To appear in *Autonomous Agents and Multiagent Systems - Adaptive Learning Agents Workshop* (AAMAS), Valencia, Spain, June 2012.

Learning Teammate Models for Ad Hoc Teamwork

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

Avi Rosenfeld
Jerusalem College of Technology
Jerusalem, 91160 Israel
rosenfa@jct.ac.il

ABSTRACT

Robust autonomous agents should be able to cooperate with new teammates effectively by employing *ad hoc teamwork*. Reasoning about ad hoc teamwork allows agents to perform joint tasks while cooperating with a variety of teammates. As the teammates may not share a communication or coordination algorithm, the ad hoc team agent adapts to its teammates just by observing them. Whereas most past work on ad hoc teamwork considers the case where the ad hoc team agent has a prior model of its teammate, this paper is the first to introduce an agent that learns models of its teammates autonomously. In addition, this paper presents a new transfer learning algorithm that can be used when the ad hoc agent only has limited observations about potential teammates.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligience**]: Distributed Artificial Intelligence—*Multiagent Systems*

General Terms

Algorithms, Experimentation

Keywords

Ad Hoc Teams, Multiagent Systems, Teamwork

1. INTRODUCTION

For autonomous agents to perform effectively in society, they should be able cooperate with other agents. One way for this to happen is for all autonomous agents to share a communication protocol or a coordination algorithm. However, agents may be developed by many sources, making it difficult to ensure that all the agents share the same information. In addition, if agents stay deployed for a long time, it is likely that new agents will come along with new and different protocols. Therefore, it is important that agents be capable of adapting to previously unseen teammates to cooperate in accomplishing their tasks.

For example, after a disaster, robots developed in many different labs and companies may be deployed to search the area and rescue victims. As these robots come from multiple sources, they may not share a coordination protocol. If there was no need for immediate action, the developers could program methods for communication or split up the task into parts by hand. If instead some of the robots are designed to cooperate with ad hoc teams, they will be able to quickly adapt to their teammates and swiftly map the area and rescue victims.

Stone et al. [17] introduced the *ad hoc team setting* as a problem in which team coordination strategies cannot be specified a priori. They presented an algorithm for evaluating ad hoc team agents based on their ability to cooperate with a set of teammates to accomplish a set of possible tasks. They argue that while previous research has focused on theoretical results, the ad hoc teamwork problem is "ultimately an empirical challenge."

This paper addresses this challenge by introducing an ad hoc agent that explicitly builds models of its teammates and plans its behavior using a sample-based method. Then, the paper evaluates the ad hoc agent's ability to cooperate with a wide variety of unknown teammates, showing that ad hoc agents can learn models for their teammates and evaluate which models are helpful, despite their current teammates differing qualitatively from the observed ones. Finally, this paper introduces a new method for transfer learning that considers data coming from multiple sources and then evaluates its effectiveness for ad hoc teams when the ad hoc agent has a small number of observations of the current agent.

2. PROBLEM DESCRIPTION

Consider the case where an ad hoc agent is trying to cooperate with a set of teammates it has never seen before. If the ad hoc agent and its teammates share a communication protocol or a method for coordination, it can directly cooperate with them. However, if its communication is limited and there is no pre-arranged coordination method, the ad hoc agent must observe its teammates and try to adapt to their behaviors. If the ad hoc agent has previously observed agents similar to its current teammates, it should try to leverage its prior experiences in order to cooperate with them more effectively.

To clarify the problem, consider the concrete example of a team of robots trying to surround and disable an intruder. The original team of robots was designed to coordinate their actions to capture the intruder quickly and reliably, but one of the original robots has since broken and the others have been damaged from constant wear and tear. Therefore, a new robot is deployed to join the team, but it does not know the specific make and model of its teammates. Fortunately, the new robot has observed similar teammates in other situations, so it has some understanding of how to cooperate with its new teammates. The problem it faces is to quickly identify which previous observations are applicable to these teammates and adapt to any differences.

2.1 Pursuit Domain

The pursuit domain is a popular problem in multiagent systems literature because it requires all of the teammates to cooperate to capture the prey [18]. The details vary, but the pursuit domain revolves around a set of agents called predators trying to capture an agent called the prey in minimal time.

In the version of the pursuit domain used in this paper, the world is a rectangular, toroidal grid, where moving off one side of the grid brings the agent back on the opposite side. Four predators attempt to capture the randomly moving prey by surrounding it on all sides in as few time steps as possible. At each time step, each agent can select to move in any of the four cardinal directions or to remain in its current position. All agents pick their actions simultaneously, and collisions are handled using priorities that are randomized at each time step. In addition, each agent is able to observe the positions of all other agents. A view of the domain is shown in Figure 1.

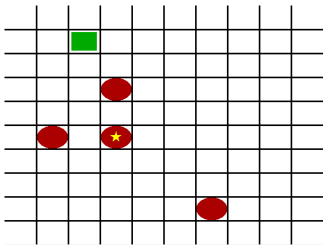

Figure 1: A view of the pursuit domain, where the rectangle is the prey, the ovals are predators, and the oval with the star is the ad hoc predator being evaluated.

2.2 Evaluation

Ad hoc team agents must be able to cooperate with a variety of previously unseen teammates to accomplish a task. To this end, we adopt the evaluation framework proposed by Stone et al. [17]. This evaluation specifies that the performance of an ad hoc agent explicitly depends on the possible teammates it will encounter as well as the potential tasks it may face. The evaluation proceeds by sampling a task and a team from the set of possible tasks and teammates. Then, it removes one of the agents from the team selected at random, replaces it with the ad hoc agent, and observes the performance of the newly created team. The overall performance of the ad hoc agent is averaged over a number of these samples.

3. METHODS

In order to cooperate effectively with its teammates, the ad hoc agent chooses its actions by planning about their long term effects. To do so, the ad hoc agent must have a model of the domain, the prey, and its teammates. This paper assumes that the ad hoc agent has a correct model of the domain and the prey, but must determine how to model its teammates.

Even if the ad hoc agent has a perfect model of its teammates, the planning problem is still difficult. With four predators and a single prey, the pursuit domain has a branching factor of $5^5 = 3125$ actions and, in a 20x20 world, there are $(20 * 20)^5 \approx 10^{13}$ different states. While it is possible to reduce the size of the state space using some symmetries of the world, planning efficiently is important in the pursuit domain.

3.1 UCT

To plan efficiently, our ad hoc agent uses UCT [13], a Monte Carlo Tree Search (MCTS) algorithm that employs upper confidence bounds for controlling the tradeoff between exploration and exploitation. Previous work has shown that UCT performs well on domains with high branching factors, such as Go [8] and large POMDPs [16]. In addition, previous work has shown that UCT can both compete with optimal planners on small pursuit problems and scale to larger problems [2].

At each time step, the ad hoc agent performs a number of rollouts, where a rollout is the simulation of an episode starting from the current world state and ending when the prey is captured. Then, the ad hoc agent uses the time it takes to capture the prey in each simulation to evaluate the actions it selected in the rollout. It selects its actions by choosing the one with the highest upper confidence bound, causing it to explore when it is unsure of the best action and exploit its knowledge when it is confident of the results.

3.2 Model Selection

Performing the simulations for the UCT rollouts requires that the ad hoc agent has a model for how its teammates behave. If there is a (presumably correct or approximately correct) single model for this behavior, the planning is straightforward. On the other hand, the problem is more difficult if the ad hoc agent is given several possible models. Assuming that the ad hoc agent starts with some prior belief distribution over which model correctly reflects its teammates' behaviors, the ad hoc agent can update these beliefs by observing its teammates. Specifically, it can update the models using Bayes theorem:

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

If the correct model is in the given set of models, then the ad hoc agent's beliefs will converge to this model.

On the other hand, if the correct model is not in the set, using Bayes rule may drop their posterior probability to 0 for a single wrong prediction. This may punish generally well-performing models that make one mistake, while leaving poor models that predict nearly randomly. Therefore, it may be advantageous to update the probabilities more conservatively. Research in regret minimization has shown that updating model probabilities using the polynomial weights algorithm is near optimal if examples are chosen adversarially [3]. Since it is expected that the ad hoc agent's models are not perfect, the agent updates its beliefs using polynomial weights:

$$\begin{aligned} \text{loss} &= 1 - P(\text{actions}|\text{model}) \\ P(\text{model}|\text{actions}) &\propto (1 - \eta * \text{loss}) * P(\text{model}) \end{aligned}$$

where $\eta = 0.5$. This scheme ensures that good models are not prematurely removed, but it does reduce the rate of convergence. We find that in practice it performs very well as the observed examples of the teammates may be arbitrarily unrepresentative of the agent's overall decision function.

Given the current belief distribution over the models, using this information, the ad hoc agent can sample teammate models for planning, choosing one model for each rollout similarly to the approach adopted by Silver and Veness [16]. Sampling the model once per rollout is desirable compared to sampling a model at each time step because this resampling can lead to states that no model predicts. Ideally, a different state-action evaluation would be stored for each model, but that would require many more rollouts to plan effectively. Instead, the state-action evaluations from all the models are combined to improve the generalization of the planning.

3.3 Learning Models

The previous sections described how the ad hoc agent can select the correct model and use it for planning, but they did not specify where these models come from in the first place. One option is that the ad hoc agent is given a model that was coded by hand, but another interesting option is when the ad hoc agent learns the model itself. If the ad hoc agent can observe representative potential teammates before interacting with them, it can build a model for their behaviors. To build these models, the ad hoc agent uses the features given in Table 1, where all positions are relative to the modeled predator. These features were chosen experimentally using cross validation on the training set. The ad hoc agent considers the previous actions of the teammate because a few of potential teammates appeared to keep an internal state that changed infrequently. In this case, a model is a mapping from these features to a next action, and a training instance represents only a single predator's actions: a trajectory of length k provides $4k$ training instances.

Description	# Features	Values
Predator Number	1	$\{0, 1, 2, 3\}$
Prey x position	1	$\{-10, \ldots, 10\}$
Prey y position	1	$\{-10, \ldots, 10\}$
Predator$_i$ x position	3	$\{-10, \ldots, 10\}$
Predator$_i$ y position	3	$\{-10, \ldots, 10\}$
Neighboring prey	1	{true,false}
Cell neighboring prey is occupied	4	{true,false}
Previous two actions	2	$\{\leftarrow, \rightarrow, \uparrow, \downarrow, \bullet\}$

Table 1: Features for predicting a teammate's actions.

In some cases, rather than having extensive observations of its teammates for learning models, the ad hoc agent will only have a small number of observations of its current teammates. It could still build a model from this data, but it may be able to improve the accuracy of the model by using information it has about similar teammates through transfer learning (TL). Following standard TL terminology, we consider the current teammates to be the *target* teammates: the goal is to improve performance when teamed with these agents. We call the previously observed teammates the *source* teammates, which can provide knowledge to transfer to modeling the target teammates. To perform this transfer, we introduce the TwoStageTransfer algorithm.

TwoStageTransfer was inspired by the TwoStageTrAdaBoost algorithm created by Pardoe and Stone [15]. TwoStageTrAdaBoost is an algorithm for transfer learning that uses the source data directly in the target task rather than transferring derived classifiers. It searches for the optimal weighting of the source data using n-fold cross validation with the target data. TwoStageTrAdaBoost focuses on transfer for regression rather than classification and treats all source data as coming from the same task. In contrast, TwoStageTransfer tackles the classification problem and uses the information that source data may come from different sources. In this case, the ad hoc agent has observed many other agents, some of which are more similar to the target teammate than others. Therefore, tracking the source of the data may be important as it allows the ad hoc agent to discount data coming from agents that are very different from it. Recent research into transfer learning has shown that such information may improve results [20, 11].

TwoStageTransfer's goal is to find the best possible weighting of each set of source data and create a classifier using these weights. The full algorithm is described in Algorithm 1. TwoStageTransfer takes in the target data set T, the set of source data sets $\mathbb{S} = \{S_1, \ldots, S_n\}$, a number of boosting iterations m, a number of folds k for cross validation, and a maximum number of source data sets to include b. We use the annotation S^w to mean the data set S taken with weight w spread over the instances. The base model learner used in this case is a decision tree learner based on C4.5 trees that handles weighted instances.

Ideally, TwoStageTransfer would try every combination of weightings, but this proves to be computationally expensive as transferring from n source data sets and considering m different weight levels leads to m^n possible combinations. In our case, we have $n = 28$ data sets and $m = 10$ weightings (as discussed in Section 4), which leads to 10^{28} combinations. These data sets are discussed in more depth in Section 4.3. Rather than try all of them, TwoStageTransfer first evaluates each data source independently, as if it were the only source data set, and calculates the ideal weight of that data source as specified in Algorithm 1. Then, it adds the data sources in decreasing order of the calculated weights. As it adds each data set, it finds the optimal weighting of that set with the data that has already been added. Finally, it adds the data with the optimal weight and repeats the procedure with the next data set. Note that this algorithm requires only $nm + nm = 2nm$ combinations to be evaluated, nm for the initial evaluations and then m when adding each n data sets. In this case, it requires only $2 * 10 * 28 = 560$ combinations to be evaluated. However, this greedy approach may not find the optimal weights. In addition, it is possible to limit the number of data sets used in the final transfer, which is desirable in our setting as training with all 50,000 instances from each of the 28 available source data sets (see Section 4) of the source data sets can exceed 4 GB of RAM usage.

Algorithm 1 Transfer learning with multiple sources

```
TwoStageTransfer $(T, \mathbb{S}, m, k, b)$
  for all $S_i$ in $\mathbb{S}$: do
    $w_i \leftarrow$ CalculateOptimalWeight$(T, \emptyset, S_i, m, k)$
  end for
  Sort $\mathbb{S}$ in decreasing order of $w_i$'s
  $F \leftarrow \emptyset$
  for $i$ from 1 to $b$ do
    $w \leftarrow$ CalculateOptimalWeight$(T, F, S_i, m, k)$
    $F \leftarrow F \cup S_i^w$
  end for
  Train classifier $c$ on $T \cup F$
  return $c$
end
CalculateOptimalWeight$(T, F, S, m, k)$:
  for $i$ from 1 to $m$ do
    $w_i = \frac{|T|}{|T|+|S|}(1 - \frac{i}{m-1})$
    Use $k$-fold cross validation on $T$
    Calculate err$_i$ from $k$-fold cross validation on $T$ using
      $F$ and $S^{w_i}$ as additional training data
  end for
  return $w_j$ such that $j = \underset{i}{\text{argmax}}(\text{err}_i)$
end
```

4. RESULTS

This section evaluates a number of ad hoc agents that vary in the type and amount of information they have about their teammates. If it has access to the true model, the ad hoc agent can plan to cooperate with its teammates optimally, as approximated by UCT(True), or use the same behavior as the missing teammate (Match(True)). However, in the fully general ad hoc teamwork scenario, such a model is not generally available. Thus, this paper instead focuses on the case in which the ad hoc agent must learn a model of its teammates by observing them. In this section, the amount of available information available to the ad hoc agent varies, ranging from observing its current teammates to only observing teammates that may differ greatly from the current ones. Additional results focus on the case in which the ad hoc agent has a small number of observations of the current teammates and must transfer information learned from prior observations of similar teammates.

To properly evaluate an ad hoc team agent, it is important to test it with a variety of possible teammates. To prevent any bias in the development of these teammates, we collected 31 agents created by undergraduate and graduate computer science students. These agents were created for an assignment in a workshop on agent design with no discussion of ad hoc teams. The students were asked to create a team of predators that captured the prey as quickly as possible. The agents produced varied wildly in their approaches as well as their effectiveness. While almost all of the agents performed well, two agents were removed due to their low performance, leaving 29 agents. In Sections 4.1 and 4.3, the teammates come from this set of 29 students, but for Section 4.2, the teammates come from the set of 12 students used by Barrett et al. [2] to prevent any bias in the selection of the unknown teammates.

For these evaluations, a random team is selected, a single agent is replaced by the ad hoc agent, and the team is evaluated based on its time to capture the prey. Throughout this paper, we refer to a teammate type as its behavior function, meaning that agents coming from different students have different types. To simulate wear and tear on the teammates, agents of the same type may further vary based on their speed. This speed is controlled by giving each agent a random chance to stay still rather than taking their desired action, which can be thought of as the gears on a robot slipping. The chance of staying still is randomly sampled independently for each agent from $[0, 0.2]$, but in training all teammates are observed with a 0.1 probability of staying still. Results are averaged over 1,000 episodes where a random student's team is selected for each episode. The random teams, probabilities of agents staying still, and starting positions are fixed across the ad hoc agents to allow for paired statistical analysis, and all statistical tests are performed as paired Student-T tests with $p = 0.05$.

When the ad hoc agent observes its potential teammates, it watches a team of four predators for 50,000 steps, resulting in a total of 200,000 training instances. These predators have a 0.1 probability of staying still. With this information, the ad hoc agent learns a decision tree using a learning algorithm based on C4.5 trees that handles weighted instances. Several other classifiers were tried including SVMs, naive Bayes, and decision lists as well as boosted versions of these classifiers, but decision trees tended to outperform these methods in a combination of prediction accuracy and training time. All model learning is performed offline, but the ad hoc agent updates its belief over the models online.

The behaviors tested are listed below, with ad hoc agents planning with the learned models compared to agents given either the true model or a set of representative, hand-coded models:

- **Match(True)**: Match teammates' behavior. The ad hoc agent knows the true model of its teammates, and behaves as the agent it is replacing would.
- **UCT(True)**: Plan with the true model. The ad hoc agent knows the true model of its teammates and plans with this model.
- **UCT(HC)**: Plan with hand-coded models. The ad hoc agent is given a set of hand-coded models to plan with.
- **UCT($\mathbf{DT_{cor}}$)**: Plan with correct decision tree. The ad hoc agent knows the type of teammates it is cooperating with and has observed this type before.
- **UCT($\mathbf{DT_{all}}$)**: Plan with all decision trees. The ad hoc agent does not know the type of teammates it is cooperating with, but it has observed several types of possible teammates, including the current teammate type.
- **UCT($\mathbf{DT_{oth}}$)**: Plan with other decision trees. The ad hoc agent does not know the type of teammates it is cooperating with, and it has observed several types of possible teammates, but *not* the current teammate type.
- **UCT($\mathbf{DT_{tra}}$)**: Plan with a decision tree created using transfer learning. The ad hoc agent knows the type of teammates it is cooperating with and has briefly observed this type, but it has also observed several other types of teammates for longer periods of time.

For the DT_{cor}, DT_{all}, and DT_{oth} models, the ad hoc agent builds a separate decision tree for each of the 29 types of teammates it has observed. In the UCT(DT_{cor}) behavior, the ad hoc agent knows the type of its current teammate and only uses its model of that type for planning, while in the UCT(DT_{all}) and UCT(DT_{oth}) behaviors it uses all available models. However, in the UCT(DT_{oth}) behavior, the true teammates are not drawn from the set of known teammate types. Finally, to create the DT_{tra} models, the ad hoc agent observes the full 50,000 steps of 28 teammate types, but only 1,000 steps of the type it is currently cooperating with. Therefore, it uses transfer learning in the form of TwoStageTransfer to reuse knowledge learned from the other 28 teammate types to create the DT_{tra} models, using the same decision tree learner as the base learner.

This section compares the performance of ad hoc agents to the unrealistic behaviors of Match(True) and UCT(True) and the previous best result of UCT(HC), leading to three main results. As described in Section 4.1, the first result is that learning models from known teammate types outperforms previous techniques. Then, Section 4.2 presents the second result, showing that these learned models still perform well for teammates of previously unseen types. Finally, Section 4.3 shows that transfer learning can improve the performance of the ad hoc agent if only a small amount of data on the current teammates is available.

4.1 Known Teammate Types

This section compares the performance of ad hoc agents that learn models of their teammates to agents that have access to true models of their teammates. Given the true behavior model of its teammates, the ad hoc agent can either behave as the missing teammate would (Match(True)), or it can plan using this true model (UCT(True)). If the teammates are optimal, Match(True) is also optimal, but regardless of the teammates' behaviors, UCT(True) should be close to optimal, with any loss caused by the approximations of the planning algorithm. Therefore, these two behaviors serve as baselines and UCT(HC) represents the current state of the art behavior [2]. The hand-coded models given to the ad hoc agent are chosen to be representative of the space of behaviors in the pursuit domain, but they may be only coarse approximations of the true teammate behavior. In addition, they may require a significant amount of time and effort on the behalf of the designer to create.

Compared to these behaviors, we introduce two new behaviors in which the ad hoc agent learns a model of its teammates. Both create models for each of the 29 types of agents it has observed, but the UCT(DT_{cor}) agent also knows the type of its current teammates. More reflective of the fully general ad hoc teamwork scenario is the UCT(DT_{all}) behavior, whereby the ad hoc agent does not know the type of its current teammates and must track its beliefs over the learned models by observing its teammates.

Figure 2 shows how an ad hoc agent planning with the learned models compares to planning with the true model or a set of hand-coded models. Unsurprisingly, the ad hoc agent performs best when planning using the True model, as it should be optimal given an optimal planning algorithm. Though conventional wisdom suggests that matching the teammates behavior would be fairly effective, this performs poorly as shown by the Match(True) bar because the teammates may be arbitrarily far from optimal. The difference between UCT(DT_{cor}) and UCT(HC) is statistically significant as is the difference between UCT(HC) and Match(True). The differences between UCT(True) and

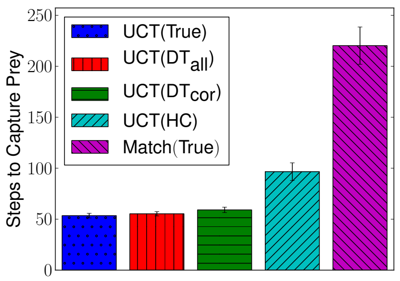

Figure 2: Interacting with observed student teammates

UCT(DT_{all}) as well as the differences between UCT(DT_{all}) and UCT(DT_{cor}) are not statistically significant.

If the ad hoc agent plans with the DT_{cor} or DT_{all} models, it outperforms an ad hoc agent that plans using the HC models, showing that learning a model of the teammates is effective. In addition, if the ad hoc agent plans with the DT_{cor} models, its performance is very close to the gold standard of planning with True models. However, using the DT_{all} models allows the ad hoc agent to outperform an agent planning with the DT_{cor} models, which is surprising given selecting from the set of DT_{all} models should increase the difficulty of the problem. We hypothesize that this difference occurs from imperfections in learning the models. Some student agents are fairly stochastic and it is possible and likely that the learned decision trees overfit this data. By using a set of models of all the agents, the ad hoc agent plans about a broader range of possible teammates and better accounts for this randomness.

4.2 Unknown Teammate Types

While the previous section focused on the case where the ad hoc agent encounters teammates of a type it has previously observed, the ad hoc agent may not always be that lucky. Instead, it may have to cooperate with teammates that are fairly different from any it has seen before. The ad hoc agent therefore plans with its 29 learned models, but encounters a 30^{th} type of teammate drawn from a set of 12 student agents described in the work by Barrett et al. [2].

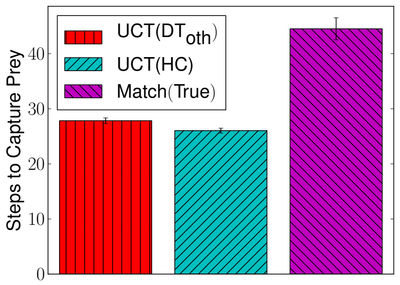

Figure 3: Interacting with the 12 unobserved student teammates from Barrett et al.

Figure 3 shows that having the ad hoc agent plan using the HC models works very well for these teammates. Planning using the DT_{oth} models performs slightly worse than the HC models due to differences between the learned models and the encountered teammates. The performance difference is statistically significant. However, the difference is small, and may be reduced by using a larger, more varied set of training agents. On the other hand, planning using the DT_{oth} models still far outperforms the Match(True) behavior, showing that the DT_{oth} models are still effective for planning.

4.3 Teammates with Limited Observations

The previous section discussed the case in which the ad hoc agent had no previous observations of its current teammates. However, this is the worst case scenario; in many cases, the ad hoc agent may build up a small number of observations of the current teammate model. Then, it can use transfer learning to reuse its observations of other agents to improve the performance of this model. In this case, the ad hoc agent has 50,000 training steps of each of the other 28 teammate types, but only 1,000 training steps of the current type of teammate. The teammates used for these tests are the same as those in Section 4.1, specifically, the set 29 teammates generated by the students.

The model learned using transfer learning is compared to the upper limit of UCT(DT_{all}) where the ad hoc agent has 50,000 training instances of the current type of teammate, though observed with a different amount of noise. The lower limit is given by only using the 1,000 training steps of the current teammate type and not performing any transfer ($\text{DT}_{\text{target}}$). Figure 4 shows that using TwoStageTransfer to perform transfer learning is helpful in this case. However, analysis of the UCT(DT_{tra}) results shows that there are a small number of very long episodes caused by inaccuracies in the learned models, but this happens less frequently when planning with the $\text{DT}_{\text{target}}$ models. Therefore, it is desirable for the ad hoc agent to use both models, which outperforms using either one individually as shown by the UCT(DT_{tra} + $\text{DT}_{\text{target}}$)bar. For all uses of TwoStageTransfer, a value of $m = 10$ was experically determined to be most effective. DT_{tra} is built using TwoStageTransfer where $b = 5$, meaning that the ad hoc agent is transferring information from the five most helpful source agents. Five was chosen as it provides a good tradeoff between performance and training time. On the other hand, the ad hoc agent could merely take the one step greedy approach by performing TwoStageTransfer where $b = 1$. If it also plans using the model learned from the target data, the ad hoc agent's performance is shown in $\text{DT}_{\text{tra}*}$ + $\text{DT}_{\text{target}}$. The differences from the UCT($\text{DT}_{\text{target}}$) behavior to each other behavior is significant, but the differences among the behaviors planning with the other learned models are not.

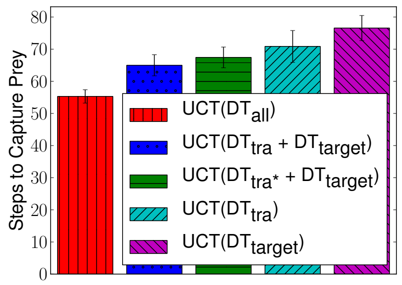

Figure 4: Interacting with partially observed student teammates

In the current setup, the ad hoc agent is given observations of 1,000 steps of the target teammate performing the task. However, the amount of target data may vary, and it is important to understand how the performance of the transfer learning algorithms react to this variable. Figure 5 shows how the performance of the agent drops off given decreasing amounts of data about the current teammate type. In the tests where only 10 and 100 steps of the target agent were observed, episodes did not complete with the capture of the prey within 10,000 steps in 13 and 12 episodes respectively out of the 1,000 episodes. These episodes are excluded from the graph as it is unclear what value they should take, but it is clear that they indicate poor performance in these settings.

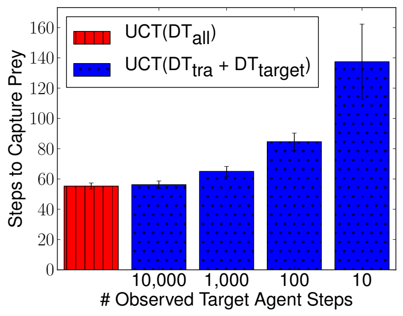

Figure 5: Varying the number of observations of the current teammate type.

5. RELATED WORK

This formulation of the ad hoc teamwork problem and the evaluation framework were proposed by Stone et al. [17]. Barrett and Stone [1] analyze current research on ad hoc teams and present several dimensions for describing ad hoc team problems. One line of early research on ad hoc teams involves an agent attempting to teach a novice agent while performing a repeated joint task [5]. Other research includes Jones et al.'s [12] research on pickup teams cooperating to accomplish a treasure hunt. In the area of robot soccer, Liemhetcharat and Veloso [14] explore ad hoc teamwork for role-based teams, and Bowling and McCraken [4] consider the case where the ad hoc agent is given a different playbook from its teammates. Further work into ad hoc teams using stage games and biased adaptive play was performed by Wu et al. [19], while Han et al. [10] explore how a single agent can affect the collective behavior of a large, multi-agent system.

The problem of opponent modeling is closely related to the problem of ad hoc teams. Where opponent modeling focuses on modeling opponents and considers the worst case scenarios for their actions, ad hoc teamwork instead focuses on cooperating with teammates and can make stronger assumptions about their actions. Important work on opponent modeling is regularly presented at the Workshop on Plan, Activity, and Intent Recognition (PAIR) as well as the Workshop on Applied Adversarial Reasoning and Risk Modeling (AARM). One interesting approach is the AWESOME algorithm [6] which achieves convergence and rationality in repeated games. Another approach is to explicitly model and reason about other agents' beliefs such as the work on I-POMDPs [9] and I-DIDs [7]. However, modeling other agents' beliefs greatly expands the space for planning, and these approaches do not currently scale to larger problems.

6. CONCLUSION

Most existing research on ad hoc teamwork focuses on the case in which the ad hoc agent has access to a correct model of its teammates. This paper is the first to have the ad hoc agent learn models of its teammates autonomously. Planning with these learned models allows the ad hoc agent to cooperate with its teammates effectively even when its teammates were not observed during the model learning. This paper also presents a transfer learning algorithm, TwoStageTransfer, that can significantly improve results when the ad hoc agent has a limited number of observations of its teammates.

While this paper answers questions about the variety of teammates that ad hoc team agents can effectively cooperate with, it also raises new questions that should be explored in future research. One possible research avenue is into communication between ad hoc teams. Wireless connectivity is becoming more and more common, and approaches such as the Robot Operating System (ROS) are standardizing communication protocols. Therefore, it is likely that ad hoc team agents may be able to communicate with their teammates, although this communication may be limited by what their teammates were developed to understand. All results in this paper are reported the pursuit domain, but future research should test whether similar algorithms perform well on other domains. In addition, this work focuses on agents that follow mostly fixed behaviors, with little adaptation to the ad hoc agent's behaviors; handling adaptive teammates is a complicated, but exciting area for future research.

7. REFERENCES

[1] S. Barrett and P. Stone. An analysis framework for ad hoc teamwork tasks. In *AAMAS '12*, June 2012.

[2] S. Barrett, P. Stone, and S. Kraus. Empirical evaluation of ad hoc teamwork in the pursuit domain. In *AAMAS '11*, May 2011.

[3] A. Blum and Y. Mansour. Learning, regret minimization, and equilibria. 2007.

[4] M. Bowling and P. McCracken. Coordination and adaptation in impromptu teams. In *AAAI*, pages 53–58, 2005.

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

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

[7] P. Doshi and Y. Zeng. Improved approximation of interactive dynamic influence diagrams using discriminative model updates. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 907–914, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[8] S. Gelly and Y. Wang. Exploration exploitation in Go: UCT for Monte-Carlo Go. In *NIPS '06*, December 2006.

[9] P. J. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multi-agent settings. *JAIR*, 24(1):49–79, July 2005.

[10] J. Han, M. Li, and L. Guo. Soft control on collective behavior of a group of autonomous agents by a shill agent. *Journal of Systems Science and Complexity*, 19:54–62, 2006.

[11] P. Huang, G. Wang, and S. Qin. Boosting for transfer learning from multiple data sources. *Pattern Recognition Letters*, 33(5):568 – 579, 2012.

[12] E. Jones, B. Browning, M. B. Dias, B. Argall, M. M. Veloso, and A. T. Stentz. Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks. In *ICRA*, pages 570 – 575, May 2006.

[13] L. Kocsis and C. Szepesvari. Bandit based Monte-Carlo planning. In *ECML '06*, 2006.

[14] S. Liemhetcharat and M. Veloso. Modeling mutual capabilities in heterogeneous teams for role assignment. In *IROS '11*, pages 3638 –3644, 2011.

[15] D. Pardoe and P. Stone. Boosting for regression transfer. In *ICML '10*, June 2010.

[16] D. Silver and J. Veness. Monte-carlo planning in large pomdps. In *NIPS '10*. 2010.

[17] 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.

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

[19] F. Wu, S. Zilberstein, and X. Chen. Online planning for ad hoc autonomous agent teams. In *IJCAI*, 2011.

[20] Y. Yao and G. Doretto. Boosting for transfer learning with multiple sources. In *CVPR '10*, June 2010.