To appear in *Autonomous Agents and Multiagent Systems* (AAMAS),
Taipei, Taiwan, May 2011.

Empirical Evaluation of Ad Hoc Teamwork in the Pursuit Domain

Samuel Barrett
Dept. of Computer Science
The Univ. of Texas at Austin
Austin, TX 78712 USA
sbarrett@cs.utexas.edu

Peter Stone
Dept. of Computer Science
The Univ. of Texas at Austin
Austin, TX 78712 USA
pstone@cs.utexas.edu

Sarit Kraus
Dept. of Computer Science
Bar-Ilan University
Ramat Gan, 52900 Israel
sarit@cs.biu.ac.il

ABSTRACT

The concept of creating autonomous agents capable of exhibiting *ad hoc teamwork* was recently introduced as a challenge to the AI, and specifically to the multiagent systems community. An agent capable of ad hoc teamwork is one that can effectively cooperate with multiple potential teammates on a set of collaborative tasks. Previous research has investigated theoretically optimal ad hoc teamwork strategies in restrictive settings. This paper presents the first empirical study of ad hoc teamwork in a more open, complex teamwork domain. Specifically, we evaluate a range of effective algorithms for on-line behavior generation on the part of a single ad hoc team agent that must collaborate with a range of possible teammates in the pursuit domain.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]

General Terms

Algorithms, Experimentation

Keywords

Ad Hoc Teams, Agent Cooperation: Teamwork, coalition formation, coordination, Agent Reasoning: Planning (single and multiagent), Agent Cooperation: Implicit Cooperation

1. INTRODUCTION

Autonomous agents, both of the software and robotic varieties, are becoming increasingly common and accepted as a part of day to day life. More often than not, these agents are deployed in settings in which they are aware ahead of time of what other agents they will encounter. In multiagent team settings, the teammates are usually deployed at the same time and by the same developers or users.

However, as agents become more robust and therefore more relied upon, they are likely to be deployed for longer periods of time and in less controlled teamwork settings. When that happens, these agents will need to be prepared to cooperate with many different types of teammates. For example, in a software setting, an agent may need to create travel plans for a client by interacting with other agents that it has not encountered before.

Cite as: Empirical Evaluation of Ad Hoc Teamwork in the Pursuit Domain, Samuel Barrett, Peter Stone, and Sarit Kraus, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. XXX-XXX.

In a recent AAAI challenge paper, Stone et al. defined an *ad hoc team setting* as a problem in which team coordination strategies cannot be developed a priori [15]. They presented an evaluation framework for measuring the ad hoc teamwork capabilities of an agent and summarized previous theoretical results. Although they emphasized that the ad hoc teamwork challenge is "ultimately an empirical challenge," to the best of our knowledge, there have not yet been any empirical evaluations of strategies for ad hoc teamwork.

This paper fills that gap. Specifically, using the evaluation framework from the aforementioned challenge paper, we evaluate and compare strategies for ad hoc teamwork in the popular pursuit domain from the multiagent systems literature [1]. In this domain, four predators must collaborate to capture a prey. In the usual setting, strategies for a full team of predators are evaluated together. In contrast, we study the effectiveness of an *individual* ad hoc team agent's strategy when combined with various sets of teammates.

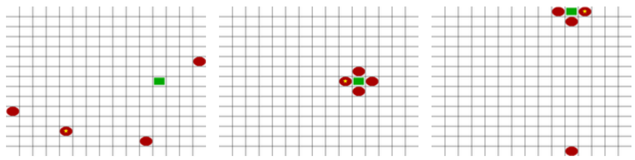
We begin with the simplest case in which the agent's teammates are homogeneous and behave deterministically, and the agent has a full model of their behavior (though it only learns of this behavior at the last minute: it still needs to determine its own behavior online). We then consider progressively more difficult scenarios in which the teammates are stochastic, heterogeneous, unknown but drawn from a distribution of types, and eventually completely unknown a priori. In so doing, we compare several different successful methods for generating teamwork behavior online. These methods range from optimal, but computationally complex, solutions to efficient, approximate sampling-based methods that incorporate Bayesian updates over the space of possible teammate behaviors.

The primary contribution of this paper is the initial empirical evaluation of ad hoc teamwork strategies. We present detailed analyses of extensive controlled empirical tests comparing generally applicable and effective algorithms for ad hoc teamwork.

The remainder of the paper is organized as follows. Section 2 describes our problem setting, including the testbed domain and evaluation framework. Section 3 introduces the space of teammates with which we test our ad hoc team agent, and Section 4 fully specifies the on-line behavior planning algorithms that we evaluate for the purposes of ad hoc teamwork. Section 5 presents the main contribution, namely detailed empirical results and analysis. Section 6 situates our contribution in the literature, and Section 7 concludes.

2. PROBLEM DESCRIPTION

The focus of this paper is an empirical evaluation of ad hoc teams. To this end, we present a well defined testing domain that requires the cooperation of a team, but still relies on each team member performing intelligently. Also, we specify a framework for evaluating and comparing the performance of ad hoc team agents.



(a) Random starting position (b) One capture position (c) Another capture position

Figure 1: Start and capture positions in the pursuit domain. The green rectangle is the prey, the red ovals are predators, and the red oval with the star is the ad hoc predator (the one under our control that is being evaluated).

2.1 Pursuit Domain

The pursuit domain was introduced by Benda et al. [1] and has been used frequently in the multiagent systems literature [18]. This problem is well suited for ad hoc team research as it requires cooperation between the agents; no agent can accomplish the task by itself regardless of its abilities. There are many variations of the pursuit domain, but they all involve a set of predators whose aim is to "capture" a prey, though the mechanics of the world and the definition of "capture" vary. A common formulation that we adopt is that the world is a toroidal grid and the predators must block all possible moves of the prey. For this work, we use a single prey and four predators, with only left, right, up, down, and no-op movements. We use a simple prey behavior that moves randomly.

Note that the world is a torus, so moving off one side of the world brings the agent back on the opposite side. This means that all four predators are required to capture the prey; it is not possible to trap the prey against the side of the board. Each agent can observe the positions of all other agents, but the agents are not capable of explicit communication. Agents start in random positions and select their actions simultaneously at each time step. Collisions are handled by ordering the agents, including the prey, randomly each time step, and performing moves in this order. If an agent's desired destination is occupied, the agent stays in its current location. Excluding collisions, all action effects are deterministic. Examples of the starting positions and capture positions can be found in Figure 1.

2.2 Ad Hoc Team Agent

For the purpose of comparing potential ad hoc team agents, we adopt the evaluation framework introduced by Stone et al. [15] and reproduced in Algorithm 1. According to this framework, the quality of an ad hoc team player depends on both the domain D and the set of possible agents A that the ad hoc agent will interact with. The algorithm compares agents a_0 and a_1 as potential ad hoc teammates of agents drawn from the set A collaborating on tasks drawn from domain D. Note that $s(B, d)$ is a scalar score resulting from the team B executing the problem d, where higher scores indicate better team performance and s_{min} is a minimum acceptable reward.

Throughout this paper, the domain D is the pursuit domain as described in Section 2.1. We consider each task $d \in D$ to be defined by the starting positions of the agents, the sequence of moves to be made by the prey, and the agent orderings for collisions. Therefore, if two different ad hoc agents perform the same actions on the same task, they will end with the same reward. The possible teammates comprising the set A are described next in Section 3.

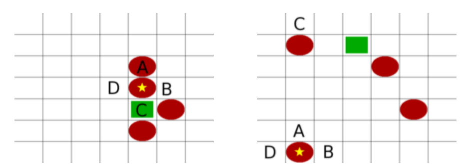
3. AGENT DESCRIPTIONS

In order to meaningfully test our proposed ad hoc teamwork algorithms, we implemented four different predator algorithms with varying and representative properties. The deterministic *greedy predator* mostly ignores its teammates' actions while the deterministic *teammate-aware predator* tries to move out of the way of its teammates, but it also assumes that they will move out of its way when needed. We expect these differences to require the ad hoc agent to adapt and reason about how its actions will interact with its teammates' actions. In addition to these two deterministic agents, we created two stochastic agents that select an action distribution at each time step. We expect it to be fairly trivial for the ad hoc agent to differentiate the deterministic agents, but harder to differentiate the stochastic agents. Finally, we tested our agent's ability to cooperate with a number of other agents for which it had no model.

Algorithm 1 Ad hoc agent evaluation

Evaluate(a_0 , a_1 , A, D):

- Initialize performance (reward) counters r_0 and r_1 for agents a_0 and a_1 respectively to $r_0 = r_1 = 0$.
- Repeat:
  - Sample a task d from D.
  - Randomly draw a subset of agents B, $|B| = 4$, from A such that $E[s(B, d)] \geq s_{min}$.
  - Randomly select one agent $b \in B$ to remove from the team to create the team B^-.
  - Increment r_0 by $s(\{a_0\} \cup B^-, d)$
  - Increment r_1 by $s(\{a_1\} \cup B^-, d)$
- If $r_0 > r_1$ then we conclude that a_0 is a better ad hoc team player than a_1 in domain D over the set of possible teammates A. Similarly, if $r_1 > r_0$ then a_1 is better.



(a) Configuration 1 (b) Configuration 2

Figure 2: World configurations that differentiate the predators' behaviors.

We will now introduce some notation to simplify the predator descriptions. Assume that a predator is at position (x, y) and is trying to move to a destination (x', y') on a world of size (w, h).

$$\Delta_x = (x' - x) \bmod w \qquad \Delta_y = (y' - y) \bmod h$$
$$\text{dim}_{\text{min}} = \text{argmin}(\Delta_x, \Delta_y) \qquad \text{dim}_{\text{max}} = \text{argmax}(\Delta_x, \Delta_y)$$
$$m_i = \underset{\text{moves}}{\text{argmin}}\, \Delta_i$$

Thus, m_i is the move that minimizes the difference to the destination for dimension i, and $\overline{m_i}$ is the move in the opposite direction. The stochastic agents use the softmax activation function, which assigns probabilities to a set of values, favoring the higher values. The temperature, τ, controls the amount of this bias, with values closer to 0 resulting in higher probabilities of the maximum value. If $v(i)$ is the value of option i, the probability of option a is

$$p(a) = \frac{\exp(v(a)/\tau)}{\sum_{i=1}^{n} \exp(v(i)/\tau)}$$

To clarify the predators' behaviors, we will show examples of their action selection on the cases shown in Figure 2, looking at the actions taken by the starred agent. The letters in the figure indicate the destination of the agent after taking one step. Note that none of the predators we created ever choose to stay still, so we do not label that action here.

3.1 Greedy Predator

The greedy predator selects the nearest unoccupied cell neighboring the prey, and tries to move towards it while avoiding immediate obstacles. It follows the following rules in order.

- If already neighboring the prey, try to move onto the prey so that if it moves, the predator will follow.
- Choose the nearest unoccupied cell neighboring the prey as the destination.
- Let $d = \dim_{\max}$. If m_d is not blocked, take it.
- Let $d = \dim_{\min}$. If m_d is not blocked, take it.
- Otherwise, move randomly.

For example, using the configurations shown in Figure 2 and taking actions as the starred agent, if the starred agent were a Greedy predator, it chooses the move taking it to cell C in configuration 1, and B in configuration 2. On average, a team of all Greedy predators captures the prey in 7.74 steps on a 5x5 world.

3.2 Teammate-aware Predator

The teammate-aware predator considers its teammates' distances from the prey when selecting its destination and uses A* path planning (an optimal heuristic search algorithm) [10] to avoid other agents, treating them as static obstacles. In contrast to the greedy predator, a teammate-aware predator that is already neighboring the prey may move towards another neighboring cell to give its spot to a farther away teammate. It is implemented as follows.

- Calculate the distance from each predator to each cell neighboring the prey.
- Order the predators based on worst shortest distance to a cell neighboring the prey.
- In order, the predators are assigned the unchosen destination that is closest to them (without communication), breaking ties by a mutually known ordering of the predators.
- If the predator is already at the destination, try to move onto the prey so that if it moves, the predator will follow.
- Otherwise, use A* path planning to select a path, treating other agents as static obstacles.

For the configurations shown in Figure 2, a Teammate-aware predator in the position of the starred predator chooses the move taking it to cell D in configuration 1, and C in configuration 2 (note that since the world is a torus, this is a single move). A team of Teammate-aware predators captures the prey in 7.41 steps on a 5x5 world.

3.3 Greedy Probabilistic Predator

The greedy probabilistic predator moves towards the nearest cell neighboring the prey, but does not always take a direct path there. The predator favors minimizing $\dim_{\max}$ and prefers m_{dim} over $\overline{m_{\text{dim}}}$.

- If already neighboring the prey, try to move onto the prey so that if it moves, the predator will follow.
- Choose the nearest unoccupied cell neighboring the prey as the destination.
- Given a destination, choose a dimension, d, to minimize using the softmax function with temperature 0.5 using the distance as v.
- Choose either m_d or $\overline{m_d}$ using the softmax function with temperature -0.5, using the distance after the move as v, but penalizing moves that are currently blocked.

On configuration 1 from Figure 2, the predator is deterministic, choosing the action taking it to position C. On configuration 2, it selects a distribution of actions, specifically the moves taking it to cells A, B, C, and D with probabilities 0.000, 0.879, 0.119, and 0.002. On a 5x5 world, a team of Greedy Probabilistic predators captures the prey in 12.88 steps.

3.4 Probabilistic Destinations Predator

The probabilistic destinations predator attempts to tighten a circle around the prey. It favors destinations that are both nearer to the prey and to itself, but may choose farther destinations to prevent getting stuck on other predators and dealing with a moving prey.

- If already neighboring the prey, try to move onto the prey so that if it moves, the predator will follow.
- Select a desired distance from the prey using the softmax function with temperature -1 using the distance as v.
- Select a destination at the chosen distance using the softmax function with temperature -1 weighted by the distance of the destination to the predator's current position.
- Let $d = \dim_{\max}$, and select m_d.
- If the destination or the next position is occupied, repeat.

For the configurations in Figure 2, a Probabilistic Destinations predator would select the move ending in C in configuration 1. On configuration 2, it would select actions taking it to cells A, B, C, and D with probabilities 0.007, 0.596, 0.388, and 0.009. A team of predators following the Probabilistic Destinations behavior capture in 9.19 steps on a 5x5 world.

3.5 Student-created Predator

In some situations, an ad hoc team agent may be aware of the space of possible behaviors from which its teammates are drawn. However in other cases, it may not know anything about them. To fairly test the latter scenario, we incorporated into our testing a number of agents that we did not create. Specifically, we used a set of agents created by undergraduate and graduate Computer Science students for an assignment in a workshop on agents. These students were initially provided with a skeleton agent and then iteratively improved their agent.

As one might expect from a class, there was a wide variety in the quality of the agents that were submitted. In order to ensure a base level of competence, we only considered agents that were able to capture the prey within 15 steps on average on a 5x5 world (i.e. $s_{min} = 15$ in Algorithm 1). Out of the 41 agents submitted, 12 of the agents met this threshold.

Due to space constraints, we cannot fully describe all of the student agents used, but here we highlight some interesting cases. One student focused on avoiding collisions at cells neighboring the prey. Therefore, this student assigned the predators an arbitrary ordering and had each predator only consider blocking a specific direction chosen based on the assigned ordering. This strategy works if all the predators have mutually complementary assignments, but can create inefficiencies when the predators start far from their desired blocking directions.

The highest performing agent from the class performed better than any of our agents on the 5x5 world, capturing in only 4.05 steps on average. This agent considers all the cells neighboring the prey, and then considers all possible assignments of these destinations to the predators. For each possible assignment, it calculates the distance from each predator to its destination. Then, it chooses the assignment that minimizes the sum of these distances. Finally, each predator chooses the move that minimizes its distance to the selected destination. This agent performs quite well, although it does not seek to avoid collisions among the predators.

4. PLANNER DESCRIPTIONS

As is clear from the evaluation framework described in Section 2.2, the main thing that distinguishes one ad hoc team agent from another is its strategy for planning and selecting actions as a function of the current task d and current set of teammates B^-. In this section we describe the ad hoc teamwork planning algorithms that we test in this paper.

One might think that the most appropriate thing for an ad hoc team agent to do is to "fit into" the team by following the same behavior as its teammates. However, in some cases, it is possible for the ad hoc team agent to improve on this or even solve for the optimal behavior, if the agent has a full model of its teammates' behaviors. Even without such a model, the ad hoc agent can approximate the optimal behavior. Indeed, in our tests, we found situations in which model-based planning even with an imperfect model outperforms the ad hoc agent mimicking its teammates' behaviors.

In some cases, the ad hoc team agent may "recognize" its teammates and be able to use its stored knowledge of their behaviors to plan its own actions. This situation is still an ad hoc team setting because the agent must generate its strategy on-line: it does not know in advance whom its teammates will be.

4.1 Value Iteration

When there is a fully known model of the environment and each agent, the ad hoc team agent can treat the domain as a Markov Decision Process (MDP) and can solve for the optimal behavior using Value Iteration (VI) [19]. Value iteration relies on dynamic programming to solve the optimal state-action values for all state-action pairs. VI initializes the state-action values arbitrarily, and then improves these estimates using an update version of the Bellman optimality equation:

$$Q(s,a) = \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a'} Q(s',a') \right]$$

where $Q(s,a)$ is the long term expected reward of taking action a from state s, $P^a_{ss'}$ is the probability of transitioning from state s to state s' after taking action a, $R^a_{ss'}$ is the corresponding reward, and γ is the discount factor. These updates are repeated iteratively until convergence. The state-action values calculated by VI are guaranteed to be correct.

However, the problem space is exponential in the size of the world, with a power proportional to the number of agents. The use of symmetries can reduce the size of this space, but in our tests, VI on a 5x5 world took approximately 12 hours on the [[removed for blind review]] computing cluster. Due to the exponential blowup of the state space, there are 100^5 states in a 10x10 world (without using symmetries) as opposed to 25^5 in a 5x5 world, so running VI on larger worlds was unfeasible.

4.2 Monte Carlo Tree Search

When the state space is large and only small sections of it are relevant to the agent, it can be advantageous to use a sample-based approach to approximating the values of actions, such as Monte Carlo Tree Search (MCTS). Specifically, we use the MCTS algorithm called Upper Confidence bounds for Trees (UCT) as a starting point for creating our algorithm [13].

MCTS does not require the complete model of the environment; it only needs a way of sampling the effects of selected actions. Furthermore, rather than treating all of the state-actions as equally likely, UCT focuses on calculating only the values for relevant state-actions. UCT does so by performing a number of playouts at each step, starting at the current state and sampling actions and the environment until the end of the episode. It then uses these playouts to estimate the values of the sampled state-action pairs. Also, it maintains a count of its visits to various state actions, and estimates the upper confidence bound of the values to balance exploration and exploitation. UCT has been shown to be effective in games with a high branching factor, such as Go [7], so it should be able to handle the branching factor caused by the number of agents.

We modify UCT to use eligibility traces and remove the depth index to help speed learning in the pursuit domain. The pseudocode of the algorithm can be seen in Algorithm 2, with s being the current state. Similar modifications were made by Silver et al. with good success in Go [14].

Algorithm 2 The Monte Carlo Tree Search algorithm used by our ad hoc agent.

```
function Select(s):
  for i = 1 to NumPlayouts do
    Search(s)
  return a = argmax_a Q(s, a)
function Search(s):
  a = bestAction(s)
  while s is not terminal do
    (s', r) = simulateAction(s, a)
    a' = bestAction(s')
    e(s, a) = 1
    δ = r + γQ(s', a') − Q(s, a)
    for all s*, a* do
      Q(s*, a*) = Q(s*, a*) + e(s*, a*) * δ/visits(s*, a*)
      e(s*, a*) = λe(s*, a*)
    s = s'; a = a';
```

4.3 Planning for uncertainty

Both of the planners described above assume that some kind of a model of the environment is known. However, it is likely that the ad hoc team agent has some uncertainty about the behavior of its teammates. One possibility is that the agent has a prior probability distribution over a set of possible behaviors, representing its belief of the likelihood of its teammates following this behavior. As the ad hoc agent gets more information about the agents from their actions, it should update its belief using Bayes theorem:

$$P(\text{model}|\text{actions}) = \frac{P(\text{actions}|\text{model}) * P(\text{model})}{P(\text{actions})}$$

If the agent has a complete model of each of the teammate types and a prior belief $P(\text{model})$, it can calculate $P(\text{actions}|\text{model})$. Finally, $P(\text{actions})$ can just be treated as a normalizing factor, to make the probabilities of the various models sum to 1.

Using this method, the MCTS-based agent can keep track of the probabilities of the different behaviors, and sample the environment accordingly. For Value Iteration, the exact solution requires recalculating the correct Q-values for each new set of probabilities, but this was not feasible for our tests. Therefore, we approximate the VI solution using a linear combination of the Q-values learned for each set of teammates:

$$Q(s,a) = p_1 * Q_1(s,a) + p_2 * Q_2(s,a) + \ldots + p_n * Q_n(s,a)$$

where $p_i = P(\text{model}_i|\text{actions})$ and $Q_i(s,a)$ are the Q-values calculated for model_i. Note that this is not guaranteed to be correct, but it works well in practice and gives us a baseline of how well an ad hoc agent can do. However, this approach still requires the ad hoc agent to know that the teammates are using one of several known behaviors.

4.4 Learning to model teammates

Sometimes the ad hoc agent will encounter teammates that do not come from its set of known behaviors, so it may need to learn a model of these teammates. To learn a model of the teammates, we used an implementation of the C4.5 algorithm for generating decision trees, provided in WEKA [9]. The decision tree was given the absolute x and y coordinates of each agent in the world, and attempted to predict the action that the agent would take in that world state. However, the ad hoc agent does not directly observe its teammates' actions; it only observes the results of the actions. For the training data, these actions were approximated by observing the agents' movements. If the agent did not move, it may have chosen not to move or it collided, so the decision tree is given both possibilities, weighted by the probability that each occurred.

In this case, the ad hoc agent starts with no information about these agents and has only a single episode to learn about its teammates, so it must adapt over a short period of time. In this paper, we assume that all the teammates are running the same algorithm, so we use a single decision tree to learn about the behaviors of all the teammates. Thus the decision tree is given three additional observations of the agents' actions at each time step. The ad hoc agent continues to use its set of known models to plan as it builds the model, tracking the probabilities of these models and the learned model using the Bayesian updates as in Section 4.3. The idea is that the agent will use the known models for its initial planning until it encounters actions that these models would not predict, at which time it will increase its reliance on the learned model.

Due to the extreme paucity of data available to the learner, it would be difficult to learn a useful model over the course of a single episode. Surprisingly, our empirical results in Section 5.6 indicate that this form of model learning is beneficial.

5. RESULTS

In this section, we evaluate and thoroughly analyze the planning algorithms in Section 4 in a series of increasingly open-ended ad hoc team scenarios. These results constitute the main contribution of this paper.

In all of our experiments, we use the evaluation framework discussed in Section 2.2. For each test, D is the set of all valid starting positions of the agents. For each episode, the starting position is randomly selected, but these positions are held constant across evaluations of the different agents. Similarly, other random factors such as the prey's action selection and collision tie-breakers are fixed with the starting positions. Therefore, if two ad hoc agents execute the same sequence of actions on the same problem, their results will be exactly the same. This approach controls for randomness in the environment and makes differences in the ad hoc agent's behavior the only cause for differences in the results.

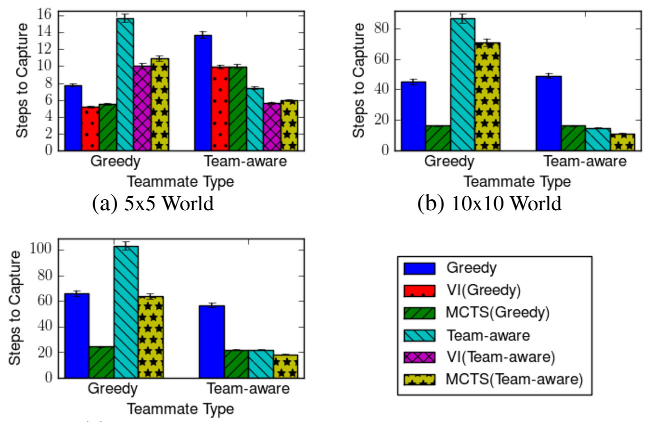
For the first set of experiments, we assume that the behavior of the teammates is known at the start of the episode (though not before), and that this behavior is deterministic (Section 5.1). Even though the ad hoc team agent has a full model of its teammates, this scenario is still an ad hoc teamwork setting because there is no opportunity for the team to coordinate prior to starting the task: the agent must determine its strategy online. In the second set of experiments, we relax the constraint that the teammates' behavior is deterministic and investigate stochastic agents (Section 5.2). Next, we explore the performance of the ad hoc agents when the teammates' behavior is not exactly known, but is instead known to be drawn from a set of known behaviors (Section 5.3). Then, we mix the teammate types to see how the ad hoc agent could cope with unplanned teams and select the correct model from a large set of possible models (Section 5.4). In Section 5.5, we test against a set of agents that we did not create and that do not fit the models given to the ad hoc agent. Finally, we enable the agent to learn models on the fly to deal with these agents (Section 5.6).

For these tests, we compare against value iteration on the 5x5 worlds as it defines the optimal policy for the ad hoc agent, but we were unable to practically run VI on larger worlds as discussed in Section 4. In the graphs, we use VI(Greedy) and MCTS(Greedy) to indicate that the planning was performed treating the teammates as Greedy Predators, and we do likewise for VI(Teammate-aware), etc. Note that all of the predators on a team follow the same behavior for Sections 5.1–5.3, but not necessarily for Sections 5.4–5.6.

In all cases, a lower number on the graphs is better, as it means that it took fewer steps to capture the prey. All results are averaged for 1,000 runs, but note that the ad hoc agent does not keep any knowledge between runs. All statistical tests are performed as paired Student-T tests to control for the randomness caused by the starting positions of the tests. The error bars shown are given as $\frac{\sigma}{n}$, where σ is the standard deviation of the lengths of the runs and n is the number of runs (1,000).

5.1 Deterministic known teammates

For our initial tests, we consider the simple case in which the ad hoc agent has an exact model of its teammates and its teammates are deterministic (either Greedy or Teammate-aware). These tests are designed to determine whether the ad hoc agent can do better than just mimicking its teammates, and how effective MCTS is at approximating the optimal behavior found by VI.



(a) 5x5 World (b) 10x10 World (c) 20x20 World

Figure 3: Results with known deterministic teammates.

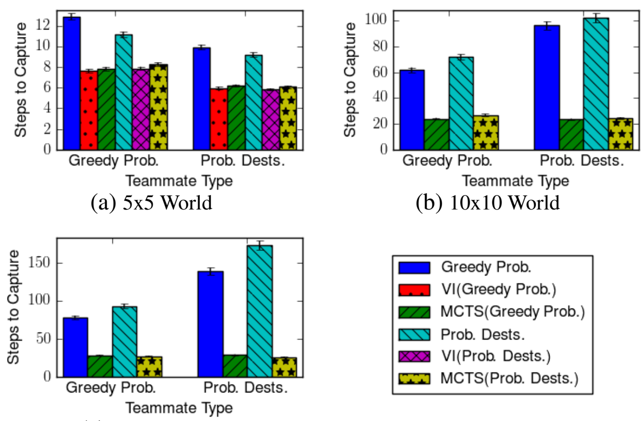
The results in Figure 3 show that the ad hoc agent can do much better than just copying the behavior of its teammates. Following the optimal behavior found by VI achieves capture in 5.19 and 5.92 steps respectively when cooperating with Greedy and Teammate-aware teammates as opposed to 7.74 and 7.41 steps when mimicking their behavior. These differences are statistically significant with $p < 0.001$. The improvements of planning over mimicking the teammates increase as the worlds get larger, although we use MCTS to approximate the optimal behavior for these worlds.

We verify that this use of MCTS is not much of a compromise, since it performs nearly as well as VI despite using much less computation time. In the 5x5 world, it takes 5.50 and 5.92 steps to capture with Greedy and Teammate-aware agents, as opposed to VI's 5.19 and 5.66 steps. The difference with the Greedy teammates is statistically significant ($p = 0.0244$), but the difference with the Teammate-aware teammates is not significant. The difference in performance could be lowered by using more playouts in the MCTS at the cost of more computation time. Given the close approximation to optimal that MCTS provides, the most important difference between the methods is the time it takes to plan. On the 5x5 world, MCTS episodes take on average less than a minute compared to VI's 12 hour computation (although VI only needs to run once, rather than for each episode). Furthermore, MCTS is an anytime algorithm, so it can be used to handle variable time constraints and can modify its plan online as the models change.

The results also show that having an incorrect model of your teammates can be costly. Even simply playing the Teammate-aware behavior when your teammates play Greedy hurts the team by a large amount. Intuitively, this seems odd as the ad hoc agent playing smarter should help the team, but the Teammate-aware behavior relies on its teammates also moving out of its way, which will not happen with Greedy teammates. Planning as if your teammates are Greedy, when in fact they are Teammate-aware is costly when mimicking their behavior or when planning using VI or MCTS. On the 5x5 world, using the wrong model results in taking 13.75, 9.92, and 9.97 steps to capture for the mimic, VI, and MCTS cases, respectively, when the teammates are in fact Teammate-aware as opposed to 7.41, 5.66, and 5.92 steps when using the correct models.

5.2 Known stochastic teammates

We now consider the case where the ad hoc agent once again has an exact model of its teammates, but this time its teammates' behavior is stochastic (either Greedy Probabilistic or Probabilistic Destinations). The goal was to test whether the ad hoc agent could plan for agents choosing from several possible actions at any time step. VI uses the entire probability distribution of possible outcomes to update its values, while MCTS samples from this distribution to approximate the values.



(a) 5x5 World (b) 10x10 World (c) 20x20 World

Figure 4: Results with known stochastic teammates.

Figure 4 shows that the MCTS-based and VI-based agents are capable of planning with this uncertainty, and still significantly outperform mimicking the behavior of its teammates. Similar to the deterministic results, MCTS performs nearly as well as VI taking 7.84 and 6.07 steps versus 7.63 and 5.82 steps with Greedy Probabilistic and Probabilistic Destinations teammates respectively on the 5x5 world. These differences are significant, but MCTS still does a good job of approximating the optimal behavior.

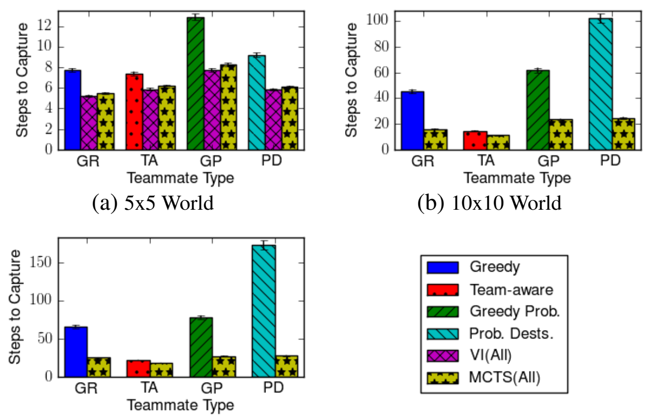
On the larger worlds, the performance of MCTS is much better than copying its teammates. For example, on the 20x20 world, the MCTS-based agent takes 24.00 steps to capture when cooperating with Greedy Probabilistic teammates compared to 78.48 steps when mimicking the teammates' behavior. Similarly, the MCTS-based agent takes 24.39 steps rather than 173.46 steps when paired with Probabilistic Destinations teammates.

Unlike the deterministic case, using an incorrect model for the teammates is not a large penalty with these agents. We believe that this is due to the overlap in the possible actions taken, and that the plans must be fairly robust to unexpected actions due to the stochasticity of the teammates.

5.3 Unknown stochastic teammates

Expanding the problem once again, all four predators used in the previous tests were used for this test, i.e. the Greedy, Teammate-aware, Greedy Probabilistic, and Probabilistic Destinations behaviors were used. Furthermore, the ad hoc agent did not know which of the four types of behavior its teammates were using. This setting gets us closer to the general ad hoc teamwork scenario, because it shows how well an ad hoc agent can do if it only knows that its teammates are drawn from a larger set A of possible teammates.

If it has a set of possible models for its teammates, ideally the ad hoc agent should be able to determine which model is correct and plan with that model appropriately. The VI and MCTS agents use the algorithm described in Section 4.3 to calculate the probabilities of each model. For both the MCTS and VI based ad hoc agents, we used a uniform prior over the teammate types, but assume that its teammates are homogeneous; i.e. there were no teams with some agents following the Greedy behavior and others following the Teammate-aware behavior. It is fairly trivial to differentiate the deterministic agents because as soon as they take one action that does not match the deterministic behavior, that incorrect model can be removed. However, the stochastic teammates are more difficult to differentiate, as there is significant overlap in the actions that are possible for them to take.



(a) 5x5 World (b) 10x10 World (c) 20x20 World

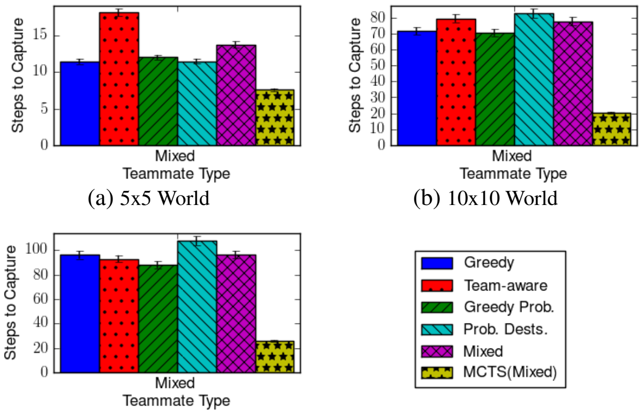
Figure 5: Results with unknown stochastic teammates. MCTS(All) means that the MCTS-based agent planned considering all homogeneous teams of the known predator models according to the current probabilities of the models.

The results in Figure 5 show that both the VI and MCTS agents perform well despite this uncertainty and determine which model its teammates are following when given the set of possible models. These results are not quite as good as if the agent had the correct models to start with, but still perform quite well. For example, on the 20x20 world, if the ad hoc agent knew its teammates were using the Probabilistic Destinations behavior, it took 26.14 steps to capture, while if it needed to select the correct model, it took 27.83 steps. On the other hand, if it mimicked its teammates, the team would have taken 173.46 steps to capture the prey.

5.4 Mixed stochastic teammates

To this point, all of the teammates have used the same behavior, a fact which was known to the ad hoc team agent. In this section, we remove that restriction, thus significantly increasing the size of the possible set of agents A, specifically from 4 to $4^3 = 64$ possible teams. Doing so again moves us towards the general ad hoc team problem, where teammates may be following a variety of behaviors and may not be coordinating with one another.

Note that the teammate types were fixed for each problem, so this variance does not affect the different ad hoc agents' evaluation. As shown in Figure 6, following any of the fixed predator types achieved fairly poor results. However, the MCTS-based agent with the knowledge that any mix of the teammates was possible performs quite well. It learns which behaviors its teammates are likely to be following and adapts appropriately. For example, on the 20x20 world, the MCTS agent takes 20.19 steps to capture rather than 96.32 if it randomly chooses a model to mimic (labeled "Mixed" in the graphs). We were unable to run VI on this case, as it would have had to learn the optimal behavior for each combination of agents ($4^3 = 64$ possible teammate combinations).



(a) 5x5 World

(b) 10x10 World

(c) 20x20 World

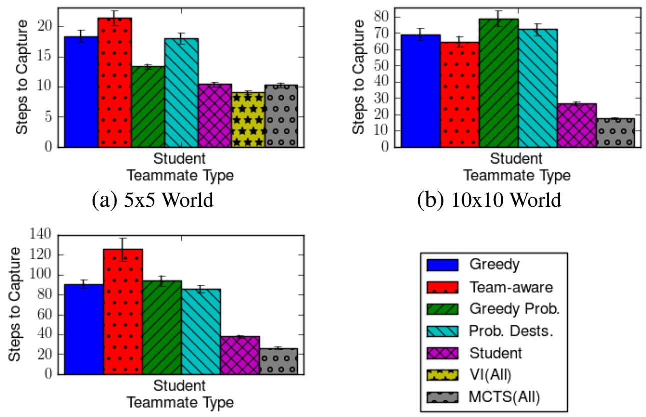
Figure 6: Results with mixed teams of stochastic teammates. The Mixed results are from the ad hoc agent randomly choosing a known predator model to mimic. MCTS(Mixed) refers to a MCTS-based agent that plans considering all heterogeneous teams of the known predator models, sampled according to the current probabilities of the models.

5.5 Unmodeled teammates

To this point, the ad hoc team agent has always had the benefit of a full model of all the teammates in A, even when it has not known a priori which types its teammates were. We now consider the case where there are agents in A for which the ad hoc team agent does not have a prior behavior model. Instead, we give the agent the same four models from before, and see how well it handles agents not following those models. To make sure we have not biased the creation of these agents, and that they truly are unknown, we used the student agents described in Section 3.5. Note that all the agents on each team used here are produced by the same student: we did not mix and match agents from different students. However, on some of the students' teams, not all of the agents use the same behavior. As before, the ad hoc agent does not store information between trials, so any learning happens during a single episode.

In this section, the ad hoc agent maintains the probabilities of the four known models as before and samples from this distribution. It does not actively consider the possibility that the teammates are unknown. Also, it assumes that all its teammates are using the same model, so it does not consider heterogeneous teams of the known models. Note that it is possible for the probability of all models to drop to 0 after a move if no known model would select that move. In this case, the agent just maintains the previous probabilities.

The results in Figure 7 show that the ad hoc agents do quite well despite the incorrect models. For example, on the 20x20 world, the MCTS agent captures in 26.47 steps rather than in 37.83 steps if it followed the student's behavior for the fourth agent. This is surprising because one would assume that planning using an incorrect model would perform worse than playing the behavior of the student's agent that the ad hoc agent replaced. Also, if the ad hoc agent follows any single one of its known models, it performs much worse than this baseline. So the ability to adapt and select the best known model at that time helps the ad hoc agent. This experiment shows that it is possible for an agent to cooperate with unknown teammates by using a set of known, representative models.



(a) 5x5 World

(b) 10x10 World

(c) 20x20 World

Figure 7: Results with student teams.

5.6 Learning to model teammates

In Section 5.5, the ad hoc agent tried to deal with the unknown agents as if they were one of the known models. Although this works fairly well in practice, the other agents may differ significantly from the known models, so it is desirable for the ad hoc agent to learn to model these agents as in Section 4.4. However, the ad hoc agent is given a very short time to learn, so we still use the set of known models and add an extra model that will be learned on the fly.

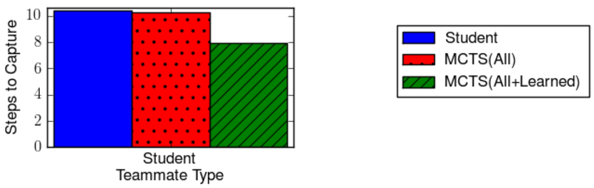
The results in Figure 8 show that learning a model of the teammates can improve performance over pretending that the teammates are following a known algorithm. Specifically, the MCTS-based agent using the learned model captured the prey in an average of 7.98 steps, as opposed to 10.26 when the agent only considered the known models. Furthermore, this is also an improvement over using the student's fourth agent, which captured in 10.40 steps on average.

This positive result is surprising due to the small number of training examples given to the agent. The episodes only lasted about nine steps on average, so the decision tree was being trained on only 27 training examples by the end of an episode on average. However, the learned model does not need to represent the entire action model of the teammates: only the states which occur. The visited number of states is likely to be small, and teammates are likely to act similarly in the visited states. Therefore, this model is much simpler to learn than a complete model. Also, we believe that a main advantage of learning the model was to prevent situations in which the agents became stuck due to collisions and incorrect predictions of the ad hoc agent.

The known models also provide a good starting point for the ad hoc agent, and it may not need to rely on the learned model too much. In our tests, for the final step of each episode, the ad hoc agent put 0.67 weight on the learned model on average, and that model was correct only 0.26 of the time. So the agent relied on the model, despite its inaccuracies. We theorize that the good performance of the system is due to the fact that it increases the options that the ad hoc agent considers, preventing it from being restricted to the actions of the known models. By itself, it is unlikely that this model would be sufficient for the agent, but when the agent uses both the learned and known models, it performs quite well.

6. RELATED WORK

The ad hoc team formulation and evaluation framework was proposed by Stone et al. [15], and there have been a few theoretical analyses of specific applications of ad hoc teams [16, 17]. Other



(a) 5x5 World

Figure 8: Results of learning models with student teams.

work in this area (though prior to the introduction of the ad hoc teamwork challenge) includes Brafman and Tennenholtz's work in which one agent teaches another while engaging in a repeated joint activity [2]. Knudson and Tumer have also investigated ad hoc teams, but in a significantly different framework [12]. Unlike our work, all of their agents adapt, and each agent is given a clear metric of its effect on the team's performance in the form of a difference objective. Furthermore, they learn over 2,000 episodes rather than our single episode. On the other hand, most prior work on coordinating teams of agents relies on explicit protocols for coordinating such as SharedPlans [8], STEAM [20], and GPGP [6]. Our work does not require these shared protocols, and does not even require the teammates to know of the ad hoc agent's existence.

The ad hoc team framework is similar to the existing opponent modeling problem. The ad hoc agent needs to model and understand its teammates, just from observing their actions, similar to opponent modeling. However, the ad hoc agent does not need to assume the worst case scenario; its teammates are not rational adversaries. In the area of opponent modeling, Conitzer and Sandholm created AWESOME, an algorithm that achieves convergence and rationality in repeated games [5]. Furthermore, Chakraborty and Stone have developed an algorithm for repeated games that handles arbitrary opponents safely and exploits memory bounded opponents [4]. This work makes weak assumptions about the adversaries, but it requires long learning times and assumes that all agents can calculate the same Nash equilibrium. Our work makes stronger assumptions about our teammates, but learns faster and makes no requirements about calculating Nash equilibria.

The pursuit domain is a well studied problem in multiagent research [18], but most research has focused on developing a coordinated team. However, some work has been done on learning to adapt to teammates. Chakraborty and Sen focus on having teammates teach novice predators, but they assume that the novices are trying to learn and share a known training protocol [3]. On the other hand, we assume that there is no shared protocol for training agents, and that there is only a single episode in which to adapt. Other work in the pursuit domain includes MAPS [21], which considers partially observable environments and more sophisticated prey behaviors, but require shared coordination algorithms. Alternatively, some approaches consider partial observability in continuous worlds [11]. However, these approaches focus on creating an entire team to solve the pursuit problem, rather than considering the case where some teammates are already following fixed behaviors.

7. CONCLUSIONS AND FUTURE WORK

This work presents the first empirical investigation of ad hoc teams, and establishes the pursuit domain as a useful domain for testing ad hoc teams. We show that an ad hoc team agent can do better than mimicking its teammates, and that efficient planning is possible using MCTS. Additionally, the ad hoc agent can differentiate its teammates on the fly when given a set of known starting models. We show that even if these models are incorrect or incomplete, as long as they are representative, they can be used to provide good performance. Finally, we show that it is possible to quickly learn models for previously unseen teammates, using known models until an accurate model is learned.

As the initial empirical investigation of ad hoc teams, this paper opens up several possible avenues for future research. In this work, we only considered teammates following fixed behaviors, and we assume that there is no explicit communication between teammates. Our ongoing research agenda includes extending to teammates that themselves learn, as well as considering the effects of a capability for partial communication among the agents. For example, the agents may then be able to communicate their desired destinations. Finally, all of the results in this paper were reported in the pursuit domain. Testing whether the same algorithms exhibit the same properties in a variety of other domains is also an important direction for future empirical research.

Acknowledgments

This work has taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by grants from the National Science Foundation (IIS-0917122), ONR (N00014-09-1-0658), and the Federal Highway Administration (DTFH61-07-H-00030). Samuel Barrett is supported by a NDSEG fellowship. Sarit Kraus as also affiliated with UMIACS and her research is supported by NSF grant 0705587 and ISF Grant #1685.

8. REFERENCES

[1] M. Benda, V. Jagannathan, and R. Dodhiawala. On optimal cooperation of knowledge sources - an empirical investigation. Technical Report BCS–G2010–28, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, Washington, July 1986.

[2] R. I. Brafman and M. Tennenholtz. On partially controlled multi-agent systems. *JAIR*, 4:477–507, 1996.

[3] D. Chakraborty and S. Sen. Teaching new teammates. In *AAMAS '06*, pages 691–693, 2006.

[4] D. Chakraborty and P. Stone. Convergence, targeted optimality and safety in multiagent learning. In *ICML '10*, June 2010.

[5] V. Conitzer and T. Sandholm. AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Mach. Learn.*, 67, May 2007.

[6] K. S. Decker and V. R. Lesser. Designing a family of coordination algorithms. In *ICMAS '95*, pages 73–80, June 1995.

[7] S. Gelly and Y. Wang. Exploration exploitation in Go: UCT for Monte-Carlo Go. In *NIPS-2006*, December 2006.

[8] B. Grosz and S. Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86:269–368, 1996.

[9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[10] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100 –107, July 1968.

[11] Y. Ishiwaka, T. Sato, and Y. Kakazu. An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning. *Robotics and Autonomous Systems*, 43(4):245 – 256, 2003.

[12] M. Knudson and K. Tumer. Robot coordination with ad-hoc team formation. In *AAMAS '10*, pages 1441–1442, 2010.

[13] L. Kocsis and C. Szepesvari. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer Berlin / Heidelberg, 2006.

[14] D. Silver, R. S. Sutton, and M. Müller. Sample-based learning and search with permanent and transient memories. In *ICML '08*, 2008.

[15] P. Stone, G. A. Kaminka, S. Kraus, and J. S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *AAAI '10*, July 2010.

[16] P. Stone, G. A. Kaminka, and J. S. Rosenschein. Leading a best-response teammate in an ad hoc team. In *Agent-Mediated Electronic Commerce: Designing Trading Strategies and Mechanisms for Electronic Markets*. November 2010.

[17] P. Stone and S. Kraus. To teach or not to teach? Decision making under uncertainty in ad hoc teams. In *AAMAS '10*, May 2010.

[18] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, July 2000.

[19] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.

[20] M. Tambe. Towards flexible teamwork. *JAIR*, 7:81–124, 1997.

[21] C. Undeger and F. Polat. Multi-agent real-time pursuit. *AAMAS '10*, 21:69–107, July 2010.