exponentially with the number of agents. Hence, our approach has a better scalability, and should better handle larger team sizes.

As an additional contribution, we provide a fully open-source version of our system to the community, making it easier for other researchers to use level-based foraging as an important benchmark problem for ad-hoc teamwork.

Acknowledgments

We would like to thank the Faculty of Science and Technology for their support.

References

[1] P. Stone, G. A. Kaminka, S. Kraus, and J. S. Rosenschein, "Ad hoc autonomous agent teams: Collaboration without pre-coordination," in *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI)*, 2010.

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

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

[4] S. V. Albrecht and S. Ramamoorthy, "Exploiting causality for selective belief filtering in dynamic bayesian networks," *Journal of Artificial Intelligence Research*, vol. 55, 2016.

[5] S. Barrett, P. Stone, and S. Kraus, "Empirical evaluation of ad hoc teamwork in the pursuit domain," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, 2011.

[6] S. Albrecht and P. Stone, "Reasoning about hypothetical agent behaviours and their parameters," in *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS'17, May 2017.

[7] A. Panella and P. Gmytrasiewicz, "Interactive POMDPs with finite-state models of other agents," *Autonomous Agents and Multi-Agent Systems*, vol. 31, no. 4, pp. 861–904, July 2017.

[8] D. Silver and J. Veness, "Monte-carlo planning in large POMDPs," in *Proceedings of the Twenty-Fourth Annual Conference on Neural Information Processing Systems*, 2010.

[9] S. V. Albrecht and S. Ramamoorthy, "A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems," The University of Edinburgh, Tech. Rep., February 2013.

[10] S. V. Albrecht and P. Stone, "Autonomous agents modelling other agents: A comprehensive survey and open problems," *Artificial Intelligence (AIJ)*, vol. 258, pp. 66–95, 2018.

[11] L. Kocsis and C. Szepesvri, "Bandit based monte-carlo planning," in *Proceedings of the 17th European Conference on Machine Learning*, 2006.

[12] L. S. Marcolino, Y. T. dos Passos, A. A. F. de Souza, A. d. S. Rodrigues, and L. Chaimowicz, "Avoiding target congestion on the navigation of robotic swarms," *Autonomous Robots*, vol. 41, no. 6, 6 2017.

[13] R. Ramaithitima, M. Whitzer, S. Bhattacharya, and V. Kumar, "Sensor coverage robot swarms using local sensing without metric information," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 3408–3415.

[14] F. R. Inacio, D. G. Macharet, and L. Chaimowicz, "United we move: Decentralized segregated robotic swarm navigation," 2016.

[15] P. Riley and M. Veloso, "Recognizing probabilistic opponent movement models," in *Robot Soccer World Cup V*, A. Birk, S. Coradeschi, and S. Tadokoro, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 453–458.

[16] S. Barrett and P. Stone, "Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork," in *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2015.

[17] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2–3, pp. 235–256, 2002.

[18] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.